

## **SESSION**

# **MICRO-CONTROLLERS, MICRO-PROCESSORS, PROGRAMMING, SOFTWARE SYSTEMS AND RELATED ISSUES**

**Chair(s)**

**TBA**



# Lumousoft Visual Programming Language and its IDE

Xianliang Lu

Lumousoft Inc. Waterloo Ontario Canada

**Abstract** - This paper presents a new high-level graphical programming language and its IDE (Integration Development Environment) for microprocessor-based embedded system. The graphical programming language allows programming in the graphical block diagram environment. In comparison with textual programming languages, it allows direct bit operation, and enables to program or lay out complicated networks for cognitive algorithm application. With the trace analyzer, the graphical language can fully recycle memory, reduce reductant code to reduce the cost, and improve performance. The graphical structure including basic units, layout format and modules are described in details. This new graphical language provides great convenience for embedded software maintenance, coding, verification, validation and reusable components.

**Keywords:** graphical; module; programming; language; IDE

## 1 Introduction

With the increasing complexity and sophistication of electronics device, the more efforts are put on embedded software that plays a pivotal role in electronic industry. There are great demands for more efficient, productive and reliable software-developing tool to facilitate embedded software design, development and maintenance.

It is proved that the blueprint is the most efficient way in engineering practice for design, validation, verification, test, maintenance and modification. Embedded software is more restrict and complex than other software field due to real time and reliability. Embedded software faults might cause serious damage to electronic device. An embedded software engineer or developer usually has electronic background and familiar with blueprints. But current embedded software design and development have to be on the text-based platform that is hard to read and catch the whole map of software project. Consequently, developing software has to take more efforts and much more time than other engineering fields like electricity, mechanics.

C/C++ language is a dominant language in embedded system due to similarity to assembly language and more readability. Comparing with assembly language, C/C++ compiler might generate bigger size of machine codes and use up more memory, sometimes may cause memory leakage and memory conflict or boundary corruption, yielding more cost

in hardware and poor quality of performance. Because it has more readability and efficiency than assembly language, it still gains dominant application in the embedded field.

In addition to textual language, visual programming language (VPL) begin to be found application in embedded system like flowcode produced by Matrix Multimedia for programming embedded devices. Most of VPL languages have limit application and less flexibility because programming capability depends on how many graphical components available from VPL provider.

Lumousoft graphical language is a visual programming language for microprocessor allowing programming in block environment with high usage of hardware resource and high quality of performance. This graphical language not only has the same programming capability as textual language such as C but also enables to program complex network for cognitive algorithm application.

## 2 Syntactic structure

Lumousoft graphical language is a programming language in block environment. In general, the textual language uses textual statement to represent flowing program path in the form of parse tree, and each tree branch is not allowed to be interconnected, an example parse tree is shown in Fig.1. Unlike textual language, Lumousoft graphical language uses block, connection line and direction symbol to form program flowing pattern, the block diagram connection pattern is not limited to parse tree, allowing branches to be interacted to form complicated networks. An example is illustrated in Fig.2. In lumousoft graphical language, a path analyzer is employed to analyze the program pattern including loop.

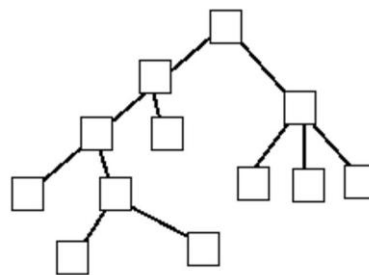


Fig. 1. Textual language parse tree

Lumousoft graphical language uses path analyzer to identify block connection pattern other than parse tree. However, in each block are there multiple lines of expression for logic or arithmetic operation, these lines of instruction are

arranged in executable sequence without jumping branch. While, each line of expression still follows parse tree for analysis and compilation.

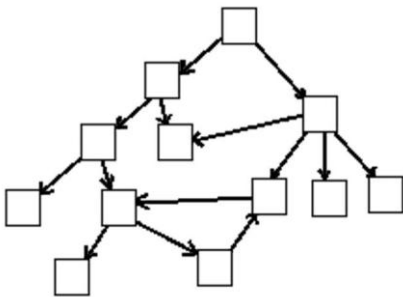


Fig. 2. Block network

Lumousoft graphical language employs path analyzer to determine network. This mechanism is different from current textual language syntactic structure and allows lumousoft graphical language to handle more complicated network to accomplish the more complex and sophisticated task.

### 3 Layout format

#### 3.1 Basic unit

Lumousoft graphical language is made of four units: block, line, direction and statements in a block. The block contains executable statement, which performs logic and arithmetic operation; line is to build connection between blocks; direction determines how the process flows. The symbol of direction is shown in Fig. 3 there are two class directions, input and output. The output can be divided further into non-conditional output (pass output) and conditional outputs (pair of true and false output and switch).

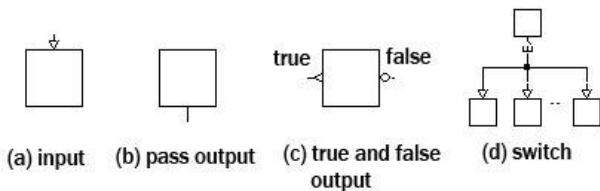


Fig. 3. Block direction symbols

Because lumousoft graphical language is a programming language, the program codes need to be modified form times to times, the symbol or shape of graphic icon might change accordingly, this will cause great inconvenience for developer and waste a lot of time to adjust symbol or icon. Therefore, Lumousoft graphical language does not adopt the standard flowchart symbols, just employs block for ease of programming.

#### 3.2 Connection models

The block layout can be in the form of series, parallel, selection, loop and bridges. Fig. 4 illustrates the basic layout

formats. With these basic layout units, a complicated network can be created.

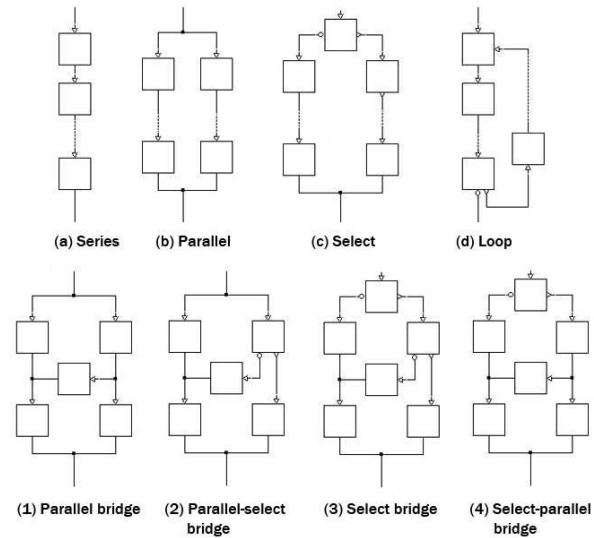


Fig. 4. Basic layout format

#### 3.3 Series

Similar to series circuit, the program executes in sequence.

#### 3.4 Parallelism

In our daily life, parallelism exists like electrical circuit. The current Lumousoft language version is applied to single chip microprocessor, not for multi-cores. In order for single core to handle parallelism, the parallel branches need to be rearranged and connected in sequence. Fig.5. demonstrates the conversion of parallel to sequence.

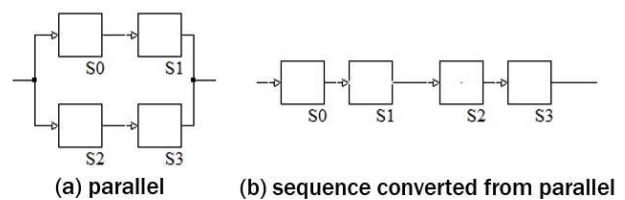


Fig. 5. Transformation form parallel to sequence

The parallelism that lumousoft graphic can handle should meet the following condition.

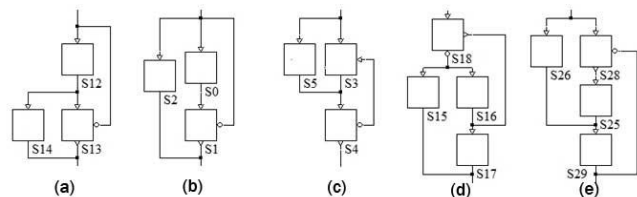


Fig. 6. Illegal parallel connection



## 4 Variable

Like other programming languages, Lumousoft graphical language has local, global, system, constant, array pointer variable. Except the regular data type, lumousoft graphical language has bit variable that is especially helpful for microprocessor application. As soon as a variable is defined, the variable can be accessed in the whole scope where the variable stays. There is no need to make declaration of variable.

### 4.1 Memory Optimization

Because lumousoft graphical language adopts block network instead of parse tree, tremendous memories might be required. The memory size is one of factors that affect the price of microprocessor. The bigger size of memory the more expensive microprocessor is.

A variable usually is assigned a memory or register to store a special value. When the content of a memory is written and the previous value of the memory is lost, we can say that the previous value is dead and the variable associated with this dead value die too. While, when the memory is written a new value, the variable associated with this memory is born. A variable has life span from the variable is assigned a new value to the last time usage before the memory was override. The variable might alive and die for many times in a program.

Lumousoft graphical language takes the advantage of trace analyzer to indentify the life span of a variable, When a memory is born or created, a memory will be allocated to the variable, and when the variable is dead the associated memory will be released and ready for a new variable. Because global, static, system variables have permanent life, only the local memory is going to be optimized in lumousoft graphical language.

### 4.2 Bit Variable and Operation

Bit is the atomic data type of data. Current high level languages like c does not allow directly operating on bit, usually using indirect way like byte bitwise operation. Lumousoft graphical language allows direct operate on bit. Here, the variable holding the bit variable is called as bit source variable, and where the bit sits in a bit source is called bit position value.

The bit variable can be defined in the variable manager dialog of IDE or use the below statement to define:

$$bitVar = \&(sourceVar \# positionVal) \quad (1)$$

Or uses the below expression to represent a bit variable

$$sourceVar \# positionVal \quad (2)$$

where

*bitVar* - bit variable

*sourceVar*- bit source variable

*positionVal*- bit position in the bit source variable

A bit value can be 0 or 1 like Boolean, when bit work with non-bit data type variable or in logical operation, the bit

variable is treated as a Boolean variable. When the bit is set the Boolean value is 1, and the bit is reset, the Boolean value 0. Therefore, when a bit variable works with other non-bit variable it works as Boolean similar to other language like c.

When a bit variable works with other bit variable it will operate according to bit value. The bit operation can be:

*and (&), or(|), xor(^), assign(=), not(~ or !),*

*and assign(&=), or assign(|=), xor assign(^=).*

When a bit source variable consists of more than one byte, lumousoft graphical compiler will use expressions (3) and (4) to find the byte that bit stay and find out the correct bit position.

$$nByte = \text{floor}(bitPos / sysBitLen) \quad (3)$$

$$nPos = bitPos \% sysBitLen \quad (4)$$

Where

*nByte* - the byte where the concerned bit stay.

*nPos* - the bit position in the *nByte*

*floor* - a function to get the round down integer value

*bitPos* - bit position

*sysBitLen* - The bit number of memory that microprocessor can handle

For example a bit variable is fifteenth bit of a bit source, the bit source is short data type and supposed that it is allocated by 2 bytes with the address of 0x20 and 0x21. The address of 0x20 is lower byte and 0x21 higher byte. The compiler will use the 7th bit of the byte with the address of 0x21 to replace this bit variable.

## 5 Modules

Lumousft graphical language is specially designed for microprocessor application, the program consists of 5 parts:

- Main process module: It is similar to c language 'main' function, as shown in Fig.10 (a). Usually There is a big loop in embedded application, a terminal block named End Loop is introduced if necessary.
- Library: support compilation for logical and arithmetical operation
- Interruption: it is to handle interruption routine for microprocessor, as shown in Fig.10 (b).
- Sub module: it is similar to the function in c language and can be used by other modules. Sub modules can exist in different files. Sub module can be either module function or inline module as shown in Fig.10 (d) and (e).
- Global initialization: It is used to initialize global, system and static variable and perform microprocessor initialization before running main program module as shown in Fig.10 (c).

For the simplicity, we use single block to represent complex block networks in Fig.10.

A sub module in lumousoft graphical language is a group of blocks to achieve a special task, and at any point of statement in a block it can be called, just the same as function in textual language. Lumousoft graphical language treats the sub module name as a variable, while the function in textual language it is just a return value. In lumousoft graphical language, if a sub module name followed by a parentheses, it indicates that the sub module will be called and assign a return value to this sub module name variable. If there are not parentheses followed, the sub module is simply a variable.

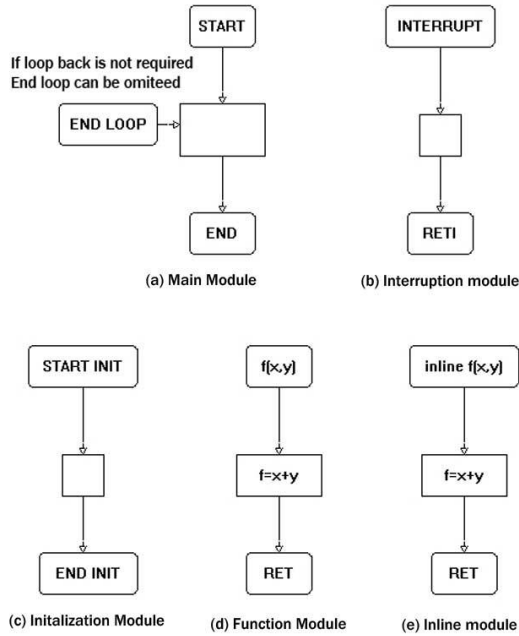


Fig. 10. Modules

The sub module can be divided into two kinds of modules, inline sub module and regular sub module, just like inline function and function in textual language. Inline sub module means that the calling statement is completely replaced with the copy of the whole group of module blocks. A regular sub module is used to be called. Because sub module name is a variable, there is no need to be declared. But if the sub module is in another file, it needs to be imported. When the file is imported, all the sub-modules in the imported file can be implemented.

Since the main module and the interruption module belong to different thread, the memory of variables should be isolated except to that of global variable. If the different thread modules call the same sub module, Lumousoft graphical compiler will make one copy of the sub module including variables for each thread usage.

### 6 Assemble Language

Since assembly language is flexible and efficient, the assembly language is still found application in embedded field. Usually assembly language is embedded in other language.

Lumousoft graphical language allows assembly language to be embedded in block environment.

Assembly language operand can be a variable defined in the module including array, pointer except bit variable. But these variables used in assembly language must be those variables that have known address through static analysis. Unhandled variable like a[i], \*p, where p=&x+i; the variable "a" is array variable and "i" is a variable, "p" is pointer variable; in other words, these unhandled variables are those that are only handled by indirect address.

Because the assembly instruction only handles single-byte variable, if a variable is composed of more than one byte, only the first byte can be handled by assembly instruction. To solve one-byte problem, we can implement pointer to handle all bytes.

### 7 IDE

Lumousoft IDE offers the facility to lay out lumousoft graphical program, compilation and generation of assembly codes, machine codes for burning to microprocessor as well as other files like block flow file which textually describes how the block flow, assignment file for variable address assignment and label of program memory address. Fig.11 shows the interface of IDE.

Lumousoft IDE graphical editor functions include drawing line, block, directions, and terminal block. The editor has functions like undo, redo, copy, delete, cut, paste, move, save, selection, search etc.

The codes in block are edited through block diagram, which pops up when double clicks on a block that wants to be edited. Fig.11 shows the block dialog and variable manager. The upper left text block is used to fill in codes. When a statement is put in, the IDE will perform lexical analysis and

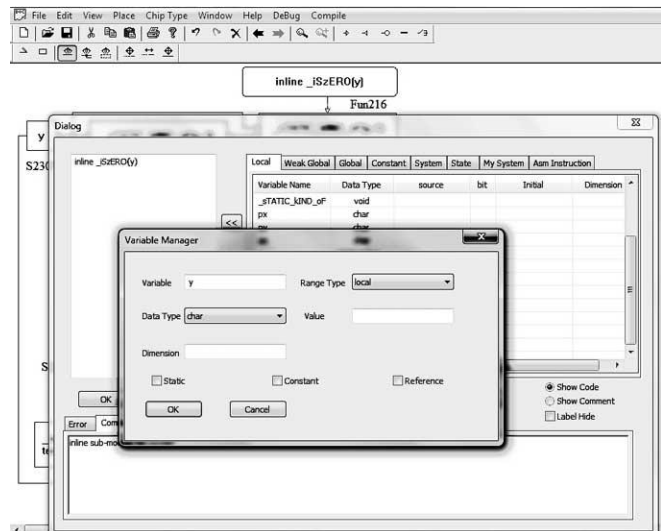


Fig. 11. Block dialog and variable manager

obtain variable, key words and other tokens. The scanned variables are put into the variable tables on the right hand side. When IDE find a new variable, the variable has default attribute with data type of char and range type of local. The

variable can be edited and change its attributes through variable manager dialog as shown in Fig.11. All the variables within the connected blocks can be accessed.

The bottom box is used for the user to write comment for this block. Codes and comments can be flipped to display by clicking on the radio button in the middle on the right hand side of block dialog.

Two situations need to be considered when click on "compile" button on the right top of the menu bar:

- The current file does not contain main module. When click on "compile", it will generate an object file with extent name "sub", which is used for other files to import.
- The current file contains main module. When click on "compile", it will generate assembly codes and machine codes for microprocessor.

## 8 Examples

### 8.1 Example 1

The example codes, shown in Fig.12, is an inline module from the Lumosoft library, which is used to judge whether the variable is zero for all kinds of data type such as bit variable, constant, pointer variable, one byte or multiple byte variable.

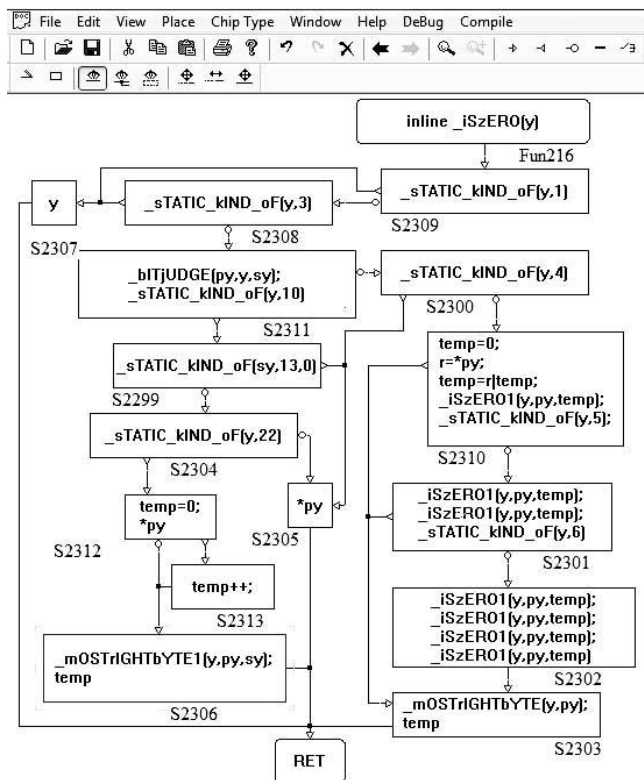


Fig. 12. An example graphical codes

Fig.13 is an alternative displays of code, showing the block comment to indicate how the program flow.

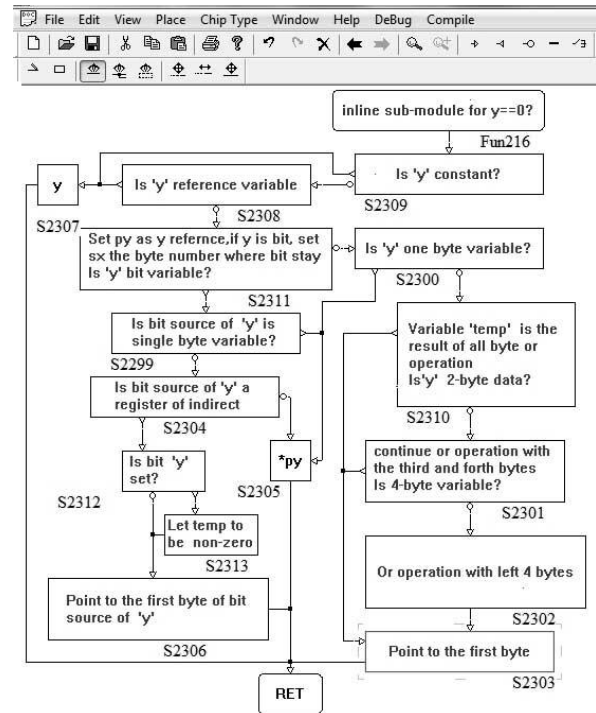


Fig. 13. Graphical pseudo codes

### 8.2 Example 2

The next example is to design two sets of led display by use of PIC16F677. The port A connects to a set of led and the port B to another set of led. In each set of led, only one led lights up at each time, all the leds display in turn. For the first set, each led lights up for 2 seconds; while, for the second set, each led lights up for 4 seconds. Interruption takes place every one millisecond to record time. Since the two sets of led operate simultaneously, we can apply parallel to handle. Fig. 14 and Fig.15 illustrate the graphical codes and machine codes respectively.

## 9 Conclusion

Lumosoft graphical language is a high-level graphical programming language, designed for microprocessor based embedded system, allowing the user to design and develop embedded software in the block environment. Lumosoft graphical language uses path analyzer to analyze block network connection pattern instead of parse tree, as a result, lumosoft graphical language can offer more flexible and intuitive method to handle more complicated algorithm with less efforts. Taking advantage of path analyzer, Lumosoft graphical language is able to determine each variable life span and make full recycle of memory. Lumosoft IDE offers the facility to layout lumosoft graphical program, compilation and generates assembly codes, machine codes. Lumosoft graphical language provides a graphical coding solution to handle embedded software engineering like other blueprints based engineering field. Because of graphical coding, lumosoft graphical language can give the user a bird view of their project with great convenience of embedded software



maintenance, review, design, coding, validation, verification, test and modification.

[2] W.Ackerman, "Data Flow Languages", IEEE Computer, pp.15-25, February, 1982.

[3] Y. Zhang and B. Xu. "A survey of semantic description frameworks for programming languages". ACM SIGPLAN, 39(3):14-30, 2004.

[4] Charles N. Fischer, Richard J. LeBlance, Jr. "Crafting a Compiler", Benjamin-Cummings Publishing Co., Inc. Redwood City, CA, USA, 2010.

[5] Dick Grune, Herri E.Bal, Cerial J.H. Jacobs and Koen G. Langendoen,"Modern Compiler Design", John Wiley & Sons, Inc., New York, NY, USA, 2000.

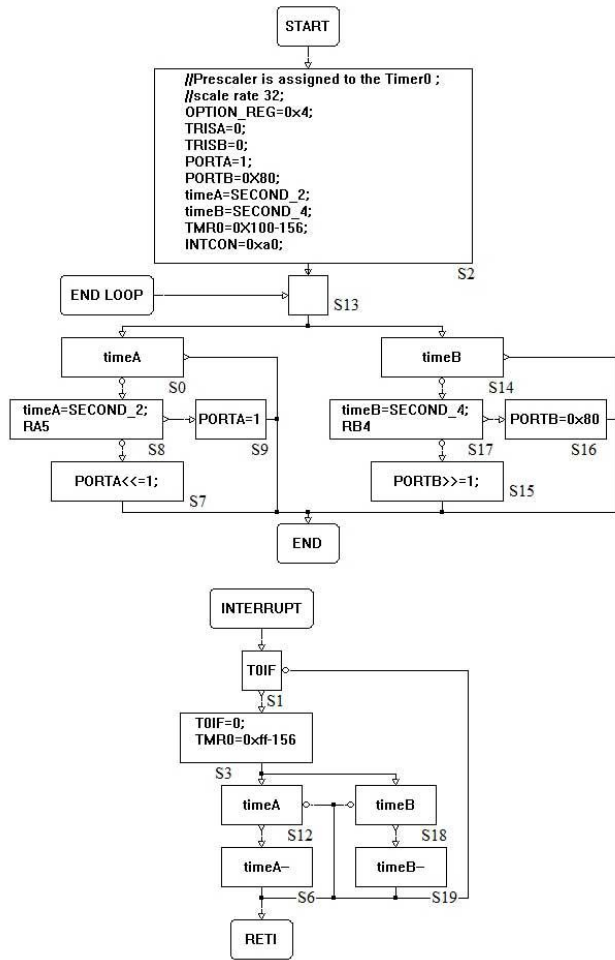


Fig. 14. Two sets of led display codes

1	30	28	00	00	00	00	00	ff	00	03	08	fe	00	0b	1d	
2	1c	28	0b	11	63	30	83	12	03	13	81	00	a4	01	20	08
3	a4	04	21	08	a4	04	03	1d	28	28	a4	01	22	08	a4	04
4	23	08	a4	04	03	1d	20	28	7e	08	83	00	7f	08	09	00
5	01	30	a2	02	03	1c	25	28	1c	28	01	30	a3	02	24	28
6	01	30	a0	02	03	1c	2d	28	15	28	01	30	a1	02	2c	28
7	04	30	83	16	03	13	81	00	85	01	86	01	01	30	83	12
8	85	00	80	30	86	00	d0	30	a2	00	07	30	a3	00	a0	30
9	a0	00	0f	30	a1	00	64	30	81	00	a0	30	8b	00	a5	01
10	22	08	a5	04	23	08	a5	04	03	1d	56	28	d0	30	a2	00
11	07	30	a3	00	85	1e	69	28	01	30	85	00	a5	01	20	08
12	a5	04	21	08	a5	04	03	1d	65	28	a0	30	a0	00	0f	30
13	a1	00	06	1e	66	28	80	30	86	00	47	28	03	10	86	0c
14	65	28	85	0d	05	10	56	28	00	00	00	00	00	00	00	00
15	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Fig. 15. Machine Codes

## 10 References

[1] Bragg, S.D. , Driskill, .G. D, "Diagrammatic-graphical programming languages and DoD-STD-2167A", AUTOTESTCON '94. IEEE Systems Readiness Technology Conference. 'Cost Effective Support Into the Next Century', Conference Proceedings, pp 211 - 220, Sep. 1994.

# Analysis and Optimization of *Paradigm* Microprograms

Victor L. Winter<sup>1</sup>, James McCoy<sup>2</sup>, Dominic Montoya<sup>3</sup>, and Greg Wickstrom<sup>4</sup>

<sup>1</sup>Department of Computer Science, University of Nebraska at Omaha, Omaha, NE, USA

<sup>2</sup>Sandia National Laboratories, Albuquerque, NM, USA

<sup>3</sup>Sandia National Laboratories, Albuquerque, NM, USA

<sup>4</sup>Sandia National Laboratories, Albuquerque, NM, USA

**Abstract**—*Microcode often plays a key role in modern processor architectures. Microcode optimization is an important topic, and opportunities for microcode optimization can present themselves at various levels of abstraction. The Paradigm System, developed as part of a joint research effort between Sandia National Laboratories and the University of Nebraska at Omaha, consists of a high-level architecture-independent microprogramming language together with its compiler. This paper discusses the artifacts and mechanisms, within the Paradigm System, that support the analysis and optimization of Paradigm microprograms.*

**Keywords:** micro-programming, microprogram optimization, microprogram analysis, program transformation

## 1. Introduction

On modern processing platforms there oftentimes exists a computational gap between the functionality provided by the *assembly language instruction set*, which is targeted by high-level language compilers, and the set of *signals* used to control hardware resources. In such an environment, microcode (*μcode*) can be effectively used to emulate the functionality of assembly language instructions that are not directly supported by the hardware.

Because *μcode* lies at the core of a processor's design, its optimization is an important topic. Efficiency gains in *μcode*, even small gains, can have a substantive impact on system-level performance. Research into the optimization of *μcode* spans algorithmic optimization of high-level *μcode* down to determining the optimal order in which microoperations (*μoperations*) should be executed.

### 1.1 Application

*Paradigm* is a high-level architecture-independent microprogramming (*μprogramming*) language that has been developed as part of a joint research effort, funded by Sandia National Laboratories (SNL), between SNL and the University of Nebraska at Omaha. The primary application motivating *Paradigm* is the development of a processor, called the *Scalable Core (SCore)* [1]. The SCore is a hardware implementation of a subset of the JVM, designed and developed at SNL, for use in high-consequence embedded systems [2].

Within the SCore, the functionality of Java bytecodes is achieved through *μprogramming*. In particular, each Java bytecode supported by the SCore is realized through a corresponding *μcode* implementation. Furthermore, native methods used in the JVM and supported by the SCore are also implemented in *μcode*.

### 1.2 Contribution

The *Paradigm* System provides a unique environment for exploring *μcode* optimization through a mixture of manual activities such as restructuring high-level *μprograms* and automated activities such as the compaction of low-level *μcode* performed by the *Paradigm* compiler.

This article will discuss the following aspects of the *Paradigm* System that facilitate *μcode* analysis and optimization.

- *extensive analysis* – The *Paradigm* compiler produces a variety of artifacts (heat maps, graphs, tables) providing developers with insight into the allocation of registers that occur during compilation as well as detailed estimates of time/space trade-offs associated with calling versus in-lining methods.
- *user defined optimizations* – *Paradigm* developers can affect compilation in three important ways: (1) through *in-lining directives*, which are source-code level directives instructing the compiler to inline particular methods, (2) through *optimizing transformations*, which are transformation rules that developers can add to the compiler itself in order to perform specific optimizations during compilation, and (3) through *timing constraints*, which are used to guide the compaction of *μcode* instructions.

The remainder of this article is structured as follows: Section 2 reviews some the basic concepts and terminology of microcoding. Section 3 overviews related work. Section 4 discusses analysis artifacts produced by the *Paradigm* compiler. Section 5 overviews user-defined optimization rules that can be added to the *Paradigm* compiler. Section 6 describes the declarative language used by *Paradigm* to specify parallel capabilities of a target machine, and Section 7 concludes.

## 2. The Basics of Micro-programming

The purpose of *μprogramming* is to orchestrate the behavior of resources in a CPU. The basic concept was developed by Maurice Wilkes [3] in 1951 who also coined the term *micro-programming*. Microprogramming (*μprogramming*) provides what amounts to a software-based alternative to the hardware-based logic boxes whose design was (and is) considered to be a bit of a black art[4].

A *μprogram* is a specification of how the resources within a CPU are to be controlled. *μprograms* can be expressed at various levels of abstraction: *High-level μprograms* strive to facilitate human comprehension, and can have syntactic and semantic similarities to high-level general-purpose programming languages such as C and Java. In contrast, *low-level μprograms* are suitable for execution on a processor. The purpose of a *μcompiler* is to translate a high-level *μprogram* into a low-level *μprogram*.

A low-level *μprogram* consists of a sequence of *μinstructions*. A *μinstruction* consists of a set of *μoperations* each of which specify the control of a fundamental resource within the processor. Typical examples of *μoperations* include:

- the transfer of data from memory to a register
- elementary operations such as shift, load, and clear performed on data residing a register
- properly updating the internal registers of the control unit in order to enable a jump

A *μoperation* consists of a set of *fields*. Fields are made up of bits whose binary values correspond to control lines. For example, a field consisting of  $k$  bits can be used to denote  $2^k$  combinations of control lines. A special case arises when  $k = 1$  for all fields. *μinstructions* constructed exclusively from *μoperations* having 1-bit fields are referred to as *horizontal μinstructions*. Horizontal *μinstructions* are long, but allow for the maximal expression of parallelism. In contrast, the signals denoted by fields for which  $k > 1$  are encoded, and *μinstructions* made up of such fields are referred to as *vertical μinstructions*. A benefit of such encoding is that the bit-width of *μinstructions* is significantly reduced. However, the parallelism which can be expressed through vertical *μinstructions* is limited and combinatory logic is needed to decode field values.

Orthogonal to the vertical/horizontal nature of a *μinstruction* is the architectural notion of how many *μoperations* a *μinstruction* can hold. If only a “few” *μoperations* can be placed into a *μinstruction* the machine has a *vertical architecture*; otherwise it has a *horizontal architecture*[5][6].

For architectures that support concurrent execution of *μoperations*, be they vertical architectures or horizontal architectures, the scheduling of *μoperations* presents an area of optimization. In this context, the goal of optimization is to produce a low-level *μprogram* having a minimal or

near-minimal number or *μinstructions*. For this form of optimization, referred to as *μcode compaction*, achieving optimal results has been shown to be NP-complete[7]. There are two types of compaction: (1) *local compaction* which focuses on restructuring the *μoperations* within *straight-line μcode*(SLM) – also known as *basic blocks*, and (2) *global compaction* whose focus spans multiple SLMs.

## 3. Related Work

Research into the design of high-level *μprogramming* languages and *μcompilers* predominantly took place during the 1970's and early 1980's. A number of papers have been published on the topic of *μcode* optimization [8], [9], [10], [5]. Agerwala [11] has written a survey on *μcode* optimization. A central issue in the type of optimization discussed in the survey is the reduction of the size of the *control memory* needed to hold a *μprogram* implementing a given function. Here, the control memory is modeled as a two-dimensional array ( $W \times B$ ) where  $W$  denotes the number of words (i.e., rows) and  $B$  denotes the number of bits (i.e., columns) in the control memory respectively. A primary goal of optimization is to reduce the control memory along either of its dimensions.

In [11], optimization strategies are categorized as being either high-level or low-level. High-level optimizations are based on dataflow analysis of the source-code and strive to discover parallelism inherent in the algorithm implementation. Optimizations possible at this level also include existing (well-known) compiler optimization techniques. Roughly stated, the result of high-level optimization is a sequence of sets, called *time frames*, whose elements are *μoperations*. This sequence of time frames is viewed as partitioning the computation defined by the high-level (input) *μprogram* in a manner that is maximally parallel irrespective of physical limitations of the host machine. After such a partitioning has been completed, low-level optimizations can be applied to map the structure onto a host machine. These low-level optimizations center on normalizing the existing partition structure so that each set in the partition can be realized by exactly one horizontal *μinstruction*.

SIMPL (Single Identity Microprogramming Language) [12] is a high-level (machine dependent) *μprogramming* language developed in the early 70's having an ALGOL-like syntax. During SIMPL compilation, a high-level *sequential program* undergoes sophisticated analysis in order to produce a highly optimized low-level *horizontal program*. SIMPL optimization is based heavily on the *single identity principle* which states that a (particular) definition for a variable holds from the point it is assigned up to the point where it is reassigned. The single identity principle forms the basis for partitioning a sequence of statements into *subblocks* each of which constitute an independent set of *μoperations*. This decomposition represents a key first step in solving the global optimization problem.

Though there were a number of  $\mu$ code language and compiler development efforts underway at the time, SIMPL was considered to be the first high-level  $\mu$ programming language in which both compilation and optimization were performed automatically. A SIMPL compiler has been developed targeting the Tucker-Flynn dynamic microprocessor [13].

Micro-C [14] is a high-level machine-independent  $\mu$ programming language compatible with C. A Micro-C  $\mu$ program can be compiled by a special compiler based on the Portable C Compiler. The output produced by this compiler is vertical (i.e., unoptimized) symbolic  $\mu$ code. This intermediate representation can then be optimized by a “straight-line” packer which translates sequences of  $\mu$ operations into horizontal  $\mu$ instructions. An assembler is then used to translate the result into executable low-level  $\mu$ code.

In [15], a language is presented in which high-level  $\mu$ programs are composed of *declaration* statements and *command* statements. The compiler for this language consists of two phases: In the first phase of compilation, the input  $\mu$ program is parsed, analyzed, and an unoptimized sequence of  $\mu$ instructions is produced. At this stage, each  $\mu$ -instruction performs exactly one *elementary operation* (i.e., a  $\mu$ operation). The second phase of compilation is an optimization phase in which a number of tables containing machine-dependent information (e.g., parallel capabilities of the hardware) are employed in order to *compact*  $\mu$ instructions taking full advantage of the parallel capabilities of the hardware.

In [16], an approach is presented where machine-independent high-level  $\mu$ code optimization is performed by the software component of a  $\mu$ code compiler and low-level machine-dependent optimization is performed by hardware residing on the host machine (i.e., the machine on which the  $\mu$ code will be executed). In this context, the goal of a *hardware microcode optimizer* (HMO) is to condense a sequence of  $\mu$ instructions (i.e., where each  $\mu$ instruction contains only one  $\mu$ operation) into a functionally equivalent sequence of  $\mu$ instructions taking full advantage of the parallel capabilities of the host machine. At a higher-level, optimization strategies are divided into two distinct categories: The *local optimization* category is performed by the hardware-based component of the compiler and focuses on the serial combination (i.e., compaction) of  $\mu$ instructions. The *global optimization* category is performed by the software-based component of the compiler and focuses on the commutative reordering  $\mu$ instruction sequences (driven by dataflow analysis) in order to more fully exploit parallelism.

## 4. Analysis

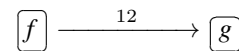
In addition to designing well-structured high-level  $\mu$ programs, developers often need to pay close attention to the consumption of resources entailed by their design. For

example, how many internal registers are needed by the compiler to compile a given high-level  $\mu$ program? What is the size, in terms of the number of  $\mu$ instructions, of the resulting low-level  $\mu$ program produced by the compiler? And, how many  $\mu$ instructions are executed when the program is run?

The *Paradigm* compiler produces three artifacts to assist developers in their optimization-oriented analysis efforts: (1) views, (2) heat maps, and (3) estimation tables.

### 4.1 Views

*Paradigm* provides a notation, called a *view*, for specifying subsets of methods. From the specification of such subsets, views can be constructed. In particular, a view is an acyclic directed graph whose nodes denote methods and whose labeled edges denote the number of internal registers allocated by the compiler relative to specific nodes. For example, consider the graph below consisting of two nodes, labeled  $f$  and  $g$ , connected by an edge labeled 12.



This graph indicates that (1) the method  $g$  is called in the body of  $f$ , and (2) at the point of the call to  $g$ , the compiler has allocated 12 internal registers local to the context of  $f$ .

A high-level  $\mu$ program may have multiple views defined for it, each of which will be output to a correspondingly named file. Such files are output in a “dot format” and can be viewed using Graphviz. Figure 1, shows an example of a view generated by the *Paradigm* compiler for a  $\mu$ program produced for a hypothetical machine.

### 4.2 Heat Maps

Heat maps are another form of feedback produced by the *Paradigm* compiler. Specifically, the *Paradigm* compiler will output twelve attributes to a file in a comma-separated value format. Attributes range from method arity, method size, reference\_frequency, inlined - called size, to (inlined - called size) \* reference\_frequency. Figure 2 shows a heat map for a hypothetical machine. In this heat map, the first grouping (in grey) is by method type (e.g., macro, subroutine, operator, operation, condition, interface). The second grouping (also in grey) is the difference between the in-lined size and the called size – this includes all overhead associated with making a method call. The size of squares in the heat map represents the called size, and the color indicates reference frequency with red denoting the most frequently referenced methods and blue denoting the least frequently referenced methods.

### 4.3 Efficiency Estimator

In order to meet resource constraints, it may be necessary for developers to optimize their high-level  $\mu$ program. To facilitate optimization, *Paradigm* provides high-level language directives that can be used to instruct the compiler to in-line various method declarations.

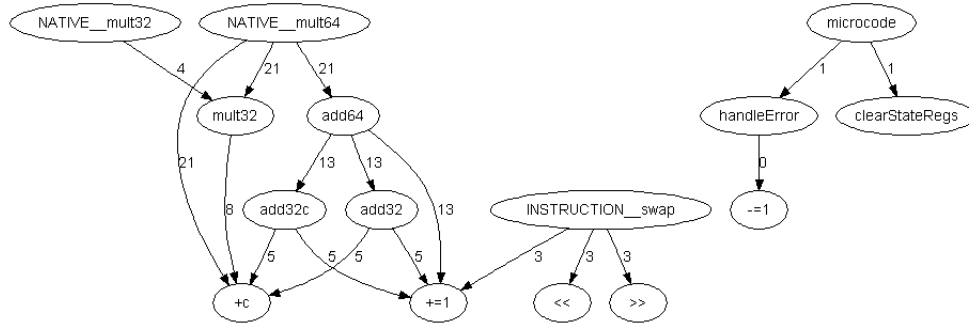


Fig. 1: A view showing internal register allocations performed by the compiler.

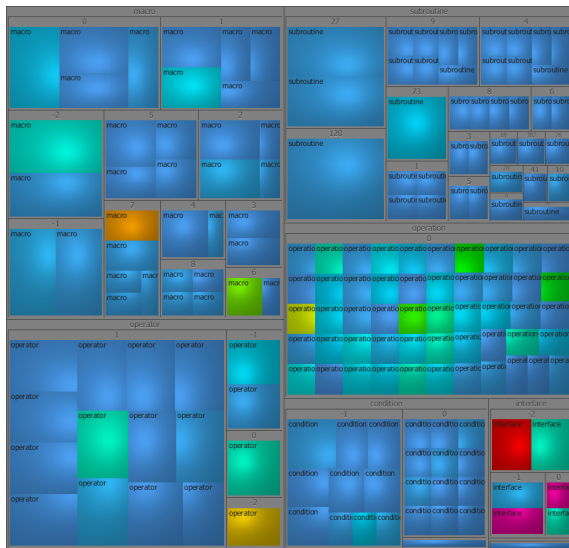


Fig. 2: Heat map of  $\mu$ code for a hypothetical machine.

Method in-lining represents a time/space tradeoff, since in-lining can cause the size of the low-level  $\mu$ code to expand dramatically. For example, suppose the body of a method  $m$  consists of 100 lines of  $\mu$ code. Further suppose that  $m$  is called in 10 places in the  $\mu$ code. If all 10 calls are in-lined, then in-lining (without compression) will yield 1000  $\mu$ instructions. In contrast, suppose that a call to the method  $m$  requires 20 lines of  $\mu$ code. In this case, calling  $m$  10 times will result in a total overhead of 200 lines of  $\mu$ code. Thus, an implementation in which  $m$  is called will contain 700 fewer lines of  $\mu$ code (i.e., 200 lines of call overhead plus 100 lines for the method body). However, it should be noted that in-lined methods always execute faster than their called counterparts since there is no call overhead associated with their execution.

The overhead associated with a method call is significant. Internal registers must be allocated for the input parameters. Instructions must be generated by the compiler to move actual parameters to the internal registers corresponding to

formal input parameters of the method. A *call* instruction must be generated by the compiler to transfer execution to the method body, and a *return* must be executed upon completion of the method body. Furthermore, moves, calls, and returns do not lend themselves to compression. In other words, only one such  $\mu$ operation will fit into a  $\mu$ instruction. Thus, going back to our previous example, if the execution of each  $\mu$ instruction takes 1 unit of time, then executing the body of  $m$  via a call will take 120 units of time.

As the body of a method gets smaller it gradually becomes more attractive to in-line a method. Eventually, a crossover point is reached where calling a method consumes more time and more space than simply in-lining a method. It should be noted that, from the point of view of development, the method is a mechanism for abstracting functionality. Thus, a best-practices approach to development would encourage the use of methods as needed to give clarity to an implementation.

The *Paradigm* compiler, provides an estimation of the effects of method call versus method in-lining. In particular, two sorted tables are produced: (1) a *static call-frequency estimation table*, and (2) an *execution path estimation table*. Examples describing the information in both of these tables are described in the sections that follow.

### 4.3.1 Example: Static Call-Frequency Estimation

Suppose method  $m1$  is an in-line method candidate having 3 formal input parameters. Furthermore, let us assume that a static inspection of the *Paradigm* application reveals that  $m1$  is called from 10 syntactically distinct locations. Similarly, suppose method  $m2$  is inline candidate method having 2 formal input parameters. Furthermore, let us assume that a static inspection of the *Paradigm* application reveals that  $m2$  is called from 20 syntactically distinct locations.

Static Call-Frequency Estimation		
Method	Move Instruction Overhead	Static Overhead Sum
$m1$	$3 * 10 = 30$	$(3 + 2) * 10 = 50$
$m2$	$2 * 20 = 40$	$(2 + 2) * 20 = 80$

It should be noted that static-call-estimation provides a fairly coarse grained and basic estimation of the overhead associated with calling methods. In particular, static call estimation does not take into account execution paths which can have multiplicative effect on the number of times a method can actually be called during runtime. For example, suppose method  $m1$  is called twice in the body of method  $m2$ , and suppose method  $m2$  is called 5 times within the  $\mu$ code. Note that in this example, there are only 2 lexical occurrences of  $m1$ . However,  $m1$  will be called a total of  $5 * 2 = 10$  times during the execution of the application. We call this second form of estimation *execution path estimation*. It should be noted that, since it does not account for loop iterations, execution path estimation is also only an estimate, albeit a more accurate one than static call estimation.

### 4.3.2 Example: Execution Path Estimation

Suppose methods  $m1$ ,  $m2$  and  $m3$  are respectively called 5, 6, and 4 times from the  $\mu$ code as shown in Figure 3. Also note, that  $m1$  is called 2 times from  $m2$  and  $m2$  is called 3 times from  $m3$ .

The execution path estimation table shows that the total calls for  $m1$  is 41. This value corresponds to the sum:  $1 * 5 + 1 * 6 * 2 + 1 * 4 * 3 * 2 = 41$ . More specifically,  $m1$  is called 5 times from the  $\mu$ code. This accounts for the  $1 * 5$  term. Next,  $m1$  is called 2 times from  $m2$ , which itself is called 6 times from the  $\mu$ code. This accounts for the term  $1 * 6 * 2$ . And finally,  $m2$  is called 3 times from  $m3$  which is called 4 times from the  $\mu$ code. This accounts for the term  $1 * 4 * 3 * 2$ .

In this example, the in-lined size for  $m1$  is 0. This is because the body of  $m1$  is empty (after the removal of the return instruction). The called size for  $m1$  is 83 and corresponds to 41 calls plus 41 returns plus the size of the declaration of  $m1$  (which is 1).

The in-lined time will always be equal to the in-lined size. The assumption here is that each row in the  $\mu$ code takes 1 unit of time to execute, and that additional compression of method bodies is not possible.

The called time for  $m1$  is 82. This corresponds to the total calls to  $m1$  times the sum of the number of moves associated with calling  $m1$  plus the number of microcode rows associated with the call-to and return-from  $m1$ .

And finally, the speed up is 100%. This number is computed using the following formula:

$$100.0 - \frac{(inlined\_execution\_time/called\_execution\_time) * 100.00}{100.00}$$

Although it is not highlighted by the example given, it should be noted that the execution path estimator accounts for the mandatory inlining of all *macros*, *interfaces* and

Method	Total Calls	Size	
		Inlined	Called
$m1$	41	0	83
$m2$	18	36	39
$m3$	4	12	12

Method	Time		% Speedup
	Inlined	Called	
$m1$	0	82	100%
$m2$	36	72	50.0%
$m3$	12	20	40.0%

Table 1: Execution path estimation.

```

interface call(LabelType toLabel) { aux_call(); }
interface return() { aux_return(); }

subroutine m1() returns void { return(); }
subroutine m2() returns void {
    m1(); m1(); return();
}

subroutine m3() returns void {
    m2(); m2(); m2(); return();
}

microcode {
    m1(); m1(); m1(); m1(); m1();
    m2(); m2(); m2(); m2(); m2(); m2();
    m3(); m3(); m3(); m3();
}

```

Fig. 3: Example used for execution path estimation

*conditions*<sup>1</sup>. This is important because such mandatory inlining can result in dramatic changes in the final size of a subroutine or operator. Also note that the size of the call and return interfaces also take inlining into account.

## 5. User-defined Optimization Rules

The *Paradigm* compiler is transformation-based and implemented in the TL System[17]. During compilation, a *Paradigm* program is passed through a number of canonical forms, each of which can be output in human-readable form. The *Paradigm* compiler is *extensible* in the sense that it supports the incorporation of user-defined transformation rules into the compilation process. Such rules provide domain experts the opportunity to perform custom optimizations specific to a particular architecture or  $\mu$ code design. Figure 4 is an example of a  $\mu$ code fragment, which can be output by the compiler, consisting of a sequence of  $\mu$ operation method calls separated by labels denoting jump destinations (e.g., starting positions of methods whose bodies have not been in-lined).

By inspection of the sequence of operations we see that a *writeReg* operation is immediately followed by a *copyReg* operation. Suppose that by combining knowledge of the

<sup>1</sup>The language *Paradigm* has five different kinds of methods. The rationale behind this is beyond the scope of this article.

```

label_f: ...
writeReg(TlType.SOME, AType.$temp_reg 3 );
copyReg( AType.$temp_reg 3 ,AType.$reg 2);

```

Fig. 4: A  $\mu$ code fragment prior to custom optimization.

hardware architecture together with our understanding of the semantics of the implementations of the *writeReg* and *copyReg* operations we conclude that the transformation shown in Figure 5 is correctness-preserving. Furthermore, suppose that additional analysis leads us to conclude that such a transformation would be correctness-preserving **in all contexts**. That is, regardless of how it gets generated by the compiler, whenever a “write” to a temp register  $X$  is followed by a “copy” from that temp register  $X$  to the register  $Y$ , then this pair of operations can be replaced by a single operation that will directly “write” to the register  $Y$ .

Given that these conditions hold, we would like to expand the functionality of the compiler to include such an optimizing transformation. *Paradigm* supports such extension of its compiler through a special transformation module in which domain experts can place custom-designed program transformations. There are no restrictions on the nature of the transformations that can be created. In particular, optimizing transformations can be developed utilizing the full capabilities of the TL system.

```

writeReg(TlType.SOME, AType.$temp_reg 3 );
copyReg( AType.$temp_reg 3 ,AType.$reg 2);
→
writeReg(TlType.SOME,AType.$reg 2);

```

Fig. 5: A custom program transformation.

## 6. *Paradigm*'s Timing Constraint Language

*Paradigm* provides a declarative language, called TCL, for specifying the timing constraints of a targeted hardware architecture. Timing constraints form the basis of a *local compaction* algorithm focusing on the compression of straight-line  $\mu$ code (SLM). Timing-constraint based optimization does not involve commutative reordering of  $\mu$ operations, instead it focuses on maximizing the compression of adjacent (i.e., associative)  $\mu$ instructions. It is worth mentioning that in the compilation stage where timing-constraint based optimization occurs, the  $\mu$ program being compiled is in a form where all non-sequential control flows are expressed in terms of jumps to labels. In this context, an

SLM is then simply the sequence of  $\mu$ instructions occurring between consecutive labels.

Conceptually, a timing constraint is a pair of logical formulas that, if satisfied by adjacent  $\mu$ instructions, prevent them from being compressed into a single  $\mu$ instruction. Compression is also (implicitly) prohibited in cases when corresponding fields, in adjacent  $\mu$ instructions, contain distinct (i.e., unequal) non-default signals.

An abstract example of the syntax of a timing constraint is shown in Figure 6. In the example,  $\mathcal{F}_1$  and  $\mathcal{F}_2$  denote the pair of logical formulas of the timing constraint named  $TC_k$ .

The evaluation of  $TC_k$  with respect to a pair of adjacent  $\mu$ instructions  $\mathcal{I}_j$  and  $\mathcal{I}_{j+1}$  proceeds as follows: If  $\mathcal{I}_j$  satisfies  $\mathcal{F}_1$  and  $\mathcal{I}_{j+1}$  satisfies  $\mathcal{F}_2$ , then we say that the the  $\mu$ instructions  $\mathcal{I}_j$  and  $\mathcal{I}_{j+1}$  satisfy the timing constraint  $TC_k$ , in which case the compression of  $\mathcal{I}_j$  and  $\mathcal{I}_{j+1}$  is prohibited by  $TC_k$ ; otherwise compression is not prohibited by  $TC_k$ .

```

constraint TC_k {
    first_row: F1;
    second_row: F2;
}

```

Fig. 6: An abstract example of a timing constraint.

TCL allows  $\mu$ code compression to be restricted by a set of timing constraints  $\mathcal{S}_{TC} = \{TC_1, \dots, TC_m\}$ . The compression of any pair of  $\mu$ instructions  $\mathcal{I}_j$  and  $\mathcal{I}_{j+1}$  is prohibited if  $\exists TC_k \in \mathcal{S}_{TC}$  such that  $TC_k$  is satisfied by the  $\mu$ instructions  $\mathcal{I}_j$  and  $\mathcal{I}_{j+1}$ .

A more detailed look at timing constraints reveals that they are logical formulas, in conjunctive normal form, whose elements are equality/inequality matching-based comparisons involving fields. An abstract example of a disjunction constraining the fields  $f1$  and  $f2$  is shown below.

```
field.f1 = field1Type.item1 | field.f2 != field2Type.item2
```

Within an element, there are three kinds of *items* that can be associated with a fieldtype: (1) a *symbolic name* denoting a constant value belonging to a type declaration, (2) a *subscripted variable* which can match with field constants (occurring in the  $\mu$ instructions in which evaluation is taking place), and (3) the keyword DEFAULT/NONDEFAULT. The scope of a subscripted variable spans an entire constraint (both formulas) and can therefore be used to express equality-based properties between fields within a constraint.

The *Paradigm* compiler provides feedback summarizing the impact of the optimizations it performs. Figure 2 shows an example of an optimization summary.

## 7. Conclusion

In the design of a high-level architecture-independent  $\mu$ programming language, a major issue that must be confronted centers on how architecture-specific information can

#### Optimization Metrics:

Standard Compiler Optimizations.

Total number of temp register optimizations = 0

Number of nop() statements removed = 0

#### Custom Optimizations.

Total number of row reductions due to custom optimizations = 0

#### Constraint-based Optimizations.

Number of row mergings prevented due to timing constraints = 1291

Number of duplicate row mergings = 500

Number of conflict-free row mergings = 1000

Total number of constraint-based row mergings = 1500

Number of rows before any optimization = 2800

Number of rows after all optimization = 1300

Size of optimized file as a percentage of the unoptimized file = 46.43%

The size of the unoptimized file was reduced by = 53.57%

Table 2: Optimization feedback provided by the *Paradigm* compiler.

be specified, as well as how the compiler for the language can utilize this information to produce efficient low-level *μcode* targeting a host machine. Addressing this issue, *Paradigm* provides a timing constraint language (TCL) for specifying the parallel capabilities of a host machine. Furthermore, the *Paradigm* compiler also provides extensive feedback on the nature of its compilation, including pretty-printed representations of the program being compiled during various stages of compilation, register usage, call frequency, and comparisons between overheads associated with method call versus method in-lining. This information can be used to guide time/space optimizations involving design level decisions such as method in-lining and can even guide the development of user-defined rule-based application-specific optimizations that can be folded into the compilation process itself.

## References

- [1] J. A. McCoy, "An Embedded System For Safe, Secure And Reliable Execution of High Consequence Software," in *Proceedings of the 5<sup>th</sup> IEEE International Symposium on High Assurance Systems Engineering (HASE)*. IEEE, 2000, pp. 107–114.
- [2] V. L. Winter, H. Siy, J. McCoy, B. Farkas, G. Wickstrom, D. Demming, J. Perry, and S. Srinivasan, "Incorporating Standard Java Libraries into the Design of Embedded Systems," in *Java in Academia and Research*, K. Cai, Ed. iConcept Press, 2011.
- [3] M. V. Wilkes, "The early british computer conferences," M. Campbell-Kelly, Ed. Cambridge, MA, USA: MIT Press, 1989, ch. The Best Way to Design an Automatic Calculating Machine, pp. 182–184. [Online]. Available: <http://dl.acm.org/citation.cfm?id=94938.94976>
- [4] R. C. Haavind, Jr, "The many faces of microprogramming: What started out as a convenience for systems designers may eventually bring computers much better tailored to users' needs," *SIGMICRO Newsl.*, vol. 2, no. 4, pp. 12–16, Jan. 1972. [Online]. Available: <http://doi.acm.org/10.1145/1316527.1316529>
- [5] D. Landskov, S. Davidson, B. Shriver, and P. W. Mallett, "Local microcode compaction techniques," *ACM Comput. Surv.*, vol. 12, no. 3, pp. 261–294, Sept. 1980. [Online]. Available: <http://doi.acm.org/10.1145/356819.356822>
- [6] S. Dasgupta, "The organization of microprogram stores," *ACM Comput. Surv.*, vol. 11, no. 1, pp. 39–65, Mar. 1979. [Online]. Available: <http://doi.acm.org/10.1145/356757.356761>
- [7] S. S. Yau, A. C. Schowe, and M. Tsuchiya, "On storage optimization of horizontal microprograms," in *Conference Record of the 7th Annual Workshop on Microprogramming*, ser. MICRO 7. New York, NY, USA: ACM, 1974, pp. 98–106. [Online]. Available: <http://doi.acm.org/10.1145/800118.803848>
- [8] J. Fisher, "Trace Scheduling: A Technique for Global Microcode Compaction," *Computers, IEEE Transactions on*, vol. C-30, no. 7, pp. 478–490, July 1981.
- [9] S. Davidson, D. Landskov, B. Shriver, and P. Mallett, "Some Experiments in Local Microcode Compaction for Horizontal Machines," *Computers, IEEE Transactions on*, vol. C-30, no. 7, pp. 460–477, July 1981.
- [10] P. Marwedel, "A Retargetable Compiler for a High-Level Microprogramming Language," *SIGMICRO Newsl.*, vol. 15, no. 4, pp. 267–274, 1984.
- [11] T. Agerwala, "Microprogram Optimization: A Survey," *Computers, IEEE Transactions on*, vol. C-25, no. 10, pp. 962–973, Oct. 1976.
- [12] C. Ramamoorthy and M. Tsuchiya, "A High-Level Language for Horizontal Microprogramming," *Computers, IEEE Transactions on*, vol. C-23, no. 8, pp. 791–801, Aug. 1974.
- [13] A. B. Tucker and M. J. Flynn, "Dynamic microprogramming: Processor organization and programming," *Commun. ACM*, vol. 14, no. 4, pp. 240–250, Apr. 1971. [Online]. Available: <http://doi.acm.org/10.1145/362575.362580>
- [14] W. C. Hopkins, M. J. Horton, and C. S. Arnold, "Target-Independent High-Level Microprogramming," in *MICRO 18: Proceedings of the 18th annual workshop on Microprogramming*. New York, NY, USA: ACM, 1985, pp. 137–144.
- [15] A. K. Tirrell, "A Study of the Application of Compiler Techniques to the Generation of Micro-code," in *Proceedings of the meeting on SIGPLAN/SIGMICRO interface*. New York, NY, USA: ACM, 1973, pp. 67–85.
- [16] J. O. Bondi and P. D. Stigall, "Designing HMO, an Integrated Hardware Microcode Optimizer," in *MICRO 7: Conference record of the 7th annual workshop on Microprogramming*. New York, NY, USA: ACM, 1974, pp. 268–276.
- [17] V. L. Winter, "Stack-based Strategic Control," in *Preproceedings of the Seventh International Workshop on Reduction Strategies in Rewriting and Programming*, June 2007.



## Rift Runner – Engineering Software for a Remotely Controlled Rover

Erik Willis, Sean Saunders, Nathan Cate, Jean-Paul Muyschondt and Devon M. Simmonds,  
*University of North Carolina Wilmington, 601 S. College Rd.  
601 South College Road, Wilmington, North Carolina, 28403  
{ emw8872, Sean\_saunders@outlook.com, nathanc\_ii@yahoo.com,  
jm5904, simmondsd }@uncw.edu*

### Abstract

*The Runner is a project based on the first-person piloting of a remotely controlled rover. The rover is controlled by an Oculus Rift, a 3d immersive headset that is designed to bring users into a deep experience of the piloting and manipulation of the vehicle. Users have the ability to upload and share their experiences through the team's web server. This paper describes a model-based approach to the project design along with results and lessons learned.*

**Keywords:** software engineering, model driven engineering, embedded software, UML, Remotely controlled vehicle.

### 1. Introduction

There is an entire world of radio-controlled (RC) vehicle enthusiasts that spend exorbitant sums of time and money on this exciting hobby. Entry-level cars start around \$120, planes from \$250 - \$300, but some planes can cost as much as \$20,000[1]. Until recently, piloting an RC vehicle has been a relatively static experience – stand in a field and watch the vehicle from a distance. As technology has progressed the dream of piloting an RC vehicle from a first-person perspective has become a reality. While the technology currently exists to pilot RC vehicles through video equipment relaying an image to a monitor, the military uses such technology to pilot drones and the market has been ripe for an affordable, first-person, 3-D version. As of 2009 there were more than 5300 aerial drones and 12,000 ground based drones [2]. The MQ-1 Predator Drone costs around \$4 million per unit [3], and the MQ-9 Reaper Drone costs upwards of \$12 million per unit [3]. Enter the Oculus Rift.

The Oculus Rift is a 3-D, immersive, virtual reality (VR) headset that is currently being coupled with video games to present a completely unique and

affordable user experience. The goal of this project is to combine a video camera equipped RC vehicle with the Oculus Rift to create an exhilarating piloting experience. While most of the drone technology is based around military uses, there are also non-military applications. The Miami-Dade County police department, for example, tested an aerial drone in 2011 with the aim of using the drone for aerial reconnaissance and suspect trailing [4]. Another drone that is non-military is the Curiosity Rover. On August 6, 2012 the Mars rover Curiosity landed on the red planet. Curiosity is scheduled to spend two years exploring Mars [5]. Our project is not intended for such serious endeavors, but simply to create entertainment value.

The Rift Runner is a project designed to enable the first-person piloting of a remotely controlled rover. Our main objective was to create a fun, usable, affordable piloting experience. The rover is controlled by an Oculus Rift, a 3d immersive headset that is designed to bring users into a deep piloting and manipulating experience of the vehicle. Vehicular control is achieved through the use of a gamepad while the Oculus Rift controls camera motion through the users head movements.

Our biggest initial concern was the extent to which we could successfully marry our various hardware solutions into a realizable platform. During initial research we discovered that video feed latency has been a problem for similar endeavors. As such we designed our hardware solution to minimize this latency and create a pleasant user experience. To go along with the vehicle we also created a web site that provides a view of the video feed and enables image captures and video segments to be saved. A database was also implemented to track user profiles and links to the image and video storage.

## 2. Background

### 2.1 Oculus Rift

The Oculus Rift is a virtual reality head mounted display (HMD) currently in development by Oculus VR. The device began when Palmer Luckey, a homeschooled tinkerer, decided to build his own virtual reality headset [6]. The project was originally debuted at the Electronic Entertainment Expo in 2012. John Carmack, cofounder of id Software and creator of the *DOOM* series, introduced the first prototype of the Oculus HMD [7]. Following the initial reveal at the E3 in June 2012 the company announced a Kickstarter campaign to raise funds to continue development of the Oculus Rift. After only four hours the company had secured its initial goal of \$250,000 and within thirty-six hours had raised more than \$1 million [8]. The Kickstarter campaign would later end having raised \$2,437,429 [9]. This funding was used to finance a developer's kit of the Oculus Rift. This developer's kit gave people early access to a reduced quality version of the final Oculus Rift but allowed game designers and other developers to begin creating and experimenting with virtual reality environments much sooner [10].

The Oculus Rift, or OR, simulates a purely visual experience for the user in their chosen environment. As a binocular HMD it is worn on the users head and features an optic display in front of each eye. The HMD fully encompasses the user's field of view ensuring as immersive an experience as possible. The OR uses a series of 3-axis gyroscopes, magnetometers, and accelerometers to make head orientation tracking nearly absolute in relation to the Earth. The gyroscopes track the angle of motion as the user moves their head; the accelerometers measure how quickly the HMD is moving; the magnetometers measure the gravitational pull of the Earth allowing the Oculus Rift to keep track of its own orientation with respect to "Up" and "Down" [11]. This head motion tracking is translated to the screen inside the HMD; when the user turns their head to the left the view turns with them. This feature combined with a 90 degree field of view creates a visual experience that mimics real life. Using the Oculus Rift is so immersive and fluid that many users have reported having motion sickness after less than a minute of use. This is caused by the mind thinking the body is moving but in reality the body remains stationary. This issue has been addressed by the company in a recent press release by Oculus VR CEO Brendon Iribe in which he states "It is going to work...It's gonna work for everybody." A reduction in screen shaking and latency has improved

this aspect of the experience for users [12]. These updates won't be seen in the Oculus Rift until the release of the second Developer's kit. The currently available developer's kit still suffers from these motion sickness inducing problems.

### 2.2 Virtual Reality

Virtual reality is a computer-simulated setting that can generate an artificial physical presence either in the real world or abstract environments. Virtual reality can trace its lineage back to the 1500's when artists would create 360 panoramic scenes that would take up entire rooms. An example of this is the *Sala delle Prospettive*, a work by Italian painter Baldassare Peruzzi [13]. It was not until 1966 that the world saw its first glimpse of virtual reality when the T-27 Space Flight Simulator was created for the U.S. Air Force Aerospace program. The simulator was designed to train pilots for space research missions [14]. In 1991 MIT graduate and NASA scientist Antonio Medina developed a system to help direct Mars robots from Earth. This system is an extension of virtual reality [15]. Until recently the technology for virtual reality has been limited in quality or availability for the general public. The Oculus Rift aims to change that.

## 3. Software Design

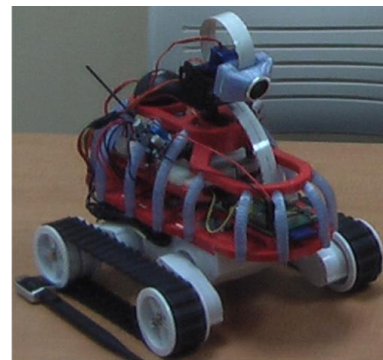


Figure 1. (a) Rift Runner Rover, (b) Oculus Rift and Controller

In this project, hardware architecture preceded software architecture and consisted of the rover (Figure 1a), and the Oculus Rift and controller (Figure 1b). As Figure 1a shows, the camera is mounted at the top and front of the rover.

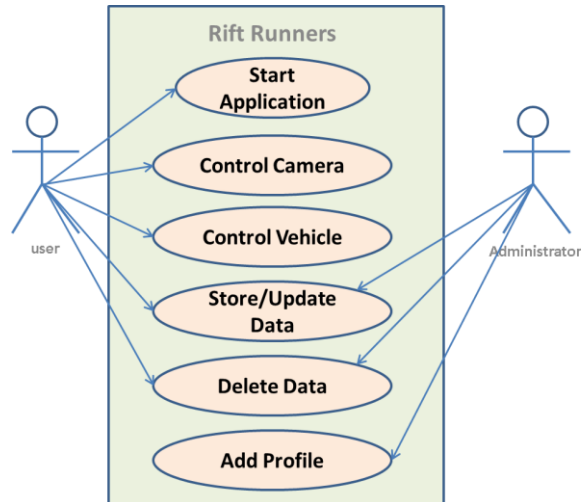


Figure 2. Use Case Diagram

The Rift Runner software was designed using a Model-Driven approach [19-21]. The design includes UML [22] use case diagram (See Figure 2), UML activity diagrams (see Figure 3), an architectural diagram (see Figure 3), and Class diagram (see Figure 4).

The Use Case Diagram (Figure 2) illustrates typical actor interaction with the Rift Runner software and rover. Actors include the end User and the systems administrator who has exclusive rights to add and delete user profiles.

### 2.1. Architectural Design

The physical architecture for this project consisted of the tripartite configuration of (1) the rover and Raspberry Pi, (2) the Oculus Rift and (3) the computer running the software. The logical architecture consisted of a client-Server model centered around a single UDP server as depicted in Figure 5. The UDP server exists on the Raspberry Pi and is responsible for pushing all the data from the main application to the correct hardware, as well as feeding the image data from the camera to the main application. The Raspberry Pi has limited computability having only 512 Mb of RAM, therefore all of the number crunching is done in the main application and then fed to the Raspberry Pi through

the UDP link. UDP offers a fast connection, a requirement to limit image lag in the OR, but has no guarantee of message integrity. Since the application is pulling information from the various input devices at a rate of 50 times a second a few misplaced packets is not mission critical.

The UDP client class packetizes all data from the main application before passing it to the UDP server. The main benefit of this architecture is that with our implementation there is a single link of communication. All data passes from the main application through the client to the server and from there to the correct hardware. The fast data transport enabled by this single, simple communication link helps to keep the latency to a minimum. While the simplicity of this link is a major asset, the fact that it is solitary (with no viable alternative) is a weakness. If this link is severed all control of the rover is lost and a system reboot is required. This is the architectural design around which this project will be based.

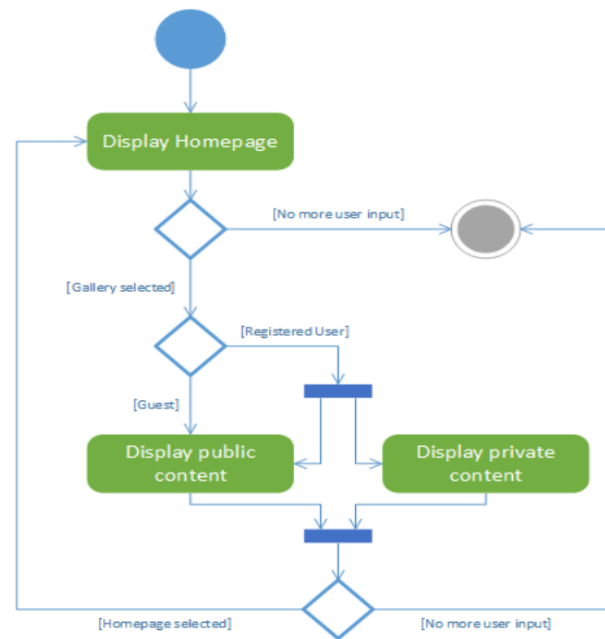


Figure 3. The initial activity diagram for the Rift Runner web site.

The Activity Diagram shown in see Figure 2 illustrates a high-level overview of user navigation options through the Rift Runner web site. Figure 4 on the other hand, illustrates rover control activities for the project. The user manipulates rover movements through a thumb stick and the UDP server is responsible for receiving, unpacking and pushing data to the hardware driver.

predetermined space. The rover control class converts these floats to integers that represent power and direction.

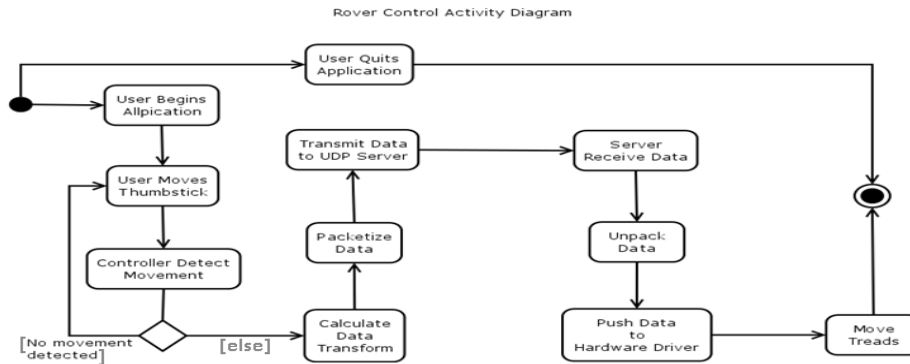


Figure 4. Rover Control Activity Diagram

The rover control class then sends these conversions to the UDP Client that packages them and sends them to the UDP server on the Raspberry Pi. The Pi sends the data to the PWM that sends the data to the motor driver where they become power designations for the individual treads that will send the rover in any desired direction.

**Camera Controls: Controller**

While the main purpose of this project is first person piloting through the Oculus Rift, the Rift is not necessary for first person piloting. When piloting without the Rift the right thumb stick of an

XBox 360 controller controls the movement of the camera. The controller feeds floating-point decimals to the camera control class in the main application. The floats represent the x, y position of the controlling joystick within a finite, predetermined space. The camera control class converts these floats to integers between 160 and 630, and 460 and 150. These values represent the desired location of the camera within the allowable movement spectrum. The values are then passed to the servo controller on the rover and the camera is moved to the desired position. The controller also controls image and video capture. Image capture is mapped to the left trigger, and video capture is mapped to the right trigger. When the left trigger is pressed,

Client-Server Architecture for the Rift Runner

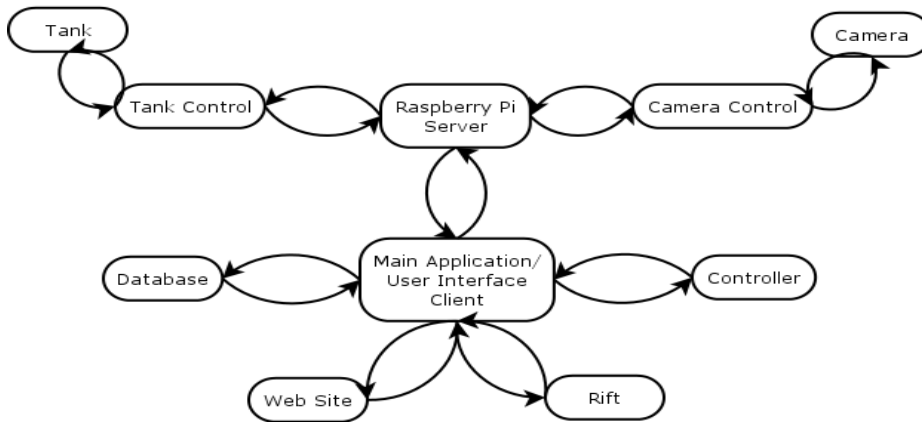


Fig. 5. Client-Server Architecture

**2.2 Subsystem Descriptions**

**Rover Controls**

The Rift Runner is controlled with the left thumb stick of an XBox 360 controller. The controller feeds floating-point decimals to the tank control class in the main application. The floats represent the x, y position of the controlling joystick within a finite,

whatever frame is currently on the camera is captured and pushed to the application, and from there the user has the option to save the image to the database or discard the image. For video capture the right trigger is depressed to signal the beginning of the capture and then depressed again to signal the end. The final feature of the controller camera control is a zero position button. In order to synch the Rift view and the camera position a zero position is needed. This zero position represents straight forward in regards to the rover heading and level with the horizon. This position is designated as 350, 350 within the servo space and when the A button is pressed the camera resets itself to this position and that is mapped to the current heading on the Rift.

### Camera Controls: Rift

The Oculus Rift controls the camera on the rover with the motion of the user's head. The OR pushes floating-point decimals to the rift camera control class in the main application. These floats represent direction of the user's gaze based on an x, y location from the midpoint of the OR viewer. The rift camera control class converts the floats to integers between 160 and 630 on the x-axis and 460 and 150 on the y-axis. These numbers represent the minimum and maximum allowable positions of camera movement.

### 3. Discussion and Lessons Learned

Over the course of the development cycle for the Rift Runner there were many factors that created limitations for the realization of the project goal. When everything is taken into consideration most of those can be placed under one of two headings: time or expertise. While each member of the group had individual expertise in areas that pertain indirectly to the project such as web development and programming, none of the team members had ever used those skills to produce anything like our final product. C++ was a new language to the team and we had to develop expertise in using the 3d printer, circuitry, and power management among others.

While the gathering of expertise made for a slow start, the hard deadline at the end of the semester made that slow start stressful. As we are all students, even mentioning time as a limiting factor seems like a "cop out". It is largely a limitation that we placed on ourselves, but in an environment devoted solely to development, our use of time would have been more focused. The Mythical Man-Month defines average output at 10 line of code per day. That definition implies a typical professional working environment, i.e. 8 hour workdays, of which 5 every week (at least)

are applied to the project. This schedule does not necessarily apply to all projects. In fact not once was there a day that did not involved the development of this project. The fact that we finished our project, with complete functionality, under these conditions makes us proud.

The main application and the Rift camera control class were developed by two different team members. They discussed what they were trying to achieve, made sure they were on the same page and built their code. One developer worked in Netbeans and the other used Microsoft Visual Studio, both applications were written in C++. When it came time to marry the code and test their compatibility we had little if any problems. It turns out that while Netbeans and Visual Studio are both IDE's that can be used to develop C++ applications they have slightly different library usage. That difference is enough to make applications native to one incompatible with the other. The Netbeans developer switched his IDE and the problem was quickly corrected.

Motors and motor drivers are necessary demons if one wishes to move some vehicle electronically. The motor driver that we chose to use has two inputs for power, a logic circuit and a power circuit. If the power circuit receives no charge then the motor will simply receive no power and will not move a tread. On the other hand, if the logic circuit receives no power the circuit board will overload and be destroyed. Fortunately we discovered this in research, not firsthand. It is imperative that we remember and heed this.

The last important piece of functionality that we developed was the video streaming. For early testing of the UDP connection a patchwork pipeline of tools was used to stream video. Once we tried to use this pipeline as our primary method we discovered that too much latency was present to make this a viable solution. Background research into video streaming revealed two likely candidates as fixes to our woes: OpenCV and GStreamer. The first attempt for a solution involved GStreamer, in retrospect it would have been advantageous to use OpenCV for the first attempt. GStreamer did not improve out latency issues. Once the OpenCV library became the primary solution focus latency improved and the project became closer to a success. The biggest hurdle during this process was cross platform compatibility, or lack thereof. The Raspberry PI OS is Linux based and the application was designed to run in a Windows environment, OpenCV was the only library we found that allowed us to easily implement this cross platform functionality.

### 3.1 Lines of Code Comparison

Table II shows a summary of our predictions for the number of lines of code for the software side of our project. At the bottom of the table are estimates for the number of man-months required to complete the project as well as budgetary estimates. Our initial estimates were very naïve.

### 3.1 Lines of Code Comparison

Section	LOC Initial	LOC Design	LOC Final
Main Application	300	300	565
Vehicle – Controller	50	315	390
Vehicle – Control Transmission	200	*165	*360
Camera – Rift Motion Tracking	10	60	90
Camera – Control Transmission (Rift)	100	*165	*360
Camera - Control Transmission (Controller)	100	*165	*360
Camera – Image/Video Capture	50	50	35
Web Site – Home Page with Login/Logout/Register	200	200	150
Web Site – Image/Video Archive	100	100	755
Web Site – User Profile/Admin	250	250	50
Database – User Profiles	50	50	10
Database – Image/Video Link Storage	50	50	10
Total:	1460	1540	<b>2415</b>
Duration: Requirements – 7 man-months Design – 8 man-months Final – 12 man-months			
Cost: Requirements - \$41,850 Design - \$47,300 Final - \$71,000			

**Table II.** Lines of Code comparison for the three phases of reporting.

Once we began working in the design phase of the project we reevaluated our estimates and found we had misjudged the number of lines of code, and by extension the amount of time and money, needed. As project completion came nearer it became clear just how misinformed we were even after the second estimations were calculated. Table II shows that the completed project contains nearly 900 more lines of code than we estimated during the Design phase. As a result of this underestimation, we nearly doubled the initial cost. Luckily this was an academic endeavor; and as such the cost was primarily in creating more work for ourselves.

## 4. Conclusion

It was our main objective to create a fun, usable, immersive piloting experience. We believe we realized our dream. The software development for the control aspects of the Rift Runner followed an iterative paradigm and software testing was executed similarly. While testing was more concerned with functionality it would be incorrect to say that testing was entirely black-box. As each class was built it was white-box tested based on its internal features. Testing began with the controller class to ensure that the output generated was precise, and to make sure the transformations produced the correct numbers required by the hardware.

The UPD client and server were next developed and tested to ensure connectivity. This process was repeated with every new addition to the software.

The overall testing cycle was: develop, then test, then debug, and then make sure it plays well previously developed components and the hardware. When we finally determined that development was complete a thorough test of the entire system was undertaken, and we were blown away with the great results. There is an immense feeling of satisfaction when everything works together well as was the case here.

## References

- [1] Lewis, Shauna. "Age not a factor among radio-controlled vehicle enthusiasts." *Eagle, The (Bryan, TX)* 14 July 2011: *Newspaper Source Plus*. Web. 3 Nov. 2013.
- [2] Singer, P.W.. "AMERICA'S NEW KILLING MACHINES; A ROBOTICS REVOLUTION IS CHANGING THE WAY WE DO BATTLE. BUT WILL THE 'UNMANNING' OF WAR DO US IN?." *Record, The (Kitchener/Cambridge/Waterloo, ON)* n.d.: *Newspaper Source Plus*. Web. 3 Nov. 2013.
- [3] "[Department of Defense Fiscal Year \(FY\) 2011 President's Budget Submission](http://www.saffm.hq.af.mil)". <http://www.saffm.hq.af.mil>. February 2010. pp. 4–118. Retrieved 28 October 2013.
- [4] Wallace, Kenyon. "Florida cops using drone with cameras." *Toronto Star (Canada)* n.d.: *Newspaper Source Plus*. Web. 3 Nov. 2013.

- [5] Knapp, Alex. "Curiosity Successfully Lands On Mars." *Forbes.Com* (2012): 10. *Business Source Complete*. Web. 3 Nov. 2013.
- [6] Luckey, Palmer. "Oculus Rift: Step Into the Game." *Kickstarter*. N.p., 26 Sept. 2012. Web. 27 Oct. 2013. <http://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game/posts/316239>
- [7] Welsh, Oli. "John Carmack and the Virtual Reality Dream." *Eurogamer.net*. EuroGamer, 7 June 2012. Web. 27 Oct. 2013. <<http://www.eurogamer.net/articles/2012-06-07-john-carmack-and-the-virtual-reality-dream>>.
- [8] Horsey, Julian. "Oculus Rift Virtual Reality Headset." *Geeky Gadgets*. N.p., 27 Sept. 2012. Web. 27 Oct. 2013. <<http://www.geeky-gadgets.com/oculus-rift-virtual-reality-headset-developer-kits-now-available-to-pre-order-video-27-09-2012/>>.
- [9] "Oculus Rift Virtual Reality Headset Gets Kickstarter Cash." *BBC News*. BBC, 1 Aug. 2012. Web. 27 Oct. 2013. <<http://www.bbc.co.uk/news/technology-19085967>>.
- [10] Lang, Ben. "Watch the QuakeCon Virtual Reality Keynotes Here." *Road to Virtual Reality*. N.p., 5 Aug. 2012. Web. 27 Oct. 2013. <<http://www.roadtovr.com/watch-the-quakecon-virtual-reality-keynotes-here/>>.
- [11] Oculus Rift: Virtual Reality 2.0. By: Mangalindan, J. P., *Fortune*, 00158259, 6/10/2013, Vol. 167, Issue 8
- [12] Thier, Dave. "CEO Promises Oculus Rift Won't Make You Sick." *Forbes*. *Forbes Magazine*, 17 Oct. 2013. Web. 27 Oct. 2013. <<http://www.forbes.com/sites/davidthier/2013/10/17/ceo-promises-oculus-rift-wont-make-you-sick/>>.
- [13] "Virtual Museum, Searchable Database of European Fine Arts (1000-1900)." *Web Gallery of Art*. N.p., n.d. Web. 27 Oct. 2013. <<http://www.wga.hu/frames-e.html?html/p/peruzzi/farnesi3.html>>.
- [14] Allen, C. H., A. B. Doty, and E. C. ... McCormick. "Space Flight Simulator for U.S. Air Force Aerospace Research Pilot School." *Journal of Spacecraft and Rockets* 3.6 (1966): 793-99. Print.
- [15] Gonzales, D., David R. Criswell, and Ewald Heer. *Automation and Robotics for the Space Exploration Initiative: Results from Project Outreach*. Santa Monica, CA: Rand, 1991. Print.
- [16] "Software Engineer: Median Salary". [http://www.payscale.com/research/US/Job=Software\\_Engineer/\\_Developer/\\_Programmer/Salary](http://www.payscale.com/research/US/Job=Software_Engineer/_Developer/_Programmer/Salary)>. 2013. Retrieved 28 October 2013.
- [17] Brooks, Frederick P. Jr. (09 1983). "[The Mythical Man-Month](#)". *PC Magazine* 2 (4): 210–240.
- [18] Danielsson, Torkel. "Oculus FPV – First Flight." <http://intuitiveaerial.com/home/2013/7/14/oculus-fpv-first-flight>>. July 2013. Retrieved 3 December 2013.
- [19] R. France and B. Rumpe. Model-driven Development of complex software: A research roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 37.54, Washington, DC, USA, 2007. IEEE Computer Society.
- [20] B. Selic. The pragmatics of model-driven development. *IEEE Software.*, 20(5):19.25, 2003.
- [21] Simmonds, D. M., Reddy, Y. R., Song, E. and Grant, E. "A Comparison of Aspect-Oriented Approaches to Model Driven Engineering", in *Proceedings of the International Conference on Software Engineering Research and Practice, (SERP), 2009*.
- [22] The Object Management Group (OMG). *Unified Modeling Language: Superstructure. Version 2.2, Final Adopted Specification*, OMG, <http://www.omg.org/uml>, February 2010.





## **SESSION**

# **EMBEDDED SYSTEMS + HPC + SENSORY DEVICES + NETWORK ON CHIP SYSTEMS AND APPLICATIONS**

**Chair(s)**

**TBA**



# Reducing DDR Latency for Embedded Image Steganography

J. Haralambides<sup>1</sup> and L. Bijaminas<sup>1</sup>

<sup>1</sup>Department of Math and Computer Science, Barry University, Miami Shores, FL, USA

**Abstract** - Image steganography is the process of encoding an image within another, larger image and is considered an encryption technique. Generalized versions of the technique enable the encryption of various data forms including text messages, files, and multimedia. Extensive research helps produce encrypted data that withstand advanced cryptanalysis. An FPGA implementation of the algorithm on an Atlys Spartan-6 development board is presented here based on Least Significant Bit replacement. Pixel mapping is performed randomly using a Galois LFSR to protect against cryptanalysis. The host image is stored on DDR2 memory utilizing a dual, bidirectional (read/write) FIFO. Reduced read DDR latency is achieved by extending LSB replacement from one to two least significant bits and by generating random blocks of four addresses. Each four-pixel block of the host image yields a single pixel of the hosted image. Write latency can be improved if RAM FIFOs are used in the memory controller.

**Keywords:** Embedded design, memory latency, encryption, image steganography, FPGA, LFSR.

## 1 Introduction

Image steganography is a well-known encryption technique that allows for a smaller image to be concealed within a larger host image. In its generalized form it allows for encryption of various data forms such as text, data files, and multimedia content [1, 7]. For uncompressed images in the spatial domain this can be achieved by replacing pixel bits of the host image by pixel bits of the hosted image. A common method of mapping involves the least significant bit (LSB) of a pixel. It is not uncommon for a larger number of lower significance bits to be replaced or modified as in LSB2 or LSB3 mapping where two or three least significant bits may be affected, respectively. In color images in RGB mode or equivalent, bit replacement may be carried over all color channels in similar fashion. LSB replacement preserves the quality of the host image and makes it undetectable to the human eye.

Various cryptanalysis techniques have been devised to detect pixel intensity alteration including data analysis [2, 6]. Such techniques may involve visual or audible detection for image or audio host files, statistical or structural analysis (pixel patterns, histograms, timestamp or content modification, checksum) [8].

The importance of steganography has led to hardware implementations of the algorithm using programmable logic [3, 4]. Embedded designs can be optimized to reduce the size and cost of the product and increase its reliability and performance. Such designs are tested for resource utilization, maximum clock speed attained, and memory space or buffering requirements.

We have implemented our design in VHDL on an Atlys Spartan-6 development board using the Xilinx ISE Design Suite platform. The “cover” or host image as well as the hosted image are stored in a 256 MB on-board DDR2 memory. A memory controller that employs two read/write FIFOs interfaces user logic. Both FIFOs hold 64 32-bit words capable of storing 64 image pixels in ARGB mode (color depth of 24 bits including eight bits for the alpha channel). This configuration enables on-line memory communication with the board’s HDMI I/O ports. Image data are transferred to the board using a simplified USB protocol that utilizes the board’s JTAG port and its FX2 microcontroller. A third read/write FIFO is employed to facilitate the transfer. This FIFO operates at a maximum clock rate of 48 Mbps.

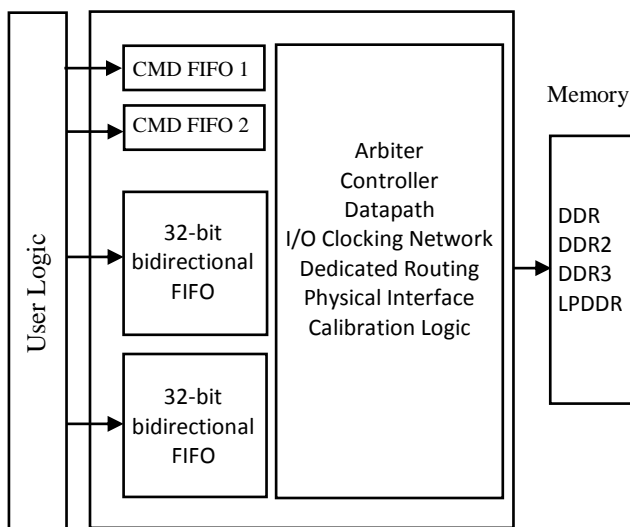
Pixel mapping between the host and the hosted image is accomplished using a pseudo-random number generator that produces non-repeating number sequences. A Galois Linear Feedback Shift Register (LFSR) is implemented for this purpose [5, 6]. Address mapping is performed based on the hosted image thus reducing overall addressing requirements. The algorithm employs the LSB2 method whereby the two least significant bits of host image pixels are replaced by selected bits of the hosted image. Reading and writing of pixels for the host image is done sequentially in bursts that equal FIFO size. Reconstruction of randomly selected pixels for the hosted image requires processing of consecutive pixel blocks of size four resulting in the reconstruction of 16 pixels per 64-pixel burst. The above algorithmic parameters allow for a substantial reduction of memory latency caused by buffering and smaller size bursts inherent in random addressing schemes.

The rest of the paper is organized as follows: In Section 2 we give a description of the algorithm and all relevant hardware components, while in Section 3 we propose an extension to the implementation to achieve reduced write latency. Future work, Conclusions, and References follow in Sections 4, 5, and 6, respectively.

## 2 The algorithm

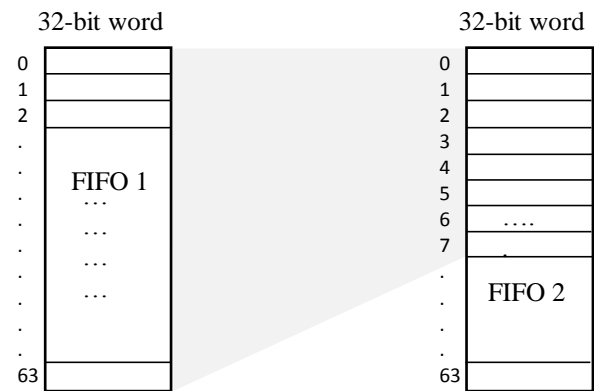
Before we give the details of the encryption/decryption algorithm, we will describe the characteristics of the memory controller for the on-board DDR2 memory that is generated by the Xilinx Memory Interface Generator (MIG).

We have elected a design that employs two bidirectional FIFOs to allow for interleaved read and write bursts for the cover and encrypted image, respectively. A read burst loads the first FIFO with pixels of the cover image while a write burst loads the second FIFO with pixels of the hosted image. Operation control is handled by corresponding command FIFOs. A simplified architecture is shown in Figure 1. Each bidirectional FIFO is capable of holding 64 32-bit words. Each word represents a pixel in ARGB mode (Alpha, Red, Green, Blue). This word configuration makes parallel processing of color channels possible. A third read/write FIFO that is used for the transfer of image data from the host computer to the FPGA board using a simplified USB protocol is omitted here for clarity purposes.



**Figure 1.** Spartan 6 Memory Controller Block (simplified).

Least Significant Bit (LSB) replacement is a common method to encrypt pixels of the smaller, hosted image to a larger cover image. In cases where protection against cryptanalysis is not pursued, pixel mapping can be performed sequentially achieving low DDR read and write latency. A single bit replacement per channel requires eight pixels of the host image to host or produce (during decryption) a single pixel of the hidden image. It also limits the size of the smaller image to that of one eighth of the host. A single 64-pixel read burst from DDR2 to FIFO 1 results in an 8-pixel write to FIFO 2. This constitutes stage 1 of the process. The process repeats in stages 2 to 8 followed by a 64-pixel write burst from FIFO 2 to DDR2. This approach makes full use of both FIFOs and is depicted in Figure 2.



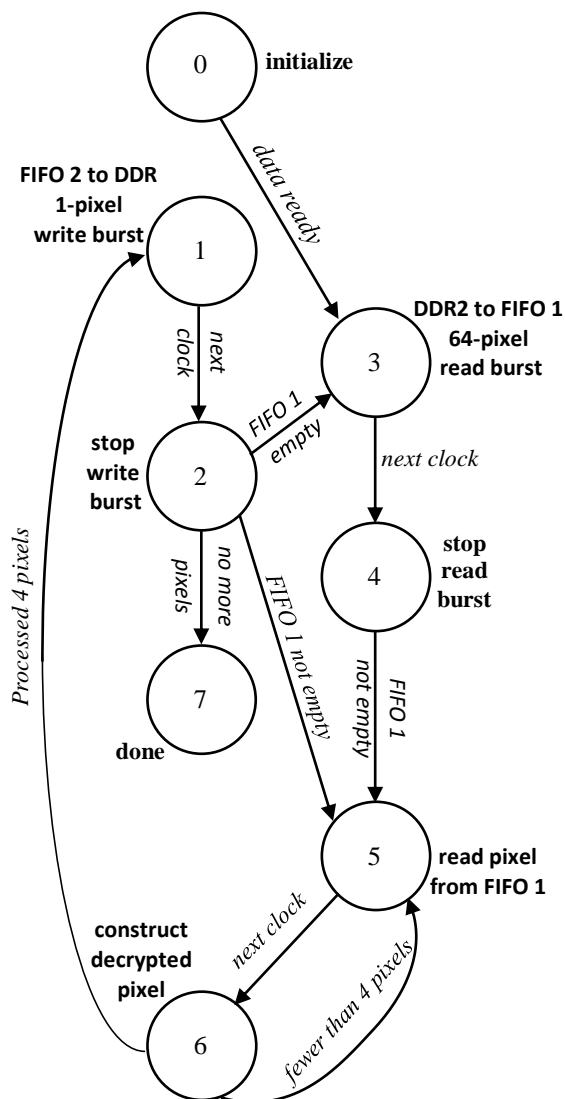
**Figure 2.** LSB single bit replacement, sequential encoding during first stage.

The above method is characterized by low latency but does not provide protection against steganalysis. For this reason, we perform pixel mapping pseudo-randomly using a Galois LFSR and LSB2 mapping. The generated addressing sequence is non-repeating and is only tested against image address boundaries. Our experiments involve host images of  $640 \times 480$  resolution for a total of 307,200 pixels or 1,228,800 bytes. Encrypted images are one quarter the size of the host image at a resolution of  $320 \times 240$  for a total of 76,800 pixels or 307,200 bytes. While DDR2 is organized as a byte-addressable unit, read and write bursts are carried at the pixel level (FIFO word size). This reduces addressing requirements for the LFSR component from 19 to 17 bits. Random addressing will occur within the image resolution boundaries specified for the encrypted image and, therefore, a total of 76,800 different addresses need be generated. A 17-bit Galois LFSR is capable of generating a total of  $2^{17} - 1 = 131,071$  addresses. In case of byte-level access, the addressing range would rise to 307,200 different addresses in which case a 19-bit Galois LFSR would be required. For random addressing performed at the pixel and byte level of the host image, these requirements necessitate the use of 19-bit and 21-bit LFSRs, respectively.

In our method, we have extended LSB replacement to 2 bits for the following reasons: a) it allows for encryption of larger images up to one-fourth of the host image, b) it reduces the number of clock cycles during the reconstruction (decryption phase) or distribution (encryption phase) of the pixels of the hosted image, and c) it has a comparable visual effect to 1-bit replacement.

A second key feature of our method is that reads from DDR2 during decryption (and, equivalently, writes during encryption) are performed in pseudo-random sequences of 64-pixel bursts. Each pixel block read results in the reconstruction of 16 pixels of the hosted image. Similarly, during encryption, 16 randomly selected pixels of the hosted image will be mapped in 64 consecutive pixels of the host image. This is a minor compromise of the mapping randomness that offers a substantial reduction in memory latency. The FSM (Finite

State Machine) depicted in Figure 3 gives an insight to the reduced latency steganography algorithm for the decryption phase. A more detailed description of the state machine follows.



**Figure 3.** FSM for steganography, decryption phase.

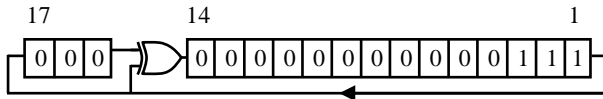
State 0 serves as the initialization state. The address of the cover image is set at 0 and that of the image to be decrypted is set at 921,600 ( $640 \times 480 \times 3$  bytes/pixel). Initialization is directly followed by state 3. During this state the command FIFO of the memory controller is set up for a read burst of 64 pixels. The address for the cover image is incremented by 256 (64 pixels  $\times$  4 bytes/pixel) for the next read burst. Data reading takes place in state 4 and the command FIFO is deactivated. Data are transferred from DDR data banks to FIFO 1 of the memory controller. Transition to state 5, the next state, occurs when signal `fifo_empty` is deasserted for FIFO 1, indicating data availability in the FIFO. At the same time, reading from FIFO 1 is enabled (FIFO data will be available in the next state). During state 5, pixel data for each of the red, green, and blue channels are placed into 8-bit shift registers. More

specifically, the two least significant bits of each of the channels are stored in the two most significant positions of the shift registers. State 6 that follows and state 5 enter into a loop that runs four times, thus acquiring all eight bits of the color channel for one pixel of the decrypted image. In state 6, shift registers shift pixel data two positions to the right making room for the next pair of pixel data. Completion of the loop leads to a write operation of pixel data to FIFO 2 and transition to state 1 where the command FIFO is set up for a write burst of one pixel. The address for the decrypted image is incremented by 4 (1 pixel  $\times$  4 bytes/pixel) to prepare for the next pixel. State 2 follows at which the command FIFO is deactivated and the decrypted pixel is written to DDR. Upon assertion of the signal `fifo_empty` of FIFO 2, the steganography process repeats if more pixels need be examined (visiting state 3 for another read burst, if FIFO 1 is empty, or state 5, if not) or terminated, otherwise (visiting state 7). The algorithmic description for the decryption phase is provided in Figure 4.

- |                |   |
|----------------|---|
| <b>Step 0.</b> | Initialize  |
| a.             | Set address of host image to 0.   |
| b.             | Set address of decrypted image to 921,600.                                      |
| c.             | Go to step 3.   |
| <b>Step 1.</b> | Set up command FIFO for write   |
| a.             | Set mode to write and burst size to 1 word.                                     |
| b.             | Set address to encrypted address.   |
| c.             | Increment host address by 1 pixel (4 bytes).                                    |
| d.             | Go to step 2.   |
| <b>Step 2.</b> | Write decrypted pixel to DDR  |
| a.             | Deactivate command FIFO.  |
| b.             | If FIFO 2 is empty  |
| i.             | If all pixels are processed, go to step 7.                                      |
| ii.            | Otherwise,  |
| 1.             | If FIFO 1 is empty, go to step 3.   |
| 2.             | Otherwise, go to step 5.  |
| <b>Step 3.</b> | Set up command FIFO for read  |
| a.             | Set mode to read and burst size to 64 words.                                    |
| b.             | Set address to host address.  |
| c.             | Increment host address by 64 pixels (256 bytes).                                |
| d.             | Go to step 4.   |
| <b>Step 4.</b> | Read data into FIFO 1   |
| a.             | Deactivate command FIFO.  |
| b.             | If FIFO 1 is no longer empty, go to step 5.                                     |
| <b>Step 5.</b> | Read pixel data   |
| a.             | Read two LSBs per color channel into two MSBs of corresponding 8-bit registers. |
| b.             | Go to step 6.   |
| <b>Step 6.</b> | Construct decrypted pixel   |
| a.             | Shift registers to the right by two bits.                                       |
| b.             | If four shifts were performed   |
| i.             | Write pixel to FIFO 2.  |
| ii.            | Go to step 1.   |
| c.             | Otherwise, go to step 5.  |
| <b>Step 7.</b> | Terminate process   |

**Figure 4.** Steganography, decryption phase.

Since addressing is carried out in blocks of four pixels, the LFSR random number generator for this method requires 17 bits. The corresponding feedback polynomial is:  $x^{17} + x^{14} + 1$ . A 17-bit Galois LFSR with an example value of 7 is displayed in Figure 5. The next value generated will be 73731. The current value of 7 is shifted one position to the right and a least significant bit value of 1 causes bits 17 and 14 to be complemented.



**Figure 5.** A 17-bit Galois LFSR.

The 17-bit Galois LFSR cycles through a maximal number of 131071 states ( $2^{17} - 1$ ). State 0 is never reached. Cycling within this period generates unique numbers that will represent non-repeating random memory addresses. Different starting values result in different random sequences. A shared key (starting value) between the sender and receiver of hidden images provides for a more secure encryption.

Encrypted images used for our implementation have a resolution of  $320 \times 240 = 76,800$  pixels requiring address values between 0 and 76799. Random numbers in excess of image resolution are skipped until a valid address is generated. In the special case of state 76800, an address of 0 is returned. To eliminate delays caused by invalid addresses, random numbers generated by the LFSR are stored in a 16-word FIFO having a word size of 17 bits. Simulation experiments have shown that the size of the FIFO is sufficient to avoid any such delays. The LFSR number generator operates independently and continuously as long as the underlying FIFO is not full.

**Table 1.** Device utilization summary (estimated values).

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	388	54576	0%
Number of Slice LUTs	705	27288	2%
Number of fully used LUT-FF pairs	308	785	39%
Number of bonded IOBs	81	218	37%
Number of BUFG/BUFGCTRLs	3	16	18%
Number of PLL_ADVs	1	4	25%

Table 1 shows the device utilization values for the implementation of the algorithm on the Atlys Spartan-6 development board. The report does not take into consideration modules required for image data transfer between the host computer and DDR memory on the board. It reflects the hardware required for the memory controller module and the steganography state machine.

### 3 Reducing memory latency

The algorithm presented in the previous section focuses on the reduction of memory delays due to random read bursts from DDR to FIFOs. These problems are alleviated by pixel blocking and 2-bit LSB replacement. In case a replacement method uses no pixel blocking, a clock cycle is dedicated to setting up the command FIFO for a single-pixel read burst for all pixels of the hosted image. For a hosted image having a resolution of  $320 \times 240 = 76,800$  pixels, a total of  $76,800$  cycles is dedicated to command FIFO setup. On the other hand, the total number of clock cycles for our pixel blocking method is dramatically reduced to  $76,800/64 = 1,200$  clock cycles. In addition to pixel blocking, LSB2 allows for the encryption of images twice as large as images using the LSB1 method at the same amount of time.

Memory latency is reduced further when consecutive addresses are accessed in a single burst as opposed to the same number of random addresses accessed in multiple bursts. Read performance due to random addressing is further deteriorated for DDR memories utilizing more than one data banks as latency for such random memory accesses increases substantially. User guide 388 published by Xilinx, Inc. offers an additional insight to memory performance as it relates to the command, read, and write FIFOs of the memory controller for Spartan-6 FPGAs.

### 4 Future work

While pixel blocking reduces memory delays due to read bursts, pixel writes are performed at single-pixel bursts. An additional improvement may be obtained if a two-level blocking technique is used. In this direction, use of a dual LFSR structure is required. The first random number identifies a block of 64 pixels from the entire image address space of the host image. These pixels will be used to construct 16 pixels of the hosted image. The second LFSR generates random numbers in the range 0 to 15 for intra-block addressing. Assuming a host image having a resolution of  $640 \times 480 = 307,200$  pixels, a total of  $307,200/64 = 4,800$  blocks must be accessed. A 13-bit Galois LFSR provides block addressing for all 64-pixel blocks of the host image as it is capable of generating  $2^{13} - 1 = 8,191$  addresses. A 5-bit Galois LFSR generates non-repeating sequences of all 16 address offsets within the block. Due to rearrangement of target addresses (addresses for the hosted image), memory controllers for DDR need to employ RAM FIFOs. Such FIFOs will enable pixel to FIFO writes at random FIFO addresses thus eliminating the need of additional registers and extra clock cycles.

## 5 Conclusions

We have implemented a reduced DDR latency image steganography algorithm on an Atlys Spartan-6 development board. Encryption and decryption are carried out using pseudo-random number generators to withstand cryptanalysis. Non-repeating addressing sequences are produced through the use of a Galois LFSR. Images are given in the spatial domain and have not been subjected to compression. They are stored in on-board DDR2 of the programmable device and are accessed in read and write bursts using bidirectional 64-pixel, 32-bit FIFOs. Pixels are word-sized in the ARGB format. An immediate reduction in clock cycles can be achieved if the least significant bit (LSB) replacement process is extended to include two bits of the host image (LSB2). In addition to added capacity for the hosted image, one half of pixel reads are sufficient to encrypt/decrypt a pixel with no visual degradation of the cover image. Additional improvements are seen in comparison to single pixel bursts when 64-pixel blocks are fetched from memory and processed as groups of 4 pixels. Each group results in the reconstruction of a pixel for the hosted image (decryption phase). If write latency reduction is desired, the present implementation can be extended to two-level random address mapping. This will require modification of memory controller FIFOs to accommodate random access of FIFO locations.

## 6 References

- [1] C. P. Sumathi, T. Santanam, and G. Umamaheswari, "A Study of Various Steganographic Techniques Used for Information Hiding", *International Journal of Computer Science & Engineering Survey (IJCSSES)*, Vol. 4, No. 6, pp. 9 – 25, December 2013.
- [2] S. Lyu , H. Farid, "Steganalysis using higher-order image statistics", *IEEE Transactions on Information Forensics and Security*, Vol. 1, pp. 111 – 119, 2006.
- [3] B. J. Mohd, S. A. Abed, T. Al-Hayajneh, and S. Alouneh, "FPGA Hardware of the LSB Steganography Method", *International Conference on Computer, Information and Telecommunication Systems (CITS)*, pp. 1 – 4, May 14 – 16, 2012.
- [4] B. V. Lakshmi, B. V. Raju, "FPGA Implementation of Lifting DWT based LSB Steganography using Micro Blaze Processor", *International Journal of Computer Trends and Technology (IJCTT)*, Vol. 6, No. 1, pp. 6 – 14, December 2013.
- [5] A. K. Panda, P. Rajput, B. Shukla, "FPGA Implementation of 8, 16 and 32 Bit LFSR with Maximum Length Feedback Polynomial using VHDL", *International Conference on Communication Systems and Network Technologies*, pp. 769 – 773, May 11 – 13, 2012.
- [6] J. Fridrich, M. Goljan, and R. Du, "Reliable Detection of LSB Steganography in Color and Grayscale Images", *IEEE Multimedia*, Vol. 8, pp. 22 – 28, 2001.
- [7] N. Provos, P. Honeyman, "Hide and Seek: An Introduction to Steganography", *IEEE Security and Privacy*, Vol. 1, No. 3, pp. 32 – 44, May 2003.
- [8] S. Lyu and H. Farid, "Steganalysis using higher-order image statistics", *IEEE Transactions on Information Forensics and Security*, Vol. 1, pp. 111 – 119, 2006.

# Dynamic-prelink: An Enhanced Prelinking Mechanism without Modifying Shared Libraries

Hyungjo Yoon<sup>1,2</sup>, Changwoo Min<sup>1</sup>, and Young Ik Eom<sup>2</sup>

<sup>1</sup>Samsung Electronics, Suwon, Gyeonggi-do, Korea

<sup>2</sup>Sungkyunkwan University, Suwon, Gyeonggi-do, Korea

**Abstract**—Prelink accelerates the speed of program startup by fixing the base address of shared libraries. However, prelink prevents the dynamic linker from loading shared libraries by using Address Space Layout Randomization (ASLR) in runtime because it modifies the program header in binary files directly.

To resolve this problem, we introduce an enhanced prelinking mechanism, called *dynamic-prelink*, which separates the memory address layout per program as well as keeps high performance of prelink mechanism. *Dynamic-prelink* records prelinked contents to a file instead of modifying shared libraries. This makes dynamic linker be able to use both ASLR and prelink mechanism. Our experimental results show that the memory address layout of *dynamic-prelinked* programs is separated per program and the dynamic linker is able to randomly load shared libraries regardless of *dynamic-prelinking*. In addition, the startup time of *dynamic-prelinked* program becomes faster than common program in the dynamic linker, about 42% on average.

**Keywords:** Dynamic linker, Prelink, ASLR, Shared library, Program startup

## 1. Introduction

The dynamic linker loads shared libraries and executables, and relocates the memory address layout of each binary. Since it affects every program on the system, dynamic linking is critical in speeding up the program startup.

Prelink accelerates the booting time and program launching time in various operating systems. Jelínek[1] found that prelink reduces processing time of dynamic linking by about 83%, when GTK+ applications are evaluated. In Android, which is most popular system in mobile devices, prelink reduces the booting time by 5%, in practice [2].

But prelink has significant drawbacks. Prelink and *Address Space Layout Randomization* (ASLR) cannot be used simultaneously on a system [2][3][14][16]. ASLR randomizes the layout of memory including stack, heap, library, and

executable for enhanced security level of system. According to this, it is difficult for an attacker to expect the randomized address of processes, and so attacks can be defeated ultimately. ASLR is valuable for defending control flow hijacking attacks and *return-to libc* (RTL) attacks [2][4][5][12].

In order to enhance the secure execution, the recent operating systems adapt ASLR rather than applying prelink. Android supported prelink up to the version of Ice-cream 4.0, but it supports ASLR in current version [10]. Several linux's distributions support PaX's implementation of ASLR by default [15]. Windows supports ASLR from Windows Vista [11], and Mac OS X and iOS supported the preload mechanism similar to prelink in the past, but ASLR is supported from the version of Mac OS X 10.8 and iOS 4.3 in the entire system [8][9].

Prelink is partially adopted per program or it is adopted in the whole system. In case that prelink is adapted per program, prelinked program should not influence the launching procedure of the other programs. But a prelinked program prevents the dynamic linker from loading common programs using ASLR on the operation system due to modifying shared libraries directly. Prelink switches the base address, which is described as PT\_LOAD segment in program header, from zero to a new value [1]. Thereafter, the dynamic linker let mmap() load libraries into arbitrary address in the memory by referencing PT\_LOAD segment. If PT\_LOAD is not equal to zero and the indicated address is allocable memory space, the kernel allocates a binary in virtual address of PT\_LOAD [6][7]. Therefore, the dynamic linker cannot always load shared libraries related to prelinking in random address if a prelinked program exists on the system. For example, we assume that a program is already prelinked. The dynamic linker always loads libc.so in the same address for every process due to fixed libc.so's program header, and then applying ASLR is limited. This causes a security problem under RTL attacks in which attacker injects malicious codes into the fixed address of libc.so [4].

To resolve this problem, we introduce novel prelinking mechanism, called *dynamic-prelink*, which separates the memory address layout per program as well as keeps high performance of prelinking mechanism. Our scheme separates the memory address layout per program and records prelinked contents to a file instead of modifying shared libraries. We make the following contributions in this paper.

- Young Ik Eom is the corresponding author of this paper.
- This research was supported by the MSIP (Ministry of Science, ICT&Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014(H0301-14-1020)) supervised by the NIPA (National IT Industry Promotion Agency).



- We find an efficient mechanism that maintains the performance of prelink and supplements prelink's weaknesses.
- We design an enhanced prelinking mechanism to perform prelinking without modifying shared library.
- We discuss the challenges to handle the separate memory address layout for each prelinked program.

We evaluate our dynamic-prelinking mechanism by comparing the performance of dynamic linker during starting each application which applies dynamic-prelink, ASLR, and original prelink. We get a result that dynamic-prelink is better than ASLR, about 42% on average. Moreover, we also check the memory address layout of the loaded program that is generated by the dynamic-prelinking mechanism to prove that programs can be loaded using both dynamic-prelink and ASLR on a system.

In the rest of the paper, we analyze the overhead of dynamic linker and the effect of prelink in Section 2. Section 3 introduces the design idea of dynamic-prelink, and we address the implementation of dynamic-prelink in Section 4. Section 5 evaluates the implementation and discusses the effect of dynamic-prelink. Section 6 concludes and discusses future works.

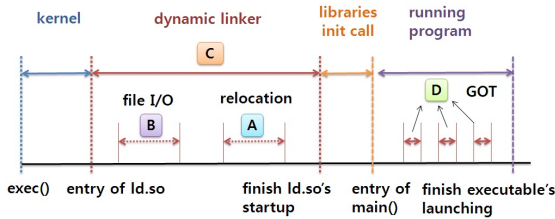


Fig. 1: Execution time during executable's launching.

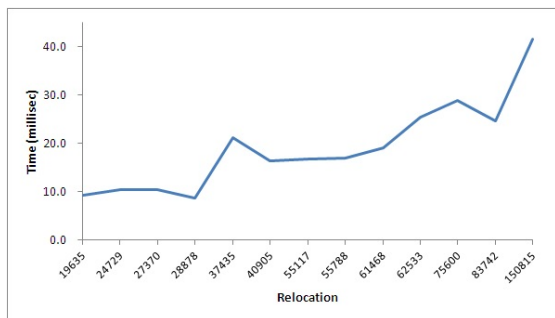


Fig. 2: Spent time in the dynamic linker according to the number of relocations.

## 2. Analysis of the dynamic linker and prelink

Most of time in the dynamic linker is spent in file I/O, relocations handling, and symbol lookups to load binaries

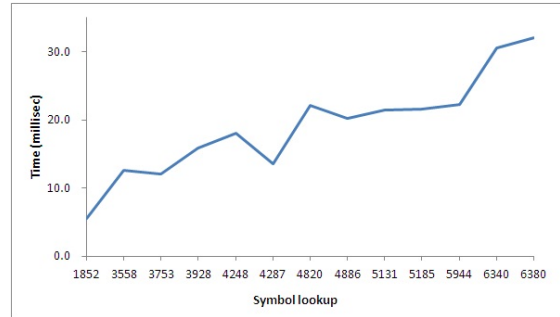


Fig. 3: Spent time in the dynamic linker according to the number of symbol lookups.

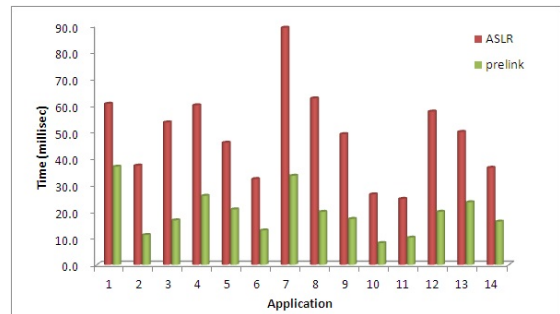


Fig. 4: Comparison of spent time in the dynamic linker during prelinked executable's launching and original executable's launching.

such as libraries and executables [1]. Every time, this operation is applied to memory pages which is written to be loaded into the memory when program is started through exec() in the dynamic linker [1]. The dynamic linker is hard to enhance the performance by controlling file I/O, but relocations handling and symbol lookups are able to be handled by the dynamic linker. So developing algorithm to reduce spent time of relocations handling and symbol lookups is rather efficient to enhance the performance of dynamic linker. Increase in the number of relocations and the number of shared libraries make the dynamic linker spend more time to search symbol scope and do symbol lookups [1]. Additional factor increasing cost is the length of symbol names mangled by C++. It makes the dynamic linker spend more time to find symbols due to increasing cost of comparing symbols. GUI programs become more and more important in recent most of the desktop platforms and the mobile platforms. Additionally, the complexity of program is also increasing. It means future programs will contain more libraries, larger relocations, more symbols and longer the length of symbol [1].

But there is limitation to enhance the performance of dynamic linker although efficient algorithm is designed. It cannot make the number of relocations and the number of symbol lookups be decreased. In terms of same ELF

binaries, it is hard to reduce the number of relocations and the number of symbol lookups.

Both Figure 2 and Figure 3 show how the number of relocations and the number of symbol lookups influence the performance of dynamic linker in Intel i3 dual core 1.2GHz. Those show that spent time in the dynamic linker linearly increases according to enlarging the number of relocations and the number of symbol lookups. Prelink executes relocation handling and the *Global Offset Table (GOT)* resolving in advance to enhance effectively the performance of dynamic linker. Prelink is able to complete relocations except some relocation related `dlopen()` and conflicted information [1]. It is absolutely better method in terms of only the performance. To verify the performance of prelink in current computing environment, we tried to compare the performance of prelinked program with original program. We divided the main activity of dynamic linker up into two types of sections: spent time in the dynamic linker before calling executable's `main()` (Figure 1c) and spent time for symbol lookup to resolve the GOT during starting the program (Figure 1d). Startup time (Figure 1c) in the dynamic linker is almost spent for file I/O (Figure 1b) and relocations handling (Figure 1a). We assumed that the total of spent time in the dynamic linker during starting the program is the sum of spent time of two type of sections (startup time in the dynamic linker and symbol lookup time). Figure 4 shows the speed of dynamic linker that loads prelinked program is faster than the speed for original program about 60% on average and the performance is enhanced in all of tested 14 applications. As a result, prelink is still effective method in terms of the performance of dynamic linker.

Table 1: The ratio of sections related the relocation in PIC

	libc.so(byte)	libstdc++.so(byte)
(A) Modified area	60260	104460
(B) Whole loaded area	1724620	904164
(C) Ratio (B/A)*100	3.4%	11.5%

### 3. Design of the dynamic-prelink

Most shared libraries are generated to the *position-independent code (PIC)* ELF. The objective of PIC is to maximize sharing the code of shared libraries and to save the memory space. The section of ELF binary built in the PIC is divided up into three types of sections; *the location-independent section*, *the location-sensitive section*, and *the relocation section* [13]. The relocation section holds information related the location-sensitive section. The dynamic linker adjusts the location-sensitive section using the relocation section during starting the program. Prelink finishes relocating operation by adjusting shared library's sections related the relocation ahead of time. It helps the dynamic linker save time in runtime because relocation and

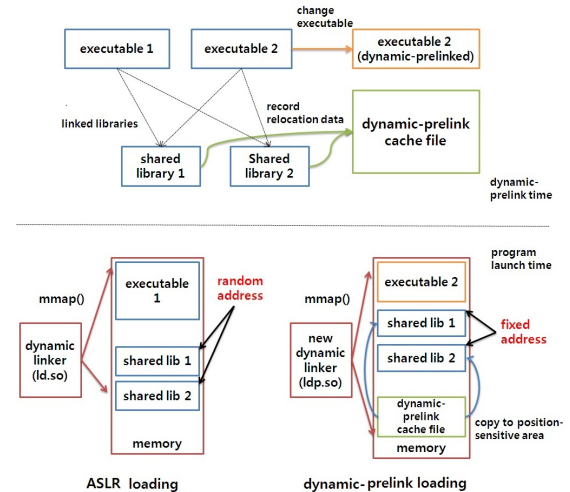


Fig. 5: The whole architecture of dynamic-prelink.

resolving the GOT are already finished. Table 1 shows the ratio of shared library's sections that are modified by prelink. The ratio of modified sections is not greater in and out 10% although there is difference depending on the property of shared library.

We get an idea from what the ratio of relocated section is very small. Our main idea is to record a relocated data to a new binary file instead of directly modifying shared libraries at prelinking time (Figure 5). So we design a new cache file, named *dynamic-prelink cache file*, to record relocated data per program. Dynamic-prelink can bring two benefits, compared with prelink. First, dynamic-prelink can create independent and random address layout per program although several programs are prelinked. Second, programs are not adopted by dynamic-prelink can be loaded normally using ASLR because dynamic-prelink do not modify shared libraries.

To support this idea, additional function is needed in the dynamic linker. So we design a new dynamic linker, called *ldp.so*, has the same function with original dynamic linker as well as new function supporting our idea. *ldp.so* is able to decode dynamic-prelink cache file and copy relocated data into the memory during the program startup. Moreover, *ldp.so* can distinguish dynamic-prelinked programs using dynamic-prelink cache file and determine the method for program loading (dynamic-prelink or ASLR) in the runtime.

### 4. Implementation

To verify the feasibility of dynamic-prelinking mechanism, we implemented dynamic-prelink and a new dynamic linker in the ubuntu 12.04. We contribute that dynamic-prelink independently creates the memory address layout of each prelinked program. It supports that non-prelinked program can be loaded using ASLR. Moreover, the new dynamic linker is developed to support dynamic-prelinking

mechanism. We start with a discussion about implementation for the dynamic-prelinking mechanism and the new dynamic linker.

## 4.1 Dynamic-prelinking

Dynamic-prelink has two important points about implementation. First, dynamic-prelink cache file is created to help dynamic linker load the program fast. Dynamic-prelink collects and records a prelinked data, called a *cached data*, to dynamic-prelink cache file when it executes prelinking. `ldp.so` copies the cached data to proper location using dynamic-prelink cache file during starting the program. Second, dynamic-prelink randomly makes the memory address layout per program. It is possible that dynamic-prelink does not modify shared libraries directly. So the base address of shared libraries is assigned randomly, and the memory address layout of each dynamic-prelinked program becomes unique on the system.

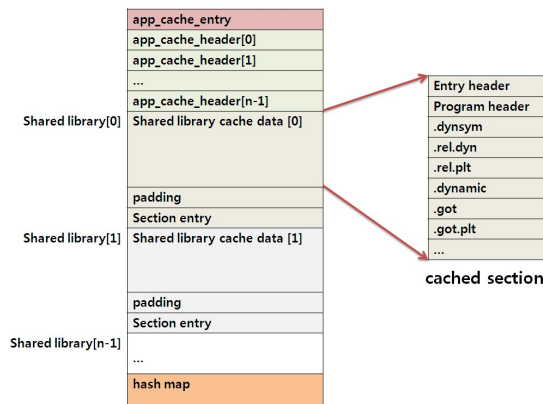


Fig. 6: The format of *dynamic-prelink cache file*.

Table 2: The list of *cached sections*

Section name	Type
.dynsym	SHT_DYNSYM
.rel.dyn	SHT_REL
.rel.plt	SHT_REL
.dynamic	SHT_DYNAMIC
.got	SHT_PROGBITS
.got.plt	SHT_PROGBITS
.data	SHT_PROGBITS
.data.rel.ro	SHT_PROGBITS
.__libc_thread_subfreeres	SHT_PROGBITS
.__libc_atexit	SHT_PROGBITS
.__libc_subfreeres	SHT_PROGBITS
.init_array	SHT_INIT_ARRAY
.fini_array	SHT_FINI_ARRAY
.tdata	SHT_PROGBITS

### 4.1.1 Dynamic-prelink cache file format

dynamic-prelink cache file is a binary which is recorded by the cached data of the location-sensitive section and the relocation section. This file is designed to manage the list of dependent shared libraries of the program and the cached data of each shared libraries. Figure 6 shows how dynamic-prelink cache file format is designed. It is composed to the field of *app\_cache\_entry*, *app\_cache\_header*, *section\_header*, *cached section*, and *hash map*. The characteristic of each field is as follows.

- *app\_cache\_entry*: This structure gives the offset of first index of *app\_cache\_header*, the size of *app\_cache\_header*, and the number of *app\_cache\_header*. It helps dynamic linker find all of index of *app\_cache\_header*.
- *app\_cache\_header*: This structure gives a cached object of each shared library. Each object gives the offset of root header for cached sections and the number of cached sections, those help dynamic linker find all of index of *section\_header*.
- *section\_header*: This structure gives relative offset for the cached data and the copied virtual address. The format of structure is the same with *Elfxx\_Shdr* of ELF.
- *cached section*: This is the set of cached data which is recorded by dynamic-prelink ahead of time. The dynamic linker can directly copy it into the memory instead of relocating operation.
- *hash map*: This element holds the offset of *app\_cache\_header* hash table to search fast object related shared object.

We try to distinguish the type of sections to select the cached data. The list of location-sensitive sections and the list of relocation sections are found by comparing original shared library with prelinked shared binary. Both *vimdiff* and *objdump*, which are utilities, are used to find the list of relocated sections. Table 2 shows the list of location-sensitive sections and relocation sections. Mainly, those sections are recorded to the cached data of dynamic-prelink cache file.

The dynamic linker does not adjust both position independent sections and irrelevant-relocation sections such as *.text*, *.eh\_frame*. But common executable is not built in PIC, is the type of ET\_EXEC. It means that all sections of executable are able to be adjusted by the dynamic linker during starting the program. It is impossible to keep the cached data to dynamic-prelink cache file. But the address of ET\_EXEC's program header is fixed during the link processing at the compile time, the executable is loaded at a known location always. In other words, the executable is originally impossible to be loaded by using address randomization. So dynamic-prelink directly modifies the executable

in dynamic-prelinking time because it does not affect the other programs.

[ 7]	.gnu.version_r	VERNEED	0003cbac	03cbac	000120	00
[ 8]	.rel.dyn	REL	0003cccc	03cccc	0051c8	08
[ 9]	.rel.plt	REL	00041e94	041e94	0014b8	08

[ 7]	.gnu.version_r	VERNEED	43dc3bac	03cbac	000120	00
[ 8]	.rel.dyn	RELA	43dcccc	03cccc	007aac	0c
[ 9]	.rel.plt	REL	43dd4778	044778	0014b8	08

Fig. 7: The size of *.rel.dyn* section is enlarged 1.5 times before and after prelinking due to changing REL to RELA and the relative offset of *rel.plt* is influenced by RELA.

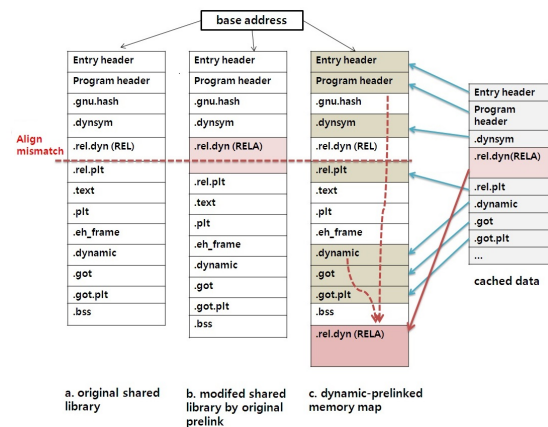


Fig. 8: Comparison with the memory address layout according to the loading mechanism. The location of RELA section is moved at the last address of loaded memory space of each shared libraries.

### 4.1.2 RELA section

we start with assumption that the data segments to be page-aligned and relative offset of shared library are identical before and after prelinking. If there becomes right assumption, the size of sections, which are loaded in the memory, is coincided before and after prelinking. But, in some cases, prelink changes DT\_REL to DT\_RELA which is the property of relocation section. RELA makes relocating operation be easy because it does not need the relocated contents in the memory [1]. *Elfxx\_Rela* contains more a member which is an explicit addend compared with *Elfxx\_Rel*, and the size of DT\_RELA is 1.5 times larger than DT\_REL. Moreover, DT\_RELA is commonly located in an intermediate position of the binary file. Figure 7 shows the size of *.rel.dyn* section is enlarged after prelinking, and the relative offset of *rel.plt* is changed. It is big problem to mechanism of dynamic-prelink due to the page alignment.

But we make one important observation about the relocation section. The relocation section only contains information about which data is relocated in runtime, and it is independent against relative offset in the binary. Namely, it is not important where the relocation section is located. The

dynamic linker commonly gets the location of relocation section from PT\_GNU\_RELRO of program header. A program can be normally operated if the location of relocation section is identical to the address of PT\_GNU\_RELRO regardless relative offset of relocation section. So we design that dynamic-prelink changes the address of PT\_GNU\_RELRO to the last address of loaded binary in case of generating DT\_RELA. Figure 8 compares the section alignment of each loaded shared libraries (original shared library, prelinked shared library, dynamic-prelinked shared library) into the memory. Prelink makes the size of shared library's *.rel.dyn* be enlarged against original shared library. But dynamic-prelink can maintain identical alignment with original shared library by locating the RELA in the last address of mapped memory.

### 4.1.3 Randomization of dynamic-prelink

when dynamic-prelink does prelinking for any program, it is possible to randomly determine the base address of shared library using */dev/random*. Dynamic-prelink sets up various prelinked memory address layout per program. Because an address layout of program becomes unique on the system, the randomization of dynamic-prelink makes it more difficult to analysis the loaded address of shared libraries.

## 4.2 New dynamic linker: ldp.so

The new dynamic linker (ldp.so) supplements additional feature for the dynamic-prelinking mechanism. First, ldp.so is able to decode dynamic-prelink cache file format and copy the cached data to proper address instead of original dynamic linker mechanism. Second, the program, which is not adopted by dynamic-prelink, is loaded normally using ASLR following original dynamic linker mechanism. ldp.so is able to distinguish dynamic-prelinked programs by checking dynamic-prelink cache file. Moreover, in case of adding new rules, ldp.so is able to dynamically determine the loading mechanism (ASLR or dynamic-prelink) in dynamic-prelinked program startup.

```

Program Headers:
Type           Offset  VirtAddr  PhysAddr  FileSiz MemSiz  Flg Align
PHDR          0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
INTERP        0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
               [Requesting program interpreter: /lib/ldp.so]
LOAD          0x000000 0x08048000 0x08048000 0x00bbd 0x00bbd R E 0x1000
LOAD          0x000efc 0x08049efc 0x08049efc 0x0272c 0x0272c RN 0x1000
    
```

Fig. 9: Modified executable's *.interp* by dynamic-prelink.

```

yoon@yoon-x280:~$ sudo cat /proc/14158/maps | grep "ld-2.15.so" & sudo cat /proc/14158/maps | grep "ldp.so"
[9] 14174
00000000-80022000 r-xp 00000000 00:07 4064129 /lib/ldp.so /lib/ldp.so
80022000-80023000 r--p 00021000 00:07 4064129 /lib/ldp.so /lib/ldp.so
80023000-80024000 rw-p 00022000 00:07 4064129 /lib/ldp.so /lib/ldp.so
[9] Exit 3
yoon@yoon-x280:~$ sudo cat /proc/14103/maps | grep "ld-2.15.so" & sudo cat /proc/14103/maps | grep "ldp.so"
[9] 14181
b7732000-b7750000 r-xp 00000000 00:07 4063356 /lib/386-linux-gnu/ld-2.15.so
b7754000-b7755000 r--p 00021000 00:07 4063356 /lib/386-linux-gnu/ld-2.15.so
b7755000-b7756000 rw-p 00022000 00:07 4063356 /lib/386-linux-gnu/ld-2.15.so
    
```

Fig. 10: ldp.so is loaded in running dynamic-prelinked program, but ld.so is loaded in running common program.

When the kernel loads and executes a process in newly constructed address space, the kernel checks the dynamic linker first in executable's *.interp* section. In user-mode, first context of process is started in the entry point of dynamic linker. The dynamic linker is also the shared library (*ld.so*), but it is hard to control the memory address layout of *ld.so* using the dynamic-prelinking mechanism. It is because there is no chance to control the memory address layout of *ld.so* in user-mode. To solve this problem, we make important modification to the dynamic linker and the executable. First, a new dynamic linker (*ldp.so*) is created without modifying *ld.so*, and we induce that the system becomes to own two dynamic linkers (*ld.so* and *ldp.so*). *ldp.so* is directly modified in order to complete the relocation by dynamic-prelink in advance. Second, executable's *.interp* section is modified to make the kernel load *ldp.so* as the dynamic linker. Figure 9 shows executable's *.interp* section is changed to *ldp.so* by dynamic-prelink. After all, programs, which are not adopted by the dynamic-prelinking mechanism, are loaded by original dynamic linker (*ld.so*) and dynamic-prelinked programs are loaded by *ldp.so* as the dynamic linker. Figure 10 shows that two dynamic linkers are able to be loaded according to a mechanism adopted in program.

## 5. Evaluation

We start with a discussion to evaluate our dynamic-prelinking mechanism by comparing the performance of dynamic linker during starting the program and verifying the memory address layout of program which is generated by dynamic-prelink.

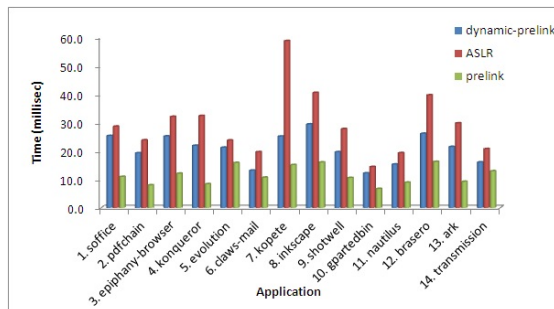


Fig. 11: Spent time before enter executable's *main()* (Figure 1c) in the dynamic linker during 14 applications startup: original prelinked program, dynamic-prelinked program, original program.

### 5.1 Performance

We verified the effect of dynamic-prelink by evaluating spent time in the dynamic linker in Intel i3 dual core 1.2GHz. We also divide the main activity of dynamic linker up into two type of sections: spent time in dynamic linker before calling executable's *main()* (Figure 1c) and spent time in symbol lookups to resolve the GOT during starting

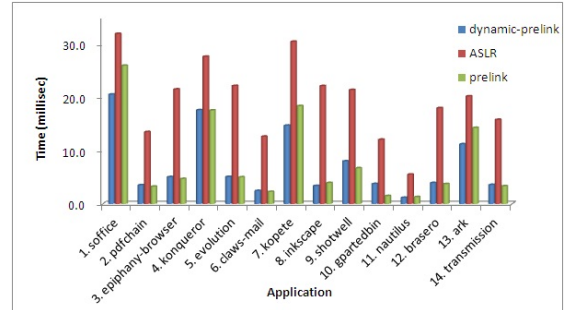


Fig. 12: Spent time to resolve the GOT (Figure 1d) in the dynamic linker during 14 applications startup: original prelinked program, dynamic-prelinked program, original program.

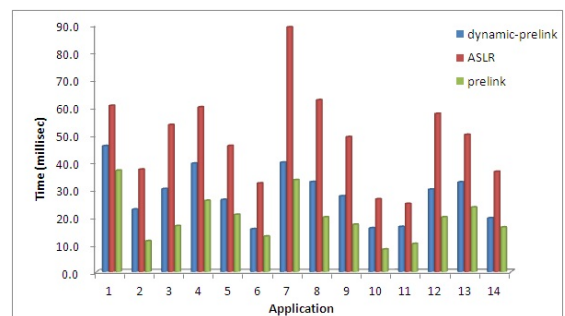


Fig. 13: Whole spent time in the dynamic linker during startup of original prelinked program, dynamic-prelinked program, and original program. This data is the sum of spent time before enter executable's *main()* (Figure 1c) and spent time to resolve the GOT (Figure 1d).

the program (Figure 1d). We compared the performance of original prelink, ASLR (RTLD\_LAZY loading), and dynamic-prelink by adding to dual\_data (We think this is the performance of dynamic linker during starting the program). Figure 13 shows the speed of the dynamic linker using the dynamic-prelinking mechanism is faster than ASLR, about 42% on average, and it is slower than original prelink, about 32% on average. The performance of dynamic-prelink presents intermediate position between ASLR and original prelink. This result is influenced by dynamic-prelink cache file. Dynamic-prelink needs additional time for file I/O and memory copy to control dynamic-prelink cache file. Figure 11 shows spent time in the dynamic linker to load dynamic-prelinked program (Figure 1c) is increased more than original prelinked program. But we should concentrate on saved time in comparison with ASLR. Dynamic-prelink can save relocating time which is similar to original prelink. So the speed of dynamic-prelink is enhanced in both spent time before calling executable's *main()* (Figure 11) and spent time to resolve the GOT (Figure 12) in the dynamic linker. Especially, spent time to resolve the GOT of dynamic-prelink is similar to these of original prelink. (Although a program

is prelinked, dynamic linker should resolve the GOT due to `dlopen()`). Namely, the performance of dynamic-prelink is better than ASLR and is similar to prelink except the overhead of dynamic-prelink cache file.

Table 3: Fixed address layout of shared libraries of two dynamic-prelinked programs. Dynamic-prelinked programs own different address layout each other.

application	library	1st address	2nd address
soffice	ldp.so	0x80000000	0x80000000
	libc.so	0x41967000	0x41967000
	libstdc++.so	0x41fd1000	0x41fd1000
nautilus	ldp.so	0x80000000	0x80000000
	libc.so	0x4b6d5000	0x4b6d5000
	libstdc++.so	0x4be77000	0x4be77000

Table 4: The memory address layout of loaded programs using ASLR in running programs which are adopted by dynamic-prelinking mechanism.

application	library	1st address	2nd address
chrome	ld.so	0xb5efe000	0xb5f3a000
	libc.so	0xaf9f2000	0xafa2e000
	libstdc++.so	0xafbe6000	0xafc22000
firefox	ld.so	0xb7726000	0xb770c000
	libc.so	0xb740a000	0xb73f0000
	libstdc++.so	0xb7619000	0xb75ff000

## 5.2 Memory address space layout

We verified the memory address space layout of processes based on a dynamic-prelinked programs and non-dynamic-prelinked programs using `/proc/[pid]/maps` to prove that two mechanisms (ASLR and dynamic-prelink) are able to be used on a system at the same time (Table 3, 4). And we checked to hold different memory address layout per program which is adopted by the dynamic-prelinking mechanism (Table 3). Table 3 shows `libc.so` and `libstdc++.so` are located in different address in running processes based on two dynamic-prelinked programs, and `ldp.so` is loaded instead of `ld.so`. It means that dynamic-prelink separates address layout of two programs. `ldp.so` is always located in the same address due to the effect of dynamic-prelink.

Another important point is that both `libc.so` and `libstdc++.so` of non-dynamic-prelinked programs are loaded using ASLR at same time in running processes based on dynamic-prelinked program (Table 4). As a result, the dynamic-prelinking mechanism does not affect the memory address layout of non-dynamic-prelinked program. Besides, the memory address layout of dynamic-prelinked program is separated with it of the other dynamic-prelinked programs.

## 6. Conclusion

Dynamic-prelink makes the dynamic linker be able to use mechanism of both ASLR and prelink at the same time. Moreover, it helps dynamic-prelinked program be loaded faster than common program in the dynamic linker, about 42% on average. Since dynamic-prelink supports independent prelinking without modifying the shared libraries, a dynamic-prelinked program does not affect the memory address layout of the other programs. And dynamic-prelink prevents exposing process's address space layout to the other programs. In addition, each program is able to be prelinked selectively according to program's importance by combining several security mechanisms to enhance system performance.

In this paper, we do not discuss *position independent executable* (PIE) for dynamic-prelink. Original prelink cannot support PIE prelinking due to modification of binary. But we have a plan to adapt dynamic-prelink to PIE. Because dynamic-prelink does not modify shared libraries, we expect that PIE can be prelinked by dynamic-prelink. In addition, we will solve a issue about the fixed base address of new dynamic linker (`ldp.so`). We anticipate that `ldp.so` is loaded into random address as well as dynamic linkers (`ld.so`, `ldp.so`) can be unified to one component by modifying kernel's `exec()` mechanism.

## References

- [1] Jelinek, Jakob. Prelink. Technical report, Red Hat, Inc., 2004. available at <http://people.redhat.com/jakub/prelink.pdf>, 2003.
- [2] Bojinov, Hristo, et al. "Address space randomization for mobile devices." Proceedings of the fourth ACM conference on Wireless network security. ACM, 2011.
- [3] van Veen, Sander Mathijs. "Concurrent Linking with the GNU Gold Linker." (2013).
- [4] Shacham, Hovav, et al. "On the effectiveness of address-space randomization." Proceedings of the 11th ACM conference on Computer and communications security. ACM, 2004.
- [5] Spengler, Brad. "Pax: The guaranteed end of arbitrary code execution." (2003).
- [6] Loosmore, Sandra, et al. The GNU C libraryreference manual. Free software foundation, 2001.
- [7] Chamberlain, Steve, and Ian Lance Taylor. "Using ld: the GNU Linker." (2003).
- [8] "Apple OS X Mountain Lion Core Technologies Overview". June 2012. Retrieved 3 December 2013.
- [9] Dai Zovi, Dino A. "Apple iOS 4 security evaluation." Black Hat USA (2011).
- [10] "Android Security". Android Developers. Retrieved 9 December 2013.
- [11] Windows, I. S. V. "Software Security Defenses." (2012).
- [12] Payer, Mathias. "Too much PIE is bad for performance." (2012).
- [13] Tool Interface Standards Committee. "Executable and Linkable Format (ELF)." Specification, Unix System Laboratories (2001).
- [14] John Moser. Prelink and address space randomization, 2006. <http://lwn.net/Articles/190139/>.
- [15] De Raadt, Theo. "Exploit mitigation techniques." (2005).
- [16] Xu, Haizhi, and Steve J. Chapin. "Improving address space randomization with a dynamic offset randomization technique." Proceedings of the 2006 ACM symposium on Applied computing. ACM, 2006.

# A Computer Engineering Approach to Detect, Prevent and Manage Diabetic Foot Diseases

<sup>1</sup>Olawale David Jegede, <sup>1</sup>Ken Ferens, <sup>2</sup>Bruce Griffith, <sup>3</sup>Blake Podaima, <sup>1</sup>Ramin Soltanzadeh

<sup>1</sup>Dept. of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada

<sup>2</sup>Avriel International Inc., Winnipeg, MB, Canada

<sup>3</sup>Internet Innovations Center, Winnipeg, MB, Canada

Ken.Ferens@umanitoba.ca

*Abstract—Diabetes is one of the leading causes of death in humans, which results from many complications that include hyperosmolar hyperglycemic state (HHS), diabetic ketoacidosis, kidney failure, heart disease, sensory neuropathy and damage to the eye. In 2011, diabetes accounted for 1.4 million deaths worldwide, making it the 8<sup>th</sup> leading cause of death. There is a need to prevent and manage this disease to avoid further complications in existing patients. This paper proposes a smart shoe system, which employs wireless technology and intelligent interpretation of sensed data to detect, prevent and manage chronic diabetic foot diseases.*

**Keywords—** Wireless Networks, Local Area Network, Wide Area Network, Sensors, Sensory Neuropathy, Diabetes Mellitus, Operating System, Application, Readings, Data.

## 1. Introduction

Diabetes is a disease that leads to too much sugar in the blood. The widespread presence of diabetes mellitus (DM) or simply diabetes cannot be underestimated. In their work “global prevalence of diabetes – estimates for the year 2000 and projections for 2030”, Wild et al [1] estimated the prevalence of diabetes for all age-groups globally to be 2.8% in the year 2000; it was also estimated to rise to about 4.4% by 2030. This rise in prevalence has been largely attributed to corresponding increase in population growth, aging, urbanization, and also increase in the prevalence of obesity and physical inactivity. According to the Danaei et al [2], in the year 2008, it was estimated that about 347 million people globally have diabetes. By 2013, the

International Diabetic Federation (IDF) reported this estimate to have increased to 382 million people with a prevalence of 8.3% [3]. The IDF also reported the following:

- The number of people with type 2 diabetes is on the increase in every country.
- The greatest number of people living with diabetes falls within age bracket of 40 to 59 years.

Unfortunately, 80% of diabetic patients live in low-and middle-income countries. The IDF also projected that the number of people with diabetes will have increased to about 592 million in less than 25 years' time. The human and economic costs of this disease cannot be underestimated. More money was spent in the treatment of diabetes in North America and the Caribbean than any other region of the world. The IDF divided the global region into 7 and reported that an estimated health expenditure of USD 263 billion was spent on diabetes in one of the regions (the North America and Caribbean Region) in 2013. This figure represents 48% of global health expenditure on diabetes. Since almost half of the death due to diabetes occurred in people whose age falls within the working class (age less than 60), it is necessary to invest in scientific approaches that can prevent and manage the disease. In Manitoba, Canada, the disease was declared a major public health issue in 1996 [4]; one in 16 Manitobans was reported to be diabetic in 2006 and the ratio keeps going up. Major factors influencing this disease in Manitoba include the higher rate of overweight and

obesity when compared to other provinces. Another main factor is that Manitoba has the highest concentration of Aboriginal people in Canada [5]. Diabetes has been reported to be prevalent among Aboriginals due to genetic factors, unhealthy lifestyle, socioeconomic problem and consumption of food with high sugar content [6]. The impact of the genetic factor is however not clear as it has been unmasked by the other factors.

In this work we have focused on preventing complications to the foot of diabetic patients. The majority of diabetic patients suffer from a complication known as *sensory neuropathy*. Sensory neuropathy is a type of peripheral neuropathy which is damage to the nerves of the human body that can result to loss of movement, loss of sensation or affect other aspect of health [7]. In particular sensory neuropathy is most common to the foot of diabetic patients; this is called diabetic neuropathy. It may cause numbness to touch or movement, and reduced or loss of sensitivity/feelings to happenings in the foot such as humidity change, temperature change, pressure change or injury. According to Armstrong et al [8], this sensory neuropathy can lead to *foot ulcers* in 15% of patients or *surgical amputation* in 85% of the patients. It has been reported that diabetes accounts for the loss of about 100,000 limbs each year in the United States [9]; foot damage instantly doubles the chances of a diabetic's dying over the next decade when compared to a non-diabetic. The neuropathic diabetic foot wounds are followed by inflammation on the affected part of the foot. Thus, it is important to measure this inflammation. Some characteristics that can distinguish spots of inflammation from non-infected spots include difference in *temperature, pressure and humidity*. In this work, our approach is to develop a smart shoe that can detect wounds that may occur to the foot of diabetic patients; thus, preventing further complications such as foot ulcers, amputations and death. This approach ensures adequate management of diabetes.

The remaining sections of this write-up are organised as follows. Section 2 represents related work on diabetic foot wounds management and

prevention. Section 3 discusses our methodology designed to monitor the conditions of the foot. In Section 4, we have discussed some of the results obtained and the implications. Section 5 concludes the paper and gives future work.

## 2. Related Work

Some computer engineering approaches have been developed to identify signs of injury in diabetic patients at the early stage. Zequera et al [10] identified several works which have measured plantar pressures in order to detect injury in patients. They advocated the quantitative evaluation of repeated measurements of “plantar pressure and postural balance”. They argue that this will help in the evolution of clinical studies for application (app) that can be used in the clinical settings to improve the diagnostic evaluation. They conjecture the following:

- The continuous performance of the Loran platform on the same individual, for three sessions, once a week, did not report any significant variability.
- The changeability of plantar pressure and postural balance measurement between individuals is considerably more than intra-individual changeability.

They established that the Loran Platform ensures the possibility of obtaining repeated measurements of the following variable: “percentage of load for each foot (LLD), body barycenter (BBx and BBy), foot barycenters (Bx and By), and a point of maximum pressure”. These variables are used to evaluate the balance control and plantar pressure distribution in healthy individuals. Kanade et al [11] also demonstrated that balance control is an essential bio-mechanical parameter that can be used to diagnose a diabetic foot. Frykberg et al [12] explored the temperature parameter in determining areas of inflammation on the foot. They introduced a novel personal home care self-assessment device that helps patients in examining the soles of their feet with a combination mirror and liquid crystal temperature



sensitive pads. The device Tempstat™ has been designed to allow patients examine the plantar surface of their foot easily as long as there is no visual impairment. The device allows patient to easily sight signs of cuts, bruises, irritation, swelling or inflammation. The device operates such that color difference between different parts of the foot indicate temperature differences between the parts; an indication of danger. Results show that the device was easy to use and enhanced the ability of patients to see the plantar surface of their feet. In using temperature-based approach, Armstrong [13] developed a simple self-evaluation home-based dermal thermometry which offers a safe and effective way to detect early signs of injury in diabetic patients. Results show that self-evaluation of the skin temperature using the developed device resulted in reduced foot ulceration in high-risk diabetic patients. Lavery et al [14] also evaluated the effectiveness of an at-home infrared temperature monitoring approach as a tool to prevent persons at high risk for diabetes-related foot complications. The reported results suggest that daily self-monitoring of foot temperatures may be an effective tool that can prevent possible foot complications in persons at high-risk of diabetic related foot complications. Another important parameter that can be used to detect inflammation is the *humidity* of the skin. Humidity simply is the quantity of water vapor in the air. The temperature of the body when humidity is taken into account is called the *heat index*. Thus, the body feels much hotter than the actual temperature whenever the relative humidity is high. The vice versa is also true. The Centers for Disease Control and Prevention [15] warns that diabetes makes it difficult for the body to handle high heat and humidity. Hot weather – temperatures of 80°F (27°C) or above, with high temperature – can have adverse effect on a diabetic patient's medication and health. In order to increase the level of accuracy of a measurement device to the sensitivity of a patient's skin, our approach measures all of the three parameters: *temperature*, *humidity*, and *pressure* at a time.

### 3. Computer Engineering Approach

Measuring the temperature, pressure and humidity of the foot can help mitigate against the risk of foot ulceration [8]. Red “hot” spots are indicators of inflammation; they are a warning sign of danger. Typically the “hot” spots are associated with a difference in the measure of any or all of temperature, pressure or humidity at different spots of the foot. The ideal is to have a uniform measure of any of the parameter across the foot. If left untreated a hot spot can lead to further complications. In this work we have identified 8 hotspots that are sufficient to monitor the well-being of the foot. The distribution of the hot spots is:

- 1 hotspot at the *big toe*
- 1 hotspot at the *third toe*
- 3 hotspots across the *ball of the foot*
- 2 hotspots in the *arch*
- 1 hotspot at the *heel*.

We have designed our smart shoe such that sensor readings (data) of the temperature, pressure and humidity are obtained for each hot spot at a set interval of time. Fig. 1 shows the distribution of the hotspots as explained earlier.

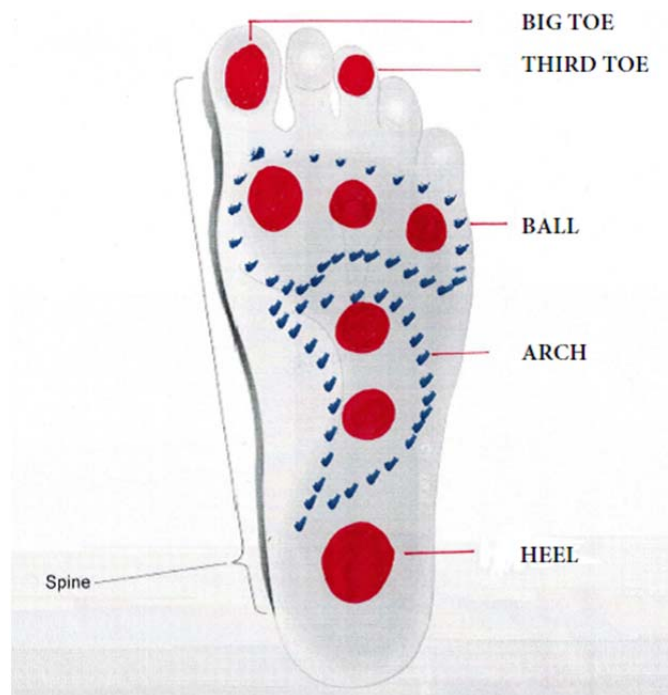


Fig. 1 Foot with hot spots.

### 3.1. System Model

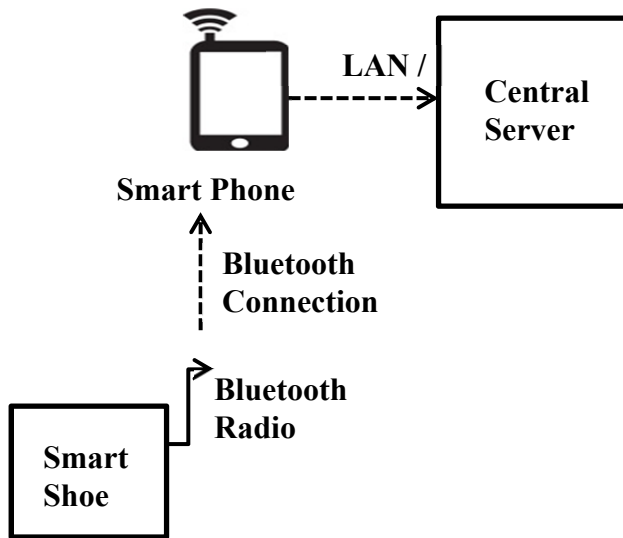


Fig. 2 System Model.

Our system model is shown in 0. The various sub systems are described below.

- **Smart Shoe:** We are developing a smart shoe that can detect danger and mitigate against further complications. We have termed the shoe “smart” because of this functionality. Within the sole of the shoe we have placed an Arduino microcontroller and connected a Bluetooth module and sensors to measure each parameter for each of the hot spots at the same time. The microcontroller is designed to obtain readings every second. The model and type of microcontroller, sensors and Bluetooth used are given in Table 1.

Table 1 Device used.

Device	Functions
ATmega328	Microcontroller
DHT22	Measures humidity and temperature
Strain gauge	Measure pressure
HC-06 serial Bluetooth brick	Enables Bluetooth communication

The DHT22 [16] and the Strain gauge [17] are the two sensors which have been placed at each hot spots. They are connected to the ATmega328 as described in the respective data sheets. We have modified existing Arduino library to obtain readings from the sensors. Since we have two sensors per hot spot, the total number of sensor required for the 8 hot spots is 16.

- **HC – 06 Serial Bluetooth Brick:** Majority of the smart phones today have Bluetooth capability. Bluetooth capability enables wireless connectivity between devices. Smart phones generally support operating systems including iOS, Android, Blackberry and Windows. Each of these operating systems today supports Bluetooth low energy. When compared to other wireless communication standards, the Bluetooth supports a much shorter propagation range (1 – 100m); thus, it consumes less propagation energy. We have chosen to use Bluetooth because a mobile station (smart phone) will typically be within the range of a Bluetooth and consequently save more phone battery power. There are a variety of Bluetooth modules available; we have chosen the HC-06 because it is a plug and play module that supports a wider range of operating system versions [18]. We have connected the HC-06 module to the microcontroller as described in its data sheet. The module allows the sensor readings on the serial port of the controller to be transmitted wirelessly to a connected smart phone.

- **Smart Phone:** A smart phone has more advanced computing capability and connectivity than a basic-feature phone. It has been reported that about 90% of mobile phones in the world run on Android and iOS mobile operating system [19]. A smart phone has the capability of running apps that can be used to monitor events from a wirelessly connected device. In this case, we have paired the smart shoe with an Android-based smart phone through Bluetooth. There is an existing Android app that connects through Bluetooth to an Arduino and displays temperature readings [20]. We have modified this app to display the sensor readings of the three parameters at one hot spot. We have used the MIT App inventor 1 platform [21] to develop the app. The

app was programmed to display the readings every one second. The smart phone is also connected to a Central Server such that the phone sends the obtained sensor readings to the server for processing.

- **Central Server:** This sub-system is designed to receive sensor readings from the phone for the purpose of data storage and interpretation by health personnel. Software will be developed on the server to store the readings and intelligently interpret the implication of the readings.

#### 4. Experimental Results

The sensors were connected to the port of the Arduino board described in the data sheet. The DHT22 (temperature and humidity sensor) and the HC-06 Bluetooth module were connected to the Arduino as given in Table 2. The strain gauge connection to the Arduino also follows the data sheet connection [22] [23]. The symbol “-” in Table 2 means there is no connection between the devices on respective columns. It is important to note that these are the readings for one hot spot. In 0, we have presented the DHT22 sensor readings obtained after every two seconds at the same environment condition.

Table 2 Connection.

Arduino	DHT22 Pin	HC-06 Pin
D0 (Rx)	-	D0
D1 (Tx)	-	D1
D2	2 (Signal)	-
Power 3.3V	-	V
Power 5V	1	-
Ground	4	G

Table 3 DHT22 Readings.

Parameter (Unit)	Value
Humidity (%)	24.0
Temperature (°C)	23.9
Temperature (°F)	75.0
Temperature (°K)	297.0
Dew Point (°C)	2.1
Dew PointFast (°C)	2.1

As expected, we obtained the same reading in 0 as the environmental condition was kept the same. Also, as expected, these readings change when the surrounding temperature and humidity conditions also change. Figure 3 shows the temperature change when the human body comes in contact with the DHT22 over 1 minute. These readings were obtained after every two seconds. Readings show a linear relationship between the temperature reading and time meaning the temperature changes at the same rate with time. The significance of these readings is that the sensor is sensitive enough to ensure that after a certain set value of any of the measured parameter (deemed not to be harmful to the human body) is exceeded, the patient is notified through a generated alarm which makes the patient to check the foot and take necessary action(s).

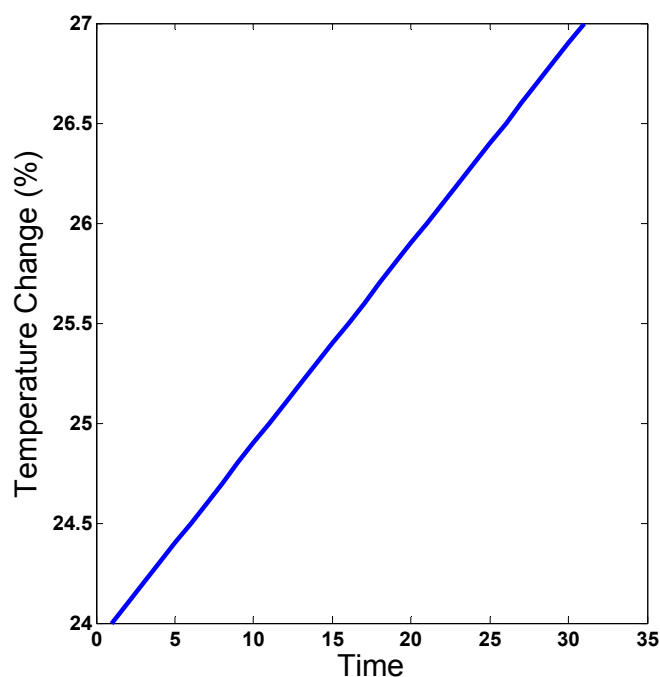


Fig. 3 Temperature readings with time interval.

#### 5. Potential Limitations

The successful usage of the wireless smart shoe depends largely on the technical awareness of the patient using it. For elderly patients who may have problems with their eyesight, finger(s)/hand(s) and other sensory organ, the usage of the smart shoe will be hindered [24]. There is also the possibility of

patients not wanting to use the device. From the economic perspective, the smart shoe may be relatively costly for some patients; government or insurance company's subsidy will go a long way to grant access to many patients. Also, in terms of the shoe's battery lifetime, there is a trade-off between the battery power consumption (lifetime) and the radio range. There are different classes of Bluetooth; classes 1, 2, and 3 with radio range of approximately 100m, 10m and 1m respectively [25]. The higher the radio range covered the higher the demand on battery power; thus, the lower the battery lifespan. The HC-06 (class 2) was used in this work because we assumed a maximum distance of 10m between the patient and the smart phone. The battery lifetime is therefore less than that of a class 1 Bluetooth. Although most Bluetooth applications are in indoor conditions, the radio range may be much lower than ideal due to signal attenuation due to obstructions (walls, etc.) and signal fading due to reflections. Another possible limitation will be the triggering of the sensors when the ambient parameter (e.g. temperature) is high although there may be no danger to the patient's foot. The alarm will mislead the patient into thinking there is a dangerous condition; this might affect the patient's future reaction to an alarm especially when there is a danger. Thus, depending on the environment/climate condition that the smart shoe is to be used, the sensors might need to be calibrated differently.

## 6. Conclusion and Future Work

This work presents a computer engineering approach to combine three important parameters that can be used to measure the condition of the foot of diabetics. Unlike previous works that had been done in this area, combining these three features should ensure more sensitivity to the conditions of the foot; thus helping to prevent possible complications that may result from untimely detection of hazardous conditions to the foot. The sensors reading of the foot temperature, humidity and biomechanical pressure are transmitted to a smart phone through Bluetooth. An app on the smart phone displays these readings at specified time. We are currently working on making use of smaller-sized humidity &

temperature sensor and combining it with the strain gauge on a PCB that is small enough to occupy one hot spot. This will be followed by a design in which we can simultaneously obtain readings from all the hot spots; the smart phone app is equally being developed to graphically display these simultaneous readings. The results obtained from measuring the strain gauge has not been presented here because of not-enough sensitivity. We are currently working on adding an amplifier to increase the sensitivity in order to obtain better readings. The results will be published as part of future work. Also, further work is to transmit the data to a central server over a LAN or WAN. The central server will have software that can store and interpret the data. This is to aid health personnel in equally monitoring and analyzing patients' conditions.

## 7. Acknowledgements

The authors will like to acknowledge the support of Dr. Bob McLeod and Dr. Marcia Friesen of the Internet Innovation center, Department of Electrical and Computer Engineering, University of Manitoba for their advice and support during this research.

## References

- [1] S. Wild, G. Roglic, A. Green, R. Sicree and H. King, "Global Prevalence of Diabetes - Estimates for the year 2000 and projections for 2030," *Diabetes Care*, vol. 27, no. 5, pp. 1047-1053, May 2004.
- [2] G. Danaei, M. M. Finucane, Y. Lu, G. M. Singh, M. J. Cowan, C. J. Paciorek, J. K. Lin, F. Farzadfar, Y.-H. Khang, G. A. Stevens, M. Rao, M. K. Ali, L. M. Riley, C. A. Robinson and M. Ezzati, "National, regional, and global trends in fasting plasma glucose and diabetes prevalence since 1980: systematic analysis of health examination surveys and epidemiological studies with 370 country-years and 2.7 million participants," *Lancet*, pp. 1-10, 2011.
- [3] International Diabetic Federation, *IDF Diabetes Atlas*, 6 ed., 2013.
- [4] Government of Manitoba, "Manitoba Healthy Living," [Online]. Available: <http://www.manitobahealthyliving.ca/are-you-at-risk>. [Accessed 21 04 2014].
- [5] Canadian Diabetes Association, "At the tipping point: Diabetes in Manitoba," [Online]. Available:

- <https://www.diabetes.ca/CDA/media/documents/publications-and-newsletters/advocacy-reports/canada-at-the-tipping-point-manitoba-english.pdf>. [Accessed 21 04 2014].
- [6] L. E. Dyck, "Diabetes and Aboriginal Canadians," Saskatchewan, 2010.
- [7] Wikipedia, "Peripheral Neuropathy," 2014. [Online]. Available: [http://en.wikipedia.org/wiki/Peripheral\\_neuropathy](http://en.wikipedia.org/wiki/Peripheral_neuropathy). [Accessed 21 04 2014].
- [8] D. Armstrong, M. Sangalang, D. Jolley, F. Maben, H. Kimbriel, B. Nixon and I. Cohen, "Cooling the foot to prevent diabetic foot wounds," *Journal of the American Podiatric Medical Association*, vol. 103, pp. 103-108, 2005.
- [9] Orpyx, "SurroSense Rx System," Orpyx, [Online]. Available: <http://orpyx.com/pages/surrosense-rx>. [Accessed 23 04 2014].
- [10] M. Zequera, L. Garavito, W. Sandham, J. C. Bernal, Á. Rodríguez, L. C. Jiménez, A. Hernández, C. Wilches and A. C. Villa, "Diabetic Foot Prevention: Repeatability of the Loran Platform Plantar Pressure and Load Distribution Measurements in Nondiabetic Subjects during Bipedal Standing—A Pilot Study," *Electrical and Computer Engineering*, vol. 2011, pp. 1-14, 2011.
- [11] R. Kanade, R. V. Deursen, K. Harding and P. Price, "Investigation of standing balance in patients with diabetic neuropathy at different stages of foot complications," *Clinical Biomechanics*, vol. 23, no. 9, p. 1183–1191, 2008.
- [12] R. G. Frykberg, A. Tallis and E. Tierney, "Diabetic Foot Self Examination with the Tempstat™ as an Integral Component of a Comprehensive Prevention Program," *The Journal of Diabetic Foot Complications*, vol. 1, no. 1, pp. 13 - 18, 2009.
- [13] D. G. Armstrong, "At-Home Skin Temperature Monitoring Reduces Diabetic Foot Ulceration," *Review of Endocrinology*, pp. 53-55, 2008.
- [14] L. A. LAVERY, K. R. HIGGINS, D. R. LANCTOT, G. P. CONSTANTINIDES, R. G. ZAMORANO, D. G. ARMSTRONG, K. A. ATHANASIOU and C. M. AGRAWAL, "Home Monitoring of Foot Skin Temperatures to Prevent Ulceration," *Diabetes Care*, vol. 27, no. 11, pp. 2642-2647, 2004.
- [15] Centers for Disease Control and Prevention, "Prepare for diabetes care in heat and emergencies," 1 07 2013. [Online]. Available: <http://www.cdc.gov/features/DiabetesHeatTravel/>. [Accessed 22 04 2014].
- [16] Arduino, "Class for DHT11, DHT21 and DHT22," Arduino, 11 02 2014. [Online]. Available: <http://playground.Arduino.cc/Main/DHTLib>. [Accessed 24 04 2014].
- [17] Digi-Key Corporation, "CEA-06-250UW-350," Micro-Measurements (Division of Vishay Precision Group), 2014. [Online]. Available: <http://www.digikey.com/product-search/en?vendor=0&keywords=1033-1013-nd>. [Accessed 24 04 2014].
- [18] Itead Studio, "Electronic Brick of HC-06 Serial Port Bluetooth. Available:," 2013. [Online]. Available: [http://inmotion.pt/documentation/others/INM-0750/DS\\_IM120710006.pdf](http://inmotion.pt/documentation/others/INM-0750/DS_IM120710006.pdf). [Accessed 24 04 2014].
- [19] C. Arthur, "Nokia revenues slide 24% but Lumia sales rise offers hope," 2013.
- [20] Kerimil, "How to control Arduino board using an Android phone and a Bluetooth module," 18 02 2013. [Online]. Available: <http://www.instructables.com/id/How-control-Arduino-board-using-an-Android-phone-a/>. [Accessed 24 04 2014].
- [21] Massachusetts Institute of Technology, "MIT App Inventor," 2014. [Online]. Available: <http://appinventor.mit.edu/explore/>.
- [22] P. Fenner, "Reading Strain Gauge Scales with Arduino," Deferred Procrastination, 2014. [Online]. Available: <https://www.deferredprocrastination.co.uk/blog/2013/reading-strain-gauge-scales-with-Arduino/>. [Accessed 04 24 2014].
- [23] E. E. S. Exchange, "How to wire up a 3-wire load cell/strain gauge and an amplifier?," Electrical Engineering Stack Exchange, 2012. [Online]. Available: <http://electronics.stackexchange.com/questions/18669/how-to-wire-up-a-3-wire-load-cell-strain-gauge-and-an-amplifier>. [Accessed 24 04 2014].
- [24] J. Woodbridge, A. Nahapetian, H. Noshadi, M. Sarrafzadeh and W. Kaiser, "Wireless Health and the Smart Phone Conundrum," in The 2nd Joint Workshop On High Confidence Medical Devices, Software, and Systems (HCMDSS) and Medical Device Plug-and-Play (MD PnP) Interoperability, San Francisco, CA, 2009.
- [25] Wikipedia, "Bluetooth," Wikipedia, 21 May 2014. [Online]. Available: <http://en.wikipedia.org/wiki/Bluetooth>. [Accessed 23 May 2014].

# Criteria-based Research on Fundamentals of Embedded System Development in Higher Education

Steffen Büchner<sup>1</sup> and Sigrid Schubert<sup>1</sup>

<sup>1</sup>Didactics of Informatics and E-Learning, University of Siegen, Siegen, Germany

**Abstract**—*Computer science in general, and embedded system development in specific, is experiencing a fast change in technology paradigms. Educators have to decide which of the current topics last long enough to be fundamental and thus, worth teaching. Based on normative and empirical data, this paper discusses which didactic criteria are needed to identify fundamentals of a discipline. The authors are focusing on embedded system development in higher education. Examples and counterexamples of fundamentals, as well as an analysis of a practical course for embedded system beginners, illustrate the benefits of this procedure.*

## 1. Motivation

Computer engineering is a rapidly evolving discipline with a large body of knowledge. It has a major influence on today's society in terms of technology expectations. This is especially true for embedded system devices in the consumer market (e.g. smart phones or printers). The DFG-funded research project *Competence development with embedded micro- and nanosystems* (KOMINA) [1] introduced the “Empirically Refined Competence Structure Model (ECSM)” to provide educational groundwork for higher education. It is a structured and rated collection of competence descriptions which are linked to embedded system development. The project members have been researching the development of embedded micro- and nanosystems (EMNS) under the presumption of diverse prior knowledge of the students. This helps to improve courses, lectures and practicals. One example is the reconstruction of a hardware-oriented practical at the University of Siegen, which now fosters the best rated competences contained in the ECSM. Based on these results, we investigated the ideas behind the most commonly used techniques, methodologies, and principles. We expect those to be valuable for:

- 1) Future learning sessions, because the nature of the idea is well known and will be applicable to different contexts and situations.
- 2) Establishing relations between different knowledge areas by referring to the underlying principles.
- 3) Helping the graduates to decide which topics and ideas will be relevant for further education. The proposed concept itself can later be used as a tool by the students.

In order to check whether an idea is fundamental to embedded system development, educators need criteria especially aligned to this field of application and the target audience. With respect to the three major curriculum layouts of computer science [2], the body of knowledge of our audience is at least 40 - 50 % computer science-related. The remaining parts are specific to the students' subsidiary subject or/and the field of application (e.g. computer scientists with minors computer engineering, media studies, electrical engineering or medical science).

## 2. Prerequisites and Methodology

The ECSM is the foundation for further research on embedded system education. It was derived from a two-staged research process. First, a set of subject-related competence descriptions were extracted from curriculum recommendations of the ACM/IEEE and the German Research Society. Those competences were clustered into four dimensions, namely:

- competences as preconditions (C1),
- development competences (C2),
- multi-level development competences (C3),
- non-cognitive competences (C4).

Every dimension (C1-C4) contains several sub-categories which themselves are built of related competence descriptions extracted from the previously mentioned documents. For the second stage of the research methodology, KOMINA asked 171 embedded system experts at German universities to rate the importance of every listed competence description. Possible answers were “very important (VI)”, “rather important (RI)”, “rather unimportant (RU)”, and “very unimportant (VU)”. This step was necessary to verify the normative proceeding done earlier. An additional benefit was, that different levels of importance and their correlations were derived. The collection of the best-voted (more than 50% voted for “very important”) competences is shown in Table 1.

The different sources in the normative proceeding lead to heterogeneous competence descriptions. Their level of detail, as well as, their specificity differs greatly. While these descriptions are sufficient enough to investigate into further research, they do not contain guidelines for course creation. Take for example the competence descriptions listed under C2.2 and C3.1 in Table 1. While the term *special constraints*

Table 1: The majority of the survey participants rated this competence as very important [1]

Competences	VI	RI	RU	VU
C1.1 Precondition Competences Mathematics	57,9	38,6	3,5	0
C2 Development Competences C2.1 They learn goal-oriented and structured proceeding for design	56,9	41,4	1,7	0
C2 Development Competences C2.2 They know the special constraints of the design of embedded systems	56,9	29,3	13,8	0
C2 Development Competences C2.3 They gain elementary knowledge and basic comprehension about the most important technologies and about the most important concepts that are needed for design and analysis of computer-assisted systems	61,4	35,1	3,5	0
C2 Development Competences C2.3 They understand the assembly and the function of all important basic circuits and computer units	54,4	36,8	8,7	0
C2 Development Competences C2.4 They can design circuits using a description language	56,9	29,3	12,7	1,7
C3 Multi-level Development Competences C3.1 They learn the acquaintance of the programming language C and its interplay with the hardware. At the same time basic functionality and the interplay of basic components of an operating systems, with the focal point on efficient resource management, are procured	59,7	36,9	3,5	0
C3 Multi-level Development Competences C3.2 They have the ability to understand the relationship between hardware concepts and the impact on the software, to create and apply efficient programs as well as building a computer of basic components	55,2	41,4	1,7	1,7
C4 Non-cognitive Competences C4.2 They can communicate competently in a professional field with engineers and electrical engineering technicians as users of computer-science	56,9	39,7	3,4	0
C4 Non-cognitive Competences C4.3 They have the openness for new ideas/requirements	62,1	32,8	5,2	0
C4 Non-cognitive Competences C4.3 They show willingness to learn	77,6	22,4	3,5	0
C4 Non-cognitive Competences C4.3 They show willingness for commitments	70,7	27,6	1,7	0

is very vague, the relation of the programming language C and its interplay with the hardware is very specific. We searched for a way to unify those competence descriptions. From the perspective of computer science education, a unification of the competence descriptions is very desirable. University teaching staff could choose the most fundamental subjects, concepts and ideas and join them in the course

design.

## 2.1 Determining Fundamentals

Different approaches to find fundamental content have been proposed for more than fifty years. The fundamental ideas are the most popular approach, first mentioned by Bruner [3]. These universal, fundamental, or great principles have been a research subject in e.g. biology[4], mathematics [5] and computer science [6]. Schwill defined the mandatory criteria for a fundamental idea by stating:

- “A fundamental idea with respect to some domain (e.g. a science or a branch) is a schema for thinking, acting, describing or explaining which
- (1) is applicable or observable in multiple ways in different areas (of the domain) (*horizontal criterion*),
  - (2) may be demonstrated and taught on every intellectual level (*vertical criterion*),
  - (3) can be clearly observed in the historical development (of the domain) and will be relevant in the longer term (*criterion of time*),
  - (4) is related to every language and thinking (*criterion of sense*).”[6, p. 7]

He later added the criterion of objective which addresses the, sometimes idealistic and unreachable, goal of an idea [7]. Meyer and Land developed a similar approach named *threshold concepts*, which is used for identifying important learning content, too. They developed the methodology after the observation “*that in certain disciplines there are 'conceptual gateways' or 'portals' that lead to a previously inaccessible, and initially perhaps 'troublesome', way of thinking about something*” [8, p. 1].

One example of a threshold concept is the *pointer*-concept in the C++ programming language. The fundamental idea behind this topic is, however, *call-by-reference*. In our opinion, fundamental ideas seem to be more appropriate for higher education, because they are technique independent and thus provide a better foundation for further learning and more possible combinations with other ideas.

## 2.2 Methodology and Rating

Schwill only examined software engineering. Domains like technical- and theoretical computer science and closely related disciplines like computer engineering have not been investigated. Thus, the fundamental ideas exposed, are not computer science-, but software-engineering specific. Additionally, the selection of ideas has been constructed pragmatically. Schwill explained his understanding of software development in relation to the Software Life Cycle and highlighted important steps which eventually led to terms worth checking (e.g. algorithms and compilers).

Another approach was proposed by Zendler & Spannagel [9]. They set up a questionnaire for experts who had to evaluate the importance of a collection of concepts using

the four criteria shown earlier. The items for this survey were taken from the ACM Computing Classification System (version 1998), resulting in a complete, but ambiguous collection of ideas which contains items like *object*, *information*, *process*, *theory* or *system*. This is problematic, considering their general nature. Deciding whether *system* is a widely applicable concept may be answered differently when thinking about autonomous systems like embedded ones, or operating systems as the abstraction layer between hard- and software.

To summarize, the results of the KOMINA project, as well as the research of Zandler & Spannagel, should be enhanced with more detailed and homogeneous content naming. The ACM and the IEEE updated their classification system recently and now provide additional information about the relation of terms (for instance *used for*, *broader term*, and *narrower term*) which puts the content in a specific context. Other options for deriving terms and subject areas include the ISO Standards catalogue, curricula recommendations, and the ACM Transactions. The *Transactions on Embedded Computing* (TECS) are of particular interest, because of the focus on actual scientific and practical problems of embedded system development. In addition, the publication frequency of TECS allows for a huge collection of terms and ideas to be analysed. From 2007 to 2013, there have been 38 issues on all topics of embedded system development, for instance *Critical Systems*, *Multimedia Systems*, or *Smart Home- and Energy-Efficient systems*. The scope of challenges and interests ranges from requirements gathering, design and implementation, to validation concerns.

### 3. Specific Adaptation of Important Criteria

Schwills goal was the determination of fundamental ideas of computer science. He took the software development process as one popular field of application to draw conclusions on the whole discipline. The research of the authors is concerned with the development of embedded systems which state a more interdisciplinary challenge than traditional software development. Influences of mathematics, physics, and electrical engineering need to be considered. Therefore, the criteria must not only be relevant to computer science, but to adjacent disciplines, too. Former research always targeted school education and thus, an implementation equal or at least similar to the spiral curriculum. While a spiral curriculum can be realized in higher education, too, it is much more difficult because of elective courses, heterogeneous student knowledge and different student minors.

Besides, the goals of higher education are different from those of school education. Whereas Schwills criteria included relations to “everyday language and thinking”, the criteria of our audience should be focused on scientific reasoning and work-life related competences.

Schwills criteria are nonetheless a good start for an amended version, aligned to the target audience and the field of application. The following section names the differences between both research settings and describes how the criteria have to be revised.

#### 3.1 Horizontal Criterion

The initial definition of the horizontal criterion can be applied to higher education for the most part. The only distinction is made on the term “discipline”. Embedded system development is interdisciplinary because it incorporates principles from different fields of research. For instance, Analog-to-digital conversion can be seen as an electrical engineering- or a computer-science-principle. In our opinion, the notion is not important as long as the relation to embedded systems development is apparent. Incorporating adjacent disciplines seems therefore reasonable.

#### 3.2 Advanced Training Criterion

The vertical criterion is removed, because the so-called *spiral curriculum* of general knowledge is not the focus of higher education. Undergraduates and graduates alike should be able to comprehend how an idea works and for what difficulties it might be useful for. Likewise, mediating the learning content to the students should be no matter of the students’ major or minor subject as long as they belong to the target audience (see Section 2). This means, that no specialists’ knowledge is needed to recognize the value of the idea, which implies that the idea has a general nature. In this way the criterion makes sure that the related learning content is a cornerstone for further education.

#### 3.3 Criterion of Time

The criterion of time is taken over without modification. As the curriculum recommendations from the ARTIST Network of Excellence [10] highlight, different disciplines have their own understanding of embedded systems, without necessarily using the same term. Investigations should therefore not only be limited to the history of computer science, but adjacent disciplines, too.

#### 3.4 Criterion of Objective

Zandler & Spannagel argue that the criterion of objective is hardly tangible. In addition, it is already justified if one can name the intention of an idea. Finding counterexamples is difficult because every idea has an intention. In the authors opinion the difference between the two terms *idea* and *concept* is never strict, with or without this criterion. In addition, an idea can only have a relation to work-life or scientific-work if it tries to fulfill an objective, regardless of reaching it or not. We see no advantage in the takeover of this criterion. Quite the reverse, adopting this criterion would overlap with our understanding of the criterion of sense.



### 3.5 Criterion of Sense

Schwill wrote about the criterion of sense: “*From the pedagogical point of view this criterion is closely linked to the Vertical Criterion. Whenever we have to teach a fundamental idea on a low intellectual level, i.e. we have to give students a first vague impression of the idea, we may begin with those situations in everyday life where a fundamental idea becomes apparent.*” [11]. In contrast to school education, undergraduates need to know where ideas are observable and applicable in professional *and* scientific fields of work. The latter is especially important for graduates which might continue their university career by becoming a PhD student. As can be seen in the quoted statement, this criterion is related to the advanced training criterion (former vertical criterion), since depicting the impact of an idea in a practical or scientific context is similar to the introduction of the idea to a group of learners on a low level.

### 3.6 Criterion of Variance

The criteria that have been described till now support a very structured methodology to determine fundamental ideas out of a pool of suggestions. By applying every criterion to every idea, a comprehensive collection of important ideas for future experts of embedded systems can be determined. In a first proof-of-concept, we tested how well the methodology categorizes ideas of current embedded system design research (see section 4). We used the TECS collection described earlier. One observation therein has been, that such an approach bears the risk to have rather similar items with only minor deviations. We therefore propose the criterion of variance, which requires a new facet not contained in any other idea, or the connection of multiple existing ideas in a new way.

## 4. Critique and Examples

While the benefits of research on fundamental ideas have been stated in the last three sections, the concept has its weaknesses, too. Those are:

- **Fundamentality of ideas:** There is no guarantee that the fundamentals examined are those, which all experts agree on. One way to ease this problem is to state the discussion of the idea under observation in the context of every criterion. Thus, the reasoning is comprehensible. In fact, one of the big advantages of the concept is that the argumentation of experts are unified on a methodological level. This does not necessarily mean that all experts agree upon the determined ideas, but that the discussion can take place with a well-defined set of criteria.
- **Completeness of the catalogue:** The list of fundamentals is most likely not complete. Taking an external body of knowledge (like TECS) as a starting point

seems to be reasonable because it provides an exhaustive view on the methods, techniques, and subjects which depict fundamental ideas.

- **Appropriateness of the criteria:** The third argument against fundamental ideas is the presumption that every idea can be justified with the proposed criteria. While it is true that some criteria are rather vague, we will give examples and counterexamples of ideas for every criterion to show their application.

The following examples and counterexamples are based on the Transactions on Embedded Computing (TECS), already mentioned earlier. These calls provide descriptive listings of topics which are currently a research subject. Every method, technique, or issue containing the word *design* or *architecture* has been recorded. Those which did not contain one of those words, but are naturally related to at least one of the keywords were recorded, too (e.g. Models of Computation).

Table 2: Concepts, Techniques, and issues related to design and architecture of embedded systems

Nr.	Items
1	Models of computation and concurrency
2	Hardware/Software Co-Design
3	Platform-based design
4	Component-based design
5	Environment-constraint aware design
6	Co-Simulation and Verification
7	Synchronous and asynchronous design
8	Massively parallel architectures
9	Reconfigurable Computing
10	Run-time systems and middleware
11	Design, verification, and evaluation methodologies
12	Design-Space exploration
13	Estimation of system characteristics in an early stage of development
14	Architecture- and domain specific languages
15	Non-functional constraint aware design
16	Network-On-Chip architecture and design
17	Heterogeneous hardware design with special resources
18	Design methodologies and languages for Real-Time embedded systems
19	Protocols, interfaces and interactions between architectures
20	Model-based design
21	Probabilistic software design and computing architectures
22	Architecture customization techniques
23	Trade-Off analysis
24	Design-time optimization for cyber-physical systems
25	Embedded wireless network architectures
26	Innovative real-time operating system architecture for embedded system

Topics which are too vague have not been included in Table 2. One example for a dismissed item is *tools, infrastructures, and architecture* found in the TECS call on Real-Time, Embedded and Cyber-Physical Systems, 2013. Keep in mind, that all given examples and counterexamples are only discussed with regard to one criterion. To be of fundamental value, all criteria have to be satisfied. A whole list of all possible combinations is beyond the scope of this article.

The horizontal criterion justifies “Models of Computation (MoCs)”. Those define rules on how information is processed inside a model. A graphical representation is commonly used. The key idea behind this concept is to map computational parameters to a descriptive language (the model). This step allows the simulation or analysis of certain tasks by using the model. Areas in which this idea is needed range from organizing the overall packet structure of functional and logical units inside the development process to communication strategies of multiple network-connected entities (e.g. Kahn-Process-Networks or State-Charts).

One counterexample of the horizontal criterion is Model-driven Design. The idea is to use the specification document as the implementation instead of separating requirements gathering (list or text document), design (abstract models) and implementation (source-code) into different documents. There are multiple reasons for such a proceeding, e.g. the closed connectedness between model and implementation, or a “narrower” translation effort between designers and customers by using domain specific languages.

Using a single modelling tool or description language is very difficult because large systems usually consist of several heterogeneous subsystems and conflicting modelling requirements [12]. Taking an even further step to domain specific models is often impractical in practice in terms of reuse of developed artifacts (e.g [13]).

The concept of parallel execution will serve as an example for justifying the advanced training criterion. It is heavily used in soft- and hardware-development. The key idea behind this concept is to separate information dependencies where possible in order to speed up the processing. This can either be in a communication or computational context. It could falsely be taken as an approach to optimization only. Hardware is naturally parallel which requires developers to think about timing and information flow. Escaping the view of sequential instructions, which is typical to computer scientists, is important for further education.

A counterexample of the advanced training criterion is the concept of massively parallel architectures and design. At first glance, the example and counterexample for this criterion seem to be similar, but they are not. While massively parallel architectures make use of parallel processing, the idea behind this concept is not just an intensification or advance of parallelism. First, it is mainly focused on computer organization or computer architecture and therefore only relevant for a subset of students. Secondly, massively parallel architectures include novel attempts for redundancy or power consumption (more on that in [14] and [15]). In addition, students have to be familiar with parallelism before investigating into massively parallel architectures. This topic is therefore too detailed to be a basic for the target audience.

Hardware/Software Co-Design is a concept which is justified by the criterion of time. Twenty years ago, the time-to-market pressure on embedded system development started to increase. In order to establish a system of reusability, programmable processors were used more and more. Because hardware and software are fundamentally different, developers and researchers evaluated design methodologies which eased the difficulties and strengthened the features of both worlds. Instead of a strict separation of hardware and software in the development process, this concept unites them on a functional level. Tasks can be assigned to hardware, software or both, making Trade-Off analysis essential. The idea behind this concept is somehow opposite to the *Separation of Concerns*-principle, recommended for traditional software development projects. It is not necessarily about a “vertical” separation but separation in terms of computation and communication, as well as architecture and application.

Following the success of a System-on-Chips, Networks-on-Chip integrate a network of switches on a chip to widen the communication bottleneck in parallel architectures. One major benefit over traditional communication design is that a developer is not directly concerned with the interconnection of logical units, besides defining them in a high level language like C++ or Java. The synthesizer/compiler will synthesize the whole system to the hardware platform, automatically refining the inter-resource communication. The idea behind the concept is to automatically organize network communication into a separate unit, comparable to processing, memory or I/O units. Hemani et al., introduced this approach in 2000 [16]. While a variety of research papers refined and extended the concept, the practical implementation is still at its beginnings. Therefore, this idea is rejected by the criterion of time.

The criterion of sense is the one which highlights the importance of a subject in terms of practical- and scientific work. The reasoning can therefore be aligned towards industry- or university-requirements, or even both. Co-Simulation and Co-Verification is such a subject which justifies both aspects and therefore satisfies the criterion of sense. A precondition for the approach is to simulate/model hardware. This is problematic, since the mapping of Analog phenomena to digital representations is not always possible. Due to this option, one can use this simulation as an advantage to design hardware and software simultaneously instead of prototyping the software with a huge risk to obtain failures at the later following step of system integration. While researchers investigate in methodologies and tools to extend existing practices, industry relies on Co-Simulation and Co-Verification to cope with time-to-market requirements [17].

Probabilistic embedded computing is rejected by this criterion. While it is a research topic on its own, the practical impact is not widespread yet. This concept aims at the implementation of embedded systems “[..]

using components which are susceptible to perturbations from various sources [..]" [18, p. 1]. The idea behind probabilistic approaches is to dismiss the objective of building reliable and predictable components in favour of methodologies which expect component failures. The overall system, however, is designed to be reliable by using redundant components.

The criterion of variance is not just a mechanism for clustering similar topics, but to differentiate ideas with strong interconnections. Take for example the related subjects Model-based design and Component-based design. They are not equal but have a strong relation to one another. Structuring a system by using components often takes place in model-based design approaches. This does however not imply that every model-based methodology utilizes components. Those two would therefore justify the criterion of variance.

One counterexample can be found in programming languages. For instance, both VHDL and Verilog are Hardware Description Languages with a similar scope of operation. Neither of them would be fundamental because the differences between them are just too small. The idea behind both is the specification of (reconfigurable) hardware by using a description language.

We analysed a laboratory course created on the foundations of the ECSM to check which relations to fundamental ideas have been incorporated.

## 5. Application of criteria on a collection of ideas

Besides the ECSM, the KOMINA project members carried out an open, non-participating observation of a laboratory course [19]. The practical was reconstructed according to the best rated competences in the ECSM. This section will highlight the relations between the practicals' implementation, the ECSM, and the fundamental ideas.

In previous years, all participants (about 60) had to design a processor on basis of an FPGA. The students now have to implement a home-automation with multiple sensors and actuators, using micro-controllers, FPGAs, and smart phones. This allows to mediate competences and ideas regarding compilers, virtual machines, and different programming paradigms. These are important in different branches of embedded system development and, thus, in line with the horizontal criterion.

The observation revealed, that most engineering students had problems in programming. Simple techniques like branches, loops, and variable declaration have been understood in Java, but often falsely applied in C. The participants were not able to implement a comparison of an integer to a register value. Without the reconstruction of the practical, those knowledge gaps would have not been noticed.

As for the advanced training criterion, relations to hands on practices for basic techniques like Ohm's law or bridge circuits can be seen. Overall, the whole practical has been designed for embedded system beginners. Thus, almost all tasks are in line with the reasoning of the advanced training criterion.

Ideas which are historically accepted have been taken over and correlate to the criterion of time. Analyzing a circuit with oscilloscopes and function-generators are techniques which pass the test of time for electrical engineering. Computer science always relied on abstract model descriptions like state machines. These can be used in connection with micro-controller- and FPGA-programming. Those tools are available within the *Virtual Workspace* (see [20]).

In the first run of the practical, the participants implemented an already designed solution for a specific controlling problem. In the second run, the emphasis has been put on the design part itself. This change is in line with demands of professionals which require more graduates knowing the overall system, instead of experts which do not have a comprehension about the relations of all parts, but in-depth knowledge in a specific area (i.e. [10]). Thus, the role of design decisions and requirements analysis were emphasized.

Another observation was surprising too. Most participants had not been able to extract needed information out of data sheets. This included hardware characteristics, interfacing mechanisms, and general information. We believe that these competences are mandatory for further education, but seldom taught. Reading data-sheet and component specification have been important for many years and will be of even more importance in an age of globalization and outsourcing. The need for those competences are in line with the high rating of "(C1.7) scientific work" in the competence structure model (see [1]). The practical course assigns this kind of search to the students to mediate the needed competences, required in industry and science (criterion of sense).

## 6. Conclusion & Further Work

We showed that the results of the KOMINA project are difficult to apply in course design, because of the heterogeneity of the competence descriptions. We therefore propose a refined version of the fundamental ideas which are particularly suitable to face this problem. The illustrated approach unifies important competence descriptions of computer-science- and adjacent disciplines. Several assumptions, like relations to every-day-life and needed profession, do not permit a simple application of Schwill's concept of the fundamental ideas to higher education. Thus, we reinvestigated into every criterion and realigned it to the target audience with the result of introducing two new criteria. These are the advanced training criterion which require ideas to be a foundation for advanced topics in further education and the criterion of variance which excludes topics

with similar ideas. A first proof-of-concept illustrated the criteria's usefulness for evaluating the benefits of learning content in a practical course.

The proposed approach has only been shown superficially. In order to receive a comprehensive catalogue of fundamentals, all embedded system-related ideas of a collection (e.g. TECS) need to be analysed. This is beyond the scope of this paper. After identifying the fundamentals, didactic concepts are needed to foster content and the competences linked to those resulting in a set of embedded system specific teaching recommendations.

## References

- [1] A. Schäfer, R. Brück, S. Büchner, S. Jaschke, S. Schubert, D. Fey, B. Kleinert, and H. Schmidt, "The Empirically Refined Competence Structure Model for Embedded Micro- and Nanosystems", in *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, New York, NY, USA, 2012.
- [2] C. Hochberger, W. Karl, R. Kröger, E. Maehle, P. Marwedel, U. Schmidtmann, and K. Waldschmidt, "Curriculum Technische Informatik in Bachelor- und Masterstudiengängen Informatik", Mar. 2011.
- [3] J. S. Bruner, *The process of education*. Harvard University Press, 1960.
- [4] P. Nurse, "The great ideas of biology", *Clinical medicine*, vol. 3, no. 6, pp. 560–568, 2003.
- [5] F. Schweiger, "Fundamental Ideas, A bridge between mathematics and mathematical education", in *New Mathematics Education Research and Practice*, J. Maass and W. Schlöglmann, Eds., Sense Publishers, 2006.
- [6] A. Schwill *et al.*, "Fundamental ideas of computer science", *Bulletin - European Association for Theoretical Computer Science*, vol. 53, pp. 274–274, 1994.
- [7] A. Schwill, "Philosophical aspects of fundamental ideas: Ideas and concepts", *Lecture Notes in Informatics*, pp. 145–157, 2004.
- [8] J. Meyer and R. Land, "Threshold concepts and troublesome knowledge (2): Epistemological considerations and a conceptual framework for teaching and learning", English, *Higher Education*, vol. 49, no. 3, pp. 373–388, 2005.
- [9] A. Zendler and C. Spannagel, "Empirical foundation of central concepts for computer science education", *Journal on Educational Resources in Computing (JERIC)*, vol. 8, no. 2, p. 6, 2008.
- [10] P. Caspi, A. Sangiovanni-Vincentelli, L. Almeida, A. Benveniste, B. Bouyssounouse, G. Buttazzo, I. Crnkovic, W. Damm, J. Engblom, G. Folher, M. Garcia-Valls, H. Kopetz, Y. Lakhnech, F. Laroussinie, L. Lavagno, G. Lipari, F. Maraninchi, P. Peti, J. de la Puente, N. Scaife, J. Sifakis, R. de Simone, M. Torngren, P. Verissimo, A. J. Wellings, R. Wilhelm, T. Willemsse, and W. Yi, "Guidelines for a graduate curriculum on embedded software and systems", *ACM Trans. Embed. Comput. Syst.*, vol. 4, no. 3, pp. 587–611, Aug. 2005.
- [11] A. Schwill, "Computer science education based on fundamental ideas", *Samways, Brain (Hrsg.): Information Technology—Supporting change through teacher education*. London: Chapman & Hall, pp. 285–291, 1997.
- [12] P. Marwedel, *Embedded Systems Design - Embedded Systems Foundations of Cyber-Physical Systems*, 2nd. Springer, 2011, ISBN 978-94-007-0256-1.
- [13] C. Bunse, H.-G. Gross, and C. Peper, "Embedded System Construction—Evaluation of Model-Driven and Component-Based Development Approaches", in *Models in Software Engineering*, Springer, 2009, pp. 66–77.
- [14] I. Ahmad, "A massively parallel fault-tolerant architecture for time-critical computing.", *The Journal of Supercomputing*, vol. 9, no. 1-2, pp. 135–162, 1995.
- [15] J. Torrellas, "Extreme Scale Computer Architecture: Energy Efficiency from the Ground Up", in *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on*, IEEE, 2013, pp. 1–1.
- [16] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist, "Network on chip: An architecture for billion transistor era", in *Proceeding of the IEEE NorChip Conference*, vol. 31, 2000.
- [17] A. Hoffmann, T. Kogel, and H. Meyr, "A framework for fast hardware-software co-simulation", in *DATE*, Feb. 13, 2006, pp. 760–765.
- [18] K. V. Palem, L. N. Chakrapani, Z. M. Kedem, L. Avinash, and K. K. Muntimadugu, "Sustaining moore's law in embedded computing through probabilistic and approximate design: retrospects and prospects.", in *CASES*, J. Henkel and S. Parameswaran, Eds., ACM, Oct. 26, 2009, pp. 1–10.
- [19] S. Jaschke, S. Büchner, S. Schubert, A. Schäfer, and R. Brück, "Competence Oriented Embedded Systems Course for Computer Science Students", in *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*, ser. WESE '12, Tampere, Finland: ACM, 2013, 6:1–6:7.
- [20] S. Büchner and S. Jaschke, "Preparation for embedded systems laboratories the virtual workspace approach", in *Global Engineering Education Conference (EDUCON), 2013 IEEE*, IEEE, 2013, pp. 171–175.

# Embedded MTT-DAS Application Prototyping on an FPGA based Multiprocessor Architecture

B.Senouci<sup>2</sup>, S.Niar<sup>1</sup>, J.Mitéran<sup>2</sup>, J.Dubois<sup>2</sup>

<sup>1</sup>University of Valenciennes Hainaut-Cambrésis ISTV2- Le Mont Houy, Valenciennes, France

<sup>2</sup>University of Burgundy, LE2I Laboratory, Dijon, France  
ben.senouci@u-bourgogne.fr

**Abstract** —Improving safety in automobile and vehicle industry is one of the biggest preoccupations of embedded systems designers. Driver Assistance System (DAS) is introduced specially to deal with this concern and proposes solutions in order to assist the car's drivers with an efficient warning. In this paper we present an FPGA based multiprocessor architecture for the developed MTT-DAS application, then we introduce the notion of software adaptation that signify the need for an embedded Operating System to fill-in the gap between the MTT software application and the HW multiprocessor architecture. Firstly we show why an embedded operating system is needed for the multiprocessor architecture based MTT-DAS application, and then we study the porting and the debug of this embedded Operating System on the top of multiprocessor FPGA architecture; secondly, we describe the different steps of porting and validating the MTT application on the hardware architecture, and discuss the HW/SW integration step around the embedded operating system layer, and lastly we present some results related to the MTT-DAS SW part and the MTT-DAS HW part.

**Keywords:** MPSoC, Embedded operating system, FPGA based design, Safety, HW/SW interface, Debugging,

## 1. INTRODUCTION

Improving safety in automobile and roads is one of the biggest preoccupations in our every day's life. Vehicles Alert Systems (VAS) is introduced specially to deal with this concern and proposes solution in order to assist the car's drivers with an automatic warning to evaluate quickly a potential dangerous situation. In literature we refer to these systems by DAS (Driver Assistance System).

One of the challenges addressed in this work is to fill in the gap between the Multiple Target Tracking MTT-DAS SW application and the FPGA based multiprocessor architecture.

Contrasting the work presented in [2] [9], where the mapping is done statically based on hand tasks allocation with the use of some HW accelerators, here in this work the mapping of the MTT's tasks is done using an eOS in order to increase the portability of the application, and allows seamless management of the huge amount of the data provided by the radars.

For that purpose we use the FPGA platform based design approach with multiple processors (eg: Xilinx Virtex family, Zynq, SmartFusion MicroSemi)

Unlike traditional MPSoC design/validation based simulation techniques; hardware platform based design [1]

[3] [7] [9] allows a global and early validation in an environment very close to the final implementation.

Recent research project focus on the use of DAS in complex environments using camera-photo embedded in the new car's generation to detect obstacles; however these techniques are not reliable in harsh weather conditions, where the visibility will be tricky, also this existing alert systems present limited functionalities with fixed architecture and put up with expensive cost.

Radar based DAS system that operates even in very hard conditions seems to be a better solution and promise a very good ratio reliability/cost in automobiles safety applications [2].

In our approach, the MTT-DAS system use several radars embedded in a car to get information and signals from obstacles and generates alarms for car driver. We make out two parts in our DAS system; the SW part that corresponds to the application executed by the processors and the reconfigurable hardware multiprocessor architecture that feet the ever increasing computation power of the SW tasks. Especially, we discuss our viewpoint on DAS application and its validation on the top of multiprocessor architecture based on a FPGA fabric. This paper focuses more on how the software adaptation is performed in order to run and validate the application.

In MTT-DAS application, the putted up HW/SW system has as objective to assist and help the driver to avoid accidents. For that purpose, the car is kitted out with several radars (sensors) that detect obstacles and transfer the data to the electronic device (embedded circuit) in order to be computed. Then, a small amount of data provided from the radars can be computed using one computing node (one processor), but once the amount of the data increase; due to the number of the obstacles that can be manifested in the same time; one computing node could not support the ever increasing power computing taking into account the real time constraints. Therefore, multiplying the number of processors and parallelizing the execution for power computing increase seems to be the best option. However it presents several challenges, especially in the control level of the architecture:

- In MTT application a huge amount of data is received from the several radars; then, the control of this data and their mapping on the computing nodes present a real bottleneck,

- When we talk about the application mapping, we will need dynamic control (ex: eOS, embedded Operating System) that manages the tasks execution on the multiprocessor architecture. The choice of this eOS in terms of organization (centralized or distributed), memory footprint, safety etc, will need for a long time investigation.
- Programming multiprocessor architectures at embedded operating system level is not only complicated, but it makes very hard to choose an inter-processor control in order that the performance benefits expected from multiprocessing will not be concealed by the communications overhead.
- Adding an eOS as an adapter layer between the application's task and the hardware architecture need for a multifaceted development, and bring us to a hard and long time debug process of the HW/SW interface.

Furthermore, this DAS case study allowed us to understand the debug difficulties in FPGA based design and validation.

The rest of the paper is organized as follows: section 2 details some related works. Section 3 describes the validation flow steps. Section 4 gives an insight of the MTT-DAS application, while section 5 describes the HW architecture. Section 6 describes the SW one. Results, discussions and analysis are provided in section 7, while section 7 concludes the paper.

## 2. RELATED WORK

Since long time the safety in automobile industry is considered as an obligation in order to provide systems that helps drivers to avoid road's accidents. Many DAS systems are available on the market place and already integrated in cars industry. Parking aid with its ultrasonic sensors or embedded cameras in the new cars generation illustrates existent examples of these systems.

EyeQ2 [5] systems is one example of a single chip dedicated to automotive security applications using vision system, that consists of two 64-bit floating-point RISC 34KMIPS processors for scheduling and controlling the concurrent tasks, five vision computing engines and three vector microcode processors.

This architecture provides support for a specific set of real time data intensive applications. In [6] authors present a dynamically reconfigurable MPSoC (Multiple Processor System on Chip) prototype for AutoVision system; It offers functions such as object edge detection or luminance segmentation, and are implemented as dedicated hardware accelerators to ensure real time processing. Mainly, these DAS based vision systems suffer from their reliability, particularly in harsh environment where the visibility become difficult or even impossible (cloudy weather, rain and snow etc.).

Most of these systems used a pure software control (embedded RTOS) in order to schedule and manage the execution of the different tasks of the system.

Many embedded OSs are available, both in the market place and in the open source community that include safety support. Examples include QNX 4 RTOS, embedded Linux, and eCos etc. The use of eOS is very helpful to enable seamless porting of multithreaded application from one architecture to another.

## 3. MTT-DAS VALIDATION FLOW

Figure 1 shows the validation steps for MTT-DAS application. In this figure, the shaded part represents the design steps that we are interested in for this work. We make out three main steps:

1) Building the multiprocessor architecture on the top of the Virtex 4 FPGA platform, then 2) Porting the MTT application on the target multiprocessor architecture taking into the account the software adaptation layer that map the concurrent MTT tasks on the different computing nodes, and 3) Debugging the HW/SW interface on the top of the FPGA fabric.

In the following paragraphs, we focus more on the shaded part in the design steps. The key focus of this study is the development and the debug of the software adaptation layer for the MTT application.

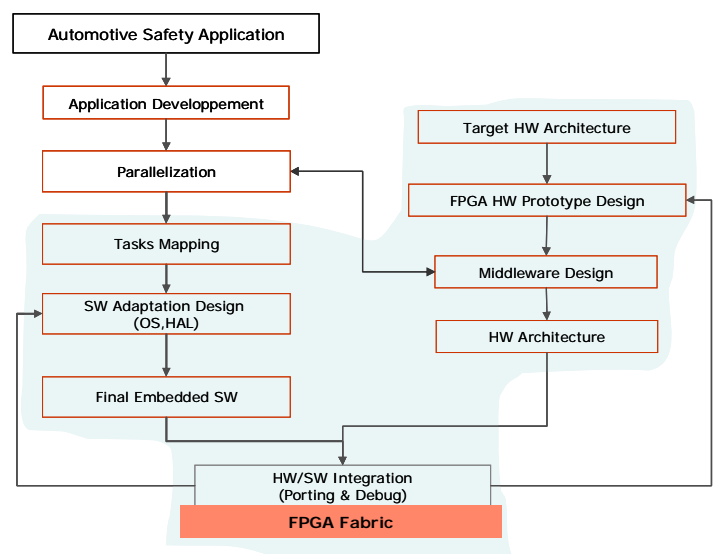


Figure 1: FPGA based HW/SW Validation flow

The gap between the application tasks of the MTT-DAS system and the target architecture needs for a software adaptation. This gap represents the non-portability of the software code on the target architecture. In FPGA based design and validation, on one hand, the target architecture is built using real components (CPUs, Memories, buses), on the other hand the software application is captured at high

level of abstraction usually with a first native validation and a lot of hypothesis on the hardware architecture, then this gap deals exactly with this discontinuity between the two parts of the MPSoC system. The software adaption layer is introduced in order to fill in this gap.

The following sections detail the major steps in the validation flow.

### 3.1 Parallelization and tasks mapping

To target the sequential application SW code on a multiprocessor architecture, the code needs to be distributed, i.e. parallelized using a parallel programming model on the architecture. In this case study, the application has been developed into concurrent tasks communicating with each other via parallel programming primitives.

As parallel programming models suited to multiprocessor architectures, there are two types: shared memory model (e.g. OpenMP [12]) and message passing model (e.g. MPI [10]). In our case study, we used hardware MPI as a parallel programming model. The message passing interface is represented by hardware communication channels implemented in the FPGA and allows the concurrent tasks to exchange data. The implementation of the hardware MPI is detailed in implementation section.

### 3.2 SW Adaptation design

It's one of the major steps in the validation flow. This software adaptation step consists in developing a software layer to feel in the gap between the application SW and the HW architecture [3]. Usually, the application's designer performs a first native validation using simulation models (RTL model) of the HW architecture (CPUs, memories, bus), in that time the application's designer stay far from the target HW architecture by taking some supposition on the booting system, scheduling strategy and the control.

Applications running on multiprocessor architecture require sophisticated software adaptation layer. The notion of the embedded operating system (eOS) is introduced precisely to deal with this control and manage the parts of the MPSoC that have been implemented in software. Embedded OS provides a suitable abstraction allowing easy mapping of application's tasks on the many processor architecture. Usually it's developed in compliance with standard application programmer interface (API-ex: POSIX), at this level it makes this process even more effective and enhances software portability and reuse across different architecture. Depending on the computing nodes (resources) available on the target hardware architecture, two main control organizations are defined.

- **Distributed:** The implementation of the eOS is distributed over multiple processors, each processor has an identical/or specific copy of the eOS image loaded in its nearby memory bank (local memory).
- **Centralized:** A single operating system to ensure all the control software of the multiprocessor architecture

loaded in the main memory (global) and shared by all the CPUs of the architecture.

A centralized control approach is used in this case study, one eOS running on the large core (PowerPC) which manages and distributes tasks on the other processor of the architecture (small cores).

### 3.3 Target HW Architecture

As we target a complicated DAS application where we trait a huge mount of information coming from several targets/radars, a hardware architecture based on multiple processor is needed. For that purpose, we built a heterogeneous multiprocessor architecture using one PowerPC for the control and several Microblaze for computing DAS tasks.

### 3.4 HW/SW debugging

After the adaptation software design, designers need to debug the porting of the software on the hardware architecture. We define the term debug by debugging the software adaptation layer on the FPGA itself and debugging the interaction between the DAS tasks and the added SW layer. In this work, HW/SW debug is done in a cycle accurate way using FPGA prototyping platform, Xilinx Virtex 4 [11], this has JTAG connector. The debugging step starts by setting the debug options in EDK environment. Then, the software part of system including the DAS application tasks, the software adaptation, that compiled and downloaded on each processor of the hardware architecture. At that time, the prototype of the entire HW/SW DAS system is ready to validate. The debugging is supported by breakpoints and source level checking.

## 4. MTT- DRIVER ASSISTANT SYSTEM APPLICATION

Figure 5 shows the graph of DAS-MTT based tasks application (Multiple Target Tracking).

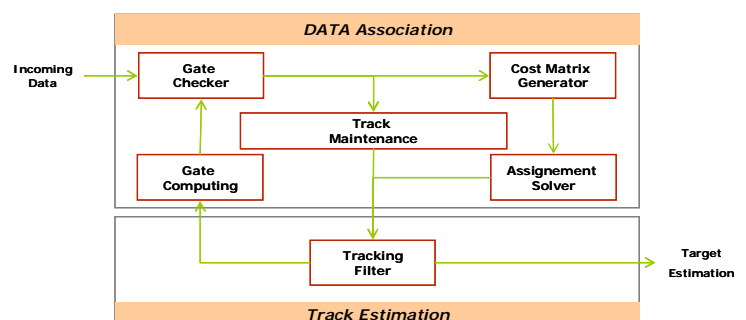


Figure 2: DAS-MTT Task Graph

Mainly, the MTT application developed is represented by five tasks communicate and exchange data with each

other. We designed and validated the DAS application case study following the validation steps of the Figure 1.

Our main goal is to provide vehicle's drivers a system that assists them to avoid the collision with any kind of obstacles during driving or parking time. Since several obstacles could be defined, then the alarm provided by the system must be appropriate for each obstacle in order to predict the driver reaction on his car. Six radars are placed on different side of the car; these radars get observations from the obstacles in form of reflected signals that considered as the input of the DAS task's graph. Several concepts are defined for DAS application:

**Target:** Stand for the *obstacle*

**Track:** Stand for the movement history of the *Target*

#### 4.1 Gate Checker

In overall, the Gate Checker task receives the input signal coming from the target via the radars, and test-out/determined which track correspond to the detected Target. Gate checking determines which observation-to-track pairings are probable. The Gate Checker tests whether an incoming observation fulfils the conditions set by the state prediction and error covariance prediction [2].

#### 4.2 Cost Matrix Generator

A mathematic computation is done by the Cost Matrix Generator task in order to calculate the "Cost" for every possible pair Target-Track. We mean here by "Cost" the numerical distance between a Target and a Track in a full matrix  $M_{ij}$ ; where "i" represent the observation signal of the target, and "j" represents a prediction in the matrix tracks history.

#### 4.3 Assignment Solver

The cost matrix demonstrates a conflict situation where several observations are potential candidates to be associated with a particular prediction and vice versa. To resolve the conflicts, the cost matrix is passed on to the Assignment Solver block which treats it as the assignment problem.

#### 4.4 Tracking Filters

The Tracking filters block has to instantiated as many times as the maximum number of targets to be tracked. In our current work we have fixed this number at 20. We use Kalman filters for this block since it is considered to be the optimum recursive Least Square Estimator (LSE) for Gaussian systems [1] [2]. The Kalman filter continuously cycles through a prediction correction loop. In the prediction step, the filter predicts the next state and error covariance associated with the state prediction. In the correction step, it calculates the filter gain and estimates the current state and the error covariance of this estimation.

### 4.5 Gate Computation

The Gate Computation sub-block of the data association receives state prediction and error covariance prediction from the tracking filters for all the targets. Using these two quantities it defines the probability gates or windows which are used to verify whether an incoming observation can be associated with an existing track. The dimensions of the gates being dictated by the prediction error covariance, these gates demarcate the probability boundaries for the next state coordinate measurements.

### 4.6 Track Maintenance

The Track Maintenance block consists of three functions: the *Observation less Gate Identifier*, the *New Target Identifier* and the *Track Init/Del*. The new target identifier starts a counter for the newly identified target. If the counter reaches 3 in five scans, the target is confirmed and a new track is initiated for it. The counter is reset every five scans. The case of *Obs-less Gate* indicates the disappearance of a target from radar. The *Obs-less Gate Identifier* looks for 3 consecutive misses in 5 scans to confirm the disappearance of a target. The *Track Init/Del* initiates new tracks or deletes existing ones when needed.

## 5. FPGA BASED HARDWARE ARCHITECTURE

### 5.1 HW Architecture Overview

Figure 3 gives a simplified block diagram of the environment setting used for our experiments. The reader looking for more details may refer to [8]. The FPGA platform is composed mainly of two PowePC hard-cores processors, one logic module about five million free gates, configurable soft processor (Microblaze), and system buses that is implemented as soft modules (OPB/PLB buses). Moreover, the whole system is connected to a PC through the JTAG port, and serial port to hyper terminal.

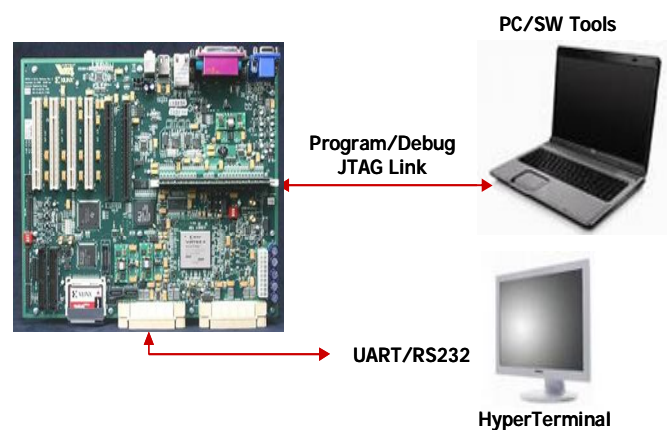


Figure 3: Virtex 4 Platform

Concerning the development environment, it is worth mentioning that we used EDK 10.1 (Embedded



Development Kit) to make up our project. In addition to hardware configuration in the form of bitstreams, we may supply software to initialize and operate the processors core. Information about the software and the object code itself for the Virtex-4/FX PowerPC/Microblaze processor is specified by a (.bmm) and corresponding (.elf) files.

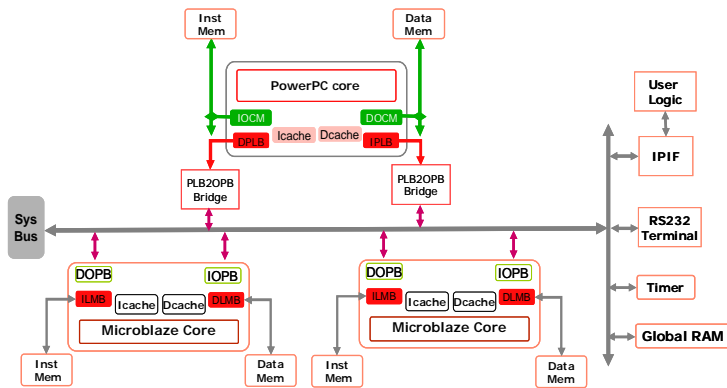


Figure 4: Multiprocessor Architecture

The configuration that we target is based on a large processor core (PowerPC) and two small cores (Microblaze) as shown in Figure 4.

## 6. EOS BASED SOFTWARE ARCHITECTURE

### 6.1 Software architecture

The MTT-DAS application is built on the top of the FreeRTOS, an open source project. In Figure 1, the green links represent the communication channels between the different tasks of the MTT-DAS application.

### 6.2 eOS Overview

- The software architecture is built around a real time embedded operating system (FreeRTOS), an open source project.

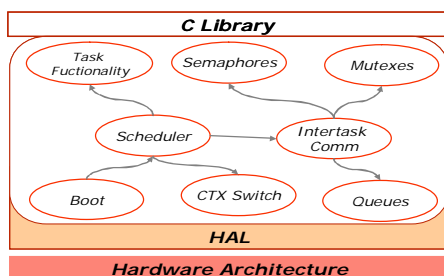


Figure 5: FreeRTOS block diagram

Figure 3 shows its typical architecture. It composed mainly of:

- Cooperative, pre-emptive and hybrid scheduler;

- Multiple inter-tasks communication schemes (queues, semaphores, mutexes) ;
- Tasks and co-routine context;
- Hardware abstraction Layer.

### 6.3 Task synchronization

Synchronization is required whenever a shared data needs to be accessed. In FreeRTOS, this is done using different primitives: Mutexes, semaphores etc

### 6.4 Middleware: Tasks Communication layer

The MTT-DAS application is described as a graph of communicating tasks, more particularly in the form of message passing network. In this formalism, the tasks communicate between them via FIFO channels (the links between the boxes in Figure 2). Our implementation of this communication library previews mainly two different implementation schemes (SW or HW). In this case, SW and HW communication FIFOs are synthesized on top of the FPGA platform and are protected by semaphores.

### 6.5 SW on HW Mapping

The designer maps the parallelized SW code of the MTT-DAS application on the built FPGA based multiprocessor architecture. This includes mapping the software parts (concurrent threads/eOS) on system memory and mapping the concurrent threads on the top of the multiprocessor platform architecture. In FPGA based design approaches, the mapping of the different MTT-DAS tasks on the HW architecture is the key process that associates the function/task to the processing node. In our case (as shown in Figure 6) the abstract architecture model consists of one big PowerPC core and several small Microblaze cores. The SW architecture is build around an eOS/FreeRTOS allowing a static and distributed tasks scheduling policy. The several processor cores (PPC, Microblaze++) a common view of the architecture's peripherals via the system bus (OPB). The execution SW file (.elf) is physically loaded in the local memory of the big core (PowerPC) and then the system boot from their to initialize the eOS and create the main task and scheduler queue of the MTT-DAS tasks. An MTT-DAS function/task can be executed only on the small core that allocated for allowing a static scheduling. In that way, the system can be more efficient, increasing the global architecture performance. As a comparison with the work presented in [9], where the mapping is done statically based on hand tasks allocation with the use of some HW accelerators, here the mapping of the MTT's tasks is done using an eOS in order to increase the portability of the application. And at this time all the MTT's tasks are implemented in software.

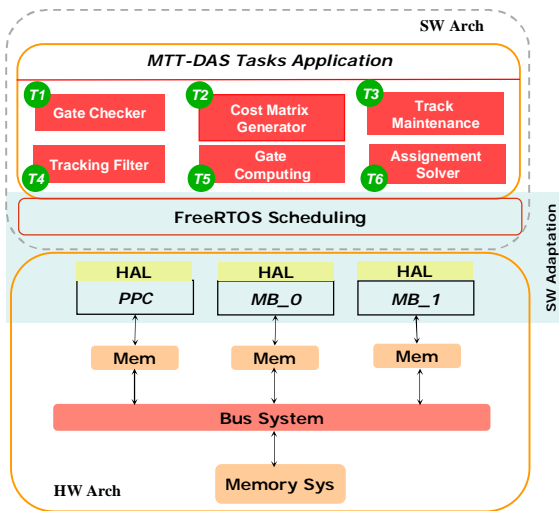


Figure 6: HW/SW MPSoC System for MTT-DAS

## 7. EXPEREMENTATIONS RESULTS

In this centralized distributed configuration, the memory footprint of the SW code (MTT-DAS + eOS) was around 14 Kbytes. This was the result of compiling more than 30 “C” source files using PPC cross compiler, with `-Os` (size optimized) as optimization option. Table 1 shows the code size of software parts of the application in terms of executable code size.

Table 1: Code size of the software part

Code	Size
MTT-DAS	6 KBytes
eOS/FreeRTOS	8 KBytes
Boot-MB	1,2 KBytes

On the HW side, the logical synthesis process is performed using XST tool. It does take about half an hour to completely synthesis the hardware architecture on the Virtex 4 platform. Table 2 shows the synthesis logic results; where the component names refer to the HW modules mentioned in Figure 4. The other columns represent the logic size in term of logic gates/Look-Up-Table, memory banks and FlipFlops.

Table 2: Synthesis results of the hardware part

Component	Slices	4 Input LUT	Slice Flip
PowerPC	79	92	119
Microblaze	755 (2)	1510 (2)	741 (2)
System Bus	122	212	11

## 8. CONCLUSION

In this paper, a Multiple Target Tracking system is presented in order to help the automobile industry to improve their safety. A HW/SW based FPGA architecture is presented for MTT fast computing. The HW architecture is built on the top of a Virtex 4 FPGA, and the SW architecture was built around an embedded RTOS (FreeRTOS) that considered as an adaptation layer for the MTT application. This adaptation layer, when the SW meets the HW, is considered as the key challenges in MPSoC HW/SW interface design. The goal behind using a multiprocessor architecture with an RTOS for the control is to increase the computing power and assure an optimal load balancing, since several obstacles can be detected in the same time, and increase hugely the amount of the data to compute. Finally, using an FPGA based design allows for the validation of the MTT application in an environment very close to final implementation with a speed close to the final MPSoC system. Acknowledgment

## 9. ACKNOWLEDGMENT

The present research work has been supported by International Campus on Safety and Intermodality in Transportation the Nord-Pas-de-Calais Region, the European Community, and the National Center for Scientific Research. The authors gratefully acknowledge the support of these institutions.

## 10. REFERENCES

- [1] Yifan He, Dongrui She, Sander Stuijk, Henk Corporaal "Efficient communication support in predictable heterogeneous MPSoC designs for streaming applications" Journal of Systems Architecture, Volume 59, Issue 10, Part A, November 2013, Pages 878-888
- [2] J.Khan, Smail Niar, Mazen Saghir, Yassin El-Hillali, Atika Rivenq, "Driver assistance system design and its optimization for FPGA based MPSoC," sasp, pp.62-65, 2009 IEEE 7th Symposium on Application Specific Processors, 2009
- [3] B.Senouci, A.Bouchhima, F.Rousseau, F.Pérot, A.Jerraya "Prototyping Multiprocessor System-on-Chip Applications: A Platform-Based Approach" Journal IEEE Distributed Systems Online archive Volume 8 Issue 5, May 2007
- [4] <http://www.freertos.org/>
- [5] G. P. Stein, E. Rushinek, G. Hayun, and A. Shashua, "A computer vision system on a chip: a case study from the automotive domain," IEEE Conference on Computer Vision and Pattern Recognition (CVPR'05), p. 130, June 2005.
- [6] C.Claus, W. Stechele, and A. Herkersdorf, "Autovision – a run-time reconfigurable mpsoC architecture for future driver assistance systems," Information Technology, vol. 49, no. 3, pp. 181–187, 2007
- [7] A.Nejad, A.Molnos, K.Goossens, A unified execution model for multiple computation models of streaming applications on a composable MPSoC, JSA Journal, November 2013, Pages 1032-1046,
- [8] "Xilinx virtex-4 fpga configuration user guide ug360 (v3.2)," November 2010.
- [9] LIU H., NIAR S., Radar Signature in Multiple Target Tracking System for Driver Assistant Application, IEEE-ACM Design Automation & Test Europe DATE'2013 (Grenoble France).
- [10] "Xilinx microblaze processor reference guide, ug081 (v11.0)," Embedded Development Kit EDK 12.1, Sep. 14 2000.
- [11] Usman Ali and Mohammad Bilal Malik. 2010. Hardware/software co-design of a real-time kernel based tracking system. *J. Syst. Archit.* 56, 8 (August 2010), 317-326. DOI=10.1016/j.sysarc.2010.04.008 <http://dx.doi.org/10.1016/j.sysarc.2010.04.008>

# Formal Modeling and Verification of Dynamic Reconfiguration of Autonomous Robotics Systems

Yujian Fu<sup>1</sup>, Mebougna Drabo<sup>2</sup>

<sup>1</sup>Department of Electrical Engineering & Computer Science, Alabama A&M University, Normal AL, U.S.A.

<sup>2</sup>Department of Mechanical Engineering, Alabama A&M University, Normal AL, U.S.A.

**Abstract**—*Dynamic reconfiguration refers to the ability of a system to dynamically change its structure and interface according to different situations. It provides feasible and flexible modeling and simulation environments with powerful modeling capability and the extra flexibility to design and analyze robotics systems. The aim of this work is the modeling and verification of autonomous robotics systems (ARMS) subject to dynamic changes using extensions of Petri nets. It's been one of the research axes of using Petri nets to model reconfigurable systems, where structure changes during runtime, especially in the high level Petri Net domain. Numerous formalisms with different particularities have been proposed. These formalisms try to deal with some aspects of these systems. In this paper, we presented a new, generic and expressive approach to the dynamic reconfiguration on the extension of Predicate Transition Nets, called Predicate Transition Reconfigurable Nets (PrTR Nets). This approach allows the dynamic changes of net structure and provides the flexibility of semantic analysis on the reconfiguration analysis of the net. The formal definition of the PrTR nets formalism will be discussed, and a case study on a humanoid robotic system will be studied. Analysis and verification of the motion scenario will be discussed and related issues will be discussed.*

**Keywords:** Reconfiguration, Humanoid robotics, Predicate transition nets

## 1. Introduction

Dynamic reconfiguration refers to the ability of a system to dynamically change its structure and interface according to different situations. It provides feasible and flexible modeling and simulation environments with powerful modeling capability and the extra flexibility to design and analyze complex systems. In addition, the goals of dynamic behaviors of autonomous robots makes them ideally suited for numerous environments with challenging terrain or unknown surroundings. Applications are apparent in exploration, search and rescue, and even medical settings where several specialized tools are required for a single task. The ability to efficiently reconfigure between the many structures and behaviors of which an autonomous robot is capable is critical to fulfilling this potential. Thus, it is important for making autonomous robotics systems (ARs) adaptable to

changes is one of the main challenge in the autonomous robotics systems.

Moreover, considering the correctness of change and ensuring the appropriate behavior during reconfiguration, it will be more important that the mechanism for change be explicitly represented into the model so that at each stage of product development, designers can experiment the effect of structural changes, by prototypes or verification tools. This means that the structural and behavioral changes are taken into account from the very beginning of the design process rather than handling by an external and global system, e.g. some exception handling mechanism, designed and added to the model describing the system normal behavior. Thus we favour and internal and incremental over an external and uniform description of changes, and a local over a global handling of changes. This approach is compatible with the bottom-up modular synthesis of Petri Nets where a complex system is derived from successive refinements of places or transitions by sub-systems.

The remainder of the paper is organized as follows: Section 2 gives related works of reconfiguration using Petri nets and some other formal methods. Section 3 introduces the formal representation of PrTRN model for autonomous systems. Section 4 describes our approach for constructing and analyzing models of a humanoid robot in the pressing button scenario. Section 5 discusses the result and concludes the paper.

## 2. Related Works

In this section, we will focus on the related works of reconfiguration of the autonomous systems using Petri Nets. Other than that, the reconfiguration specification using other formal methods will be discussed also.

Several research works have been in the specification of reconfiguration using Petri Nets. Valk's self-modifying nets [18], [19] is considered as an early attempt for an extension of Petri net model with a built-in mechanism for handling changes. The changes are captured by two fundamental functions precondition and post condition. The notion of systems of replacement of matrices takes the place of the notion of systems of replacement/addition of vectors that characterize Petri nets.

In the work of [4], [3], [16], a class of high level Petri nets, called reconfigurable nets, is defined. Reconfigurable

nets can dynamically modify their own structure by rewriting some of their net structure that described by rewriting rules associated with the transition. A reconfigurable net is a Petri net with local structural modifying rules performing the replacement of one of its subnets by another subnet. The tokens in a deleted place are transferred to a created one. These nets were used for modeling dynamic changes within workflow systems as in [8]. It was shown that boundedness of a reconfigurable net can be decided by constructing a simplified form of Karp and Miller's coverability tree, however this construction did not allow to decide whether a given place of the net is bounded. However, the reconfiguration only take place in consideration of the structural changes. The rewriting rules that associated with the transition cannot be executed simultaneously with regular transition firing – each transition has two exclusive firing conditions.

On top of the work of [14], [15], to overcome the drawback of reconfigure transition, a Flexible nets was proposed in [13]. Reconfiguration of the structure of the net is interpreted as an operation that manipulates this structure by manipulating its components which are signed objects. The presence of a positive object (resp. negative object) in some place can be a reason to add (resp. delete) this object to (resp. from) the structure of this net. The formalism proposed is called Flexible Nets and reflects the idea that the model has a dynamic structure. This structure can be expanded, shrunken, or destroyed.

In [23], the authors proposed PrN (Predicate/Transition nets) to model mobility. The main concepts of of mobility is introduced as an agent. The agent space is composed of a mobility environment and a set of connector nets that bind mobile agents to mobility environment. Agents are modeled through tokens. So these agents are transferred by transition firing from a mobility environment to another. The structure of the net is not changed and mobility is modeled implicitly through the dynamic of the net.

In [17], authors proposed MSPN (Mobile synchronous Petri net) as formalism to model mobile systems and security aspects. They introduced notions of nets (an entity) and disjoint locations to explicit mobility. A system is composed of set of localities that can contain nets. To explicit mobility, specific transitions (called autonomous) are introduced. Two kinds of autonomous transition were proposed: new and go. Firing a go transition move the net from its locality towards another locality. The destination locality is given through a token in an input place of the go transition. Mobile Petri nets (MPN) [2], [1] extended colored Petri nets to model mobility. MPN is based on  $\lambda$ -calculus and join calculus. Mobility is modeled implicitly, by considering names of places as tokens. A transition can consumes some names (places) and produce other names. The idea is inherited from  $\lambda$ -calculus where names (gates) are exchanged between communicating process. MPN are extended to Dynamic Petri Net (DPN) [2]. In DPN, mobility is modeled explicitly, by adding

subnets when transitions are fired. In their presentation [2], no explicit graphic representation has been exposed.

In [5], authors studied equivalence between the join calculus [9] (a simple version of  $\pi$ -calculus) and different kinds of high level nets. They used "reconfigurable net" concept with a different semantic from the formalism presented in this work. In reconfigurable nets, the structure of the net is not explicitly changed. No places or transitions are added in runtime. The key difference with colored Petri nets is that firing transition can change names of output places. Names of places can figure as weight of output arcs. This formalism is proposed to model nets with fixed components but where connectivity can be changed over time. Label associated is used in the reconfigure transition in "Labeled Reconfigurable Nets" [14], which gives information about mobility.

In nest nets [21], tokens can be Petri nets. This model allows some transition when they are fired to create new nets in the output places. Nest nets can be viewed as hierarchic nets where we have different levels of details. Places can contain nets that their places can also contain other nets. So all nets created when a transition is fired are contained in a place. So the created nets are not in the same level with the first net. This formalism is proposed to adaptive workflow systems.

The power of Petri nets resides in its verification methods. To ensure verification of high level Petri nets, some works were proposed. In [5], author proved the equivalence between Reconfigurable nets and the join calculus. Reconfigurable nets can be interpreted in join calculus and so can be verified. In [6], P/ $\omega$  nets are translated into linear logic programming. Author of [17], encoded Synchronous mobile nets in rewriting logic; they can use Maude to verify specifications. In this paper, we have first way through simulating the net or drawing automatically its reachability tree. The second way that requires more development in future papers, consists of the unfolding of the flexible nets into the Dynamic Nets. These last one can be transformed into CPN.

In this work, we attempt to provide a formal and graphical model for dynamic reconfiguration of robotic systems. We extended Predicate Transition nets with reconfigure guard function that is associated with (reconfigurable) transitions that are enabled when reconfiguration. Change of behavior is modeled explicitly by the possibility of adding or deleting nets which can be single node (transitions or places). Modification can happen when the reconfiguration function is satisfied and tokens available in the reconfigurable place. This allows different type of mobilities of the system behavior changes. The system is described by component based architecture, therefore, the mobility and bindings to resources can be modeled by output from the different components. We have introduce new sorts and operations on the new sorts to specify the mobility and compatibility of system changes. The calculus of system changes and

decision making will be modeled in separate components. The concept of component based model of autonomous behavior is inspired by SAM model [20], [22] that must compute this type of information.

The proposed formal specification approach should include all the regular motions, ensure the correctness of these normal behavior, describe the reconfiguration and ensure the correctness of the reconfiguration by guaranteeing the motion will be continued and finished successfully.

### 3. Modeling Dynamic Reconfiguration of Autonomous Robotics Systems

In this section, we will present the Predicate Transition Reconfigurable Nets (PrTR Nets) and Analysis of the PrTRN nets. PrTRN nets are an extension on the Predicate Transition Nets (PrT nets) with reconfiguration functions.

#### 3.1 Predicate Transition Reconfigurable Nets (PrTRN Nets)

In the PrTR nets, we consider following three aspects of extension – net structure, signature, and net inscription (mainly guard). The dynamic semantics and state system will be discussed later based on the above extension. To allow the changes of place and transition, two special sorts  $Pl$  and  $Tr$  are defined, representing types of place and transition respectively. An instance of the sort  $Pl$  can be a place in a PrTR net, or can be a closed PrTR subnet that starts and ends with places. Similarly, An instance of the sort  $Tr$  can be a transition in a PrTR net, or can be an open PrTR subnet that starts and ends with transitions. A super sort  $PNS$  is defined as compatible with these two sorts  $Pl$  and  $Tr$  to facilitate the net refinement and substitution.

In the traditional PrT nets, the operations on the regular sorts can be regular operations including arithmetic operations, relational and logic operations, and set operations. Considering the new sorts are the set of regulars sorts and new sorts, we define operations of the sort of place and transition as following:

- 1) The arithmetic operations on the new sorts are not allowed.
- 2) The set operations can be the same on the new sorts provided that treats the new sorts as set.
- 3) The substitution is defined on the new sorts  $Pl$ ,  $Tr$  and  $PNS$ .

*Definition 1 (Predicate Transition Reconfigurable Nets):* Precisely, a Predicate Transition Reconfigurable Nets (PrTR Nets) is an extension on the traditional Predicate Transition Nets (PrT Nets) [12], [11] and is defined by a tuple  $(P^R, T^R, F, \Sigma^R, Eq^R, \varphi^R, L, G, G^R, M_0)$ , where:

- 1)  $P^R$  is a finite set of places that includes reconfigurable place,  $T^R$  is a finite set of transitions ( $P^R \cap T^R = \Phi, P^R \cup T^R \neq \Phi$ ), and  $F$  is a set of arcs or flow relations between each pair of  $P$  and  $T$ , e.g.  $F \subseteq$

$(P^R \times T^R) \cup (T^R \times P^R)$ . The tuple  $(P, T, F)$  forms a basic Petri net structure.

- A reconfigurable place is nothing more than a regular place but is used to hold the specific tokens (such as those with sorts  $Pl$ ,  $Tr$  and/or  $PNS$ ). Graphically, a reconfigurable place is denoted by a double line circle.
  - A reconfigurable transition is similar as regular transition but the is enabled and fired by reconfigurable guard  $G^R$ . A reconfigurable transition is graphically defined by a double lined box.
- 2)  $\Sigma = \langle St, Op \rangle$  consists of some sorts ( $St$ ) of constants together with set of operations ( $Op$ ) and relations on the sorts. We define three special sorts  $PNS$ ,  $Pl$  and  $Tr$  represent PrT net system, place and transition. The operation on the regular sorts are still same. In addition, we define two new operations as follows:
    - subsort (denoted by  $\preceq$ ) relation is defined as:  $Pl \preceq PNS$ ,  $Tr \preceq PNS$ , and  $\neg Pl \preceq Tr$  and  $\neg Tr \preceq Pl$ .
    - $\Theta$  will return the sort of any variables, terms and expressions defined on the place, arc and transition, i.e.,  $Op = Op \cup \Theta$  or  $\Theta \in Op$ .
    - $\lambda$  is a merge operation defined on the sorts  $St$  such that
      - $\forall v_1, v_2 \in Var$  (where  $Var$  is the set of variables).  $\lambda$  can be performed iff  $\Theta(v_1) = \Theta(v_2)$ .
      - $\lambda: Var_1, \dots, Var_n \rightarrow \Theta(Var)$  where  $i \in [1..n]$  and  $Var = Var_1 \cup \dots \cup Var_n$ .
    - A dual operation  $\gamma$  is a split operation defined on the sorts  $St$  such that
      - $\forall v \in Var$  (where  $Var$  is the set of variables).  $\gamma$  can be performed iff  $\Theta(v_1)$  returns a multiple set, i.e., the capacity of  $\Theta(v_1)$ ,  $|\Theta(v_1)| > 1$ .
      - $\gamma: Var \rightarrow \Theta(Var_1, \dots, Var_n)$  where  $i \in [1..n]$  and  $Var = Var_1 \cup \dots \cup Var_n$ .
      - $\gamma$  is a reverse operation of  $\lambda$ , and vice versa.
  - 3)  $Eq$  defines the meanings and properties of operations in  $OP$ .
  - 4)  $\varphi^R: P \rightarrow St$  is a relation associated each place  $p$  in  $P$  with a subset of sorts.
  - 5)  $L$  is a labelling function on places, transitions, arcs, and variables. Given a place  $p \in P$  or a transition  $t \in T$ ,  $L(p)$  returns the name of place  $p$ ,  $L(t)$  returns the name of transition  $t$ . Given an arc  $f \in F$ , the labelling function of  $f$ ,  $L(f)$ , is a set of labels associated with the arc  $f$ , which are tuples of constants ( $CONs$ ) and variables ( $X$ ), which is best described by  $L(f, Terms_{\Sigma, X})$ . We use  $Terms_{\Sigma, X}$  represents the expressions on the label of arc  $f$ . We use  $L(Terms_{\Sigma, X})$  to represent  $L(f, Terms_{\Sigma, X})$  when

there is no confusion in context. If  $f \notin F$ ,  $L(f) = \Phi$ .

- 6)  $G$  is a mapping from transitions to a set of inscription formulae. The inscription on transition  $t \in T$ ,  $R(t)$ , is a logical formula built from variables and the constants, operations, and relations in structure  $\Sigma$ ; variables occurring free in a formula have to occur at an adjacent input arc of the transition.
- 7)  $G^R$  is a reconfigurable mapping function defined on the reconfigurable transitions and returns a set of inscription formulae. The reconfigurable function  $G^R$  is boolean function that defined on the evaluation of the variables in the reconfigure place. After evaluating the variable of reconfiguration field, the reconfigurable function  $G^R$  will output the corresponding token which is a subnet that the system is supposed to take.
- 8)  $M_0$  is the initial or current marking with respect to sort, which assigns a multi-set of tokens to each place  $p$  in  $P$  with the same sort,  $M_0 : P \rightarrow MCONs$ .

In the above definition, a reconfigure place is nothing more than a regular place with sorts and holds tokens. The sorts of reconfigure place is defined as a tuple of  $\langle reconfig, PNS \rangle$ , where *reconfig* is a status that is set when the reconfiguration is needed, *PNS* is the PrTR subnet systems that described by the above definition. The subsort relation ( $\preceq$ ) defines an implicit compatible sorts – any token with the sort of *PNS* can be used to substitute the token with the sort of *Pl* or *Tr*. Any subnet of PrTR net  $N$  can have sort, i.e.,  $\varphi(N) = \cup_i(\varphi(p_I) \cup \varphi(p_O))$ , where  $p_I$  is the set of all input places and  $p_O$  is the set of all outgoing places. On top of the above, we define following compatible relation:

- Given any two subnets  $N_1$  and  $N_2$ , we can say the sorts of two subnets  $N_1$  and  $N_2$  are compatible iff the sorts of input places  $p_i^1$  (where  $i \in [1..k]$ ) and  $p_i^1 \in N_1$ ,  $\cup_i St(p_i^1)$  and  $p_j^2$  (where  $j \in [1..m]$ ) and  $p_j^2 \in N_2$ ,  $\cup_j St(p_j^2)$  are same.
- Given any two subnets  $N_1$  and  $N_2$ , we can say the sorts of two subnets  $N_1$  and  $N_2$  are compatible iff the sorts of expressions defined in the guard  $G$  of all interface transitions are same, i.e.,  $G^1(t_i)$  (where  $i \in [1..k]$ ) and  $t_i \in N_1$ ,  $G^2(t_j)$  (where  $j \in [1..m]$ ) and  $t_j \in N_2$ , are same.

### 3.2 Dynamic Semantics

The dynamic semantics of PrTR nets are defined by two conditions – enabled and firing of a transition. In order to define the enabled and firing of a transition, we first give the precondition and postcondition of a transition in a PrTR net.

Let  $L(f, Terms_{\Sigma, X})$  be label expression that associates with arc  $f = (p, t) \in F \vee f = (t, p) \in F$ . For any place  $p \in P$ , we can define the two functions – precond for the token consuming ( $precond(L((p, t), Terms_{\Sigma, X}))$ ) and and

postcond for token producing ( $postcond(L(f, Terms_{\Sigma, X}))$ ) for a transition  $t$  as follows.

*Definition 2 (Precondition of t):* Precondition of a transition  $t$ , denoted by  $precond(t)$  or  $\cdot t$ , is defined as the set of all input places (including the reconfigure place) of the transition that hold tokens, the sorts of the tokens are compatible with the expressions in the guard of the transition  $G(t)$  ( or  $G^R(t)$ ) so that  $G(t)$  ( or  $G^R(t)$ ) is true.

$\forall p \in P.precond(L((p, t), Terms_{\Sigma, X})) : p \rightarrow L((p, t), Terms_{\Sigma, X}) : \alpha$  where  $L(f, Terms_{\Sigma, X}) : \alpha \in \varphi(p)$ ,  $(p, t) \in F$ , and  $M'(p) = M(p) - precond(L((p, t), Terms_{\Sigma, X}))$ ,

*Definition 3 (Postcondition of t):* Postcondition of a transition  $t$  is defined as producing the tokens to the output places of the transition with the compatible sorts of the output places, e.g.,

$\forall q \in P.postcond(L(f, Terms_{\Sigma, X})) : p \rightarrow L((p, t), Terms_{\Sigma, X}) : \alpha$  where  $L(f, Terms_{\Sigma, X}) : \alpha \in \varphi(p)$ ,  $(p, t) \in F$ , and  $M'(p) = M(p) \cup postcond(L((p, t), Terms_{\Sigma, X}))$ .

From above two functions of precondition and postcondition, we can see that for any transition  $t$ , the tokens consumed in the incoming places of the transition  $t$  can be described by a substitution of label expression in the function  $precond(L(f, Terms_{\Sigma, X}))$ , if the substitution of label expression in the token set of preset of transition  $t$ ; while the tokens produced in the postset of the transition  $t$  can be described by a substitution of label expression in the function  $postcond(L(f))$ , if the substitution of label expression satisfy the sort of postset of the transition  $t$ . The token set of preset and postset of transition  $t$  can be described by the substitution of sorts of preset and postset, i.e.,  $pre(t)(\varphi(p) : \alpha)$  and  $post(t)(\varphi(p) : \alpha)$ . Therefore, we have enabling and firing conditions as follows:

*Definition 4 (Enabled t):* A transition  $t$ , denoted by  $enabled(t)$ , is enabled if either the guard  $G(t)$  that is true or the reconfigure function  $G^R(t)$  is true.

$precond(L(f, Terms_{\Sigma, X})) \in pre(t)(\varphi(p) : \alpha)$

In Fig. 1, there are two transitions and three places with the initial markings ( $M_0$ ) in places  $p_1$  and  $p_3$ . The net structure is shown in the left box, while the net inscription is shown in the right box. The guard of transition  $t$   $G(t)$  is not true since the two transitions  $t$  and  $t_1$  are mutual exclusively enabled. Thus the example of Fig. 1 is non-deterministic reconfiguration. When the transition  $t_1$  is enabled if the guard  $G(t)$  and reconfiguration function  $G^R(t)$  are true under the substitution of tokens in the places  $p_1$  and  $p_3$ .

If transition  $t_1$  is enabled, required tokens specified by the label expression of input arcs of the transition must be available in the preset of the transition. If the transition  $t_1$  is fired, those required tokens are consumed and produce some tokens that satisfy the label expression of output arcs of the transition. Both consumed tokens and produced tokens must have the same sort of incoming places of the transition and

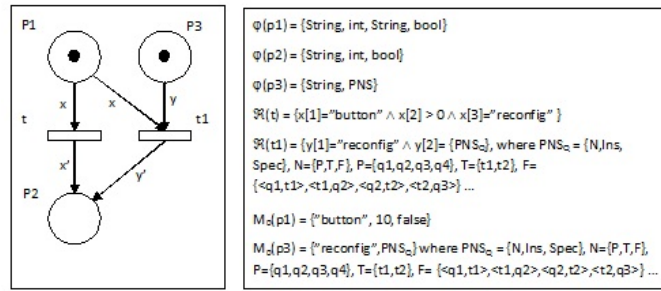


Fig. 1: Example of Reconfiguration (before reconfiguration t is enabled)

outgoing places of the transition respectively. In Fig. 1, by the initial markings in the places  $p_1$  and  $p_3$ , we can say that the transition  $t_1$  is enabled. Similarly the transition  $t$  is also enabled. It is worth to note that they cannot be fired at the same time.

*Definition 5 (Firing of t):* A transition  $t$  can be fired if

- the transition  $t$  is enabled  $enabled(t)$ , i.e.,  $G(t)$  or  $G^R(t)$  is true and
- the tokens in the input places are consumed and there are tokens produced to the output places.  
 $postcond(L(f, Terms_{\Sigma, X})) \in post(t)(\varphi(p) : \alpha)$
- in the case of reconfiguration, in addition to the tokens consumed in the regular input places, the token in the reconfigure place will be consumed and a subnet with the sort of  $PNS$  will be output. In this case,  $G(t)$  will be false and  $G^R(t)$  is true even  $enabled(t)$  and  $enabled(t^R)$  can be true. In other words  $t$  and  $t^R$  cannot be fired simultaneously.

Allowing both  $t$  and  $t^R$  to be enabled can cause the non-deterministic firing of the two transitions. In reality, the truth is  $t$  won't be able to enabled due to the disfunction of normal behavior. This can be controlled by the motion planner or supervisory controller.

In Fig. 2, it shows after the firing of transition  $t_1$ , the net is reconfigured and the markings is updated. The updated net is specified in the reconfigure place  $p_3$ . When the reconfiguration happen, the status should indicate that the system needs to reconfigure which is specified in the last field of reconfigure place. All the three fields make the reconfiguration has higher priority, the reconfigure function will ensure the condition is true so that the system can update to the new topology.

## 4. A PrTR Net Representation of Pressing Button Humanoid Robot

The component based architecture of a humanoid robot system with hands and legs is specified in Fig. 3. In

Fig. 3, four components are specified as *MotionPlanner*, *HandMotionComponent*, *LegMotionComponent*, and *TransEnvironment*. Each component block is defined by following the above component definition. Each component captures a set of operations or functions of subunit or subsystems with required ports definition. For instance, component of *MotionPlanner* will accept all sensing data, coordinate the tasks and sends out the commands to other components. The component *HandMotion* will conduct the hand motion including moving forward or backward in specified angle and direction. Similarly for the component of *LegMotion*. Once the command is sent out from *MotionPlanner*, the command will take all the parameters that are needed by *HandMotion* and *LegMotion* component. This will ensure these components can finish the task continuously without frequently often communication to *MotionPlanner*. The component *TransEnvironment* is an alia component for the environment that sent out from *MotionPlanner*. This is just for representation purpose. By using the component *TransEnvironment* some environment behavior is simply captured and the main behavior of the hand motion of the humanoid robot can be clearly represented. The component *HandMotion* can be split into two subcomponents *LeftHandMotion* and *RightHandMotion*. Similarly for the component of *LegMotion*. In the current version, we only focus on the reconfiguration of hand motion.

It is clearly to note that the behavior of each component is represented by a PrTR net. The double circled places are reconfiguration places. The double lined box is reconfiguration transition. Due to space limitation, we only show the presentation of net structure. For the net inscription, you may refer to the report [10].

### 4.1 Verification & Discussion

In our work we used Maude [7] model checker to verify the property specification in PrTRN model. To automatically implement the model checking using Maude, we design an

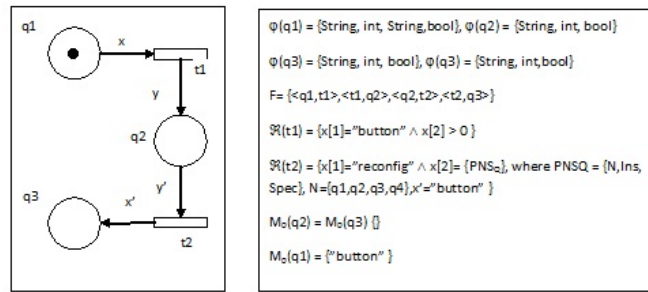


Fig. 2: Example of Reconfiguration (after reconfiguration)

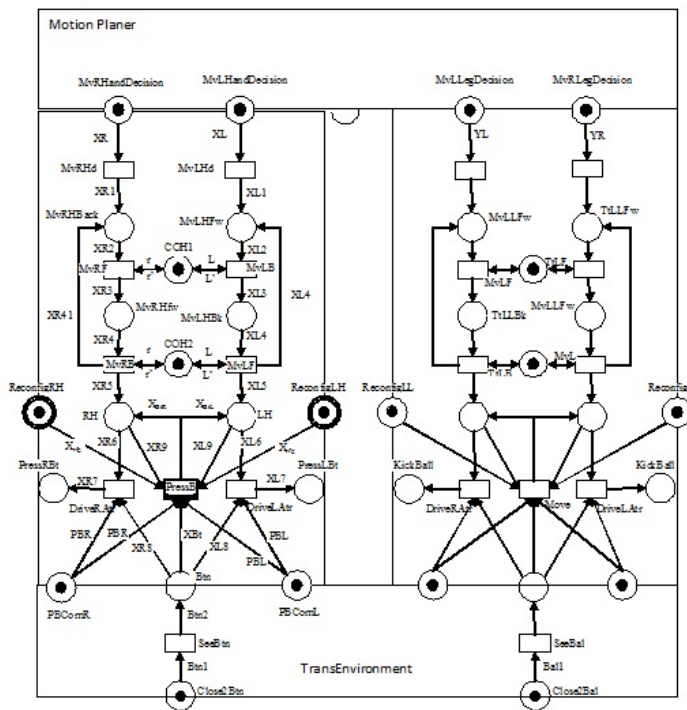


Fig. 3: PrTR Model of Reconfigurable Humanoid Robot

algorithm to translate the PrTRN to the Maude programming language.

All the properties are verified against the model and several problems are identified and fixed. The most important findings using verification is to ensure the system properties and detect errors in the design model. We have specified several properties regarding to the above categories. By running the Maude model checker, several design errors are identified. Thus the model has been fixed and updated for each detected error. Now, it is error free for the current

model. For instance, in the balance category, the properties was false in any initial conditions. The problem is the places (*COH1* and *COH2*) do not hold the behavior of another arm. To fix it, the token will be sent back to these places. However, the properties are still false. We found that this is caused by the one of the initial condition missed in these places. In addition to it, the guard condition of the transition *MvLB* and *MvRF* are updated with one predicate to check the initial case of moving arms.

Reconfiguration category is hard to verified. There are



several problems identified. First, the properties are false because the tokens that were consumed from the place *Btn* in the normal behavior cannot be available during reconfiguration. In other words, the button is not available during reconfiguration, while in reality, the button is available all the time. This is easy to fix by sending the token back to the correct place. Similar problems in the places of *PBComR* and *PBComL*, which indicates the pressing command is still valid.

## 5. Conclusion

Petri nets are an elegant model for concurrency. Combining with graphical representation and its mathematical background, Petri Nets have been widely used to specify and verify concurrent multi-processes systems. In this paper, we have proposed “Predicate Transition Reconfigurable Nets” (PrTRN), a formalism to specify reconfiguration in the autonomous robotics systems with dynamic structure. We have shown the expressiveness of this formalism through the modeling of pressing button scenario of humanoid robot. The use of PrTRN facilitates the tasks of the developer that want to realize formal specification of robotics systems.

The future works include two aspects – transparency and automation. We will develop some transformation rules that help the translation from PrTRN toward regular PrT nets or even Place Transition nets. So that automatic code generation from PrTRN can be realized. To improve the verification at the analyzing level, we need to work on the automatic verification tool that can be used to verify the PrTRN net automatically without the interaction of human beings. Therefore the PrTRN models and the required proprieties will be automatically related and checked.

## Acknowledgment

The authors would like to thank all reviewers for the kindly comments and suggestions on this work.

## References

- [1] A. Asperti and N. Busi. Mobile petri nets. *Mathematical. Structures in Comp. Sci.*, 19(6):1265–1278.
- [2] A. Asperti, N. Busi, P. Porta, and S. Donato. Mobile petri nets. Technical report, 1996.
- [3] E. Badouel, Éc. Nat, S. Polytechnique, and Y. Cameroun. Modeling concurrent systems: Reconfigurable nets. In *In Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'03)*, pages 1568–1574. CSREA Press, 2003.
- [4] E. Badouel and J. Oliver. Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes within Workflow Systems. Research Report RR-3339, INRIA, 1998.
- [5] M. G. Buscemi and V. Sassone. High-level petri nets as type theories in the join calculus. In *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures, FoS-SaCS '01*, pages 104–120, London, UK, UK, 2001. Springer-Verlag.
- [6] I. Cervesato. Petri nets and linear logic: a case study for logic programming. In *In Proc. of GULP-PRODE'95*, pages 313–318. Palladio Press, 1995.
- [7] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2):187–243, 2002.
- [8] C. Ellis, K. Kedara, and G. Rozenberg. Dynamic change within workflow systems. In *Proceedings of conference on Organizational computing systems*, COCS '95, pages 10–21, New York, NY, USA, 1995. ACM.
- [9] C. Fournet and G. Gonthier. The Join Calculus: A Language for Distributed Mobile Programming. *Applied Semantics*, pages 268–332, 2002.
- [10] Y. Fu and S. Drager. Reconfiguration Analysis of Automatic Robotics Systems. Technical report, Air Force Research Lab, August 2012.
- [11] H. J. Genrich. Predicate/Transition Nets. *Lecture Notes in Computer Science*, 254, 1987.
- [12] X. He. A formal definition of hierarchical predicate transition nets. In *Proceedings of the 17th International Conference on Application and Theory of Petri Nets*, pages 212–229, London, UK, 1996. Springer-Verlag.
- [13] L. Kahloul, A. Chaoui, and K. Djouani. Modeling and analysis of reconfigurable systems using flexible petri nets. In *Proceedings of the 2010 4th IEEE International Symposium on Theoretical Aspects of Software Engineering, TASE '10*, pages 107–116, Washington, DC, USA, 2010. IEEE Computer Society.
- [14] K. Laid and C. Allaoua. Code mobility modeling: a temporal labeled reconfigurable nets. In *Proceedings of the 1st international conference on MOBILE Wireless MiddleWARE, Operating Systems, and Applications, MOBILWARE '08*, pages 34:1–34:6, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [15] K. Laid and C. Allaoua. Coloured reconfigurable nets for code mobility modeling. *Int. J. of Computers, Communications & Control*, 2008:358–363, 2008.
- [16] M. Llorens and J. Oliver. Structural and dynamic changes in concurrent systems: Reconfigurable petri nets. *IEEE Trans. Comput.*, 53(9):1147–1158, Sept. 2004.
- [17] F. Rosa-Velardo, O. Marroquín-Alonso, and D. de Frutos-Escrig. Mobile synchronizing petri nets: A choreographic approach for coordination in ubiquitous systems. *Electron. Notes Theor. Comput. Sci.*, 150(1):103–126, Mar. 2006.
- [18] R. Valk. Self-modifying nets, a natural extension of petri nets. In G. Ausiello and C. Böhm, editors, *Automata, Languages and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 464–476. Springer Berlin / Heidelberg, 1978.
- [19] R. Valk. Generalizations of petri nets. In J. Gruska and M. Chytil, editors, *Mathematical Foundations of Computer Science 1981*, volume 118 of *Lecture Notes in Computer Science*, pages 140–155. Springer Berlin / Heidelberg, 1981.
- [20] W. M. P. van der Aalst, K. M. van Hee, and R. A. van der Toorn. Component-based software architectures: A framework based on inheritance of behavior. *Science of Computer Programming*, 42(2-3):129–171, 2002.
- [21] K. M. van Hee, I. A. Lomazova, O. Oanea, A. Serebrenik, N. Sidorova, and M. Voorhoeve. Nested nets for adaptive systems. In *Proceedings of the 27th international conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN'06*, pages 241–260, Berlin, Heidelberg, 2006. Springer-Verlag.
- [22] J. Wang, X. He, and Y. Deng. Introducing Software Architecture Specification and Analysis in SAM through an Example. *Information and Software Technology*, 41(7):451–467, 1999.
- [23] D. Xu and Y. Deng. Modeling mobile agent systems with high level petri nets. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 5, pages 3177–3182 vol.5, 2000.

# A COGNITIVE APPROACH FOR STROKE REHABILITATION BY ACUTE HAND MONITORING

Hema Barathi.R<sup>1</sup>, Ramya Priyadarshini.D<sup>1</sup>, Sowmya.K<sup>1</sup>, Ramalatha M<sup>1</sup>  
Department of Information Technology, Kumaraguru College of Technology,  
Coimbatore, Tamil Nadu, India

**Abstract** - Cognitive rehabilitation for non-traumatic brain injuries focuses purely on retrieving the cognitive abilities of a person after an injury or an illness affects one's brain. Successful rehabilitation for stroke has often resulted from using effective motion sensing. The existing systems include a virtual reality environment along with a high definition camera used for motion capture. The major drawbacks found in these systems are that they measure only wider hand and leg movements and are expensive. We have developed a Cognitive approach for Stroke Rehabilitation by acute hand monitoring, which measures acute displacements of hand that is used in bringing effective cognitive rehabilitation for stroke patients at the rehab stage. Flex sensors are attached to the fingers of the patient to measure even a slightest jitter and corresponding magnified finger movement is displayed which motivates the patient.

**Keywords:** Cognitive rehabilitation, stroke rehabilitation, hand monitoring.

## 1 Introduction

The three inborn skills of every human being namely the sensory, the motor and the cognitive skills are subjected to various experiments every now and then as they could lead to the sprouting of new innovative therapies and rehabilitation processes for both traumatic and non-traumatic brain injuries. According to the official journal of Indian Academy of Neurology, of late, India is silently witnessing the stroke epidemic and serious efforts must be taken to fight against it. The currently existing stroke rehabilitation in India primarily involves the participation of a Neurologist, a Psychiatrist, a Psychiatrist, a Physical Therapist, an Occupational Therapist and a Speech Therapist. The combined contributions of all the above mentioned doctors enable a stroke patient to undergo the rehabilitation process successfully. This conventional rehabilitation process involves more patience and round the clock presence of another person which could be tiring.

According to studies made by The George Institute for Global health, India, 87% of stroke patients in low and middle- income countries such as

India have no access to Western form of stroke rehabilitation as they are really expensive. On summing up all these disadvantages on the existing rehabilitation processes, there is a need for new, economic friendly and easily accessible stroke rehabilitation process and our attempt is focussed on providing such rehabilitation process to the lower economy stroke patients.

## 2 Literature survey

The survey on the existing rehabilitation systems and techniques gave us many fruitful insights which helped us to define our system's objective. The goal of Virtual Reality systems is to put the people with disabilities in control of their own activities with one such instance described by Norman [6]. The pros and cons of virtual rehabilitation have been clearly explained by Grigore [3]. However a major disadvantage would be the attitude of the therapist and inadequate communication between the patient and the therapist. A SWOT analysis conducted by Albert and Gerald [1] shows us that the major strength is the motivation and confidence it creates in the patients. But the acceptance of the patients and doctors is been a major concern. The major contributions of VR have been through the development of games and virtual environment for the patient with few instances such as hop hop frog and bubble pop in [9]. Another development of VR is the use of system from the home with remote monitoring from the clinic [2].

A basic convenient home based virtual reality rehabilitation system would normally consist of a laptop or a computer with suitable user interfaces and joy sticks [8]. Another approach is to obtain postures from the human and create key frame animations from poses captured from a single camera. A very interesting approach was to use music in stroke rehabilitation [5]. Since music combines three cognitive formats namely motor, iconic and verbal; it promotes rehabilitation which reduces anxiety and confusion thus improving motivation. Colour gloves have different colour patches based on which the motion of the hand is being monitored. According to [10] the motion of the gloved hand is tracked by

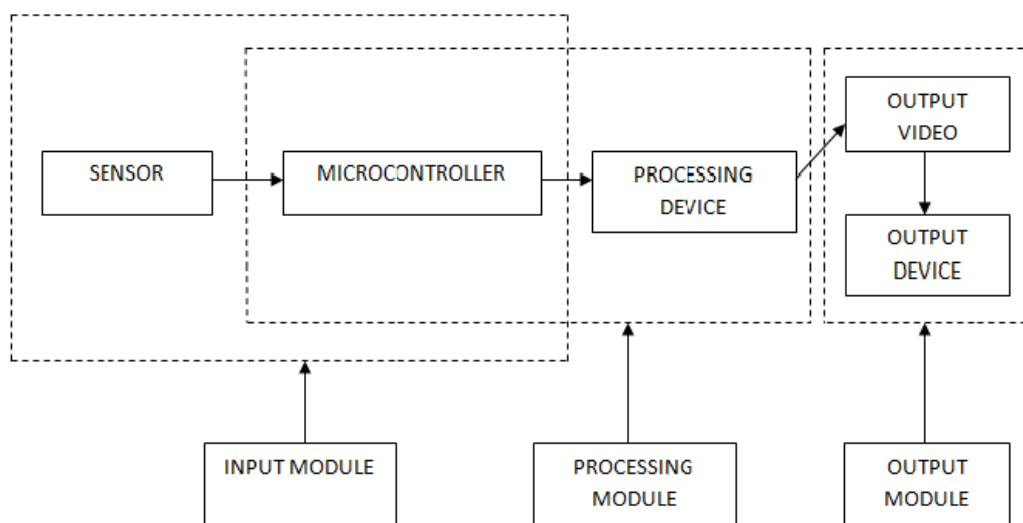
capturing it through camera, which is later, processed in the computer. But a drawback of using this technology is that an exact motion of the hand can never be tracked. Later cost efficient gloves were developed such as one described in [7]. But it had a serious disadvantage of increasing the physical inconvenience to the patients. Another glove developed with the same goal was described in [4] where they used cloth glove with colour markers. Based on the colour of the marker the motion of the hand is detected.



*Figure 1: Colour gloves for stroke patients*

### 3 Inference of the survey

From the survey on virtual rehabilitation, it was obvious that the concept of virtual rehabilitation works effectively to a greater extent unlike its other rehabilitation counterparts. One main inference made from the survey is that the concepts of virtual and mental rehabilitation are technologies that are yet to be utilized by Indian hospitals. These technologies are things of the present at western countries and are proved to be effective. So by implementing virtual rehabilitation concept through a cognitive approach, the treatment for paralysis patients in India can become more effective than those physical rehabilitation practises that are currently in existence. The survey taken on the existing virtual rehabilitation practises lead to several interesting perspectives. One important perspective is that all the research papers that were considered for the survey had a very common drawback. All the systems proposed outwardly restricted the physical freedom enjoyed by the patients. This in turn affects the rehabilitation process to some extent. Hence it was necessary to design a system that effectively enhances the patient's physical freedom. The survey on data and colour gloves contributed to most of the objective of the project. Colour gloves and data gloves are customized equipments which cost material as well as money. Since 5 out of 6 people are attacked with paralysis, it is practically impossible for patients to afford the gloves system for their rehabilitation. Thus through this survey we conclude that we must develop a system that is devoid of costly gloves but should implement the functionalities that are effectively performed by gloves.



*Figure 2: Block diagram of the system*

## 4 Overview of the system

Based on the inferences made from the literature survey, we formed the overall block diagram of the system which has three modules as depicted above. The input module consists of a microprocessor which receives the signal from the sensor that is fixed on the patient's hands. The microprocessor used here is Arduino Uno and the sensor is flex sensor. Arduino Uno was selected for the system because of its simplicity and its nature of being programmer friendly. Its ADC feature and greater compatibility makes it easier to use and program. For the sensor, first we considered the three axis accelerometer ADXL335 but we had a difficulty of placing it on the patient's finger because of its bulkiness. To overcome that particular difficulty and also to achieve our objective of maximum physical freedom to the patient, we chose the flex sensors. Flex sensors are made up of carbon resistors and are available in various lengths. We used the 2 inch flex sensors and the very thin structure of the sensor is apt to place right on the fingers – on the back side of the hand.

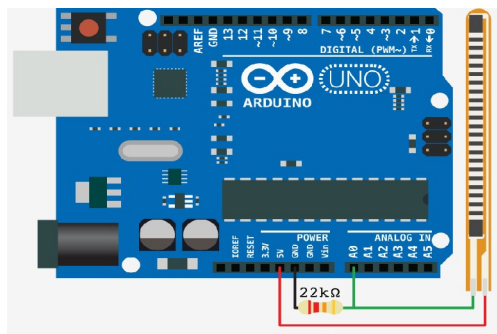


Figure 3: Hardware of the system

The processing part composes of the computer to which the Arduino is attached. The application that takes care of the processing part is the windows form application which is developed in the C# language using the Windows Visual Studio's .NET framework. The form initiates the communication with the arduino with a button press event and once it receives a value from the arduino, it triggers a video file to play.

The output module is primarily a display device which displays the video file that the windows form has retrieved. The display device will be of user's choice say a TV screen, a projector screen or a computer screen.

As any other real time system, our system too has certain basic requirements as given below:

- There should be an attendant/nurse for operating the application.

- The application is compatible with Windows OS only.
- The videos suitable for the patients must be selected for customization of application.
- The patients are subjected to only cognitive rehabilitation.

## 5 Working of the system

The efficiency of the system depends on how well the system copes with the real time input it receives and also how well the communication takes place between the modules. An executable windows form application is installed in the attendee's computer, the person who is in charge of the system.

The Arduino codes runs indefinitely until it gets an input from the flex sensor. Once an impulse is received, the arduino sends the angle of movement received from the flex sensor to the application. A sample screen shot with received angle measure is shown below:

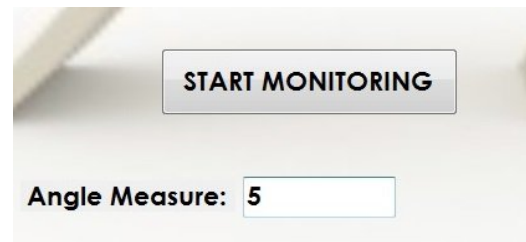


Figure 4: Angle Measure

Once the angle measure is received, the application displays options for customization. The screen shot is given below:

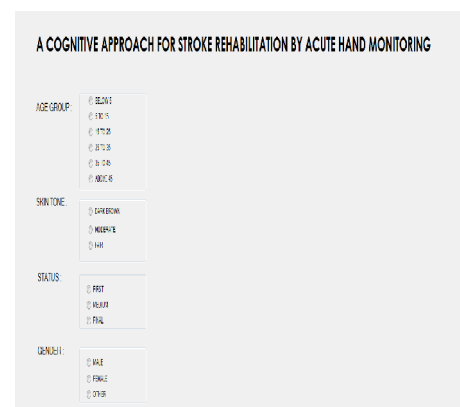


Figure 5: Customization options

The customization options are age group, skin tone, status and gender. The options are initially

picked up and registered to the system under the patient's name except for the status option. The status of the patient is determined by the angle measure. Angle measure interval of 0 -25 is determined to be 'initial' stage, 25 - 75 is the 'middle' stage and above 75 is the 'final' stage.

According to this system, the video will bring up a cognitive impulse in the patient and thus motivates him/her to move his/her finger a few degrees more than the actual measured movement. By doing so, the patient is expected to get back one's lost motor skill over a period of time.

## 6 Results

The developed system is a real time system that is yet to be subjected on stroke patients. With the help of our researches and the methods we adopted to develop the systems, we could obviously pick up time duration and the degree of movement of the finger as the primary metrics of efficiency of the system.

## 7 Conclusion and future plans

The system provides a method of providing encouragement to the afflicted patients. The prototype has been tested for one finger which can be expanded for all fingers and finally any movement. This makes the system fairly suitable for the relearning process. In addition the concept can be extended to gaming and virtual reality systems. Though the system that we developed is expected to provide rehabilitation that is less sustainable when compared to systems which consider muscle point activation in hands, this can be taken a starting point and more sophisticated permanent methods can be used for chronic patients. The further enhancement of the system will be in terms of making the rehabilitation process a stimulated virtual reality process.

## 8 References

- [1] Albert "Skip" Rizzo, GerardJounghyun Kim(2005) "A SWOT analysis of the field of virtual reality rehabilitation and therapy, Presence - Teleoperators and Virtual Environments – Special Issue, Vol. 14, No. 2, 119-146, April 2005.
- [2] S.H.Brown, J.Langan, K.L.Kern, E.A.Hurvitz "Remote monitoring and quantification of upper limb and hand function in chronic disability conditions", International journal on Disability and human development, Vol. 10, Iss. 4, Jan, 2011.
- [3] Grigore Burdea, "Keynote Address : Virtual rehabilitation :Benefits and challenges ", First International workshop on Virtual reality, 2002.
- [4] Jang Han Lee, JeonghunKu,WongeunCho,Won Yong Hahin, InY.Kim, Sang-Min Lee, Younjookang, Deog Young Kim, Taewon Yu, Brenda K.Wiederhold, Mark D.Wiederhold, Sun I.kim (2003) "A Virtual Reality system for the assessment and rehabilitation of the activities of daily living " in Cyberpsychology & Behavior, Vol.6, No.4, July, 2005.
- [5] Jimson Ngoe, TomoyaTamei, Tomohiro Shibata "Continuous Estimation of finger joint angles using music activation Inputs from surface EMG signals",www.biomedical-engineering-online.com/content/8/1/2.
- [6] Norman Alm, John L. Arnott, Iain. R. Murray, Iain Buchanan " Virtual reality for putting people with disabilities in control ", International conference on Systems, Man and Cybernetics, Vol.2, 1174-1179, Oct 1998.
- [7] Paul G. Kry, "Interaction capture and synthesis of human hands", Article, www.researchgate.net, Jan 2005.
- [8] Rosa Maria Esteves Moreira da Costa, Lu'is Alfredo Vidal de Carvalho(2004) "The acceptance of virtual reality devices for cognitive rehabilitation. :A report of positive results with Schizophrenia " in Computer Methods and Programs in Biomedicine, Vol 73, Iss. 3, 173-182, Mar 2004.
- [9] Uri Feintuch, Maya Tuchner, AdiLorber- Haddad, ZeevMeiner, Shimon Shiri "VirHab-A virtual reality system for treatment of chronic pain and disability", Virtual reality International conference, 83-83, June 2009.
- [10] Wai-Chun Lam, Feng Zou, Taku Komura "Motion editing with data glove ", Proceedings of the 2004 ACM SIGCHI International Conference on advanced computer entertainment technology, 337-342, Sept 2004.

# A Primer for Mapping Techniques on NoC Systems

M. Sacanamboy<sup>1</sup>, F. Bolaños<sup>2</sup>, and R. Nieto<sup>3</sup>

<sup>1</sup>Department of Electronics and Computer Sciences, Pontificia Universidad Javeriana, Cali, Valle del Cauca, Colombia

<sup>2</sup>Department of Electrical Engineering and Automatics, Universidad Nacional de Colombia, Medellín, Antioquia, Colombia

<sup>3</sup>Department of Electrical Engineering, Universidad del Valle, Cali, Valle del Cauca, Colombia

**Abstract** - *This paper is aimed to present a detailed description of the main factors which must be considered for task mapping onto Network on Chip (NoC) systems. A survey of the most representative and outstanding reported works is presented, along with conclusions and future work regarding such a review.*

**Keywords:** NoC (Network on Chip) Systems, Task mapping.

## 1 Introduction

In order to cope with performance requirements imposed by applications, current computing systems are moving to multicore platforms. Among such high performance systems, embedded systems represent a big fraction of the market, involving a plethora of devices such as portable devices, vehicles, wireless sensors, home devices, and so on.

Embedded systems are designed to implement special features, and are different from generic computing systems in the fact that they are devoted to implement one or more specific functionalities. Such systems are constrained by the applications, which impose operating conditions related to some figures of merit, such as performance, real time, power consumption, cost, etc. In order to cope with such constraints, designers have conceived systems with several processing cores, which may be different from each other, and are organized on a single chip (MPSoC) [1, 2].

Heterogeneity in current MPSoC systems is related with the variety of features which are present on each system core, and allows achieving flexibility in dealing with several kinds of applications. Many of current research efforts rely on improving the interconnection and synchronization systems of such cores, for the sake of speeding up the overall performance of the system. Interconnection buses are running out of capacity when dealing with a larger number of nodes inside the system, so it is mandatory to conceive efficient and structured communication architectures for these MPSoC systems [3].

NoC systems are a current approach aimed to achieving such interconnection objectives. Among some of its appealing features, the use of NoCs is preferred because of their scalability, high performance, and modularity. Particularly, by using NoC systems it is possible to achieve concurrent communications, as well as high components reusability.

NoC systems are composed of nodes and a communication architecture, which is based on network interfaces (NIs) and routers. Routers are plugged to communication channels, and nodes access such resources by means of the NIs. Nodes are often related to computational or storage resources, or a combination of both.

One of the most critical stages in designing a current embedded system is the mapping of tasks onto the available resources of the NoC. Such stage depends on the application, as well as the target NoC architecture. Some factors which are related to such an important design stage are [4]: Application constraints, figures of merit for system optimization, available mapping tools and their limitations, available information of the system. Because of all of these issues, task mapping is classified as a hard NP-problem [5].

This work is aimed to present a first review of several factors which are involved with task mapping methodologies in NoC systems. The paper is organized as follows: Section 2 summarizes the key factors which must be taken into account in the task mapping stage for NoC systems. Section 3 surveys some of the most representative works on this issue. Conclusions and future work are presented on Section 4.

## 2 Key factors on task mapping

Due to its criticality, some key factors must be considered in the stage of mapping of tasks onto a NoC system. Such key factors are described below.

### 2.1 Target architecture

The target architecture is related to whether nodes on the NoC system are heterogeneous or homogeneous.

Heterogeneity is the most common case, because this factor may improve system performance in presence of different kinds of applications. Heterogeneity refers to having several kinds of nodes in the system (i.e., nodes may be different among them).

## 2.2 Abstraction level of the application specification

The abstraction level in which applications are described is a key factor in mapping tasks of such applications to the available resources. The first possible approach on this subject is to use Register Transfer Level, or RTL. RTL is a valuable tool for modeling and designing complex systems, and often relies on hardware description languages, such as VHDL (VHSIC Hardware Description Language) and Verilog. Such tools allow modeling a part of the NoC system such as the communication system, or even the entire system [6].

The second reported approach is based on transaction-level modeling or TLM. Transactions are defined as the event of synchronization or data exchange among system modules. This approach is appealing because it allows performing a functional verification of the system, and the modeling is based on languages such as SystemC [7]. TLM has been used successfully for synthesizing high speed MPSoC systems [8], and for modeling the communications infrastructure of a NoC [9].

## 2.3 Figures of merit

This factor refers to the optimization criteria which must be considered along the optimization process related to the mapping stage. Such optimization can be viewed as a solutions space exploration, where each solution represents a single design choice with different values for the objective functions. The task mapping process must find an acceptable solution within the space with allowable and optimized values for such functions. Among the most common figures of merit used for such optimization process, we may find: power consumption, delay time, mapping time, temperature, mean number of hops across the network, network contention, mean channel occupancy, bandwidth, and so on.

## 2.4 Common-domain semantic

This is a medium level representation which combines information both from the high level application description and from the implementation platform. Among the plethora of representations available for these purposes, graph-based approaches are the most common, with instances such as task graphs (TG), communication task graphs (CTG), communication weight graphs (CWG), communication resources graph (CRG), annotated task graphs (ATG),

synchronous and asynchronous data flow graphs (SDFG and ADFG), and so on. Some other kinds of such medium-level representation are the Petri Networks (PN), and the Kahn Process Networks (KPN).

## 2.5 Topologies

Topology refers to the way in which system nodes are physically interconnected. Topologies may be classified as either regular or irregular. Some instances of common topologies are meshes, torus, rings, and spidergon ones. Regular topologies are more constrained with respect to the connections distribution, which are generated by means of mathematical functions [2, 17]. Irregular Topologies are often the mixture of two or more regular forms, which leads to hybrid, hierarchical or totally irregular topologies.

## 2.6 Optimization algorithms

As already mentioned, the mapping stage relies on an optimization process, which searches along a solutions space, the design with a better tradeoff among the chosen figures of merit. The kind of optimization algorithm used for task mapping has a direct impact in the communications nature [10]. For instance, off-line (static) optimization forces to having predictable communication assessments, whilst dynamic algorithms allow a more flexible communication scheme.

A subset of static algorithms encompasses the so called exact approaches, which are based on mathematical modeling of the optimization problem. Integer Linear Programming (ILP), Non Integer Linear Programming, and Mixed Integer Linear Programming, are well-known instances of exact algorithms, but their drawback relies on their poor convergence performances as the problem size increases [12].

On the other hand, search-based techniques are divided in heuristic and deterministic algorithms. Deterministic algorithms are devoted to search along the whole solution space, whereas heuristic algorithms use the previous experience in order to improve the searching process. Among heuristic algorithms there are some approaches which work with evolutive techniques (transformative) and some others which produce partial solutions in an iterative fashion until a good-enough solution has been reached (constructive). Dynamic algorithms are all based on heuristics. They must be quick enough to deliver a reasonably good solution in run time, without sacrificing task mapping quality. Table 1 summarizes the taxonomy above described.

Table 1. Taxonomy of the Optimization Algorithms.

Algorithm	Nature	Kind
First Free (FF)	Dynamic	Heuristic
Nearest Neighbor (NN)		
Packing-based Nearest Neighbor (PNN)		
Minimum Maximum Channel Load (MMC), Minimum Average Channel Load (MAC)		
Path Load (PL)		
Best Neighbor (BN)		
Dynamic Spiral Mapping (DSM)		
Lower Energy Consumption based on Dependencies-Neighbor (LECDN)		
ILP, NILP, MILP	Static	Exact
Genetic Algorithm, Particle Swarm Optimization, Ant Colony Organization, Population-Based Incremental Learning	Static	Heuristic – Transformative
Binomial, Mapping Algorithm, Constructive Mapping Algorithm, Chain Mapping Algorithm, Mapping on Noc, Simulation Environment Mapping, LMAP Algorithm, Simulated Annealing, Onyx, Search Tabu	Static	Heuristic – Constructive
Branch and Bound	Static	Deterministic

## 2.7 Tools

Some software tools are available for supporting the task mapping stage in NoC design environments. Among such tools the following can be mentioned below.

- SUNMAP selects the best topology according to application constraints (power, bandwidth, communication delay) and generates the nodes allocation for the target application and architecture. The process involves three steps. Firstly, routing algorithm and allocation objectives must be selected. In second place, the best topology is chosen and thirdly, a model of the system is provided through SystemC descriptions [13].
- SMAP is a mapping and simulation tool developed in the Matlab environment, which provides several optimization choices for the solution space exploration. Some of these options are Genetic Algorithms, random, and spiral. The communication among nodes can be simulated with both deterministic routing algorithms (such as XY) or adaptive algorithms (such as west-first or backtracking). Some figures of merit assessments, such as power consumption or execution time, are provided by the tool [14].
- HeMPS is a custom platform for design and simulation of NoC-based MPSoCs. This tool is based on the Hermes network, a Noc with a two dimensional mesh topology, a XY routing algorithm, and a wormhole commutation mode. Nodes in Hermes may be a MIPS processor, a RAM memory module, a DNA module, or a NI module. First design stage in HeMPS implies identifying the application specifications and constraints. After that, some hardware platform parameters (such as the size of the network, packet size, memory size, etc.) must be settled and the partitioning algorithm is able to start. The last stage implies the task mapping of the application on the selected platform. Both static and dynamic mapping is supported. The designer is able to integrate hardware and software components to perform a simulation and validation of the whole system. The final stage generates a description of the platform by means of a Hardware Description Language (HDL) [15].
- OPNEC is an open code platform for designing and simulating NoC systems. It supports a variety of 2D and 3D NoC architectures and several topologies (mesh, torus, ring, bus). It is also capable of working with both static XY routing algorithms and adaptive approaches, and supports several kinds of processor and memory modules. Several optimization objectives might be used, such as energy consumption and communication delay. Energy assessments are achieved by means of RTL models and are aimed to provide estimations of the whole network system.



Table 2. Summary of reported mapping solutions

Ref.	Factor		
	Optimization Criterion (Figures of Merit)	Common Domain Semantic / Optimization Algorithm	Target Architecture and Abstraction Level
[18]	Execution time	CTG / Exact optimization	Homogeneous architectures and Algorithm abstraction
[19]	Energy Consumption	CTG / Heuristic Constructive	
[20]	Energy Consumption	TG / Exact optimization	Heterogeneous architecture and Algorithm abstraction
[21]	Communication cost	APCG / Heuristic Transformative	
[22]	Communication volume	CTG / Heuristic Transformative	
[23]	Multi-objective	TG / Heuristic Transformative	
[24]	Bandwidth, Area	CTG / Heuristic Constructive	
[25]	Energy Consumption, Latency	ARG / Heuristic Constructive	
[26]	Communication Cost, Bandwidth	CG / Heuristic Constructive	
[27]	Execution Time	AG / Dynamic	Heterogeneous architecture and TLM abstraction
[28]	Execution Time Energy Consumption, Average channel load, Latency	AG / Dynamic Optimization	
[29]	Energy Consumption	CTG / Dynamic optimization	
[30]	Execution Time	SDFG / Heuristic Transformative	Heterogeneous architecture and RTL abstraction
[31]	Energy Consumption, Execution Time	CWG, CRG / Heuristic Constructive Dynamic	

### 3 Reported mapping solutions

This section summarizes the most representative reported works in the subject of mapping solutions aimed to

NoC systems. In order to efficiently present such information, Table 2 relates some of the literature references, with some key factors on task mapping, as described in Section 2. Some abbreviations are used in Table 2. Their meaning is as follows. CTG: Communication Task Graph; TG: Task Graph; APCG: Application Characteristics Graph; CG: Core Graph; AG: Acyclic Graph; SDFG: Synchronous Dataflow Graph; CWG: Communication Weights Graph; CRG: Communication Resources Graph.

In Table 2, all reported solutions work with mesh topologies with exception of reference [30]. None of them is aimed to hierarchical, hybrid, or irregular topologies.

### 4 Conclusions

This document introduces a brief summary on task mapping techniques for NoC Systems. A taxonomy of key factors involving task mapping methodologies is introduced. Finally, a survey of reported works in literature, regarding tasks mapping is also presented. This survey includes some of the key factors previously presented.

According with the results summarized in Table 2, most of the mapping solutions reported in literature are aimed to mesh NoC topologies. Network regularity and ease of simulation and implementation might be part of the reasons why mesh topologies are preferred. As future work, we devise the study of mapping solutions in hierarchical and more complex network topologies.

### 5 Acknowledgements

The authors would like to thank Universidad Nacional de Colombia, Pontificia Universidad Javeriana Cali and Universidad del Valle, because of its support in the development of this work.

### 6 References

- [1] A. Guerre, N. Ventroux, David R, Merigot A. "Hierarchical network-on-chip for embedded many-core architectures," Fourth ACM/IEEE International Symposium on Networks-on-Chip (NOCS), pp. 189-196. 3-6 May 2010.
- [2] M. A. Siala, S. B. Saoud. "A survey on existing MPSoCs architectures," International Journal of Computer Applications (0975 – 8887), vol. 19 (3), pp. 28-41, Abr 2011.
- [3] F. Moraes, N. Clazans, A. Mello, L. Moller y L. Ost. "HERMES: an infrastructure for low area overhead packet-switching Networks on Chip," Elsevier, INTEGRATION, the VLSI journal 38, pp.69-93, 2004.

- Available at <http://www.sciencedirect.com/science/article/pii/S0167926004000185>.
- [4] P. Mesidis. Mapping of Real-time Applications on Network-on-Chip based MPSoCs. Tesis de Maestría Universidad de York.2011.
- [5] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Free- man & Co., 1990.
- [6] M. Grange, A. Y. Weldezion, D. Pamunuwa, R. Weerasekera, Lu. Zhonghai, A.Jantsch, D. Shippen. "Physical mapping and performance study of a multi-clock 3-Dimensional Network-on-Chip mesh," IEEE International Conference on 3D System Integration, pp. 1-7, 28-30 Sept. 2009.
- [7] F. G. Assia, *Transaction Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer 2005.
- [8] V. Zadrija, V. Sruk. "Mapping algorithms for MPSoC synthesis," Proceedings of the 33rd International Convention MIPRO, pp.624-629, 24-28 May 2010.
- [9] R. Lemaire, S. Thuries, F. Heitzmann, C. Helmstetter, P. Vivet, F. Clermidy "A flexible modeling environment for a NoC-based multicore architecture," IEEE International, pp. 140-147, 9-10 Nov. 2012.
- [10] Guerre A, Ventroux N, David R, Merigot A. "Hierarchical Network-on-Chip for Embedded Many-Core Architectures," Fourth ACM/IEEE International Symposium on Networks-on-Chip (NOCS). pp. 189-196. 3-6 May 2010.
- [11] P. K. Sahu, S. Chattopadhyay. "Survey on application mapping strategies for Network on Chip design," *Journal of Systems Architecture*, vol. 59, Issue 1, pp. 60-76, Jan 2013.
- [12] O. He, S. Dong, W. Jang, J. Bian, and D. Z. Pan, "UNISM: Unified Scheduling and Mapping for General Networks on Chip," *IEEE Trans. VLSI Syst.*, vol. 20, no. 8, pp. 1496-1509, 2012.
- [13] S. Murali, G. De Micheli. "SUNMAP: a tool for automatic topology selection and generation for NoCs," *Design Automation Conference*. pp. 914-919, 7-11 Jul 2004.
- [14] S. Saeidi, A. Khademzadeh, A. Mehran. "SMAP: An intelligent mapping tool for Network on Chip," *International Symposium on Signals, Circuits and Systems*, vol. 1, pp. 1-4, 13-14 Jul 2007.
- [15] E.A. Carara, R.P. de Oliveira, N.L.V. Calazans, F.G. Moraes. "HeMPS-A framework for NoC-Based MPSoC generation," *IEEE International Symposium on Circuits and Systems, (ISCAS)*, pp. 1345-1348, 24-27 May 2009.
- [16] C. Jueping, H. Gang, W. Shaoli, Y. Lei, L. Zan, H. Yue. "OPNEC-sim: An Efficient Simulation Tool for Network-on-Chip Communication and Energy Performance Analysis," *10th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pp. 1892-1894, 1-4 Nov. 2010.
- [17] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann/Elsevier, 2003.
- [18] S. Tosun. "Cluster-based application mapping method for Network-on-Chip," *Advances in Engineering Software*, vol. 42, n.10, pp. 808-874, Oct 2011.
- [19] L. Zhong, J. Sheng, M. Jing, Z. Yu, X. Zeng, D. Zhou. "An Optimized Mapping Algorithm Based on Simulated Annealing for Regular NoC Architecture," *IEEE 9th International Conference on ASIC (ASICON)*, pp. 389-392, 25-28 Oct. 2011.
- [20] E. Khajekarimi, M. R. Hashemi. "Energy-Aware ILP Formulation for Application Mapping on NoC Based MPSoCs," *21st Iranian Conference on Electrical Engineering (ICEE)*, pp. 1-5, 14-16 May 2013.
- [21] Y. Z. Tei, M. N. Marsono, N. Shaikh-Husin, Y. W. Hau† "Network Partitioning and GA Heuristic Crossover for NoC Application Mapping," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp.1228-1231, 19-23 May 2013.
- [22] A. Racu, L.S. Indrusiak, "Using Genetic Algorithms to Map Hard Real-Time on NoC-based Systems," *7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pp. 1-8, 9-11 Jul 2012.
- [23] F. Bolaños. *Mapping Techniques for Embedded Systems Design with Reliability Considerations*. Tesis de Doctorado Universidad de Antioquia 2012.
- [24] H. M. Harmanani, R. Farah. "A Method for Efficient Mapping and Reliable Routing for NoC Architectures with Minimum Bandwidth and Area," *Joint 6th International IEEE Northeast Workshop on Circuits and Systems and TAISA Conference, NEWCAS-TAISA*, pp. 29-32, 22-25 June 2008.
- [25] M. Janidarmian1, A. Khademzadeh, M. Tavanpour. "Onyx: A new heuristic bandwidth-constrained mapping of

cores onto tile-based Network on Chip,” IEICE Electronics Express, vol.6, n.1, pp. 1–7, 2009.

[26] Murali S, De Micheli G. Bandwidth-Constrained Mapping of Cores onto NoC Architectures. Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04). Vol 2, pp.896-901. 16-20 Feb. 2004.

[27] E. Carvalho, N. Calazans, F. Moraes. “Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs,” 18th IEEE/IFIP International Workshop on Rapid System Prototyping, pp. 34-40, 28-30 May 2007.

[28] A. K. Singh, T. Srikanthan, A. Kumar, W. Jigang. “Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms,” Journal of Systems Architecture: the EUROMICRO Journal, vol. 56, n. 7, pp. 242-255, Jul 2010.

[29] L. Ost, G. Almeida, M. Mandelli, E. Wachter, S. Varyani, G. Sassatelli, L. S. Indrusiak, M. Robert, F. Moraes. “Exploring Heterogeneous NoC-based MPSoCs: from FPGA to High-Level Modeling,” Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on. pp. 1-8. 20-22 June 2011.

[30] G. Wang, W. Gong, B. DeRenzi, R. Kastner. “Ant Colony Optimizations for Resource and Timing Constrained Operation Scheduling,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, n. 6, pp. 1010-1029, Jun 2007.

[31] C.A.M. Marcon, E. I. Moreno, N. L. V. Calazans, F.G. Moraes. “Comparison of network-on-chip mapping algorithms targeting low energy consumption,” Computers & Digital Techniques, IET, vol. 2, n.6, pp. 471-482, Nov 2008.

# Intelligent Assistive Device to Monitor Ankle-Foot Orthosis

Vengateswaran.S<sup>1</sup>, Ramani.V<sup>1</sup> and Ramalatha.M<sup>1</sup>

Department of Information Technology, Kumaraguru College of Technology,  
Coimbatore, Tamil Nadu, India

**Abstract** - Active Ankle foot orthosis is used to compensate for muscle weakness, called Drop Foot caused by stroke, spinal cord injury, multiple sclerosis or any gait pathologies associated with neuromuscular disorders. People with drop foot need assistance in taking a step since the movement is impaired due to weakening and stiffening of muscles in different portions of the leg. This paper describes a system for actuating a drop foot using electric stimulation for muscles when the disorder is acute. The position of foot is sensed and used as a feedback to controller to make a decision on whether actuation is required. The device also alerts the care taker of the patient through mobile device connected through Bluetooth and generates a progress report. A laboratory model of the proposed system was implemented. The control device can be used in several applications involving control of different types of orthoses used for gait correction.

**Keywords:** Ankle foot orthosis, drop foot, electric stimulation, passive and active AFO, semiactive AFO.

## 1 Introduction

Abnormal gaits are seen in patients with neuromuscular disorders, such as stroke, cerebral palsy (CP), amyotrophic lateral sclerosis (ALS), or multiple sclerosis (MS), where pathologies of the ankle-foot present an everyday problem, particularly over a period of time. Drop foot is one example. This is because, due to the damage of the long nerves or of the brain or spinal cord, there is uneven spread of reflexes over the different muscles of the leg. The anterior muscles of the lower leg which work for dorsiflexion become weaker while the posterior muscles become stiffer. The result of this is a drop foot. The obvious result of this disorder is the tripping and falling of the patient due to the fact that the foot drops in swing phases causing toe strikes instead of heel strikes.

This has to be treated in two phases: The first and foremost is to provide walking assistance to the patient through artificial means. The second phase is a prolonged benefit where the development of abnormal gaits can be prevented over time. Though much research has gone on in providing assistance to

drop foot both in the way of physical assistance as well as providing a confidence to the patient, most widely used solution is to wear a passive orthotic device, such as an ankle-foot orthosis (AFO), which forces the joint angle to be close to 90° [2].

### 1.1 Ankle Foot Orthoses

AFOs are of many types. It is most commonly classified as passive, semi-active, and active. Passive devices are the static devices which do not have any electronics in them but they may have mechanical elements such as springs or dampers, which enable motion control of the ankle joint during gait. These devices have no automation in sensing or responding. Semi-active AFOs are the ones using computer to control variance in compliance or damping of the joint in real-time. Here the control can be automated but the monitoring is done manually. Fully active AFOs are meant to have onboard or tethered sources of power, one or more actuators to move the joint, sensors, and a computer or onboard electronics used to control the application of assistance during gait.

Active ankle-foot orthoses (AAFOs) are orthotic devices intended to assist or to restore the motions of the ankle-foot complex and are based on force-controlled actuator [1].

These devices are more concentrated on the actuation of the affected limb and these devices have been developed to assist the patient but the progress of the patient is not monitored. They are not focused on long term assessment. When the device is simply mounted on patient to make the limb move, it does not mean that the patient will recover. There is a need to monitor the patient continuously and motivate them by showing them their progress.

Laboratory monitoring is usually implemented in fully active AFOs. But the monitoring is done by connecting the device to the computer. This makes the device non portable. In order to make the device portable with real time monitoring, we need to establish wireless communication between the device and the monitoring system. In this case we cannot make the

monitoring device to be at one place. We need to make the monitoring device also portable.

## 1.2 Android Device Monitoring

Android is the most powerful mobile operating system, which is used in most of the mobile devices now-a-days. It has wide range of interfacing facilities in it to communicate with other embedded devices. But using the android operating system and be able to get information on the condition of a patient from a long distance will keep the family happier.

## 2 Literature survey:

Veneva I[2010] developed an autonomous adaptive system for actuation, data processing and control of active ankle-foot orthosis. This system is composed of control system with actuation by sensing the improper foot position and viewing the results on a separate GUI display for monitoring. This system is just for laboratory monitoring with a graphical program written in MATLAB. The system cannot be used for continuous monitoring [1]. Yong-Lae Park et al [2011] have created the design of an active soft ankle foot orthotic device powered by pneumatic artificial muscles for treating gait pathologies associated with neuromuscular disorders. The design is inspired by the biological musculoskeletal system of a human foot and a lower leg, and mimics the muscle-tendon-ligament structure. A key feature of the device is that it is fabricated with flexible and soft materials that provide assistance without restricting degrees of freedom at the ankle joint .But the system is too heavy to mount on the patient [2].

Blaya and Herr [2004] developed an adaptive control system for ankle foot orthosis. They used a serious elastic actuator and ankle angle sensor which is a circular potentiometer to detect the angle by resistance encountered. They applied variable-impedance on the adaptive basis by using computer controlled AFO. In this system importance for continuous monitoring is not given [3]. Kenneth Alex Shorter et al. [2013] have done study on Technologies for Powered Ankle-Foot Orthotic Systems. They made a complete comparison of strategies about all AFOs, on powering, structure and efficiency of the AFOs that are currently available on the field [4].

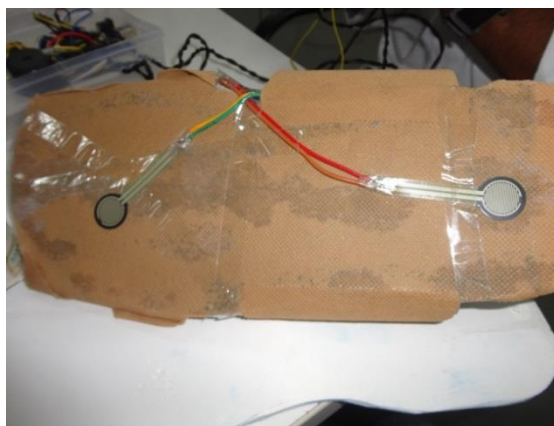
## 3 Method

The active ankle-foot orthoses has three basic components: Electro-Mechanical, Control and Alerting and report generation unit. The Control Unit receives values of the ankle's angle and toe clearance from sensors (Flex sensor to detect the angle and pressure sensor to detect the toe clearance). By

receiving the values from the sensors, the arduino microcontroller generates the flexion/extension motions and controls the Electro-Mechanical Unit.



**Figure 1 - Ankle Angle detection Sensor (Flex sensor)**



**Figure 2 - Force Sensor embedded inside insole**

### 3.1 Electro-Mechanical Unit

Electro-Mechanical Unit consists of the Actuator which actuates the leg to the normal position whenever it detects a drop foot condition on the leg. The flex sensor is placed on the top of the leg across the ankle as shown in figure 1. The pressure sensors will be placed inside the insole as shown in the figure 2. The actuator does not affect the normal leg movement. It actuates the leg only on the condition of drop-foot.

### 3.2 Alerting and Report generation Unit

Alerting and Report generation unit is a separate device which is powered by android.

### 3.3 Control Unit

The control unit sends the leg position to that device. The device is meant to be with care taker. The device alerts the care taker in case of any instability found during gait. It also records the situations of instability of the patient. It stores data as number of times the patient faced instability condition in the day. A weekly report will be generated on the device and it is sent to the respective doctor.

## 4 System Design

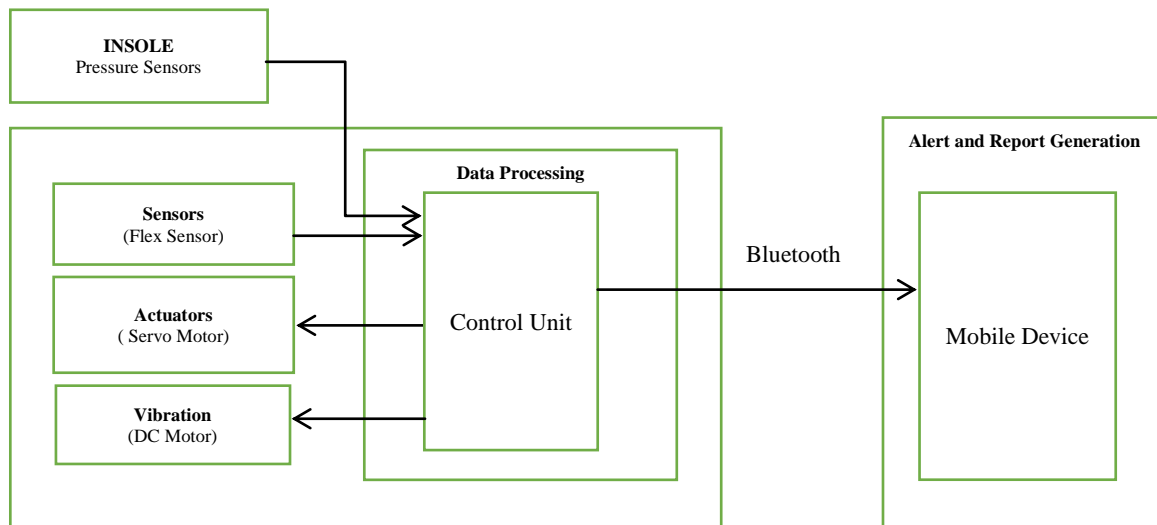


Figure 3 – System Design

The complete autonomous system consists of four primary components - sensing, data processing, actuation and alerting and report generation.

The sensor system has been mounted onto two basic components: insole with pressure sensor and flex sensor for ankle angle detection. During walking, the processing unit gathers and digitizes the information from the sensors. In monitoring mode these data are transferred through the Bluetooth to android mobile device for alerting and report generation.

The system power source is an important part of the project. Three separate power sources are required. One is to power arduino, second is for the actuation part and next is for Bluetooth module.

### 4.1 Sensing Unit

Sensing unit consists of three sensors. Two pressure sensors are used for toe clearance and one flex sensor for ankle's angle detection. Two pressure

sensors will be embedded inside the insole. The flex sensor will be placed above the ankle with the help of socks. The pressure sensors gives value depending on the pressure/force applied on it. We need to map the values according to our need. We have setup a threshold value above which we detect the toe on ground state.

Same as the pressure sensor, the flex sensor values have to be mapped in order to get the ankle's angle.

### 4.2 Data Processing Unit

Data Processing Unit consists of the arduino UNO which is microcontroller featuring basic hardware peripherals such as Analog to Digital Converter (ADC), serial communication to connect with Bluetooth and PWM (Pulse Width Modulation) signal to drive motor according to input signal. All the sensors will be connected to the analog input pins of arduino. The arduino converts the analog signal into digital signal to process it.

The condition for actuation and alerting is that the ankle must be leaned in such a manner it becomes  $180^\circ$  and the toe must be on the ground. When the foot is dropped, the system should actuate the leg to normal position. Servo motors are used for the part of actuation of leg. When the foot drops to the improper position, the system automatically actuates the ankle to move the foot to normal position.

Improper condition is found by the three sensors namely S1, S2 and F1. The two pressure

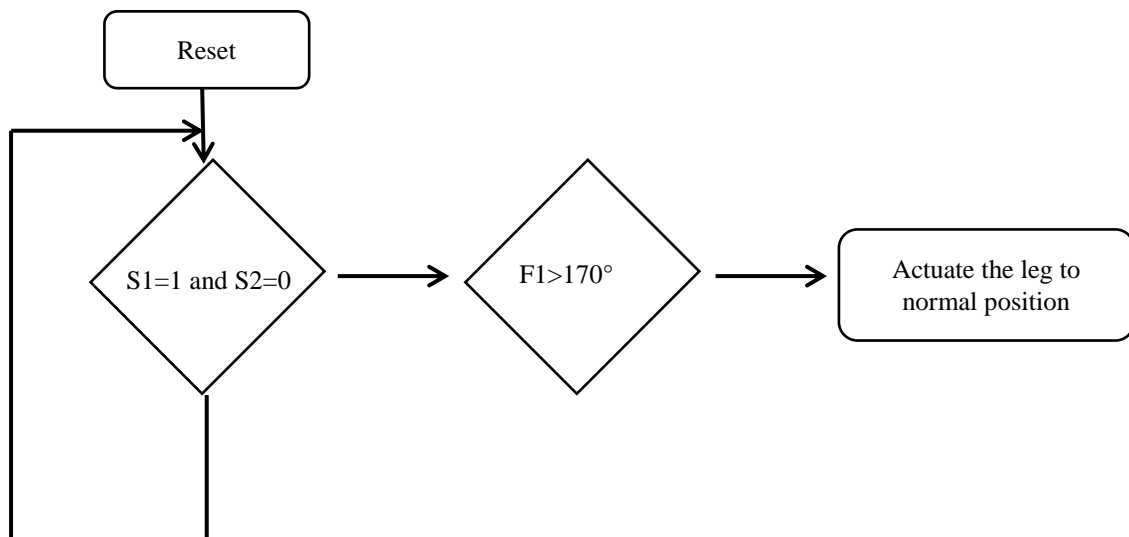


Figure 4 – Flow of Data Processing unit

sensors are S1 and S2 where S1 is fixed on the toe and S2 is on the sole below the ankle. The angle of ankle is determined by the flex sensor F1.

### 4.3 Actuation Unit

Actuation unit consists of the servo motors. The servo motors have the torque of 60KG in normal conditions. If the person is heavier than 60KG then we need to opt for higher torque servos. When the drop foot condition is met, the servo rotates back to the normal position (90°) thus actuating the foot to normal position. The servos will be fixed near the ankle.

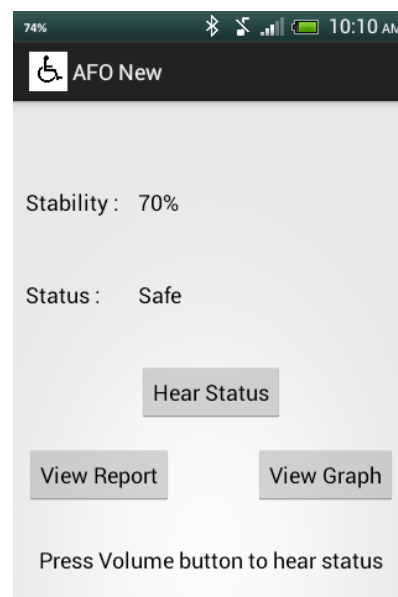
### 4.4 Alerting and report generation Unit

The flex and the pressure sensor values will be sent to the android device which is connected to the Bluetooth transmitter mounted on the leg. The android device processes it and alerts the care taker whenever there is an instance of about to fall or a fall. It also stores the occurrence of instability in the database consistently. It sends a weekly report to the registered medical specialist about the number of occurrences per day on the week through email so as to keep track.

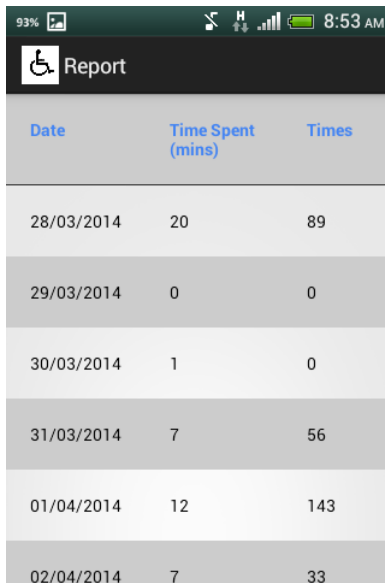
## 5 Experimental Results



Figure 5 - Ankle Foot Orthosis with alerting and Monitoring

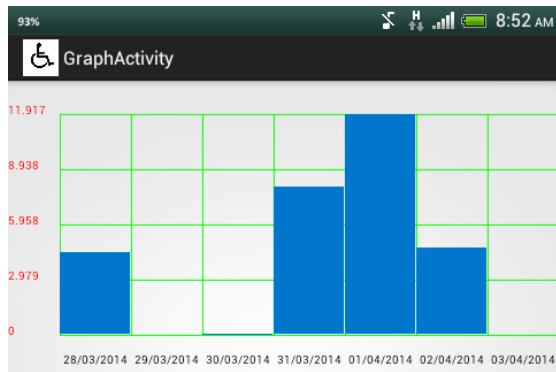


(a)



Date	Time Spent (mins)	Times
28/03/2014	20	89
29/03/2014	0	0
30/03/2014	1	0
31/03/2014	7	56
01/04/2014	12	143
02/04/2014	7	33

(b)



(c)

**Figure 6 – Android Application for mobile monitoring**

**a - Status Screen , b - Consolidated Progress Data, c - Consolidated Progress Analysis**

The device has been tested on two patients affected by cerebral palsy for a week by letting them wearing it. As commonly known, the patient affected by cerebral palsy lacks the ability to walk normally and is prone to fall often. Hence this device has been designed to help them walk normally as any other normal human. There are some situations where they were meant to fall by misplacing their leg but at those situations they have been brought to normal state by the device. Such situations have been saved in the database and the report has been generated. The Graph has also been generated to have instant understanding about the patient's health condition.

## 6 Conclusion

The monitoring system attached to the actuating system gives overall progress report on the patient's condition which shows the continuous improvement of the patient. This device has been designed at the request of and in consultation with the therapists at the Occupational therapy unit of Kovai Medical Centre and Hospital, Coimbatore.

## 7 References

- [1] Veneva I, "Intelligent Device for Control of Active Ankle-Foot Orthosis", Proceedings of the 7<sup>th</sup> IASTED International Conference Biomedical Engineering (biomed2010), February 17-19, 2010 Innsbruck, Austria.
- [2] Yong-Lae Park, Bor-rong Chen, Diana Young, Leia Stirling, Robert J. Wood, Eugene Goldfield, and Radhika Nagpal, "Bio-Inspired Active Soft Orthotic Device For Ankle Foot Pathologies", 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, September 25-30, 2011. San Francisco, CA, USA.
- [3] Joaquin A. Blaya and Hugh Herr, "Adaptive Control of A Variable-Impedance Ankle-Foot Orthosis To Assist Drop-Foot Gait", IEEE Transactions On Neural Systems And Rehabilitation Engineering, VOL.12, No. 1, MARCH 2004.
- [4] Kenneth Alex Shorter, Jicheng Xia, Elizabeth T. Hsiao-Wecksler, Member, IEEE, William K. Durfee and G'eza F. Kogler, "Technologies for Powered Ankle-Foot Orthotic Systems: Possibilities and Challenges", IEEE/ASME Transactions on Mechatronics, VOL. 18, NO. 1, FEBRUARY 2013
- [5] Anindo Roy, Hermano Igo Krebs, Dustin J. Williams, Christopher T. Bever, Larry W. Forrester, Richard M. Macko, and Neville Hogan. "Robot-Aided Neurorehabilitation: A Novel Robot for Ankle Rehabilitation", IEEE Transactions on Robotics, VOL.25, No. 3, JUNE 2009.



# FPGA Realization of Hybrid Carry Select-cum-Section-Carry Based Carry Lookahead Adders

V. Kokilavani

Department of PG Studies in Engineering  
S. A. Engineering College  
(Affiliated to Anna University)  
Chennai 600 077, India

P. Balasubramanian

Department of Computer Science and  
Engineering  
S. A. Engineering College  
Chennai 600 077, India

H. R. Arabnia

Department of Computer Science  
University of Georgia  
415 Boyd Building  
Athens, Georgia 30602-7404, USA

**Abstract**—FPGA based synthesis of conventional carry select adders, carry select adders featuring add-one circuits (binary to excess-1 code converters), carry select adders sharing common Boolean logic term, hybrid carry select-cum-carry lookahead adders, and hybrid carry select-cum-section-carry based carry lookahead adders are described in this paper. Seven different carry select adder structures corresponding to 32 and 64-bit addition were described topologically using Verilog HDL, and were subsequently implemented in a 90nm FPGA (Spartan-3E). The results obtained show that the carry select adder utilizing section-carry based carry lookahead logic encounters minimum data path delay among all its counterparts.

**Keywords**—Carry select adder; Carry lookahead; FPGA; High-speed design; Binary to excess-1 converter; Common Boolean logic

## I. INTRODUCTION

The carry select adder (CSA) is a high-speed adder [1] with typical propagation delay of  $O(\sqrt{n})$ , where 'n' denotes the adder size. With respect to physical realization of CSAs, there are three basic types – a topology which consists of full adder modules and 2:1 multiplexers (MUXes), an architecture which consists of full adders, binary to excess-1 code converters (BECs) and 2:1 MUXes, and another structure which is built on the basis of sharing of common Boolean logic (CBL) term. In the existing literature, conventional CSAs with and without BEC logic, hybrid CSAs<sup>1</sup> encompassing both CSA and carry lookahead (CLA) adder topologies, and CSAs based on CBL term sharing have been implemented in ASIC and/or FPGA platforms [2] – [12]. In this paper, seven different CSAs have been constructed using Verilog HDL in a topological sense viz. conventional CSA (CCSA), CSA incorporating BEC logic (CSA-BEC), hybrid CSA with a CLA adder in the least significant stage (CSA\_CLA), hybrid CSA with CLA adder in the least significant stage and featuring BEC logic (CSA-BEC\_CLA), CSA based on CBL term sharing (CSA-CBL), hybrid CSA with a section-carry based carry lookahead (SCBCLA) logic incorporated in the least significant stage (CSA\_SCBCLA), and finally, CSA with a least significant SCBCLA section including BECs (CSA-BEC\_SCBCLA). Among these, the last two hybrid CSA architectures represent the novelty component of this paper. Referring to a recent work [13], it was shown that the SCBCLA adder promised better performance in terms of delay than a traditional CLA adder. For example, a 64-bit SCBCLA adder exhibited 14% less data path delay than a conventional 64-bit CLA adder.

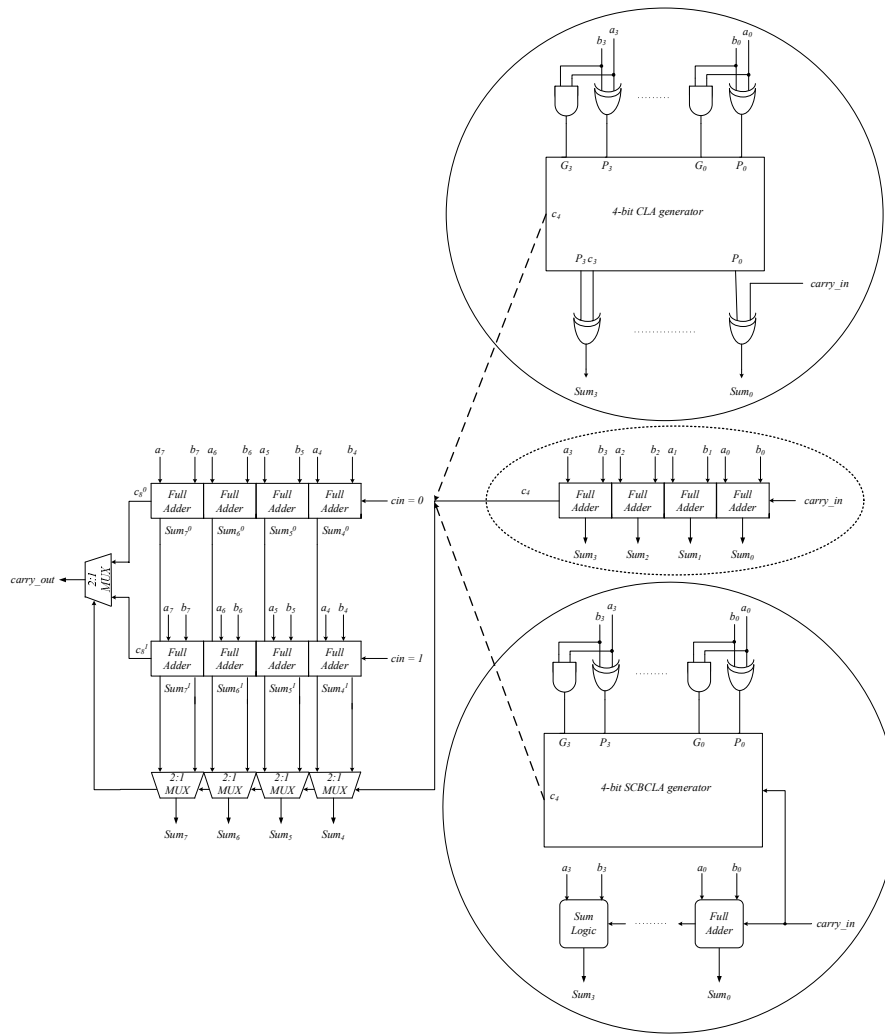
<sup>1</sup> Hybrid carry select and carry lookahead adders shall commonly be referred to as hybrid carry select adders in this paper for simplicity.

In the rest of this paper, with an 8-bit addition as a running example, Section 2 discusses the basic architectures of CCSA, CSA-BEC and CSA-CBL adders. Section 3 deals with hybrid CSA topologies featuring a CLA in the least significant stage as a replacement for the ripple carry adder (RCA). The new CSA\_SCBCLA and CSA-BEC\_SCBCLA adder architectures are also described in this section. Section 4 presents the delay and area results for seven different CSA variants corresponding to 32-bit and 64-bit additions, based on synthesis targeting a 90nm FPGA, followed by the conclusions.

## II. CONVENTIONAL CARRY SELECT ADDERS

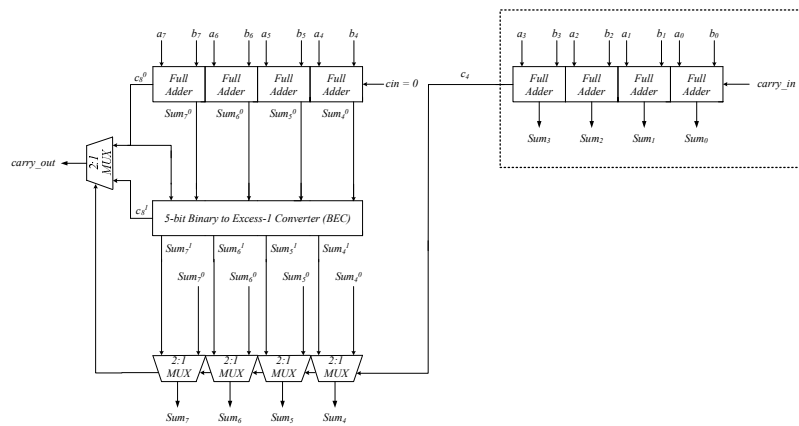
The traditional CSA architectures are shown in Figure 1, for the example case of 8-bit addition. Figure 1(a) shows the CSA partitioning the specified data inputs into two groups and addition within the groups are carried out in parallel using a dual RCA, composed from full adder blocks. The full adder is an arithmetic building block that adds an augend and addend bit (say,  $a_i$  and  $b_i$ ) along with any carry input ( $c_m$ ), producing two outputs, namely sum ( $Sum_i$ ) and a carry output ( $c_{out}$ ). In case of the CCSA shown in Figure 1(a), the full adders present in the most significant nibble position are duplicated with carry inputs of 0 and 1 assumed, i.e. a 4-bit RCA with a carry input of 0 and another 4-bit RCA with a carry input of 1 are realized. Both these RCAs have the same augend and addend inputs. While the least significant 4-bit RCA would be adding augend inputs ( $a_3$  to  $a_0$ ) with addend inputs ( $b_3$  to  $b_0$ ), the more significant 4-bit RCA would be adding in parallel augend inputs ( $a_7$  to  $a_4$ ) with addend inputs ( $b_7$  to  $b_4$ ), with 0 and 1 serving as input carries. Due to two addition sets, two sets of sum outputs and output carries are produced – one based on 0 as carry input and another based on 1 as carry input, which are in turn fed as inputs to 2:1 MUXes. The number of MUXes used depends on the size of the RCA duplicated. To determine the true sum outputs and the real value of carry overflow of the higher order nibble position of the CCSA, the carry output ( $c_4$ ) from the least significant 4-bit RCA is used as the common select input for all MUXes corresponding to more significant RCA stage, thereby the correct result pertaining to either RCA with 0 as carry input or RCA with 1 as carry input is output.

The CSA-BEC category is rather different from the CCSA in that instead of having an RCA with a presumed carry input of 1 in a more significant position, BEC circuit is introduced. The BEC logic adds binary 1 to the least significant bit of its binary inputs and produces the resultant sum at its output. As seen in Figure 1(b), the BEC accepts as input the sum and carry



(a) 8-bit conventional CSA featuring dual RCAs (CCSA type)

Note: Circuits enclosed within top and bottom circles represent 4-bit CLA and 4-bit SCBCLA adders respectively. Circuit enclosed within the ellipse signifies 4-bit RCA. Usage of 4-bit CLA adder instead of 4-bit RCA results in CSA\_CLA configuration. Alternatively, usage of 4-bit SCBCLA adder instead of 4-bit RCA leads to CSA\_SCBCLA architecture



(b) 8-bit conventional CSA incorporating add-one circuit (BEC logic): CSA-BEC structure

Note: Circuit enclosed within the rectangle represents 4-bit RCA. Usage of 4-bit CLA adder instead of 4-bit RCA results in CSA-BEC\_CLA configuration. Alternatively, usage of 4-bit SCBCLA adder instead of 4-bit RCA leads to CSA-BEC\_SCBCLA architecture

Fig. 1. Conventional CSA topologies (with/without BEC logic), which may embed CLA and SCBCLA sections to form hybrid CSA architectures

outputs of the RCA having a presumed carry input of 0, adds 1 to the input, and produces the resulting sum as output. Now the correct result exists between choosing the output of the RCA featuring an input carry of 0, and the output of the BEC logic. Again, carry output  $c_4$  of the least significant RCA is used for determining the correct set of outputs. The logic diagram corresponding to the 5-bit BEC is shown in Figure 2, and its governing equations are,

$$Sum_4^1 = \overline{Sum_4^0} \tag{1}$$

$$Sum_5^1 = Sum_5^0 \oplus Sum_4^0 \tag{2}$$

$$Sum_6^1 = Sum_6^0 \oplus (Sum_5^0 \bullet Sum_4^0) \tag{3}$$

$$Sum_7^1 = Sum_7^0 \oplus (Sum_6^0 \bullet Sum_5^0 \bullet Sum_4^0) \tag{4}$$

$$c_7^1 = c_7^0 \oplus (Sum_7^0 \bullet Sum_6^0 \bullet Sum_5^0 \bullet Sum_4^0) \tag{5}$$

The CSA structure constructed on the basis of CBL term sharing is depicted through Figure 3. The CSA-CBL adder is founded upon the functionality of the full adder block, whose underlying equations are given below assuming  $a$ ,  $b$  and  $c_{in}$  as the primary inputs and  $Sum$  and  $C_{out}$  as the primary outputs.

$$Sum = a \oplus b \oplus c_{in} \tag{6}$$

$$C_{out} = (a + b) \bullet c_{in} + (ab) \bullet \overline{c_{in}} \tag{7}$$

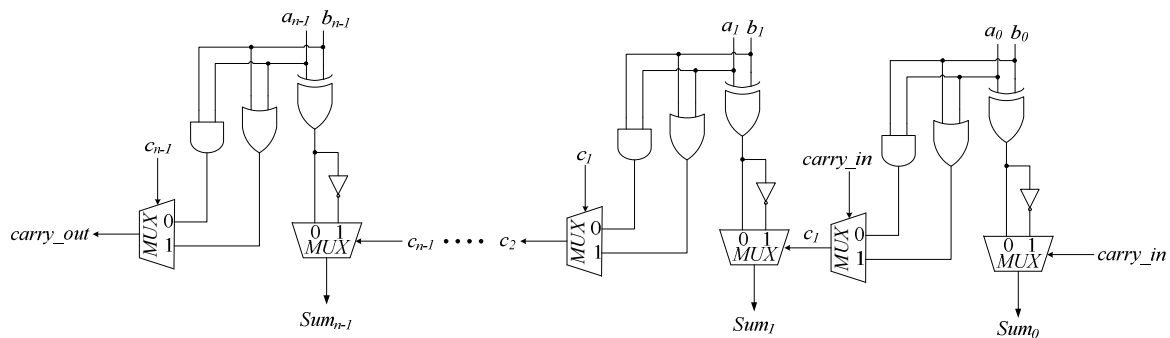


Fig. 3. CSA topology based on sharing of CBL term

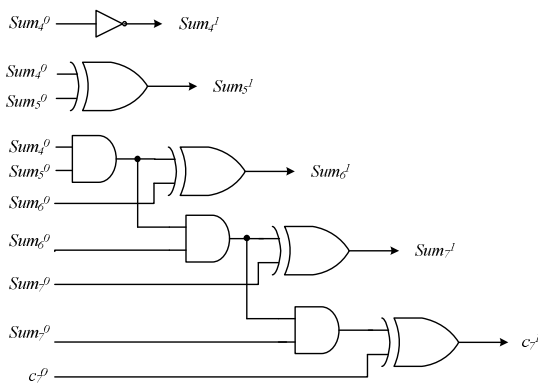


Fig. 2. 5-bit BEC (add-one) logic circuit

From (6) and (7), it may be understood that for a carry input of 0, equations (6) and (7) reduce to:  $Sum = a \oplus b$  and  $C_{out} = ab$  respectively, while for an assumed carry input of 1, equations (6) and (7) become  $Sum = \overline{a \oplus b}$  and  $C_{out} = a + b$ . Based on this principle, the sum and carry outputs for both possible values of input carries are generated simultaneously and fed as inputs to two 2:1 MUXes. The correct sum and carry outputs are then determined with the carry input serving as the select input for the two MUXes. Though exorbitant dual RCAs and RCA with BEC logic structures are eliminated through this approach, leading to substantial savings in terms of area and possibly less power dissipation, nevertheless, since carry propagation occurs from stage-to-stage; the data path delay varies proportionately with the size of the cascade. As a consequence, the delay of the CSA-CBL adder tends to be close to that of RCA, which is confirmed through simulations.

### III. HYBRID CARRY SELECT ADDERS

Apart from synthesizing basic CSA topologies viz. CCSA and CSA-BEC variants, hybrid CSA architectures involving CLA and SCBCLA logic in the least significant stage were also synthesized with the intention of minimizing maximum combinational path delay. It is well known that a CLA adder is faster than a RCA, and hence it may be worthwhile to include a CLA adder in the CSA structure to replace the least significant

RCA to mitigate the propagation delay. Although the concept of CLA is widely understood, the concept of SCBCLA may not be well known and hence to elucidate the distinction between CLA and SCBCLA modules, sample 4-bit lookahead logic realized using these two styles is portrayed in Figure 4 for an illustration. For details regarding diverse SCBCLA logic implementations and realization of various SCBCLA adders, the interested reader is directed to references [13] [14], which constitute prior works within the realm of synchronous and self-timed (asynchronous) design. The SCBCLA generator shown within the circle in Figure 4 produces look-ahead carry signal corresponding to a section or group of adder inputs, while the conventional CLA generator shown within the rectangle produces look-ahead carry signals corresponding to each pair of augend and addend inputs. The SCBCLA module

differs from a conventional CLA module in that bit-wise look-ahead carry signals need not be computed. The XOR and AND gates used for producing propagate and generate signals ( $P_0 - P_4$  and  $G_0 - G_4$ ) are highlighted using dotted lines in Figure 4.

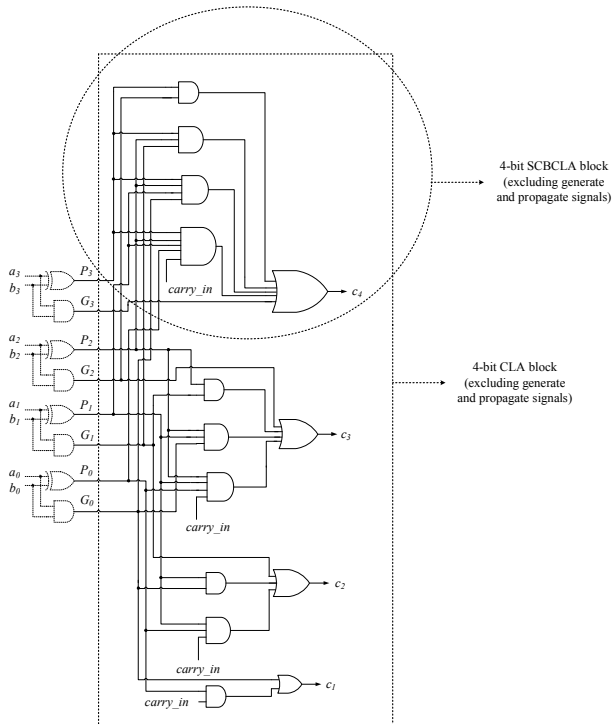


Fig. 4. 4-bit CLA and SCBCLA generator modules

Exemplar 8-bit hybrid CSAs with/without BEC logic and featuring traditional CLA adders in the least significant stage viz. CSA\_CLA adder and CSA-BEC\_CLA adder are shown as part of Figure 1 due to space constraints. They are obtained by replacing the least significant RCAs shown within the ellipse and rectangle in Figures 1(a) and 1(b) with the 4-bit CLA adder shown enclosed within the circle at the top of Figure 1(a). Similarly, 8-bit hybrid CSAs with/without BEC logic and featuring SCBCLA adders in the least significant stage viz. CSA\_SCBCLA and CSA-BEC\_SCBCLA adders are obtained by replacing the least significant RCAs shown within the ellipse and rectangle in Figures 1(a) and 1(b) with the 4-bit SCBCLA adder shown within the circle at the bottom of Figure 1(a). Unlike a typical CLA adder which consists of propagate-generate logic, CLA generator, and series of XOR gates to produce sum outputs, the SCBCLA adder contains propagate-generate logic, SCBCLA generator, full adders, and sum logic as shown in Figure 1. The sum logic is basically derived from the full adder in that only the sum output is produced with no extra carry output. While rippling of carries occurs within the carry-propagate adder portion constituting the SCBCLA adder, which produces the requisite sum outputs, the look-ahead carry signal pertaining to an adder section is generated in parallel.

#### IV. RESULTS AND INFERENCES

32 and 64-bit conventional and hybrid CSAs corresponding to various architectures were described topologically in Verilog HDL and were synthesized targeting a 90nm FPGA device

(Spartan-3E: XC3S1600E). The maximum combinational path delay has been estimated after automated place and route and is ascertained from the design summary. The critical path timing and area results (in terms of number of LUTs) of different CSA structures are mentioned in Table 1. Several carry chain partitions were considered for the 32-bit and 64-bit CSAs and among them; the optimized delay value is found out and listed in Table 1. The optimum delay and area values corresponding to 32 and 64-bit CSAs are highlighted in bold-face in the Table. Percentage increases in delay for different CSAs in relative comparison with the CSA\_SCBCLA adder is indicated within brackets in the third column of the Table.

The 32-bit RCA exhibits maximum propagation delay of 30.604ns, while the 32-bit CSA\_SCBCLA adder encounters approximately half its data path delay and exhibits the least latency among all CSAs. For 64-bits, it is a similar story with the CSA\_SCBCLA adder featuring the least latency and encounters just about one-third the delay of 64-bit RCA, whose critical path delay is 71.555ns. Considering both 32 and 64-bit additions, it is found that the CSA\_SCBCLA adder leads to a delay optimal solution minimizing the best delay metrics of conventional CSAs (CCSA and CSA-BEC) and CSA\_CLA by 24.7% and 15.6% respectively. However, with respect to area occupancy CSA-CBL adders are preferable, which consume 59.4% less LUTs than CSA\_SCBCLA adders on average.

TABLE I. MAXIMUM PATH DELAY AND AREA OF 32 AND 64-BIT CSAS CORRESPONDING TO CONVENTIONAL AND HYBRID ARCHITECTURES

CSA Size	Type of CSA Adder Architecture	Maximum Delay (ns); %age delay ↑	Area (# LUTs)
32-bits	CCSA	19.109; (28.4%)	150
	CSA-BEC	19.481; (30.9%)	120
	CSA-CBL	37.604; (152.6%)	<b>63</b>
	CSA_CLA	18.992; (27.6%)	150
	CSA-BEC_CLA	19.481; (30.9%)	120
	CSA_SCBCLA	<b>14.887</b>	143
	CSA-BEC_SCBCLA	19.534; (31.2%)	127
64-bits	CCSA	28.335; (34.4%)	330
	CSA-BEC	28.610; (35.7%)	241
	CSA-CBL	70.525; (234.5%)	<b>129</b>
	CSA_CLA	23.606; (11.9%)	263
	CSA-BEC_CLA	28.293; (34.2%)	241
	CSA_SCBCLA	<b>21.084</b>	331
	CSA-BEC_SCBCLA	27.602; (30.9%)	257

#### REFERENCES

- [1] O.J. Bedrij, "Carry-select adder," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 3, pp. 340-346, 1962.
- [2] Y. Kim, L.-S. Kim, "64-bit carry-select adder with reduced area," *IET Electronics Letters*, vol. 37, no. 10, pp. 614-615, 2001.
- [3] R. Yousuf, Najeeb-ud-din, "Synthesis of carry select adder in 65nm FPGA," *Proc. IEEE Region 10 TENCON Conference*, pp. 1-6, 2008.
- [4] H.G. Tamar, A.G. Tamar, K. Hadidi, A. Khoei, P. Hoseini, "High speed area reduced 64-bit static hybrid carry-lookahead/carry-select adder," *Proc. 18th IEEE International Conference on Electronics, Circuits and Systems*, pp. 460-463, 2011.
- [5] Y. He, C.-H. Chang, J. Gu, "An area efficient 64-bit square root carry-select adder for low power applications," *Proc. IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 4082-4085, 2005.
- [6] M. Alioto, G. Palumbo, M. Poli, "A gate-level strategy to design carry select adders," *Proc. IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 465-468, 2004.

- [7] W. Jeong, K. Roy, "Robust high-performance low-power carry select adder," *Proc. Asia and South Pacific Design Automation Conference*, pp. 503-506, 2003.
- [8] Y. Chen, H. Li, K. Roy, C.-K. Koh, "Cascaded carry-select adder (C<sup>2</sup>SA): a new structure for low-power CSA design," *Proc. International Symposium on Low Power Electronics and Design*, pp. 115-118, 2005.
- [9] J. Monteiro, J.L. Guntzel, L. Agostini, "A1CSA: An energy-efficient fast adder architecture for cell-based VLSI design," *Proc. 18<sup>th</sup> IEEE International Conference on Electronics, Circuits and Systems*, pp. 442-445, 2011.
- [10] A. Neve, H. Schettler, T. Ludwig, D. Flandre, "Power-delay product minimization in high-performance 64-bit carry-select adders," *IEEE Transactions on VLSI Systems*, vol. 12, no. 3, pp. 235-244, 2004.
- [11] B. Ramkumar, H.M. Kittur, "Low-power and area-efficient carry select adder," *IEEE Transactions on VLSI Systems*, vol. 20, no. 2, pp. 371-375, February 2012.
- [12] I.-C. Wey, C.-C. Ho, Y.-S. Lin, C.-C. Peng, "An area-efficient carry select adder design by sharing the common Boolean logic term," *Proc. International Multiconference of Engineers and Computer Scientists*, vol. II, pp. 1091-1094, 2012.
- [13] K. Preethi, P. Balasubramanian, "FPGA implementation of synchronous section-carry based carry look-ahead adders," *Proc. IEEE 2<sup>nd</sup> International Conference on Devices, Circuits and Systems*, pp. 260-263, 2014.
- [14] P. Balasubramanian, D.A. Edwards, H.R. Arabnia, "Robust asynchronous carry look-ahead adders," *Proc. 11<sup>th</sup> International Conference on Computer Design*, pp. 119-124, 2011.



**SESSION**  
**POSTERS**

**Chair(s)**

**TBA**





# The development of NFC-based medication management system for elderly patients

Taebok Yoon<sup>1</sup>, and Jonghee Lee<sup>2</sup>

<sup>1</sup>Department of Computer Software, Seoil University, Seoul, South of Korea

<sup>2</sup>Department of Strategic Planning, INPOESTI Company, Seoul, South of Korea

**Abstract** - *With entering into an aged society, more and more people pay attention to their health and disease. Especially, it is significant for senior citizens with chronic disease to have a medication system as it is directly related to their lives. This study suggests the NFC-based medication management system for elderly patients. By developing hardware and software, the suggested method can assist elderly patients to take medicines properly without missing or abusing through guidance and alarm and maintain their health.*

**Keywords:** near field communication (NFC), medication management system, elderly patients

## 1 Introduction

The aged population has been rapidly increasing due to improvement of living environment and development of medical technology. The ratio of Korea's aged population, which entered into the aged society with over 7 percent in 2000, is forecast to rise to 14.3 percent in 2022 which means the aged society, and 19.7 percent in 2026, the super aged society[1]. Most of developed countries also have more proportion of aging population earlier so that more of high-tech companies in Silicon Valley, U.S. are developing technologies not on young generation but on the elder or their children or providers who live with to their load off. In addition, businesses are developing various goods for the aged or handicapped as they are on rise as a major customer layer. Compared with other age groups, senior citizens need more attention on management of their health and diseases. Researches show that most of the elder, who are over sixty-five years, has more than three chronic diseases including high blood pressure. A clinical statistics indicated that four out of ten senior citizens take more than four medications. Some seniors are even hospitalized from drugs' side effects. However, they need to be cautious not to worsen diseases or ruin their health by not taking medications to avoid side effects. These elderly patients understand how important regular medications are to treat their diseases, but have trouble taking medicines due to cognitive impairment and

decreased memories[2]. They need special attention as they take various medicines for chronic diseases and complications, so there are more possibilities to have side effects or harmed from drug interactions. This research will analyze cases of the elderly patients' medication manage system and suggest NFC-based smart medication management system.

## 2 Related works

There are precedent medication management products such as medication organizers with alarm system, pill bottles with a built-in wireless module, medication alarms and information management software. Firstly, the product of Apex Medical Corp., U.S. is a medication organizer with alarm system to set timer with LED screen and its buttons. GlowCap of Vitality is a pill bottle with a built-in wireless modem of Telit to alarm patients to take their medicines.



Fig 1. (Up) Apex Weekly Pill Turtle Organizer, (Down) Vitality-Glow Caps[3]

This research was supported by Leap Technology Development Program funded by Small and Medium Business Administration (SMBA). (No. C0142730)

They also have functions such as monitoring when patients took medications, reporting if patients took medication on time and requesting to pharmacies to fill medications by pressing a button. However, it is uncomfortable for elderly patients to take medicines out of their package to put into cases. There is an Android application named "Pills on the go", a medication alarm and information management software. This app can support basic medication-related alarming but can't manage medication as it is software.

### 3 NFC-based medication management system

This suggested system is divided into a bottle-typed medicine attachment for NFC-based medication management and NFC-based medication information management software. First, a cap is developed for NFC-based medication information management which is attachable to existing medication for serving elderly patients' medication easily. On top of the NFC-based medication information management cap, LED Display module is installed with functions of display and alarm so that it rings once it's time for a user to take medicines. Then, if the user's smart phone reads NFC tag module on the bottle cap, the user can input and check medication schedule of the medicine with ease. The user also can turn off the alarm (light and/or sound) on the cap for medication information management, which is also significant information to check medication. The entire operation process of suggested system is as Figure 2.

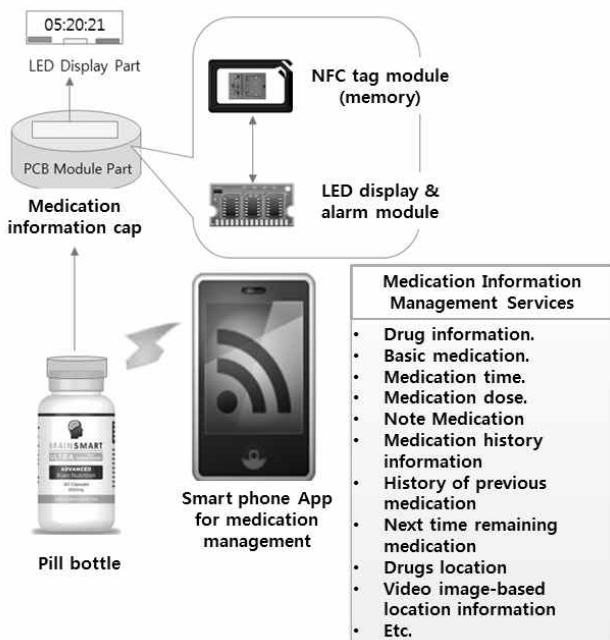


Fig 2. The workflow for medication management services

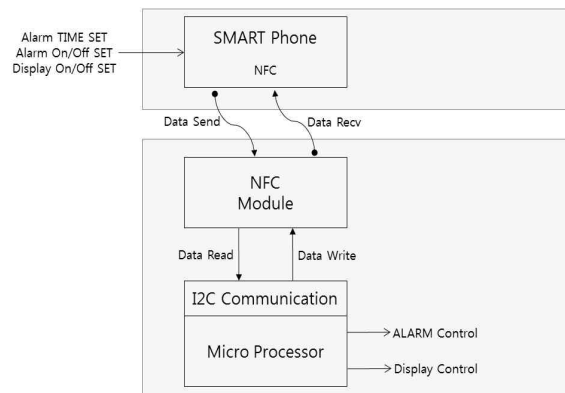


Fig 3. Block diagram of medication Information cap

The attachable cap of bottle-typed NFC-based u-medication information management is consisted of NFC module to save and transfer medication information and control device module to manage medication information. The NFC module already has a NFC Read/Write module and a memory module to memory medication information and the device module is consisted of a medication time alarm LED, a Beep and a switch. Medication information management control software is made of I2C Communication, NFC function control software, LED display and alarm control software. The data flow and function is as Figure 3.

### 4 Conclusions

This study has introduced NFC-based smart medication management system for the elderly patients. To enhance convenience and availability of the suggested system, surveys were conducted for pharmacies and elderly patients and reflected the results on the product plan and development. With this product, elderly patients can manage their medication information easily so that they can prevent missing or abusing essential medications. For further research, purchasers' taste and preference on various cap design and production will be considered before deploying on a commercial scale

### 5 References

[1] Report of an aging society analysis, Statistics Korea Website, "[http://kostat.go.kr/portal/korea/kor\\_ki/2/4/index.board?bmode=read&aSeq=198957&pageNo=&rowNum=10&amSeq=&sTarget=&sTxt=](http://kostat.go.kr/portal/korea/kor_ki/2/4/index.board?bmode=read&aSeq=198957&pageNo=&rowNum=10&amSeq=&sTarget=&sTxt=)". (accessed Mar., 10, 2014)

[2] Westbrook J.I., Lo C., Reckmann M.H., Runciman W., Braithwaite J., Day R.O., "The effectiveness of an electronic medication management system to reduce prescribing errors in hospital," Proceedings 18th National Health Informatics Conference, Aug., 2010.

[3] Vitality-GlowCaps Website, "<http://www.vitality.net>". (accessed Mar., 14, 2014)

# The Modelling Design for Arm Strength Training Machine with Biofeedback

Tze-Yee Ho<sup>1</sup>, Yuan-Joan Chen<sup>2</sup>, Mu-Song Chen<sup>3</sup>, Chih-Hao Chiang<sup>1</sup>, Wei-Chang Hung<sup>1</sup>

<sup>1</sup>Dept. of Electrical Engineering, Feng Chia University, Taichung, Taiwan, R.O.C

<sup>2</sup>Dept. of Info. Management, Ling Tung University, Taichung, Taiwan, R.O.C

<sup>3</sup>Dept. of Electrical Engineering, Da-Yeh University, ChangHua, Taiwan, R.O.C

**Abstract** - This paper develops a friendly human interface for an arm strength training machine with biofeedback system to reduce the occurrence of improper operation of exercise machine by real time monitoring. In order to shorten the development schedule duration and reduce the manpower needs, the modelling design of the arm strength training machine with biofeedback is presented for the consideration of system integrity on chip in future work. Finally, a prototype of arm strength training machine with biofeedback system is implemented and demonstrated. The experimental results show the feasibility of the modelling designed system.

**Keywords:** arm strength training machine, biofeedback

## 1 Introduction

A study indicates that chronic stroke patients who gained maximal functional benefits from the biofeedback intervention initially had greater active range of motion at all major upper extremity joints [1]. Consequently, the proper utilization of electromyographic biofeedback can lead to substantial improvements among select chronic stroke patients and can be of considerable functional benefit to others. Therefore, the usage of EMG not only can help the physical therapy but also achieve the more effective rehabilitation [2]. An arm strength training machine (ASTM) based on an embedded microcontroller system that utilizes a PMSM motor drive to simulate the stack of iron weights has better performance than that of the conventional exercise apparatus is presented in [3]. The ASTM with biofeedback system not only can help the physical therapist diagnose the progress of rehabilitation, but also raise the user exercise desire. The modeling design methodology has been widely applied to electronic device as well as electrical equipments for reduction of development schedule duration. Recent years, the highly development of semiconductors has made the system on chip become more easy to realize. Therefore, the modeling design of an arm strength training machine with biofeedback that includes a microcontroller module, protection circuit module, gate drive module, inverter module, speed and current sensor module, communication interface module, and the EMG amplifier module, is presented in this paper. The software programs are written in C language and programmed based on the MPLAB integrated development environment (IDE) tool by Microchip technology incorporate

[4]. Finally, the experimental results show the feasibility and fidelity of the complete designed system.

## 2 The Modelling of ASTM

The modeling of an arm strength machine with EMG sensor based on an embedded microcontroller system is shown in Fig. 1. It consists of a microcontroller module, protection circuit module, gate drive module, inverter module, speed and current sensor module, communication interface module, and the EMG amplifier module. The embedded microcontroller, such as the dsPIC 30F4011, or TI TMS28F335, as well as integration of several peripheral assembling the embedded microcontroller module, is the core controller of the ASTM. The independent power source module provides a 5-volt and four sets of 15-volt voltage to microcontroller and the gate of MOSFETs, respectively. The gate drive module is designed to support PWM signal to the gate of switching MOSFET. The photocoupler TLP250 is used for electrical isolation module between the microcontroller system and the high DC voltage bus voltage as well as the independent power source. The motor currents are sensed through the speed and current detection module. The ACS 712-20 current sensor IC which has the resolution of 100 mV per ampere, is adopted for stator phase current detection.

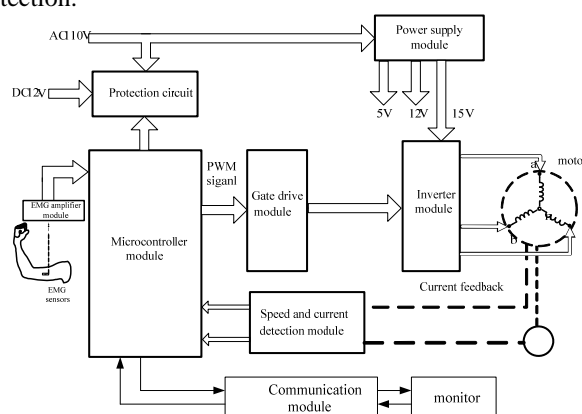


Fig. 1. The system modelling of ASTM with EMG.

The EMG amplifier module consists of the EMG electrodes, the EMG amplification circuits and the bandpass filter. Two electrodes of EMG sensor are attached to the surface skin of an arm. A third electrode is attached to the common point for

voltage reference. The potential difference is generated when the muscle group contracts and then fed into the instrumentation amplifier for amplification. Therefore, system modelling design described above, such as embedded microcontroller module, gate drive module, speed and current sense module and EMG amplifier module bring the design more friendly and thus shorten the development schedule.

### 3 The Experimental Results

The prototype of arm strength training machine is tested under different load conditions in which are fulfilled with the dynamometer. Fig. 2 shows the command setting of desired force 2 kg and desired speed of 10 cm/sec. The waveform of EMG signal is also displayed in the lower part of Fig. 2, so that the user can monitor the muscle action in real time when she/he operates the ASTM. This proves that the EMG signal biofeedback can reflect the muscle contraction and be displayed on the human interface when user operates the ASTM. The current response while the ASTM being manipulated by 2 kg force is shown in Fig. 3. The experiment is repeated by the same cycle of 60 seconds. Observing the waveform of Fig. 9, it can be seen that the motor draws about the 300 mA current to counter the force exerted by the user. This verifies the system design feasibility. The data displayed in Fig. 3 is firstly saved in the memory and then sketched by using the Microsoft EXCEL software. The integrity of complete system design by modelling each function block of the embedded microcontroller module, independent power module, communication interface module, gate drive module, and inverter module, speed and current sensors as well as the EMG amplifier module together has been verified by experimental tests. Finally, the practical system configuration of designed ASTM with EMG sensor is shown in Fig. 4.



Fig. 2. The test for force 2 kg-cm, speed 10 cm/sec and EMG

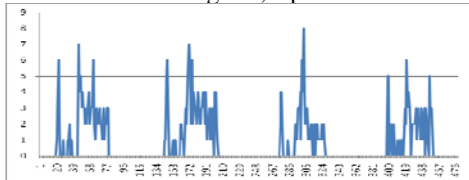


Fig. 3. The current response for 2 kg-cm being applied and released



Fig. 4. The practical system of ASTM with EMG sensors.

### 4 Conclusions

The establishment of interactive communication between ASTM and the user, can make the exercise and rehabilitation therapy become more friendly. This paper develops a friendly human interface for ASTM with EMG system to reduce the occurrence of improper operation of exercise machine by real time monitoring. The experimental results demonstrated by using the human interface display, has verified the system integrity for system hardware and software design. The modelling design of the arm strength training machine with biofeedback is also presented for the consideration of system integrity on chip in future work. The experimental results show the feasibility and fidelity of the complete designed system. Therefore, system modelling design described above, such as embedded microcontroller module, isolation module, current module and EMG module can bring the design more friendly and thus shorten the development schedule.

### 5 Acknowledgment

The authors thank to fund support of the National Science Council at Taiwan under project NSC 102-2221-E-035-047, for part of this work.

### 6 References

- [1] Steven L. Wolf and Stuart A. Binder-Macleod, "Electromyographic Biofeedback Applications to the Hemiplegic Patient," *Physical Therapy, Journal of American Physical Therapy*, vol. 63, no. 9, pp 1393-1403, Sep. ,1983.
- [2] Tony Boucher, Sharon Wang, Elaine Trudelle-Jackson, Sharon Olson, "Effectiveness of surface electromyographic biofeedback-triggered neuromuscular electrical stimulation on knee rehabilitation," *North American Journal of Sports Physical Therapy*, vol. 4, no. 3, pp 100-109, August 2009.
- [3] Tze-Yee Ho, Mu-Song Chen, Hung-Yi Chen, and Po-Hung Chen, "The Design and Implementation of Arm Strength Training Machine", *IEEE EUROCON 2013*, July 1-4, 2013, Zagreb, Croatia.
- [4] dsPIC30F4011/4012 Data Sheet, Microchip Technology Inc., 2005..

# Instruction Cache Pre-Loading Method to Reduce Initial Cold-Misses using SDRAM Burst Transfer

Hyo-Joong Suh<sup>1</sup>

<sup>1</sup> School of Computer Science and Information Engineering,  
The Catholic University of Korea, Bucheon-si, Gyeonggi-do, Korea

**Abstract** - Minimize of response latency of user interactive systems such as a mobile phone is important issue due to seamless user responsiveness. But initial slack of execution is invoked by concentration of cold-misses at an application beginning time. This study focuses on every SDRAM access by cache misses incurs wasteful latencies by RAS-CAS driving sequence of SDRAM, and proposes a pre-loading method to remove the unnecessary access latency by cache image transfer using SDRAM burst transfer mode. Simulation results show the proposed method improves initial slack of execution.

**Keywords:** access latency, cold-miss, instruction cache, image pre-loading, SDRAM

## 1 Introduction

User interactivity is important part of comfortable use of Smartphones, and waiting time of initial execution-to-response is very important due to many users feel which is responsiveness of devices. In terms of execution, from the user invoke till the user-ready, it has several steps; program loading to main memory (DRAM) by OS, processor jumps to the application code, and instruction cache loads instruction codes in the cache.

Since 1993, Synchronous DRAM (SDRAM) was widely accepted in the electronic industry, and the density was quadrupled about 3 years, and the data rates also doubled by adopting double-data-rate method (DDR, DDR2, DDR3). But the access latency of the SDRAM was not improved which compared to the data rate, and its access sequence (RAS-CAS driving method) was not changed also. Due to this reason, the access latency from the RAS driving is around 25~30ns, while the burst data rate is around 2ns [1].

Fig. 1 is a standard DDR2 SDRAM cycles that show the access delay from the RAS active. It shows the initial RAS-CAS active consumes lots of clock cycles. In terms of the access latency, cascaded (burst) transfer of SDRAM is very fast that was comparable to the processor speed.

This RAS-to-DATA access latency makes a big suffer to processors for every cache misses because most of cache misses incurs RAS-to-DATA delay. Furthermore, if user starts the application in the device, most of concentrated cold misses are concentrated at the beginning stage of application, which followed by user interaction stage.

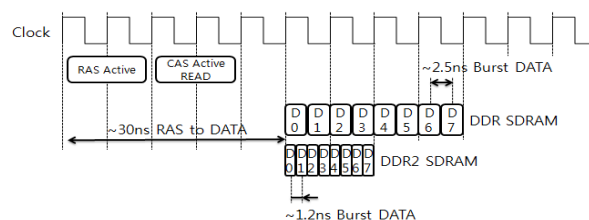


Figure 1. DDR and DDR2 SDRAM access latencies

## 2 Instruction Cache Loading

To remove the initial cold misses on instruction cache, this study proposes a burst SDRAM-to-cache loading method which gains a benefit of latency hiding of burst transfer mode. The burst transfer of SDRAM is very useful to match the processor clock speed, while the burst data are limited to consecutive address only. Generally, the cache data were not consecutive, thus the cache contents from the SDRAM cannot utilize the gain of the burst mode [2].

For utilizing the burst mode, proposed method requires extra cache image that holds aggregated cache data, and tag generation (address) also. From the application starting, operation system loads the executable code and cache image from storage to SDRAM, and this transfer (program loading) was generally by using burst mode of SDRAM and storage also (e.g. synchronous NAND flash).

Clearly, proposed loading sequence requires extra time that proportional to the cache image size, but it still maintains some gains in two terms;

**Clean loading:** General case, the program loading from storage (NAND flash) to SDRAM. Executable and cache image are transferred using burst mode. By the latency hiding effect of burst mode, extra cycles for (small) cache image transfer are relatively tiny compared to the wasted RAS-to-DATA cycles by every cold misses.

**Re-execute:** Special case of mobile OS, in case of mobile OS (e.g. Android), many terminated programs occupies the SDRAM (empty process in Android) until the OS reallocates the memory to the other programs. In this case, the program and the cache image can be remained in the SDRAM. Thus if re-execute the program starts from the cache image transfer using burst mode with the full benefit of the latency hiding.

### 3 Cache Structure Modification

To accomplish of the cache image loading, instruction cache data and tag must be loaded simultaneously. General cache data/tag cannot loaded explicitly, thus the proposed method requires cache modification. Fig. 2 shows some added data path, multiplexor, tag generator, and path selector to loading the cache image.

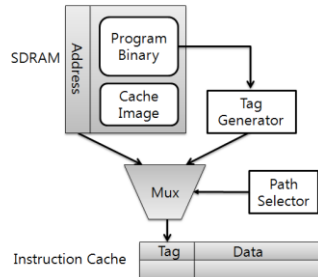


Figure 2. Tag generator and path selector for image loading

When the OS loads a program to the SDRAM, address positioning is run-time event, thus the tag address must be generated run-time also. Hence the tag generator combines the base address and the offset from the cache image.

### 4 Image Creation and OS Support

The cache image can be generated at the compile time if a program has static execution property with some fixed target processor. But it is almost impossible to the real-world processors and programs. Thus the cache image can be created during the program installation steps. Fortunately, state-of-the-art mobile OS (such as the Android and the iOS) supports powerful installer which manipulates the program binary and additional data in the NAND storage. The *cache image creator* can be easily implanted in the installer program. Fig. 3 shows sequence of image creation, OS and H/W part.

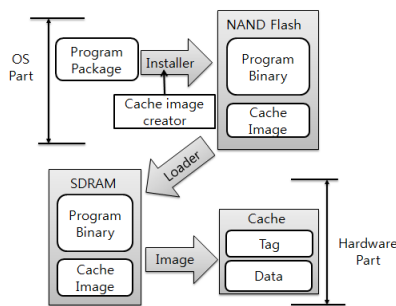


Figure 3. Image creation, OS and hardware part

### 5 Simulation Results

For the evaluation of the proposed method, The SimpleScalar simulator[3] was used to implement the modified cache architecture, pre-loading of the cache image, and the cache image creator. Evaluated programs were selected from MiBench embedded benchmark suite which selected from embedded system applications [4]. Tab. 1 shows the programs and parameters for the evaluation.

Table 1. Selected programs from MiBench

Application	Program	Parameters
Network	Patricia	Small.udp
Automotive	qsort	Input_small.dat
	bitcount	100 iterations
Security	rijndael	input_small.asc, encoding

Fig. 4 shows total number of reduced cache misses by the proposed method. Simulation results exhibits around 30% of cache misses can be removed by the cache image loading, thus the RAS-to-DATA access latency can be reduced proportional to the reduced miss count.

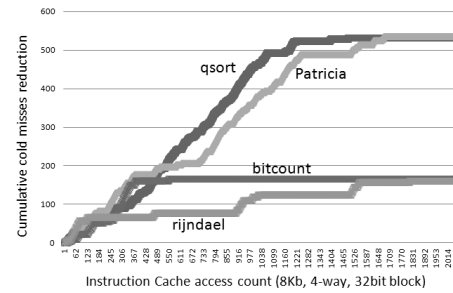


Figure 4. Reduced cold misses

### 6 Conclusion

Generally, bursty cold misses are concentrated at beginning stage of a program execution, and every cache block loading requires wasteful RAS-to-DATA access latency by the SDRAM.

The proposed method solved this issue by the cache image loading using the burst transfer of a SDRAM with modified OS and cache structure.

By the simulation results, proposed method shows about 30% of cache misses on the beginning stage can be removed by the cache image loading without wasteful RAS-to-DATA latencies.

### Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2013R1A1A2057967)

### References

- [1] Samsung Electronics. DDR2 SDRAM Device Operating & Timing Diagram, 2007.
- [2] H.-J. Suh, T. Kim. "Burst Loading Method of Instruction Cache Image for Program Latency Reduction and Energy Saving"; Journal of KIISE: Computing Practices and Letters, Vol.19, No.4, 163-170, 2013.
- [3] D. Burger, T. M. Austin. "The SimpleScalar tool set, version 2.0"; ACM SIGARCH Computer Architecture News, Vol.25, Issue 3, 13-25, 1997.
- [4] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown. "MiBench: A free, commercially representative embedded benchmark suite"; Proc. IEEE Intl. Work. Workload Characterization, 3-14, 2001.

**SESSION**

**LATE BREAKING PAPERS AND POSITION  
PAPERS: EMBEDDED SYSTEMS AND  
APPLICATIONS**

**Chair(s)**

**Prof. Hamid R. Arabnia**





# An Adaptive Cryptographic and Embedded System Design with Hardware Virtualization

Chun-Hsian Huang

Department of Computer Science and Information Engineering,  
National Taitung University, Taiwan

**Abstract**—*This work proposes an adaptive cryptographic and embedded system (ACES) design that can adapt its hardware and software functionalities at runtime to different system requirements. By using the hardware virtualization technique in the ACES design, a fixed set of logic resources can be configured as different hardware modules at runtime to support multiple software applications. Further, by taking the advantages of architectural characteristics of FPGAs, the ACES can support high-performance computing for computing-intensive functions such as cryptographic and image processing functions. Experiments with ubiquitous computing applications have also demonstrated that the ACES can accelerate by up to 26.5x the processing time required by using the software solution. Compared to the traditional embedded system design, the ACES can reduce 29% of slice registers and 33% of slice LUTs required for supporting all the five required hardware functions. Through the advantage of system adaptation, the ACES can also dynamically reduce its power consumption at runtime, according to different environmental conditions.*

**Keywords:** Adaptability, hardware virtualization, partial reconfiguration

## 1. Introduction

As network technology scaling, transferring information and data between different electronic devices becomes more and more convenient. Further, with the increase in user requirements, mobile ubiquitous computing applications enable information processing to be thoroughly integrated into everyday living. In such a ubiquitous computing environment, services and devices can be dynamically adapted to changing environments.

The target applications of this work are mainly used in the ubiquitous computing environments, especially for the field of image processing and cryptographic applications. To support dynamically changing and unpredictable ubiquitous computing applications, adaptability becomes a key requirement in providing high-performance computing and complete data protection on the network in this work. However, in the most existing dynamic adaptive approaches [1]–[5], only software services and applications can be adapted, and hardware devices support the changing software applications passively. This means that hardware functions

cannot be reconfigured at runtime, which also leads to the inefficient use of hardware resources. To be able to not only adapt on-demand functionalities but also provide better system performance, designing an efficient embedded system architecture to meet the dynamic requirements of various environmental situations becomes very important.

This work tries to solve the above problem about system adaptability and performance by proposing an *Adaptive Cryptographic and Embedded System (ACES)* design. Figure 1 gives an example for illustrating the practicability of the proposed ACES. Here, real-time image are captured from the camera and then displayed on the monitor. The filters are used to reduce the effects of the noise in the source images for further image processing applications. The images can also be transferred to a client via the network. To ensure the security of data transfers on the network, all data are first encrypted and then transferred to the client. Based on the effects of noise in the source images and the security requirements of data transfers, the ACES can adapt on-demand its filter and cryptographic functions for providing better *Quality-of-Service (QoS)*.

Figure 1 gives the ideal blueprint to apply the ACES to ubiquitous computing environments. It must solve the following issues related to ubiquitous computing, including 1) what method can make hardware adaptable? 2) how to use hardware resources efficiently? 3) how to support high-performance computing and reduce power consumption at runtime?

To make hardware adaptable, the ACES design integrates the dynamic partial reconfiguration technology from Xilinx [6]. Thus, one part of the FPGA device in the ACES is being reconfigured, while other parts can remain operational without being affected by reconfiguration. This shows that the filter and cryptographic hardware functions in the ACES can be dynamically adapted to different environmental requirements. Further, the partial reconfiguration technology can also be considered as the gate-level hardware virtualization technique, using which multiple applications can access a fixed set of logic resources in a temporally exclusive way. Thus, the utilization of hardware resources can be increased significantly. For system performance, computing-intensive functions such as filter and cryptographic functions are implemented in hardware, so that the ACES can take the advantages of architectural characteristics of FPGAs for

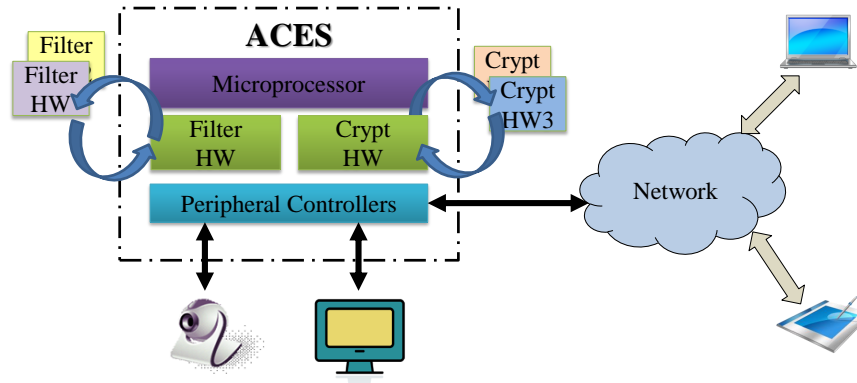


Fig. 1: Application Example

further enhancing system performance. Further, the ACES contains the blank modules to disable the functionality of the partial reconfigurable region in the FPGA. When no requests are received from software applications, the blank module can be used in the ACES to reduce power consumption at runtime.

The rest of this paper is organized as follows. Section 2 introduces the related work. The detailed ACES design is illustrated in Section 3. Section 4 presents our experiments and analyses, and conclusions are given in Section 5.

## 2. Related Work

In a ubiquitous computing environment, computing devices can be adapted to environmental changes for satisfying user requirements. To support the capability of adaptation more efficiently, Efstratiou et al. [1] proposed an architecture that could support adaptive context-aware applications. However, their infrastructure only notified applications about the environmental changes, and application themselves needed to trigger the adaptive mechanism. Instead of the passive application adaptation, Ghim et al. [2] further proposed a reflective approach to dynamic adaptation that could perform adaptation operation triggered by changes in the policy and context. The other existing work [3]–[5] also adopted the software solutions to adapt itself to different system requirements. However, in the above designs [1]–[5], hardware cannot be adapted to different requirements, which also restricts system adaptation and performance.

As for our target cryptographic applications, the corresponding algorithms are usually computation-intensive, hard real-time and non-adaptive to changing network conditions. The algorithms make different tradeoffs between security and complexity. To allow multiple tradeoffs and to adapt to changing network conditions at runtime, a data protective process needs a high-speed and flexible embedded system. T. Wollinger and C. Paar [7] demonstrated the advantages of reconfigurable devices for cryptographic applications in embedded systems, including architecture efficiency, resource

efficiency, throughput, and algorithm agility. Lager et al. [8] also compared a full-software design with a coprocessor design embedded with an FPGA device that could be configured with *Data Encryption Standard* (DES), triple DES, and *Route Coloniale 4* (RC4) hardware cores. Compared to the software solution, the performance of the FPGA-based design was significantly enhanced due to the specific architecture. In other related researches such as [9], [10], they leveraged the advantages of reconfigurable FPGA to further enhance the performance of cryptographic hardware applications. All the above researches [7]–[10] have demonstrated that reconfigurable FPGAs are very suitable for implementing cryptographic applications.

By using the architectural advantages of FPGAs, cryptographic functions of the ACES are implemented in hardware to support high-performance computing. Compared to the software solutions [1]–[5], the ACES design supports the hardware virtualization technique, so that system adaptability and hardware resource utilization can be enhanced significantly. The details of the ACES design will be introduced in Section 3.

## 3. Adaptive Cryptographic and Embedded System Design

To support high-performance and adaptive features, the proposed ACES design is realized on an FPGA device, as shown in Figure 2. The capture controller is responsible for capturing real-time images from the camera, while the display controller is responsible for displaying the processing results on the monitor. Before the processing results are transferred to a client via the network, they are first encrypted through the cryptographic hardware function. To reduce the effects of noise in the source images, the filter function is also integrated into the image processing hardware function.

Besides realizing the image processing function and the cryptographic function as hardware circuits for enhancing

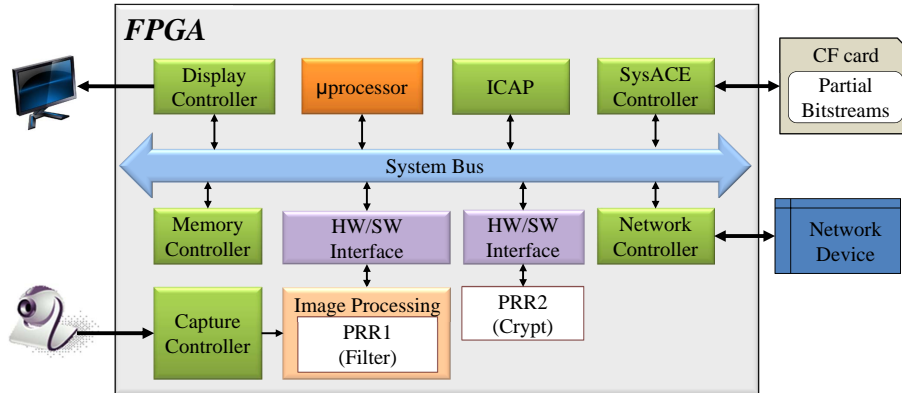


Fig. 2: Adaptive cryptographic and embedded system design

system performance, the ACES further integrates the hardware virtualization technology to increase the utilization of hardware resources. As shown in Figure 2, two *Partially Reconfigurable Regions* (PRRs), namely PRR1 and PRR2, are implemented in the FPGA for configuring the filter function and the cryptographic function, respectively. Thus, the logic resources of each PR region can be reconfigured as different hardware functions, according to system requirements. Therefore, besides the traditional software adaptation, the ACES can also support hardware adaptation.

Based on the partial reconfiguration flow [6], two blank modules are also individually generated to disable the functionalities of PRR1 and PRR2. All the partial bitstreams corresponding to the reconfigurable hardware modules are stored in a CF card, and they are accessed by using the SysACE controller. To support system adaptability, an *Internal Configuration Access Port* (ICAP) [11] controller is also implemented in the ACES for configuring the corresponding partial bitstreams. Through the ICAP controller, the ACES can dynamically adapt its hardware functionalities at runtime, without the user's intervention. To provide efficient hardware/software communication and to support complete system adaptation, a hardware/software communication interface, a virtualizable and hierarchical design, and an adaptation policy are also proposed in the ACES. The details are described in the following sections.

### 3.1 Hardware/Software Communication Interface

To support seamless data transfers between reconfigurable modules and the microprocessor efficiently, a hardware/software interface component based on the *Intellectual Property Interface* (IPIF) is proposed in the ACES, as shown in Figure 3. It contains a bidirectional buffer and a device interrupt controller. Through the hardware/software interface component, the processing results of the reconfigurable hardware module can be stored in the bidirectional buffer sequentially, while the microprocessor can read the

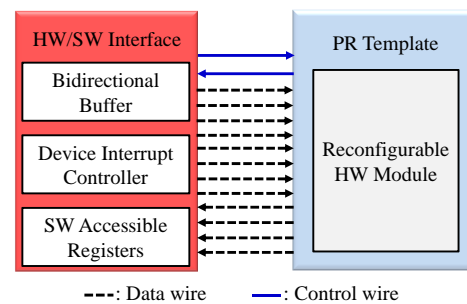


Fig. 3: Hardware/software interface component and PR template

processing results from the bidirectional buffer at the same time. To ensure that all processing results can be read by the microprocessor in real-time, the device interrupt controller is used to notify the microprocessor to read the processing results.

To enhance system scalability, our previously proposed partially reconfigurable template (PR template) [12] is also used in the ACES to ease the integration of user-designed hardware functions with different I/O interfaces. The PR template consists of eight 32-bit input data signals, one 32-bit input control signal, four 32-bit output data signals, and one 32-bit output control signal. To bridge with the interface of the PR template, the proposed hardware/software interface component also contains fourteen software accessible registers for the microprocessor to access the reconfigurable hardware module. Therefore, through the use of the PR template and the hardware/software communication interface component, new user-designed hardware functions can be easily integrated into the ACES.

### 3.2 Virtualizable and Hierarchical Design

To provide a complete hardware virtualization mechanism, besides the support of the hardware/software interface design as described in Section 3.1, the device drivers corresponding to different hardware functions need to be also implemented

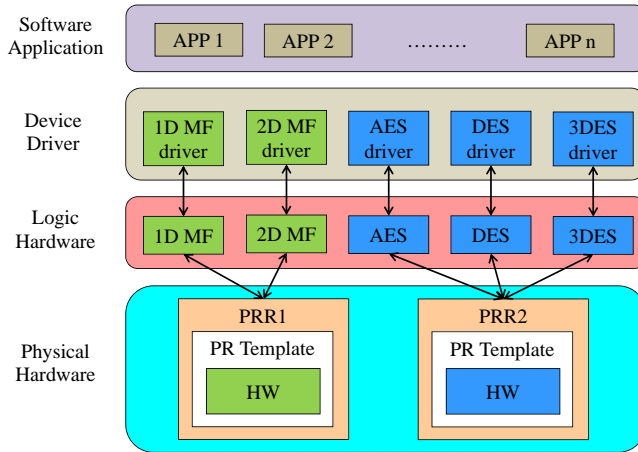


Fig. 4: Virtualizable and hierarchical design

in the ACES. In the design of the device driver, the related *Application Programming Interfaces* (APIs) are provided for software applications to interact with the software accessible registers in the hardware/software interface component. As a result, through the APIs, a software application can easily access the reconfigurable hardware modules.

According to the requirements of our target applications, a software application need to access only one of the filter functions and one of the cryptographic functions at a time instant. When the traditional embedded system design method is used to support multiple functions, all the corresponding hardware designs need be configured in the system at design-time. Although this method enables system performance to be significantly enhanced, because of hardware acceleration; however, system adaptation and the utilization of hardware resources also degrade. To solve the above problem, the ACES is thus realized as a virtualizable and hierarchical design, as shown in Figure 4. Here, besides the software application layer and the device driver layer, the hardware design of the ACES is divided into two layers, including the logic hardware layer and the physical hardware layer.

Through the partial reconfiguration technique, the required filter and cryptographic functions can be dynamically configured in PRR1 and PRR2, respectively. This also indicates that, only two hardware functions are configured in the system at a time instant. The virtualization layer, including the logic hardware layer and the physical hardware layer, abstracts the real hardware characteristics. Furthermore, in the ACES, all the device drivers corresponding to the reconfigurable hardware modules are also provided for software applications. From the viewpoints of software applications, the ACES can support all the hardware functions, even though not all hardware functions are configured in the system at the same time. As a result, through the virtualizable and hierarchical design, system adaptation and the utilization of hardware resources can be further enhanced.

### 3.3 Adaptation Policy

In our current implementation, the system adaptation mechanism is realized as a software program executed on the microprocessor. The ACES design contains two types of adaptable hardware functions, including the filter function and the cryptographic function.

The hardware filters are responsible for reducing the effects of noises in the source images. The quality of source images is classified into different levels according to the *Signal-to-Noise Ratio* (SNR). Different hardware filters are individually associated with their corresponding efficiencies for the reduction of noise in the images. With the increase in the effects of noise, the ACES can dynamically configure the corresponding hardware filter to reduce the effects of noise in the source images. In contrast, the ACES can reconfigure the blank module to improve system performance, when the effects of noise in the source images decrease.

The cryptographic functions are responsible for supporting the service of *Secure Socket Layer* (SSL). When a client makes a request for data transfers, the ACES thus negotiates with it to adopt the same cryptographic function for ensuring the security of data transfers on the network. If the negotiation succeeds, the ACES then configures the requested cryptographic function into the FPGA to adapt itself to different security requirements. Additionally, when no requests for data transfers on the network are received, the ACES can also reconfigure the blank module to improve system performance and reduce power consumption. The related experiments will be discussed in Section 4.

## 4. Experiments and Analyses

To demonstrate the practicability of our proposed method, real applications are implemented in the ACES. In the following sections, we will introduce the experimental setup, the system resource analysis, the power consumption analysis, and the system performance analysis.

### 4.1 Experimental Setup

The ACES design was implemented on the Xilinx ML605 FPGA development board [13] with a Virtex-6 XC6VLX240T FPGA chip. A soft-core MicroBlaze microprocessor [14] at 100 MHz was integrated into the ACES design. Two hardware median filters, including one-dimensional (1D) median filter and two-dimensional (2D) median filter, and three cryptographic functions, including *Advanced Encryption Standard* (AES), DES, and triple DES, were also implemented in the ACES. Two different sized PR regions, namely a small PRR1 and a large PRR2, were implemented for the dynamic configuration of median filter functions and cryptographic hardware functions, respectively. As shown in Figure 5, the small PRR1 configured with 2D median filter and the large PRR2 configured with triple DES are highlighted for displaying the relative locations



Fig. 5: FPGA implementation result

Table 1: Resource Usage

	#Slice registers	#Slice LUTs
1DMF	45	101
2DMF	730	1,795
AES	1,042	3,627
DES	3,955	6,152
3DES	11,457	18,312

1DMF: one-dimensional median filter. 2DMF: two-dimensional median filter.  
3DES: triple DES.

in the implementation result of ACES. Further, a software solution was also implemented and executed on the host computer (Intel Core<sup>TM</sup> i7-3770 3.40GHz, 32GB RAM) for the comparison with the ACES design.

In the experiments, a point target detection function called PMCE [15] was adopted as the main image processing application. Real-time  $320 \times 240$  pixel images were captured from the camera for the application of point target detection, which were then encrypted using the cryptographic functions for data transfers on the network.

## 4.2 Resource Utilization

Compared to a conventional embedded design that requires all the five functions to be implemented and integrated into the system design, the ACES design can support all the five functions by implementing only two PR regions. The resource usages for the five hardware functions, including 1D median filter, 2D median filter, AES, DES, and triple DES, are given in Table 1.

To further compare with the conventional embedded system design, Figure 6 gives a comparison on the numbers of slices registers and those of slice LUTs required for sup-

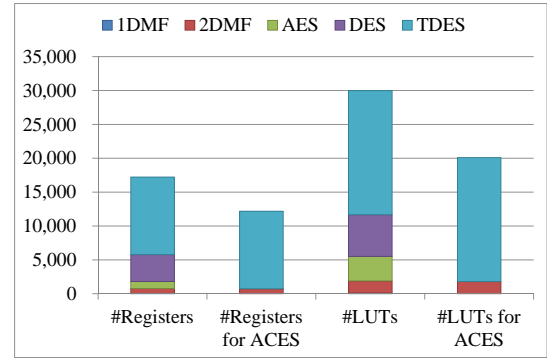


Fig. 6: Resource analysis

Table 2: Power consumption

	Dynamic (W)	Quiescent (W)	Total (W)
1DMF,DES	0.672	4.780	5.452
2DMF,AES	0.765	4.783	5.548
2DMF,3DES	1.020	4.791	5.812
PRR1BM,PRR2BM	0.594	4.778	5.372

1DMF: one-dimensional median filter. 2DMF: two-dimensional median filter.

3DES: triple DES. PRR1BM: blank module for PRR1. PRR2BM: blank module for PRR2.

porting all the five functions. Experimental results show that the ACES needs at most 12,187 slice registers and 20,107 slice LUTs in terms of the Xilinx Virtex-6 XC6VLX240T FPGA. This presents the maximal resource usage by the reconfigurable modules of 2D median filter and triple DES. Compared to the conventional embedded system design, the ACES can reduce 29% of slice registers and 33% of slice LUTs in the Xilinx Virtex-6 XC6VLX240T FPGA. Furthermore, by using the hardware/software interface and the PR template, as described in Section 3.1, new user-designed hardware functions can be easily integrated into the ACES. This shows that, besides having efficient system scalability and adaptation, the ACES can also support a larger number of hardware functions by using the capability of hardware virtualization.

## 4.3 Power Consumption

Besides supporting higher resource utilization as described as Section 4.2, the ACES design can also reduce power consumption. To perform the experiment on power consumption, the Xilinx XPower estimator [16] was used to measure the power consumption of the placed and routed netlists for different combinations of hardware functions in the ACES. Here, our measured results, including the dynamic power, the quiescent power, and the total power, in watt (W) are given in Table 2. Considering the worst case of using maximum power for each of the two PRRs, that is, 2D median filter in PRR1 and triple DES in PRR2, the ACES requires 5.812 watt.

Table 3: Configuration Time

	Function	Time (ms)
PRR 1	PRR1BM	174
	1DMF	192
	2DMF	192
PRR 2	PRR2BM	2,009
	AES	2,227
	DES	2,227
	3DES	2,009

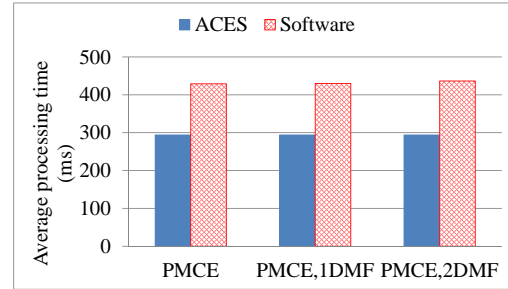
1DMF: one-dimensional median filter. 2DMF: two-dimensional median filter.  
3DES: triple DES. PRR1BM: blank module for PRR1. PRR2BM: blank module for PRR2.

When the effects of noises in the source images decrease and no requests for data transfers on the network are received, the ACES can reconfigure the blank modules for PRR1 and PRR2 to reduce its power consumption. As shown in Table 2, compared to the ACES configured with 2D median filter and triple DES hardware functions, when the corresponding blank modules are configured in the ACES, the total power consumption can be reduced by 0.44 watt. This shows that, through system adaptation, the power consumption of the ACES can be further reduced at runtime, according to different environmental conditions. This feature also benefits the development of low-power embedded systems.

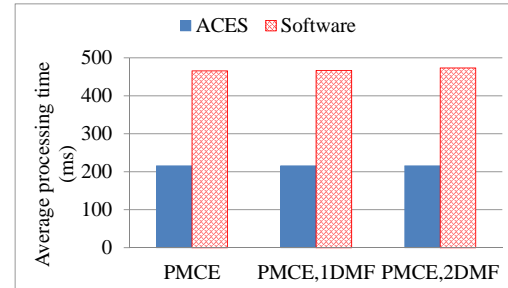
#### 4.4 System Performance

Compared to the conventional embedded system design, for hardware function switching, the ACES contains an additional time overhead, that is, the configuration time. The configuration time for each hardware function is given in Table 3. We can observe that, the configuration times for the hardware functions configured in PRR1 are approximately the same, while that for the hardware functions configured in PRR2 are also approximately the same. This is because the configuration time is directly proportionate to the bitstream size, which in turn is directly proportionate to the size of the PR region. To reduce the reconfiguration time overhead, in this work, the configuration prefetch approach [17] is also applied to the ACES.

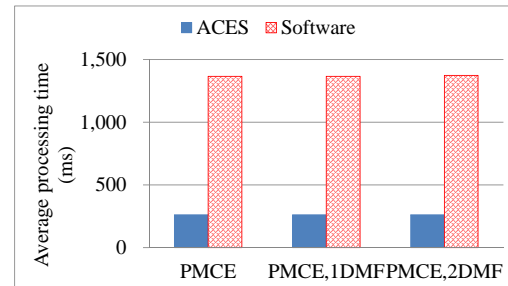
To further analyze system performance, 100 to 1,000 real-time  $320 \times 240$  pixel images were applied to the software solution and the ACES design. Figures 7(a), 7(b), and 7(c) show the average time to process an image frame by using AES, DES, and triple DES, respectively. Here, each cryptographic function was also individually cooperated with three different image processing applications, including the pure PMCE function, the PMCE function with 1D median filter, and the PMCE function with 2D median filter. We can observe that, compared to the software solution, the ACES can efficiently enhance system performance. According to the experimental results, the ACES can accelerate by up to 1.5x, 2.2x, and 5.2x the times required by using the



(a) Average processing time using AES



(b) Average processing time using DES



(c) Average processing time using triple DES

Fig. 7: Average processing time using AES, DES, and triple DES

software solutions, when the pair of AES and the 2D median filter, that of the DES and the 2D median filter, and that of the triple DES and the 2D median filter, respectively, are configured in the FPGA.

For the current ACES implementation, the data transfers between the microprocessor and the cryptographic function are mainly through the software accessible registers. In order to further analyze the execution process for each cryptographic function, the average time per register access for different numbers of registers were measured as illustrated in Figure 8. We can observe that the average time per register access gradually becomes a constant (196 ns). This is because the operation of register access is mainly through the system bus, and thus the measured time also contains the initialization time over the bus. Considering the number of register access for each cryptographic function and the experimental results as shown in Figure 8, the pure

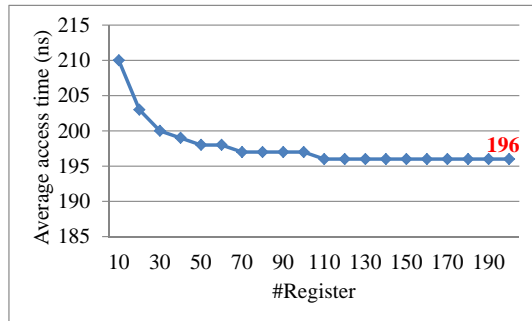


Fig. 8: Average access time per register

execution times required by using the AES, DES, and triple DES can be further calculated. In fact, the pure execution times required by using the AES, DES, and triple DES only take around 7%, 19%, and 29%, respectively, of the measured time as shown in Figures 7(a), 7(b), and 7(c). The experimental results show that, when the time per register access is not considered, the ACES can accelerate by up to 26.5x the time required by using the software solution. This also demonstrates that, the ACES design leverages the architectural features of FPGAs efficiently, so that system performance can be enhanced significantly.

## 5. Conclusion

The proposed adaptive cryptographic and embedded system (ACES) design can not only provide high-performance computing by using the architectural advantages of the FPGA device, but also can adapt its functionalities to different system requirements. Through the hardware virtualization technique in the ACES, system adaptation and the utilization of hardware resources can be further enhanced. Experiments with real applications have also demonstrate that ACES can accelerate by up to 26.5x the processing time required by using the software solution. Further, through the ability of system adaptation, the power consumption of the ACES can also be reduced at runtime, according to different environmental conditions.

## References

- [1] C. Efstratiou, K. Cheverst, N. Davices, and A. Friday, "An architecture for the effective support of adaptive context-aware applications," in *Proceedings of the 2nd International Conference on Mobile Data Management (MDM 2001)*, January 2001, pp. 15–26.
- [2] S.-J. Ghim, Y.-I. Yoon, and J.-W. Choe, "A reflective approach to dynamic adaptation in ubiquitous computing environment," in *Proceedings of the International Conference on Networking Technologies for Broadband and Mobile Networks (ICOIN 2004)*, February 2004, pp. 75–82.
- [3] J.-Z. Sun, "Adaptive determination of data granularity for QoS-constraint data gathering in wireless sensor networks," in *Proceedings of Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing (UIC-ATC)*, June 2009, pp. 401–405.
- [4] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, and G. Papadopoulos, "A development framework and methodology for self-adapting applications in ubiquitous computing environments," *Journal of Systems and Software*, vol. 85, no. 12, pp. 2840–2859, December 2012.
- [5] J. Zhou, E. Gilman, J. Palola, J. Riekkki, M. Ylianttila, and J. Sun, "Context-aware pervasive service composition and its implementation," *Personal Ubiquitous Computing*, vol. 15, no. 3, pp. 291–303, March 2010.
- [6] Xilinx Inc., "Partial Reconfiguration User Guide - UG702," January 2012.
- [7] T. Wollinger and C. Paar, "How secure are FPGAs in cryptographic applications," in *Proceedings of the 13th IEEE International Conference on Field Programmable Logic and Applications (FPL03)*, August 2003, pp. 1–3.
- [8] A. Lagerer, A. Upegui, E. Sanchez, and I. Gonzalez, "Self-reconfigurable pervasive platform for cryptographic application," in *Proceedings of 16th IEEE International Conference on Field Programmable Logic and Applications (FPL06)*, August 2006, pp. 777–780.
- [9] N. Mentens, K. Sakiyama, L. Batina, I. Verbauwhede, and B. Preneel, "FPGA-oriented secure data path design: implementation of a public key coprocessor," in *Proceedings of the 16th IEEE International Conference on Field Programmable Logic and Applications (FPL06)*, August 2006, pp. 133–138.
- [10] R. Laue, O. Kelm, S. Schipp, A. Shoufan, and S. Huss, "Compact AES-based architecture for symmetric encryption, hash function, and random number generation," in *Proceedings of the 17th IEEE International Conference on Field Programmable Logic and Applications (FPL07)*, August 2007, pp. 480–484.
- [11] Xilinx Inc., "LogiCORE IP XPS HWICAP - DS586," June 2011.
- [12] C.-H. Huang and P.-A. Hsiung, "Model-based verification and estimation framework for dynamically partially reconfigurable systems," *IEEE Transactions on Industrial Informatics (TII)*, vol. 7, no. 2, pp. 287–301, May 2011.
- [13] Xilinx Inc., "ML605 Hardware User Guide - UG534," October 2012.
- [14] —, "MicroBlaze Processor Reference Guide, Embedded Development Kit - UG081," January 2012.
- [15] C.-H. Huang, "An FPGA-based point target detection system using morphological clutter elimination," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2013, pp. 2436–2439.
- [16] Xilinx Inc., "XPower Estimator User Guide - UG440 (v13.4)," January 2012.
- [17] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Physically-aware HW-SW partitioning for reconfigurable architectures With partial dynamic reconfiguration," in *Proceedings of the 42nd ACM/IEEE Design Automation Conference (DAC'05)*, Jun. 2005, pp. 335–340.

# Design of a Standalone FORTH Interpreter for the Microchip PIC24F Family

Byron A. Jeff

## *Abstract*—

A complete standalone FORTH kernel is implemented for the PIC24FV16KM202 microcontroller using several advanced techniques and saved flash storage of user programs. This facilitates development using a variety of student devices using limited software on the students device.

*Index Terms*—Embedded Development, Programming Languages, Embedded Systems Education

## I. INTRODUCTION

Clayton State University's Department of Computer Science and Information Technology's Computer Science program has incorporated embedded systems development in assembly language as part of the laboratory experience in Computer Organization, Architecture, and Operating Systems courses. In addition, as one of the leaders in the use of technology in the classroom, Clayton State University policy requires each student to have access to a laptop in the classroom. However, student laptops, as Bring Your Own Device technology can be difficult to support due to the varied platform and operating system choices [5]. The FORTH interpreter described is designed to limit the required software on the student machine to a terminal emulator while providing a high level language environment for embedded systems development. A FORTH implementation requires several design choices including Microcontroller selection, FORTH Virtual Machine Model and Mapping, Threading Model, Primitive Selection, Dictionary Implementation, and Internal control constructs. The following sections describes each of choices.

## II. MICROCONTROLLER SELECTION

The microcontroller selected for the project is the PIC24FV16KM202. This processor was selected because it comes in a 28 pin DIP package, contains an internal oscillator (with PLL) and has 5V power model that simplifies breadboard based development for students. In addition the 2K RAM and 16K flash program memory are sufficient to support a FORTH kernel and RAM based user code development and execution. The CPU model with 16 16-bit registers coupled with indexed addressing modes including auto increment/decrement simplifies mapping to the FORTH VM Model. While the part is a Harvard architecture with separate program and data spaces, program memory can be mapped into data memory which facilitates an execution model from both Flash and RAM. The chip is self flash programmable with a minimum write/erase row

size of 32 instructions. This small row size facilitates incremental flash extension of the FORTH executive. The part has a price point of less than three dollars USD in quantity.

## III. FORTH VIRTUAL MACHINE MODEL AND MAPPING

Several FORTH Virtual Machine models are described in [3]. While the PIC24FV16KM202 register and addressing model can support any of the specified models, the MachineFORTH model was chosen for implementation. The implementation is a 16 bit cell, with A/B scratch registers, and 64 cell data and return stacks. Specific PIC24FV16KM202 registers are used for the instruction pointer, instruction register, top of stack, data and return stack pointers, and scratch registers.

## IV. THREADING MODEL

Standard FORTH high level words are implemented by compiling an address for each word in a definition. Threading is the process of transitioning from the code of one word to the next in a definition. The challenging part of this task is that standard FORTH has several different types of words which each execute a different action including Code/primitive words implemented in native assembly, High level FORTH words, Variables, Constants, Words with user defined actions. Because of these differences each word address must be able to identify the action associated with that word.

The PIC24FV16KM202's Harvard architecture further complicates the model because actual code can only be executed from the flash program memory. However, in order to facilitate the interactive nature of the FORTH interpreter, new code must be compilable to and executable from the chip's data memory. [1] describes the several FORTH threading models including indirect, direct, primitive centric, and hybrid models.

Direct threading fails in the PIC Harvard Architecture environment because native code cannot be executed from data memory. Indirect threading works in the Harvard Architecture environment because only addresses are stored in high level FORTH words. However, this comes with the execution performance penalty because each word execution, regardless of type, requires a second indirect reference to determine how to process the word.

Primitive-centric threading was chosen because it completely separates code areas (primitives) from data areas (high level forth words are all primitive addresses). Therefore, no native code is required anywhere other than primitive words. This fits the Harvard architecture model of the

Department of Computer Science and Information Technology, Clayton State University, 2000 Clayton State Blvd, Morrow GA, 30260, USA, email: ByronJeff@mail.clayton.edu Phone: (678) 466-4411



PIC24FV16KM202 and facilitates execution of high level FORTH words directly from data memory. In addition performance of the threading model is improved over indirect addressing by removing the extra level of indirection required to access primitives. The hybrid indirect execution mechanism for dynamic word interpretation is implemented to augment the primitive-centric threading. This is accomplished by preceding each word's data (starting at the Data Field Address: DFA) by an address for a code routine that can process the word (Code Field Address: CFA). The CFA is used only for dynamic interpretation and compilation of words. Direct execution is performed via primitive centric coding.

## V. PRIMITIVE SELECTION

A FORTH kernel consists of a blend of primitive words, which are coded in native assembly language, and high level FORTH words, which consist of a string of primitives in the primitive centric threading model. Primitive selection is the division of the activities of the kernel between these two types of words. Highly portable FORTH systems start with a very limited number of natively coded primitive words and has a large amount of the kernel implemented as high level FORTH words. Efficient systems implement, or compile the entire system into the native language.

The PIC24FV16KM202 has a rich instruction set with manipulations for bit, byte, word (16 bit), and double word (32 bit) data. Primitive selection consists of the most common arithmetic, logic, shifting, and bit manipulation instructions. For both performance reasons and to exploit the rich Instruction Set Architecture of the chip, virtually all instructions in the above named categories are implemented as primitives. In addition FORTH words for stack manipulation and memory operations, scratch register operations, and I/O operations to the serial port are primitives.

## VI. DICTIONARY IMPLEMENTATION

The FORTH dictionary serves both as the primary conversion system of text into word addresses, and as a repository for the storage of the CFA for each word. Each dictionary entry/header is a fixed structure using a linked list. FORTH headers contain the link to the next word, a hash of the word and the CFA of the word.

In order to implement a fixed structure a 32 bit BUZHASH [6] is used in lieu of storing actual words for each entry. The BUZHASH is a cyclic XOR hash algorithm that encodes each character as a 32 bit random token. The tokens table is generated such that each of the 32 bit columns contains an equal number of 1 and 0 bits, which guarantees a uniform random distribution. The implementation minimizes potential collisions and is a perfect hashing algorithm for all of the words in the FORTH kernel for this project. Each interpreted word is hashed and dictionary lookup consists of matching the hashes. In the event that there is a collision, the first match is used.

## VII. INTERNAL CONTROL CONSTRUCTS

The standard FORTH mechanism of braces with offsets is complete for implementing standard selection and repetitive control structures. But it presents significant challenges in the development phase of a FORTH kernel, which is mostly written in high level FORTH, before the text interpreter and compiler are fully implemented.

Factor [4] employs an alternative mechanism for conditional, repetitive, and functional mapping execution. Nameless code blocks, called quotations, are used as parameters for control structures. In the FORTH kernel, nameless code blocks are not linked into the FORTH directory, which defines the names for FORTH words. Instead quotations are given only an assembly language label, and are included within the context of the FORTH words that utilize them.

The if combinator from Factor used for selection proved cumbersome to implement due to the requirement that quotations for both the true and false quotations are always required. As an alternative the IF, ELSE, and ELSEIF words were implemented separately with each only using a single quotation for execution. The challenge in the implementation is the fact that the ELSE and ELSEIF words are dependent on the state of the flag from the execution of prior IF or ELSEIF words. The IF word transfers the flag to the return stack. The ELSEIF words updates the flag on the return stack. The ELSE and ENDIF words both remove the flag from the return stack. Finally an IFONLY word was implemented indicating that no further flag testing would be done. The flag is not transferred to the return stack at all when the IFONLY word is used.

Quotations are also used to implement looping constructs. The DOWHILE word expects a flag and a quotation on the data stack. The word transfers the quotation address to the return stack so that loops can be nested. The quotation is executed as long as flag on the top of the data stack is true. At the end of the loop execution, the quotation address on the return stack is removed.

## VIII. TEXT INTERPRETATION AND COMPILATION

Text interpretation and compilation use the same execution path: scan the next word, determine its meaning, interpret/compile based on the current state. Words are identified as numbers or dictionary words using recognizers [2]. There is a recognizer for words in the dictionary along with hex, decimal, octal, and binary numbers. Once a word is recognized, the recognizer will either interpret the word or compile it into the next available RAM location. The compiler can be extended in FORTH by the use of immediate words which are interpreted even when the system is in compiling mode.

## IX. INTERNAL FLASH STORAGE

Development would be challenging if newly compiled system words could not be saved. The FORTH kernel implements a CHECKPOINT word, which saves all used RAM and system variables into an area of flash program

memory along with a BUZHASH of the area as a checksum. At power on, the kernel will verify the saved memory area against the saved checksum. If there is a match, the RAM is automatically reloaded with save flash memory. This system facilitates development over multiple power cycles.

## X. CONTRIBUTIONS AND FUTURE WORK

A complete standalone FORTH kernel is implemented for the PIC24FV16KM202 microcontroller using several advanced techniques and saved flash storage of user programs. This facilitates development using a variety of student devices using limited software on the students device. Further work will examine the implementation of turnkey applications, and the extension of the FORTH kernel with the permanent addition of user words into flash memory.

## REFERENCES

- [1] M. Anton Ertl. Threaded code variations and optimizations. In *EuroForth 2001 Conference Proceedings*, pages 49–55, 2001.
- [2] Bernd Paysan. Recognizers. In *28th EuroForth Conference*, pages 108–110, 2012.
- [3] Stephen Pelc. Updating the Forth virtual machine. In *24th EuroForth Conference*, pages 24–30, 2008.
- [4] Sviatoslav Pestov, Daniel Ehrenberg, and Joe Groff. Factor: A dynamic stack-based programming language. In *Proceedings of the 6th Symposium on Dynamic Languages, DLS '10*, pages 43–58, New York, NY, USA, 2010. ACM.
- [5] Ieda M. Santos. Use of students personal mobile devices in the classroom: Overview of key challenges. In Theo Bastiaens and Gary Marks, editors, *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2013*, pages 1585–1590, Las Vegas, NV, United States, October 2013. AACE.
- [6] R. Uzgalis and M. Tong. Hashing myths. Technical report, Department of Computer Science University of Auckland, 1994.

# Design and Implementation of a Microcontroller Based Egg Incubator with Digital Temperature read out.

Anthony Obidiwe<sup>1</sup>, Chukwugoziem Ihekweaba<sup>1</sup>, Patrick Aguodoh<sup>1</sup>.

Computer Engineering Dept. Michael Okpara University of Agriculture, Umudike, Abia State Nigeria.

## Abstract:

Modern day incubators need accurate and precise temperature monitoring for optimal performance and output. The operating temperature range of conventional incubators lies within 92<sup>0</sup>F-102<sup>0</sup>F. Hence, the work presented here involves the design of an intelligent automated incubator system with a digital readout display, which is capable of continuously monitoring and maintaining the operating temperature (37<sup>0</sup>C) using an automatic switching technique. In retrospect, the work takes a panoramic and synoptic view of the history of incubator systems. It finally moves ahead to present a hardware system as well as an attendant software driver derived for a microcontroller based egg incubator system.

## Introduction:

In this present age of information technology, the control and automation of devices, machines and systems are mostly achieved through mechatronic means with emphasis on soft control. This is mostly achieved by the use of programmed microcontrollers. Consequently, this research paper is geared towards the design of a microcontroller based egg incubator with temperature value display. The electronic system is designed using an 89C59 micro controller and real time software written in assembly language. At the input of the system is a linear temperature sensor: an LM35IC. The output of the sensor is fed into an ADC 0804 (Analog to digital converter) that converts the analog signal to an 8-bit parallel digital output. Port 0 and port 1 of the 89C51 micro controller respectively receives the 8-bit parallel data. A seven segment display, common cathode type is interfaced to the micro controller that displays the temperature numerical values between 00<sup>0</sup>C to 99<sup>0</sup>C. Also an interface card realized with the combination of a transition switch and an electromagnetic relay is connected to the micro controller output as an interface to the heating element of the incubator. The entire system is designed in such a way that the sensor obtains the value of the heat generated by the heater. At a record of 37<sup>0</sup>C the heater is automatically switched off, subsequently, the egg hatching operation is initialised. The primary motivation for the work lies in the

fact that the end product is an electronic automated system capable of incubating eggs from the livestock to an optimal condition necessary for hatching. In addition, the work is multidisciplinary and represents a form of synergy between three engineering disciplines namely Computer Engineering, Agricultural Engineering and Industrial Engineering. A product of this nature is necessary in a modern livestock farm. It can also be used in biomedical engineering after a few modifications. A product of this standard can also be deployed to incubate premature animals by veterinary operatives.

## An evolution of temperature control systems and the incubator:

The history of incubator design can be traced back to prehistoric times in line with the agrarian endeavours and exploits of the Egyptian and Chinese dynasties. Egyptians before the time of Moses were able to design and construct hatcheries with a capacity of ninety thousand birds. A few of these hatcheries are still operative, and even as late as the 1950s, were producing almost 90% of all the chicks in Egypt.

The Chinese by the year 1,000 B.C. had also devised two successful methods of hatching eggs which resulted in high yields. The Greeks in pre-historic times also produced hatcheries as described in detail in the works of Aristotle by utilizing rotten manure. It can be said that modern designs of incubators began in the 16<sup>th</sup> century when Cornelis Drebbel of Holland together with his contemporary; J. Kepler around 1624 developed an automatic temperature control system for a furnace. This was motivated by his belief that base metals could be turned to gold by holding them at a precise constant temperature for long periods of time. He also used this temperature regulator in an incubator for hatching chickens. Temperature regulators were studied by J.J. Becher in 1680, and used again in an incubator by the Prince de Conti and R.-A.F. de Réaumur in 1754. The "sentinel register" was developed in America by W. Henry around 1771. He suggested its use in chemical furnaces, in the manufacture of steel and porcelain, as well as in the temperature control of the hospital. It was not until 1777, however, that a temperature regulator suitable for

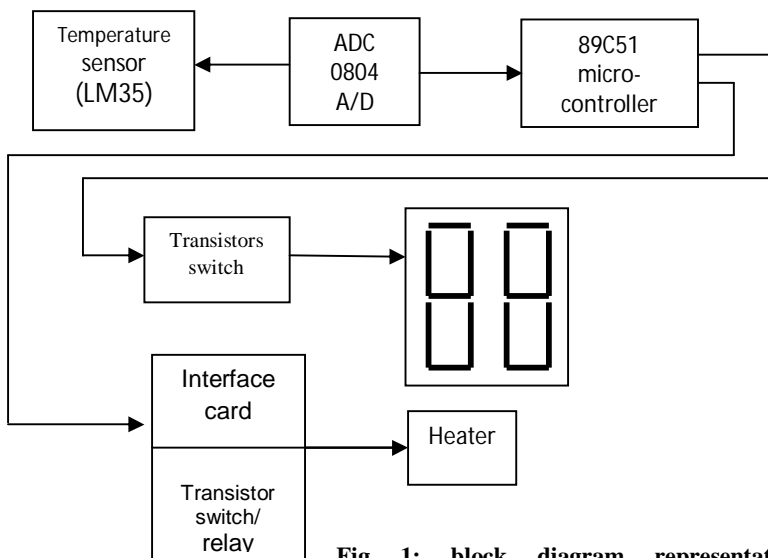
industrial use was developed by Bonnemain, who used it for an incubator. His device was later installed on the furnace of a hot-water heating plant.

The first successful commercial machine was the hot water incubator made by Hearson in 1881. In 1895 Cypher put his 20,000 duck egg model on the market. The first all-electric automated machine did not appear until 1922.

### Methodology:

The micro controller based egg incubator with temperature meter display is designed in six units. The units are connected together to derive the functional hardware. At the input is a temperature sensor LM358; a linear temperature sensor from national semiconductor. This particular unit senses the heat from the incubator chamber. The second unit is the ADC 0804; this unit is the analog to digital converter (A/D) that converts analog signal to digital 8-bit parallel output. This data is fed into port 0 and 1 respectively of the 89C51 microcontroller. The third unit is the 89C51 micro-controller designed to perform the micro-program control. The fourth unit is a transistor static switch realized with two (2) PNP bipolar junction transistors; the BC327. These two transistors switch from the least significant bit (LSB) to the most significant bit (MSB) of the two (2) seven segment displays that display between 00°C to 99°C. The fifth unit is the interface card realized with the combination of a transistor switch and an electromagnetic relay. Here the heater is connected as an external device through a 13-amps socket outlet. The final and the sixth unit is the digital temperature display configured with the two (2) seven segments common cathode type Light Emitting Diodes(LEDs).

Fig 1 below is an illustration of the system and its component units.



**Fig 1: block diagram representation of the microcontroller based egg incubator with digital temperature display.**

### System specification:

1. The Temperature sensor must be a linear temperature sensor, with adjustable current source such as the LM35. Alternatively the LM334 or LM391 from National Semiconductor can be utilised.
2. A quasi-crystal of 10MHz must be hooked to the micro controller to maintain a steady clock frequency for the microcontroller internal clock.
3. A common cathode ssd, RS25 miniton display is chosen as display for the system.
4. A static switch for the ssd through the port 2 and port 3 must be configured using PNP silicon bipolar junction transistor to pave way for conformity to their individual biasing requirements based on the input output (I/O) logic.
5. The electromagnetic relay must exhibit the following parameters: Coil resistance of 400Ω, maximum current rating of 10Amps, and voltage of 12V max.
6. The microcontroller must be fortified with a power up reset at the pin 9, precisely manually operated to reset the system at any time.

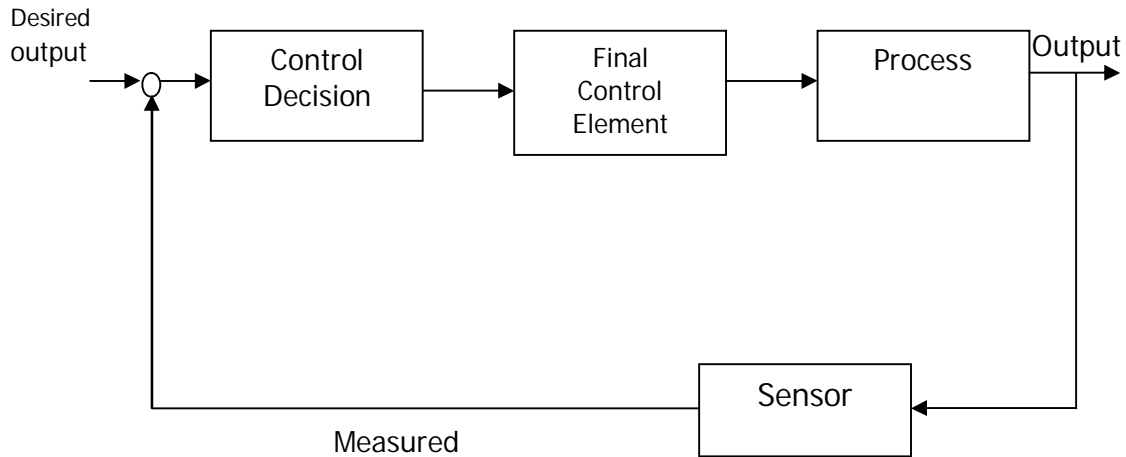
### Basic theory:

The microcontroller based egg incubator is basically a feedback control system. Feedback control is the basic mechanism by which systems, whether mechanical, electrical, or biological, maintain their equilibrium or homeostasis. In the higher life forms, the conditions under which life can continue are quite narrow. A change in body temperature of half a degree is generally a sign of illness. The homeostasis of the body is maintained through the use of feedback control [Wiener 1948]. A primary contribution of C.R. Darwin during the last century was the theory that feedback over long time periods is responsible for the evolution of species. In 1931 V. Volterra explained the balance between two populations of fish in a closed pond using the theory of feedback.

Feedback control may be defined as the use of difference signals, determined by comparing the actual values of system variables to their desired values, as a means of controlling a system. In feedback control, the variable being controlled is measured and compared with a target value. This difference between the actual and desired value is called the error. Feedback control manipulates an input to the system to minimize this error. Fig. 2 below shows an overview of a basic feedback control loop. The

error in the system would be the Output - Desired Output. Feedback control reacts to the system and works to minimize this error. The desired output is generally entered into the system through a user interface.

The output of the system is measured (by a flow meter, thermometer or similar instrument) and the difference is calculated. This difference is used to control the system inputs to reduce the error in the system.



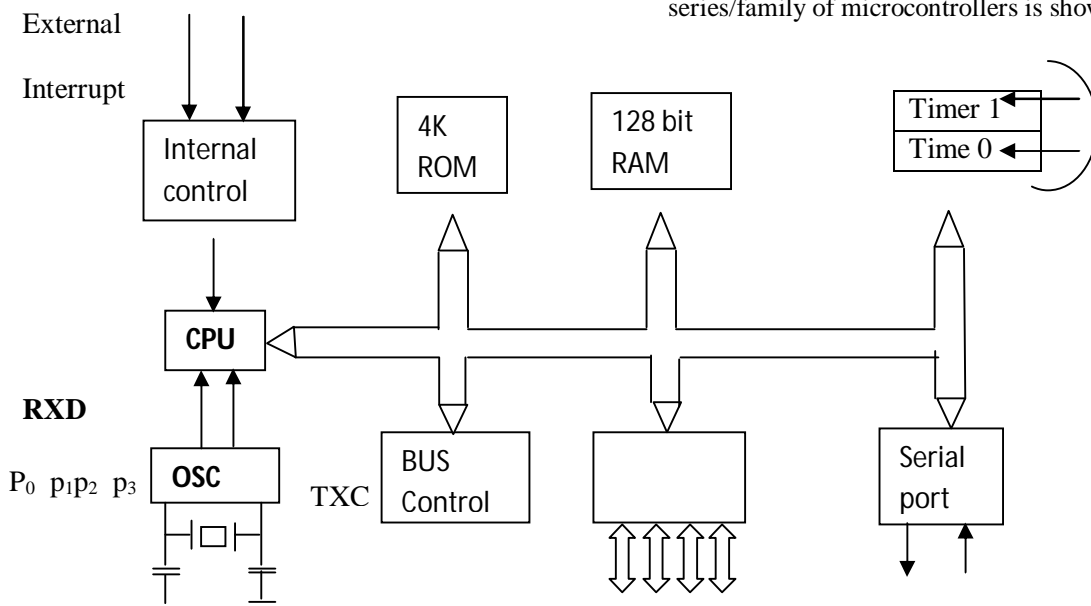
**Fig 2. Feedback control loop**

On the other hand, any control system that does not use feedback information to adjust the process is classified as open loop control. In open loop control, the controller takes in one or several measured variables to generate control actions based on existing equations or models.

Modern control theory is used to model natural control systems, such as those that occur in living organisms or in society, and to design man-made control systems such as those used to control aircraft, automobiles, satellites, robots, and industrial processes.

In this present age of information and communication technology, the emphasis is on soft control, typically involving the use of a programmed microcontroller interfaced with sensors, actuators, electronic devices and electromechanical systems.

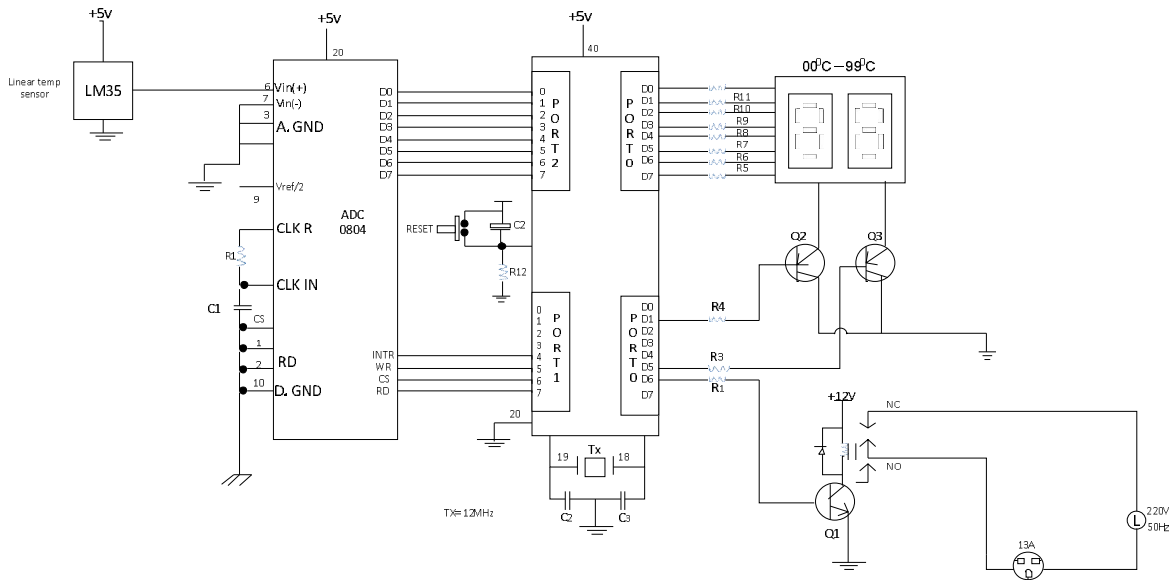
Basically, the microcontroller, is an integrated circuit(IC) containing specialized circuits and functions that are applicable to various designs, especially mechatronic systems. It contains a microprocessor, memory, I/O capabilities, and other on-chip resources. It is basically a microcomputer on a single IC. The basic architectural structure of an 8051 series/family of microcontrollers is shown in fig 3 below:



**Fig 3: Architectural Structure of the 8051 Microcontroller.**

### System hardware development

The system hardware is implemented as shown in the fig 4 below:



**Fig 4 System Hardware Implementation**

### Functional description of the system operation

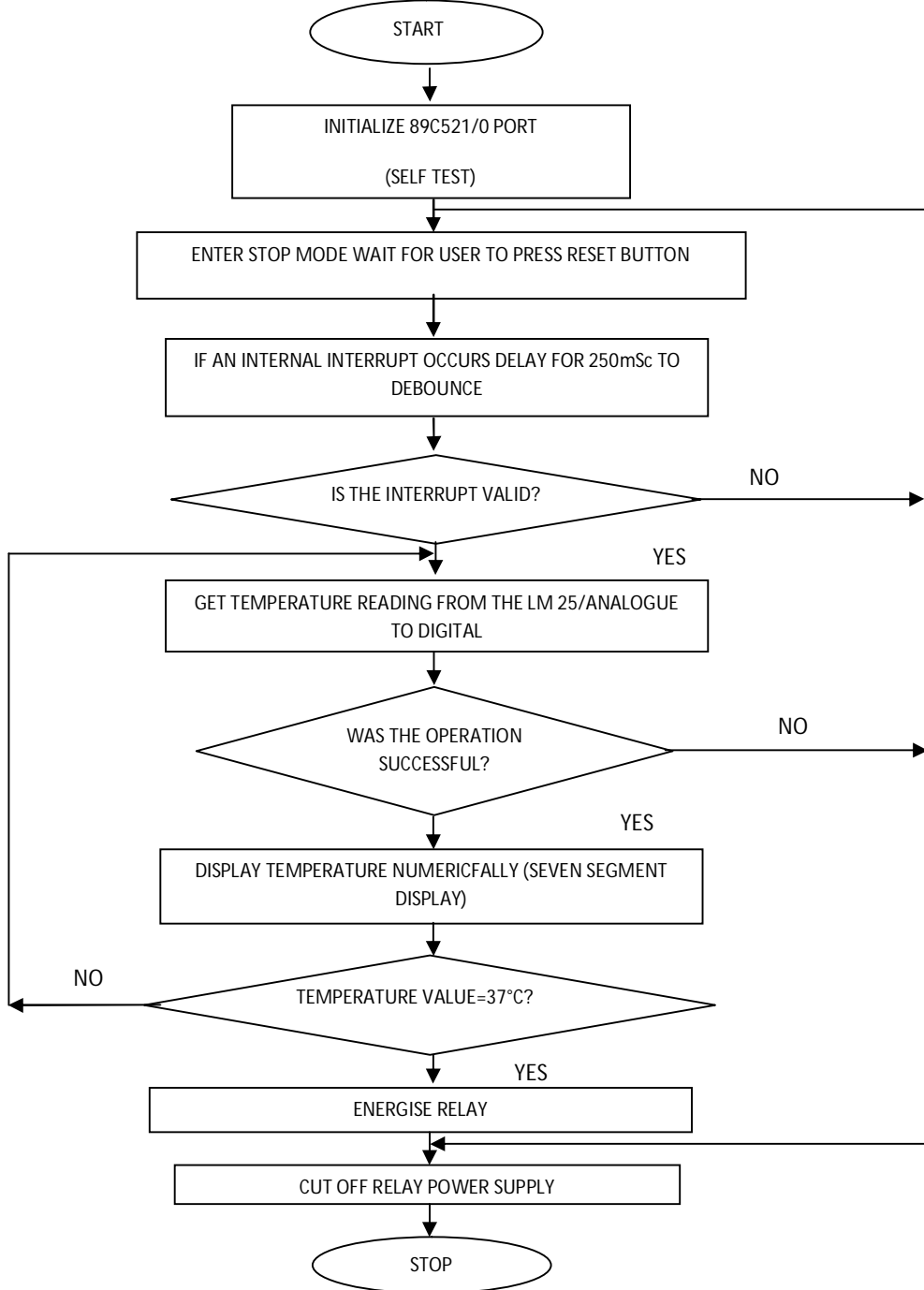
The microcontroller based egg incubator with temperature display is designed with LM35IC linear temperature sensor that serves as the heat detector from the incubators chamber, the output of this sensor runs in millivolts and is purely analog in nature. The ADC0804 Intel semiconductor converts the analog signal from the output of the sensor into 8-bit digital parallel output usually fed into port 0 and port 1 respectively of the 89C51 microcontroller. R1 and C1 are frequency determining components for the ADC. The 89C51 microcontroller is embedded with real time software written in assembly language. The ROM that contains the software reads the data via port0 and port 1 and the read data is written by the RAM into port2 and port 3 in the form of machine language. Pin 9 of the microcontroller contains a manually operated power up reset realized with  $R_{12}/C_2$ , while a 12MHz quartz crystal is connected across pin18 and pin 19 to avoid frequency of the micro controller's internal clock.

$R_2 - R_5$  are current limiting resistors for the seven segment display that displays the temperature of the incubator numerically.  $R_2 - R_4$  are biasing resistors and  $Q_1 - Q_3$ .

$Q_3$  are transistor static switches that link the microcontroller to the 7 segment display. The seven segment display displays from the most significant bit (MSB) to the least significant bit (LSB).  $Q_1$  is only biased through  $R_2$  whenever the system reads the temperature of  $37^{\circ}C$  thereby energizing the relay. Consequently the normally closed contact opens then the heater circuit brakes and hatching starts. The component values are determined as appropriate.

### Software system

In order to enable the controller perform the appropriate and specified control functions, a software that models the system operation is designed and implemented with respect to the flow chart shown in fig 5 below:



**Fig.5 System flow chart**

The source codes were developed in assembly language. The program is assembled using the ASM51 assembler to translate the source code into object code.

#### **Control program operation mode**

As the digital egg incubator is powered on, the control program will display “000 on the seven segment display module. The result button is then operated which allows the system to carry out a self test by checking the state of all pins to know whether their voltage levels are as expected.

If the Vcc (pin 40) and the ground (pin 20) of the microcontroller are in good (normal) order and 18 pins are certified to be the right logic state, the system then displays “000” on the seven segment display showing its readiness to read increase in temperature.

The microcontroller based egg incubator displays numerically the change in temperature, however, when a temperature value of 37°C is attained, the control program causes the relay to switch ON the power and then hatching starts. After which the reset button is operated to reinitialize the system for the next reading.

## Testing

Every component used to actualize this project was properly checked and tested using both analog and digital meter before construction. The following tests were carried out successfully.

### 1. Test For Continuity

This is carried out to ensure if the components were properly soldered to avoid a bridge on the copper strip board since the copper strip board is internally connected horizontally. If there is any bridge, cut can be made on the copper strip board. It was also used to determine if the component were properly connected to each other properly.

### 2. Test For Performance

This test was used to know the performance level of the entire circuitry i.e its ability to hatch egg at the temperature of 37°C.

### 3. Test For Thermal Stability

This was carried out to ascertain if the component used are not overheated when powered which may cause excessive flow of current or bridge in the circuitry. Components such as transistors, capacitors, ICs are properly checked to know their thermal level, if stable or not. By so doing, it gives the difference between practical values and theoretical values which may be caused due to tolerance or losses as a result of soldering.

## Suggestions for further improvement

The automated incubator system designed and implemented in the present work can be improved upon by the integration of a knowledge based network/system into the operation of the controller. Although this may result in a more complex design, the system will become more versatile, adaptive and robust and will consequently be able to meet up more stringent specifications.

## Conclusion

This microcontroller based egg incubator with temperature display has been built, tested and is working to the conformity of the design. It has been shown to be a versatile machine capable of incubating egg at the temperature of 37°C. Such system can be produced locally with a minimum of complex circuitry, resulting in a technological tool of utility and economy suited to the need of modern poultry industries.

## References

- Gibson, G. and Liu, Y., Microcomputers for Engineers and Scientists, Prentice-Hall, Englewood Cliffs, NJ, 1980.
- Herschede, R., “Microcontroller Foundations for Mechatronics Students,” masters thesis, Colorado State University, summer 1999.
- Microchip Technology Inc., www.microchip.com, 2001.
- MicroEngineering Labs, Inc., www.melabs.com, 2001.
- M. Morris Mano: Computer System Architecture, Third Edition, Prentice Hall, New Jersey 1976.
- McGraw-Hill Encyclopedia of Science And Technology, vol. 1, 1997.
- Olsen, H. Second Generation I.C. Voltage Regulator, Ham Radio Magazine, 1973.
- Horowitz, P. and Hill, W., The Art of Electronics, 2nd Edition, Cambridge University Press, New York, 1989.
- Predko, M. Digital Electronics Demystified, McGraw-Hill (2004).
- Ronald, J. Tocci, Neal, S. Widmer., Ninth Edition. Digital System: Principles and Applications, Dorling Kindersley India Pvt Ltd, (2008).
- Rosch, W. L. Hardware Bible Premier Edition, Sama. Macmillan Computer Publishing. 2000.



Shadda, R.S. A Text Book of Applied Electronics, 2<sup>nd</sup> Edition. S. Chad and Company Ltd, New Delhi, 1996.

V.K Mehta, Rohit Mehta, S Chand.Principles of Electronics, Eleventh Edition, S Chand and Company, New Delhi 1980 (reprinted 2010)

History of incubators. [www.pleysierincubators.com](http://www.pleysierincubators.com). 2009.

K. Ogata. Modern Control Engineering, third edition.Printice Hall Upper Saddle River New Jersey.1997.

# Hardware/software co-design of particle filter in grid based Fast-SLAM algorithm

B.G. Sileshi<sup>1</sup>, C.Ferrer<sup>2</sup>, J.Oliver<sup>3</sup>

<sup>1,3</sup>Departament de Microelectrònica i Sistemes Electrònics, Universitat Autònoma de Barcelona  
Bellaterra (Barcelona), Spain

<sup>2</sup>Institut de Microelectrònica de Barcelona (CNM-CSIC)  
Bellaterra (Barcelona), Spain

**Abstract**-A hardware/software co-design based on system on a chip method for particle filter in a grid based Fast-SLAM algorithm is presented in this work. By giving more emphasis on those steps of the algorithm that requires intensive computations, hardware blocks are design in order to speed up the computational time and interfaced with a central Microblaze soft core processing core. The proposed hardware/software resulted in an improvement in the overall execution time of the algorithm.

**Keywords:** Particle filters, Hardware/software co-design, computational complexity.

**Conference Name:** ESA

## 1. Introduction

Particle filters (PFs) are among the estimation frameworks that offer superior flexibility on addressing non-linear and non Gaussian estimation problems [1]. As a result they have been applied to a wide variety of real world problems such as navigation and positioning [2-6], tracking [7-8] and robotics[9]. Their flexibility comes from their efficient representation of a wide range of probability densities by sets of points (particles). Such representational power of particle filters, however, comes at the cost of higher computational complexity that has so far limited their applications in different types of real time problems.

To achieve real time performance in the application of particle filters, hardware or hardware/software based implementation is required to accelerate the computational time. In the literature several studies tackles the real time constrains of the particle filter through hardware implementations. In this sense, FPGA hardware platforms are considered to be one of the choices for such implementations of the particle filter due to the possibility to introduce massive parallelization and their low power consumption properties which is critical for many application such as navigation.

Many studies have been proposed for the hardware implementations of particle filtering on the FPGA platforms. The authors in [10] studied the implementation of particle filter on an FPGA for a mobile robot localization problem, where adapting the size of the particle set is used in reducing

the run-time computation complexity of the algorithm. In [11] the FPGA hardware implementation of particle filter for location estimation is presented and compared with the software solution running on an ARM7-based microcontroller. A recent study in [12] presented a heterogeneous reconfigurable system consisting of an FPGA and CPU for a simultaneous mobile robot localization and people tracking application. In this study they propose a method to adapt the number of particles dynamically and utilize the run-time re-configurability of the FPGA for reduced power and energy consumption. From their study up 7.39 times speed up is achieved compared to the Intel Xeon X5650 CPU software implementation. The authors in [13] presented a hardware/software co-design approach based on system on a chip technique on a NIOS II processor to calculate the weight for each particle and a hardware implementation to update the particles. They claim that their proposed method significantly improved the efficiency and it also provides flexibility in design for various applications due to the software implementation of the importance weight step.

The study in [14] presented the hardware architecture for an FPGA implementation of the sampling importance resampling (SIR) particle filter. The hardware architecture is simulated with Modelsim and the real time performance of the hardware architecture is also evaluated with use of UART for the input sensor data. In [15] a System-on-Chip architecture involving both hardware and software components for a class of particle filters is presented. In this work parameterized framework is used to enable the reuse of the architecture with minimal re-design effort for a wide range of particle filtering applications. In reference [16] three different hardware architectures are proposed and suggested the use of a piecewise linear function instead of the classical exponential function to in the decrease the complexity of the hardware architecture in the weighting step.

With the objective in solving the high computational requirements of the particle filter, this work presented a hardware/software co-design approach for the implementation of the particle filter algorithm with the following specific contributions ;

- As the underlying computations in particle filtering vary from one application to the other, a generic approach is presented for the analysis, design and speedup of the particle

filter computational steps. The same approach can be used in developing other application specific particle filtering implementations.

- An embedded hardware/software implementation of the particle filter in grid-based Fast SLAM algorithm is presented.

- Fast hardware Gaussian random number generator design and its application in particle filter for grid based Fast SLAM algorithm is presented.

- The design and implementation of custom CORDIC hardware module for the evaluation of the complex mathematical operations involved in each step of the particle filter algorithm is presented.

The organization of the rest of the paper is as follows: Section 2 introduces briefly the SIRF particle filter and its application in specific to SLAM algorithm called Grid based Fast SLAM. Section 3 is a discussion on the identification of the computational bottlenecks of the particle filter algorithm and the partitioning of the different steps for hardware and software implementations. Section 4 is dedicated to the discussion on the hardware architecture design of the hardware partitions. Section 5 is a discussion on the proposed hardware/software architecture of the particle filter followed by section 6 with explanations on the implantation results. Finally section 7 concludes the paper.

## 2. Particle filter and particle filter based SLAM algorithms

Simultaneous localization and mapping (SLAM) is the problem of localizing and building a map of a given environment simultaneously. It is usually described with a joint posterior probability density distribution (equation 1) of the map ( $m$ ) and the robot states ( $x_t$ ) at time  $t$  given the observations ( $z_{0:t}$ ) and control inputs ( $u_{0:t}$ ) up to and including time  $t$  together with the initial state of the robot ( $x_0$ ).

$$p(x_t, m | z_{0:t}, u_{0:t}, x_0) \quad (1)$$

For the computation of the conditional probability given by equation 1 for all times  $t$ , a recursive solution based on the Bayes Theorem is used in SLAM by starting with an estimate for the distribution  $p(x_{t-1}, m | z_{0:t-1}, u_{0:t-1})$  at time  $t-1$  and following a control  $u_t$  and observation  $z_t$ . Therefore, the SLAM algorithm is implemented with the two standard recursive time update and measurement update steps. Where the recursion is a function of the robot motion model  $p(x_t | x_{t-1}, u_t)$  and an observation model  $p(z_t | x_t, m)$ .

*Time Update:*

$$p(x_t, m | z_{0:t-1}, u_{0:t-1}, x_0) = \int p(x_t | x_{t-1}, u_t) p(x_{t-1}, m | z_{0:t-1}, u_{0:t-1}, x_0) dx_{t-1} \quad (2)$$

*Measurements Update*

$$p(x_t, m | z_{0:t}, u_{0:t}, x_0) = \frac{p(z_t | x_t, m) p(x_t, m | z_{0:t-1}, u_{0:t-1}, x_0)}{p(z_t | z_{t-1}, u_{0:t})} \quad (3)$$

The solution to the probabilistic SLAM problem involves an efficient and consistent computation of the prior and posterior distributions (prediction and update equations). The different approach that has been used for performing such computation includes the Kalman filters, the particle filters, maximum a posteriori estimator [17].

Among the very few methods that use the particle filter for the whole SLAM problem is the grid-based Fast SLAM [17] algorithm, which is used in this work. As the particle filter is the core of this algorithm and the main focus of this work brief discussion is provided as follows.

In generic Sequential Importance Sampling (SIS) particle filtering method a probability density function is approximated by a discrete set of particles or samples with their associated weights  $\{x_t^i, w_t^i\}_{i=1}^N$ . Where,  $i$  and  $N$  denotes the index of particle and total number of particles respectively.  $x_t^i \in R^{d_x}$  and  $w_t^i$  denotes the state of the particles and their weights respectively and  $d_x$  is the dimension of the state. One of the most widely adopted variant of the SIS particle filtering is the Sampling Importance Resampling particle filter (SIRF) where it recursively propagation of the particle set  $S_t = \{x_t^i, w_t^i\}_{i=1}^N$  at time  $t$ , from the set  $S_{t-1}$  at the previous time  $t-1$  through sampling, importance weight and resampling steps.

*Sampling*

In this step new particles are drawn from the prior density  $p(x_t^i | x_{t-1}^i)$  which is defined by the system equation  $x_t = f_t(x_{t-1}, v_{t-1})$ . Where  $f_t$  is possibly non-linear function of the state  $x_{t-1}$  and process noise  $v_{t-1}$  at time  $t-1$ .

*Importance Weight*

In this step a weight is assigned to each particle according to the measurement  $z_t$ .

$$w_t^i \propto (w_{t-1}^i) p(z_t | x_t^i) \quad (5)$$

where,  $p(z_t | x_t^i)$  is the measurement likelihood.

Resampling

While the sampling and Importance weight steps are performed at every time step  $t$  when new measurements are available, particles may become degenerate (increase in variance of the particles) after few iterations, the drawback of SIS particle filtering. The resampling step overcomes this problem by replicating particles with large weights and discarding those with small weights.

In the resampling step of the particle filter, different resampling algorithms [18] can be used as they are non application specific algorithms. The specific resampling method adopted in this study is Independent Metropolis Hasting (IMHA) resampling algorithm. This specific resampling algorithm has the lowest computational complexity compared to most conventional resampling schemes like systematic resampling [19]. As a result of such interesting characteristics it is the preferred choice in this work.

### 3. Computational bottlenecks identification and hardware/software partitioning

In the speedup of the overall computation in particle filters, a preliminary study in the identification of the critical bottlenecks in the algorithm is necessary for the design of hardware modules to accelerate those computational bottleneck steps. Our study is based on the particle filter in a grid based Fast SLAM algorithm discussed in section 2. In this section detailed study is undertaken for the identification of the computational bottlenecks among each step of the algorithm for an embedded implementation on FPGA platform.

For each step of the particle filter algorithm in the grid based Fast SLAM algorithm, profiling is done using a hardware timer. The summary of the results obtained from such study is given in Table I. As per the profiling results shown in Table I, the computation of sine and cosine functions account for most of the execution time in the sampling and importance weight steps of the particle filter algorithm. Furthermore, the generation of Gaussian and uniform random numbers accounts to most of the execution time in the sampling and resampling steps of the particle filter respectively. The computation of an exponential function also contributes in the computational complexity in the importance weight step of the algorithm for the evaluation of each particle weight.

The profiling information of the particle filter given in Table I can be used in the hardware/software partitioning i.e. which parts of the algorithm should be implemented in hardware and which ones can be kept in software (run on the FPGA's embedded processor). It is clear from Table I that, the sine and cosine functions and random number generation

TABLE I  
IDENTIFICATION IN COMPUTATIONAL BOTTLENECKS IN THE SAMPLING, IMPORTANCE AND RESAMPLING STEPS

Function	Execution time profiling		
	Sampling	Weighting	Resampling (IMHA) with 100 particles
Sin/cos	45.91%	75.34%	-
Gaussian random number	53.62%	-	-
Uniform random number	-	-	60.71%
Exponential	-	7.65%	-

are the critical bottlenecks of the grid based Fast SLAM algorithm and required to be implemented in hardware.

The hardware implementation for the sine, cosine and exponential functions is based on the **CO**ordinate **R**otation **D**igital Computer (CORDIC) algorithm [20]. The uniform and Gaussian random number generation is based on the Tausworthe[21] and Ziggurat[22] algorithms respectively. The details of these algorithms are provided below and their hardware designs are given in section 4.

#### A. CORDIC Algorithm

CORDIC is an iterative algorithm that requires simple shift and addition operations, for hardware realization of basic elementary functions. It is based on the rotation of a vector in a Cartesian coordinate system and the evaluation of the length and angle of the vector. It can operate in one of three configurations: circular, linear, or hyperbolic. Within each of these configurations the algorithm functions in one of two modes rotation or vectoring.

$x_{i+1} = x_i - \mu d_i y_i 2^{-i}$ $y_{i+1} = y_i + d_i x_i 2^{-i}$ $z_{i+1} = z_i - d_i e_i$		
<b>Circular</b>	<b>Linear</b>	<b>Hyperbolic</b>
$\mu = 1$	$\mu = 0$	$\mu = -1$
$e_i = \tan^{-1} 2^{-i}$	$e_i = 2^{-i}$	$e_i = \tanh^{-1} 2^{-i}$
Rotation mode: $d_i = \text{sign}(z_i)$		
Vectoring mode: $d_i = -\text{sign}(y_i)$		

Figure 1 The unified CORDIC algorithm [23,25].

Using the unified CORDIC iteration equation in figure 1, wide range of functions can be evaluated. However the discussion here is focused to the evaluation of the sine, cosine, exponential and logarithmic functions (both required in the Ziggurat algorithm).

For the evaluation of exponential and logarithmic functions, the CORDIC is required to operate in hyperbolic configuration ( $\mu = -1$ ) with rotation ( $d_i = \text{sign}(z_i)$ ) and vectoring ( $d_i = -\text{sign}(y_i)$ ) modes respectively. For the evaluation of sine and cosine functions it is required to operate in Circular ( $\mu = 1$ ) rotation mode ( $d_i = \text{sign}(z_i)$ ).

After  $i$  iterations, the unified CORDIC equations given in figure 1 converge to the following set of equations.

$$\begin{aligned} x &= K(xf_1 z - yf_2 z) & (6) \\ y &= K(yf_1 z - xf_2 z) & (7) \\ z &= 0 & (8) \end{aligned}$$

where, in circular-rotation mode of configuration the functions  $f_1$  and  $f_2$  correspond to the sine() and cosine() respectively and the constant  $K = 1.646$ . For hyperbolic-rotation configuration  $f_1$  and  $f_2$  correspond sinh() and cosh() functions and the hyperbolic constant  $K = 0.828159$ .

The evaluation of the sine and cosine functions are realized by setting  $x = 1/K, y = 0$  and  $z$  as the input argument to the CORDIC algorithm. In case of exponential function evaluation is obtained by setting  $x = 1/K, y = 0$  and  $z$  as the input argument and applying the property:

$$\exp(z) = \sinh(z) + \cosh(z) \quad (9)$$

For the evaluation of natural logarithmic function, the CORDIC has to be configured in hyperbolic-vectoring mode and setting  $x = 1, z = 0$  and  $y$  as an input argument, the algorithm converges to:

$$x = K' \sqrt{x^2 - y^2} \quad (10)$$

$$y = 0 \quad (11)$$

$$z = z + \tanh^{-1}\left(\frac{y}{x}\right) \quad (12)$$

And the evaluation of natural logarithm function is obtained indirectly by:

$$\ln w = 2 \tanh^{-1}\left(\frac{y}{x}\right) \quad (13)$$

Where,  $x = w + 1$  and  $y = w - 1$

As a CORDIC algorithm works for a limited range of the input arguments for the evaluation of the elementary functions, it is required to extend the range of the inputs for each mode of operation by applying proper pre-scaling identities. This is achieved by dividing the original input arguments to the CORDIC algorithm by a constant to obtain a quotient  $Q$  and

TABLE II  
PRE-SCALING IDENTITIES FOR FUNCTION EVALUATIONS [23-24]

Identity	Domain
$\sin\left(Q\frac{\pi}{2} + D\right) = \begin{cases} \sin D & \text{if } Q \bmod 4 = 0 \\ \cos D & \text{if } Q \bmod 4 = 1 \\ -\sin D & \text{if } Q \bmod 4 = 2 \\ -\cos D & \text{if } Q \bmod 4 = 3 \end{cases}$	$ D  < \frac{\pi}{2}$
$\cos\left(Q\frac{\pi}{2} + D\right) = \begin{cases} \cos D & \text{if } Q \bmod 4 = 0 \\ -\sin D & \text{if } Q \bmod 4 = 1 \\ -\cos D & \text{if } Q \bmod 4 = 2 \\ \sin D & \text{if } Q \bmod 4 = 3 \end{cases}$	$ D  < \frac{\pi}{2}$
$\exp(Q \log_e 2 + D) = 2^Q (\cosh D + \sinh D)$	$ D  < \log_e 2$
$\log_e(M2^M) = \log_e(M) + E \log_e 2$	$0.5 \leq M < 1.0$

remainder  $D$  [23]. In the case of the sine and cosine functions the constant value corresponds to  $\frac{\pi}{2}$  and  $\log_e 2$  for the exponential and logarithmic functions. The pre-scaling identities for all the required functions are given in Table II.

### B. Ziggurat algorithm

For the generation of the normal random numbers required in the sampling step of the particle filter a Ziggurat algorithm is used in this work. In the Ziggurat method, normally distributed random variates are generated considering set of points  $C = \{(x, y)\}$  under the curve of a probability density function  $y = f(x)$  and a subset the points  $Z$  i.e.  $Z \subset C$  by uniformly taking the random points  $(x, y)$  until  $(x, y) \in C$ .

The function  $f(x)$  is divided in to  $n$  rectangular blocks  $R_i$ , where  $i = 0, 1, 2, \dots, (n - 1)$ , of equal area  $v$  except the bottom block which consists of a rectangular block joined with the rest of the density starting from a point  $r$ . The rectangular blocks extend horizontally from  $x = 0$  to  $x_i$  and vertically from  $f(x_i)$  to  $f(x_{i-1})$ .

- Given the set  $\{x_i\}_{i=0}^{n-1}$  generate normal random number  $x$ .
- 1. Choose an index  $0 \leq i \leq n - 1$  at random with uniform probability  $1/n$
- 2. Draw a random number  $U_0$  from a uniform distribution  $U(0,1)$  and let  $x = U_0 x_i$ .
- 3. **REPEAT**
- 4. **IF**  $i \geq 1$  and  $x < x_{i-1}$  **RETURN**  $x$ . } *Rectangular*
- 5. **ELSEIF**  $i = 0$  (Generate an  $x$  from the tail  $x > x_n$ )
- 6. **DO**
- 7. Generate i.i.d. uniform (0, 1) variates  $u_0$  and  $u_1$
- 8.  $x \leftarrow -\ln(u_0)/r, y \leftarrow -\ln(u_1)$
- 9. **WHILE**  $(y + y) < x^2$
- 10. **RETURN**  $x > 0? (r + x) : -(r + x)$  } *Tail*
- 11. **ELSE**
- 12. Draw a random number  $U_1$  from the uniform distribution
- 13. **If**  $U_1 |f(x_{i-1}) - f(x_i)| < f(x) - f(x_i)$  **RETURN**  $x$ . } *wedge*
- 14. **UNTIL FALSE**

Figure 2 Ziggurat Gaussian random number generation algorithm.

As can be seen from the pseudo code of the Ziggurat algorithm given in figure 2, a table of  $x_i$  points and their corresponding function values  $f_i$  are required for the algorithm. Normally, the number of the rectangular blocks  $n$  is chosen as a power of 2 (64,128,256). For  $n = 128$  a value of  $r = 3.442619855899$  is used is to determined the  $x_i$  points that are required in the hardware implementation of the algorithm [21].

### 4. Hardware designs

In this section the hardware design of the CORDIC and Ziggurat algorithms is presented.

The implementation of the CORDIC module comprises of three hardware computational blocks; CORDIC-core, pre-scaling and post-scaling. The CORDIC-core implements the unified equations given in figure 1. The pre-scaling and post-scaling modules implement the equations given in table II for extending the range of the input arguments for the different functions.

To avoid the need for using multiple instances of the CORDIC module for the evaluation of different functions, a run-time configuration to the CORDIC module is provided through its *config* port. With this port the CORDIC module can be configured to operate any one of the functions as per the requirement during run-time. This helps to minimize the extra resource requirement if many instances of the CORDIC module is used for the evaluation of every function. The CORDIC-core in the proposed architecture of the CORDIC module, implements a serial CORDIC with an accuracy of 24 bits. Figure 3 shows the input/output interfaces to the CORDIC module. Where it has two possible inputs  $U_0$  and  $U_1$ , and a two bit configuration input port *config* to choose a specific function for evaluation. Depending on the configuration bits on the *config* input interface to the CORDIC module, one of the functions are evaluated and the result is provided at the output interface. Even though not shown in figure 3, there are also other control signals to the input and output interface of the CORDIC module for its correct functionality.

The hardware architecture of the random number generator module is given in figure 4 and it comprises of the Tausworthe URNG, CORDIC and Ziggurat NRNG sub-modules. The Tausworthe URNG module is responsible for generation of two uniform random numbers ( $U_0$  and  $U_1$ ) that

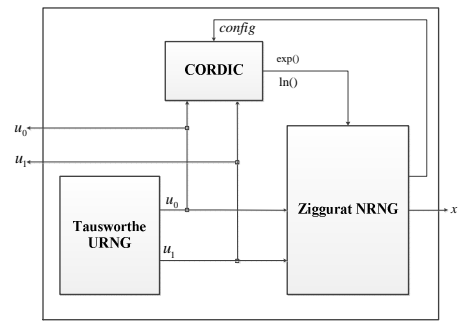


Figure 4 Top level architecture of uniform and Gaussian random number generator

are used by the Ziggurat module and in the resampling step of the particle filter. The CORDIC sub-module implements the architecture given in figure 3.

The hardware design of the Ziggurat sub-module shown in figure 5 follows the description of the Ziggurat algorithm given in figure 2, where it composed of individual hardware blocks to generate the normal random number from the rectangular, wedge and tail region of the distribution (figure 2 ). All the three modules (wedge, rectangular and tail) work independently of one another. For the generation of random numbers from the wedge region (lines 8-9) the evaluation of an exponential function is required and in the case of the tail region (line 6) it is required to compute the natural logarithm function. For the computation of these functions a CORDIC algorithm is used.

As the Ziggurat algorithm requires simultaneous access to the coefficients  $x_i$ 's and  $f_i$ 's, it's important to provide these values in parallel manner to speed up the normal random number generation process. To achieve this, the random index  $i$  is divided in to even and odd values and the respective values of the  $x_i$  and  $f_i$  are stored in separate memories (figure 6). For example, if the generated index  $i$  is an odd value the  $x_i$  and  $x_{i-1}$  are read from the odd and even memories at the same memory index positions simultaneously. However, if the

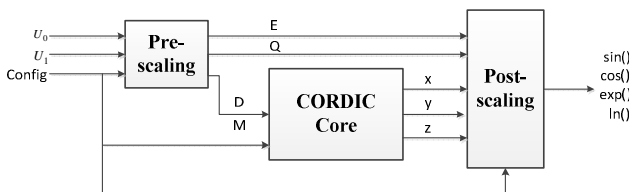


Figure 3 Architecture of the CORDIC module.

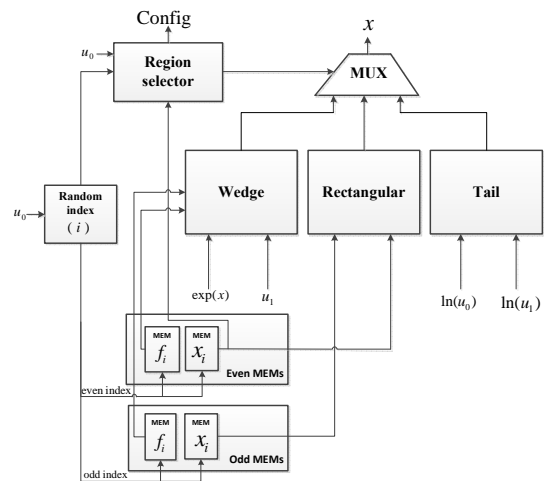


Figure 5 Architecture of the Ziggurat module.

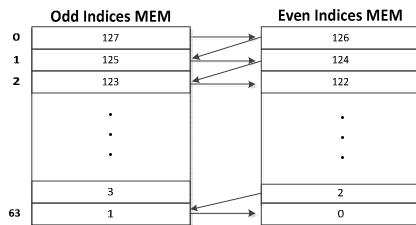


Figure 6 Illustration for the storage of the  $x_i$  and  $f_i$  coefficients in memory and the parallel access of the  $x_i, x_{i-1}, f_i$  and  $f_{i-1}$  for  $n=128$  generated index is with an even value, then  $x_i$  is read from the even memory and the  $x_{i-1}$  is read from the odd memory at the next memory position in parallel. As a result the parallel access of the coefficients is achieved with such simple but effective method.

### 5. Proposed hardware/software architecture

The architecture of the proposed system for the particle filter in a grid based Fast SLAM algorithm is given in figure 7, where it comprises of an embedded Microblaze processor responsible for the execution of software functions, particle filter hardware accelerator (PF HW accelerator), timer and UART cores for the purpose analysis and verification of the system. The PF HW accelerator composed of CORDIC and random number generator cores explained in section IV and they are connected to the Microblaze soft-core processor through a dedicated one to one communication bus (Fast simplex Link) for fast streaming of data.

In the implementation of the grid based Fast SLAM algorithm, real time odometry and laser data collected from a mobile robot platform are stored in a Block RAM memory to be used for post processing. The odometry data is used by the sampling step to generate new particle instances and the laser data is used in the importance weight step for the evaluation

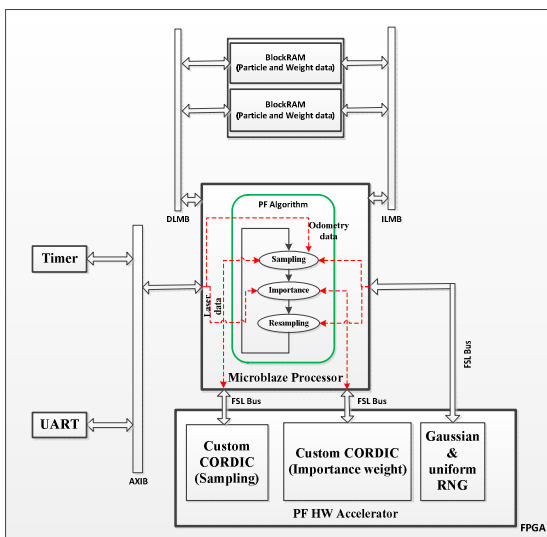


Figure 7 Architecture of the hardware/software co-design system

of particle weights. The particles and their associated weights are stored in Block RAM for fast accessing. For accommodating all the code and data in the embedded processor two Block RAMS units are used in our implementation.

Individual CORDIC hardware modules are assigned for the evaluation of the elementary functions in the sampling and importance weight steps. Uniform and Gaussian random numbers are provided to the resampling and sampling steps of the particle filter respectively by the random number generator module.

### 6. Hardware/software co-design implementation results and discussion

The implementation of the architecture given in figure 7 is performed on a Xilinx Kintex-7 KC705 FPGA device running at 100MHz. The design of the hardware modules is written in VHDL language. For the implementation of the Ziggurat module a value of  $n = 128$  is used and, the  $x_i$  and  $f_i$  coefficients are represented as fixed point numbers with  $Q_{8:24}$  format (i.e. 8 bits for the integer part and 24 bits for the fractional part). The same fixed point format is used for the representation of the different data in the CORDIC module design. For the variables in the software part of the algorithm a 32 bit floating point representation is used by enabling the floating point unit (FPU) of the Microblaze processor.

The summary of the execution time results in clock cycles for the three steps of the particle filter in the grid based Fast SLAM algorithm is given in table III. The obtained results in general shows that the hardware acceleration leads to better speed up in the execution time of the particle filter in all the three steps of the algorithm. In particular, significant speedup is achieved in the sampling step which can be attributed to the fast generation of Gaussian random numbers by the random number generator hardware module.

As the results of table III shows, the importance weight step takes relatively a large number of clock cycles compared to the sampling and resampling steps. This is due to the fact that, for the evaluation of the weight of each particle in the importance weight step, first it is required to transform every

TABLE III  
EXECUTION TIME FOR SAMPLING, IMPORTANCE WEIGHT AND RESAMPLING STEPS IN EMBEDDED SOFTWARE AND HARDWARE/SOFTWARE CO-DESIGN

	No. of Particles (N)	Execution times in clock cycles(100Mhz)		
		Sampling	Weighting	Resampling
Embedded software implementation	20	$N \times 47182$	$N \times 3933958$	127732
	40			228982
	60			329879
	80			431962
	100			533126
HW/SW Embedded implementation	20	$N \times 337$	$N \times 219935$	5129
	40			10719
	60			16321
	80			21937
	100			27537

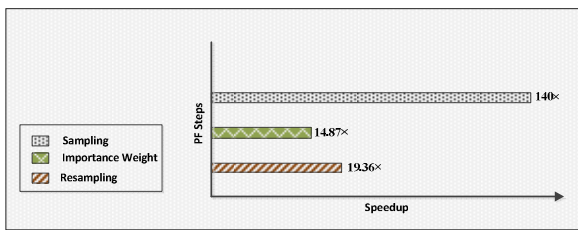


Figure 8 Speedup in HW/SW co-design of particle filter

laser scan measurements point data (range and bearing angle) from a robot frame of reference to a global frame of reference where normally more than hundreds of laser scans points have to be evaluated from the sensor. This requires the evaluation of sine and cosine functions for every scan point. After this step follows the search for the closest point between every laser scan end point and occupied points in a map (which is a computationally intensive step). However, at this stage this step is implemented in software in the embedded Microblaze processor but in the future we intend to speed up this process in hardware. Furthermore, computation of an exponential function is required in the calculation of the weight of each particle for every laser scan points. These are the main reasons attributed to the relatively large clock cycles obtained in the importance weight step.

## 7. Conclusions

This work presents a hardware/software co-design approach to speed up the computational time of the particle filter in a grid based Fast SLAM algorithm. With initial identification of the critical bottlenecks in each step of the particle filter algorithm, hardware acceleration block are designed and implemented to accelerate the computational time. Such simple and effective approach resulted in an improvement in the speedup of 140x, 14.87x and 19.36x in the sampling, importance weight and resampling steps respectively (figure 8). Such an approach can also be applied to other applications of the particle filter due the flexibility in the hardware/software co-design approach.

## REFERENCES

- [1] B. Ristic, S. Arulampalam, and N. J. Gordon, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*, Artech House Publishers, Norwood, MA, 2004
- [2] Gustafsson, F., "Particle filter theory and practice with positioning applications," *Aerospace and Electronic Systems Magazine*, IEEE, vol.25, no.7, pp.53,82, July 2010.
- [3] Nordlund, P.-J.; Gustafsson, F., "Sequential Monte Carlo filtering techniques applied to integrated navigation systems," *American Control Conference*, 2001. Proceedings of the 2001, vol.6, no., pp.4375,4380 vol.6, 2001
- [4] Atia, M.M.; Georgy, J.; Korenberg, M.J.; Noureldin, A., "Real-time implementation of mixture particle filter for 3D RISS/GPS integrated navigation solution," *Electronics Letters*, vol.46, no.15, pp.1083,1084, July 22 2010
- [5] M. M. Atia, M. J. Korenberg, and A. Noureldin, "Particle-Filter-Based WiFi-Aided Reduced Inertial Sensors Navigation System for Indoor and GPS-Denied Environments," *International Journal of Navigation and Observation*, vol. 2012.
- [6] Georgy, J.; Noureldin, A.; Goodall, C., "Vehicle navigator using a mixture particle filter for inertial sensors/odometer/map data/GPS integration," *Consumer Electronics*, IEEE Transactions on, vol.58, no.2, pp.544,552, May 2012
- [7] Happe, M., et al.: A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking. *Journal Real-Time Image Processing* (2011).
- [8] Medeiros, H.; Park, J.; Kak, A., "A parallel color-based particle filter for object tracking," *Computer Vision and Pattern Recognition Workshops*, 2008. CVPRW '08. IEEE Computer Society Conference on, vol., no., pp.1,8, 23-28 June 2008
- [9] G. Grisetti, C. Stachniss and W. Burgard "Improved techniques for grid mapping with Rao-Blackwellized particle filters", *IEEE Trans. Robot.*, vol. 23, no. 1, pp.34-46 2007
- [10] Chau, T.C.P.; Luk, W.; Cheung, P.Y.K.; Eele, A.; Maciejowski, J., "Adaptive Sequential Monte Carlo approach for real-time applications," *Field Programmable Logic and Applications (FPL)*, 2012 22nd International Conference on, vol., no., pp.527,530, 29-31 Aug. 2012
- [11] Fross, D.; Langer, J.; Fross, A.; Rössler, M.; Heinkel, U., "Hardware implementation of a Particle Filter for location estimation," *Indoor Positioning and Indoor Navigation (IPIN)*, 2010 International Conference on, vol., no., pp.1,6, 15-17 Sept. 2010.
- [12] T.C.P. Chau, X. Niu, A. Eele, W. Luk, P.Y.K. Cheung, and J.M. Maciejowski "Heterogeneous Reconfigurable System for Adaptive Particle Filters in Real-Time Applications" In *Proc. Int. Conf. on Reconfigurable Computing: Architectures, Tools and Applications (ARC)*. Pages 1–12. Springer. Mar. 2013.
- [13] Shih-An Li; Chen-Chien Hsu; Wen-Ling Lin; Jui-Pin Wang, "Hardware/software co-design of particle filter and its application in object tracking," *System Science and Engineering (ICSSE)*, 2011 International Conference on, vol., no., pp.87,91, 8-10 June 2011
- [14] Binli Ye; Yunhua Zhang, "Improved FPGA implementation of particle filter for radar tracking applications," *Synthetic Aperture Radar, 2009. APSAR 2009. 2nd Asian-Pacific Conference on*, vol., no., pp.943,946, 26-30 Oct. 2009
- [15] S.Saha, N.K.Bambha, and S.S.Bhattacharyya, "Design and implementation of embedded computer vision system based on particle filters," *Comput. Vision Image Understanding*, vol. 114, no. 11, pp. 1203–1214, 2010
- [16] HAA El-Halym, I Mahmoud, S Habib, Proposed hardware architectures of particle filter for object tracking. *EURASIP J. Adv. Signal Process.* 2012, 17 (2012).
- [17] S.Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. Intelligent robotics and autonomous agents. The MIT Press, August 2005.
- [18] R.Douc; O.Cappe, "Comparison of resampling schemes for particle filtering," *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, vol., no., pp.64, 69, 15-17 Sept. 2005.
- [19] Sileshi, B.G.; Ferrer, C.; Oliver, J., "Particle filters and resampling techniques: Importance in computational complexity analysis," *Design and Architectures for Signal and Image Processing (DASIP)*, 2013 Conference on, vol., no., pp.319,325, 8-10 Oct. 2013
- [20] J. E. Volder., *The CORDIC Trigonometric Computing Technique*, IRE Trans. Electronic Computers, pp. 330-334, Sept 1959.
- [21] G. Marsaglia and W. W. Tsang, "The ziggurat method for generating random variables," *Journal of Statistical Software*, vol. 5, pp. 1–7, 2000
- [22] P.L 'Ecuyer, Maximally equidistributed combined Tausworthe generators, *Mathematics of Computation*, vol. 65,no. 213, pp. 203–213, 1996.
- [23] J.S. Walther. "A Unified Algorithm for Elementary Functions", *Proceedings of the Spring Joint Computer Conference*, 1971, pp. 379-385.
- [24] Boudabous, A.; Ghazzi, F.; Kharrat, M.W.; Masmoudi, N., "Implementation of hyperbolic functions using CORDIC algorithm," *Microelectronics*, 2004. ICM 2004 Proceedings. The 16th International Conference on, vol., no., pp.738,741, 6-8 Dec. 2004
- [25] B. Lakshmi and A. S. Dhar, "CORDIC Architectures: A Survey," *VLSI Design*, vol. 2010, Article ID 794891, 19 pages, 2010.



# The Design of an embedded Self-Diagnostic Hybrid Aquarium Control System

Tochukwu Chiagunye, Chukwugoziem Ihekweaba, Henrieta Udeani

Computer Engineering Dept. Michael Okpara University of Agriculture, Umudike, Abia State Nigeria.

## Abstract

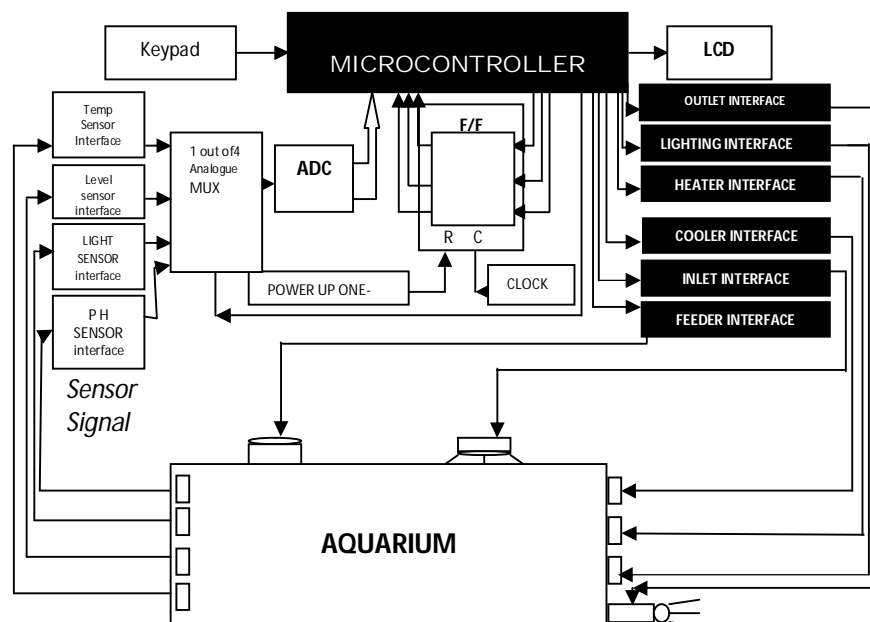
An aquarium control system which permits the monitoring and management of most of the parameters of the electrical devices that can be found in an aquarium has been designed in the course of this project. The microcontroller used to realize it combines a real time clock and a temperature sensor in order to control the seven relays used for switching of the various aquarium resources. The liquid crystal display shows the current-date and time, the temperature, water level and light level as detected by the sensors. Also, it permits one to visualize each port status. Other important features include an automated mechanical feeding device for the aquarium as well as automated water draining and refilling operations of the aquarium after a specific time frame. This system offers flexibility in the control of operations by providing a user interface with which parameters can easily be adjusted. The flexibility that exists in the system makes room for future enhancement.

**Keywords;** Aquarium Control, multi-disciplinary system, Self-Diagnostic, Software Agent.

## 1. Introduction

The development of hybrid systems consisting of software, electronics and mechanical components

which are operating in a physical world is usually a challenge. These challenges arise from the need to develop complex firmware products that take the constraints of the physical world into account. The task is made even more challenging due to the fact that these types of systems often are developed out of phase. Initially, the mechanical parts are designed, followed by the electronics and finally the system software is developed. Any problems discovered late in the development process, can really only be corrected in the software without causing significant delays to the complete project due to longer iterative cycles in the electronics and mechanical development. These very late changes often increase the complexity of the software and the risk of introducing new bugs. Hence, a well thought-out software design can be compromised. In order to avoid situations like this, early feedback at the basic systems level is invaluable. In order to develop hybrid systems, engineers from different backgrounds and with diverse fields of expertise are involved, making communication much harder than in mono-disciplinary projects. It is close to impossible for each individual engineer to foresee all the cross-discipline consequences of a given design decision. The system in Fig 1 below describes the system controlling the level of water, temperature level, light intensity and the feeding of fish in an Aquarium.



The Sensor interface (temperature,level,P<sup>H</sup> and light) to the system are used to monitor the changes in the set physical conditions of the system and transmits same to a microcontroller through an Analog-to-Digital converter. Subsequently the system calculates and actuates the temperature,level and light transducers as well as the electric motor control signals. Also, a reagent fluid is pumped into the water in order to lower the bacteria level below a predetermined threshold as detected by the P<sup>H</sup> sensor.

**2. Materials and Methods**

The hardware components used in this design include;sensors, relay and driver, Electric motor,Liquid crystal display, Light emitting diodes, Transistors,Capacitors, Resistors and the Microcontroller.The LM 35 Temperature sensor is an integrated circuit sensor that can be used to measure temperature hence the analogue electrical output signal is proportional to the temperature in degree celsius. The light dependent resistor is a resistive light sensor that changes its electrical resistance from several thousand ohms in the dark to only a few hundred ohms when light falls upon it. The net effect is a decrease in resistance for an increase in illumination. ULN 2003A relay driver is a high-voltage, high current Darlington transistor array. Each driver consists of seven NPN darlington pairs that feature high-voltage outputs with common-cathode clamp diodes for switching inductive loads. The collector – current rating of a single darlington pair is 500mA. The darlington pairs can be paralleled for higher current capability for the output to the feeder system.

In this project, the seven NPN darlington pairs, 1C to 7C are configured as follows:

- 1C is used to drive the inlet pump
- 2C is used to drive the outlet pump
- 3C is used to drive the heater
- 4C is used to drive the cooler
- 5C-7C are used to drive the feeder system

The Liquid crystal display (LCD) is a thin, flat electronic visual display that uses the light modulating properties of liquid crystals (LCs). This project used LCD 16 x 4 as a display system. The LCD 16 x 4 has four rows, where each row displays 16 characters. Here the first two rows display the sensor values, sensed by the temperature, level and light sensors, while the last two rows display the condition of the output, for example if the pump is ON the display on 3rd row is pump ON, if heater and light is ON, then the 3rd

and 4th rows will display heater and light ON respectively. Light emitting diodes emit light when an electric current passes through them. The buzzer is an alarm system that gives a warning sound during fault condition. A major component deployed in this system is the microcontroller. Microcontrollers are independently programmable and can have a great deal of additional functionality combined on the same integrated circuit. A typical microcontroller can access from a megabyte to a gigabyte of memory, and is capable of processing 16, 32 or 64 bytes of information or more with a single instruction. In contrast to the microprocessor, a microcontroller includes a central processing unit, memory and other functional elements, all on a single semiconductor substrate, or integrated circuit.

A typical microcontroller might have a core microprocessor, a memory controller, an interrupt controller, and both asynchronous and synchronous serial interfaces. The advantage of a microcontroller as compared with a microprocessor is that the microcontroller can be used in an autonomous way. No external circuitry is needed for their operation. This is why their use is very wide spread in relatively straight forward applications, such as in small electronic products.The specific type of microcontroller used in this project is a microcontroller from the ATMEL technology incorporated AT 89C51.The design approach used in this project is the top down design. Here the whole system was broken down into different smaller modulus.

The final step is integration of the different modules to form the system required. It is important to note that any of the above modules can be tackled first and important information that can be used for the other recorded appropriately for reference purpose.

**3. System Design**

This work is divided into hardware and software sections. The hardware part consist of three sub-systems which include; the input sub-system, control sub-system, and output sub-system. The software unit is an Agent based control program.

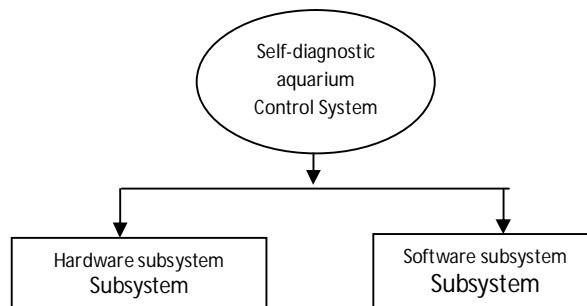


Fig 2 Block Diagram of the system

The input- sub system (monitoring unit) consists of a temperature sensor circuit, water level sensor circuit and light sensor circuit. The temperature sensor senses the water temperature inside the aquarium. The water level sensor functions to detect the level of water inside the main aquarium and reserve tank. The function of the light sensor is to detect light intensity outside the aquarium. The Analogue-to-Digital converter (ADC) sends the digitized data to the microcontroller.

The control sub-system consists of an AT89C51 microcontroller, 16MHz crystal oscillator, 5V regulator used to regulate the voltage to 5V, reset button and other basic ports of the microcontroller. The control sub-system processes digitized data received by the data acquisition system and sends digitally processed data to the LCD and other output devices through the output interface.

The output sub-system consists of 74ACT574 Octal D-Type Flip-Flop with tri-state output. This is used to latch the output port 0 of the microcontroller to the LCD, as well as the ULN 2003A relay driver that drives the resources feeder,heater,cooler,inlet pump, outlet pump, LCD and the resources interfaced to the aquarium.

Assembly language is used to develop codes for the microcontroller to enable it read the values sent by the sensors and take appropriate actions. Visual basic programme is used to interface the operations of the system to the liquid crystal display (LCD) and the assembly language program running on the hardware in order to read the user set point values and the current values.

**Hardware Sub-System Design**

**ADC Configuration**

Analog-to-Digital converters are among the most widely used devices for data acquisition. Since physical quantities such as temperature, pressure, humidity, etc, need to be converted to analogue electrical signals, there arises the need for an analogue-to-digital converter which in turn converts the analogue signals to digital quantities, for the use of the Microcontroller. The ADC used is ADC0804. For the ADC selected, The frequency of operation;

$$f = \frac{1}{1.1RC} \text{ (Hz)}$$

Here, R = 10k, c=150pF

$$f = \frac{1}{1.1 \times 10^4 \times 150 \times 10^{-12}}$$

$$f = 606.0606 = 606\text{Hz}$$

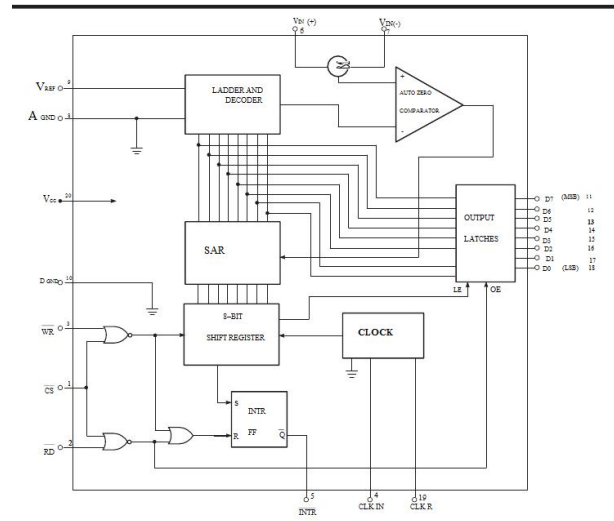


Fig 3 Block diagram of ADC0804 (copy right (c) 2009-2013 Texas instruments incorporation)

**Table i V<sub>ref</sub>/2 relationship with V<sub>in</sub> range**

V <sub>ref</sub> /2 (v)	V <sub>in</sub> (v)	Step size (mV)
Not connected	0 to 5	5/256 = 19.53
2.0	0 to 4	4/256 = 15.62
1.5	0 to 3	3/256 = 11.71
1.28	0 to 2.56	2.56/256 = 10.0
1.0	0 to 2	2/256 = 7.81
0.5	0 to 1	1/256 = 3.90

The output voltage,  $D_{out} = \frac{V_{in}}{\text{Step size}}$

Thus for the temperature sensor; LM 35 with input of 250mV to the ADC, the digital output from the ADC after conversion is:

$$D_{out} = \frac{250\text{mV}}{10\text{mV}} = 25 \text{ (in } ^\circ\text{C) as temperature reading.}$$

**Water Level Sensor interface design**

The water level sensor used in this project is a comparator; LM324 (Low Power Quad Operational Amplifiers). If the water sensor probe detects water, the voltage output from the LM324 to the controller is 0v. Conversely, the voltage from the LM324 to the controller is about 3.8v.

In practise, when the liquid comes in contact with the electrode tip, a conductive path is established between the sense electrode and the tank wall/reference electrode. This current is sensed, amplified and made to operate a relay whose contacts in turn are used for annunciation/control.

**The Control System Design**

The 89C51 Microcontroller was used to implement the control subsystem of the project.

For the RC Auto reset network used in this design, t (time delay) must be at least two (2) instruction time cycles. However, the basic instruction cycle of the microcontroller is 12 clock pulses. The frequency of oscillation chosen for the Microcontroller is 16MHz

$$f = 16\text{MHz}$$

$$T = 1/f = \frac{1}{16\text{MHz}} = 6.25 \times 10^{-8} \text{ sec}$$

For an instruction cycle,

$$T = 6.25 \times 10^{-6} \times 12 = 7.5 \times 10^{-7} \text{ secs.}$$

using 2 instruction cycles,

$$2T = 2 \times 7.5 \times 10^{-7} = 1.5 \times 10^{-6} \text{ secs}$$

$$= 1.5\mu \text{ secs.}$$

t = In 2RC, where t = 1.5µ secs  
 choice of 10µf capacitor was made  
 Taking t = 7ms, R = ?

Therefore,  $7 \times 10^{-3} = 0.693 \times R \times 10 \times 10^{-6}$

6

$$R = \frac{7 \times 10^{-3}}{6.93 \times 10^{-6}} = 1010.100\Omega$$

$$R = 1\text{k}\Omega$$

The number of reset network used is one

(1)

The crystal oscillator is a thin slice of natural quartz or a syntheses material.

The piezoelectric crystal is used as a resonant circuit, in place of an alternative; LC circuit. The Piezoelectric effect of the crystal helps it to vibrate mechanically when excited electrically thus producing an AC voltage output. The resonant frequency is fixed by the speed of the crystal. Typical values of the frequency are 0.5 to 30MHz. The value used for this project was 16MHz.

Finally, the transparent latch and the relay driver are incorporated into the system as already outlined. The final circuit derived is shown in fig 4 below.

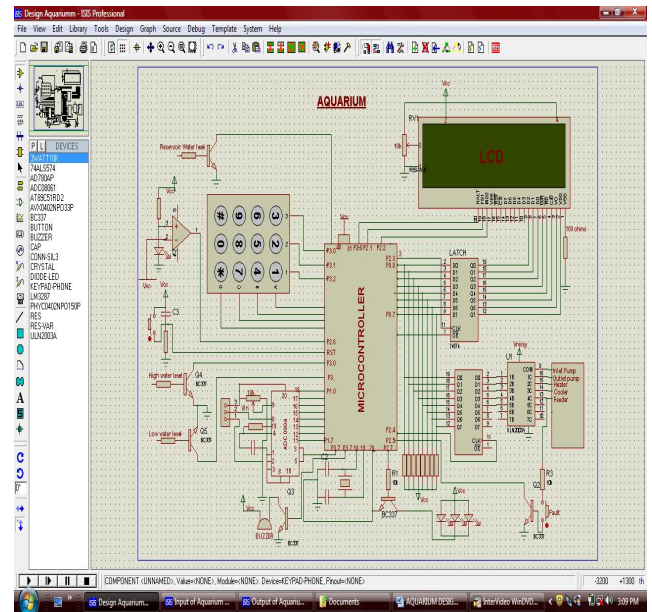
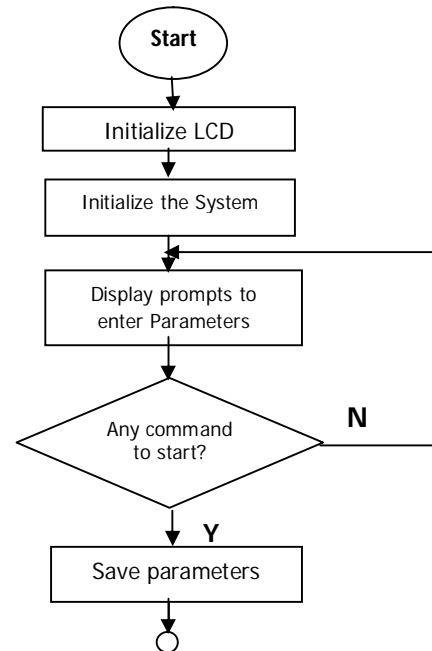


Fig 4: Circuit Diagram

**The Software Sub-System Design**

The Flow chart of fig 5 shows the detailed control flow from initialization to the last command stage that derives the feed sub-routine and refill sub-routine.

Software Sub-System Flow Chart



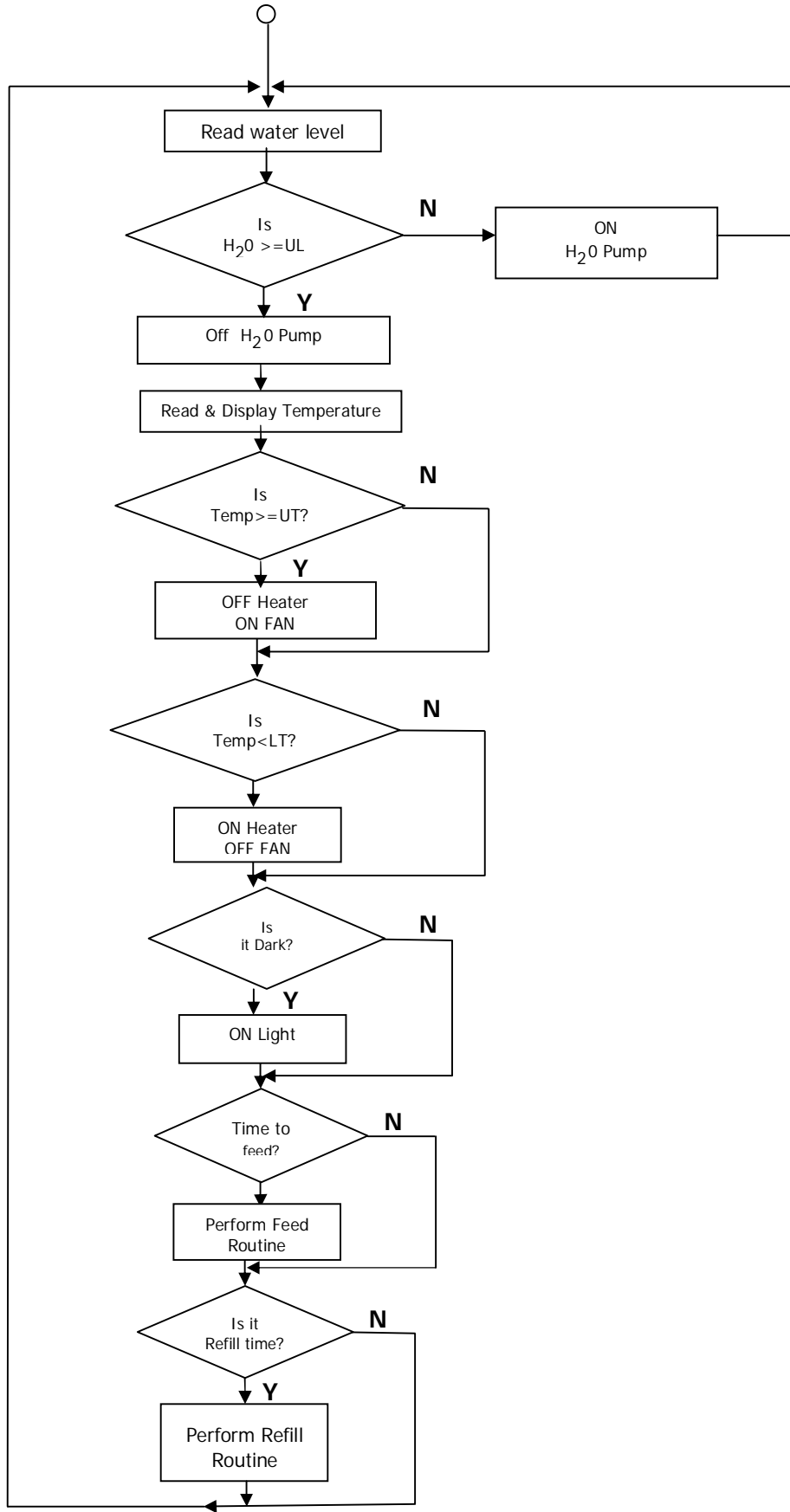


Fig 5: Software Sub-System Design Flow Chart

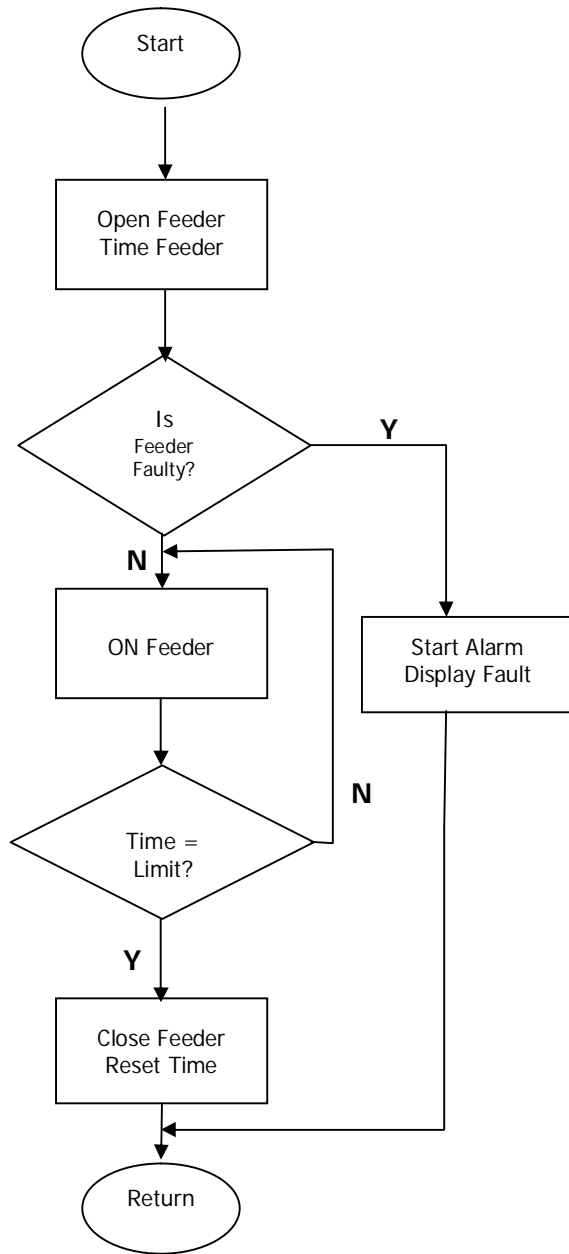


Fig 6: Feeder Routine Flow Chart.

Fig 6 above shows the Feeder routine flow chart. This routine checks the set time to feed and the condition of the feeder, whether it is faulty or not, to determine what action to take. Here an Agent that monitors and displays the fault condition if any is used to constantly monitor the feeder.

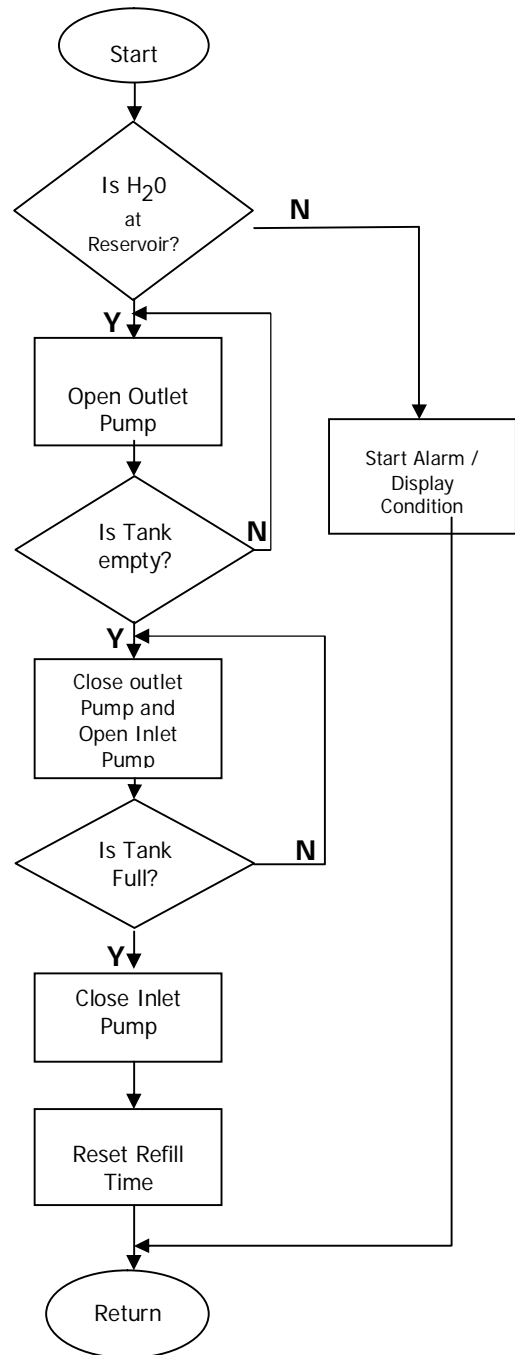


Fig 7: Refill Routine Flow Chart

The refill Sub-Routine flow chart shown in fig 7 monitors the conditions of refilling, that is, checks the water at the reservoir, the refill time to either refill or start Alarm while displaying the fault condition.

#### 4.0 Tests and Results

**Table ii Test results**

Circuit	Output (dc-voltage)	Display
Temperature sensor ( 35) <ul style="list-style-type: none"> <li>Ice (0 °c)</li> <li>Boil water (100 °c)</li> </ul>	0 v 1 v	Tested ok
Water sensor <ul style="list-style-type: none"> <li>Sense water</li> <li>Sense no water</li> </ul>	0 v 3.8v	Tested ok
Lighting sensor <ul style="list-style-type: none"> <li>Dark environment</li> <li>Bright environment</li> </ul>	0 v 2v – 4v	Tested ok
Push button <ul style="list-style-type: none"> <li>Closed circuit (PUSH)</li> <li>Open circuit (normal)</li> </ul>	0 V 5V	Reliable
Microcontroller output <ul style="list-style-type: none"> <li>Data logic 1</li> <li>Data logic 0</li> </ul>	4.7v 0V	Tested ok

After the correction of the errors encountered during the sub-system tests, the overall system test was carried out and the required result was achieved with the hardware and software parts harmoniously working together.<sup>9</sup> Multidisciplinary Project of this sort is not left with challenges like integration, finance and environmental<sup>10</sup> factors, all these were handled with the help of software agent based prototype system that is working perfectly well.

#### 5.0 Recommendation and Conclusion

The main goal of the work is to design a self-diagnostic multi-disciplinary embedded system. The project meets all the objectives set forth while satisfying the constraints.

The system was designed to be much user-friendly by providing the owner with keypad and liquid crystal displays interface where one can stipulate set-point and monitor the changes in trends. The detailed analysis of this project design is made available to expose the underlying technology of this project and related works. This is particularly significant in view of the fact that food security has become a very serious global issue.

#### REFERENCES

1. Merriam-Webster Online Dictionary "Definition of aquarium". <http://www.m-w.com/dictionary/aquarium>. Retrieved 2013-10-03
2. Adey, Walter H.; Loveland, Karen (1991). *Dynamic Aquaria*. San Diego: Academic Press. ISBN 0-12-043792-9
3. <http://www.aquariumslife.com/headline/amazon-biotope-video/>. Retrieved 2013-08-25
4. "A Preventative Maintenance Schedule". [http://www.aquariumfish.net/information/maintenance\\_schedule.htm](http://www.aquariumfish.net/information/maintenance_schedule.htm). Retrieved 2013-09-20
5. Sanford, Gina (1999). *Aquarium Owner's Guide*. New York: DK Publishing. pp. 180–199. ISBN 0-7894-4614-6.
6. STMicroelectronics LM324 low power quad operational amplifiers. Italy: Data sheet. June 1999.
7. [www.ti.com](http://www.ti.com) (2009-2013) Texas Instruments Incorporation. Retrieved 2013-09-12
8. <http://www.atmel.com>. 8-bit Microcontroller with 4k bytes flash. Retrieved 2013-09-15

[www.en.wikipedia.org/wiki/Transducer](http://www.en.wikipedia.org/wiki/Transducer).

<http://eedept.ncue.edu>. Retrieved 2013-10-08

<http://www.datasheetarchive.com/> ADC 0804. Retrieved 2013-10-16.

# A Consolidated Review on Embedded Micro-Controllers for Pace Maker Applications

**Abhishek Sharma**

Accendere Knowledge Management Services Pvt Ltd  
Flat No. 302, Plot No. 553, Rama Residency, KPHB VI Phase, Kukatpally, Hyderabad – 5000851  
India.

**Abstract**— This review paper is based on the real world applications of various microcontrollers in the field of pacemakers. The study conducted is based on various factors such as cost, power consumption, power saving, reliability, efficiency etc. These features differentiate each microcontroller from the other. The goal of this study is to study each microcontroller that are being employed in the field of pacemakers and based on the observations; we can find the best alternative to the presently used microcontrollers. That means, finding a microcontroller which will function as a close substitute to the presently used microcontroller. The microcontrollers which showed similar characteristics to the ones presently being used are suggested to be used for serving the purpose of real time embedded pacemakers and related applications. The various microcontrollers that were studied included PIC (10F,12F,18F) series, MegaAVR(ATMEGA 32, ATMEGA 328P) series, Freescale RS08 Family & TI MSP4300XX series of microcontrollers. By studying their diverse characteristics such as program and data memory, clock cycles, memory, data retention rate, peripherals, I/O features etc., various other microcontrollers were searched for which showed more or less the same characteristics as the above listed ones had. Such similar microcontrollers are hence suggested to be used in the pacemaker and other related application. Suggested microcontrollers include Intel 87C52, PIC (10F, 12F, 16F87x, 18Fxx2) series, FRDM-KL 25Z, Freescale RS08 Family, TI MSP 430XX series of microcontrollers.

**Keywords:** pacemaker, embedded hardware, review, micro-controllers

## 1. INTRODUCTION

Microcontrollers are being used today in huge variety of applications such as mobile phones, auto mobiles, CD/DVD players, Pacemakers, ATM machines, cameras, washing machines, microwave ovens, EVG machines and the list goes on and on.

Microcontrollers in pacemakers are being studied and we find the most suitable applications of various microcontrollers in:

- A biomedical sensor system for real-time monitoring of astronauts' physiological parameters during extra-vehicular activities
- A heart disease recognition embedded system with fuzzy cluster algorithm
- Realtime & Embedded System Testing for Biomedical Applications
- Design Overview Of Processor Based Implantable Pacemaker
- Design and Implementation of Portable Health Monitoring system using PSoC Mixed Signal Array chip
- Adaptive Electrocardiogram feature extraction on distributed embedded systems
- Real time monitoring of ECG signal using PIC and web server
- Differential diagnosis of QRS complex Tachycardia and Tachyarrhythmia in Noisy ECG Signals through Fuzzy Neural Signal Processing Embedded System
- Hardware Embedded System on a Chip for the Normal ECG Recognition
- The Design of Ecg Signal Generator Based on ARM9

## 2. BRIEF LITERATURE SURVEY ON THE CURRENT PACEMAKER MICROCONTROLLERS

- Manasi Safaya developed the fetal pacemaker circuit using PIC 10F202 which was successful in pacing at the desired heart rate set by the user (Doctor). For a heart rate of 120bpm, the circuit pulsed every 500ms.
- Jacob I.Laughner, Erik R. Zellmer, Matthew R. MacEwan, Sophia X.Cui, Igor R. Efimov, Scott B. Marrus, Carla J. Weinham, Jeanne M. Nerborne successfully implemented fully implantable mouse pacemaker based on wireless power transfer and control capable of 30 days of in vivo pacing using PIC12F675. All experiments were performed on anesthetized animals with a hand-held transmitter. Integration of transmitter coil into the mouse housing was done. The system was designed to generate voltage fields in excess of the 3.9 V cap on the receiver throughout the cage to eliminate communication issues between the transmitter and receiver to ensure that mice will



always be paced independent of the location they occupy in the cage.

- Cliff Nixon, James Smith, Tony Ulrich, Rebecca Davis, Christopher Larson, Kua Cha achieved desired sensitivity of  $\pm 20\%$  in an academic dual chamber pacemaker by using PIC18F452. It was also possible to attain an error of less than 1% under all input via adjustment of the digital controlled voltage reference at the comparator with respect to measured results. The pacing circuit provided a pulse width of .05ms to 1.9ms, with a .2ms tolerance through digital control of the semiconductor switches involved in pacing. The pacing amplitude was variable from 1.2 to 7v, with a tolerance of 12%. The program for measuring lead impedance was not written due to time constraints. The theory for performing this operation was verified, and simulation yielded excellent results. The requirements for this circuit were lenient, allowing 25% error. Simulation proved that an error of only 1% could be obtained. Taking into account resistor tolerances and voltage drops across switches, the error is expected to be no greater than 5%. The system was fully functional and met all specifications regarding control of the rate limits and blanking periods.

- Kenneth Chee showed that the MCU/FPGA and the FPGA implementation had a 128% and 86566% increase in speed, respectively, when compared to a MCU implementation. The increase in speed also increased the power consumption due to the parallelization of the algorithm implementation. By adding an FPGA the MCU/FPGA and FPGA implementation had a maximum instantaneous power increase of 204% and 105%, respectively, when compared to the MCU implementation. This was done using ATMEGA 328P using the Kendall Tau algorithm for adaptive pacing.

- Carlos Cassillas used a High performance, low cost, low power MCU i.e. Freescale RS08 (MC9RS08KA2). System is battery operated with Continua Health Alliance Connectivity over USB. It makes use of only 1 Lead. However, a full 12 lead ECG can also be implemented using DSC MC56F8013.

- Santosh Chede and Kishore Kulat employed TI MSP430F1611 in appropriate measuring and modeling scheme which was successfully implemented to measure instantaneous current and to derive energy consumption. They successfully developed low power processor based implantable pacemaker and estimated software related current/ energy consumption.

Since the embedded portable devices (in our case a pacemaker) are generally battery operated, therefore they should be designed in such a manner that minimum power is consumed. Design constraints such as size, weight, encapsulating material & longer battery life are mainly emphasized. Therefore, design must be optimized. VLSI based analog/digital custom processor and interfacing peripherals used in pacemaker increases the cost and time to market.

Safety issues related to pacemakers and defibrillators due to firmware problems, software related issues etc. is studied. The devices which showed reasonable probability to cause adverse effects on health or even death were classified as Class I by Food and Drug Association. Software is not reviewed by FDA during pre market submission. There are no set standards for

this because code length may be as long as 80,000 lines or more. But a few guidelines are recommended for software review. It is the manufacturer's responsibility to demonstrate the safety & efficacy of device software. The methodologies used to monitor this are documentation of code inspections, static analysis, module level testing & integration testing. However, these tests are basically open loop tests and thus they fail to check the correctness of device software. Effective software evaluation & verification methodology is needed to analyze the risks and certify the medical device software during pre-market submission phase.

Today's testing methods are erroneous and traditional methods are ineffective because testing depends upon a particular patient's state and organ. The problem changes as the testing environment changes, the latter not being under the control of the tester.

Implantable medical devices are primary example of medical cyber physical system where software's safety and effectiveness has to be evaluated within a closed loop context of a patient. The major challenge is to generate physiologically relevant test such that device does not generate irrelevant therapy & does not adversely affect the condition of patient. Also the mechanism must be interactive and adaptive i.e. it must take into account the previous values as well as present values. Traditional testing methods are basically open loop tests for error debugging involving methods such as theorem proving, constraint logic programming & symbolic execution, model checking, using an event flow model and using a Markov chains model. Therefore, testing becomes complex and difficult when program under test is non deterministic. The primary approach to a system level testing of medical devices is unit testing in which pre recorded electrogram and electro cardiogram signals are played back just like a tape is played in a tape recorder. However this method is unable to check safety violations due to inappropriate stimulus generated by the pacemaker. Heart rate should be properly maintained. It should not be too fast (Tachycardia) or too slow (Bradycardia). It should be at a normal pace (Normal Sinus Rhythm). There should be synchronism between atrioventricular muscles. Pacemaker mediated Tachycardia introduces following complications:

- Endless Loop Tachycardia
- Atrial Flutter & pacemaker mode-switch function

Embedded system must be reliable and should deliver high performance. Reliability is difficult to maintain using traditional designs because large systems are complex & complexity introduces errors. High performance adds to complexity. A major limitation of Application level simplex architecture is that bugs present in microprocessor, RTOS are not handled safely. The system level simplex architecture has 2 types of faults:

- Logical Faults: Occurs when incorrect value is encountered by the controller.
- Resource sharing faults: These are caused by failures in common resources among components.

### 3. MICROCONTROLLER PACEMAKERS CURRENTLY BEING USED

PIC 10F series: (PIC 10F202)

Highlighted Features:

1. SOT 23 Package.
2. High Performance RISC CPU.

PIC 12F series: (PIC12F675)

Highlighted Features:

1. High-Performance RISC CPU.
2. Small Size.
3. Low Cost.
4. Internal Memory enables 14 different programmable stages.

PIC 18F series: (PIC 18F452):

Highlighted Features:

1. High-Performance RISC CPU.
2. Responsible for directing the pacing of heart.
3. Wide academic use & flexible I/O ports.
4. I/O ports include several communication protocols and interrupts that streamline programming and increase response times.

MegaAVR Series: (ATMEGA 328P)

Highlighted Features:

1. High Performance, Low Power AVR® 8-Bit Microcontroller with Advanced RISC Architecture.
2. 1 ALU limits it to 1 instruction executing at a time.
3. High performance is delivered using FPGA instead of multiple MCUs.
4. It lets designer to define the required hardware & minimizes power consumption.

5. MCU acts as master and FPGA acts as slave.

Freescale RS08: (MC9RS08KA2):

Highlighted Features:

1. Low cost.
2. Ultra low end 8bit MCU.
3. ICS—Internal clock source avoids the use of the external oscillator and provides a bus clock of up to 10 MHz for fast code execution.
4. MTIM—8-bit modulo timer is used to define time intervals used in heart beat detection algorithms.
5. ACMP—Analog comparator has internal threshold levels and offers the possibility of adding an external reference.
6. BDM—Background debug module provides a single-wire in-circuit debug interface.
7. 6-pin and 8-pin packages—Small packages for small designs. Enough pins for heart rate monitor implementation.

TI MSP430 series: (MSP 430F1611):

Highlighted Features:

1. Ultra low power MCU.
2. 5 low power modes to extend battery life in portable biomedical applications.

a. *Tabular Representation of the same is given below:*

Apart from the highlighted features, various other features were studied as shown in Table 1.

Based on the Table 1 mentioned below, advantages and disadvantages of the microcontrollers, some new and similar microcontrollers for pacemaker applications can be suggested.

Type of Microcontroller	Number of instructions	Addressing Modes	Operating Characteristics	Operating Current	Standby Current	Data Retention	Temperature Range	Operating Voltage	Timer
PIC10F202	33 single word instructions 12 bit wide	Direct, Indirect and Relative	4MHz internal clock, 1µs instruction cycle	<175µA@2V, 4MHz, typical	100nA@2V typical	>40 years	Industrial: -40° C to 85° C Extended: -40° C to 125° C	2V-5.5V	8bit real time clock/counter TMR0 with 8bit programmable prescaler
PIC12F675	35 single word single cycle 1µs except for program branches	Direct, Indirect and Relative	DC 20MHz clock	8.5µA@32kHz,2V,typical 100µA @1MHz,2V,typical	1nA@2V, typical	>40 years	-40° C to 85° C	2V-5.5V	8bit real time clock/counter TMR0 with 8bit programmable prescaler
PIC18F452	16bit wide instructions, 8bit wide data	Literal, Direct & Indirect	DC 40MHz osc/clock input	<1.6µA typical @5V,4MHz 25µA typical @3V,	<0.2µA typical	>40 years	-40° C to 125° C	2V-5.5V	Two 16bit timer/counter with

	path			32kHz					prescaler  One 8bit/16bit timer/counter with prescaler  One 8bit/16bit timer/counter with 8bit period register
ATMEGA328P	131 powerful instructions, 15 bit wide	Direct, Indirect, Indirect with displacement, Indirect with pre-decrement, Indirect with post-decrement	DC 20MHz clock	40mA per I/O pin	0.07mA-0.15mA @6MHz	20years @ 85°C, 100 years @ 25°C	-40°C to 85°C	1.8-5.5V	Two 8bit timer/counter with separate prescaler & compare mode.  One 16bit timer/counter with separate prescaler, compare & capture mode
Freescale MC9RS08KA2	Simplified S08 instruction set with added high performance instructions	Inherent, Relative, Immediate, Tiny, Short, Direct, Extended, Indexed	20MHz internal clock source upto 10MHz internal bus operation	120mA	NA	Min 15years, 100 years typical	-55°C to 150°C	0.3-5.8V	MTIM: 8bit modulo timer

Table 1: Key features of presently used microcontrollers

		PIC10F200	PIC10F202	PIC10F204	PIC10F206
<b>Clock</b>	<b>Max frequency of operation(MHz)</b>	4	4	4	4
<b>Memory</b>	<b>Flash Program Memory</b>	256	512	256	512
	<b>Data Memory(Bytes)</b>	16	24	16	24
<b>Peripherals</b>	<b>Timer Module(s)</b>	TMR0	TMR0	TMR0	TMR0
	<b>Wake up from sleep on Pin Change</b>	Yes	Yes	Yes	Yes
	<b>Comparators</b>	0	0	1	1
<b>Features</b>	<b>I/O Pins</b>	3	3	3	3
	<b>Input only Pins</b>	1	1	1	1
	<b>Internal Pull-ups</b>	Yes	Yes	Yes	Yes
	<b>In-Circuit serial Programming</b>	Yes	Yes	Yes	Yes
	<b>Number of Instructions</b>	33	33	33	33
	<b>Packages</b>	6-pin SOT-23 8-pin PDIP	6-pin SOT-23 8-pin PDIP	6-pin SOT-23 8-pin PDIP	6-pin SOT-23 8-pin PDIP

Table 2: Key features for suggested PIC 10F Microcontrollers

<b>Device</b>	<b>Program Memory</b>	<b>Data Memory</b>		<b>I/O</b>	<b>10 Bit A/D (ch)</b>	<b>Comparators</b>	<b>Timers 8/16bit</b>
	<b>FLASH(words)</b>	<b>SRAM (bytes)</b>	<b>EEPROM (bytes)</b>				
<b>PIC12F629</b>	1024	64	64	6	--	1	1/1
<b>PIC12F675</b>	1024	128	128	6	4	1	1/1
<b>PIC12F683</b>	2048	128	256	6	4	1	2/1

Table 3: Key features for suggested PIC 12F Microcontrollers

Device	On-Chip Program Memory		On-Chip RAM(bytes)	Data EEPROM(bytes)
	FLASH(bytes)	Single Word Instructions		
PIC18F242	16k	8192	768	256
PIC18F252	32K	16384	1536	256
PIC18F442	16K	8192	768	256
PIC18F452	32K	16384	1536	256

Table 4: Key features for suggested PIC 18F Microcontrollers

Key Features	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	---	PSP	---	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 instructions	35 instructions	35 instructions	35 instructions

Table 5: Key features for suggested PIC 16F Microcontrollers

#### 4. SUGGESTED MICROCONTROLLERS FOR PACEMAKERS APPLICATIONS

The names in bold letters are the microcontrollers that have not been used in pacemakers while the ones which are underlined and bold are the microcontrollers used in pacemakers.

1. The **Intel 87C52** is a MCS-51 CMOS single-chip 8-bit microcontroller with 32 I/O lines, 3 Timers/Counters, 6 Interrupts/4 priority levels, 8K Bytes On-Chip ROM/EPROM, 256 Bytes on-chip RAM, Programmable Serial Channel with Frame Error Detection. It can be used instead of **ATMEGA328P**.

2. The **PIC10F series(200,204,206)** are functionally similar to **PIC10F202** because they are from the same family of microcontrollers. A tabular representation highlighting different features is shown in Table 2.

3. The **FRDM-KL25Z** is an ultra-low-cost development platform for Kinetis L Series KL1x (KL14/15) and KL2x (KL24/25) MCUs built on ARM® Cortex™-M0+ processor. Features include easy access to MCU I/O, battery-ready, low-power operation, a standard-based form factor with expansion board options and a built-in debug interface for flash programming and run-control. The **FRDM-KL25Z** is supported by a range of Freescale and third-party development software.

4. The **PIC12F675** belongs to PIC12F series of microcontrollers. The functionalities are listed in Table 3 along with those of **PIC12F629** and **PIC12F683**.

5. The only difference between **MC9RS08KA2** and **MC9RS08KA1** is the On-chip Flash EEPROM that is:

- MC9RS08KA2: 2048 bytes
- MC9RS08KA1: 1024 bytes.

6. **PIC18F452** belongs to **PIC18Fxx2 series** as shown in Table 4.

7. **PIC16F87X series(873,874,876,877)** are highlighted in Table 5.

8. **TI MSP430F1611** has 48KB, 256B Flash Memory, 10KB RAM whereas **TI MSP430F2131** has 8KB,256B Flash Memory,256B RAM.

#### 5. CONCLUSIONS

The microcontrollers mentioned in section 4 can be successfully used to serve the application of pacemaking. The preferred microcontrollers as shown in the various tables can replace the ones used presently. The key features highlighted in the above tables makes the suggested group of microcontrollers a potential candidate for pacemaker applications.

#### 6. REFERENCES

- [1] Longjian Xu, Houwu Zhang, Kaixue Yao “The analysis and design of diphasic pacemaker pulse system based on microcontroller”, Coll. of Comput. Sci. & Inf., Guizhou Univ., Guiyang, China.
- [2] Jacob I. Laughner, Erik R. Zellmer, Matthew R. MacEwan, Sophia X. Cui, Igor R. Efimov(Department of Biomedical Engineering, Washington University in Saint Louis, Saint Louis, Missouri, United States of America), Scott B. Marrus, Carla J. Weinheimer(Department of Internal Medicine, Division of Cardiovascular Sciences, Washington University in Saint Louis, Saint Louis, Missouri, United States of America), Jeanne M. Nerbonne(Department of Developmental Biology, Washington University in Saint Louis, Saint Louis, Missouri, United States of America) in “A Fully Implantable Pacemaker for the Mouse: From Battery to Wireless Power”.
- [3] D. Radhakrishnan, “A microcontroller based pacemaker tester,” in Journal of Network and Computer Applications, Volume 19, Issue 4, October 1996.
- [4] Carlos Casillas(RTAC America, Guadalajara, Mexico), “Heart Rate Monitor and Electrocardiograph Fundamentals”.
- [5] Santosh Chede and Kishore Kulat, “Design Overview of Processor based Implantable Pacemaker,” Department of Electronics and Computer Science Engineering, Visvesvaraya National Institute of Technology, Nagpur, India.
- [6] Kityee Au-Yeung, Chad R Johnson and Patrick D Wolf, “A novel implantable cardiac telemetry system for studying atrial fibrillation”, Department of Biomedical Engineering, Duke University, Durham, NC 27708, USA.
- [7] Irena Jekova and Vessela Krasteva, “Real time detection of ventricular fibrillation and tachycardia”, Centre of Biomedical Engineering, Bulgarian Academy of Sciences, Acad. G. Bonchev str. bl. 105, 1113 Sofia, Bulgaria, 2004 Physiol. Meas. 25,1167

# Design and Implementation of Microcontroller-Based R-F Remote Controlled safe with Digital Code Lock and Indicator

BY

**Kenneth Nwachukwu-Nwokefor.C., Ogechi Ihekweaba, Emenike Ukeje.**

Computer Engineering Dept. Michael Okpara University of Agriculture, Umudike, Abia State Nigeria.

## **Abstract**

The paper work presents a brief history and evolution of key locks, as well as the various key types and key locks. Also, the basic concepts of telecommunications and modern digital systems are elucidated. The system architecture of a microcontroller-based radio frequency remote controlled lock and indicator is showcased with the attendant description of the operational details. The paper further describes the system implementation. Finally, tests were performed in stages and the results obtained were then used to conclude that an elegant Microcontroller-Based Code Lock System has been achieved.

**Keywords:** Microcontroller, Radio-Frequency, Key-locks, Digital systems, Code Lock.

## **Introduction**

In this present age of digital technology, the concept of a digital revolution carries the ramifications of paradigm shift from traditional to new ways of doing thing in real-time this has given rise to revolutionary trends that has orchestrated industrialization in the world today, and consequently improved the socio-economic, political and technological base of many countries and the world in general. Also, the recent technological advancement in management information systems and sharing of information across nations and amongst the international community has also reshaped our society to a great extent, this has no doubt been made possible by the efforts of various technologies advancements in micro-electronics, devices, machines and micro-controller-based systems with most at times computer interface which has yielded

an information technology society where a successful and dynamic relationship between engineering and societal needs meet.

After the industrial revolution, the world directed towards the information age with the development of computers during the 1950's. This was followed by the invention of Integrated Circuits (ICs) in the 1980's which led the electronics world to the peak by its compact size, weight, and cost with an attendant increased quality and reliability in real-time as well as the launching of communication satellite in 1962. The world converted into a global village such that we are not only able to use the products of engineering, but also have been challenged to study about existing technologies with a view to creating our own idea with a view to solving contemporary problems.

This work is aimed at considering the possibility of designing a Microcontroller-Based gate opener with Digital Code Lock through a Radio Frequency (RF) Remote Controller which can enhance Security in homes, offices, etc. This of course is an indispensable concept in industrial electronics. More so, The value of security to man is applicable to the facilities used by him. Most gates in industrial settings are fortified with automation. This microcontroller based project for safeguarding a house or office focuses on the use of RF remote to control the entrance and exit from a gate with an embedded personalized code lock.

## Brief Historical backgrounds of Locks

Security is the concern of people around the world. Beyond hiding object or constantly guarding them, the most frequently used option is to secure them with a device.

Locks are the most widely employed security devices. They are found on anything to which access must be controlled, such as vehicles, storage containers, doors, gates and windows. The security of any property or facility relies heavily on locking devices. Locks merely deter or delay entry and should be supplemented with other protection devices when a proper balance of physical security is needed. And assessment of all hardware including door frames and jams, should be included in any physical security survey. Locking devices vary greatly in appearance as well as function and application, as written in a book by Sandra Kay Miller (1996).

In 1778, English man Robert Barron patented a completely new kind of lock, the lever tumbler lock, which originally only uses two tumblers along with traditional wards – obstructions for key bit. The idea of building tumblers into the lock itself was completely revolutionary, and represented the first “modern” door lock. The Barron lock was manufactured completely by hand and was pricey for its day. Part of Barron lock was soon copied and was used and developed by competitors in his country.

Jeremiah Chubb of Portsmouth, England, invented a detected lock, a type of lever tumbler that instantly stopped working if the wrong key was used. Only the original key or a special regular key would activate the lock again. Chubb invented the lock in 1818 in response to a government – sponsored context to develop an unpick able lock. The Chubb lock had a spring –loaded bolt which was held in closed position by four oblong perforated brass lever tumblers. The tumblers were placed one on top of the other and locked on one end with a

dowel. The other end of the tumblers was lifted by the bit of the key, which was shaped like a sort of stair case, each step corresponding to one of the levers.

In 1824, Charles patented an improved version that didn't need a regulator key to reset the lock. The original lock had four lever tumblers,

Another locksmith, American Alfred Charles Hobbs, invented the protector lock. Hobbs became famous as the first man to be able to pick the Chubb detector lock at the 1851 world fair in London (The Great Exhibition), but in 1947 Charles and his son John increased the number to six. They later invented another device that made it impossible for a lock – picker to detect the additional tumbler. In 1984 Chubb and Sons Lock & safe Co Ltd became a part of the Racal Electronic group, making the Chubb lock group one of the largest in its field in the world. Another company, Williams Holdings, brought Chubb Security in 1997; but sold it on to in 2000 after poor sales figure.

In his book on locks, Allen Palms (1999) stated that, in the 15th to 16th century, sacristies were added on Swedish church on the north side of the building outside the sanctuary. They were built with sturdy stone walls and arched stone or brick ceilings, and the windows are protected by sturdy iron bars. Between the sanctuary and the sacristy was a heavy wooden door with wrought- iron hinges. It often had iron fittings or iron stripes riveted on. The door was locked with a wooden block lock, a smith tumbler lock, or in the 18th century, with a large rot iron rim lock.

A time lock is a part of a locking mechanism commonly found in bank vault and other high- security container. The lock is a timer designed to prevent the opening of the safe or vault until it reaches zero, even if the correct combination(s) are known. Most safes or vaults utilize at least two independent clock mechanism as a fail –safe system to guarantee the unlocking of the safe peradventure one of the timers (called movement) fails. Only one needs to reach zero in order for the safe to be opened. Time



locks can typically be set from 15minutes to 144hours (6days).

Time locks were originally created to prevent criminals from kidnapping and torturing the person(s) who knows the combination, and the using the extracted information to later burgle the safe of vault.

There is also the time delay combination lock which will open at anytime with the correct combination, but will not actually unlock until a set delay period elapses, usually less than one hour. Modern time locks are electronic but otherwise serve the same function as the old mechanical wind- up time lock, however, when the battery dies, the time lock unlocks.

### **Materials and Methods.**

This electronic system is designed with an 89C51 of the 8051 microcontroller family. An electromagnetic relay driven by a transistor static switch is interfaced to the microcontroller. Also four light emitting diodes (LEDs), red, yellow, green and blue serve as a form of indicators (LED indicator) when the remote is in use, presence of an RF signal, when the gate is open and closed respectively and an interfaced soft coding key pad.

A relay is an electrically operated switch. Many relays use an electromagnet to operate a switching mechanism mechanically, but other operating principles are also used. Relays are used where it is necessary to control a circuit by a low power signal (with complete electrical isolation between control and controlled circuits), or where several circuits must be controlled by be signal. The first relays were used in long distance telegraph circuits, repeating the signal coming in from one circuit and retransmitting it to another. Relays were used extensively in telephone exchanges and early computers to perform logical operations.

The system components were all assembled and placed together in modules. They were tested individually and then placed on a bread board. The system was tested, and when proved to be working, they were replaced on a Vero

board. The components were soldered on the Vero board using soldering lead and soldering iron. After the entire system had been tested it was packaged in a casing for better presentation.

A type of relay that can handle the high power required to directly drive an electric motor is called a contactor. Solid-state relays control power circuits with no moving parts, instead using a semiconductor device to perform switching. Relays with calibrated operating characteristics and sometimes multiple operating coils are used to protect electrical circuits from overload or faults; in modern electric power systems these functions are performed by digital instruments still called protective relays.

When electromagnetic radiation interacts with single atoms and molecules, its behavior also depends on the amount of energy per quantum (photon) it comes.

### **Basic Theory; Electromagnetic Spectrum**

The electromagnetic spectrum is the range of all possible frequencies of the electromagnetic radiation. The electromagnetic spectrum of an object is the characteristic distribution of electromagnetic radiation emitted or absorbed by that particular object. The electromagnetic spectrum extends from low frequencies used for modern radio to gamma radiation at the short wave-length end, covering wavelengths from thousands of kilometers down to a fraction of the size of an atom. The long wavelength limit is the size of the universe itself, while it is thought that the short wavelength limit is in the vicinity of the Planck length, In principle the spectrum is infinite and continuous.

Electromagnetic waves are typically described by any of the following three physical properties: the frequency  $f$ , wavelength  $\lambda$ , or photon energy  $E$ . The frequencies range from  $24 \times 10^{23}$  Hz (1 GeV gamma rays) down to the local plasma frequency of the ionized interstellar medium (-1 kHz). Wavelength is inversely proportional to the wave

frequency, so gamma rays have very short wavelengths that are fractions of the size of atoms, whereas wavelengths can be as long as the universe. Photon energy is directly proportional to the wave frequency, so gamma rays have the highest energy and radio waves have very low energy.

These relations are illustrated by the following equations:

$$f = c/\lambda, \text{ or } f = E/h, \text{ or } E = hc/\lambda \dots \dots \dots (1)$$

Where:  $c = 299.792,458$  m/s is the speed of light in vacuum and

$h = 6.62606896(33) \times 10^{-34}$  Js =  $4.13566733(10) \times 10^{-15}$  eV is Planck's constant.

Generally, electromagnetic radiation is classified by wavelength into radio wave, microwave, infrared, the visible region we perceive as light, ultraviolet. X- Rays and

Radio frequency (RF) , is a rate of oscillation in the range of about 30 kHz to 300 GHz, which corresponds o the frequency of electrical signals normally used to produce and detect radio waves. RF usually refers to electrical rather than mechanical oscillations, although mechanical oscillations do exist.

The communication is done through a resonator, a circuit with a capacitor and an inductor forming a tuned circuit. The resonator amplifies the oscillations within a particular frequency band, while reducing oscillations at other frequencies outside the band. The conductor or the capacitor of the tuned circuit is adjustable allowing the user to change the frequencies at which it resonates. The resonant frequency of a tuned circuit is given by the formula

$$f_0 = 1/2\pi\sqrt{LC} \dots \dots \dots (2)$$

Where  $f_0$  is the frequency in Hertz, L is inductance in henries, and C is capacitance in farads.

### Special Properties of RF Electrical Signals

An electrical current that oscillates at RF has unique characteristics, one of such property is the ability to ionize air thus, creating a conductive path through it. This property is exploited by 'high frequency' units used in electric arc welding, although strictly speaking, these machines do not typically employ frequencies within the HF band. Another special property is that RF current cannot penetrate deeply into electrical conductors but flows along the surface of conductors; this is known as the skin effect. Another property is the ability to appear to flow through paths that contain insulating materials, like the dielectric insulator of a conductor. The degree of effects of these properties depends on the frequency of the signals.

### Bands

The band is a small section of the spectrum of radio communication frequencies, in which channels are usually used or set aside for the same purpose. Above 300 GHz, the absorption of electromagnetic radiation by earth's atmosphere is so great that the atmosphere is effectively opaque, until it becomes transparent again in the infrared and optical window frequency ranges.

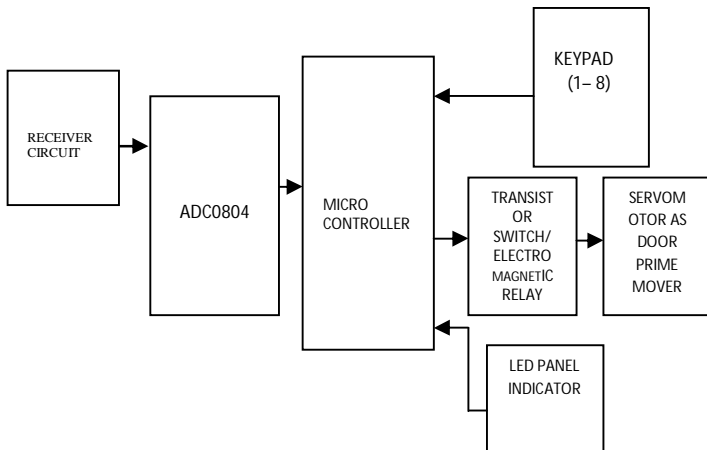
To prevent interfaces and allow for efficient use of the radio spectrum, similar services are allocated in bands. Each of these bands has a basic band-plan which dictates how it is to be used and shared, to avoid interfaces and to set protocol for the compatibility of transmitters and receiver.

Bands are divided at wavelengths of  $10^{\text{th}}$  meters, or frequencies of  $3 \times 10^{11}$  hertz.

**Table 1. Radio Frequency Bands.**

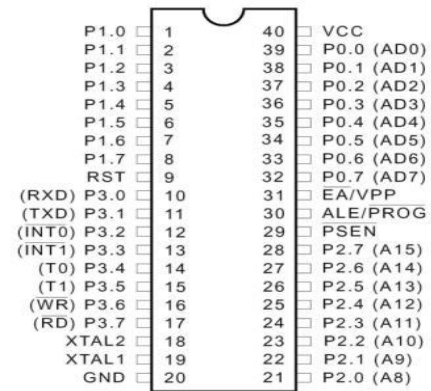
BAND NAME	ITU BAND	FREQUENCY AND WAVELENGTH IN AIR
Sub-hertz	0	<3 Hz > 10,000km
Extremely low frequencies	1	3 - 30 HZ 100,000 km – 10,000 km
Super low frequency	2	30 - 300 HZ 10,000 km – 1000 km
Ultra low frequency	3	300 - 3000 HZ 1000 km – 100 km
Very low frequency	4	3 - 30 kHz 1000 km – 100 km
Low frequency	5	30 - 300 kHz 10 km – 1 km
Medium frequency	6	300 – 3000 kHz 1 km – 100 m
High frequency	7	3 - 30 MHz 10m – 1m
Very high frequency	8	30 - 300 MHz 10m – 1m
Ultra high frequency	9	300 – 3000 MHz 100mm – 10mm
Super high frequency	10	3 – 30 GHz 100mm – 10mm
Extremely high frequency	11	30 – 300 GHz 10mm – 1mm
Terahertz	12	300 – 3000 GHz 1mm - 100µm

**SYSTEMS SCHEMANTIC BLOCK DIAGRAM**



**FIG.1, The System Schematic Block Diagram.**

**Basic Features of a Microcontoller Layout.**



**FIG.2, MICROCONTROLLER PIN LAYOUTS**

A microcontroller can be considered a self- contained system with a processor, memory and peripherals and can be used as an embedded system. The majority of microcontroller in used today is embedded in other machinery, such as automobiles, telephones, appliances and peripherals for computer systems. These are called embedded systems. While some embedded systems are very sophisticated, many have minimal requirements for memory and program length, with no operating system, and low software complexity. Typical input and output devices include switches, relays, solenoids, LEDs, small or custom LCD displays, radio frequency devices, and sensors for data such as temperature, humidity, light level etc. Embedded systems usually have no keyboard, screen, disks, printers, or other recognizable I/O devices of a personal computer, and may lack human interaction devices of any kind.

**Programs:** Microcontroller programs must fit in the available on –chip program memory, since it would be costly to provide a system with external, expandable, memory. Compilers and assemblers are used to convert high- level language codes into a compact machine code for storage in the microcontroller’s memory. Depending on the device, the program memory may be permanent, read –only memory that can be programmed at the factory, or program may be field- alterable flash or erasable read – only memory.

**Interrupt:** microcontrollers must provide real time (predictable, though not necessarily fast) response to events in the embedded system they are controlling. When certain events occur, an interrupt system can signal the processor to suspend processing the current instruction sequence and to begin an interrupt service routine (ISR, or “interrupt handler”). The ISR will perform any processing required based on the source of the interrupt before returning to the original instruction sequence. Possible interrupt sources are device dependent, and often include events such as an internal timer overflow, completing an analog to digital conversion, a logic level change on an input such as from a button be pressed, and data received on a communication link. Where power consumption is important as in battery operated devices, interrupts may also wake a microcontroller from a low power sleep state where the processor is halted until required to do something by a peripheral event. Etc.

**Systems Circuit Diagram**

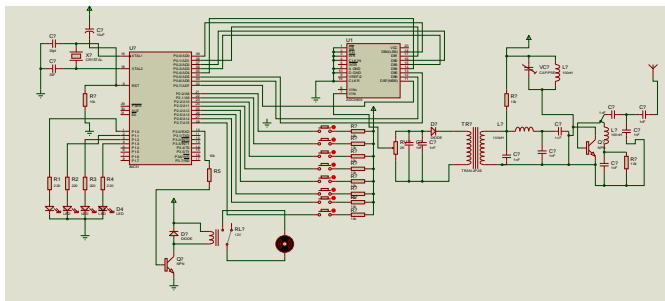


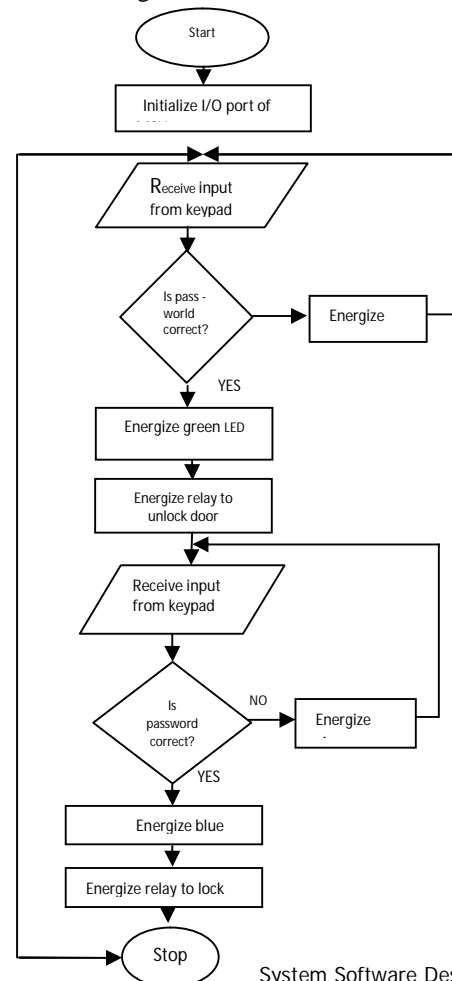
FIG.3, SYSTEMS CIRCUIT DIAGRAM.

**System Design and Implementation**

The microcontroller based RF remote controller gate opener with digital code lock is designed with a digital keypad introduced at port two of the microcontroller. The keypad is designed to communicate with the gate. The pull up resistors, R1 to R8 ensure that the switches of the keypad do not float. A power reset realized using RC network is connected to the pin 9 of the microcontroller to enhance automatic reset. Pins 18 and 19 of the microcontroller is connected to a 12MHz quart crystal in

order to avoid frequency drift, that is, t ensures that there is frequency stability at the microcontroller where the controlling software is stored is designed to read the input data in the form of a handshake from the keypad.

When a correct password is entered through the keypad, the data is fed into the microcontroller and the RAM writes the data in the form of machine language which is then outputted. An electromagnetic relay is connected at the common collector of the transistor, such that whenever the connection is driven to saturation, the relay is energized and the normally closed contact of the relay opens while the normally open contact closes. When this happens, the gate unlocks. This electromagnetic relay is connected to drive the servomotor which energizes the gate. The servomotor must confirm to the following; 12v AC maximum of 3Amp and coil resistance of 600ohms and 300watt maximum power output. A green LED at he indicator panel glows when the correct password is entered on the keypad to unlock the gate while a blue light glows to lock the gate.



System Software Design Flow Chart

### System Tests and Result.

Each unit was tested individually using a multi-meter. Some malfunctioned and were replaced with those of appropriate values.

After assembling each module in the breadboard, a visual inspection was carried out. Each module was re- tested before being transferred to the Vero board. The components were then soldered and the Vero board re-tested.

The system was re- inspected to ensure that the wired circuit was properly built before powering the system (continuity test). After the system was coupled, it was tested using a power source and it worked in accordance with the system design objective. That is, the red light glows to signify power, the yellow light glows when there is the presence of an Radio Frequency signal, the green light glows when the gate is open, while the blue comes up when the gate it is locked.

### System Maintenance Considerations

Every working system must be carefully monitored to ensure that its initial objectives are still being met.

Maintenance, therefore should involve all activities and precautions taken to ensure optimal performance of a system. It also involves correction of faults or errors created in the cause of using the system

To prevent system failure, it is necessary to advice on the following;

- Ensure that a system operated under specification
- Ensure proper training of persons that are to use the system
- Avoid contact with liquids which could cause the system to malfunction.
- Keep a maintenance log book as this will help to keep track of possible faults or problems that arise as the system is being used, in other to assist future improvements.

The reasons that satisfy the need for system evaluation and maintenance include;

- To confirm that the initial objectives of the system are constantly being met.
- To deal with unforeseen problems arising from operation.
- To ensure that the system is able to cope with changing requirements of its operational site.

### Suggestions for Further Improvements.

The system can be improved upon by applying some Artificial Intelligence Techniques to make it a more complex and robust and result oriented system , thereby enhancing its capacity to perform the various set functions in real-time.

### Summary and Conclusion

This project was actualized by the design, construction and implementation of a microcontroller- based RF remote control gate opener with digital code lock. It should be noted that the system has been designed in such a way that it can be modified with respect to change in the future, as regard technological growth.

Furthermore, it is evident to say that the project so conceptualized, designed and produced has shown the effectiveness and efficiency in improving comfort and security, conserving energy and most especially saving time.

Also it is worthy to note that the work pattern of any organization may be affected by the introduction of a new system. The change from an old system to a new one is usually not easy until the detail of the essence of the new system is properly communicated to the users within the environment.

### References

- Alan Muinford (1993): The IT Manager: Security And Training, Prentice-Hall, Inc. New Jersey.
- Alan Pritchard (1997) A Guide To Computer Literature: An Introductory Survey Of The Sources Of Information, Clive Bingley Ltd, United kingdom.

- Allen Palms (1999): Lock: Locks and locked, Time square, United Kingdom.
- Anderson, F. (1998): Computer In Agriculture, Acta-Hall, United kingdom.
- Assa Abloy (1997): Door Opening Solutions, Whispering fuse, Maryland.
- Baker C, R. (1999): An Analysis Of Fraud On Locking Devices, Spectra Inc, New York.
- Boxus, P. (1998): Multiple-Cycle Micro propagation Control, Specks Publishers, London.
- Carter G.N (1963): An Input-Output Analysis Of The Nigerian Economy Cambridge press, London.
- Dennis Roddy and John Coolen (2002): The Small World Of Microcontrol, Computer Prints, New York.
- Freeman, R. and Meed J. (1993): How To Study Effectively, Collins Educational, London.
- Irving Gottlied (1973): Basic Electronics Procedures, Foulsham-Tab Limited, USA.
- Jeffrey Whirtten, and Lanni Bentley, (2001): System Analysis And Design, (Fifth Edition), Mc Graw Hill Publisher, New York.
- Kevin Dittmam (1987): Dictionary Of Computer Information Processing Telecommunication (second Edition), Mc-Graw Hill Inc.
- Loveday, R. (1964): A First Course In Statistics, Newman Books, New York.
- Michaelson, H. B. (1990): How To Write And Publish Research Engineerring Paper And Reports, (Third Edition): Phoenix Az, Oryx.
- Morris Mano and Charles Kime (1991): Logic And Computer Design Fundamentals, Space Design Inc. Arkansas.
- Osy Igweonyia (2002): Standard Methods In Business Research (Millennium Edition), New Generation Books, Enugu.
- Sandra kay Miller (1996): How to analyze security Needs 2010 Penton Media, Inc.  
<http://en.wikipedia.org/wiki/time.lock>

# Wireless Analyzer of ISOBUS

Enkhzul Doopalam<sup>1,2</sup>, Luubaatar Badarch<sup>3</sup>, Amartuvshin Togooch<sup>2</sup>, Enkhbaatar Tumenjargal<sup>2</sup>,  
Woonchul Ham<sup>3</sup>, Kahng Hyun Kook<sup>1</sup>

<sup>1</sup>Electronic and information Department, Korea University, Room 342 Bio-Technology Building, 136-701, Anam-dong 5-ga, Seongbuk-gu, Seoul, Republic of Korea

<sup>2</sup>School of Information and Communications Technology, Mongolian University of Science and Technology, Ulaanbaatar, Mongolia

<sup>3</sup>Chonbuk National University, Chollabuk-do, Jeonju, South Korea

**Abstract**— *Communication between ECUs (Electronic Control Units) in agricultural machineries tends to use ISO11783 widely, that is PGN (Parameter Group Number) based communication protocol lays on CAN protocol by altering its identifier part. Messages in line are transferred and received between ECUs according to ISO11783 standard. This paper discusses about design of wireless monitoring system. We used an ARM Cortex-M3 core microcontroller embedded development board and marvel8686 wireless module. The wireless ISOBUS monitoring system, attached into communication line, reads messages interpret them, and display them on the screen in easily comprehensible form. It can be used to generate messages and monitor the traffic on physical bus systems. The monitoring system connected to ECUs, monitor and simulate real traffic of communication and functionality of the ECUs. In fact, in order to support our work, we have implemented the monitoring tool. The development consists of two parts: GUI of the application and firmware level programming. Hence the monitoring system is attached to the communication line and equipped by wifi module; farmer/dispatcher in a farm monitors all messages in communication line on personal computer and smart device.*

**Keywords**— ISO 11783, PGN, CAN, wifi, ARM Cortex-M3.

## 1 Introduction

Agricultural machinery control is an interdisciplinary field of study concerning the integration of mechanics, electronics, and software engineering expertise. Today a new generation of tractors exists with capabilities so advanced they can be assumed in many of the roles and responsibilities once entrusted to their human counterparts. This evolution in tractors is the direct result of continuing research advancements among its constituent disciplines. The ISO 11783 [11]~[16][17] standard has and, continues to be, an active area of research within the agricultural engineering community.

The ISO 11783 standard was jointly developed by tractor and implements manufacturers including the AGCO Corporation [18], AGROCOM [19], DICKEY-Jonh Corporation [20], Deere & Company [21], and Müller-Elektronik [22]. These manufacturers have also created a specification defining how this standard should be recognized. This specification is commonly known as ISOBUS. All packets, except for the request PGN and address claim packets, contain eight bytes of data and

standard header which contains an index called parameter group number (PGN), which is embedded in the CAN message's 29-bit identifier [1-3]. A PGN identifies a message's function and associated data [4-6]. To implement and develop the networked tractor system we need to analyze and control all messages in communication line. For this purpose we implemented ISOBUS PGN analyzer in the previous work. When a PGN analyzer reads data from the ISOBUS it needs to know how to interpret what it is seeing and display the output in an easy to read format. It can be used to generate and monitor the traffic on physical bus systems. In order to support our work, we have implemented the PGN analyzer tool in personal computer. To advance our PGN analyzer here we developed web based application in STM32F103 development board with wireless module for smart devices. The general architecture of our development system is shown in figure 1.

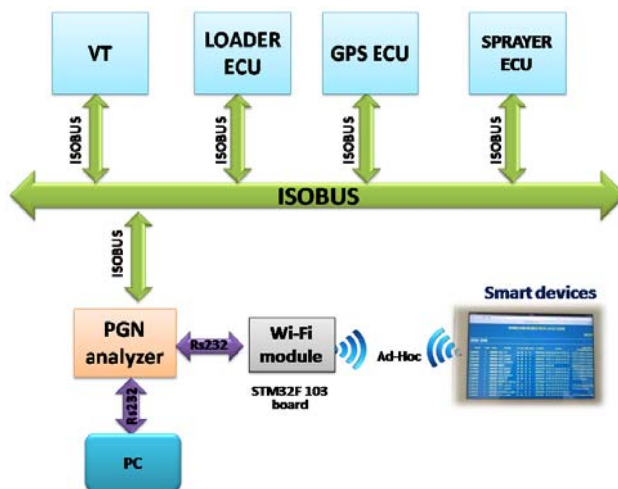


Figure 1. General architecture of development system

Development of analyzer consists of three parts: GUI of application in personal computer, web based application in development board and firmware level programming.

## 2 Hardware design of ISOBUS PGN analyzer

The PGN analyzer is implemented on the embedded board where main CPU is 32 bit, 72MHz CortexM3 cored STM32F107 development board with two CAN interfaces. Once development board of analyzer has two

serial interfaces we use one serial interface for the communication between the analyzer and application program in PC and communication between analyzer and wi-fi development board for smart devices. The CAN1 channel is used to monitor the ISOBUS. However, CAN2 channel can be used as well, if there is need of monitoring two ISOBUSs. The status information of the PGN analyzer is depicted on the LCD display of the board. We can see the appearance of the PGN analyzer in the Fig. 2. Here ISOBUS connected PGN analyzer and sample ECUs are depicted. We implemented sample ECUs, for example: GPS sensor, lighting, sprayer, tractor ECU and VT.

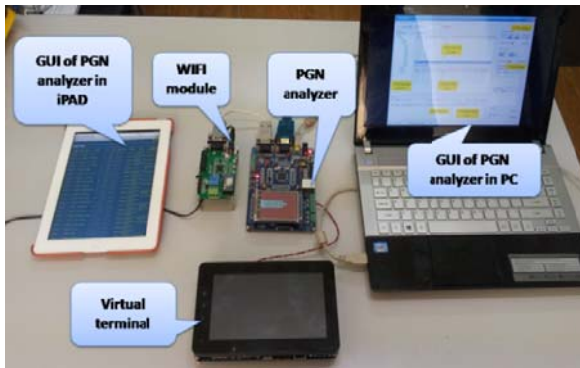


Figure 2. PGN analyzer connected with ISOBUS ECUs

### 3 Firmware level programming

The hardware programming is implemented in the firmware level. The Fig. 3 shows the main structure of firmware level program. PGN analyzer's firmware program has been five main functions:

- Receive data from the personal computer via RS232 and the ISOBUS via CAN[1] interface
- Processing received data both RS232 and CAN interface
- Processed data send to the personal computer via RS232 and the ISOBUS via CAN interface
- Processed data send to the wi-fi development board via RS232 interface
- Wifi development board display processed data in web based application for smart devices

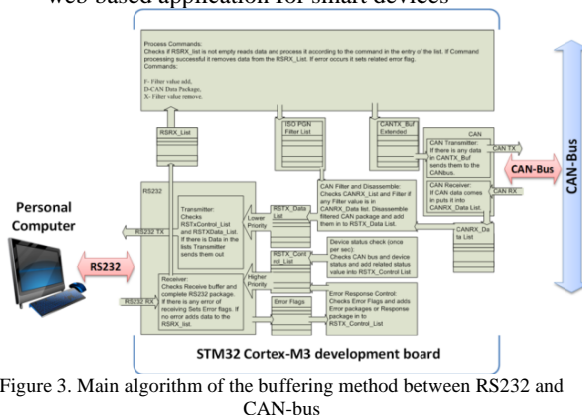


Figure 3. Main algorithm of the buffering method between RS232 and CAN-bus

Receiving part recognizes and accumulates RS232 data into one CAN packet. Application program and firmware program has the predefined structure of data sequence to send and receive CAN package, control and status data.

Receiving part of the firmware program's RS232 data sequence recognition and accumulation process is shown in the Fig.4.

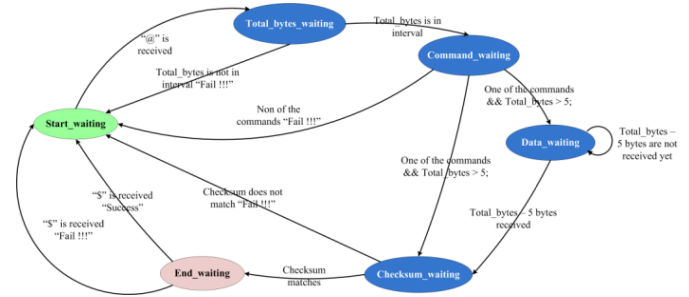


Figure 4. State machine diagram of PGN analyzer

There are six steps, we can see from the Fig.4; start waiting, total bytes waiting, command waiting, data waiting, checksum waiting and end waiting.

During RS232 data sequence receiving function, the Cortex-M3 CPU monitors via SYSTICK time counting value, in order to recognize data loss from the PC. The waiting time is 200 ms, which is allowed to wait for the next expectable byte. If there no byte is received anymore in 200ms, receiving status shifts in to *start\_waiting* state and losses all the in complete bytes that are received yet. The CAN receiving part of the firmware level programs receives CAN packet from the CAN1 peripheral of the STM32F107 and check it that the CAN packet is allowed to be sent to the PC by filtering with the PGN values in the filter list.

If the packet is allowed, it repackaged into RS232 data sequence to be sent to the PC: adding head and tail of the sequence ('@' and '\$'), command ('T'), number of total bytes and checksum.

Another function of the firmware level programming is to inform the status of the hardware in to the application program via RS232 per second. It sends the status with the command 'L' (device is alive-status) per second. Therefore, application program can easily recognize whether the hardware is functioning or not at moment.

### 4 Graphical User Interface of PGN Analyzer

There are two graphic user interfaces. One is dedicated for the personal computer which is implemented in the Borland Delphi 7.0[10] environment with the object oriented Pascal language. Another GUI is for smart or mobile device which is developed in HTML5. Delphi it is effective to develop this kind of system, because it has rapid application development (RAD) interface and good tool to help against the



developing mistakes which cause understanding problems.

The GUI of PGN analyzer program's backbone some application programs running for parsing RS232 serial data to convert CAN packet and reverse procedure. We used the CAN server, CAN messenger, and CAN logalizer console application tools for parsing CAN packets, those original made from the IsoAglib. In this parsing procedure using named pipes to connect a GUI to a console application in windows environment.

Named pipes allow two processes to share data bi-directionally synchronously or asynchronously. This makes them ideal for putting a GUI front-end on a console application. Software developers usually provide a rich GUI to easily run an application in a desktop environment. However, users sometimes prefer a console application that typically provides access to the fullest range of application options via console interface and may be composed into a pipeline of user interface-less processing steps. The Fig. 5 illustrates received CAN packet parsing procedure until GUI. We capture serial data from RS232 interface then convert to CAN messenger understandable format.

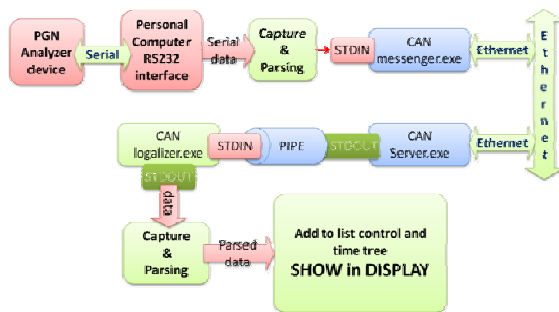


Figure 5. Received CAN packet parsing procedure shown in GUI

The CAN messenger send to CAN server by using Ethernet network. The CAN server console application program's standard output connected the virtual pipe. The pipe connected to CAN logalizer[8] console application's standard input. The logalizer analysis standard input data and interpreting user understandable format. The final those interpreted data parsed then add to list control and tree control.

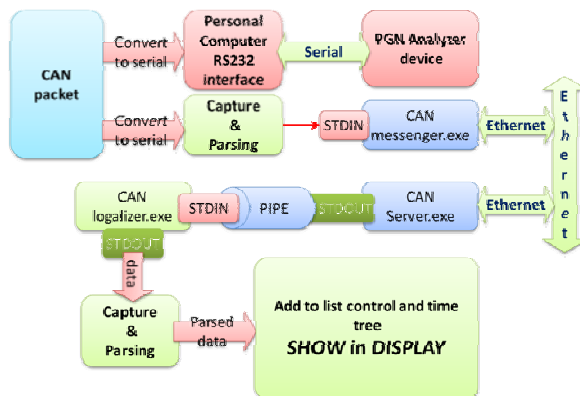


Figure 6. Transmitted CAN packet shown in GUI

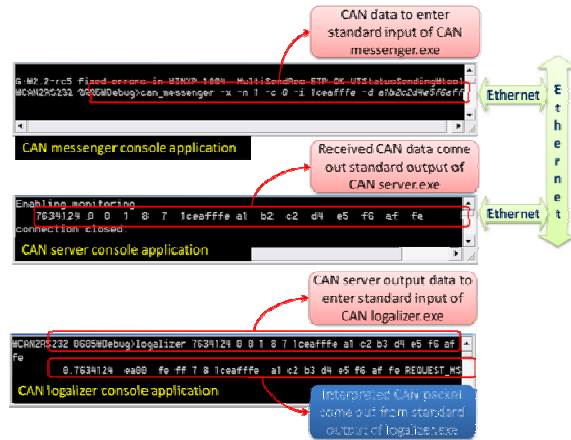


Figure 7. CAN packet parsing and interpreting procedure in the command line

Transmitting procedure is similarly the receiving in GUI. The Fig.6 and 7 are shown in the block diagram of transmission procedure for PGN analyzer. The sample received CAN packets interprets is shown in the Fig.8.

Time	PGN	Src	Dest	Prio	Disc	ID	D0	D1	D2	D3	D4	D5	D6	D7	ISO11783 Interpretator
0.0003234	00F	00	FF	2	0	0000100	00	32	b1	1c	57	8a	e3	00	NMEA_GPRS_POSITL...
0.0003402	00F	00	FF	2	0	0000100	00	32	b1	1c	57	8a	e3	00	NMEA_GPRS_POSITL...
0.0003569	00F	00	FF	2	0	0000100	00	32	b1	1c	57	8a	e3	00	NMEA_GPRS_POSITL...
0.0003734	00F	00	FF	2	0	0000100	00	32	b1	1c	57	8a	e3	00	NMEA_GPRS_POSITL...
0.0003902	00F	00	FF	2	0	0000100	4c	31	b1	1c	75	e7	e2	00	NMEA_GPRS_POSITL...
0.0004070	00F	00	FF	2	0	0000100	4c	31	b1	1c	75	e7	e2	00	NMEA_GPRS_POSITL...
0.0004238	00F	00	FF	2	0	0000100	4c	31	b1	1c	75	e7	e2	00	NMEA_GPRS_POSITL...
0.0004406	00F	00	FF	2	0	0000100	4c	31	b1	1c	75	e7	e2	00	NMEA_GPRS_POSITL...
0.0004574	00F	00	FF	2	0	0000100	4c	31	b1	1c	75	e7	e2	00	NMEA_GPRS_POSITL...
0.0004742	00F	00	FF	2	0	0000100	4c	31	b1	1c	75	e7	e2	00	NMEA_GPRS_POSITL...
0.0004910	00F	00	FF	2	0	0000100	4c	31	b1	1c	75	e7	e2	00	NMEA_GPRS_POSITL...

Figure 8. Sample captured data from our PGN analyzer device

The GUI software of personal computer is written in object oriented Pascal only runs under the Windows operational system. The web based GUI is written in HTML5. The main interfaces for the user are shown in the Fig. 9 and Fig 10.

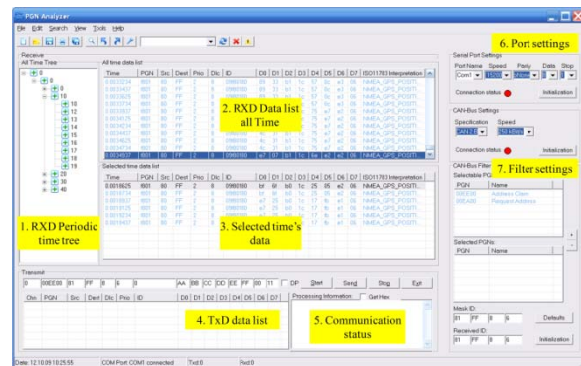


Figure 9. GUI of the PGN Analyzer



Figure 10. Web based GUI of the PGN Analyzer

## 5 Conclusions

In this paper we presents the hardware and software development of wireless ISO11783 parameter group number (PGN) analyzer device that is implemented in STM32F107VC Cortex-M3 development board with the Wi-Fi module.

In programming of ISO 11783 PGN analyzer, we focused on both of firmware programming and GUI on the monitoring computer and smart devices. The main role of the firmware programming is capturing CAN packet and converting to RS232 serial data format or receiving RS232 data from computer, converting it into CAN data and sending. GUI of PGN Analyzer receives RS232 data and converting to CAN packets in order to monitor them. Those converted CAN packets send to some of hidden application programs, by using one useful programming technique pipe. We used two virtual pipes for parsing and interpreting CAN packets. Finally parsed and interpreted data is shown on GUI of PGN analyzer in ISO11783 standard form. Even all converting, parsing and interpreting procedures are simultaneously made in two development boards, final displaying GUIs are different. GUIs are written by the Delphi visual programming language and HTML5. Hence our web based application is written in HTML5 which has a good opportunity to support other mobile devices.

## 6 REFERENCES

- [1] Robert Bosch, GmbH, "CAN specification," *Germany*, 1991.
- [2] E. Tumenjargal, L. Badarch, H. Kwon, and W. Ham, "Embedded software and hardware implementation system for a human machine interface based on ISOAgLib," *Journal of Zhejiang University-Science C-Computers & Electronics*, vol. 14, pp. 155-166, Mar 2013.
- [3] K. Hyeokjae, T. Enkhbaatar, and H. Woonchul, "Implementation of Virtual Terminal Based on CAN by Using WinCE Platform Builder 6.0," *Key Engineering Materials*, vol. 480, pp. 938-943, 2011.
- [4] W. Ham, T. Enkhbaatar, B. Luubaatar and K. Hyeokjae, "Implementation of ECU for Agricultural Machines Based on ISOAgLib Open Source", presented at the 11th Int. Conf. Precision Agriculture, Indianapolis, Indiana, USA, Jul. 15-18, 2012.
- [5] T.Enkhbaatar, B.Luubaatar, K.Hyeokjae and W.Ham, "Design and Implementation of Virtual Terminal Based on ISO11783 Standard for Agricultural Tractors," presented at the 11th Int. Conf. Precision Agriculture, Indianapolis, Indiana, USA, Jul. 15-18, 2012.
- [6] E. Tumenjargal, L. Badarch, K. Hyeokjae, and W. Ham, "Software Development Tool for Agricultural Machinery Based on IsoAgLib Open Source Library," presented at the 2012 ASABE Annual Meeting, Dallas, Texas, USA, Jul. 29 – August 1, 2012.
- [7] Md. M. K. Sarker, D. Park, W. Ham, E. Tumenjargal, and J. Lee, "Embedded Workbench Applications of GPS Sensor for Agricultural Tractor," presented at the WorldComp'12, Las Vegas, Nevada, USA, Jul. 16-19, 2012.
- [8] A. Spangler and M. Wodok, "IsoAgLib—Development of ISO 11783 Applications in an Object Oriented way," ed, 2010.
- [9] S. Ingle, S. Dessai, and R. Gore, "Development of Software for CANlog Device to Determine the Performance of Tractor", *International Journal of Recent Trends in Engineering*, Vol. 1, No. 3, May, 2009.
- [10] G. Craessaerts, K. Maertens, and J. De Baerdemaeker, "A Windows-based design environment for combine automation via CANbus", *Journal of Computers and Electronics in Agriculture*, pp. 233–245, 2005.
- [11] ISO 11783-1, "Tractors and machinery for agriculture and forestry - Serial control and communications data network," Part 1: General standard for mobile data communication, International Organization for Standardization, 2007.
- [12] ISO 11783-2, "Tractors and machinery for agriculture and forestry - Serial control and communications data network," Part 2: Physical layer, International Organization for Standardization, 2002.
- [13] ISO 11783-3, "Tractors and machinery for agriculture and forestry - Serial control and communications data network," Part 3: Data link layer, International Organization for Standardization, 2007.
- [14] ISO 11783-4, "Tractors and machinery for agriculture and forestry - Serial control and communications data network," Part 4: Network layer, International Organization for Standardization, 2001.
- [15] ISO 11783-5, "Tractors and machinery for agriculture and forestry - Serial control and communications data network," Part 5: Network management, International Organization for Standardization, 2007.
- [16] ISO 11783-6, "Tractors and machinery for agriculture and forestry - Serial control and communications data network," Part 6: Virtual terminal, International Organization for Standardization, 2004.
- [17] P. Fellmeth, "CAN-based tractor-agricultural implement communication ISO 11783," *CAN Newsletter*, vol. 9, 2003.
- [18] AGCO Corporation. FieldStar, the Science of Agriculture, Virtual Terminal User's Guide. Publication No. 79015206 (English), February 2002, Duluth, GA.
- [19] AGRCOM Gmbh and Agrarsystem KG. CEBIS MOBILE VA User Guide. Manual (English). 2009, Bielefeld, Germany.
- [20] DICKEY-John Corporation. Auto Section Control System, Operator's manual. Publication No. 11001-1561B-201207 (English). 2012, Auburn, IL, USA.
- [21] Deere and Company. GreenStar 3 Display 2630 operator's manual. Publication No. OMPFP12408 (English). 2012, California, USA.
- [22] Müller-Elektronik GmbH. ISOBUS-Terminals flexible and future-proof through APP & GO. 10/11. 2012, Salzkotten, Germany.