Performance and Energy Evaluation of ARM Cortex Variants for Smart Cardiac Pacemaker Application

Safwat Mostafa Noor and Eugene John
Department of Electrical and Computer Engineering,
University of Texas at San Antonio, San Antonio, TX, USA

Abstract - *Embedding microprocessors in implantable devices such as cardiac pacemakers improved their ability to treat complex heart conditions effectively. Future cardiac pacemakers are expected to evolve in features, gaining secure wireless connectivity, longer battery life, and increased operational reliability. Implementing such features in a power constrained pacemaker requires a deep understanding of the power consumption behavior of the underlying processor, especially for computing the expected workloads. In this paper, the popular ARM Cortex series of processors are evaluated against anticipated future workloads of a smart cardiac pacemaker. Simulation results are analyzed to understand the tradeoffs in instruction set design and the importance of a dedicated floating point calculation unit. The simulation results are backed by data collected from the execution of the programs on an actual Cortex-M4 processor with a floating point unit. The instantaneous power consumed by the processor is monitored, and possible improvement techniques are discussed. Execution time and total energy per operation are summarized to conclude the feasibility of existing embedded processors for future cardiac pacemaker application.*

Keywords: ARM, Thumb, Cardiac Pacemaker, Pacemaker Security, STM32

1 Introduction

The use of cardiac pacemakers for the treatment of common heart diseases, such as arrhythmia, increased by 55.6% between the years 1993 to 2009 with approximately 2.9 million patients receiving a permanent pacemaker implant [1]. A pacemaker monitors the cardiac signals of the heart, determines the need for artificial pacing and generates electric impulses to synchronize the heart's rhythm. The technology used in cardiac pacemakers matured over the years in ensuring efficient and reliable operation of its primary functionality. The introduction of microprocessors in cardiac pacemakers in the 1980s [2] made it possible to achieve programmable operating behavior and configurable pacing logic for individual patients; significantly improving the effectiveness of this treatment method. Present day cardiac pacemakers are carefully crafted embedded systems, typically hosting ASIC (Application Specific Integrated Circuit) components, low power processing units, analog filters and charge pump circuitry and a low self-discharge battery [3] as the source of energy. With the help of recent technological advancements, pacemaker manufacturers can equip modern pacemakers with advanced features such as adaptive pacing [2], ultra-low power operation for longer battery life (7-10 years) [3] and wireless telemetry for programming and monitoring [4].

Future cardiac pacemakers are expected to push the boundaries of low-power embedded system design and take advantage of the ubiquitous wireless connectivity present around the patient and in the hospital's infrastructure. Similar to existing wearable devices, future pacemakers can benefit immensely from having the ability to communicate with the patient's smartphone via Bluetooth Low Energy (BLE) or connect to other low power wireless networks. One can imagine a scenario, where a pacemaker can be programmed to transmit alerts automatically via the connected smartphone or low power network when an emergency occurs. This technology can also be used to enable remote diagnosis and treatment. However, the integration of smart computing and connectivity into such a critical application introduces security concerns and power consumption challenges. Secure communication is a serious demand for next generation pacemakers due to vulnerabilities and risks that are currently present [5]. The increase in power consumption from the addition of new features also appears as an obstacle to fulfilling the requirement of a long battery life. A wirelessly connected pacemaker will require higher degrees of security, combined with extreme power efficiency. The energy footprint of both existing and anticipated features must conform to the current power budget of the cardiac pacemaker's battery to ensure a sustainable move forward.

This research attempts to determine the potential workloads in a future cardiac pacemaker and evaluate the performance of these workloads on popular embedded processors, namely the ARM Cortex variants. Utilizing architectural simulation, the computational requirements for each benchmark program are analyzed, and the appropriateness of existing ISAs (Instruction Set Architecture) are studied. The analysis is further authenticated by utilizing an MCU (Micro-Controller Unit) development board to measure the energy consumption of representative programs to evaluate real world performance of the ISA.

2 Pacemaker workloads and Processors

2.1 Heart Signal Processing

The human heart contracts and expands in a specific sequence periodically to distribute blood to the body and lungs.

This movement consists of two steps called the *"diastole"* and *"systole"*. In these two stages, electric impulses are generated and sent to the heart's myocardium muscles via special conduction fibers. These impulses are the cardiac signals that are typically monitored in an ECG (Electrocardiogram). Cardiac signals which are monitored by sensing leads inserted in the heart are called IECG (Intra-cardiac Electrocardiogram). IECG usually contains noise due to muscle activity and physical movement of the lead [2]. This signal is filtered and sampled by dedicated circuitry and then digitally processed to detect the fundamental features of the signals associated with the heart's diastole and systole stages. A complete ECG cycle consists of P, Q, R, S and T components and are visualized in Figure 1.

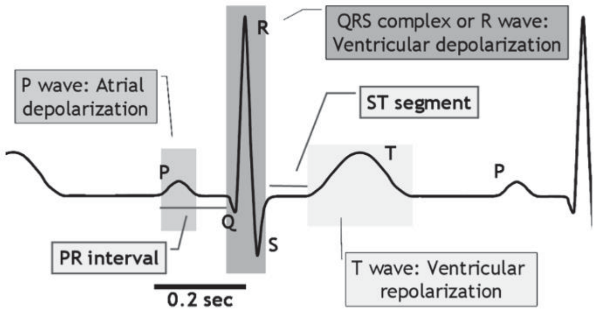

QRS complex or R wave: Ventricular depolarization
R
P wave: Atrial depolarization
ST segment
T
P
P
Q
S
PR interval
T wave: Ventricular repolarization
0.2 sec

Figure 1: Illustration of a typical electrocardiogram [6]

The nature of an ECG signal has been thoroughly studied and is typically found to have an amplitude in the range of *2mV* peak-to-peak and a bandwidth of *0.05Hz-150Hz* [6]. To accurately determine the existence and occurrence time of a P wave, QRS complex, ST segment and T wave, the time-frequency component of the signal needs to be extracted via a frequency domain analysis. Common signal processing techniques for this purpose include Fourier transform or Short Time Fourier Transform [6]. Detection of an abnormality in the electrocardiogram after extracting the frequency information is trivial compared to the computation involved in the Fourier transform of a large sample size. The Fast Fourier Transform (FFT) is an efficient algorithm for this task and is used as one of the benchmark programs in this paper. The *"FFT"* program is collected from the *MiBench* embedded benchmark suite [7] and a large sample size (8192) is used to examine the performance of the simulated processors. To cover other mathematical operations that might be a part of the detection process, the *"basicmath"* benchmark from *MiBench* is also selected. The *basicmath* program performs a series of common mathematical operation. To simulate the process of generating an artificial pacing signal, the *"ECGSYN"* benchmark program from *ImplantBench* [8] is used in this research. The *ECGSYN* program generates a synthesized ECG signal which can be utilized by the pacemaker's processor to determine the pacing amplitude and duration.

2.2 Security and Reliability

Security is one of the prime concerns in any wireless communication. When wireless functionality is introduced in a cardiac pacemaker, new life-threatening risks emerges. Exploitable vulnerabilities have been demonstrated by researchers [5] in multiple present day wireless cardiac pacemakers which relied on proprietary encryption mechanisms for securing their wireless data transmission. To eliminate such risks, the adoption of industry standard security schemes used in TLS (Transport Layer Security) is desirable. Unfortunately, such computational load requires feasibility study in the power constraint environment of a pacemaker. The *AES* (Advanced Encryption Standard) used by TLS for securing communication data packets is a computationally demanding task and is studied in this research as a benchmark.

To ensure the reliability of transmitted and received data, error checking hash functions are typically used. For a wirelessly connected pacemaker, cryptographic hash functions such as *SHA* (Secure Hash Algorithm) and *CRC* (Cyclic Redundancy Check) can be employed to ensure the reliability of the critical configuration and transmitted data. The list of simulations, therefore, includes benchmark programs for 32 bit *CRC* and *SHA* function as well. The programs representing *AES*, *SHA* and *CRC* are named *"rijndael"*, *"SHA"*, and *"CRC32"* respectively and are all collected from the *MiBench* suite.

2.3 ARM Cortex Processors

ARM Cortex processors are popular choices for embedded systems ranging from high-performance applications to low-power battery operated deeply embedded systems. The Cortex range mainly has three variants, the Cortex-A series, Cortex-R series and Cortex-M series. The major differences between these variants are shown in Table 1.

Table 1. ARM Cortex series comparison [11]

Features	Cortex-A	Cortex-R	Cortex-M
ISA	ARM	ARM	Thumb
Inst. Bits	32	32	16
FPU	Yes	Yes	Optional
DSP Inst.	Yes	Yes	Yes (M4)
Dynamic Power	80μW/MHz	120μW/MHz	8μW/MHz
Application	High Performance	Real Time	Embedded

Cortex-A and Cortex-R processors are intended for high-performance real-time applications such as smartphones and automotive applications. Cortex-M processors cater for low power embedded applications. Despite the similar naming, these variants are largely different at the ISA level. The Cortex-A/R supports the full 32-bit ARM instruction set whereas the Cortex-M only supports a compact 16-bit subset called the "Thumb Instruction Set". The ISA domains can be visualized in Figure 2. The Cortex-M series is less capable but is more efficient in code size and power consumption for simpler workloads typically found in deeply embedded systems. Given the fact that Cortex-M processors can be coupled with an optional floating point unit; it is expected to perform equally well as a Cortex-A for certain applications. To run the simulations, the selected benchmark programs are compiled for

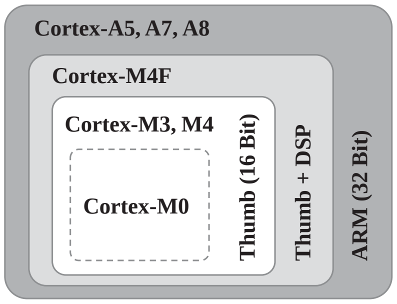


Figure 2: The ARM and Thumb instruction set domains.

three ISA and FPU (Floating Point Unit) configurations: ARM-FPU, THUMB-FPU, and THUMB-NoFPU. Subsequently, an MCU development board bearing a Cortex-M processor coupled with an FPU is used to run some representative programs. The execution time and power consumption footprint on the actual hardware are measured and analyzed.

3 Simulation and Test Methods

The GEM5 architectural simulator [9] was used for simulating ARM and THUMB ISA. The binaries were compiled with *–O3* level optimization and static linking. The standard input files/parameters provided with the benchmarks were used during simulations. The hardware used for measuring power and execution time was an STM32F4-DISCOVERY board shown in Figure 3.

Figure 3: The STM32F4-Discovery Board used for execution time and power measurement.

The onboard MCU (STM32F407VG) was clocked at a relatively slow clock speed of *16MHz*. The clock was generated using the internal RC oscillator to reduce the power consumption. The MCU also includes a low power standby mode which was measured to consume *~2μA* at *3V*. With the built-in RTC (Real Time Clock) enabled this figure goes up to *~3.4μA*. For measuring the current consumption for the benchmark subroutines, the execution sequence shown in Figure 4(a) was followed. The MCU was put in standby mode

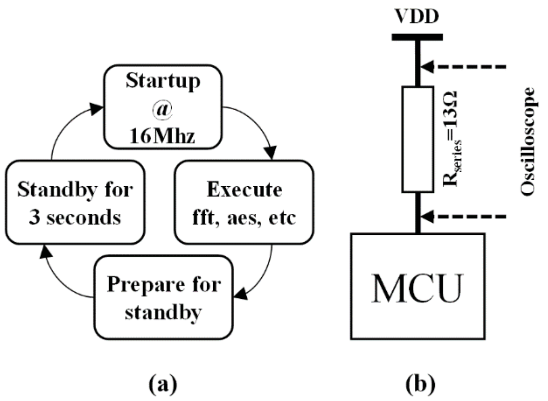


Figure 4: (a) The program sequence for testing (b) The test hardware setup for measuring current

with the RTC running, configured with an RTC wakeup interrupt. The MCU remained in standby mode for a predefined amount of time and consumed *3.4μA* as measured previously. After the standby time passes, the RTC generated interrupt wakes the processor up. The subroutine under test is the first code that is executed after wake up. After the computation is done, the MCU returns to standby mode. This cycle repeats. Since actual processing lasts for a very limited time, it was not possible to use a regular ampere meter to perform the current measurement. To capture the current consumed during this fast transition between active and sleep state, a small valued resistance was connected in series between the VDD and the MCU. The voltage drop across this resistor was measured using an Oscilloscope. The measurement configuration is shown in Figure 4(b). The measured voltage was later converted to current and the total energy consumption was calculated.

4 Experimental Results

4.1 Simulation Results

The GEM5 simulator reports detailed statistics about each simulation. The parameters of interest are execution time, IPC and instruction mix. The statistics of floating point instructions are especially important as it helps to justify the need for a dedicated FPU for a given task. The instruction mix of the programs compiled for three different ISA configurations is shown in Figure 5. In both the ARM-FPU and Thumb-FPU configurations, only three benchmarks (*basicmath, FFT, ecgsyn*) utilized floating point instructions. From this point onwards, these programs will be referred to as floating point benchmarks. The remaining benchmarks (*AES encode, AES decode, SHA,* and *CRC32*) did not perform any floating point calculations and will be referred to as integer benchmarks. For the Thumb-NoFPU configuration, the compiler did not generate any floating point instructions as there was no floating point unit available on the processor. In this configuration all floating point operations were performed through floating point emulation subroutines that rely on the integer calculations, thus resulting in a larger percentage of integer instructions. The most important observation in instruction mix was the similarity of the instruction distribution between ARM-FPU

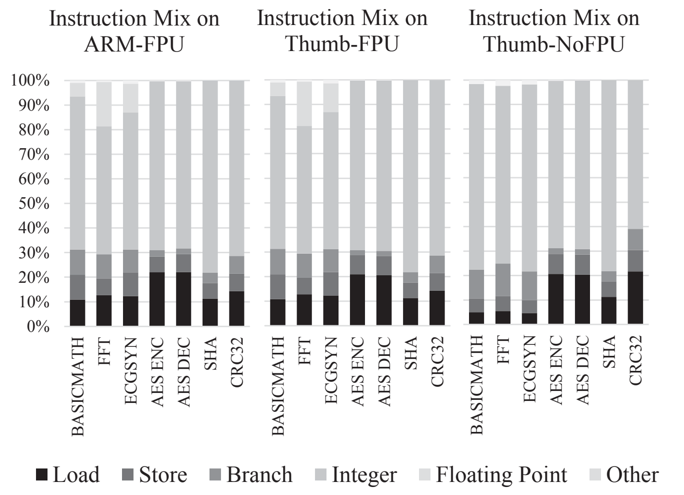


Figure 5: Instruction mix of the benchmark programs compiled for three different ISA platforms.

and Thumb-FPU configurations. This indicates that for the set of workloads at hand, the ARM ISA does not provide any substantial benefit over the Thumb subset in terms of functionality and code size. The program execution performance of the two ISA can still be different given the dissimilarity in their datapath and pipeline configuration.

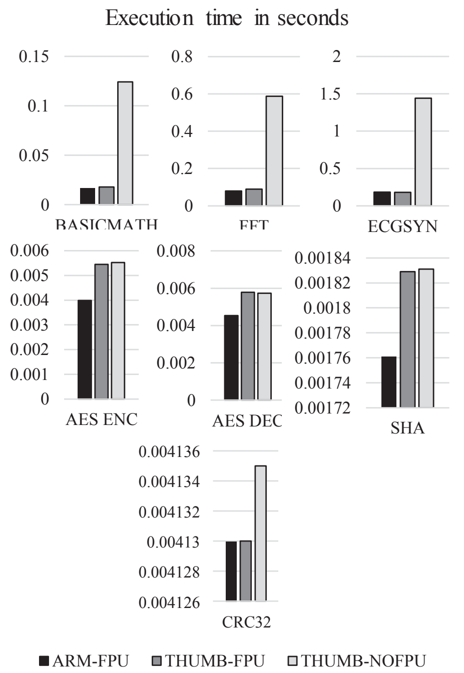


Figure 6: Comparison of execution time of the benchmark programs compiled for three different ISA platforms.

The execution time and IPC for the three configurations can be observed in Figure 6 and Figure 7 respectively. For the floating point benchmarks, the execution time difference for Thumb-NoFPU is substantially greater. This large time requirement is not acceptable for programs such as *FFT*, which is a critical operation that needs to process heart signal samples in real time. In these benchmarks, the Thumb-FPU configuration yields similar execution time as the more capable ARM-FPU configuration. For the remaining integer benchmarks, the ARM ISA exhibits faster execution time than the Thumb sets. However, the time requirement of these programs are smaller, and the difference can be considered as negligible. In the IPC chart of Figure 7, benefits of the ARM

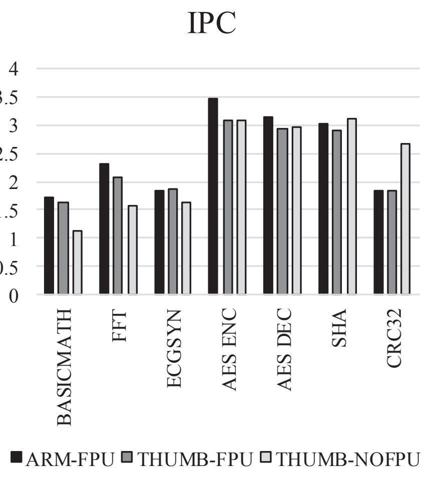


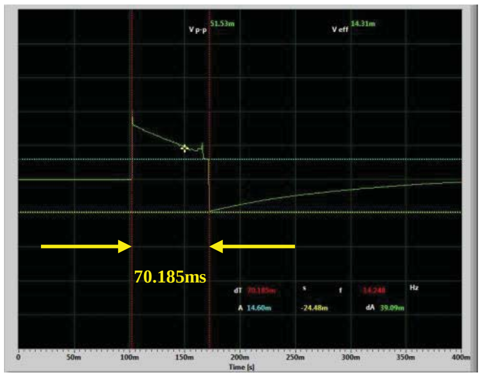
Figure 7: Instruction Per Cycle (IPC) of the benchmark programs compiled for three different ISA platforms

ISA can be observed as it achieves more instruction per cycle (IPC) than other configurations. The average difference with the Thumb-FPU offering and Thumb-NoFPU is *0.14* and *0.16* respectively.

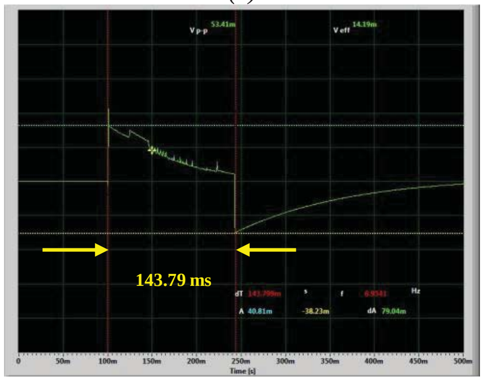
The simulation results clearly indicate that for the set of tasks in future cardiac pacemakers both the ARM-FPU and Thumb-FPU are favorable ISA configurations for performance. Looking at power consumption, the better option is the Thumb or Cortex-M offerings given their low dynamic power consumption of *8μW/MHz* [11]. In contrast, the Cortex-A offers *80μW/MHz* [11]. Modern pacemakers consume *6μW* to *13μW* at *2.8V* [3] depending on their operating state. The power consumption of Cortex-M remains mostly within these limited power budgets.

4.2 Power Consumption Measurements

The benchmarks used in the simulations are designed to evaluate the processor's ability to compute the algorithms of the given task. For practical power consumption estimation, representative programs for *FFT*, *AES encode/decode* [10] and *SHA-256* hash function [10] are used. These programs are optimized for embedded platforms but do not utilize any dedicated hardware on the MCU. Figure 8 shows the voltage measured across the series resistor for the *FFT* operation. The duration of the operation for a 2,048-point input data is about *70ms* with the FPU enabled and *144ms* without the FPU enabled. Figure 9 shows oscilloscope readings for the *AES* (encode and decode) and *SHA* programs. As expected, enabling the FPU did not make any difference in the execution time. The measurements are taken with the FPU disabled to avoid any unwanted power consumption. At the beginning of the *AES* and *SHA* hash functions, the algorithms need to load a large key value stored in the flash memory resulting in the large current

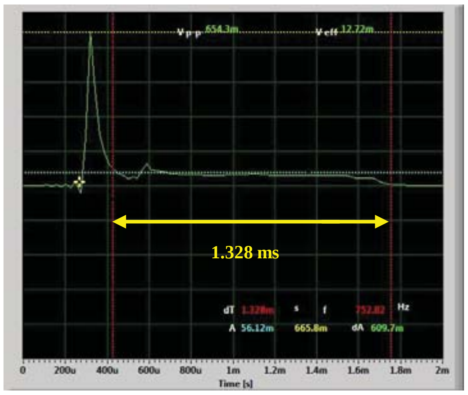


(a)

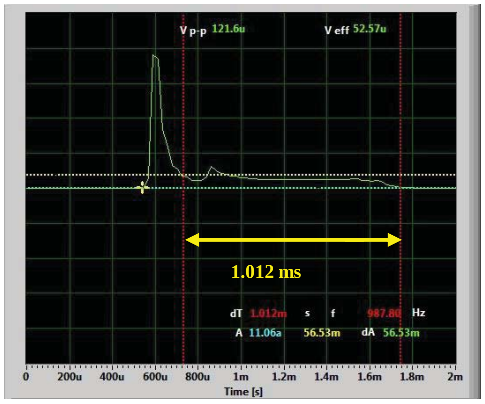


(b)

Figure. 8. Execution time measurement for FFT operation (a) with FPU (b) without FPU.



(a)



(b)

Figure 9. Execution time measurement for (a) AES operation (encode followed by decode) (b) SHA hash function calculation

spike seen on the oscilloscope. The remaining portion of the consumption envelope is contributed from the actual computation. To avoid the initial surge, the key values can be pre-loaded in the RAM if large enough memory is available. The execution time for *AES* and *SHA* was *1.33ms* and *1.01ms* respectively. The total energy consumed can be calculated by the following equation:

$$E_{total} = \frac{Vdd}{R_{series}} \sum V_s \times t_s [J] \qquad (1)$$

Where, E_{total} is the total energy in Joules, V_{dd} is the supply voltage (3V), R_{series} is the series resistor for measurement *13Ohms*, V_s is the amplitude of the sampled voltage and t_s is the oscilloscope sample hold time. Using equation (1), the energy calculated for the three selected programs are listed in Table. II. The results indicate the FFT as the largest power consuming

Table II. Execution time and energy consumption of selected programs.

Program	Time	Energy
FFT_FPU	70.2ms	7.96mJ
FFT_NoFPU	143.8ms	10.8mJ
AES	1.3ms	32µJ
SHA	1.0ms	25.8µJ

program among the other selected tasks. Given this demanding nature of Fourier transform computation, many pacemaker manufacturers rely on ASIC-based approaches for signal processing needs. However, the FFT operation on an ARM based SoC can be further optimized by implementing an integer based algorithm and utilizing DMA (Direct Memory Access) peripherals to transfer data from the source to the RAM. Some additional power consumption was captured in this experiment by reading the input data from the onboard flash memory. Further studies can be conducted by executing critical operations from only RAM with properly determined sample size to reduce the current.

5 Conclusion

The Thumb instruction subset from ARM was designed to be energy efficient in deeply embedded applications. Coupled with a dedicated FPU, this platform can perform efficiently in both floating point and integer workloads. However, for integer based workloads, energy consumption can be further reduced by using more streamlined FPU-less and slowly clocked processing chips. Positive results were observed for encryption and secure hash calculation, which consumed energy in the range of micro joules on the test hardware. On the other hand, Signal processing tasks were more demanding. The execution time and power consumption observed in this paper bring forward the differences in the energy footprint of potential workloads of a future cardiac pacemaker. Given the asynchronous execution nature of the programs and the difference in their energy footprint, the use of heterogeneous architectures such as ARM *"Big.Little"* architecture [12] promises lower power operation. Application specific optimization of each processing core on a heterogeneous architecture can be potentially utilized for low power operation in next generation of cardiac pacemakers.

6 Acknowledgement

Research reported in this paper was supported in part by National Institute of General Medical Sciences of the National Institutes of Health under award number 1SC3GM096937-01A1. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health

7 References

[1] Greenspon, Arnold J., Jasmine D. Patel, Edmund Lau, Jorge A. Ochoa, Daniel R. Frisch, Reginald T. Ho, Behzad B. Pavri, and Steven M. Kurtz. "Trends in permanent pacemaker implantation in the United States from 1993 to 2009: increasing complexity of patients and procedures." J. of the American College of Cardiology. Vol. 60(16), pp.1540-1545, (2012)

[2] Haddad, Sandro AP, Richard PM Houben, and W. A. Serdijin. "The evolution of pacemakers." IEEE Engineering in Medicine and Biology Magazine, May/June (2006)

[3] Biotronik Effecta Pacemaker Technical Manual, BIOTRONIK SE & Co. KG, Available: http://www.biotronikusa.com/manuals/ (2010)

[4] Lakshmanadoss, U., Shah, A., Daubert, J. P., in "Modern Pacemakers: Present and Future", M. K. Das, Editor, Intech Publisher, Croatia (2011), Chapter 8, pp.129-145.

[5] Halperin, D.; Heydt-Benjamin, T.S.; Ransford, B.; Clark, S.S.; Defend, B.; Morgan, W.; Fu, K.; Kohno, T.; Maisel, W.H., "Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses". Proceedings of IEEE Symposium on Security and Privacy, (2008) May 18-22; Oakland, California

[6] Haddad, Sandro AP, and Wouter A. Serdijn. "In Ultra Low-Power Biomedical Signal Processing", Springer Netherlands, (2009), Chapter 2, pp.14-26

[7] Guthaus, Matthew R., Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, and Richard B. Brown. "MiBench: A free, commercially representative embedded benchmark suite." Proceedings of the IEEE International Workshop on Workload Characterization, (2001) December 2; Austin, Texas

[8] Jin, Zhanpeng, and Allen C. Cheng. "ImplantBench: Characterizing and projecting representative benchmarks for emerging bio-implantable computing." IEEE Micro issue 28, no. 4, pp 71-91, (2008)

[9] Nathan, Binkert, Beckmann Bradford, and Black Gabriel. "The gem5 simulator." ACM SIGARCH Computer Architecture News, vol. 39(2), pp.1-7, (2011)

[10] Texas Instruments, C Implementation of Cryptographic Algorithms, Application Report. Available: http://www.ti.com/lit/an/slaa547a/slaa547a.pdf, (2013)

[11] ARM Cortex Series Documentation, ARM, Available: http://www.arm.com/products/processors/cortex-a/index.php (2015)

[12] Jeff, Brian. "Big. LITTLE system architecture from ARM: saving power through heterogeneous multiprocessing and task context migration." Proceedings of the 49th Annual Design Automation Conference. ACM, 2012.