A Dependable Language for Low Power Embedded Systems

Yuan-Ming Chang, Chia-Chen Hsu, and Jenq-Kuen Lee
Department of Computer Science, National Tsing-Hua University, Taiwan

Abstract—Minimizing the power consumption is crucial for embedded systems. Previous researches have successes for power optimization with stateless components in embedded processors. Recently, researches have started to design the architecture in minimizing the supply voltage for stateful components. However, lowering voltage also increases the risks of reliability. In this paper, we present a dependable language, which defines several expanded syntax rules. With this language, developers can describe the region of critical data and the region hoping for stored in low voltage region of memories. The language provides ways for programmers to participate in exploiting the variability and reliability issues of hardware designs.

1. Introduction
Minimizing the power consumption is crucial for embedded systems[1][2]. One method to reduce overall power consumption is lowering the supply voltage. For example, Abdel-Majeed et al.[3] propose a drowsy state which uses retention voltage to keep data alive in a lower voltage. While lowering supply voltage can save power consumption, it also hurts the reliability of the system. To be exact, the probability of error increases as the supply voltage is lowered[4][5]. There are researches which are motivated to detect soft error and present approaches to recover faults. Gao et al.[6] proposes explicit output comparison to identify faults. However, these researches are lack of flexibility, such as deciding which parts of the programs should be executed in low power mode or should be protected.

To enable programmers to participate in exploiting the variability and reliability issues of hardware designs, we propose a dependable language. By proposed pragma, programmer can decide whether the data/functions be protected, or whether the data/functions be put in low power mode. With cooperated architecture, our system can partially execute low power mode and guarantee the reliable of certain instructions.

2. Syntax Rules for Dependable Language
We propose several dependable pragma, which can support all kinds of data type, such as char, int, short, long. Besides, they can also support several conditional statements, such as for,while, if, do. In the following, we introduce proposed dependable language and their meanings.

• reliable
reliable means declared value/function needs to be protected. Following are some examples:
reliable int value means the declared value is a reliable integer type and it needs to be protected.
reliable for (...) means the declared for-loop is a reliable for-loop, and all values in the loop scope need to be protected.
reliable( output | count ) for (...) means variable "output" and "count" in the loop scope need to be protected, and others remain normal.

• dllpRegion
dllpRegion means declared value/function would be put in memory with low supply voltage or in certain memory region. Following are some examples:
dllpRegion int value means the declared value is a dllpRegion integer type, and it is stored with low support voltage.
dllpRegion for (...) means the declared for-loop is a reliable for-loop, and all values in loop scope would be stored with low voltage.
dllpRegion( output | count > $r1 ) for (...) means that, in the for loop, variable "output" and "count" would be stored in assigned region, $r1.

• reliable dllpRegion
reliable dllpRegion means declared value/function would not only be put in certain memory (low voltage region or other certain region) but also be protected. For example:
reliable dllpRegion int value means the declared value is a reliable dllpRegion integer type, and it needs to be protected and be stored with low support voltage.
reliable dllpRegion for (...) means the declared for-loop is a reliable dllpRegion for-loop, and all values in loop scope needs to be protected and be stored with low support voltage.
reliable( output | count ) dllpRegion( output | count > $r1 ) for (...) means variable "output" and "count" in the loop scope need to be protected and be stored in region0($r0) and region1($r1).
respectively.

```plaintext
reliable( output | count ) dllpRegion( count > $r1 
| $other > $r0 ) for (...) means variable "output" 
and "count" in the loop scope need to be protected. 
Moreover, "count" value should be stored in region 
1($r1), and other values are stored in region 0($r0).
```

To illustrate how to use proposed language, we take a 
**Low Power Smart Trash** as an example. The trash keeps 
in sleep mode most of time. In sleep mode, the support 
voltage of memory is kept low to save power. When people 
throw the garbage to it, the trash occurs interrupt and 
detects whether the trash is filled. The sample code of 
smart trash is shown in Listing 1 and the interrupt function 
is shown in Listing 2. In Listing 1, inPin0 and inPin1 
are the input ports of the trash(GPIO pins of sensor). We 
should guarantee the accuracy of the input ports because 
the system relies on it to get inputs. Since the input ports 
of a system are frequently used and important, we put 
them in regular memory. If inputs were put in low voltage 
region of memory, the system is difficult to be recovered 
when the error occurs. **Threshold**, declared in line 6, is the 
value based on it the system judge whether the trash is 
filled. We declare **Threshold** in **dllpRegion** type rather than 
putting in the memory of regular voltage because it can be 
recovered once the error occurs. While we also declare it in 
**reliable** type because **reliable** promises the system would 
detect errors, if any, and solve them. Without declaring in 
**reliable** type, the systems never check the correctness of 
the data. **lightSensor** and **pressureSensor** are the value of 
current pressure and light. When people throw trash, the 
interrupt is triggered (as shown in Listing 2). We have 
**lightSensor** / **pressureSensor** get data from reading inPin0 / 
inPin1. Since **lightSensor** and **pressureSensor** refresh every 
time when calling interrupt function, they are relatively 
less important than inPin0 / inPin1. Therefore we declare 
them in **dllpRegion** type, so does the **Text**. By this example, 
we present how to save energy without lose accuracy by 
flexibly using proposed dependable language.

```plaintext
void interrupt_handler (){ 
    /* read the input pin. */
    lightSensor = digitalRead(inPin0);
    pressureSensor = digitalRead(inPin1);
    if (pressureSensor > threshold
        && lightSensor)
        sendData(Text);
    ... 
    ...
}
```

Listing 2: Interrupt function

3. Conclusion

We propose a prototype of the dependable language 
which allows users to feasibly store data in low voltage 
region of memories and protect critical data. We also use 
an example of smart trash to demonstrate how to use 
this language. The language is our attempt to provide 
programmers ways to exploit the variability and reliability 
issues of hardware designs.

Acknowledgment

The work is supported in part by III and by Taiwan MOST.

References


