Extracting Low-Power Code Pattern Method Through Power Measurement of Software Code

Bo Kyung Park and Su Nam Choi

Abstract In the 4th Industrial Revolution, embedded systems were composed of high specification hardware for high performance and low power consumption. The software operating in such an environment should be capable of stable operation in an environment where resources are limited, such as limited hardware and memory, while maintaining original performance. This increase in power consumption leads to a reduction in the available time and an increase in heat generation. Software using high-end hardware increases the power consumption of the device. Eventually, these problems lead to SW/HW errors and shortened device life. This paper proposes a method to minimize power consumption through low power consumption code patterns in the core control structures (loop, branch, modulization, parameter passing). To make this possible, we define some low power code patterns. We can optimize the core code's performance and power efficiency by improving the most complex areas of the software code.

Keywords Low power consumption · Low-power code pattern · Performance

1 Introduction

The use of smart embedded devices has been rapidly spreading in the 4th Industrial Revolution fields such as drones, autonomous robots, and smartphones. The software operating on these embedded systems must maintain the original high performance developed in the existing development environment. Furthermore, software must perform reliable behavior even in environments with limited hardware resources, such as limited power and memory [1]. This increase in power consumption leads

B. K. Park (✉) · S. N. Choi
Chinju National University of Education, Jinju 52673, Gyeongnam, Korea
e-mail: parkse@cue.ac.kr

S. N. Choi
e-mail: csnpower@cue.ac.kr

J. S. Park et al. (eds.), *Advances in Computer Science and Ubiquitous Computing*,
Lecture Notes in Electrical Engineering 1028,
https://doi.org/10.1007/978-981-99-1252-0_89

to a reduction in the available time and an increase in heat generation. Eventually, these problems lead to malfunctioning the device and shortened life.

In order to solve the problem of stable operation of the embedded system, studies are conducted to lower the amount of power consumed and increase the efficiency of energy use. In particular, there are many studies on low-power systems. Research on low-power software design techniques, such as operating systems and application programs, which consume as little power as possible and maintain performance quality, has been conducted in the recent years. However, the design and development of low-power software depend on the developer's skill and intuition without software quality standards [2].

To solve this problem, this paper proposes a method for minimizing power consumption through the power measurement of the software code. This method measures the power used by each code in the high-level step. Then we extract a low-power code pattern that can reduce power. We apply this improved code with the newly defined low power pattern in this paper to reduce the amount of power consumed by the software code.

The composition of this paper is as follows. Section 2 introduces code-based low-power research. Section 3 explains the newly defined low-power code pattern through power measurement experiments. Finally, Sect. 4 describes the conclusions of this study and future research directions.

2 Related Works

Software optimization research for low power has techniques to reduce power consumption at the code level. Researchers define it as Energy Bug or Energy Code Smell. This coding technique can reduce power consumption by analyzing codes constituting program logic.

The Energy Code Smell is a pattern that is likely to reduce power consumption [3]. This pattern is classified into nine different patterns (Parameter by value, Self-Assignment, Mutual exclusion OR, Switch Redundant assignment, Dead local store, Repeated conditionals, Non-short circuit, Useless control flow). These classified patterns reduce power consumption through refactoring. This method can reduce energy consumption, but it can increase the opposite. Therefore, all patterns are not effective at low power. Table 1 gives Energy Bad Smell [4] defined in other research that makes up for defects.

Table 1 Energy bad smell

Energy Bad Smells	Definition	Pattern and definition

Complex expression	Code containing very complex expressions	if((a = = b) && (b = = c) ‖ (a + d > = 0)){
		//Statement
		}else if(!(a = = b) && (b = = c)
		‖ (a + d < 0)){
		//Statement
		}else{ //Statement}

Common sub-expression	Multiple codes containing the same task	or (If Statement){
		a = x + y + 1;
		b = (x + y) + z;}

Tail recursion	Code that is called recursively	int decrease(int a){
		if(a > 0){
		return decrease(a – 1);
		}else{ return 0;}}

Loop structure	A loop that can be optimized by adjusting the structure	– Multiple Index Variable – Double-loop structure – Declaration of global variable in loop
Dead code	Code that does not run under any conditions	Codes that are not used in entire execution

3 Low-Power Code Pattern to Minimize Power Consumption

We mention about low-power software code patterns to minimize power consumption. In order to propose a method of minimizing power consumption through source code power measurement, we measure the power consumption of code patterns within the basic control structures using ARM's ULINK Plus module [5]. We then classify the patterns based on the language paradigm to present a low-power code pattern.

3.1 Low-Power Pattern

① First, we configure the environment for source code power measurement. To do that, we use ARM's ULINK Plus module and Keil MCU Eval Board to measure the power of the source code [6]. ② After building the source code measurement environment, we run the Keil uVision5 IDE to create a new project. Keil uVision5 IDE is software that supports code measurement of software. ③ Enter the source code to measure power in Keil uVision5. ④ When the environment is configured, run the project. Projects are executed in the order of Build, Run, and Debug. ⑤ When entering the debugging mode, the power consumption of the input source code can

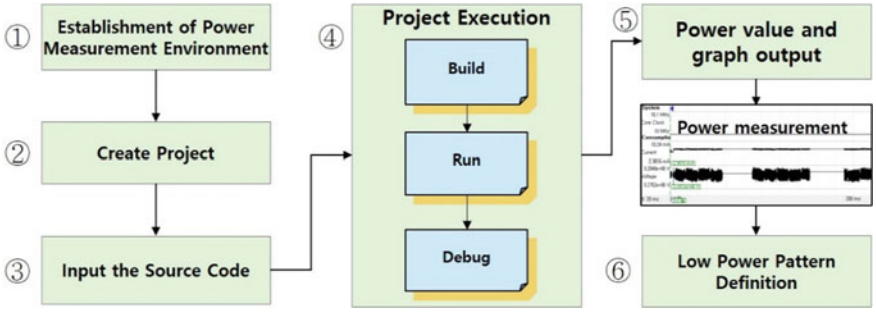

Fig. 1 Procedure of the low power pattern extraction through the power measurement of the source code

be measured. Finally, ⑥ Define the low power pattern by referring to the measured power value and graph (Fig. 1).

3.2 Power Measurement Environment

We mention how to provide a power measurement environment. Connecting the ULINK plus module to the MCBSTM32F200 Board makes software development possible with the Keil tool, and the code result can be checked on the LCD screen. Keil is power measurement software provided by ARM. The connection method is as follows. First, connect the 25 mA module to the ULINK plus module, then connect to the 3V3uC pin of the MCBSTM32F200 Board. Then, power is supplied through the ground. It is connected to the USB port of the PC to supply power. Finally, connect the board and module to the PC and create a new project in the Keil uVision5 IDE. When everything is done, create a c file. We can measure power consumption by writing code (or loading code) and running in debugging mode (Fig. 2).

In ①, it indicates the start and end points of the program. To measure the power during program execution, we must measure the current used up to this start and end point. In ②, the voltage is constantly supplied with 3.3 V. Actually, there is

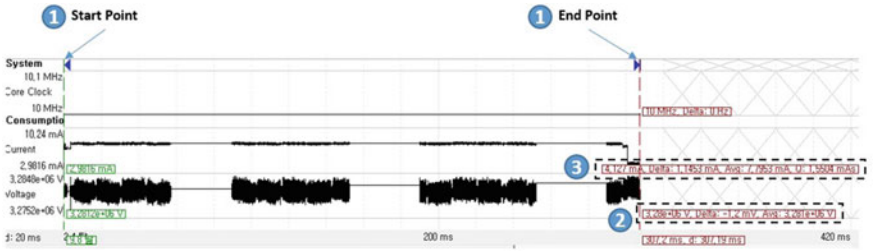

Fig. 2 Graph of the measurement results

a difference on the graph, but it is measured at 3.3 V because the difference is a small amount that changes in 1/1,000,000 units. When we click the start point with a mouse, a green line appears. When we place the cursor on the desired point, various information is displayed as shown on the right. The data of ③ shows the current amount, the delta current amount between the start point (green line) and the end point (red line), the average current amperage amount, and the cumulative current amperage amount from the start point to the end point. By substituting this into Eq. (1), the amount of power can be measured. Equation (1) is the power formula.

$$P = V \times I \tag{1}$$

3.3 Low-Power Code Pattern Definition

To measure the power of software code, we analyze the code power pattern of the procedural language paradigm. Even the code that prints the same result is written differently depending on the developer's coding tendency. For example, multiple conditions can be added at one time by using the AND or OR operator on one condition in a grade program. On the other hand, we can write code using multiple if statements. Alternatively, we can use if ~ then ~ else and switch statements. In this case, power consumption may be reduced or increased depending on the conditional statement. Through these comparisons, this study defined a total of four procedural code patterns and measured power. Each pattern measures power at least 50 times and averages it (Table 2).

"Double if control statements (①)" uses two if statements to operate on two conditional statements. && operation (②) is performed in the conditional statement. The current average value of ① is 1.591362 mA. The current average value of ① is 1.591362 mA. Equation (1) is used for power consumption. Then, the power consumption of ① is 5.2514946 mW. Since the average current value of ② is 1.584462 mA, the power consumption of ② is 5.2287246 mW. Therefore, it is possible to reduce power consumption by using the && operator in one conditional expression rather than multiple if statements.

In ③ Multiple if the else, we judge various conditions with one variable. In ④ Switch ~ Case, we judge various conditions as switch case statements like ③. The current average value of ③ is 1.802542 mA. ③'s power consumption is 5.9483886 mW. The average value of ④ is 1.725534 mA, so the power consumption is 5.6942622 mW. Therefore, it is possible to reduce power consumption using a switch case statement rather than multiple if-then-else.

"Loop Down Count (⑤)" outputs A while i decreases from 100 to 0 by one. "Loop Up Count (⑥)" outputs A 100 times as i increases from 0 to 100 by one. The average current value of ⑤ is 1.594938 mA, so the power consumption is 5.2632954 mW. The current average value of ⑥ is 1.598812 mA, so the power

Table 2 Measurement results of low-power code pattern

Item	Pattern name		
Code pattern	1) Double if control statements	2) if statement that includes && operation	

	int main(){
	SystemClock_Config();
	int a = 80, b = 90;
	if (a > = 80) {
	if (b > 90) {
	GLCD_DrawString(0,0,“A”);}}
	return 0;}
		int main() {
		SystemClock_Config();
		int a = 80, b = 90;
		if (a > = 80 && b > 90) {
		GLCD_DrawString(0,0,“A”);}
		return 0;}	

Measured value	Average value: 1.591362 (mA)	Average value: 1.584462(mA)	
Item	3) Multiple if the else	4) Switch ~ Case	

Code pattern	int main(){
	SystemClock_Config();
	int a = 80;
	if (a = = 80) {
	GLCD_DrawString(0,0,“A”);
	}else if (a = = 70){
	GLCD_DrawString(0,0,“B”);
	}else if (a = = 90){
	GLCD_DrawString (0,0,“C”);
	}else if (a = = 60){
	GLCD_DrawString (0,0,“D”);} return 0;}
		int main() {
		SystemClock_Config();
		int a = 80;
		switch (a){
		case 80:
		GLCD_DrawString (0,0,“A”); break:
		case 70:
		GLCD_DrawString (0,0,“A”); break:
			case 90:
			GLCD_DrawString(0,0,“A”);
			break:
			case 60:
			GLCD_DrawString(0,0,“A”);
			break:}
			return 0;}

Measured value	Average value: 1.802542 (mA)	Average value: 1.725534 (mA)	
Item	5) Loop down count	6) Loop up count	

(continued)

Table 2 (continued)

Item	Pattern name		

Code pattern	int main(){
	SystemClock_Config();
	int i;
	for (i = 100; i > 0; i–){
	GLCD_DrawString(16 * i,0,"A");}
	return 0;}
		int main(){
		SystemClock_Config();
		int i;
		for (i = 0; i < 100; i++){
		GLCD_DrawString(16 * i,0,"A");}
		return 0;}	

Measured value	Average value: 1.594938 (mA)	Average value: 1.598812 (mA)	
Item	7) For loop	8) While loop	9) Do-While loop

Code pattern	int main(){
	SystemClock_Config();
	int i = 0;
	for (i = 0; i < 100;i++){
	GLCD_DrawString (16 * I,0,"A");
	} return 0;
	}
		int main(){
		SystemClock_Config();
		int i = 0;
		while(i < 100){
		GLCD_DrawString (16 * I,0,"A");
		i++ ;
		}return 0;}
			int main() {
			SystemClock_Config();
			int i = 0;
			do {
			GLCD_DrawString (16 * I,0,"A");
			i++ ;
			} while(i < 100);
			return 0;}

Measured value	Average value: 1.802542 (mA)	Average value: 1.725534 (mA)	Average value: 1.740036 (mA)

consumption is 5.2760796 mW. Therefore, the use of Loop Down Count can reduce power consumption.

The power consumption of "For Loop (⑦)" is 5.2741656 mW. The power consumption of "While Loop (⑧)" is 5.2287246 mW. The power consumption of "Do-While Loop (⑨)" is 5.2587678 mW. Using the Do-While statement can consume the least amount of power, but the use of the While statement is recommended because the statement must be executed first due to the nature of the Do-While statement.

4 Conclusion

In this study, we propose a method to minimize power consumption through low power consumption code patterns in key control structures (loop, branch, modulation, parameter passing) to develop high quality software. We presented a code that consumes high power consumption through power measurement of high-level codes. However, the difference in power measurement values is insufficient. Therefore, we will measure the power of more complex and larger codes. We will also need to produce reliable results through various experimental data, systematic experimental scenarios.

References

1. Hyun Sik A, Bo Kyung P, Chul RY, Kim, Du K, Kim (2020) Code visualization approach for low level power improvement via identifying performance dissipation. KIPS Trans. Comp. Comm. Sys. 9(10), 213–220
2. Bo Kyung P, Byungkook J, Kim RYC (2019) Improvement practices in the performance of a CPS multiple-joint robotics simulator. Appl Sci 10, 185–198
3. Vetro V, Ardito L, Procaccianti G, Morisio M (2013) Definition, implementation and validation of energy code smells: an exploratory study on an embedded system. In: The third international conference on smart grids, pp 33–39
4. Jae-Wuk L, Doohwan K, Jang-Eui H (2016) Code refactoring techniques based on energy bad smells for reducing energy consumption. KIPS Tr. Software Data Eng 5(5):209–220
5. ARM ULINK Plus. http://www.emthink.com/ulink/plus
6. ARM Keil MCU Eval Board. http://www.emthink.com/mcb