國立臺灣大學電機資訊學院資訊工程學研究所

碩士論文

Graduate Institute of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

# 智慧型個別工作耗電量分析工具

# An Intelligent Task-Based Power Consumption Measurement Tool

曾信雄

Hsin-Hsiung Tzeng

指導教授：薛智文 博士

Advisor：Hsueh Chih-Wen, Ph.D.

中華民國 97 年 7 月

July, 2008

# An Intelligent Task-Based Power Consumption Measurement Tool

By
Hsin-Hsiung Tzeng
A Thesis Submitted To
Institute of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
For The Degree of Master
in
Computer Science and Information Engineering
July, 2008

# 智慧型個別工作耗電量分析工具

指導教授：薛智文 博士　　研究生：曾信雄

國立台灣大學資訊工程學研究所

## 摘要

現今的嵌入式裝置被廣泛的運用於日常生活中，由於半導體產業的高成長率，處理器的速度也變的越來越快；但是處理器速度越快，相對的能耗量也就越大。因此能耗量測成為了嵌入式系統裡面一個重要的議題。模擬量測可以提供使用者非常細微的分析結果，但是在分析過程當中所要花的時間太長；傳統的實體量測可以提供一個快速且真確的量測結果給使用者，但是往往無法觀察出系統在執行時的情形；在這篇論文中，我們在一個別工作量測工具上延伸了觀測硬體之功能，並且提出了一個方法對周邊輸入輸出裝置作觀察。相信這套能耗量測分析工具的實驗方法能對此工具的開發者與使用此工具之使用者有所幫助。

關鍵字：嵌入式系統、能耗值、模擬量測、實體量測、個別工作量測

## Abstract

Energy consumption measurement is very important for developers of modern embedded systems. Basically, there are two types of measurement methods: the simulation-based and the physical-based. For energy consumption analysis, the simulation-based measurement is slow and inaccurate. Though the typical physical-based measurement is fast and accurate, while it can not get detail system information. Therefore, we propose a task-based measurement that bases on the physical-based measurement to provide system information. However, previous works can not use system information to derive more detailed hardware dependent information well, such as the power consumption of CPU, memory, I/O devices, etc. In this thesis, we propose a novel idea to extract the power consumption of I/O devices, and verify our work through experiments on DMA. Experiments show that we can use our method to have an acceptable result. We believe this thesis is helpful for not only users who would like to measure the energy consumption of applications, but also provide a fast approach to developers who would like to get more hardware dependent information.



**Keyword:** embedded system, energy consumption, simulation-based, physical-based, task-based.

# Contents

# List of Figures

# Chapter 1

# Introduction

In recent years, the number of embedded devices show high growth rates. Mobile technology today makes us more and more convenient. Figure 1.1 shows the forecast of semiconductor global sales, the growth rate is about 9% per year. It is amazing that experts point out an average of three embedded devices per person worldwide will own in 2010. As the maturity of semiconductor becomes higher and higher, microprocessors own a rapid speed, therefore the power dissipation will increase heavily. As we know, embedded devices mostly use batteries as their power source, so energy profiling becomes an urgent issue on system design. Not only embedded devices, but also the data center even desktop concern about this issue because the cost of energy is extremely large. When developers who try to measure a high performance and low-power system, they need to know the tradeoff between software application and architectural level. Consequently, a power consumption analysis tool will be essential.

Researches on energy profiling in embedded system try to meet the follow-

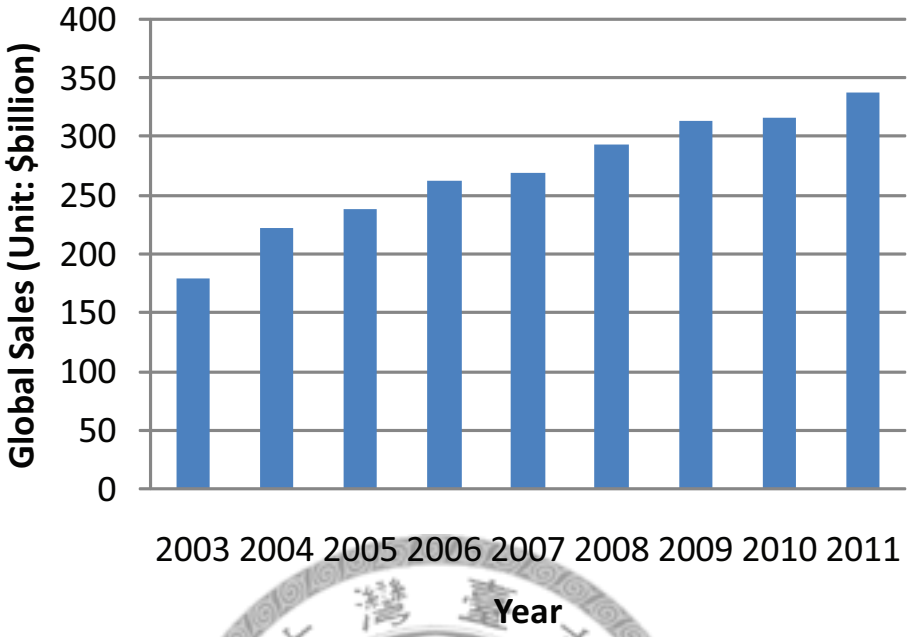



Figure 1.1: The forecast of Semiconductor Global Sale

ing requirement: accuracy, fine-granularity, easy-to-use, and low cost. Moreover, embedded systems nowadays are multi-tasking, they are more complex to measure the energy consumption. However, energy profiling tools at present do not provide a good way of accurate measurement on system-level. Existing tools can be simply classified into two types: simulation-based and physical-based measurement tools. Simulation-based measurement tools can analyse detail activities of the system. When we would like to measure a program, we just set the parameters of the system architecture, then the profiling tool will simulate the program execution on the simulated platform and analyse its usage of system component. We also need to know the power consumption of components in advance. In other words, the measurement result is not the

exact energy consumption of the system. Physical-based measurement tools provide the exact power consumption of the target hardware component. They are very fast and objective. However, the drawback of this kind of tools is hard to get the information of system activities. Fortunately, there exists a kind of physical-based measurement tools called task-based measurement tools. Task-based measurement tools can analyse the power consumption precisely on system-level. However, we still have problem on the task-based measurement tool. The problem is the result of the measurement suffers from dynamically changed power consumption of I/O devices. So our work is trying to further analyse the power consumption of I/O devices, and provide more accurate measurement results.

In this thesis, we adopt a task-based measurement tool developed by Tu [9]. We solve the problem that task-based measurement tools are not aware of I/O device by a novel method: grouping. We also provide an approach that is easy to modify the tool. This approach is for developers who use task-based measurement tools to extend the hardware-aware functionality, and for users who want to measure the energy consumption of applications, we can also provide a convenient and rapid way of measurement. Finally, we demonstrate the power consumption of DMA by using our approach.

The rest of this thesis is organized as follows. We will discuss the related work and background in Chapter 2. In Chapter 3, we talk about the

tool enhancement. And Chapter 4 provides a flow of a approach to extend a hardware-aware functionality for our tool. Chapter 5 shows the verification of our approach and experimental results of power consumption of DMA. Finally, this thesis is concluded in chapter 6.

# Chapter 2

# Related Work and Background

As researches on power-aware issues become more important on portable embedded systems. In order to estimate the energy consumption, we need an energy profiling tool to help us. Basically, we can categorize those tools into two types: simulation-based tools and physical measurement-based tools. In this thesis, we adopt a tool that is a kind of physical measurement-based tools. This chapter, we discuss related work and background knowledge about simulation-based tools, physical-based tools, and the task-based energy consumption analysis tool we adopt.

## 2.1 Simulation-Based Tool

In simulation-based measurement tool, the system will be abstracted into various components. The energy consumption of a program can be estimated by summing all energy consumption of all components. In [3], Chunling et al classify simulators by different granularity. According to their levels of

system and and component abstraction, there are transistor-level, cycle-level (Microarchitecture-level), instruction-level, and system-level. Simulators belong to transistor-level characterize models of transistors and estimate voltage and current behavior over time. They are also called circuit-level or gate-level. They are useful for integrating the circuit design. Due to the heavy time-consuming problem, they are not suitable to evaluate power consumption of large programs. Simulators belong to cycle-level simulate the execution at the level of individual cycles. Usually, they are used for simulations of modern superscalar processors because their granularity are moderate. Three examples of cycle-accurate simulators, Wattch [1], SimplePower [13], and Sim-Panalyzer [6], have been applied to various platforms, including ARM SA1110, Alpha, and PISA. Simulators belong to instruction-level simulate coarser granularity. This kind of simulators performs instruction-level energy profiling of the instruction set of the target processors. Normally, instruction-level simulators are faster than cycle-level and transistor-level. JouleTrack [7] is one of the example that belongs to instruction-level simulator. And simulators belong to system-level are hardware dependent architecture. These systems are composed of many different hardware components, and characterize the energy consumption of each system in different states. Duke University [8] extends the POSE to a palm OS simulator. So we can find out that it is a tradeoff between the granularity and the speed. Although simulation-based tools with coarse granularity can speedup the time consumed during the measurement, still there exist a gap of

speed between simulation-based tools and physical measurement tools.

## 2.2 Physical Measurement-Based Tool

In stead of abstracting the system, the energy of a profiled computer system will be measured directly by an external hardware in physical measurement-based tools. Usually, the external hardware will be digital multimeter or oscilloscope. Advantages of physical measurement-based tools are fast and accurate, but there must have a precise mapping between the measurement result from oscilloscope and the profiled program segment. Physical-based measurement tools use the concept of trigger point to matching the oscilloscope and the profiled segment. System designers often use this kind of tool for power-aware issue, and researchers [4, 11, 10] of the simulation-based tool also need this kind of tool to validate the result. There are highly dependency on precision and the sample rate. The higher sample rate is, the more precise measurement result we have. PowerScope [2] is a hardware instrumentation tool proposed by Flinn and Satyanarayanan. PowerScope maps energy consumption to program structure like the way CPU profilers map processor cycles to specific processes and procedures. The architecture of this tool is composed of three components: system monitor, energy monitor, and energy analyzer. The system monitor samples the program counter and the process identifier. The energy monitor stores the current samples. And the energy analyzer maps samples to specific procedures, so it can determine the power consumption of the specific process. Research

in [3], Hu, Jimenez, and Kremer used the oscilloscope trigger module to solve the mapping problem. They proposed a SimPoint idea to overcome hardware limitations and measure long program. In SimPoint, a program execution is partitioned into intervals with a fixed number of instructions. The similar intervals behavior are clustered into a phase. However, the sample rate of this approach must less than 3700 samples/sec, and it is not high enough as a result of the long communication cost between oscilloscope and data-acquisition machine. Moreover, this approach lacks for considering the impact from the execution activities of operating system. In [5], Dongkun Shin et al proposed an energy profiling tool named SES. SES integrates the profiled system and measurement circuit into a PCI adapter. SES collects the data of power consumption in a cycle-by-cycle resolution and without any additional measurement equipment. But the drawback is the environment of profiled system is fix and there are many hardware restrictions to developers. So we have a conclusion about this section, traditional physical measurement-based tools can not provide a high sampling measurement, and seldom consider the execution activities of operating system, such as context switch, kernel process, interrupts, etc.

## 2.3 Task-Based Energy Consumption Analysis Tool

In this thesis, we enhance the task-based tool designed by Tu [9]. It is a kind of physical measurement-based tools with high sample rate. This section we will talk about the features, system architecture, and flow path of the measurement

of this tool.

### 2.3.1 Features

This task-based energy consumption analysis tool is a kind of physical measurement-based analysis. Instead of the external oscilloscope, the author used a data acquisition card: NI-5112 DAQ card for higher sampling to reduce the error and enhance the reliability. This data acquisition card has a high speed PCI bus as its interface and the PCI latency is insignificantly small about 32 clocks, so the acquiring and processing large waveforms is much faster than external instruments. Moreover, the original data type double in NI-5112 DAQ card was modified to float before saving the measured data to hard disk, so it can reduce the redundancy of memory space. The author also changed the file format to save the measured data file from text to binary. Compare with text file, binary file is more efficient format because of less disk usage and no need to translate textual characters. We conclude the characteristics above, the sample rate of this tool is up to 3M and much higher that other researches.

In order to reduce the error, the author drop the original transformer because it has a jitter with around 1.6% error rate. Instead, he used a stable power supply as the power source. In addition, this tool selects a highly accurate sense resistor with resistance shift only less than 0.5% to avoid the effect from the resistor error.

There are two synchronized channels called channel0 and channel1 of DAQ card. Channel0 attached to the left and right side of the sense resistor fetches the measured voltage. Channel1 attached at the GPIO (general-purpose input-output) pin of the evaluation board senses the variation of the pin voltage. When measuring the desired program interval, the trigger generation code for toggling the GPIO pin voltage must be inserted at the beginning and the end of the interval respectively. Based on the two synchronous channels, the analysis tool will match the measured interval within the measured data from channel0 by determining the trigger point within the ones form channel1 to recognize the corresponding data form channel0.

Not only the measured program interval, but the system activities are considered by the following technique similar to the approach as mentioned above. The author uses a system call for toggle and inserted it into *schedule*() and *do_IRQ*() in Linux kernel. Consequently, this analysis tool is sufficient to consider that the system activities from *schedule*() and from *do_IRQ*(). The former will select a process to execute, and the latter will execute the corresponding interrupt handler while the timer's interrupt every 10ms for the running period of a process or a hardware interrupt occurs.

This tool adopt Linux procfs (proc file system) mechanism to record process execution information at runtime. The procfs is a virtual file system, and it is not associated with a block device but exists only in memory. It takes

the advantage of procfs to save data to memory so that the overhead is much smaller than to secondary storage. Based on the author's experiment, procfs mechanism is only 2% plus power and is 93 times faster than saving the same data to an opened file in flash. Therefore, this is a low-overhead approach for recording process activity.

### 2.3.2 System Architecture

The system architecture is composed of the measured evaluation board, a power source, and a DAQ card. The evaluation board is the popular embedded system development board, where an operating system and some user programs can run. In order to get a precisely result of measurement, we also use the power source supplies the energy to measure the evaluation board's power. The DAQ card is an internal device with PCI bus interface.

We use the Creator S3C2410 Development Kit made by Microtime Computer Inc board as our measured evaluation board in this tool. Several characteristics mentioned below:

- ARM920T Core (200MHz)
- 16MB Flash ROM
- 64MB SDRAM
- A Linux 2.4.18 porting

The DAQ card this tool adopt is the National Instrument NI-5112 High Speed Digitizer. Its main specification is described as follows:

- 2 simultaneous channels
- Max 100M samples/sec
- 16MB memory per channel
- 8-bit resolution

The output of power supply is more stable than transformer. Moreover, the power source in this tool uses a power supply but not the Creator's transformer capable of reducing 1.6% error.

### 2.3.3 Operation Flow Path

The profiling procedure is composed of three stages: pre-process stage, measurement stage, and analysis stage. The profiling procedure is shown in Figure 2.1 and explained in the following sections.

#### The Pre-Process Stage

At the pre-process stage, we have to insert the trigger generation codes at the beginning and the end of the profiled program to determine what program segment we want to measure. In this stage, we also need to set up the hardware. As shown in Figure 2.2, one of the DAQ card channel (channel0) are responsible for measuring the voltage across a sense resistor $R$ which is connected in

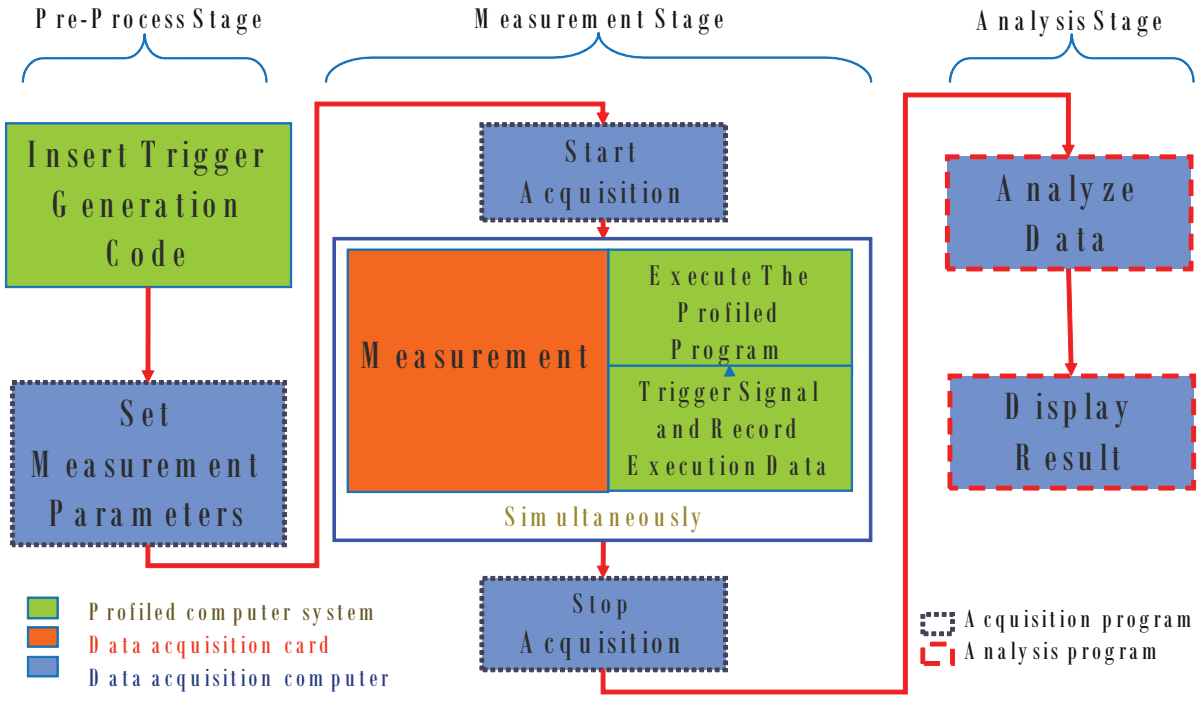



Figure 2.1: Operation flow path of the procedure

series with the power supply and the profiled computer system. Another one (channel1) is connected to the GPIO (general-purpose input-output) pin of the profiled computer system. After that, we set the measurement parameters in the acquisition program, including sample rate, data range, saved file path, etc. The acquisition program we implement is to control the DAQ card.

#### The Measurement Stage

At the measurement stage, the DAQ card would begin to measure by starting acquisition program. After that, we execute the profiled program in the profiled computer system. The trigger generation code we inserted at the beginning and the end of the profiled program segment would toggle the GPIO pin voltage,

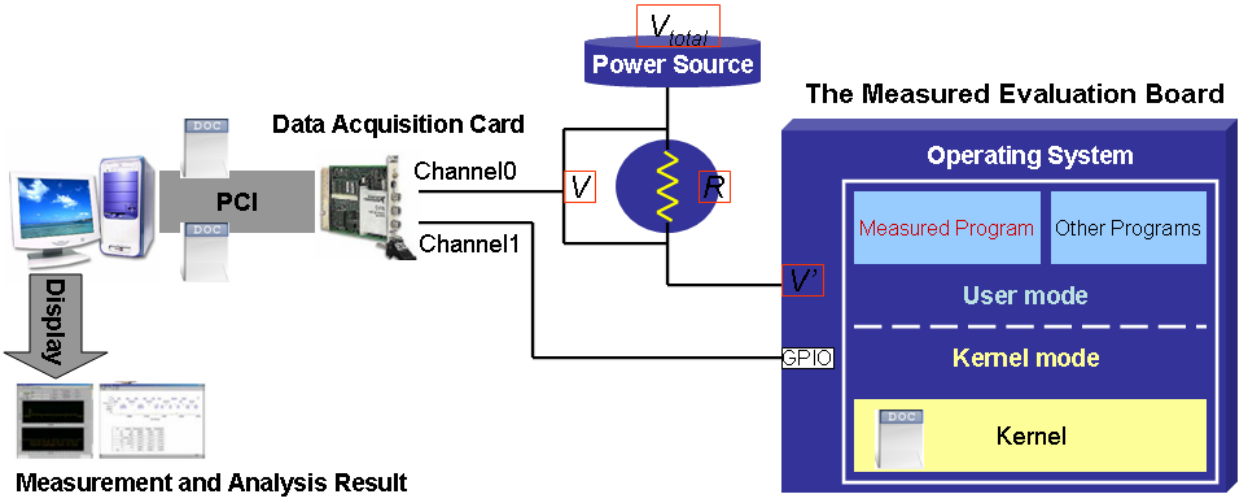



Figure 2.2: System Architecture

and then channel1 would sense the change of pin voltage. So we can find the measured interval in channel0 by comparing channel0 and channel1's data in the analysis stage. In addition to the beginning and the end of the profiled program, the functions context switch or an interrupt would trigger the toggle and record some data in kernel to log this event. After measuring, we stop the acquisition through the acquisition program. The DAQ card would produce two log files from the two channels. On the other side, the profiled system would produce the log file for analysis program.

#### The Analysis Stage

At the final stage, we complement an analysis program to analyze the three log files from the measurement stage. As shown in Figure 2.2, from the measured voltage $V(t)$ and other hardware parameters, the power of the profiled computer

system can be calculated as

$$P(t) = \frac{V(t)}{R} \cdot V' = \frac{V(t)}{R}(V_{total} - V(t)) \tag{2.1}$$

where the profiled system voltage $V'$ is $V_{total}$ subtract $V(t)$, and $V_{total}$ is the power supply voltage. The energy consumption of a period in the profiled computer system is calculated by

$$E(t) = \sum P(t)\Delta t = \frac{1}{S}\sum P(t) \tag{2.2}$$

$\Delta t$ represent the period between each sample, the $S$ is the sample rate of the DAQ card. The analysis program integrates the energy data from DAQ card and the process execution data from kernel to produce an overall system running information of the measured interval, which includes scheduling result with power information, execution time, energy of every process, etc.

# Chapter 3

# Tool Enhancement

We observe that Tu's [9] work can only adopt sample testing programs to measure. Wu [12] revised this tool to measure more complex programs. However, there are some problems need to valuate. The biggest one is the power consumption of every processes is muddled together. Because the power we measure by the oscilloscope is the total power of the board, we can not know the exact power consumption that belong to the process we want to measure. This chapter, we illustrate how to reduce the error rate of the tool and its overhead the measurement require. Furthermore, we take advantage of the task-based analysis tool that consider the system activity like interrupt and scheduling to adopt a new feature to estimate the power consumption of hardware by using a novel method: scenario grouping. This method can measure the exact power consumption caused by I/O devices without too much effort. Since we know the power consumption of I/O devices, we can subtract it if it is unconcerned with the task we want to measure.

## 3.1 Eliminate Redundancy

The original architecture inserts toggles in function do_IRQ() and schedule() at kernel. As shown in Figure 3.1, the error rate is very high. The reason that causing high error rate is the unceasing interrupts, and the time between two interrupts are too close so that the data can not be retrieved correctly by the oscilloscope. Because there are too many interrupts occur, like timer interrupt or some hardware, and every interrupt will be handled by the function do_IRQ(), so it is not applicable that inserts the toggles in function do_IRQ(). We observe that they inserted those toggles just only want to recognize different tasks. In fact, we can identify two different tasks just use the function schedule() without insert the toggles in function do_IRQ(). Figure 3.2 shows a desirable result that after removing the toggles in function do_IRQ(), we obviously reduce the error rate of program measurement. Moreover, because we do not need to burden the heavy routine after interrupt occur anymore, so we can reduce the information that are stored by the proc filesystem. The number of records that is measured by the original tool is about 398 per second an average. After eliminating the redundancy of unnecessary toggles, the number of records is down to 52 per second an average.

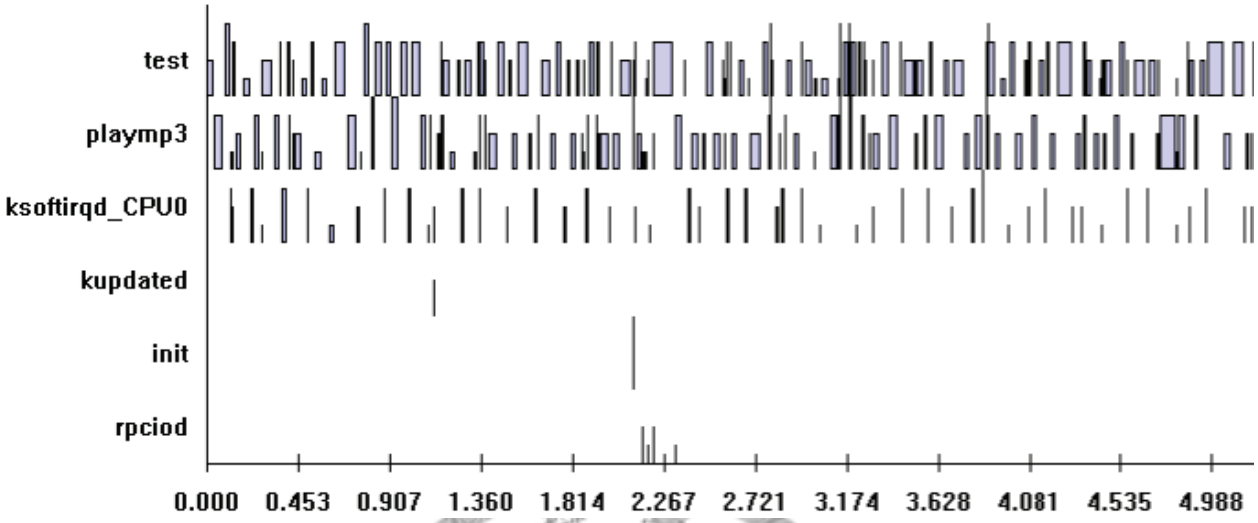



Figure 3.1: Insert toggles in function do_IRQ() and schedule()

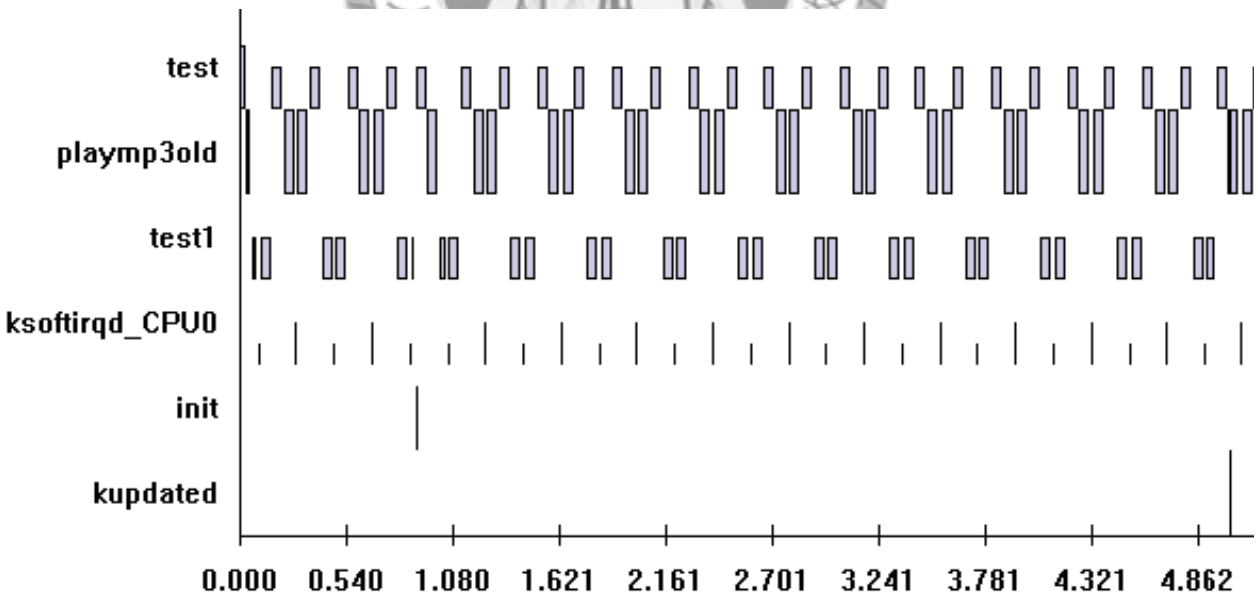



Figure 3.2: Only insert toggle in function schedule()

## 3.2 Hardware-Aware Feature

Hardware-aware means we can estimate the power consumption of the desired hardware component. Pervious work only used toggles in the kernel to distinguish different tasks. Nevertheless, we underprice the advantage of considering the system activity. Once we own the information of system activity, we can use those information to measure the power consumption of the hardware. Now we will make good use of the advantage of task-based measurement tools to add a hardware-aware feature. This section, we will first define the problem of the task-based analysis tools, and give some assumptions to make our method of measurement work.

### 3.2.1 Problem Define

First, this tool has a problem that is when we want to measure a desired program, the result of the measurement could involve the power consumption of I/O device that is independent with this program.

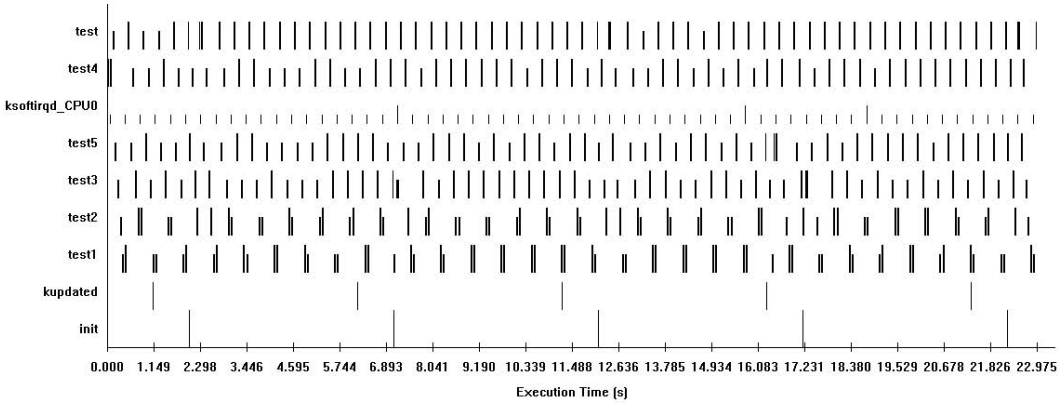
Second, we investigated the power consumption relation about hardware. The experiments in Figure 3.3 and Figure 3.4 show that six and three tasks which every task continuous doing multiplication respectively. And other tasks are invoked by the system, because the influence of those tasks is not big, so we omit them here. According to these two figures we know that the power consumption of each task in two figures does not have too much different. Because

those two experiments have the same usage of hardware. Compare to Figure 3.2, the experiment have additional application playmp3. we found the task playmp3 has more power consumption. The additional power consumption is due to the utilization of the hardware is vary, like the workload of processor, activity of memory, or caching would be different, and the whole computing would involve other hardware access. Through those experiments, we know the power consumption is highly dependent upon hardware utilization.

Concluding the description above in this subsection, we propose the following hypothesis that forms the basis to meet our goal: *The power consumption of the hardware would be identical with its hardware that have the same utilization regardless the content of work.*

### 3.2.2 Power Measurement Using Scenario Grouping

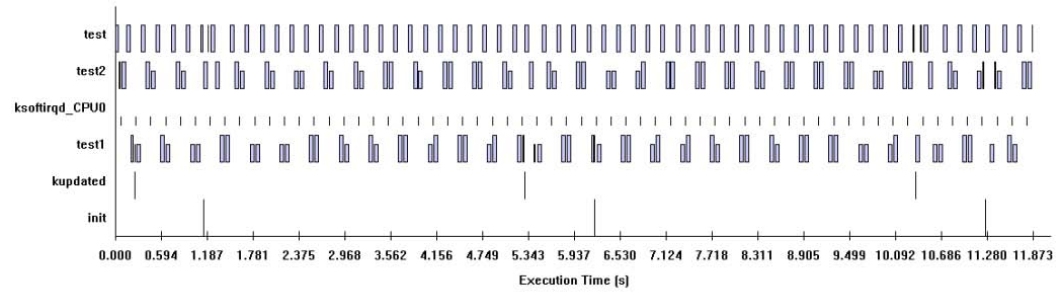
We use a measurement program to segment itself into different scenarios according to different usage of I/O devices. The usage of I/O devices means the utilization of I/O devices when the system is act. The purpose of measurement program is to observe the program we want to analysis. As we know, the system involves lots of variables like cpu utilization, memory, cache for example. In order to reduce the inaccuracy of measurement result, the measurement program do nothing but a infinite loop. Thus, we can observe the changing of the power consumption without too much interference. Before the measurement, The system will build a table to map the usage of I/O devices. During the




The Process Information in Measured Interval

| Pid | Command | Execution Time(s) | Energy (J) | Avg Power (W) |
|---|---|---|---|---|
| 41 | test | 3.756778 | 9.577350 | 2.549352 |
| 38 | test4 | 3.730552 | 9.497790 | 2.545948 |
| 3 | ksoftirqd_CPU0 | 0.627091 | 1.571447 | 2.505932 |
| 39 | test5 | 3.699237 | 9.417446 | 2.545781 |
| 37 | test3 | 3.700206 | 9.420288 | 2.545882 |
| 36 | test2 | 3.699523 | 9.416770 | 2.545401 |
| 35 | test1 | 3.760143 | 9.573346 | 2.546006 |
| 6 | kupdated | 0.000247 | 0.000633 | 2.563570 |
| 1 | init | 0.001376 | 0.003567 | 2.592370 |
| Total | | 22.975151 | 58.478638 | 2.545299 |

Figure 3.3: Six tasks are doing the same work




The Process Information in Measured Interval

| Pid | Command | Execution Time(s) | Energy (J) | Avg Power (W) |
|---|---|---|---|---|
| 43 | test | 3.755228 | 9.543653 | 2.541431 |
| 42 | test2 | 3.790567 | 9.620790 | 2.538087 |
| 3 | ksoftirqd_CPU0 | 0.627192 | 1.566569 | 2.497751 |
| 41 | test1 | 3.699367 | 9.386019 | 2.537196 |
| 6 | kupdated | 0.000219 | 0.000561 | 2.560229 |
| 1 | init | 0.000825 | 0.002143 | 2.597808 |
| Total | | 11.873399 | 30.119736 | 2.536741 |

Figure 3.4: Three tasks are doing the same work

measurement, once the I/O device is active, the toggle will trigger the system to record the I/O device information according to the table. For example as shown in Figure 3.5, if we have I/O device1 and I/O device2 that we can grab its information from toggles. At the analysis stage, the analysis tool will segment tasks into different scenarios. Every scenario represents a situation of the usage of I/O devices. In this case, we have four different scenarios. S0 represents the scenario that no I/O devices are active record by the table, S1 represents the I/O device1 is active but the I/O device2 is not, S2 represents the I/O device1 is not active but I/O device2 is, and the S3 means the I/O device1 and I/O device2 are both active. At analysis stage, the analysis program will collect those segmentations which are labeled as the same scenarios into a group. In this case, those segmentations which were labeled as S0, we will collect them into G0. Those segmentations which were labeled as S1, we will collect them into G1. So the power consumption of hardware1 will be G1 - G0 or G3 - G2. And the power consumption of hardware2 will be G2 - G0 or G3 - G1. That is to say, if we got more different scenarios, our measurement result would be more precisely.

## 3.3 Comparison With Simulation-Based Measurement Tools

Taking three common used simulation-based measurement tools: Wattch [1], SimplePower[13], and Sim-Panalyzer [6] as examples. Wattch can not measure the power consumption of I/O devices because it does not have I/O model-

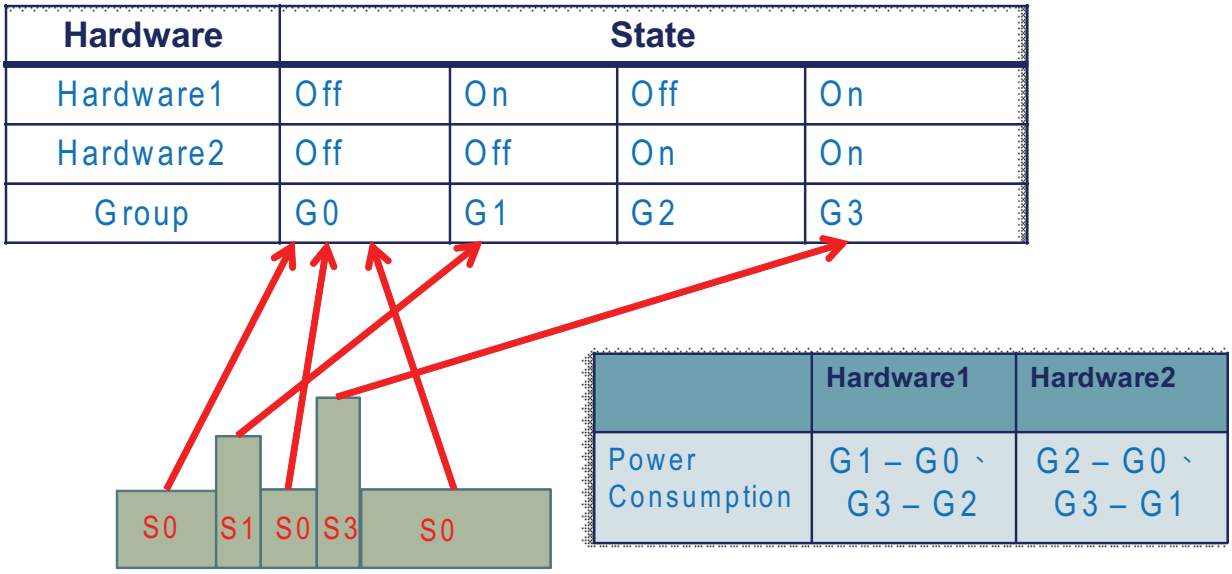



Figure 3.5: Using Grouping to measure the power consumption of I/O device

ing support. Sim-Panalyzer does not have any I/O modeling support neither. Sim-Panalyzer support the measurement of I/O devices, but it can only offer general information about I/O transmissions like the power consumption between memory and I/O bus, it can not measure more specific information of I/O devices like the power consumption of I/O device when it is active. We found most simulation-based measurement tools do not provide a good way of power consumption measurement of I/O devices. Compare with those three profiling tools, our tool performs a better way of I/O device measuring. We can measure the power consumption of the desired I/O device easily by using our tool.

# Chapter 4

# Measurement Approach

The profiling procedure has three stages: pre-process stage, measurement stage, and analysis stage. Because this analysis tool has new hardware-aware feature, and the main goal of this thesis is provides a hardware aware measurement approach. Therefore, the changing of the procedure is avoidless. This chapter, we provide an approach of extending hardware-aware features. Moreover, we will extend a DMA-aware feature on our tool as an illustration.

## 4.1 Approach of Measurement With Hardware-Aware Feature

As shown in Figure 4.1. The approach is based on the profiling procedure. The whole approach will take place at pre-process stage. Descriptions at block diagram with green color mean actions which execute on profiled computer system. The profiled computer system includes the program we want to measure and the operating system we use. And descriptions at block diagram with blue color mean actions which execute on the power consumption analysis tool. The

power consumption analysis tool include the acquisition program and analysis program.

### 4.1.1 Profiled Embedded System

First, kernel modification means the embedded operating system we use to measure our profiled programs. We must know what hardware we want to measure. And we insert toggles at kernel. In general, the activity of the hardware will accompany with its interrupt service routine. So we can insert toggles at the interrupt handler hardware to get the starting time of hardware access. Also we must determine the stoping time of the hardware access. After that, we need to add the activity character for power consumption analysis tool. Second, we insert triggers on user application. User modification means that we must design a measurement program to observe the changing of the power consumption of different groups.

### 4.1.2 Power Consumption Analysis Tool

Once the modification of profiled embedded system is complete. Then we can gain the additional information of I/O devices for our use. So the analysis program also has some modifications. The analysis program must recognize which new hardware activity character we add. So we need to register new hardware activity as hardware scenario on our analysis tool to build a grouping table. Once we have the grouping information, we can know the power consumption

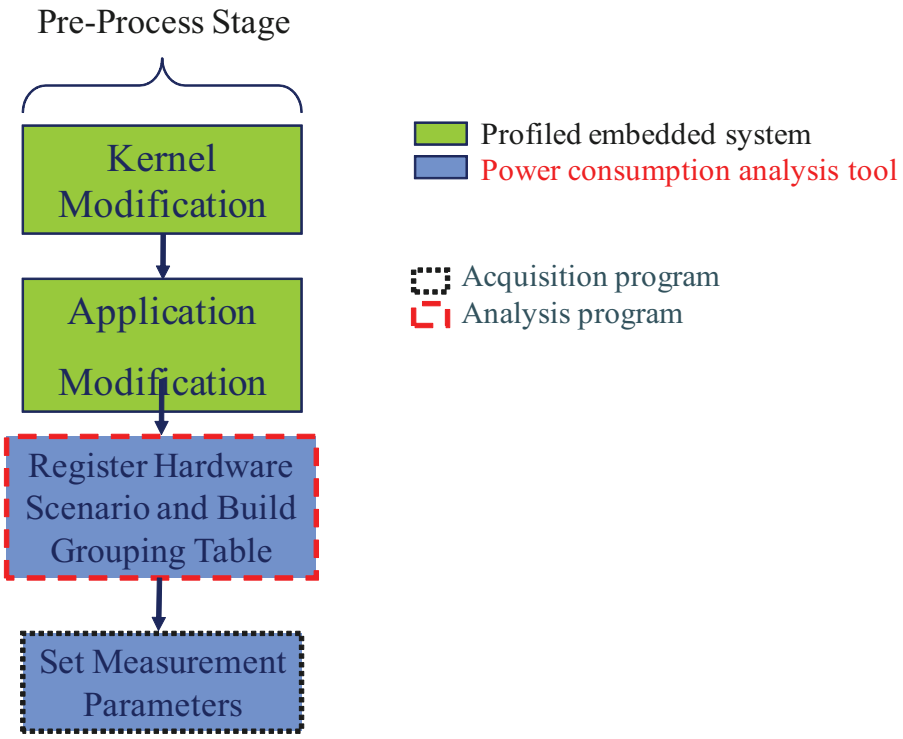



Figure 4.1: The flow path of the approach

of I/O devices.

## 4.2 Implementation of DMA-Aware Functionality

Because DMA is the common used solution for handling the transmission of data between low speed I/O devices and memory. Those I/O devices include disk, VGA card, sound card etc. Moreover, simulation-based measurement tools do not provide a good way of measuring the power consumption of I/O devices. This section, we will illustrate how to use the approach to measure the power consumption of DMA.

Table 4.1: Activity character

| character | execution occurrence |
|---|---|
| a | At the start of measured interval |
| b | At the end of measured interval |
| s | At the start of *schedule*() |
| t | At the end of *schedule*() |
| d | At the start of *dmaDone_irq_handler*() |

### 4.2.1 Profiled Embedded System

We remove redundant *do_IRQ*() toggles. In stead, we insert new toggles in *dmaDone_irq_handler*(). The function *dmaDone_irq_handler*() is an interrupt handler specific for DMA when the DMA access is act. As DMA access is act, the handler will load a buffer and transfer data to the destination until there is no data. By inserting these new toggles, we can grab the starting time of DMA. Thus, if we can also know the stoping time of DMA, we can use them to observe the additional power consumption that caused by DMA. And the number of records is up from 50 to 60. Compare with the number of inserting toggles in function *dmaDone_irq_handler*(), inserting toggles in *dmaDone_irq_handler*() is less overhead that proc filesystem must deal with. The original activity character will be modified as shown in Table 4.1.

As mentioned in Chapter 3, we need a measure program which do nothing

but an infinite loop. This program will be executed until the measurement procedure is complete. And we also insert trigger point on the program we want to measure. So we can observe the DMA after the end of the program we want to measure, and use the measure program to preform grouping on analysis tool.

### 4.2.2 Power Consumption Analysis Tool

In analysis program, we add three structures as shown in Figure 4.2. The structure str_group is used as an array. Every element segment the execution of program by different scenarios. The components of the structure str_gruop including group_table, start, stop, energy, power. The group_table is used to collect the elements in the same scenario. The elements start and stop record the starting time and the stoping time of element respectively. The elements energy and power will record the energy and the power of the measurement program respectively. Also, we add a structure str_mdma to observe the execution of DMA.

```
enum group_table
{
    G1,
    G2
};

struct str_group
{
    enum group_table table;
    UINT start;
    UINT stop;
    float energy;
    float power;
};

struct str_mdma
{
    UINT cnt;
    float power;
    float energy;
};
```

Figure 4.2: Structures for measuring the power consumption of DMA

# Chapter 5

# Experiments For DMA Power Consumption

This chapter, we will show experiments result of DMA power consumption measurement. First, we interpret the goal of those experiments, and then we will show results. Finally, we provide a formula for DMA energy consumption.

## 5.1 Onboard Experiments With DMA

Transactions of data of the codec are handled by the DMA. So we can use a playmp3 as the program we want to measure. As the program playmp3 is in execution, the DMA controller will start to transfer data between memory and codec. Our experiments are based on this program and a measure program as mentioned before. In order to increase the number of samples of G0 (means the scenario that the DMA is not active), we disable the DMA for the first half of time in the whole measurement. In our approach, the power consumption measured by the tool is including the power that DMA controller transfers the

data between memory and codec, and the power by the codec when the codec is making sound.

Because DMA controller have two main protocols, demand mode, and handshake mode. We will measure our DMA on both mode. The first experiment is a discussion of the access time of DMA. We will discuss the behavior of DMA execution time. Because we found the arrival time of DMA is steady. So we try to calculate the DMA per request to gain the stoping time of DMA. The second experiment is an evaluation of DMA to show our approach is feasible. Once we verify the feasibility of our approach on the second experiment, the third experiment will explore the measurement of DMA on different buffer size. And the last experiment will discuss the measurement on different bit rate of mp3 files.

### 5.1.1 Access Behavior

To get the starting time of DMA is quite sample, but to get the stoping time is more complicate. We may design a polling program to observe the stoping time of DMA. But it is not practical because we would effect the result of measurement. So we hope to avoid using polling as we can. In our case, we found that the arrival time of DMA is steady. Because the serial bus must feed the data to the codec in time or the codec would not make sound correctly. So we can calculate the execution time of DMA by the clock frequency of bus and

the transfer size per request of DMA as

$$ExeuctionTime = \frac{DSZ}{SCLK} \cdot TSC = \frac{BufferSize}{SCLK} \tag{5.1}$$

DSZ is the size of data that DMA performs one atomic transfer, default is 16 bit. SCLK is the clock frequency of serial bus that the DMA transfer. And TSC is the count that DMA controller perform data transfer per request. So DSZ multiply TSC and then divide SCLK will be the execution time of DMA access per request. In other words, if we know the buffer size of DMA and the clock frequency of serial bus, we can calculate the execution time to gain the stoping time.

The first experiment is a comparison between a measurement of the DMA execution time using polling and a measurement by calculating. We investigate two different protocols to see if the measurement of execution time using calculating is feasible. As shown in Figure 5.1, we can find the execution time of two different protocols are quite similar because the DMA access through the serial bus is on time. Although the execution time of different protocols are similar, but the access pattern are different. So the power consumption will be different. We will discuss on the following experiments. We also find the relation of the execution time of DMA and buffer size have direct ratio. And the execution time between two methods of measurement is very close. The measurement using polling is a little more then calculating. But the time will not over then 0.1%. So we have a conclusion, the execution time of DMA can

Table 5.1: Execution Time of Different Buffer Size

| Buffer Size | 16KB | 32KB | 64KB | 128KB | 256KB |
|---|---|---|---|---|---|
| HandSahke Mode | 92880 | 185761 | 371519 | 743039 | 1486077 |
| Demand Mode | 92880 | 185761 | 371519 | 743039 | 1486077 |
| Calculating | 92880 | 185761 | 371519 | 743039 | 1486077 |

be found by calculating.

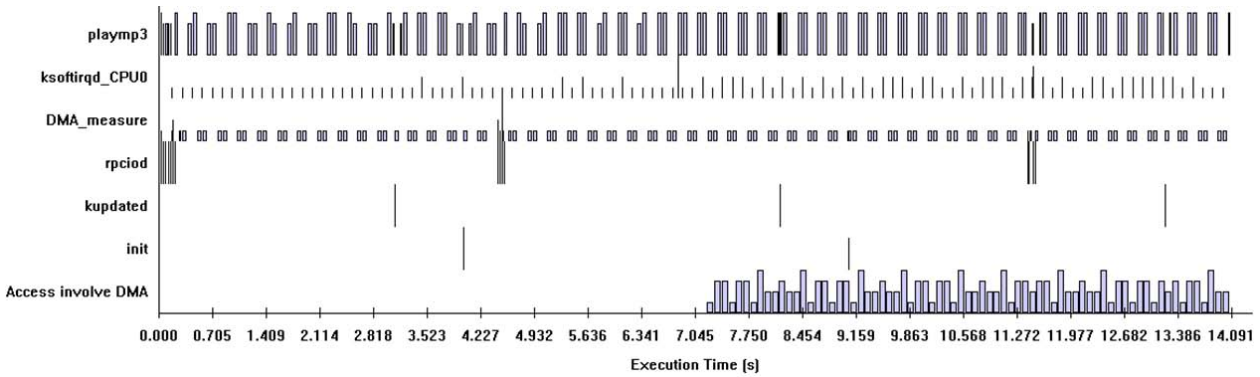
### 5.1.2 Evaluation of Measurement

Once we have the the starting time and the stoping time of DMA, we can use our tool to analysis the power consumption of DMA. In this experiment, we use the handshake mode as our criterion, and the buffer size is 16 KB. Figure 5.1 shows the result of measurement. In task playmp3, the DMA will soon occur that will causing lack of sampling time of G0. Because we want to increase the sampling number of G0, so in the first half of the measurement we disable the DMA. Furthermore, we can see the power consumption of DMA after the first half of measurement. The power consumption when the execution time involve DMA is instable because of the DMA would occur on different tasks. Finally the measurement result will be G1 - G0.

After one hundred measurements, the result of coefficient of variation as in table 5.2 is about 6.5%. The the coefficient of variation (CV) is a normalized measure of dispersion of a probability distribution. In other words, the vari-

Table 5.2: Average and Coefficient of Variation

| Average of 100 Data | Coefficient of Variation |
|---|---|
| 0.02037535 | 6.5% |




The Process Information in Measured Interval

| Pid | Command | Execution Time(s) | Energy (J) | Avg Power (W) |
|---|---|---|---|---|
| 30 | playmp3 | 6.516754 | 18.545586 | 2.845832 |
| 3 | ksoftirqd_CPU0 | 1.077387 | 2.921119 | 2.711299 |
| 29 | DMA_measure | 6.490649 | 17.406475 | 2.681777 |
| 27 | rpciod | 0.004980 | 0.014634 | 2.938546 |
| 6 | kupdated | 0.000304 | 0.000876 | 2.882344 |
| 1 | init | 0.000549 | 0.001561 | 2.842795 |
| Sample of G1 | | 3.144539 | 8.466534 | 2.692456 |
| Sample of G0 | | 3.346110 | 8.939941 | 2.671741 |
| Total | | 14.090624 | 38.890251 | 2.760009 |

Figure 5.1: The measurement result

ability of measurement result is not over then 6.5%. We think the error rate of the measurement result is acceptable so that our measurement approach can be verified.

### 5.1.3 Power Measurement on Different Buffer Size

After we verify the feasibility of our tool. The third experiment we investigate the power consumption of different buffer size. The result shows in Figures 5.3.

The bigger buffer size per request that DMA access, the less power consumption of DMA. Because, the behavior of DMA is the same in different size, the only different is the bigger buffer size that DMA uses, the less request that DMA requires. This will reduce the number of routines of DMA request like interrupt and some settings on driver level.

When DMA perform atomic transfer, the signal XnXDREQ will be synchronized to signal XnXDACK called double sync. Figure 5.2 shows two different modes of DMA. In demand mode, If XnXDREQ remains asserted, the next transfer starts immediately. Otherwise it waits for XnXDREQ to be asserted. In handshake mode, if XnXDREQ is deasserted, DMA deasserts XnXDACK in 2 cycles. Otherwise it waits until XnXDREQ is deasserted. In other words, demand mode will not perform double sync again until the next request occurs. So the effort that DMA in handshake mode is more than in demand mode.

#### 5.1.4 Power Measurement on Different Bit Rate

In the last experiment, we observe the application level with the same DMA access pattern in both modes. In other words, we try to fix the behavior of DMA. We use different bit rate of mp3 file as our variable. Those bit rate of mp3 files are 128KB, 160KB, and 192KB respectively. The bigger that bit rate of mp3 file is, the more effort that processor must handle. The purpose of this experiment is to see if the tool would be effected by the application level or not when our measurement subject is fixed. The result is just as we expect as

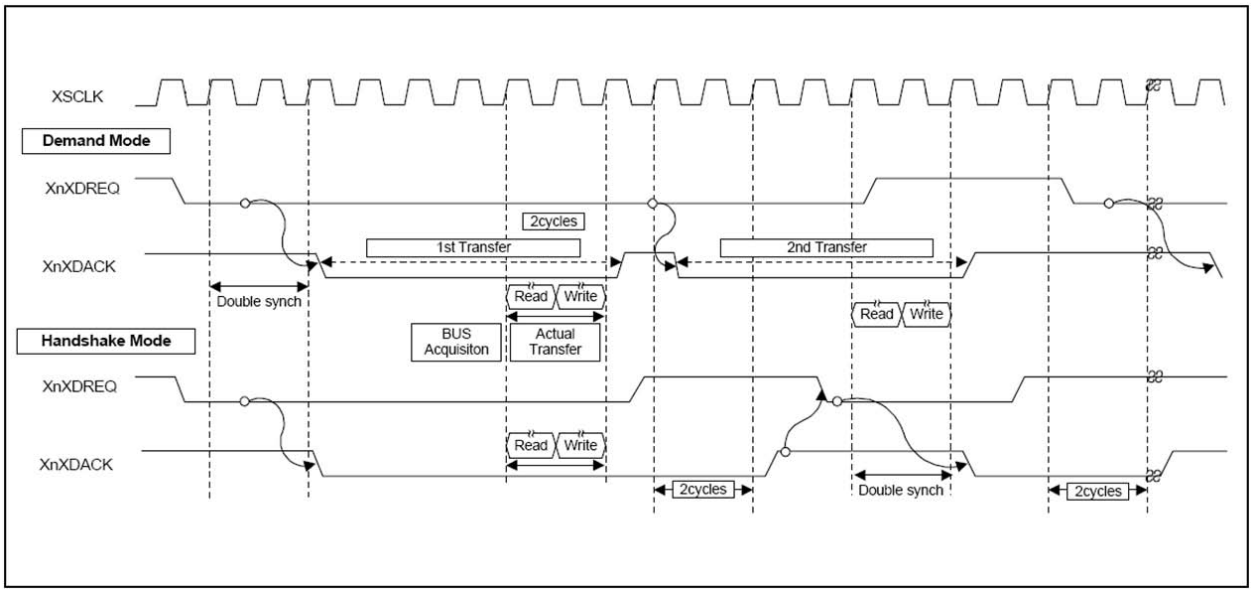



Figure 5.2: The measurement result

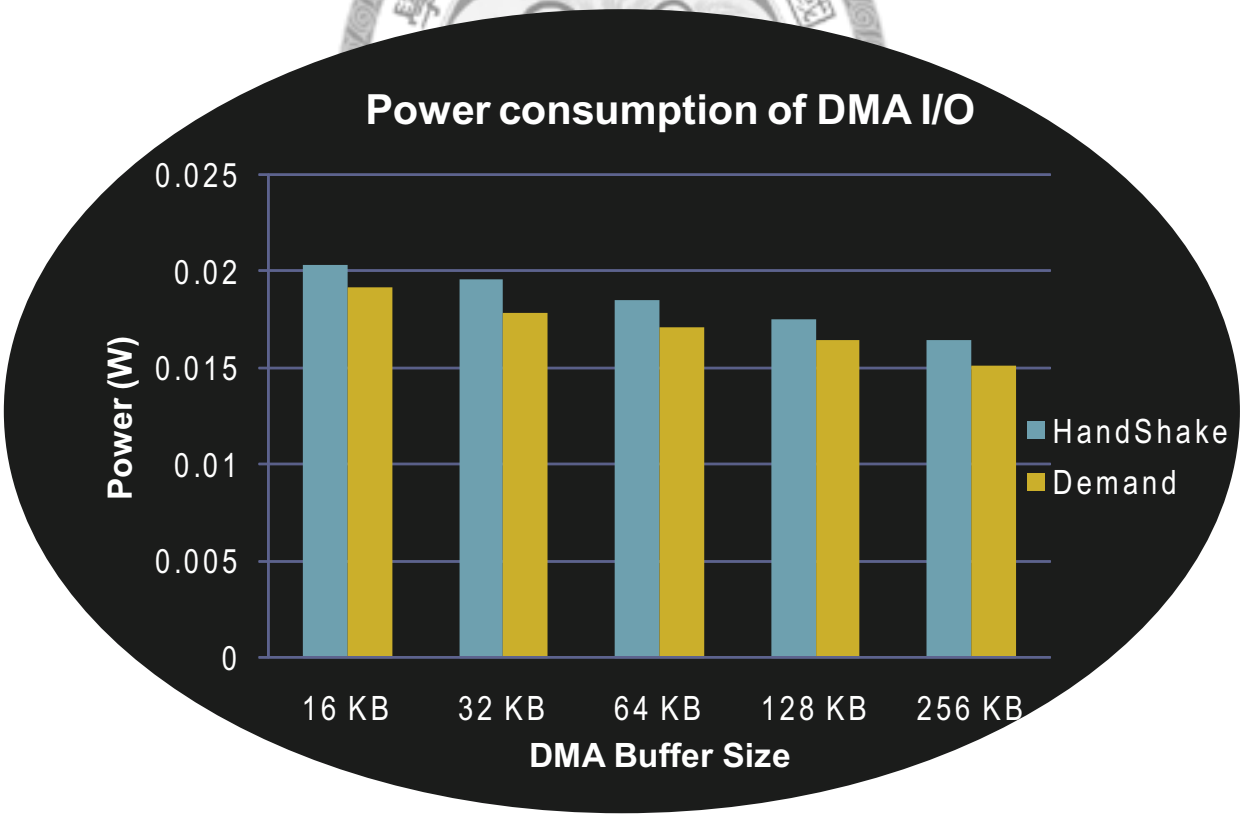



Figure 5.3: Power consumption of DMA with different buffer sizes

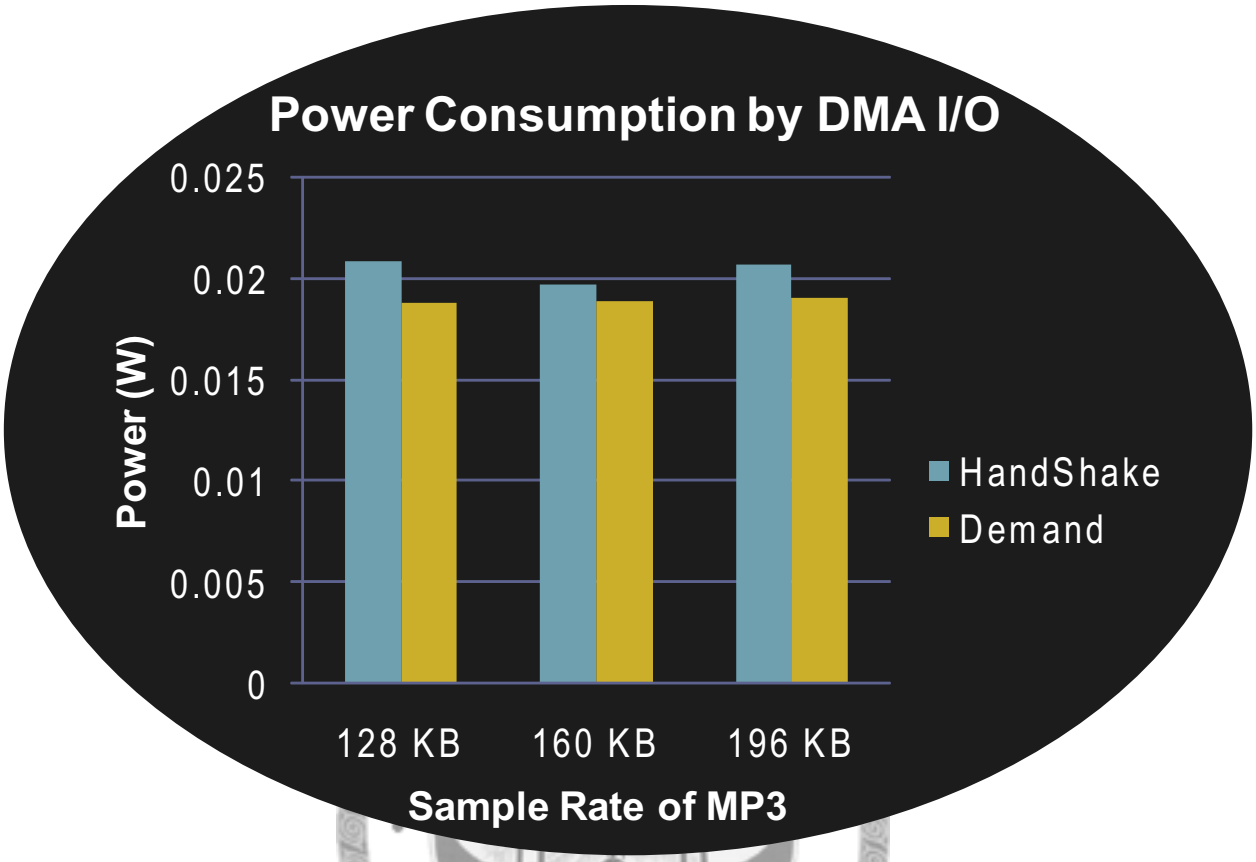



Figure 5.4: Power consumption of DMA with different bit rates

shown in Figure 5.4. The measurement result is that the bit rate of mp3 file does not effect the power consumption of DMA.

## 5.2 A Formula For DMA Power Consumption

Through those experiments, we verify the feasibility of this tool. So we can use a formula to compute the DMA's energy with our aid by using our tool. The formula is similar to the original formula as bellow

$$E(T) = \sum P_d(t_i) \cdot \Delta t = \frac{1}{S} \sum P_d(t_i), 1 \leq i \leq n \tag{5.2}$$

T denotes the sum of $t_i$, which $t_i$ means the execution time of every DMA request, $P_d(t_i)$ denotes average power consumption DMA consume, and n denotes the numbers of DMA request. So our tool provide a convenient way that perform the power consumption measurement not only the relation between different tasks, but also a hardware power consumption.

Current measurement tools can not verify our work, because the measurement result of our tool is totally different with simulation-based tools and physical-based tools. The measurement result of our tool is the additional power consumption of DMA when the target task use I/O devices. Simulation-based tools do not know the exact power consumption of specific I/O devices, and physical-based tools hardly tell apart the power consumption of I/O devices when they are used by a task. As a developer of application or system design, we believe that our analysis result is more helpful then simulation-based tools and physical-based tools.

# Chapter 6

# Conclusion and Future Work

Although we know the most valued power consumption comes from the CPU or memory, but the power consumption caused by I/O devices can not be unregarded, because there are many I/O devices in a computer system like disk, LCD, network etc. Therefore, the total power consumption from I/O devices would not less then the CPU or memory. In this thesis, we first enhance our task-based measurement tool by removing the redundant $do\_IRQ()$ toggles, and reduce about 80% the number of records that system must store. This tool has a problem that is when we want to measure a desired program, the result of the measurement could involve the power consumption of I/O devices that is independent with this program. In order to solve this problem, we propose a novel method of scenario grouping. Consequently, we can use this method to gain the power consumption of I/O devices that is separated from total power consumption of the system. Moreover, we provide an approach for developers who are eager to extend the hardware-aware functionality of task-

based measurement tools. At last, we use the approach we propose to implement the DMA-aware functionality for our tool as a demonstration, and use this new feature to observe the power consumption of DMA. This measurement result reflects the current power consumption of tasks directly, it is not possible to gain this information from physical-based tools or simulation-based tools, so we believe our tool can provide an easy-to-use, low cost, flexible way for developers to use.

For the future work, we would like to adopt more hardware scenarios for our tool like LCD, network etc. Since the more scenarios we get, the more precise measurement result we have. On the other hand, we would like to focus on the most important power consumption source: CPU. Since we can use a measurement program to observe the power consumption of I/O devices, we will try to design a similar way to analysis the additional power consumption of CPU. Finally, we hope our tool can provide a precise result for users on applications of the embedded system.

# Bibliography

[1] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *proceedings of the 27th Annual International Symposium on Computer Architecture*, page 83, June 2000.

[2] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings of IEEE Workshop Mobile Comput. Syst. Applicat.*, pages 2–10, Feb 1999.

[3] Chuling Hu, Danial A. Jimenez, and Ulrich Kremer. Toward an evaluation infrastructure for power and energy optimizations. In *proceedings of the 19th IEEE Internationl Parallel and Distributed Processing Symposium*, 2005.

[4] $\mu$Clinux, Embedded Linux/Microcontroller Project. http://www.uclinux.org/. 1998.

[5] Dongkun Shin, Hojun Shim, Yongsoo Joo, Han-Saem Yun, Jihong Kim, and Naehyuck Chang. Energy-monitoring tool for low-power embedded

programs. *IEEE Design and Test of Computers*, 19:7–17, July-Aug 2002.

[6] Sim-Panalyzer. http://www.eecs.umich.edu/ panalyzer.

[7] A. Sinha and A. P. Chandrakasan. Jouletrack - a web based tool for software energy profiling. In *Design Automation Conference, pages 220-225*, 2001.

[8] K. Komarov T. L. Cignetti and C. S. Ellis. Energy estimation tools for the palmtop:. In *International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems, pages 96-103*, 2000.

[9] Wei-Yu Tu and Chih-Wen Hsueh. A task-based energy consumption analysis tool in multi-tasking systems. Master's thesis, Department of Computer Science and Information Engineering National Chung Cheng University, 2006.

[10] Bokyung Wang and Suresh Singh. Computational energy cost of tcp. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 785–795, March 2004.

[11] Qiang Wu, V.J. Reddi, Youfeng Wu, Jin Lee, Dan Connors, David Brooks, Margaret Martonosi, and Douglas W. Clark. A dynamic compilation framework for controlling microprocessor energy and performance. In *Microarchitecture, 2005. MICRO-38. Proceedings. 38th Annual IEEE/ACM International Symposium*, Nov 2005.

[12] Wei-Kai Wu and Chih-Wen Hsueh. Energy consumption measurement and analysis toll on mp3. Master's thesis, Department of Computer Science and Information Engineering National Taiwan University, 2007.

[13] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: a cycle-accurate energy estimation tool. In *Proceedings of the 37th conference on Design automation*, pages 340–345, June 2000.