Profiler for Control System in System Level Design

Miaw Torng-Der, Yuki Ando, Shinya Honda, Hiroaki Takada, Masato Edahiro
Nagoya University
Nagoya, Japan
{miaw, y_ando, honda, hiro, eda}@ertl.jp

Abstract— This paper introduces a profiler architecture for control systems in system-level design. When designing a control system, we need to consider two things. The first thing is the asynchronous signal coming from sensors and actuators, which is called an interrupt request signal. The second thing is a process that should take a higher priority and be activated by an interrupt request signal, which is known as an interrupt handler. However, existing profilers cannot obtain the information of the interrupt request signal or the interrupt handler. Therefore, we develop a profiler for control systems can provide information about interrupts and interrupt handlers. Through an experiment, we demonstrate that the profiler provides information about interrupts with a 21.08% increase in area.

I. Introduction

In recent years, system-on-chips (SoC) for control systems have become large scaled and complicated. For example, an SoC for robot control systems requires processors, control IPs, and accelerators to realize high-performance control systems. Because systems like robot control system are so complicated and takes a lot of time to be developed, we developed a system-level design tool called SystemBuilder[1] to approach these problems. We evaluated its performance while exploring the architecture with SystemBuilder. If a system did not fit the constraints, we will need to find and solve the bottleneck. Therefore, bottleneck analysis methods are important for architectural exploration.

Hardware (HW) simulators and profilers are the two methods that are most used for finding bottleneck processes. HW simulators such as ModelSim[2] simulates the behavior of HW on PC. Thus, the accuracy and execution speed is a trade off for simulators. If the accuracy is required, the execution speed will be sacrificed. The other side, profiler like gprof[3] can get execution information. Designers add options when compiling and linking and the measuring code will be inserted automatically. When designers execute the program with measuring code on real machines, behaviors of functions like frequency and execution time will be profiled. Since profilers are executed and can gain the information on real machines, they can achieve better accuracy than simulators. Existing profilers such as gprof can only profile software (SW) processes but not profile HW processes. SoC design requires the partition of the HW and SW, so it is requirement to collect synchronously information.

SystemBuilder can provide execution information of both HW processes and SW processes[1]. However, the current profiler cannot provide profiles like asynchronous events (interrupt request signal), event processes, the processes activated by events (interrupt handler)[6]. To support bottleneck analysis for control systems, execution information of events and event processes are needed.

This paper proposes a profiler for control systems. The proposed profiler records not only the behavior of the interrupts, but also the statistics information of the control system. Contributions of the proposed method are:

- A function to record behavior of interrupt request signal (IRQ signal)
- A function to record statistical information
- A profiler that records the behaviors of SW processing, HW processing, and interruption at the same time

The main content of this paper is as follows: Section II introduces related works. Section III presents the details of control system model, architecture, and profiling flow of the profiler. Section IV demonstrates the effectiveness of the proposed method through a experimental study on a UART system. Section V gives conclusions and suggestions for future work.

II. Background

A. Related Research

There are two main approaches to realizing profilers, which are SW based profilers and HW based profilers. gprof is an example of a SW based profiler. It inserts measuring code by adding options when compiling and linking. It will profile the function by sampling program counter (PC). Hence gprof profiles by insert SW code, the execution overhead is large, and the accuracy will be reduced. Another problem of gprof is that the error rate grows is as the execution time is closing to the sampling time. Therefore, gprof does not suit for systems which have strict time constrain like embedded systems.

SnoopP[4] and LEAP[5] are examples for HW based profiler. SnoopP uses the field programmable gate array (FPGA)

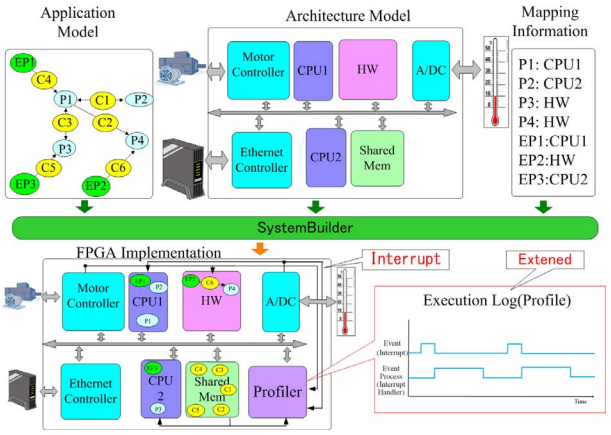


Fig. 1. Design flow of SystemBuilder

which is reconfigurable, to measure the execution time of functions. SnoopP will prepare same number segments as functions wants to profile. Each segment is composed by 2 comparator and 1 counter. Comparator will compare the PC value. If the PC value is in the specific address region, the counter will start counting execution time. LEAP is the improved version of SnoopP, which uses address hash and reduces counter to realized the area- and power-efficient. Since the SnoopP and LEAP are profiling not by insert SW code but by HW, the execution overhead is low and provide a clock accuracy execution informtion. Since SnoopP and LEAP are profiling from PC, they can only provide SW's execution information, but cannot provide HW's.

B. *System Level Design Tool: SystemBuilder*

We have developed a system level tool, SystemBuilder to support developing large scale and complicated SoC. Fig.1 shows the design flow of SystemBuilder. SystemBuilder can generate HW description, SW description, by input application model (C language),mapping information and architecture model. The SW description will be converted to executable SW binary file by compiler, and the HW description will be converted to HW image file by synthesis tools. The system than be executed by download SW binary file and HW image file to FPGA board.

SystemBuilder provides profile function to support system design efficiency, called process profiler. Process profiler provides both SW processes and HW processes execution information by its SW/HW mixed architecture. Different from Gprof, SnoopP and LEAP, Process profiler do not profile from PC. It profiles from channel communication with clock accuracy. Even though this method create some execution and area overhead, it's still tolerable[1]. Hence the existing SystemBuilder's profiler profiles from channel communication, important information like events and event processes which do not activated by channel communication cannot be profiled.

C. *Features of Control Systems*

Control System consists of sensors, actuators, network, external I/O and SoC that executes control processes. Typical control systems will first get information from I/O devices such as a sensor. Because it is not efficient to wait for I/O data without doing anything, processors will run other process until the interrupt request (IRQ) signal, which is an asynchrounous signal process by I/O data. Interrupt process will compute input data and output output data. The output data will later be sent to I/O device like actuator. As above, control systems provide feedback function by input, control processing, and than output. Various control system are realized by looping this feedback flow.

Take a collision avoidance system as an example. The collision avoidance system has ultrasonic sensors as its input device, and brake as output device. The main processor does not only handle the collision avoidance process, but it also required to do other processes such as GPS navigation, CD player, etc. Ultrasonic sensors measure the distance between the automobile and its surrounding obstacles around. If the distance between the automobile and the barricade is too short, the ultrasonic sensor will send IRQ signal to the main processor. The main processor executes various processes at ordinary time, but if the IRQ signal comes the collision avoidance process will take higher priority. If the ordinary process did not pause and the collision avoidance process did not take high priority, the automobile might bump into the barricade. The control process will tell the brake, an output device, to be activated or not. Therefore, IRQ signal and interrupt process play an important role in control system.

D. *Tasks for Providing Control Systems' Execution Information*

For providing execution information of control systems, we have two tasks. The first one is to support events and event processes. As section C mentioned, events and event processes play important roles in control systems. Therefore, information like maximum interrupt latency and how many interrupt occurs are wondered to know, when designing a control system.

The second is that the process profiler is not flexible since there is only one tracing pattern. The process profiler starts/ends by SW setting. But this method has two problems. The first one is the memory is not enough to save all execution information hence there is limitation for on-chip memory on FPGA board. The second problem is it is hard to analyze with only waveform. Sometimes we want to know information like maximum interrupt latency or the frequency of interrupts when desiging control systems. But it is hard to know these information with only waveform execution information. Therefore, statistical information is required.

III. Profiler for Control Systems Design And Implement

To solve the problems mentioned in section II.D, we focused on profiling information about interrupt. We have designed a profiling architecture that can profile interrupts with various tracing patterns.

A. Designing Profiler for Control Systems

Profiler for control system realizes the following functions.

- Extension of visualized information
  - SW processes, HW processes
  - SW event processes, HW event processes
  - Events
- Extension of statistical information
  - Interrupts/processes active times
  - Maximum interrupt latency
- Extension of profiler stopping methods
  - SW stop (conventional):(Fig.2(a))
  - SW stop (ring buffe):(Fig.2(b))
  - Trigger:(Fig.2(c))
  - SnapShot:(Fig.2(d))

First, we have extended visualized information for supporting control systems. SW processes and HW processes are already provided in process profiler. SW event processes, HW event processes, and event are the features of control systems which mentioned in section II.C. To analyze control systems, we need to get the execution information of SW/HW event processes and events. By recording, visualizing the execution information, we can know when do events occur, execution time of event process, or interrupt latency. It is also possible to analyze control systems' bottleneck or find out the state that did not satisfy the constraints by visualized execution information and other extensions.

Then, we have extended two statistical information functions, because it is hard to analyze the control systems with only visualized information. The first statistical information function is providing interrupts/processes active times. The on-chip memory of FPGA board has size limitation, it is not enough to store all execution information , especially for long execution time. Therefore, we have extended a function to provide interrupts/processes active times. This function has an advantage, that can record the information for a long time because the function only records active times. Another statistical information functions is providing maximum interrupt latency. This function can provide both SW and HW maximum interrupt latency. And we can set the maximum interrupt latency get from this function as trigger condition(mentioned later) to identify the bottleneck.

Last, we have extended 4 kinds of stopping methods to avoid the problem that memory cannot store all execution information. The first method is SW stop (conventional) (Fig.2(a)). The profiler starts and ends by setting from SW. The second method is SW stop (ring buffer) (Fig.2(b)). This method changed the conventional execution information storage memory to ring buffer memory. This method can avoid the problem of memory cannot store all execution information and records the execution information before stop point. The third method is Trigger (Fig.2(c)). Trigger can record specified patterns set by user for one time. The profiler will write the profile into the ring buffer continuously until the profiler stopped by trigger. When trigger has been activated, the profiler will stop recording and keep the execution information recorded until it is read by SW. We can know the execution information before and after specified patterns by Trigger. Therefore, we can specified the event process is affected by which process and vice versa. The fourth method is SnapShot (Fig.2(d)). SnapShot is like Trigger but it records the specified pattern for several times. By SnapShot we can know if the specified patterns will affect later or not.

B. Implementation and Profiling Flow of Visualized Execution Information

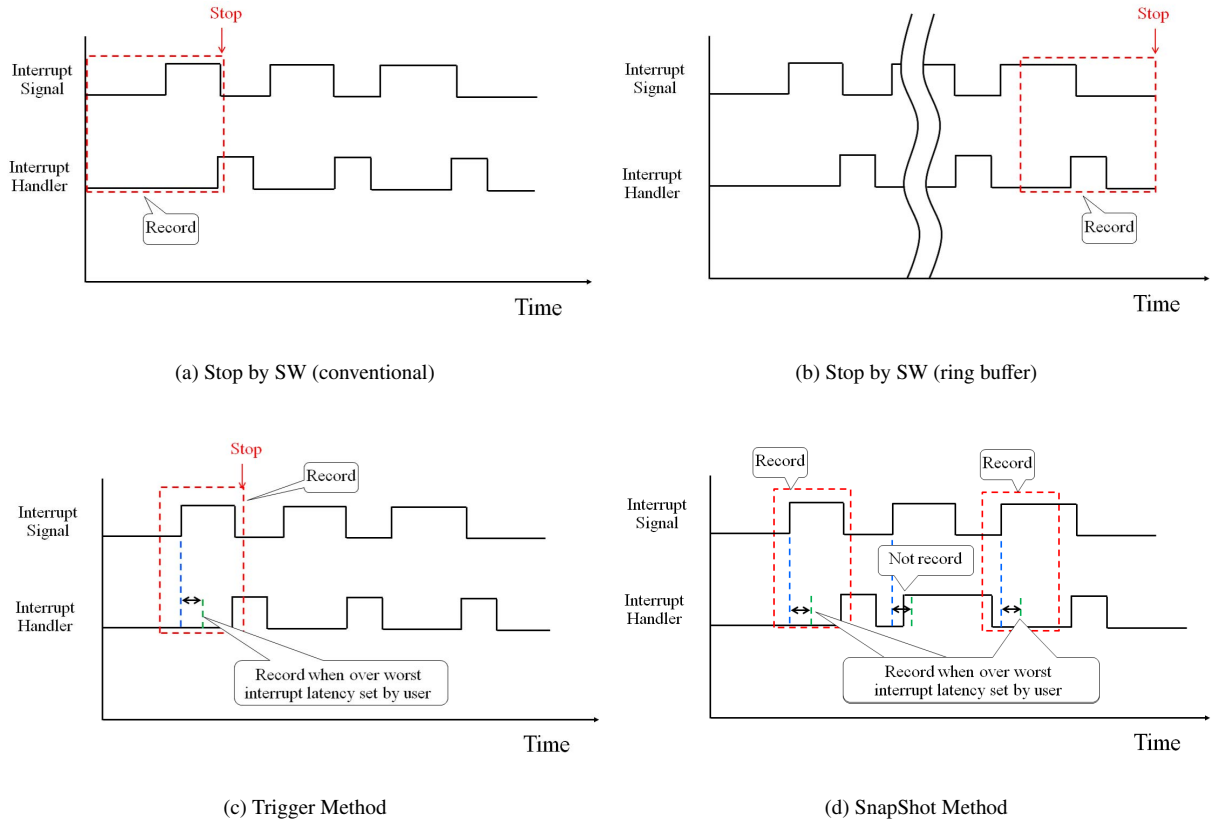
In this section, implementation and profiling flow of extended visualized information will be introduced.

B.1. Implemented Architecture

Fig.3 shows the architecture of profiler for control systems. HW implemented components are shown as below.

- Timer: Counting system clock.
- Trace register: Keeping states of processes and sending it to tracer. There are 16 trace registers in the trace register. The tracer access SW processes and SW event processes by bus. HW processes, HW event processes, and events access the trace register by 1 bit line. When the state of process changed, the value corresponding trace register will also be changed.
- Tracer: Tracing the execution information of processes, events and event processes from trace register. Then send the execution information to writer through FIFO. Execution information data is composed by the states of processes from trace register and relative execution time from timer. Upper 16 bits are relative execution time and lower 16 bits are process states.
- Writer:Writer writes the execution information data received from tracer into the execution information storage memory.
- Execution information storage memory: Memory for storing the execution information. After the end of the main processing, the execution information will be read out from the memory by SW.

Hence SystemBuilder uses Real-time operating system (RTOS)[7][8] to manage SW tasks and interrupts. Profiling SW execution information from RTOS is possible. When RTOS receives an IRQ signal, the executing task or interrupt



(a) Stop by SW (conventional)

(b) Stop by SW (ring buffer)

(c) Trigger Method

(d) SnapShot Method

Fig. 2. Profiler's stopping method

handler has lower priority will be saved into register and the interrupt handler has higher priority will be executed. After higher priority interrupt processing is finished, the RTOS will check if there are other higher priority interrupt handler or task. If there is a task with higher priority than the task saved, RTOS will enter the dispatch and executes the task with higher priority. If not, RTOS will return the saved task and resume it. Therefore, it is possible to profile SW execution information by insert measuring code before, after dispatch, and after task return. We have inserted 4 SW measuring function as follow to record SW execution information.

- Enter_Interrupt(): Being called before interrupt process. First, this function gets the ID of task/interrupt handler which is executing and set the value of corresponding trace register to 0(inactive state) because the task/interrupt will be saved by RTOS. Then, the function will gets the ID of interrupt handler which will be executed, and set the value of corresponding trace register to 1(active state).
- Leave_Interrupt(): Being called after interrupt process. The function gets the ID of interrupt handler which will finish and set the value of corresponding trace register to 0(inactive state).
- Enter_DSP(): Being called when enter the dispatcher. The function reads the task ID which will finish and set the value of corresponding trace register to 0(inactive state). That is because if task changing is required, RTOS will enter dispatch and change the task. If task changing is not required, RTOS will return the saving task.
- Leave_DSP(): Being called after dispatch processed. Hence both after dispatcher processed and return, the task will be executed, this function read the ID which will be executed and set value of corresponding trace register to 1(active state).

B.2. Profiling Flow

The profiling flow is showed as below.

1. Change system description: Adding profiling start/end function to the place where the point to start/end the profiling are wondered.
2. Synthesis by SystemBuilder: Adding profile enable option when the system is synthesized. SystemBuilder will implement the profiler automatically. At the same time, the corresponding ID file of processes, event processes, and events will also be generated.

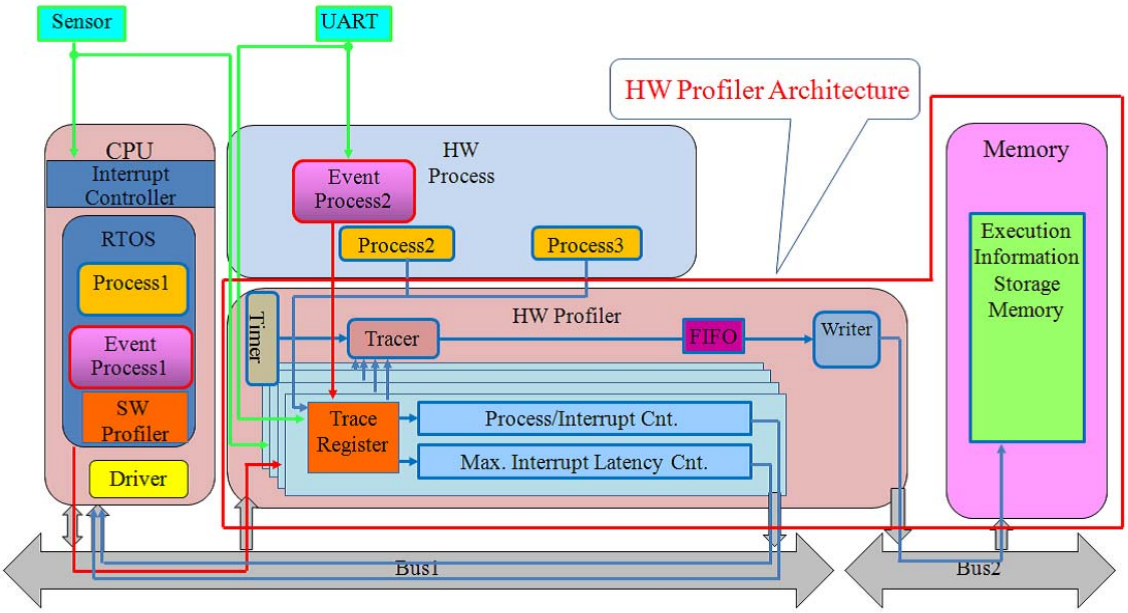


Fig. 3. HW profiler architecture

3. Generate binary/image file: Using external tool and compiler to generate SW binary file and HW image file.

4. Get and analyze the execution information: Getting the execution information by download and execute the SW binary file and HW image file on FPGA board. By entering execution information data and ID file, the analyze tool will generate the execution information to a waveform file. The waveform can be used for bottleneck analysis.

C. *Implementation of Statistical Information*

As section II.D mentioned, it is hard to analyze statistical information like maximum interrupt latency or interrupt/processor active times with only waveform execution information. To solve these problem, the statistical information has been designed and implemented. The implementation for providing statistics information will be described in this section.

C.1. Interrupts/Processes Active Times

As Fig.3 showed, the interrupts/processes active times counters (Process/Interrupt Cnt. in Fig.3) are connected with the interrupts or processes want to profile through the corresponding trace register with 1 bit line. The counter counts up when active state is detected, in the range sets by user. After measuring, the value of interrupt/processor active times counter will output by SW.

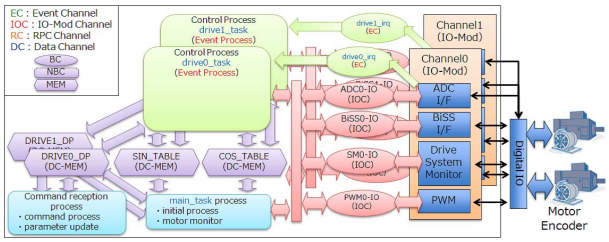


Fig. 4. System for experiment

C.2. Maximum Interrupt Latency

The maximum interrupt latency counters (Max. Interrupt Cnt. in Fig.3) are connected with the event and event process wants to profile through corresponding trace register with 1 bit line each. The counter will starts to count up since the interrupt signal is detected, and stop since the interrupt handler active is detected. If the interrupt latency is larger than former, the value of output register will than be overwriten. After measuring, the value of maximum interrupt latency counter will output by SW.

IV. Experimental Study

We evaluate the proposed profiler by using a 2 axis motor control system. The experimental enviroment is shown as follows.

- FPGA board: Altera DE-2 115
- FPGA chip: Cyclone IV
- Processor: NiosII（100MHz）

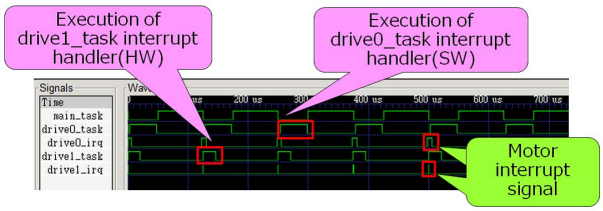


Fig. 5. Visualized profiling result

TABLE I
INTERRUPTS ACTIVE TIMES

Process/Event/Event Process Name	Activate times
drive0_irq	175
drive1_irq	175

- Altera multiaxis control board

Fig.4 shows the experiment system model. This system includes drive0_irq and drive1_irq as event, drive0_task as SW event process drive1_task as HW event process, main_task and command reception process as SW process. Main_task initials the sin/cos table, control parameter Drive System Monitor, PWM, and ADC at first. Then main_task continues monitoring motor states. If an error occurs, main_task will initial the control system again. Command reception process updates the control parameter when command is received. Every brushless DC (BLDC) motor has a control process, an event channel, and control parameters. ADC interrupt has a 16KHz frequency. When ADC interrupt occurs, the control process will calculate and update the value of PWM.

Then, the profiler will record the execution information of drive0_irq, drive1_irq, drive0_task, drive1_task and main_task for 11ms, after receives profiling start command.

Fig.5 is a part of visualized profile for 2 axis motor control system. From Fig.5, we have verified the visualized execution information of SW event process, HW event process and event can be provided by the proposed profiler. We can confirm that event and event process occurs in the order and when SW event process is executing, the SW process will be in an inactivated state. Therefore, we can make sure that the execution information is correct.

Table.I shows the profiling result of interrupts active times. Because the interrupt occurs every 62.5μs and profiling for 11 ms, the interrupt should activate for 176 times. The result of interrupts/processes active times has 1 time difference due to the timing of the profiler starts/end profiling. Therefore, the profiling result can be considered as a correct result.

TABLE II
MAXIMUM INTERRUPT LATENCY

	Implement	Maximum Interrupt Latency(Cycle)
drive0_task	SW	332
drive1_task	HW	2

Table.II shows the maximum interrupt latency of drive0_task and drive1_task. From Table.II, it shown that HW event will activate in 2 cycles when event occurs, but the SW event may need 332 cycles. That is because the SW event needs to wait for other interrupt handlers which have higher priority, but HW the event does not.

The area of the proposed profiler has a 21.08% area increase compared with process profiler, which is because the profiler for control system has 16 interrupts/processes active times HW modules and 8 maximum interrupt latency HW module implemented dynamically.

V. CONCLUSION AND FUTURE WORK

In this paper, we introduced a RTOS/HW mixed based profiler for control systems. The proposed profiler records SW execution information by inserting measuring code in RTOS so it can record both event processes and processes executed on the processor. Through its HW architecture, the HW processes, HW event processes, and event can be provided. In order to execute information recording more flexibly, and to analyze information more efficiently, we have designed functions providing statistical information and profiler stopping methods. The statistical information has been implemented in this paper. The implementation of profiler stopping methods and auto-synthesis tools will be done in the future. We also plan to extend statistics information and profiler stopping methods, including average interrupt latency, minimum interrupt period and worst case execution time etc.

ACKNOWLEDGEMENTS

This work is in part supported by STARC(Semiconductor Technology Academic Research Center).

REFERENCES

[1] S. Shibata, S.Honda, H.Tomiyama, and H. Takada, Advanced System-Builder: A Tool Set for Multiprocessor Design Space Exploration, In Proc. International SoC Design Conference (ISOCC), pp.79-82, 2010

[2] Mentor Graphics Modelsim http://www.mentor.com/products/fv/modelsim/

[3] GNU gprof., https://sourceware.org/binutils/docs-2.21/gprof/

[4] L. Shannon and P. Chow, Using reconfigurability to achieve real-time profiling for hardware/software codesign. FPGA'04, pp.22-24, 2004

[5] L. Shannon and P. Chow, Low-Cost Hardware Profiling of Run-Time and Energy in FPGA Embedded Processors, Application-Specific Systems, Architectures and Processors (ASAP), pp.61-68, 2011

[6] Y. ANDO, Y. ISHIDA, S. HONDA, H. TAKADA, M. EDAHIRO, System-level design method considering the interrupt processing, IEICE Technical Report, vol. 113, no. 320, pp.119-124, 2013

[7] Toppers ASP kernel, https://www.toppers.jp/asp-kernel.html

[8] Toppers FMP kernel, https://www.toppers.jp/fmp-kernel.html