

POWER MANAGEMENT SCHEME FOR FPGA-BASED
CUSTOMIZABLE INTERNET OF THING (IoT) SENSOR
NODES

MASTER OF SCIENCE (COMPUTER SCIENCE)

FACULTY OF INFORMATION AND COMMUNICATION
TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN
MARCH 2022

**POWER MANAGEMENT SCHEME FOR FPGA-BASED
CUSTOMIZABLE INTERNET OF THINGS (IoT) SENSOR NODES**

By

A dissertation submitted to the Department of Computer and Communication
Technology,
Faculty of Information and Communication Technology,
Universiti Tunku Abdul Rahman,
in partial fulfillment of the requirements for the degree of
Master of Science (Computer Science)
March 2022

ABSTRACT

POWER MANAGEMENT SCHEME FOR FPGA-BASED CUSTOMIZABLE INTERNET OF THINGS (IoT) SENSOR NODES

Field-programmable gate array (FPGA)-based sensor nodes are popular for their flexible design approach and field re-configurability. RISC32, one of the recent Internet of things (IoT) processors proposed for developing FPGA-based sensor nodes, has the ability to reconfigure the microarchitecture dynamically according to program workload. This helps in reducing the dynamic energy consumption required for completing program execution. However, such an approach does not minimize the static energy consumption, which is important in FPGA-based systems. In this study, two known low-power techniques compatible with FGPA were implemented in RISC32: clock gating (CG) and dynamic voltage–frequency scaling (DVFS) techniques. In addition, a software tool (Energy Reduction Program Analyzer) was developed to estimate the parameters that can configure the sensor node to achieve minimum energy consumption, targeting the typical IoT application scenario. Experimental results show that the low-power techniques applied in this work can reduce the energy consumption by 47% compared to the original RISC32. In particular, combining low-power techniques has shown improved

energy saving compared to single low-power technique: 45% improvement versus CG, 11.54% improvement versus DVFS, and 40% improvement versus partial reconfiguration.

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my supervisors, Dr. Chang Jing Jing and Mr. Mok Kai Ming, for their guidance, inspiration, and enthusiasm, which enabled the completion of this research project. I would also like to give a special appreciation to our research team member, Dr. Lee Wai Kong, for his advice on the practical IoT application and the experimental flows prior to the completion of the experimental work. Last but not least, I would like to thank to my family for their full support in order for me to pursue my interest.

APPROVAL SHEET

This dissertation entitled “**POWER MANAGEMENT SCHEME FOR FPGA-BASED CUSTOMIZABLE INTERNET OF THINGS (IoT) SENSOR NODES**” was prepared by TAN BENG LIONG and submitted as partial fulfillment of the requirements for the degree of Master of Science (Computer Science) at Universiti Tunku Abdul Rahman.

Approved by:



(Dr. Chang Jing Jing)

Date: 7th March 2022

Supervisor

Department of Computer and Communication Technology

Faculty of Information and Communication Technology

Universiti Tunku Abdul Rahman



(Mr. Mok Kai Ming)

Date: 7th March 2022

Co-supervisor

Department of Computer and Communication Technology

Faculty of Information and Communication Technology

Universiti Tunku Abdul Rahman

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

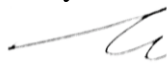
Date: 7th March 2022

SUBMISSION OF DISSERTATION

It is hereby certified that **TAN BENG LIONG** (ID No: **17ACM06813**) has completed this dissertation entitled "**POWER MANAGEMENT SCHEME FOR FPGA-BASED CUSTOMIZABLE INTERNET OF THINGS (IoT) SENSOR NODES**" under the supervision of **Dr. Chang Jing Jing** (Supervisor) from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology , and **Mr. Mok Kai Ming** (Co-Supervisor) from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my dissertation in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,



(TAN BENG LIONG)

DECLARATION

I hereby declare that the dissertation is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTAR or other institutions.



Name: TAN BENG LIONG

Date: 7th March 2022

TABLE OF CONTENTS

	Page
ABSTRACT	ii
ACKNOWLEDGEMENTS	iv
APPROVAL SHEET	v
SUBMISSION SHEET	vi
DECLARATION	vii
LIST OF TABLES	x
LIST OF FIGURES	xi
LIST OF ABBREVIATIONS/NOTATION/GLOSSARY OF TERMS	xii
CHAPTER	
CHAPTER 1 INTRODUCTION	1
1.1 Background	1
1.2 Problem Statement	6
1.3 Objectives	6
CHAPTER 2 LITERATURE REVIEW	8
2.1 Low-Power Techniques	8
2.1.1 Clock Gating	8
2.1.2 Power Gating	8
2.1.3 Dynamic Voltage and Frequency Scaling	9
2.1.4 Partial Reconfiguration	11
2.1.5 Power Management Module	12
2.1.6 Other Techniques	13
2.2 Research gap	13
CHAPTER 3 RISC-LP: THE PROPOSED LOW-POWER FPGA-BASED SENSOR NODE	15
3.1 System Overview	15
3.2 Power Management Unit	18
3.2.1 Development of Clock Gating	22
3.2.2 Development of Dynamic Voltage and Frequency Scaling	25
3.4 Partial Reconfiguration	41
3.5 Xilinx Vivado Synthesis Results	43
CHAPTER 4 ENERGY REDUCTION PROGRAM ANALYZER	45
4.1 Step 1: Program Segmentation Process	48
4.2 Step 2: Link the Nodes	49
4.3 Step 3: Simulate Program for Clock Cycle Calculation.	50

4.4	Step 4: Node classification	51
4.5	Step 5: Identify Program Type	52
4.6	Step 6: Search for a Potential Node to Insert Low-Power Instructions.	53
4.7	Step 7: Insert Low-Power Instructions based on the Program Type	54
4.7.1	Polling-based Program	54
4.7.2	Timer-Interrupt-based Program	57
4.8	Example of EFRA on tuning frequency to optimal value.	59
CHAPTER 5 ENERGY REDUCTION EXPERIMENT AND RESULT		62
5.1	Polling-based Test Program	67
5.1.1	Program Behavior	67
5.1.2	Result based on Polling Test Program	67
5.2	Interrupt-based Test Program	70
5.2.1	Program Behavior	70
5.2.2	Result based on Interrupt Test Programs	71
5.3	Comparison with Existing Works.	73
CHAPTER 6 CONCLUSION		75
REFERENCES		

LIST OF TABLES

Table	Page
Table 3.1: Specification of the RISC32-LP.	17
Table 3.2: DVFS-related input and output pins in the PMU.	18
Table 3.3: CG-related input and output pins in PMU.	20
Table 3.4: IO CG control unit's LUT.	23
Table 3.5: CPU core CG control unit's LUT.	23
Table 3.6: F–V pairs.	26
Table 3.7: The number of FFs used in each range of LDMC.	29
Table 3.8: SPI command values.	29
Table 3.9: DVFS-CU input and output pins.	31
Table 3.10: Description of each FSM state for DVFS-CU.	34
Table 3.11: Output pins value of each FSM state for DVFS-CU (DVFS-CU).	36
Table 3.12: CDC circuit input and output pins.	38
Table 3.13: FPGA resource difference between RISC32 and RISC32-LP.	44
Table 4.1 Pseudo code of the example user program.	47
Table 4.2 Assembly code of the example user program.	47
Table 4.3 Description for symbols used in this chapter.	48
Table 4.4 Pseudo algorithm used by ERPA to insert DVFS and TMA instructions in a polling-based program.	54
Table 4.5 Pseudo algorithm used by ERPA to insert DVFS and TMA instructions in an interrupt-based program.	57
Table 5.1 Short description for each test cases.	66
Table 5.2 Energy consumption of polling test program.	67
Table 5.3 Energy saving of polling test program.	68
Table 5.4 Energy consumption of interrupt test program.	71
Table 5.5 Energy saving of interrupt test program.	72
Table 5.6 Comparison with the existing low power techniques.	73
Table 6.1: Static and dynamic energy breakdown of 256-byte interrupt-based test program at 40 MHz.	77

LIST OF FIGURES

Figure	Page
Figure 3.1: RISC32-LP architecture with IoT program compilation process..	15
Figure 3.2: DVFS and CG components in the PMU.	17
Figure 3.3: PMU block diagram in RISC32-LP.	18
Figure 3.4: Clock gated module in RISC32-LP.....	24
Figure 3.5: BUFHCE's waveform.	24
Figure 3.6: Voltage regulator's schematic.	27
Figure 3.7: LDMC circuitry.	28
Figure 3.8: Block diagram of DVFS-CU.	31
Figure 3.9: State diagram for DVFS-CU FSM.	33
Figure 3.10: clk1 and clk2 signals from the PLL.....	34
Figure 3.11: Multi-clock domain in RISC32-LP and CDC circuit location. ...	37
Figure 3.12: Block diagram of the CDC circuit.....	38
Figure 3.13: Details of the CDC circuit.	39
Figure 3.14: Example waveform of the CDC circuit.	40
Figure 3.15: RISC32's microarchitecture in the ME mode.	41
Figure 3.16: RISC32's microarchitecture in the PE mode.	41
Figure 3.17: Static and reconfigurable region in RISC32's core (Kiat et al., 2020).	42
Figure 4.1 Flow of ERPA analysis.	46
Figure 4.2 Assembly code of the example user program and the result after segmentation process.	49
Figure 4.3 Example of the cyclic node graph created.....	50
Figure 4.4 Q _{NODE} content based on the example user program.	50
Figure 4.5 Classification of nodes in the cyclic graph.....	52
Figure 4.6 Node arrangements in Q _{NODE} based on the example user program (timer-interrupt-based).....	53
Figure 4.7 ERFA steps on insert SB instruction for DVFS feature in RISC32- LP.....	59
Figure 5.1 Hardware setup for energy measurement during the experiment...	62
Figure 5.2 Power consumption of polling test program (TST _{MAX}) with 256- byte data size.....	67
Figure 5.3 Energy saving for (TST _{PR} vs TST _{MAX}) and (TST _{COMB} vs TST _{MAX}). TST _{PR} did not achieve energy saving from 64-byte to 256-byte due to overhead.....	68
Figure 5.4 Energy saving for (TST _{CG} vs TST _{MAX}), (TST _{DVFS} vs TST _{MAX}), (TST _{COMB} vs TST _{MAX}) and (TST _{ERPA} vs TST _{MAX}). Energy saving in TST _{COMB} is lower than TST _{ERPA} due to PR overhead shown in Figure 5.3.	69
Figure 5.5 Power consumption of interrupt test program (TST _{MAX}) with 256- byte data size.....	70

LIST OF ABBREVIATIONS

ASIC	Application-specific integrated Circuit
BER	Bit error rate
C	Transistor capacitance
CDC	Clock domain crossing
CG	Clock gating
DVFS	Dynamic voltage and frequency scaling
DVFS-CU	DVFS control unit
ERPA	Energy Reduction Program Analyzer
f_{clock}	Operating clock frequency of a system
FF	Flip-flop
FPGA	Field-programmable gate array
ICAP	Internal configuration access port
I_{ccint}	Current flow in the FPGA chip
I_{GATE}	Gate leakage
I_{GIDL}	Gate-induced drain leakage
IoT	The Internet of Things
I_{REV}	Reverse bias junction leakage
I_{static}	Electric current in an idle system
I_{SUB}	Sub-threshold leakage
LDMC	Logic delay measurement circuit
LUT	Look-up table
ME	Multi-cycle microarchitecture execution
MFF	Main flip-flop

P	Power consumption
P_{dym}	Dynamic power consumption
PE	Pipeline microarchitecture execution
PLL	Phase-locked loop
PMU	Power management unit
PR	Partial reconfiguration
$P_{short_circuit}$	Short circuit power consumption
P_{static}	Static power consumption
R	Resistance
RAM	Random access memory
RISC	Reduced instruction set computer
RU	Reconfigurable unit
SFF	Slow flip-flop
SoC	System on chip
TMA	Toggle microarchitecture
V_{ccint}	Voltage supply for FPGA chip
V_{dd}	Voltage supply of a system
$V_{shunt_resistor_anp}$	Amplified potential different on the shunt resistor
The following abbreviations are used in Chapter 4	
$ndata_dep_acc$	Number of data dependencies accumulated for consecutive IO nodes, starting from CPU node until <i>node_lp_config</i> is detected.
<i>node_lp_config</i>	Potential node in which to insert low-power instruction.
$Q_{ndata_dep_acc}$	Queue that stores $ndata_dep_acc$ of each check point.

Q_{NODE}	Queue that stores the nodes' properties based on the program execution flow; nodes in the loop are unrolled before storing.
$Q_{\text{node_lp_config}}$	Queue that stores the indices that correspond to <i>node_lp_config</i> .
$Q_{\text{TIO_acc}}$	Queue that stores $T_{\text{IO_acc}}$ at each check point.
T_{EXE}	$T_{\text{NODE_TOTAL}} * 25 \mu\text{s}$ (period of the 40 MHz frequency); program execution time (in μs) derived from $T_{\text{NODE_TOTAL}}$.
T_{IDLE}	$(T_{\text{TIMER_INT}} - T_{\text{NODE_TOTAL}}) * 25 \mu\text{s}$. The available program slack for a timer interrupt-based program, converted from clock cycle count to microseconds (μs).
$T_{\text{IO_acc}}$	Total clock cycle count accumulated for consecutive IO nodes, starting from a CPU node until <i>node_lp_config</i> is detected.
$T_{\text{NODE_TOTAL}}$	Total clock cycle count (from all nodes) needed to complete a superloop
$T_{\text{o_DVFS}} (\mu\text{s})$	Overhead (in μs) when changing operating frequency and voltage.
$T_{\text{o_PR}} (\mu\text{s})$	Overhead (in μs) for the PR operation.
$T_{\text{TIMER_INT}}$	Timer interrupt value in clock cycle count (measured at maximum frequency, 40MHz)

CHAPTER 1

INTRODUCTION

1.1 Background

Internet of Things (IoT) refers to a system of interrelated computing systems which enables them to exchange data with each other, or to the main host (e.g., cloud server, computer, or smartphone). IoT devices are typically embedded with electronics, software applications, sensors, actuators, and network connectivity. In the recent decades, IoT technology has brought a significant transformation in the manner in which we live and work. For example, people are now able to monitor and control their vehicles, home appliances, and even their pets from long distance as long as there is an Internet connection for both devices. With IoT becoming increasingly pervasive in our everyday lives, the demand and requirement for IoT devices have also become increasingly complex.

Like other electronic devices, a processor is compulsory for IoT devices to process outgoing data or incoming instruction. However, the design of an IoT processor possesses an additional challenge in terms of power efficiency. This is attributable to the nature of IoT devices, which are often portable, lightweight, and largely depend on battery lifetime. Hence, in this study, low-power techniques were applied in the RISC32 processor to improve its power efficiency.

The power dissipation associated with a processor can be classified into two categories: dynamic power dissipation and static power dissipation. Dynamic power dissipation is mainly caused by switching activities during the

operation of the processor. The content stored in the register or memory of the processor will always change, which means the charge and discharge of the transistor will occur.

Static power dissipation is caused by leakage of current in a gate, which is classified into four types:

- Sub-threshold leakage (I_{SUB}): the current flows between the source and drain of a MOSFET when a transistor is in the weak inversion region.
- Gate leakage (I_{GATE}): the current that flows between the gate and to substrate through the oxide layer due to the gate oxide tunneling and hot carrier injection.
- Gate-induced drain leakage (I_{GIDL}): the current leak between the drain and substrate, which is caused by a high field effect in the MOSFET drain.
- Reverse bias junction leakage (I_{REV}): current leak caused by minority carriers drift in the reversed-biased regions.

More specifically, the total power consumption of a processor can be described using the following equation:

$$\mathbf{P} = \mathbf{P}_{dym} + \mathbf{P}_{short_circuit} + \mathbf{P}_{static}$$

Where P_{dym} is the dynamic power consumption, $P_{short_circuit}$ is the short-circuit power consumption, and P_{static} is the static power consumption. The dynamic power component is given as $P_{dym} = \sum(C * V_{dd}^2 * f_{clock})$, which is mainly caused by the switching of activities during the operation of the processor. The $C * f_{clock}$ in the equation represents total switching activities (charge or discharge) taking place in a transistor per second, where C is the capacitance

of transistor in the system and f_{clock} is the operating clock frequency of the system. The rate of transistor charge and discharge depends on f_{clock} . For example, a processor with higher clock frequency will have a shorter clock period resulting in more switching activities throughout the program execution. The V_{dd} denotes the voltage level supplied to the system. A system with a higher voltage supply can switch the logical level (“0” and “1”) of the gate faster. This means, a processor with higher f_{clock} (performance) will require a higher V_{dd} to ensure fast logical level switching within a short f_{clock} period.

Static power, on the other hand, is represented by $P_{\text{static}} = V_{\text{dd}} * I_{\text{static}}$, where V_{dd} is the voltage supply to the system and I_{static} is the current consumed by the system when the processor is turned on without the switching activities (idle state). However, since the static current has the relationship of $I_{\text{static}} = V_{\text{dd}}/R$, static power can be represented by $P_{\text{static}} = V_{\text{dd}}^2/R$, where R is the resistance that is fixed for an implemented design. Therefore, the static power can be reduced by lowering the voltage supply. However, it should be noted that a system requires a minimum voltage supply to operate correctly. Hence, the voltage supply cannot be reduced below the minimum level.

Some of the popular techniques worth mentioning are dynamic voltage and frequency scaling (DVFS), clock gating (CG), power gating, etc. However, not all low-power techniques are suitable to be implemented in all processors. For example, Peng et al. (2013) implemented the instruction-cycle-based dynamic voltage scaling on digital signal processor, which has both complex and simple instructions, but this technique does not help much in simple reduced instruction set computer (RISC) processor since most of the RISC

instructions have similar latency. Another technique called power gating is also not suitable for designs on Field-programmable gate array (FPGA), as FPGAs do not offer to support the low power feature.

In this project, FPGA was used to implement the RISC32 processor as an IoT processor. Therefore, we could not apply the low-power techniques that require modifications on transistor level. In this work, partial reconfiguration (PR) and CG (Sterpone et al., 2011) are combined with DVFS to achieve significant reduction of energy consumption.

CG is one of the simplest low-power techniques. It reduces dynamic power by deactivating the clock signal supplied to circuit regions, which are idle during run-time. The clock signal is basically not allowed to switch.

DVFS, one of the commonly used low-power techniques, is an obvious choice for RISC32. It tunes supply voltage and operating frequency for power reduction. Since both the dynamic and static power consumption are related to supply voltage V_{dd} , reducing V_{dd} can decrease power consumption significantly. At the same time, operating frequency (f_{clock}) in the FPGA can be reduced to further decrease dynamic power consumption. In the past, DVFS has been proposed in many application-specific integrated circuit (ASIC) designs, but it is challenging to adopt the same in an FPGA because it is not widely supported by existing development tools. Recently, Nunez-Yanez et al. (2015, 2017) and Wu et al. (2014) proposed techniques to enable DVFS implementation in FPGA devices, opening up the possibility of applying DVFS in FPGA-based IoT sensor nodes.

Besides DVFS, PR feature in the FPGA has been proposed (Kiat et al., 2020) to reduce power consumption. Energy consumption is reduced by

switching between two microarchitectures (pipeline and multi-cycle) through PR. Switching of microarchitecture is done by a customized instruction, toggle microarchitecture (TMA), based on the characteristics of the given tasks.

Although DVFS and the PR technique can effectively reduce the power consumption in FPGA-based sensor nodes, configuring the system to work at the lowest power level may not be beneficial due to the following challenges:

- 1) When power consumption is reduced by DVFS (i.e., reducing V_{dd}), the operating frequency must be reduced to avoid timing errors, which, in turn, decreases execution speed. This can result in higher total energy consumption if the designated tasks take a long time to complete.

- 2) Some IoT applications require data to be sent at fixed intervals to ensure the timeliness of data processing at the gateway or in a cloud server. Energy-efficient sensor nodes, although highly desired, must not violate such timing requirements.

- 3) IoT applications come with a variety of energy and speed requirements (Hempstead et al., 2008). It can be challenging to manually configure parameters for DVFS and PR to achieve good energy efficiency and speed for each IoT application.

To solve these “one-size-never-fits-all” challenge, a new software tool, called the Energy Reduction Program Analyzer (ERPA), is introduced herein. This tool enables automation of energy management in FPGA-based IoT sensor nodes. By analyzing the behavior of a program, ERPA automatically determines the best configurations for both DVFS (the frequency-voltage pairing) and PR (the microarchitecture).

The contributions of this work are summarized below:

1) Low-power techniques (DVFS, PR, and CG) were developed and applied to the RISC32 FPGA-based IoT sensor node to achieve lowest energy consumption. The energy reductions achieved through these low-power techniques were compared and analyzed in detail.

2) A new software tool, Energy Reduction Program Analyzer, was developed to automatically determine the best configuration for various low-power techniques. The instructions related to the low-power configurations are inserted into programs automatically at the appropriate locations to achieve optimal energy reductions.

1.2 Problem Statement

In Kiat et al. (2020), RISC32 is not implemented with any known low-power techniques. Hence, the energy reduction is not maximized. Moreover, most of the existing works are focused on using a single low-power technique. To maximize the energy reduction in RISC32, multiple low-power techniques have been implemented. However, after combining the low-power techniques, RISC32 has several configurations with varied energy consumption and performance levels. Hence, there is a need to ascertain the best configuration for the implemented low-power techniques to achieve maximum energy reduction.

1.3 Objectives

The main goal of this research is to reduce energy consumption of the RISC32 for IoT applications. Several low-power techniques have been implemented on the RISC32, which allowing the RISC32 to trade-off between power and performance during program execution. Also, a software tool,

namely, Energy Reduction Program Analyzer has been developed to determine (based on the program behavior) the best configuration while using the low-power techniques. The prime objectives of this research are enumerated as follows:

1. To implement low-power techniques (DVFS, PR, and CG) on the RISC32 FPGA-based IoT sensor node to reduce its energy consumption.
2. To develop a new software tool, Energy Reduction Program Analyzer, for automatic configuration of the low-power techniques.

CHAPTER 2

LITERATURE REVIEW

This chapter discusses several known low-power techniques that are applicable to FPGA power management are presented. These techniques include CG, power gating, and DVFS. Besides that, it discusses a few new trends of low-power techniques such as optimizing frequently used features in a system and reuse hardware resources to implement multiple low-utilization circuits by using the PR feature.

2.1 Low-Power Techniques

2.1.1 Clock Gating

CG is one of the most commonly employed low-power techniques. It gates the clock signal of an idle circuit with an AND gate. This prevents unnecessary clock switching, which consumes dynamic energy. However, the AND gate can potentially cause glitchy output. Hence, Sterpone et al. (2011) proposed a reconfigurable CG technique that replaces the AND gate used in conventional CG. A controller is required to disable/enable the clock signals to localize regions. This is achieved through command transmitted via Internal Configuration Access Port (ICAP).

2.1.2 Power Gating

Power gating is another technique useful in reducing the energy consumption in an FPGA. It gates the power supply to reduce idle power. Power gating can save more power compared to CG but requires more circuitry to support this feature. For example, there is a need to save and restore the registers' content in the power-gated circuit during power-down

and power-up phases. Bsoul et al. (2015) proposed a dynamically controlled power gating technique applicable to an FPGA. This technique turns on/off the power switch at run-time, and the authors reported a power saving of 83%. The control signal of the power switch is connected to the general-purpose routing fabric of the FPGA, which allows the FPGA itself to turn on/off the power switch. However, this technique requires customization on the circuitry inside the FPGA chip, which is not available in most of the commercial FPGA chips. Another similar work was presented by Hosse et al. (2014), who applied power gating technique to Zynq-7000, which consists of dual-core ARM Cortex-A9 processor (the processing system) and a Xilinx 7 series FPGA. The authors reported a power saving of 96%, where they used ARM Cortex-A9 to control the power rail of FPGA chip. This design requires an extra monitor system to control the power supply, which consumes extra power. Moreover, many IoT sensor nodes do not process complex computational tasks. A high-end processor like Cortex-A9 can be too power-hungry and unnecessary.

2.1.3 Dynamic Voltage and Frequency Scaling

DVFS is another popular low-power technique that adjusts power (voltage supply) and system performance (clock frequency) during application execution. Most of the prior work that implements DVFS focuses on how to select the best frequency–voltage ($f-v$) pair with the help of additional circuits. For instance, Nunez-Yanez et al. (2015, 2017) proposed an in-situ detector to be inserted into critical paths between two flip-flops (FFs) of their target design. The in-situ detector consists of a main flip-flop (MFF) and a slow flip-flop (SFF). Both FFs are driven by the output of the critical paths, but the input to the SFF is slightly delayed compared to MFF. By observing the output

of the MFF and SFF, the best operating frequency can be determined. In Nunez-Yanez et al. (2015), the critical paths between FFs and memory, which are called block RAMs, are included, while in Nunez-Yanez et al. (2017), just the path between FFs is included.

Wu et al. (2016) proposed a free razor technique to scale the supply voltage, to achieve energy efficiency, in addition to a forward error correction module to maintain the accuracy of the system. Based on the bit error rate (BER) sensor feedback, the voltage regulator can scale down the voltage supply as long as the BER is low enough to tolerate system noise. One drawback of this system is that it has an extra correction circuitry, which potentially degrades the processor performance if an error occurs. These techniques are not suitable to be implemented on RISC32 because adding such circuitry to RISC32 is too costly in terms of hardware resources, as RISC32 is aimed at implementing IoT sensor nodes.

Nunez-Yanez et al. (2015) combined the previously developed power gating (Hosseinabady and Nunez-Yanez, 2014) and DVFS techniques to lower the energy consumption in Zynq-7000 SoC ZC702 evaluation board. They also used the PR technique to switch between two different hardware configurations, namely, ME1 and ME6. The former represents the hardware configuration with a single execution unit, whereas the latter involves six execution units, where the execution unit refers to the MicroBlaze processor developed on the programmable logic. The switch between two hardware configurations is controlled by the ARM dual-core Cortex-A9 processor through the Processor Configuration Access Port. This work shows that an

energy reduction as high as 60% is achieved when the tasks are executed on the ME6 hardware configuration.

2.1.4 Partial Reconfiguration

Energy consumption can also be reduced by using PR, which allows the reuse of the same hardware area to implement several low-utilization circuits. For instance, Nunez-Yanez et al. (2017) presented a technique to lower the energy consumption of the Zynq-7000 SoC ZC702 evaluation board by using PR. This technique involves switching between two different hardware configurations with one and six execution units. The authors reported the energy reduction to be as high as 60%. Tamimi et al. (2018) proposed a reconfigurable architecture to implement a soft-core processor. They integrated functional units with low utilization (e.g., floating-point units) into look-up table (LUT)-based reconfigurable units (RUs). The low-utilization functional units only configure in RU when it is required to perform a specific operation. This technique successfully reduced hardware resources by 30.7%, reducing static power and energy consumption by 32.5% and 36.9%, respectively.

Kiat et al. (2020) proposed a feature in RISC32 that can dynamically switch between pipeline microarchitecture execution (PE) and multi-cycle microarchitecture execution (ME) to reduce energy consumption. PE can achieve higher throughput but requires more hardware resources (pipeline registers, forwarding circuits, a branch predictor, an interlock controller, etc.). On the other hand, ME has lower throughput, but it requires fewer hardware resources. Hence, to achieve better energy efficiency, PE is used to execute CPU-bound tasks to complete them within a short time. However, for IO-

bound tasks, the CPU spends most of its time idle, waiting for the IO instructions to be completed. Since ME consumes less power than PE, it is more advantageous to use ME in such a scenario.

2.1.5 Power Management Module

Existing power management modules are usually designed to support DVFS techniques. To achieve that, the power management module is required to determine the suitable location in the user program to apply DVFS as well as the right frequency to be configured, according to the program's behavior. In a method proposed by Tatematsu et al. (2011), the execution frequency is selected by using the greedy algorithm. On the other hand, Qin et al.'s (2019) method uses linear programming to determine the optimum frequency in every task. However, the software performance needs to be maintained while trying to reduce energy consumption. To address this issue, Wu et al. (2017) modelled the long-term deadline-aware task scheduling and Deng et al. (2020) proposed cuckoo search algorithm based on Gaussian random walk and adaptive discovery probability to reduce energy under specific performance constraints. In another similar work, Huang et al. (2018) developed a scheduling method, which allows task scheduling with fault tolerance. These prior works evaluated the energy reduction achieved by DVFS based on their simulation result.

In this work, a similar idea was used to develop a power management scheme including not only the DVFS configuration but also the PR configuration. The developed ERPA needs to identify the IO-bound instructions in the program to insert power management instructions effectively for energy reduction purposes.

2.1.6 Other Techniques

Besides these popular low-power techniques, energy consumption can also be reduced by developing function-specific hardware modules. For instance, See et al. (2020) introduced RISC32-E, which integrates the RISC32 sensor node with an AES-128 coprocessor based on instruction in-order issue, partial out-of-order completion. RISC32-E is able to perform encryption in counter mode 200% faster compared to software encryption on RISC32. Owing to the reduced encryption time, the energy consumption for encryption tasks on RISC32-E was also reduced by 99% as compared to RISC32. However, this energy reduction technique is applicable to the encryption task only.

Recently, Brandalero et al. (2019) presented MuTARe, a single ISA heterogeneous chip multiprocessor (CMP) with an additional voltage rail that enables it to operate in the near-threshold-voltage regime. MuTARe improved the design of CMP, which is more suitable for complicated and unpredictable IoT workloads. Due to the complexity of the involved IoT tasks, they proposed a dynamic binary translation (DBT) hardware module to automatically transform the code for reconfigurable acceleration.

2.2 Research gap

To maximize the energy efficiency, it is logical to extend the RISC32 to RISC32-LP by incorporating low-power techniques DVFS and CG, which can be applied to FPGA platforms. With the implementation of DVFS and CG, RISC32-LP can trade-off its performance to reduce power consumption.

After integrating DVFS and CG on top of PR, RISC32-LP has multiple options to reduce energy. Hence, a tool is required to regulate these low-power

techniques. In this work, we decided to develop a software that can calculate the position to activate the desired low-power technique based on application's behavior and insert low-power instructions into the application accordingly. Alternately, we could develop a hardware to profile the energy consumption and adjust the low-power technique on-the-fly. However, this hardware would have consumed extra hardware, which is costly for a sensor node device.

CHAPTER 3

RISC-LP: THE PROPOSED LOW-POWER FPGA-BASED SENSOR

NODE

3.1 System Overview

The proposed low-power FPGA based sensor node (RISC32-LP) is an extension of the RISC32 (Kiat et al., 2020) that aims to further reduce the energy consumption for IoT application. Figure 3.1 shows the architecture of both (a) RISC32 and (b) RISC32-LP with the IoT program compilation process.

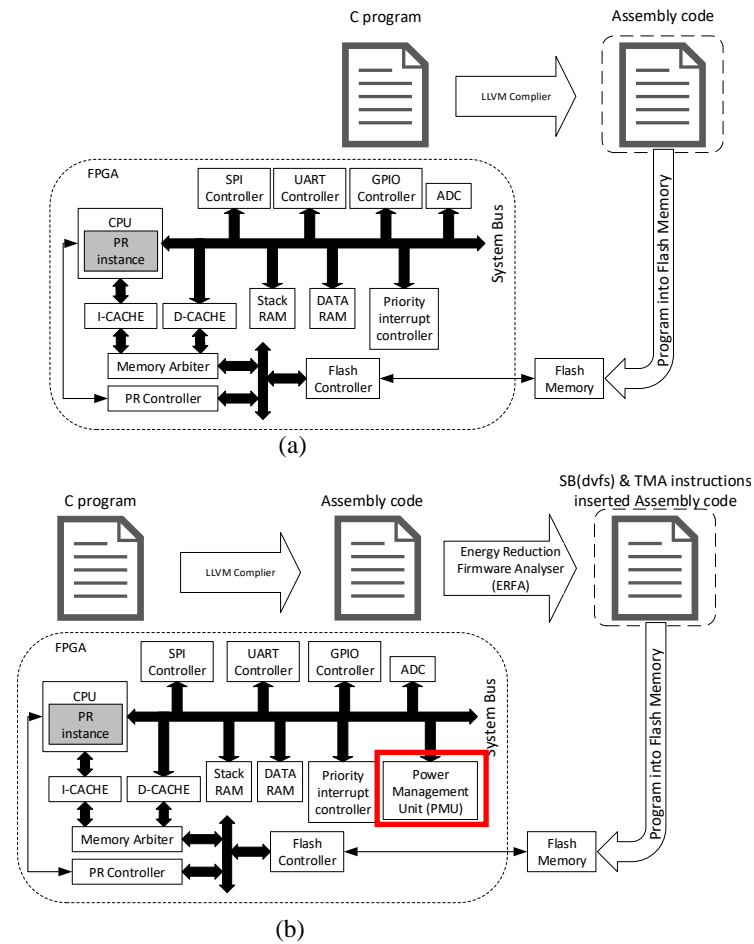


Figure 3.1: RISC32-LP architecture with IoT program compilation process.

The previous project, RISC32, is an MIPS-ISA-compatible 32-bit IoT processor. It supports only a small subset of the full MIPS instruction set (MIPS32, 2000) adequate for an IoT sensor node. The PR feature of its microarchitecture has been found to be able to reduce the dynamic energy consumption (details described in Section 3.3).

In this study, the RISC32 was extended by adding an extra power management unit (PMU) and a novel post-compilation energy optimization software – Energy Reduction Program Analyzer (ERPA). Previously, to run an IoT application in RISC32, the C program was compiled into MIPS assembly code via an LLVM compiler. Each instruction in the MIPS assembly code was then converted to their equivalent hexadecimal code and then configured into FPGA. RISC32-LP, on the other hand, uses the MIPS assembly code generated by the ERPA. The ERPA inserts the relevant low-power instructions into the user program at appropriate locations automatically. This has greatly reduced the effort in manually determining the low-power configurations to achieve energy reduction.

Table 3.1 shows the specification of the RISC32-LP, which employs a combination of low-power techniques, namely, CG, DVFS, and PR of the microarchitecture, to achieve a significant reduction in energy consumption. The RISC32-LP CPU runs at maximum 40 MHz, while the IO systems run at 10 MHz.

Table 3.1: Specification of the RISC32-LP.

		Multi-cycle	Pipeline
Frequency (MHz)		40 MHz to 20 MHz	40 MHz to 20 MHz
Cycle per instructions		3–5	1
Branch predictor		-	64 entries 4 ways associative
Hardware differences. Place in reconfigurable region (PR instance)		Data-path unit, Control unit finite state machine	Data-path unit, branch predictor, pipeline registers, hazard circuitry.
Power management unit		CG, DVFS, PR	
Common features (Static Region)	Memory system	4 kB boot ROM, 128 kB user access flash, 8 kB RAM (data and stack), 1 kB i-cache, 32 B d-cache, 512 B memory-mapped I/O register	
	Communication interface	ADC, UART, SPI, 32 GPIO pins. The IO systems run at 10 MHz.	
Partial address	Bitstream start	0x00A0_0000	0x00A8_0000
Bitstream size		3737 kB	
FPGA board		Nexys 4 DDR (XC7A100T)	
FPGA resources (Overall)	LUT	5631	6264
	LUTRAM	127	311
	FF	2782	3037
	BRAM	3.50	3.50
	IO	46	46
	BUFG	5	5

The remainder of this chapter discusses the PMU, while the details of ERPA are presented in Chapter 4.

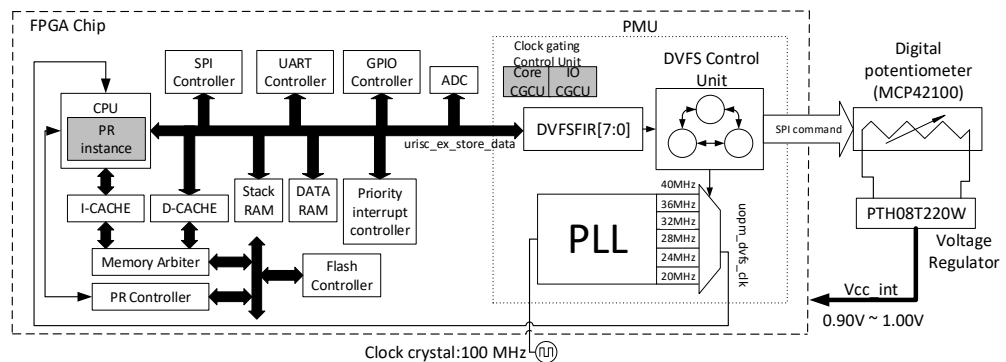


Figure 3.2: DVFS and CG components in the PMU.

3.2 Power Management Unit

The PMU is connected to the CPU through IO bus and consists of the CG and DVFS modules, as shown in Figure 3.2. The CG module enables/disables the CG function automatically based on the idle status of other modules in RISC32-LP. On the other hand, DVFS module provides the programmable parameter and f-v pairs that allow the RISC32-LP to execute IoT program at different performances. These two techniques work independently.

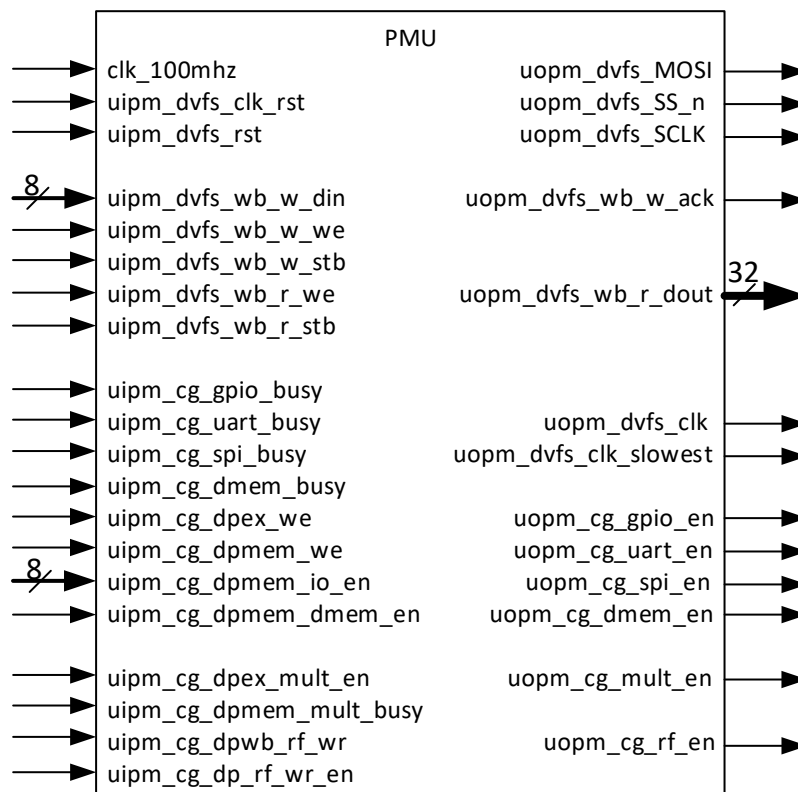


Figure 3.3: PMU block diagram in RISC32-LP.

Table 3.2: DVFS-related input and output pins in the PMU.

Pin name: clk_100mhz	Pin direction: input
Source -> Destination: FPGA board's clock crystal -> PMU	
Pin function: A 100 MHz clock signal from FPGA board.	
Pin name: uipm_dvfs_clk_rst	Pin direction: input
Source -> Destination: Global clock reset -> PMU	
Pin function:	
1: reset DVFS module in PMU and set the output clock (uopm_dvfs_clk) to default value (40 MHz).	
0: -	

<p>Pin name: uipm_dvfs_rst Pin direction: input</p> <p>Source -> Destination: Global rest -> PMU</p> <p>Pin function:</p> <p>1: reset other modules in PMU except the DVFS module.</p> <p>0: -</p>
<p>Pin name: uipm_dvfs_wb_w_din[7:0] Pin direction: input</p> <p>Source -> Destination: Data-path unit -> PMU</p> <p>Pin function: data input bus for DVFS control register (i.e., DVFS frequency index register, DVFSFIR).</p>
<p>Pin name: uipm_dvfs_wb_w_we Pin direction: input</p> <p>Source -> Destination: Address Decoder Block -> PMU</p> <p>Pin function: Wishbone standard write enable signal – indicate that the current bus cycle is for write or read access</p> <p>1: Write to DVFSFIR in PMU</p> <p>0: -</p>
<p>Pin name: uipm_dvfs_wb_w_stb Pin direction: input</p> <p>Source -> Destination: Address Decoder Block -> PMU</p> <p>Pin function: Wishbone standard strobe signal – indicate valid data transfer cycle</p> <p>1: activate SPI controller for write access</p> <p>0: deactivate SPI controller for write access</p>
<p>Pin name: uipm_dvfs_wb_r_we Pin direction: input</p> <p>Source -> Destination: Address Decoder Block -> PMU</p> <p>Pin function: Wishbone standard write enable signal – indicate that the current bus cycle is for WRITE or READ access.</p> <p>1: -</p> <p>0: Read from DVFSFIR in PMU</p>
<p>Pin name: uipm_dvfs_wb_r_stb Pin direction: input</p> <p>Source -> Destination: Address Decoder Block -> PMU</p> <p>Pin function: Wishbone standard strobe signal – indicate valid data transfer cycle.</p> <p>1: activate SPI controller for read access</p> <p>0: deactivate SPI controller for read access</p>
<p>Pin name: uopm_dvfs_MOSI Pin direction: output</p> <p>Source -> Destination: PMU -> External voltage regulator</p> <p>Pin function: Master data output pin (follow standard SPI). Send configuration data to external voltage regulator’s control register.</p>
<p>Pin name: uopm_dvfs_SS_n Pin direction: output</p> <p>Source -> Destination: PMU -> External voltage regulator</p> <p>Pin function: Slave select pin (follow standard SPI).</p> <p>0: Data sending to external voltage regulator through uopm_dvfs_MOSI is valid.</p> <p>1: -</p>
<p>Pin name: uopm_dvfs_SCLK Pin direction: output</p> <p>Source -> Destination: PMU -> External voltage regulator</p> <p>Pin function: SPI clock signal for data synchronization across devices.</p>
<p>Pin name: uopm_dvfs_wb_w_ack Pin direction: output</p> <p>Source -> Destination: PMU -> Data-path unit</p> <p>Pin function: Wishbone standard acknowledge signal – indicate the</p>

<p>termination of a normal write cycle. 1: normal bus cycle termination 0: no bus cycle termination</p>
<p>Pin name: uopm_dvfs_wb_r_dout[31:0] Pin direction: output Source -> Destination: PMU -> Data-path unit Pin function: Wishbone standard read data output bus.</p>
<p>Pin name: uopm_dvfs_clk Pin direction: output Source -> Destination: PMU -> all clock signal in RISC32-LP Pin function: Clock signal used to drive RISC32-LP.</p>
<p>Pin name: uopm_dvfs_clk_slowest Pin direction: output Source -> Destination: PMU -> all IO modules' slowest clock signal pins. Pin function: Always load slowest clock signal in RISC32-LP (20 MHz). Used by the IO module to generate their own baud-rate.</p>

Table 3.3: CG-related input and output pins in PMU.

<p>Pin name: uipm_cg_gpio_busy Pin direction: input Source -> Destination: GPIO unit -> PMU Pin function: Indicate that the GPIO unit is enabled and operating.</p>
<p>Pin name: uipm_cg_uart_busy Pin direction: input Source -> Destination: UART unit -> PMU Pin function: Indicate that the UART unit is enabled and operating.</p>
<p>Pin name: uipm_cg_spi_busy Pin direction: input Source -> Destination: SPI unit -> PMU Pin function: Indicate that the SPI unit is enabled and operating.</p>
<p>Pin name: uipm_cg_dmem_busy Pin direction: input Source -> Destination: data ram -> PMU Pin function: Indicate that the data RAM is enabled and operating.</p>
<p>Pin name: uipm_cg_dpex_we Pin direction: input Source -> Destination: Data-path unit -> PMU Pin function: Indicate that the current EX stage cycle is for read or write access. 1: WRITE 0: READ</p>
<p>Pin name: uipm_cg_dpmem_we Pin direction: input Source -> Destination: Data-path unit -> PMU Pin function: Indicate that the current MEM stage cycle is for read or write access. 1: WRITE 0: READ</p>
<p>Pin name: uipm_cg_dpmem_io_en[7:0] Pin direction: input Source -> Destination: Address Decoder Block -> PMU Pin function: Indicate which IO unit is selected to be accessed in current MEM stage cycle. [0]: General-purpose register – used by the PR controller. [1]: GPIO [2]: PIC (Programmable interrupt controller) [3]: SPI [4]: UART [5]: ADC</p>

[6]: - [7]: PMU
Pin name: uipm_cg_dpmem_dmem_en Pin direction: input Source -> Destination: Address Decoder Block -> PMU Pin function: 1: Indicate that the data RAM is selected to be accessed in current MEM stage cycle. 0: -
Pin name: uipm_cg_dpex_mult_en Pin direction: input Source -> Destination: Multiplier -> PMU Pin function: 1: Indicate that the MULT instruction has arrived at EX stage of RISC32-LP. Multiplier needs to operate in the next clock cycle. 0: -
Pin name: uipm_cg_dpmem_mult_busy Pin direction: input Source -> Destination: Multiplier -> PMU Pin function: 1: Indicate that the multiplier is operating. 0: -
Pin name: uipm_cg_dpwb_rf_wr Pin direction: input Source -> Destination: -> PMU Pin function: 1: Indicate that the register file writing enable signal is set at WB stage of RISC32-LP. Register file is needed to operate in next clock cycle. 0: -
Pin name: uipm_cg_dp_rf_wr_en Pin direction: input Source -> Destination: -> PMU Pin function: 1: Write enable signal for register file in RISC32-LP. 0: -
Pin name: uopm_cg_gpio_en Pin direction: output Source -> Destination: PMU -> GPIO's clock buffer Pin function: 1: Allow the GPIO clock signal switch at the same frequency as the IO clock. 0: Gate the GPIO clock.
Pin name: uopm_cg_uart_en Pin direction: output Source -> Destination: PMU -> UART's clock buffer Pin function: 1: Allow the UART clock signal switch at the same frequency as the IO clock. 0: Gate the UART clock.
Pin name: uopm_cg_spi_en Pin direction: output Source -> Destination: PMU -> SPI's clock buffer Pin function: 1: Allow the SPI clock signal switch at the same frequency as the IO clock. 0: Gate the SPI clock.
Pin name: uopm_cg_dmem_en Pin direction: output Source -> Destination: PMU -> Data RAM and stack RAM's clock buffer Pin function:

1: Allow data and stack RAM clock signal switch at the same frequency as the IO clock. 0: Gate data and stack RAM clock.	
Pin name: uopm_cg_mult_en Source -> Destination: PMU -> Multiplier's clock buffer Pin function: 1: Allow the multiplier clock signal switch at the same frequency as the CPU clock. 0: Gate the multiplier clock.	Pin direction: output
Pin name: uopm_cg_rf_en Source -> Destination: PMU -> Register file's clock buffer Pin function: 1: Allow the register file clock signal switch at the same frequency as the CPU clock. 0: Gate the register file clock.	Pin direction: output

3.2.1 Development of Clock Gating

CG reduces the dynamic power consumption by disabling some of the circuits in RISC32-LP that are not in use. In RISC32-LP, CG is applied to the IO modules, stack RAM, data RAM, multiplier, and register files, which are likely to remain idle during IoT program execution. These IO modules and functional units are partitioned as units or blocks in RISC32-LP, which have made CG easier to implement and maintain.

A module is considered to be in idle state whenever it is not being accessed or enabled. For example, the IO module UART is identified as idle when its enable flag is not set. A functional module, such as the multiplier, is identified as IDLE when there is no multiplication instruction decoded in RISC32-LP. When a particular functional or IO module is identified as idle, the CG circuitry disconnects the clock tree of that module. No clock signal would drive the module, and unnecessary switching activities are avoided. Table 3.4 shows the IO CG control signals, and Table 3.5 shows the CPU core CG control signals generated by the corresponding control unit.

Table 3.4: IO CG control unit's LUT.

Outputs				Inputs							
uopm_cg_gpio_en	uopm_cg_uart_en	uopm_cg_spi_en	uopm_cg_dmem_en	uipm_cg_gpio_busy	uipm_cg_uart_busy	uipm_cg_spi_busy	uipm_cg_dmem_busy	uipm_cg_dpex_we	uipm_cg_dpmem_we	uipm_cg_dpmem_io_en[7:0]	uipm_cg_dpmem_dmem_en
1	1	1	1	x	x	X	X	1	x	x	x
1	0	0	0	0	0	0	0	0	1	[1]	0
0	1	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	[4]	0
0	0	0	1	0	1	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	[3]	0
0	0	0	1	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1

Table 3.5: CPU core CG control unit's LUT.

Outputs		Inputs			
uopm_cg_mult_en	uopm_cg_rf_en	uipm_cg_dpex_mult_en	uipm_cg_dpmem_mult_busy	uipm_cg_dpwb_rf_wr	uipm_cg_dp_rf_wr_en
1	0	1	x	x	x
1	0	x	1	x	x
0	1	x	x	1	x
0	1	x	x	x	1

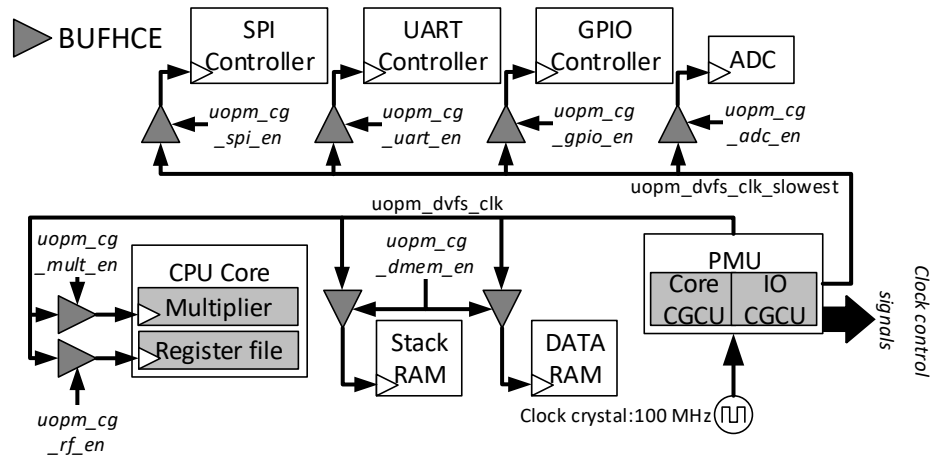


Figure 3.4: Clock gated module in RISC32-LP.

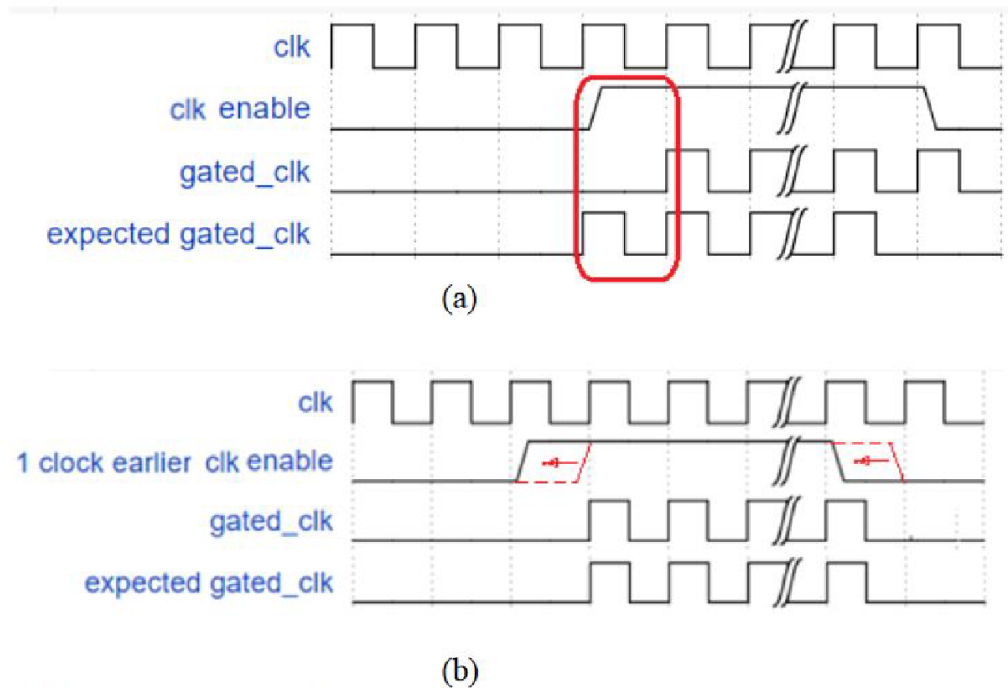


Figure 3.5: BUFHCE's waveform.

As shown in Figure 3.4, the clock tree of each module is driven by a clock buffer (BUFHCE), which is used to connect or disconnect the clock tree. Note that the clock control signals are named as `uopm_cg_X_en`, where X is the name of the clock-gated module. For instance, `uopm_cg_uart_en` refers to the clock control signal for the UART module. All clock control signals are generated by the IO CG control unit (IO_CGCU) and the core CG control unit

(Core_CGCU) in the PMU. In this work, the BUFHCE is used to implement CG instead of an AND gate because the former can ensure a glitch-less output.

Both RISC32 microarchitectures (PE and ME) have the following five stages: instruction fetch (IF), instruction decode (ID), execution (EX), memory (MEM), and write-back (WB) stage. Note that even though BUFHCE is a type of clock driver available on Xilinx 7-series FPGA, it will cause an output signal (which is the clock signal driving the selected modules) delayed by one clock cycle as illustrated in Figure 3.5(a). To avoid this problem, the clock control signal of each BUFHCE is generated one clock cycle before the module is being accessed (see Figure 3.5(b)). For example, IO and memory modules are only being accessed during the MEM stage. Hence, the required clock control signals are asserted at an earlier stage i.e., the EX stage. For register file and multiplier, the clock control signal is asserted in the MEM and ID stage, respectively.

According $P_{\text{dym}} = \sum (C * V_{\text{dd}}^2 * f_{\text{clock}})$, the dynamic energy reduces when the parameter f_{clock} decreases.

3.2.2 Development of Dynamic Voltage and Frequency Scaling

In RISC32-LP, the DVFS is applied to the CPU and memory system (except the FPGA external flash memory) to dynamically scale its operating frequency and voltage automatically. A DVFS module that supports six f-v pairs was developed, based on the maximum number of frequencies supported by a phase-locked loop (PLL) in Artix 7. The desired range of the operating frequencies can be chosen based on application. In our case, the range of 40 MHz to 20 MHz was chosen. This range is equally divided into six frequencies (40 MHz, 36 MHz, 32 MHz, 28 MHz, 24 MHz, and 20 MHz)

since the PLL in Artix-7 could only generate six frequencies. Hence, only six f-v pairs were developed as listed in Table 3.6

Table 3.6: F-V pairs.

DVFSFIR[7:0]	0	1	2	3	4	5
Frequency (MHz)	40	36	32	28	24	20
Voltage (V)	1.00	0.98	0.96	0.94	0.92	0.90

The f-v pairs are represented as a single-digit index (0 to 5) stored in a small memory. To select the desired f-v pair, the store byte (SB) instruction is used by the program to write the corresponding index value into the DVFS frequency index register (DVFSFIR). The DVFSFIR is IO memory mapped to the address 0xbfff ff3f. For example, in “sb \$t1, 0x3f(\$s2)”, \$t1 holds the index value 0x02, which corresponds to 32 MHz and 0.96 V, while \$s2 is used to hold the IO memory map base address (0xbfff ff00) to all the IO registers, including DVFSFIR. Hence, the sum of \$s2’s content and offset (0x3f) yields the address of DVFSFIR register, which the index (0x02) in \$t1 will be transferred to. Figure 3.2 shows the main components of DVFS module residing in the PMU.

To determine the voltage for each frequency, the following steps were taken. However, to achieve lowest energy consumption for a given task, the most suitable f-v pair is first determined by the ERPA based on the task’s behavior. Then, the following steps will run:

- 1) PLL generates the six clock frequencies listed in Table 3.6. One of the clock frequencies will be used as the operating frequency for the RISC32-LP based on DVFSFIR.

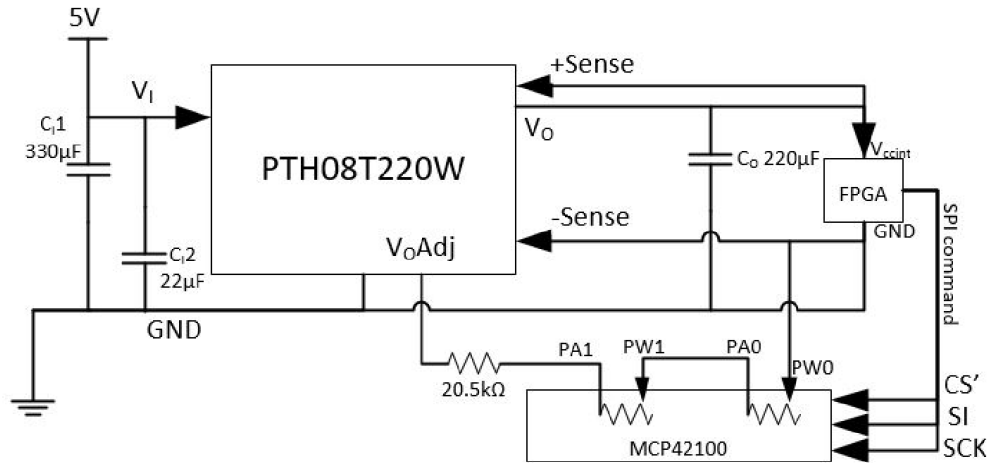


Figure 3.6: Voltage regulator's schematic.

2) A voltage regulator is designed to generate the corresponding supply voltage at the selected operating frequency. In this work, the voltage regulator provides the voltage supply in the range of 0.90–1.00 V, based on the reliable safe voltage range for the target FGPA (Datasheet, Xilinx. 2015). Figure 3.6 shows the voltage regulator, PTH08T220W. It can produce variable output voltage by adjusting the digital potentiometer (MCP42100). Based on the index value in DVFSFIR, the DVFS control unit (DVFS-CU) sends a 16-bit SPI command to the digital potentiometer to adjust the resistance, which, in turn produces the desirable supply voltage value.

3) A logic delay measurement circuit (LDMC) is developed to determine the optimal voltage supply for each operating frequency. It is used to ensure that in each $f-v$ pair, the voltage supply will provide enough drive strength for the corresponding clock frequency in RISC32-LP. If a voltage supply does not provide enough drive strength at the designated clock frequency, timing error would occur in RISC32-LP, whereby the data will not be timely captured by the registers. On the other hand, excessive voltage supply will cause RISC32-LP to consume extra energy, which is not the aim of our project.

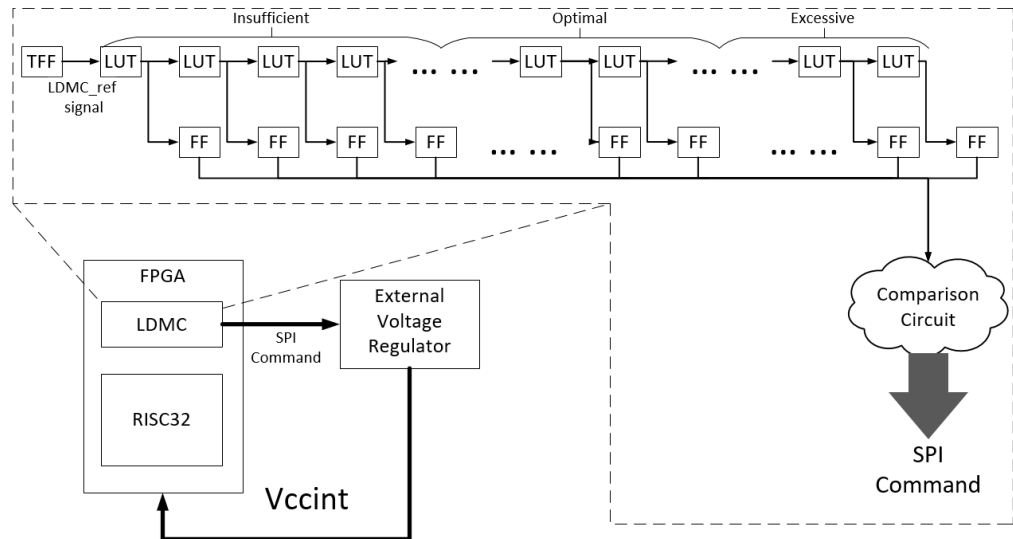


Figure 3.7: LDMC circuitry.

The LDMC is implemented in FPGA together with RISC32-LP (referred to as RISC32-LP+LDMC in the following paragraph). By default, RISC32-LP is operating at 40 MHz with a 1.0 V voltage supply. To find out the optimal voltage supply values for each of the clock frequencies (36 MHz, 32 MHz, 28 MHz, 24 MHz, and 20 MHz), the operating frequency of RISC32 is lowered down accordingly, and the LDMC responds to the frequency change by generating an SPI command to adjust the voltage supply accordingly. The voltage supply values measured at each of the above-mentioned frequencies are recorded in Table 3.6. The details of LDMC operation are described in the next paragraph.

The details of LDMC operation are described here. The LDMC is made up of a serial chain of LUT and FFs. The number of LUTs and FFs used depend on the critical path delay of RISC32-LP. Conceptually, the chain is grouped into three ranges of logic delays, which can be used to indicate the signal strength of the critical path: insufficient, optimal, and excessive. The LUTs are used to create the logic level delays, and the FFs are used to store the outputs of the LUTs to indicate how far in terms of delay the LDMC_ref

signal has propagated. We can think of the LDMC_ref signal as a ruler to indicate whether the critical path signal is in the insufficient, optimal, and excessive delay range. The number of FFs used in each range is specified in Table 3.7.

Table 3.7: The number of FFs used in each range of LDMC.

Logic level delay range	Number of FFs in series
Insufficient	82
Optimal	8
Excessive	8

For example, to get the optimal voltage at 36 MHz, the frequency of RISC32-LP+LDMC was tuned to 36 MHz manually. Consequently, the clock period was stretched longer, allowing more time for the LDMC_ref to propagate from the Toggle flip-flop (TFF) and through the LUTs, reaching the excessive range within a clock period. This means that 1.0 V is too excessive to be used as the supply voltage for the 36 MHz operating frequency. Hence, the comparison circuit outputted an SPI command, which was used by the potentiometer and external voltage regulator to lower the supply voltage by one level. This was repeated until the LDMC_ref reached the optimal range. The voltage was then recorded to form the f-v pair at 36 MHz as shown in Table 3.6. The above process was repeated to obtain all the f-v pair values and their corresponding SPI command values as shown in Table 3.8

Table 3.8: SPI command values.

DVFSFIR[7:0]	0	1	2	3	4	5
Frequency (MHz)	40	36	32	28	24	20
Voltage (V)	1.00	0.98	0.96	0.94	0.92	0.90
SPI command	0x12F9	0x1277	0x1242	0x1229	0x1218	0x120D

The SPI command values have been hardcoded in Verilog in the PMU. After we obtaining the above values, the LDMC can be removed so that the PMU can have a shorter response time for every frequency change.

By default, the f-v pair has been set to 40MHz-1V, which is the highest performance available experimented in RISC32-LP. When the SB instruction is executed, it triggers the signal `urisc_ex_store_data` (refer to Figure 3.2) to activate the DVFS operation to start the frequency adjustment. The f-v pair's index supplied by SB is first stored in DVFSFIR, and then compared with the previous DVFSFIR value to decide whether to increase or decrease the frequency. If the targeted frequency is lower than the previous one, DVFS-CU first selects the requested operating frequency via the multiplexer, then issues the 16-bit SPI command to the digital potentiometer to reduce the voltage supply step-by-step based on voltage value in Table 3.8. If the voltage supply is reduced before the frequency, the voltage supply can cause timing error. Hence, the frequency has to be reduced first before configuring the voltage supply. On the other hand, if the targeted frequency is higher than the previous one, voltage supply is increased first before the DVFS-CU selects the higher operating frequency. This is done to ensure that the drive strength of the voltage supply is enough to support the higher operating frequency.

To control the switching of f-v pairs, a DVFS-CU was developed. Figure 3.8 shows the block diagram of DVFS-CU, and Table 3.9 describes the function of each pin.

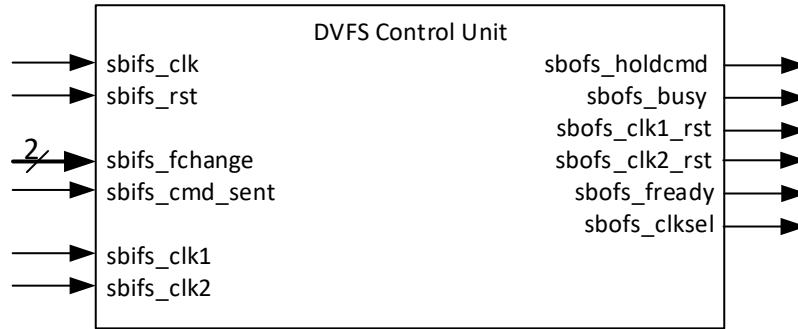


Figure 3.8: Block diagram of DVFS-CU.

Table 3.9: DVFS-CU input and output pins.

Pin name: sbifs_clk	Pin direction: input
Source -> Destination: PMU -> DVFS-CU	
Pin function: 100 MHz clock signal from FPGA board. This clock signal drives the DVFS-CU itself only. It is not used to supply to other part of RISC32-LP.	
Pin name: sbifs_rst	Pin direction: input
Source -> Destination: PMU -> DVFS-CU	
Pin function:	
1: reset DVFS-CU.	
0: -	
Pin name: sbifs_fchange[1:0]	Pin direction: input
Source -> Destination: PMU -> DVFS-CU	
Pin function:	
Indicate that the new request RISC32-LP core clock frequency is lower, higher, or equal compared to the current RISC32-LP core clock frequency.	
00: Equal.	
01: -	
10: New < Current	
11: New > Current	
Pin name: sbifs_cmd_sent	Pin direction: input
Source -> Destination: PMU -> DVFS-CU	
Pin function:	
1: Indicate that the previous SPI command is sent to the digital potentiometer (used to configure the voltage supply level of external voltage regulator).	
0: -	
Pin name: sbifs_clk1	Pin direction: input
Source -> Destination: PLL (in PMU) -> DVFS-CU	
Pin function: The clock frequency is used to drive the RISC32-LP core when the DVFS-CU is in "RESET_CLK2," which is the DVFS-CU's idle state.	
Pin name: sbifs_clk2	Pin direction: input
Source -> Destination: PLL (in PMU) -> DVFS-CU	
Pin function: The clock frequency that is used to drive the RISC32-LP core when PMU is configuring new request clock frequency. This is to prevent the whole RISC32-LP stall when the PMU is modifying the clock frequency.	

<p>Pin name: sbofs_holdcmd Pin direction: output Source -> Destination: DVFS-CU -> SPI transmitter in the PMU Pin function: 1: Hold / Do not allow the value the SPI command change while command transmission is in progress. 0: -</p>
<p>Pin name: sbofs_busy Pin direction: output Source -> Destination: DVFS-CU -> PMU Pin function: 1: The DVFS-CU is in progress of change f-v pair. 0: The DVFS-CU is in "RESET_CLK2" state / idle state.</p>
<p>Pin name: sbofs_clk1_rst Pin direction: output Source -> Destination: DVFS-CU -> PMU Pin function: 1: Reset clk1 (no switching). 0: -</p>
<p>Pin name: sbofs_clk2_rst Pin direction: output Source -> Destination: DVFS-CU -> PMU Pin function: 1: Reset clk2 (no switching). 0: -</p>
<p>Pin name: sbofs_fready Pin direction: output Source -> Destination: DVFS-CU -> PMU Pin function: 1: clk2 is set to new requested frequency. 0: -</p>
<p>Pin name: sbofs_clkssel Pin direction: output Source -> Destination: DVFS Control Unit -> PMU Pin function: 1: select clk2 as clock signal that drive RISC32-LP core. 0: select clk1 as clock signal that drive RISC32-LP core.</p>
<p>Pin name: bufgmux_cnt_rst Pin direction: internal signal Source -> Destination: N/A Pin function: Reset bufgmux_cnt register.</p>
<p>Register name: bufgmux_cnt Register function: Register in DVFS-CU. A counter used to count until nine clock cycles (100 MHz). Nine clock cycles is the latency required for bufgmux switch from clk1 to clk2 or vice versa.</p>
<p>Pin name: clkcounter_cnt_rst Pin direction: internal signal Source -> Destination: N/A Pin function: Reset clkcounter_cnt register.</p>
<p>Register name: clkcounter_cnt Register function: Register in DVFS-CU. A counter used to count until four clock cycles (100 MHz). Four clock cycles is the latency needed for clk1 and clk2 to switch to other new</p>

frequency safely.

There are a total of eight states in the DVFS-CU finite state machine (FSM) as shown in Figure 3.9. Each state is described in Table 3.10, and the output pin value for each state is listed in Table 3.11.

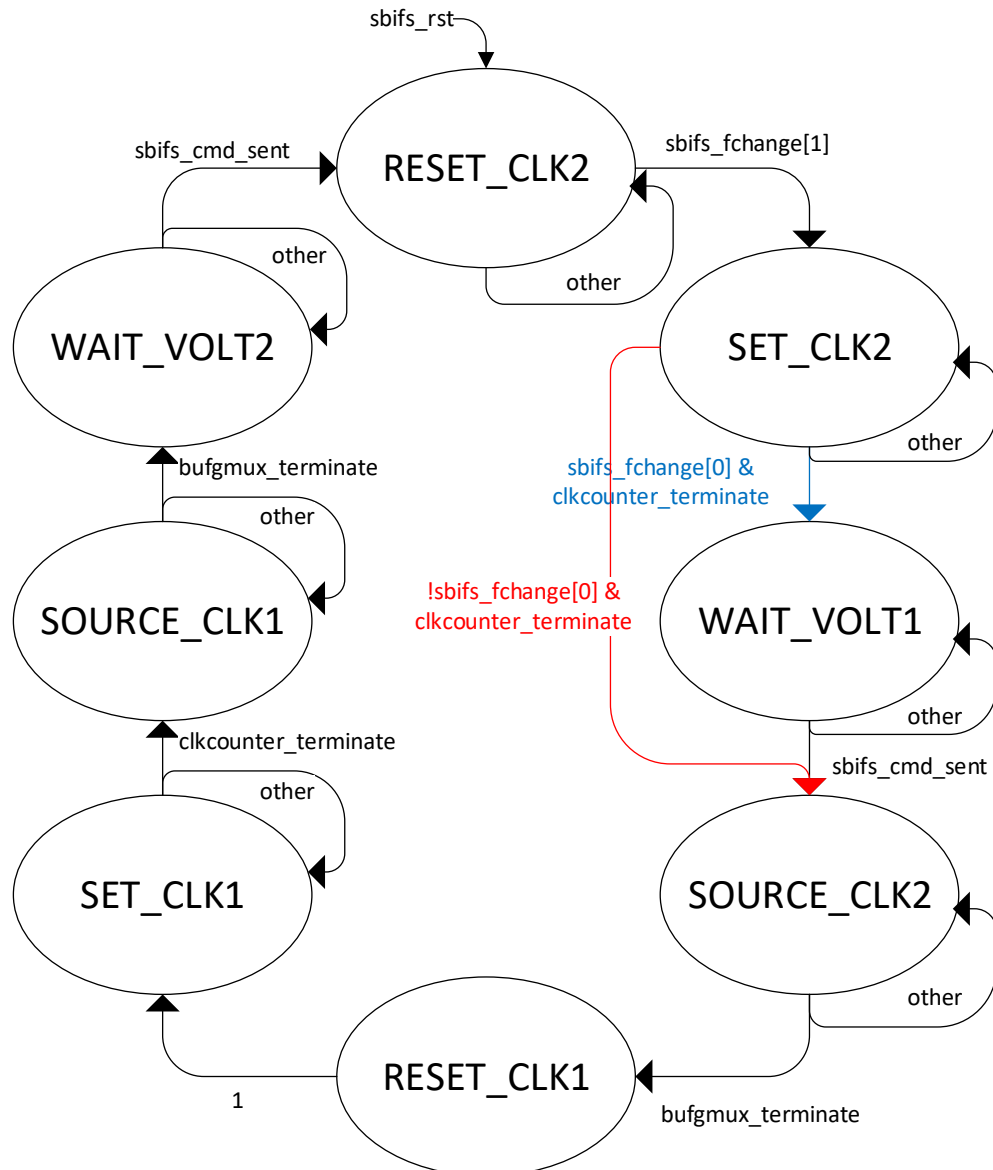


Figure 3.9: State diagram for DVFS-CU FSM.

The selected clock frequency of the PLL is duplicated into two clock sources: clk1 and clk2 as shown in Figure 3.10. The reason is to ensure that the CPU can continuously operate with a stable clock source: while one clock source is setting up (unstable), the other clock source can be used.

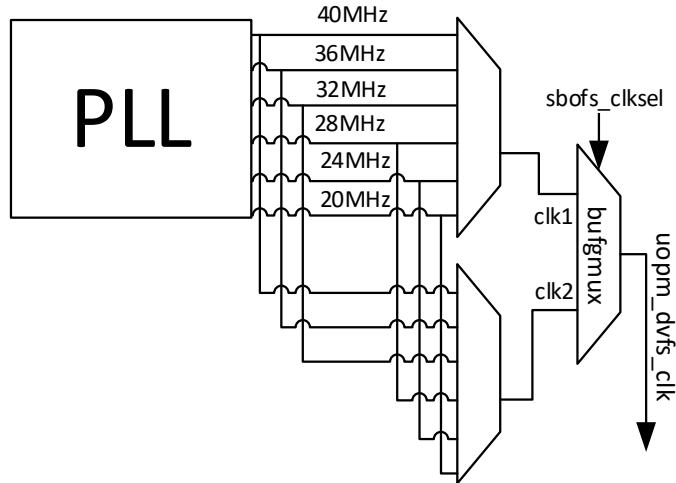


Figure 3.10: clk1 and clk2 signals from the PLL.

Table 3.10: Description of each FSM state for DVFS-CU.

Present state of DVFS-CU	Next state of DVFS-CU	Description
RESET_CLK2	sbifs_fchange[1]: SET_CLK2 other: RESET_CLK2	Idle state. If the new request frequency is different from the current frequency, go to SET_CLK2 state. Remain idle if no frequency change requested.
SET_CLK2	sbifs_fchange[0] & clkcounter_terminate: WAIT_VOLT1 !sbifs_fchange[0] & clkcounter_terminate: SOURCE_CLK2 other: SET_CLK2	Set up clk2 to the new requested frequency value. Need to wait for the clk2 finish setting up (clkcounter_terminate==1) before going to the next state. If the new frequency is higher than current frequency, go to WAIT_VOLT1. If the new frequency is lower than the current frequency, go to SOURCE_CLK2. Wait for acknowledge signal (clkcounter_terminate).
WAIT_VOLT1		Wait for new voltage supply level set up before switching to a higher frequency.

	sbifs_cmd_sent: SOURCE_CLK2	Go to SOURCE_CLK2 once ready.
	other: WAIT_VOLT1	Wait for acknowledge signal (sbifs_cmd_sent).
SOURCE_CLK2		Change the RISC32-LP core clock signal from clk1 to clk2. This process is handled by BUFGMUX_CTRL, which is a combination of clock buffer and multiplexer available in Xilinx FPGA chip.
	bufgmux_terminate: RESET_CLK1	Go to RESET_CLK1 after BUFGMUX_CTRL completed the switching between clk1 and clk2.
	other: SOURCE_CLK2	Wait for acknowledge signal (bufgmux_terminate).
RESET_CLK1		Reset clk1 before changing clk1 to new frequency.
	Always true: SET_CLK1	
SET_CLK1		Set up clk1 to the new requested frequency value. Need to wait for the clk1 finish setting up (clkcounter_terminate==1) before going to the next state.
	clkcounter_terminate: SOURCE_CLK1	Go to SOURCE_CLK1 once it is ready.
	other: SET_CLK1	Wait for acknowledge signal (clkcounter_terminate).
SOURCE_CLK1		Change the RISC32-LP core clock signal from clk2 back to clk1.
	bufgmux_terminate: WAIT_VOLT2	Go to WAIT_VOLT2 after BUFGMUX_CTRL completed the switching between clk1 and clk2.
	other:	Wait for acknowledge signal

	SOURCE_CLK1	(bufgmux_terminate).
WAIT_VOLT2	sbifs_cmd_sent: RESET_CLK2 other: WAIT_VOLT1	Wait for new voltage supply level set up. This is mainly for the case that the new frequency is lower than the current frequency. (Skipped WAIT_VOLT1) Go to RESET_CLK2 once done. Wait for acknowledge signal (sbifs_cmd_sent).

Table 3.11: Output pins value of each FSM state for DVFS-CU (DVFS-CU).

Present State of DVFS-CU	Output pins value
RESET_CLK2	sbofs_holdcmd = 1 sbofs_clk2_rst = 1 bufgmux_cnt_rst = 1 clkcounter_cnt_rst = 1
SET_CLK2	sbofs_holdcmd = 1 sbofs_busy = 1 bufgmux_cnt_rst = 1
WAIT_VOLT1	sbofs_busy = 1 bufgmux_cnt_rst = 1 clkcounter_cnt_rst = 1
SOURCE_CLK2	sbofs_holdcmd = 1 sbofs_busy = 1 clkcounter_cnt_rst = 1
RESET_CLK1	sbofs_holdcmd = 1 sbofs_busy = 1 sbofs_fready = 1 sbofs_clkssel = 1 bufgmux_cnt_rst = 1 clkcounter_cnt_rst = 1
SET_CLK1	sbofs_holdcmd = 1 sbofs_busy = 1 sbofs_clkssel = 1 bufgmux_cnt_rst = 1
SOURCE_CLK1	sbofs_holdcmd = 1 sbofs_busy = 1 clkcounter_cnt_rst = 1
WAIT_VOLT2	sbofs_busy = 1 bufgmux_cnt_rst = 1 clkcounter_cnt_rst = 1

Clock Domain Crossing in RISC32-LP

To implement DVFS, the processor RISC32-LP requires multi-clock domain since the IO system is running at 10 MHz. Figure 3.11 illustrates the clock domain of each area in RISC32-LP.

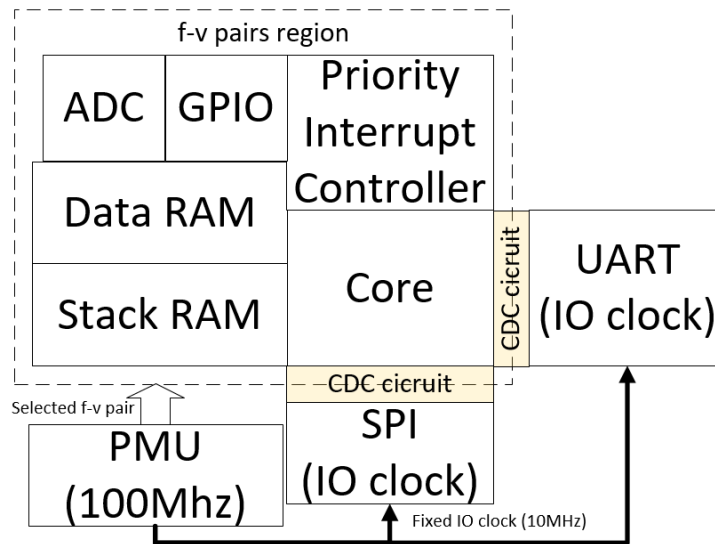


Figure 3.11: Multi-clock domain in RISC32-LP and CDC circuit location.

To resolve multi-clock domain metastability issue, a simple clock domain crossing (CDC) circuit is implemented in between the different clock domain areas as shown in Figure 3.11. Figure 3.12 shows the block diagram of the CDC circuit, and Table 3.12 describes the pins of the CDC circuit. The details of CDC circuit and example waveform are shown in Figure 3.13 and Figure 3.14, respectively.

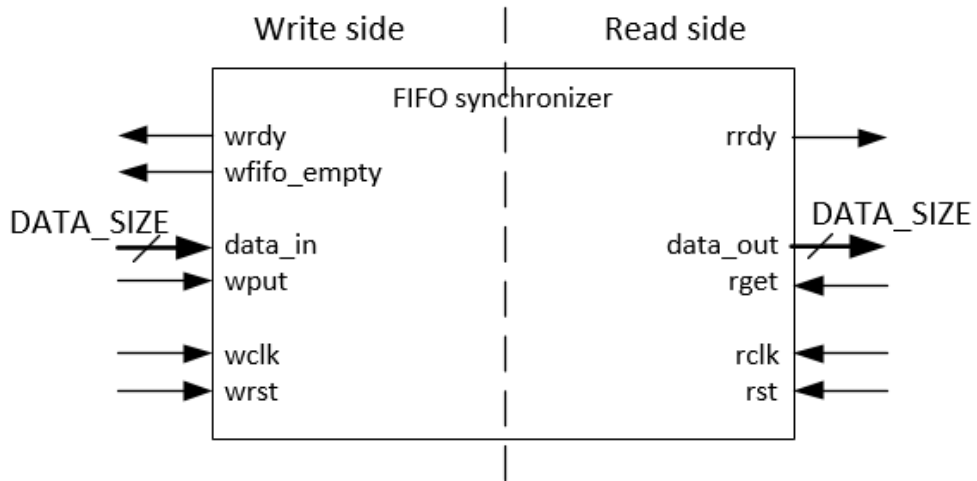


Figure 3.12: Block diagram of the CDC circuit.

Table 3.12: CDC circuit input and output pins

Write side	
Pin name: wrdy	Pin direction: output
Pin function: 1: CDC FIFO ready to receive data (writeable) 0: CDC FIFO is full (not writeable)	
Pin name: wfifo_empty	Pin direction: output
Pin function: 1: CDC FIFO empty 0: CDC FIFO not empty	
Pin name: data_in	Pin direction: input
Pin function: Data bus. Data to be write into FIFO.	
Pin name: wput	Pin direction: input
Pin function: Write enable pin. 1: Write data 0: -	
Pin name: wclk	Pin direction: input
Pin function: Clock signal. Need to use the same clock with the source that wrote the data into FIFO.	
Pin name: wrst	Pin direction: input
Pin function: Reset pin for write side.	
Read Side	
Pin name: rrdy	Pin direction: output
Pin function: 1: Indicate there is data in FIFO and ready to be read. 0: No data available to read	
Pin name: data_out	Pin direction: output
Pin function:	

Data bus. Data to be read from FIFO	
Pin name: rget	Pin direction: input
Pin function: Read enable pin. 1: Read data. 0: -	
Pin name: rclk	Pin direction: input
Pin function: Cock signal. Need to use the same clock with the destination that read the data from FIFO	

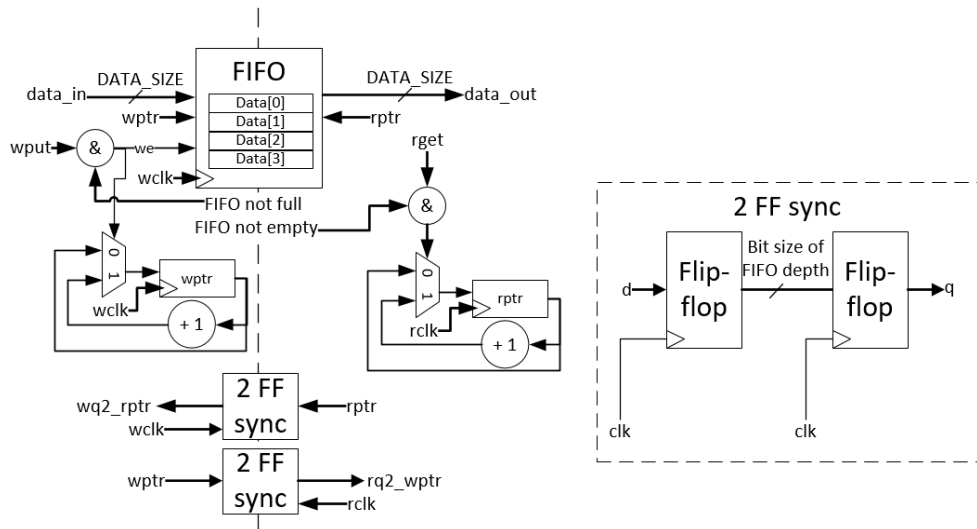


Figure 3.13: Details of the CDC circuit.

The CDC circuit is used to sync the control signals and data bus between the modules operating at different clock frequencies. For example, the core and the UART. The UART is driven by a 10 MHz clock (IO clock). However, the core of RISC32-LP could be driven by any frequency available in f-v pairs (20 MHz to 40 MHz).

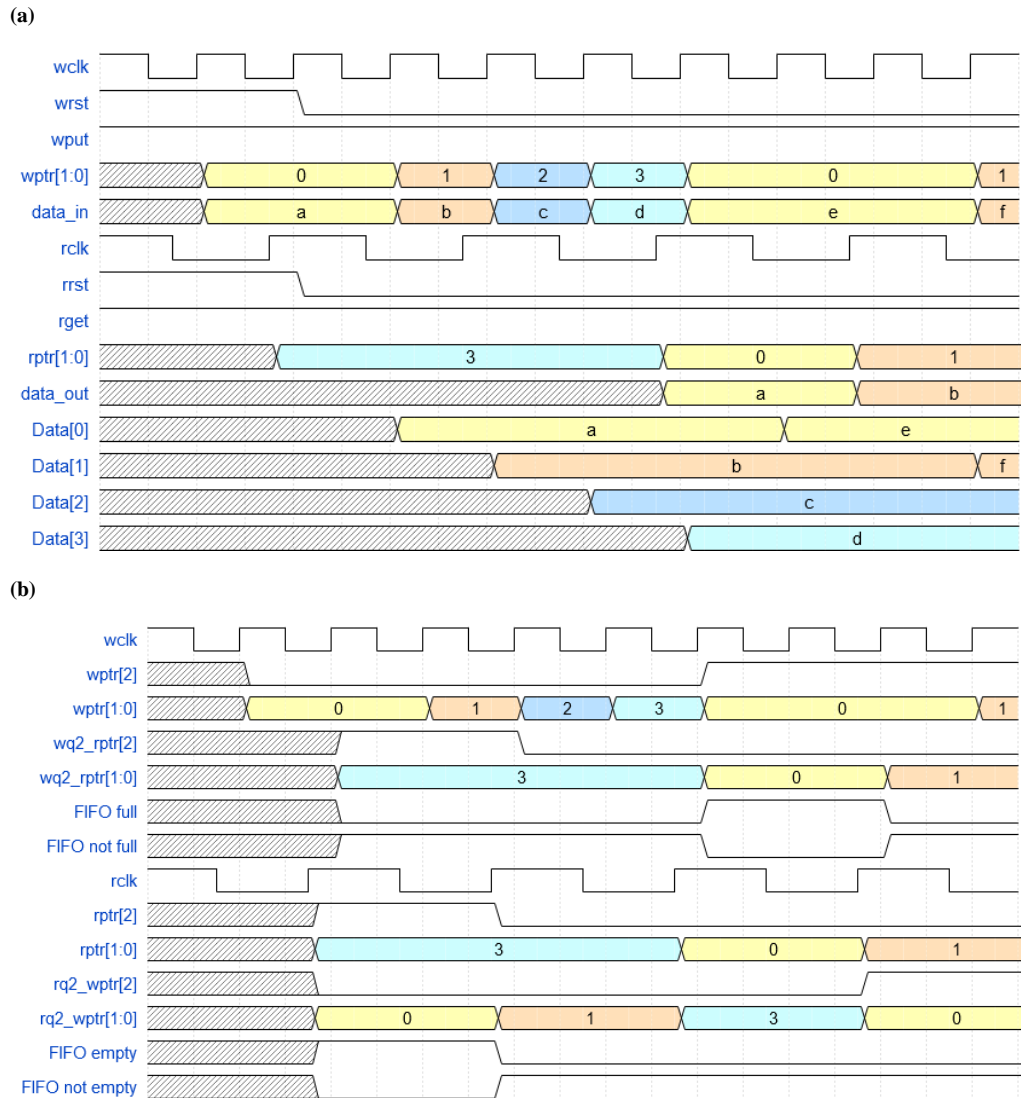


Figure 3.14: Example waveform of the CDC circuit.

Figure 3.14 illustrates the waveform behavior of the CDC circuit when the signals are passed from the core to the UART or SPI. The example in the figure shown are the core signals on the *write* side of the CDC block and the UART signals are on the *read* side of CDC block. Figure 3.14 (a) shows how the data is stored in the CDC FIFO. The waveform shows the case whereby the UART always reads the data from the core when the data is available. Figure 3.14 (b) shows how the *read* pointer and the *write* pointer affect the FIFO-not-full and FIFO-not-empty conditions. The former indicates that the FIFO is still able to receive new data, whereas the latter indicates that there is

new data that can be read from the FIFO. When the FIFO is full, the wrdy pin of the CDC circuit will be de-asserted and the core cannot write data until the pin is asserted again. On the other hand, the UART can read from the CDC circuit if the rrdy is de-asserted, which means that the FIFO is empty.

3.4 Partial Reconfiguration

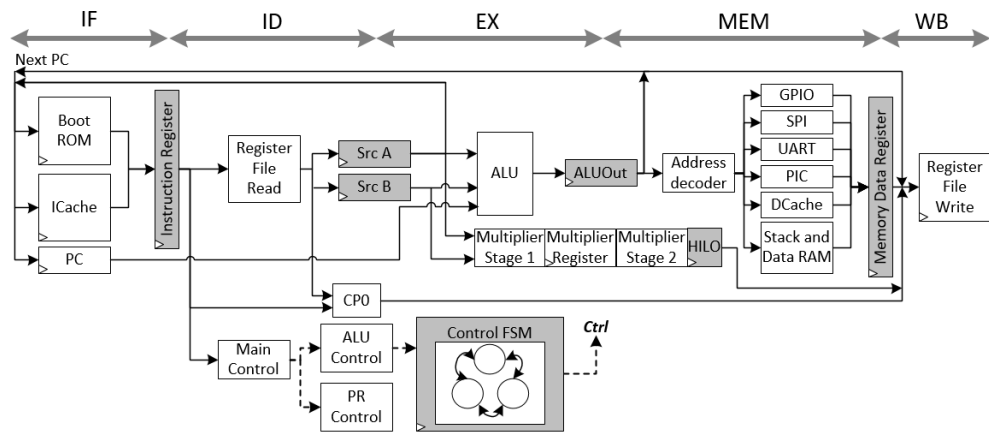


Figure 3.15: RISC32's microarchitecture in the ME mode.

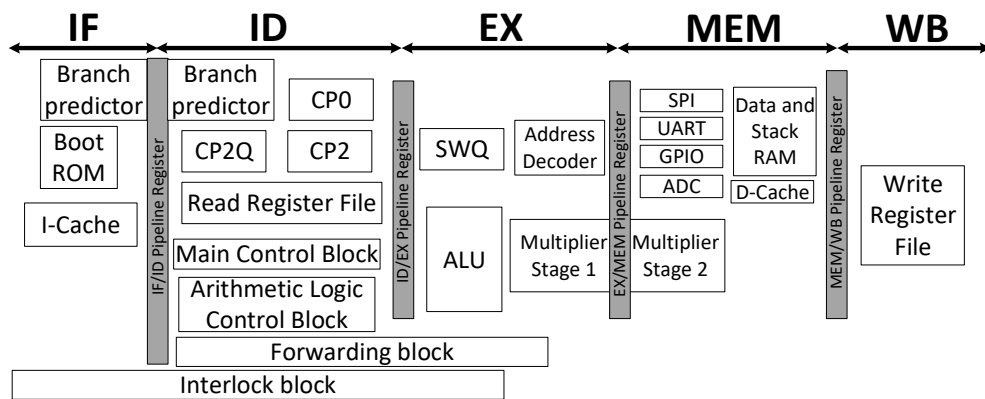


Figure 3.16: RISC32's microarchitecture in the PE mode.

Kiat et al. (2020) proposed a unique feature in RISC32 – a processor with a partial reconfigurable microarchitecture. It can dynamically switch between multi-cycle microarchitecture execution (ME) (Figure 3.15) and pipeline microarchitecture execution (PE) (Figure 3.16) to reduce the energy consumption. PE can achieve higher throughput at the expense of more

hardware resources (pipeline registers, forwarding circuits, branch predictor, interlock controller, etc.). On the other hand, ME has lower throughput, but it requires lesser hardware resources. In CPU-bound tasks, PE has a better energy efficiency because the designated tasks can be completed within a short time. However, for IO-bound tasks, the CPU is spending most of its time idle, waiting for the IO instructions to complete. Since ME consumes less power compared to PE, it is more advantageous to use ME in such a scenario. Note that for IO-bound task, both ME and PE take the same amount of time to complete, because the bottleneck lies in communication instead of computation.

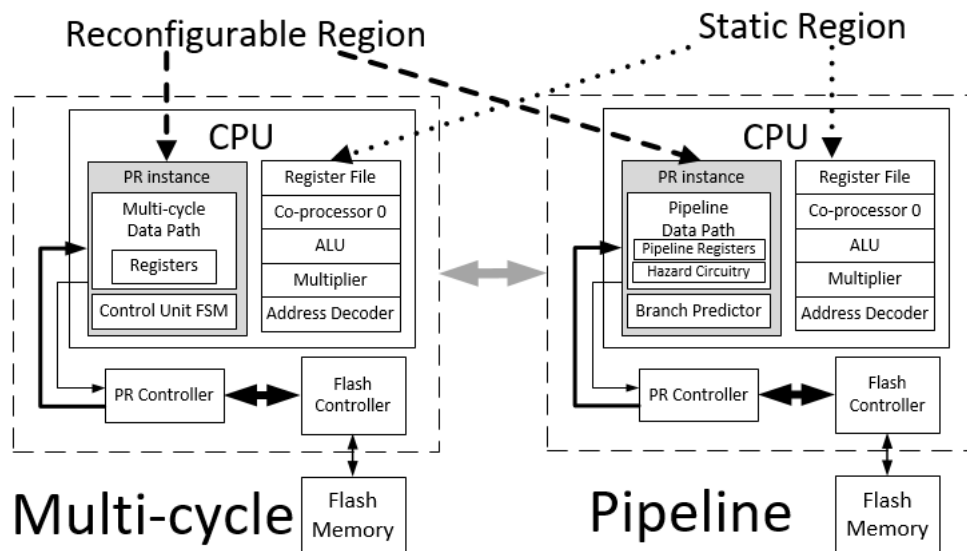


Figure 3.17: Static and reconfigurable region in RISC32's core (Kiat et al., 2020).

The PR feature on FPGA allows designers to reconfigure partial region of the FPGA during operation. In RISC32, PR only takes place within the CPU core, which is aimed to switch between PE and ME. Referring to Figure 3.17, the CPU core is divided into two regions: static and reconfigurable. The functional units that are required for both PE and ME are placed in the static

region, where no PR can take place. On the other hand, the reconfigurable region is reserved for the remaining hardware portion that is needed to form the PE or ME data path.

To facilitate the reconfiguration of microarchitecture from the program, a new instruction TMA was introduced in RISC32 (Kiat et al., 2020). Once the TMA instruction is detected, the processor execution stalls, and the PR controller starts the PR process on the reconfigurable region. First, the microarchitecture that is currently being used (i.e., PE or ME) is identified. Next, the PR controller retrieves the partial bit-stream of the targeted microarchitecture from the flash memory. For instance, if the current microarchitecture is ME, then the PE (target) will be loaded, and vice versa. The partial bit-stream is used to program the reconfigurable region through ICAP on the Artix-7 FPGA chip. After the configuration is completed by the PR controller, the processor continues to execute the instruction that was stalled previously, with the newly configured microarchitecture. The time taken for a PR operation to complete is based on the system frequency and the size of the partial bit-stream. As reported by Kiat et al. (2020), it takes 44 ms for a 20 MHz system frequency with 175,624 bytes of partial bit-stream.

3.5 Xilinx Vivado Synthesis Results

Table 3.13 shows that only a very small amount of extra FPGA resource is required in RISC32-LP as compared to RISC32 after the implementation of PMU in RISC32-LP.

Table 3.13: FPGA resource difference between RISC32 and RISC32-LP.

FPGA resources		RISC32		RISC32-LP		Extra resource used by RISC32-LP	
		Multi-cycle	Pipeline	Multi-cycle	Pipeline	Multi-cycle	Pipeline
LUT	63,400	5,181 (8.37%)	5,849 (9.23%)	5,631 (8.88%)	6,264 (9.88%)	450 (0.51%)	415 (0.65%)
LUTRAM	19,000	127 (0.67%)	311 (1.64%)	127 (0.67%)	311 (1.64%)	0 (0%)	0 (0%)
FF	12,680	2,258 (1.78%)	2,545 (2.01%)	2,782 (2.19%)	3,037 (2.4%)	524 (0.41%)	492 (0.39%)
BRAM	135	3.5 (2.59%)	3.5 (2.59%)	3.5 (2.59%)	3.5 (2.59%)	0 (0%)	0 (0%)
BUFG	32	2 (6.25%)	2 (6.25%)	5 (15.63%)	5 (15.63%)	3 (9.38%)	3 (9.38%)
BUFHCE	96	0 (0%)	0 (0%)	6 (6.25%)	6 (6.25%)	6 (6.25%)	6 (6.25%)
SLICE	15,850	1,746 (11.02%)	1,997 (12.60%)	1,876 (11.84%)	2,072 (13.07%)	130 (0.82%)	75 (0.47%)

CHAPTER 4

ENERGY REDUCTION PROGRAM ANALYZER

In the previous chapter, DVFS and CG techniques were developed and combined with the microarchitectural reconfiguration technique introduced by Kiat et al (2020) to further reduce the energy consumption. However, the best configuration (f-v pair and PE/ME) that achieves the maximum energy reduction largely depends on the user program. Since IoT technologies are applied in various fields with diverse characteristics, the user program pattern in IoT sensor nodes differs in many ways. For example, some IoT applications have specific time constraints to be fulfilled. While achieving low energy consumption in a sensor node is important, the time constraints should not be overlooked, because doing so may affect the response time of the entire IoT system.

To determine the best energy-saving configuration automatically, a software tool, named ERPA, was developed. In particular, the ERPA analyzes the given user program being executed on RISC32-LP and selects the best locations in the program to insert the instructions that execute the DVFS (i.e., SB) or the best microarchitecture reconfiguration (i.e., the TMA).

The user program for RISC32-LP is assumed to be developed in assembly code. If high-level language (e.g., C) is used to develop the programs (See, 2017), the code must be compiled into assembly code before it is analyzed by the ERPA. ERPA analysis flow is illustrated in Figure 4.1. It is grouped into seven steps, which will be explained in more detail later.

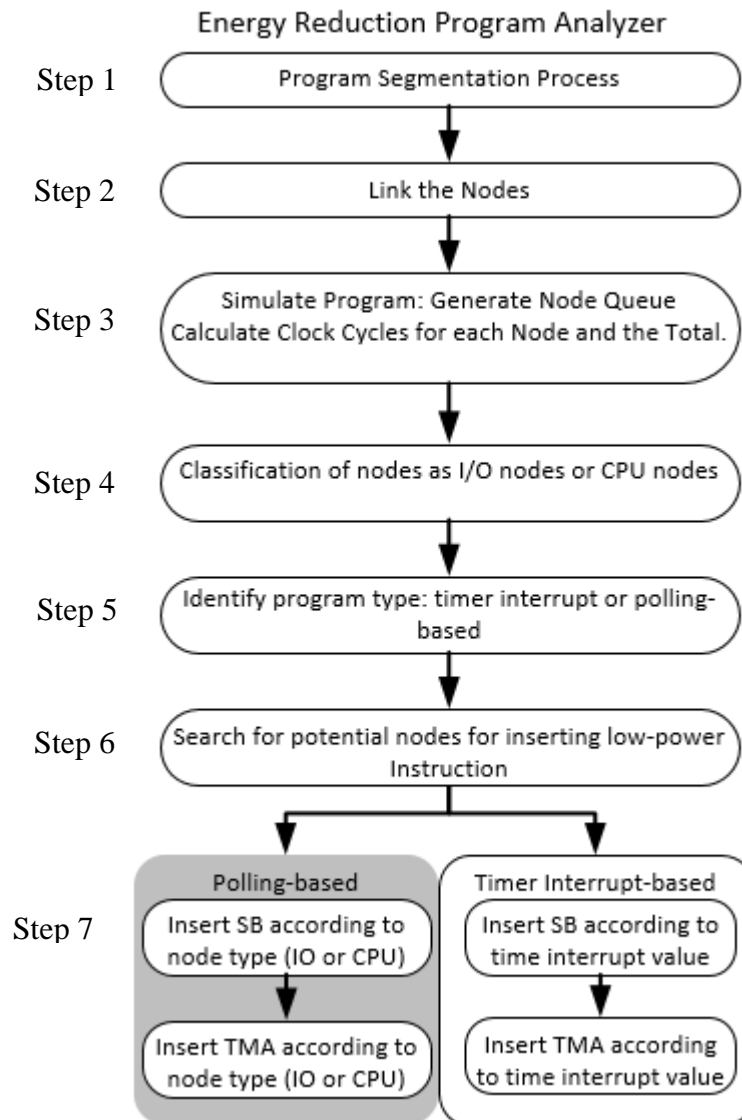


Figure 4.1 Flow of ERPA analysis.

To better illustrate how the ERPA works, an example user program shown in Table 4.1 is used for explanation. This program executes an infinite while loop (often referred to as a superloop), wherein ADC sampling and conversion can be implemented through a polling- or interrupt-based method. The corresponding assembly code is shown in Table 4.2. The steps to carry out program analysis by the ERPA tool are described in detail in the subsequent section.

Table 4.1 Pseudo code of the example user program.

```
while true do
    data = ADC.getData();
    temp = ConvertADC(data);
    idle (250000);
end
function ConvertADC(value) is
    return value*100/4096;
end
```

Table 4.2 Assembly code of the example user program.

```
//Start of while loop
loop:      lw    $s1, [adc]
           sw    $s1, [data]
           jal   conv
           sw    $v0, [temp]
           mtc0 $zero, $9
           li    $t2, 1250
poll:      mfc0  $t1, $9
           sub   $t3, $t2, $t1
           bgtz  $t3, poll
           j     superloop
//Start of ConvertADC() function
conv:      li    $t1, 100
           lw    $a1, [data]
           mult  $a1, $t1
           mflo  $t2
           srl   $v0, $t2, 12
           jr    $ra
```

Table 4.3 lists all the important variables used in this section.

Table 4.3 Description for symbols used in this chapter.

Variable	Definition
$T_{\text{TIMER_INT}}$	Timer interrupt value in clock cycle count (measured at the maximum frequency, 40 MHz)
$T_{\text{NODE_TOTAL}}$	Total clock cycle count (from all nodes) required to complete a superloop
T_{IDLE}	$(T_{\text{TIMER_INT}} - T_{\text{NODE_TOTAL}}) * 25 \mu\text{s}$. The available program slack for a timer interrupt-based program, converted from clock cycle count to microseconds (μs).
T_{EXE}	$T_{\text{NODE_TOTAL}} * 25 \mu\text{s}$ (period of the 40 MHz frequency); program execution time (in μs) derived from $T_{\text{NODE_TOTAL}}$.
Q_{NODE}	Queue that stores the nodes' properties based on the program execution flow; nodes in the loop are unrolled before storing.
$T_{\text{IO_acc}}$	Total clock cycle count accumulated for consecutive IO nodes, starting from a CPU node until <i>node_lp_config</i> is detected.
$Q_{\text{TIO_acc}}$	Queue that stores $T_{\text{IO_acc}}$ at each check point.
<i>node_lp_config</i>	Potential node in which to insert low-power instructions.
$Q_{\text{node_lp_config}}$	Queue that stores the indices that correspond to <i>node_lp_config</i> .
$n_{\text{data_dep_acc}}$	Number of data dependencies accumulated for consecutive IO nodes, starting from a CPU node until <i>node_lp_config</i> is detected.
$Q_{\text{ndata_dep_acc}}$	Queue that stores $n_{\text{data_dep_acc}}$ of each check point.
$T_{\text{o_DVFS}} (\mu\text{s})$	Overhead (in μs) when changing operating frequency and voltage.
$T_{\text{o_PR}} (\mu\text{s})$	Overhead (in μs) for the PR operation.

4.1 Step 1: Program Segmentation Process

The first step carried out by the ERPA is to segment and label the given program (which includes the user program, the exception handler, and the interrupt service routine) into nodes based on jump instructions, branch instructions, or a new label, before the program flow changes. An example is illustrated in Figure 4.2. Each instruction in a node is executed before branching to another node. This way, the program flow can be systematically predicted by the ERPA. In this step, each node is named based on an index

and the corresponding label. For example, in Figure 4.2, “loop” is split into two segments due to the jump instruction (jal), where the first node is named 0_loop and the second node is named 1_loop.

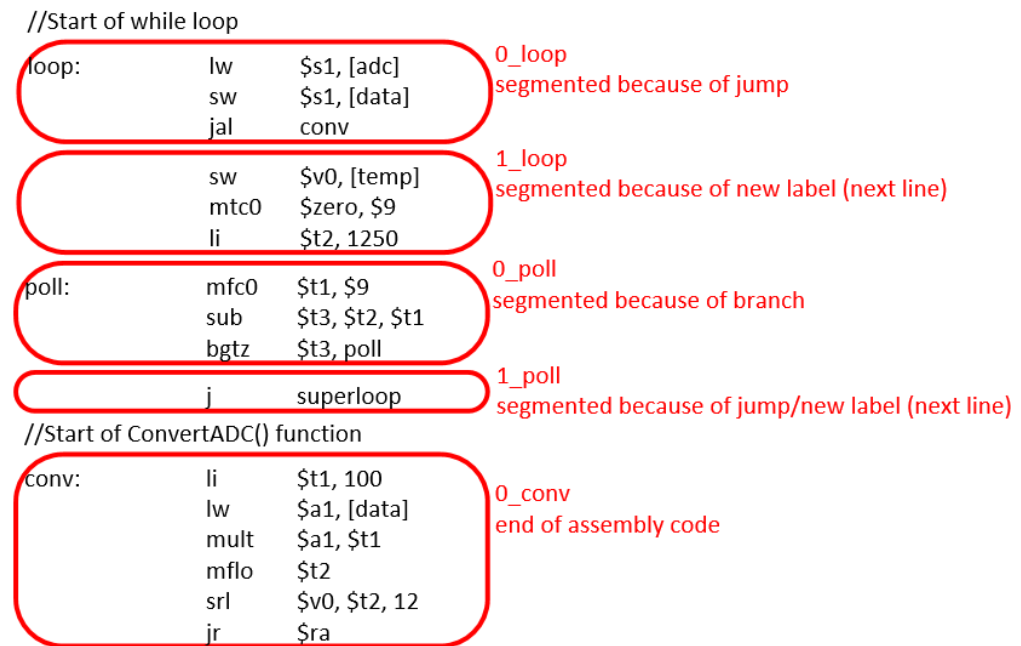


Figure 4.2 Assembly code of the example user program and the result after segmentation process.

4.2 Step 2: Link the Nodes

After the assembly code is segmented into nodes, they are linked according to the program flow. A cyclic node graph will be formed at the end of this step as shown in Figure 4.3.

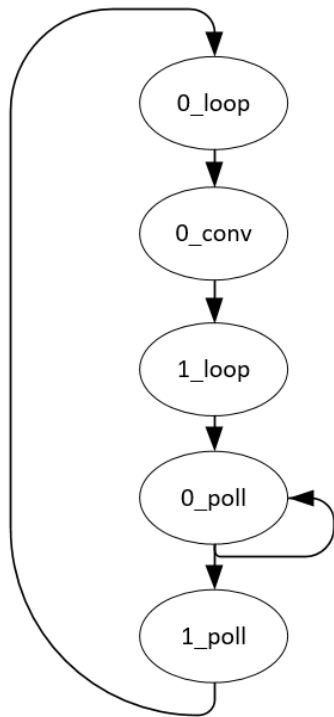


Figure 4.3 Example of the cyclic node graph created.

4.3 Step 3: Simulate Program for Clock Cycle Calculation.

After the node graph is generated, the ERPA simulates the program and records the clock cycle counts that each node is needed to execute, including cache-miss and IO-bound tasks. During this simulation, the sequence for node execution is stored in the node queue (Q_{NODE}). The simulation ends when every node in the graph is simulated.

Figure 4.4 shows an example of the Q_{NODE} content, the clock cycle count for each node, and the total clock cycle count for program execution. This information is necessary to determine a suitable clock frequency and the microarchitecture to be used for each node.

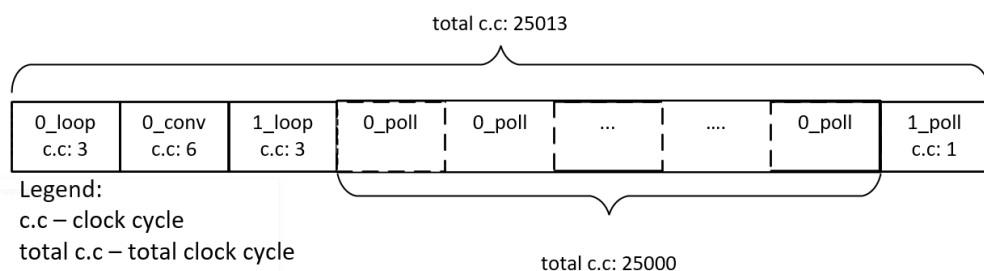


Figure 4.4 Q_{NODE} content based on the example user program.

4.4 Step 4: Node classification

Next, the ERPA classifies the nodes into IO nodes (for IO-bound tasks) or CPU nodes (for CPU-bound tasks). This classification process is needed to determine the suitable clock frequency and the microarchitecture to be used to maximize energy reduction. The classification result of the example user program is shown in Figure 4.5. A CPU node is expected to have a longer execution time when the clock frequency is reduced or when the microarchitecture switches from PE to ME. On the other hand, the execution time for an IO node is not affected in this situation. At this step, any conditional branching loops are identified, wherein the loop can be constructed from one or more nodes. The instructions of the identified branching loops are analyzed and classified into either IO or CPU nodes. In Figure 4.5, the circled 0-poll node is the only branching loop to be identified, so only the instructions of this particular node are analyzed. The node classification process is described below.

1. Identify the exit-condition register of the branching loop. The last instruction will always be the conditional branching instruction. Therefore, \$t3 from `bgtz $t3,poll` are labelled as the exit condition register.
2. Identify the registers that the exit-condition register (\$t3) depends on. In our example shown in Figure 4.2, it is the sub `$t3, $t2, $t1`. Since the target of this instruction is the exit condition register (\$t3), \$t1 and \$t2 are labelled as dependent registers. Continue this process to trace the dependent register(s) until the first instruction of the loop has been reached. In this example, the operation stops at `mcf0 $t1, $9`, where \$t1

depends on the value of \$t9, so the dependent register \$t1 is replaced with \$9. The operation concludes that the dependent registers for \$t3 are \$t2 and \$9.

3. Compare exit-condition and dependent registers. If the exit-condition register differs from any of the identified dependent registers, then the node is an IO node; otherwise, it is classified as a CPU node.

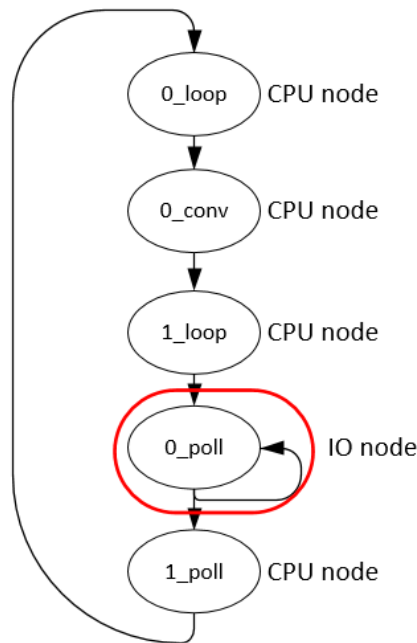


Figure 4.5 Classification of nodes in the cyclic graph.

4.5 Step 5: Identify Program Type

In this step, the program pattern is identified as either timer interrupt or polling-based, which can be checked from the timer interrupt enable (IE) flag. If the IE flag is set, then the program is identified as timer-interrupt-based; otherwise, it is polling-based. For a timer-interrupt program, every superloop (the infinite loop shown in Table 4.2) must be completed within the timer interrupt value ($T_{\text{TIMER_INT}}$). This parameter is useful for indicating when a new superloop will begin execution. We assume that $T_{\text{TIMER_INT}}$ is invariably larger than the total number of clock cycles needed to complete a superloop

($T_{\text{NODE_TOTAL}}$). However, for a polling-based program, there will be no time constraint on completing a superloop.

4.6 Step 6: Search for a Potential Node to Insert Low-Power

Instructions.

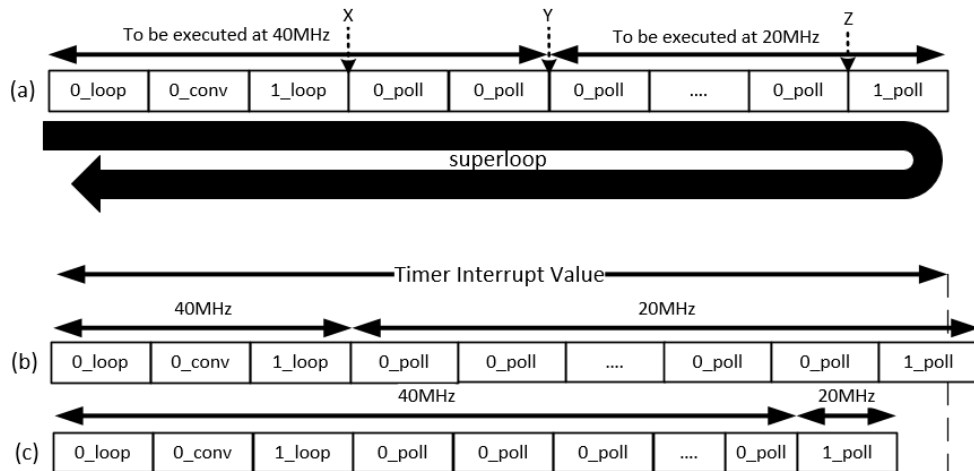


Figure 4.6 Node arrangements in QNODE based on the example user program (timer-interrupt-based).

Finding a potential node for inserting low-power instructions (SB or TMA) is only done once in a superloop. If a node is in a loop and is chosen for low-power instructions insertion, then the problem of repeatedly executing the low-power instruction will arise due to the looping. Figure 4.6 (a) illustrates this problem. Consider the case if we execute the program from the beginning until point Y at 40 MHz and switch to 20 MHz thereafter. Then, 0_poll will be selected for insertion of the SB (20 MHz) instruction. However, in reality, the program will start using 20 MHz at point X, since that is the starting point for 0_poll loop execution, as shown in Figure 4.6 (b). This will cause the superloop program execution time to increase and exceed $T_{\text{TIMER_INT}}$. Therefore 1_poll instead of 0_poll is selected for inserting SB (20 MHz) as shown in Figure 4.6 (c). Even though the energy savings are not as good as the

ideal case in Figure 4.6 (a), the program will be completed within the $T_{\text{TIMER_INT}}$ constraint, which will ensure timing correctness in program execution. The potential node identified through this process is marked as `node_lp_config` (low-power configurable node). Note that not all nodes marked as `node_lp_config` will have low-power instructions inserted. The selection will be based on the time overhead from T_{o_DVFS} and T_{o_PR} .

4.7 Step 7: Insert Low-Power Instructions based on the Program Type

In this phase, the best $f-v$ pair for a particular node is selected to optimize the energy performance. At the same time, the ERPA determines if it is worth inserting the TMA instruction for a further energy reduction from changing the microarchitecture.

4.7.1 Polling-based Program

Table 4.4 Pseudo algorithm used by ERPA to insert DVFS and TMA instructions in a polling-based program.

Function used:	
<code>get_clock_cnt()</code> – Return the clock count needed to complete a node execution.	
<code>get_data_dep()</code> – Get <code>ndata_dep_acc</code> from the nodes.	
<code>set_freq(X)</code> – Insert the SB instruction into the corresponding node. Parameter X is the operation frequency to be set.	
<code>insert_TMA()</code> – Insert TMA instruction into the corresponding node.	
<code>push(Y)</code> – Push parameter Y as the last element in the queue.	
<code>at(Z)</code> – Get the element at index Z in a queue.	
<code>is_node_lp_config()</code> – Return true if a node is <code>node_lp_config</code> else return false.	
τ = threshold for microarchitecture switch (PE to ME) based on <code>ndata_dep_acc</code> .	
<code>index_lp</code> = selected node from <code>Q_{index_pl_config}</code> for low-power instruction insertion.	
<code>size_QNODE</code> = size of <code>QNODE</code>	
1	for <code>n = size_QNODE-1, size_QNODE-2, ..., 0</code> do
2	If <code>QNODE.at(n)</code> is CPU then
3	<code>T_{IO_acc} = 0;</code>
4	<code>ndata_dep_acc = 0;</code>
5	else
6	<code>T_{IO_acc} += QNODE.at(n).get_clock_cnt();</code>

```

7         ndata_dep_acc += QNODE.at(n).get_data_dep();
8     end
9     if (QNODE.at(n).is_node_lp_config()) then
10         QTIO_acc.push(TIO_acc);
11         Qindex_lp_config.push(n);
12         Qndata_dep_acc.push(ndata_dep_acc);
13     end
14 end
15 size_QTIO_acc = size of QTIO_acc
16 for n = 0, 1, 2, ..., size_QTIO_acc-1 do
17     if QTIO_acc.at(n) > To_DVFS then
18         QNODE.at(index_lp).set_freq(20);
19         if (QTIO_acc.at(n) > To_DVFS + To_PR &&
20 Qndata_dep_acc.at(n) >  $\tau$ ) then
21             QNODE.at(index_lp).insert_TMA();
22         end
23     else
24         QNODE.at(index_lp).set_freq(40);
25     end
26 end

```

For a polling-based program, a CPU node always executes at the highest performance (at 40MHz in PE mode) to minimize the program execution time. However, an IO node executes at the lowest operating frequency when the time spent by consecutive IO nodes is more than T_{o_DVFS} ; otherwise, it remains at 40 MHz. Referring to lines 1–8 in Table 4.4, the Q_{NODE} is scanned from the last node to the first. If an IO node is detected, the clock cycle counts (T_{IO_acc}) of the consecutive IO nodes are accumulated. If a CPU node is detected, T_{IO_acc} is reset to zero, because the node does not contribute to the clock cycle count accumulation of consecutive IO nodes. Next, in lines 9–13, when a `node_lp_config` node is detected (whether IO or CPU), the current T_{IO_acc} and node n are saved into Q_{TIO_acc} and $Q_{index_lp_config}$, respectively. This process continues until all nodes in Q_{NODE} are checked. Subsequently, in lines 16–26 of Table 4.4, each element in Q_{TIO_acc} is compared with the configuration time for DVFS (T_{o_DVFS}). The element with a

value larger than T_{o_DVFS} is obtained, so that the corresponding node can be identified and switched to the lowest frequency (20 MHz). This is done to ensure that the time spent in IO nodes (T_{IO_acc}) is longer than the frequency configuration time (T_{o_DVFS}), so that it is worth executing the DVFS operation.

At the same time, the IO nodes are configured to ME via the TMA instruction for further energy reduction, provided the following condition is met: the accumulated data dependency count ($n_{data_dep_acc}$) of the IO nodes is larger than the defined threshold (τ). In our experiment, we found that, during the IO transfer, when the forwarding circuitry (in PE mode) is actively resolving data dependencies, relatively high power is consumed. Hence, the data dependency occurrence or count is an important factor for determining microarchitecture reconfigurations between PE and ME. Kiat et al. proved that the microarchitecture PR process managed to save energy when the program's data size was at least 512 bytes or more (Kiat et al, 2020). Hence, to ensure that the PR approach saves energy, τ is obtained by calculating the $n_{data_dep_acc}$ of the test program executed in the ME microarchitecture with a 512-byte data size. In other words, if the energy reduction in ME mode is more than the energy overhead for PR, then ME is selected for the IO nodes; otherwise, PE is used. The accumulated data dependency value ($n_{data_dep_acc}$) is calculated in the same manner as T_{IO_acc} ; $n_{data_dep_acc}$ is stored in $Q_{n_{data_dep_acc}}$ when $node_lp_config$ is detected (for IO and CPU nodes). In lines 18–21 of Table 4.4, the ERPA checks each element in $Q_{n_{data_dep_acc}}$ and Q_{TIO_acc} . If the n -th element in Q_{TIO_acc} is greater than $T_{o_DVFS} + T_{o_PR}$ and the n -th element in $Q_{n_{data_dep_acc}}$ is greater than the τ defined earlier, then TMA will be inserted into the corresponding $node_lp_config$ node so that

it operates in ME. This ends the insertion of low-power instructions for polling-based programs.

4.7.2 Timer-Interrupt-based Program

Table 4.5 Pseudo algorithm used by ERPA to insert DVFS and TMA instructions in an interrupt-based program.

Function used:	
get_clock_cnt() – Return the clock count needed to complete a node execution.	
get_data_dep() – Get ndata_dep_acc from the nodes.	
set_freq(X) – Insert the SB instruction into the corresponding node. Parameter X is the operation frequency to be set.	
insert_TMA() – Insert TMA instruction into the corresponding node.	
push(Y) – Push parameter Y as the last element in the queue.	
at(Z) – Get the element at index Z in a queue.	
is_node_lp_config() – Return true if a node is node_lp_config else return false.	
getExtraTime(X) – Return additional node execution time when the operating frequency is lowered from 40MHz to X.	
getExtraCycleTMA() – Return additional clock cycles needed for the node to complete execution in ME compared to PE.	
τ = threshold for microarchitecture switch (PE to ME) based on ndata_dep_acc.	
index_lp = selected node from $Q_{\text{index_pl_config}}$ for low-power instruction insertion.	
size_QNODE = size of QNODE	
1	for n = size_QNODE-1, size_QNODE-2, ..., 0 do
2	if QNODE.at(n).is_node_lp_config() then
3	if $T_{\text{idle}} > Q_{\text{NODE}}.\text{at}(n).\text{getExtraTime}(20) + T_{o_DVFS}$ then
4	QNODE.at(n).set_freq(20); f = 20;
5	else
6	if $T_{\text{idle}} > Q_{\text{NODE}}.\text{at}(n).\text{getExtraTime}(24) + T_{o_DVFS}$ then
7	QNODE.at(n).set_freq(24); f = 24;
8	else
9	if $T_{\text{idle}} > Q_{\text{NODE}}.\text{at}(n).\text{getExtraTime}(28) + T_{o_DVFS}$ then
10	QNODE.at(n).set_freq(28); f = 28;
11	else
12	if $T_{\text{idle}} > Q_{\text{NODE}}.\text{at}(n).\text{getExtraTime}(32) + T_{o_DVFS}$ then
13	QNODE.at(n).set_freq(32); f = 32;
14	else
15	if $T_{\text{idle}} > Q_{\text{NODE}}.\text{at}(n).\text{getExtraTime}(36) + T_{o_DVFS}$ then
16	QNODE.at(n).set_freq(36); f = 36;
17	else
18	QNODE.at(n).set_freq(40); f = 40;
19	end
20	end
21	end
22	end

23	end
24	$Q_{\text{index_lp_config}}$.push(n);
25	$T_{\text{idle}} = T_{\text{idle}} - Q_{\text{NODE}}.\text{at}(n).\text{getExtraTime}(f)$;
26	end
27	end
28	if $T_{\text{idle}} > T_{o_PR} * 2$ then
29	$T_{\text{idle}} -= T_{o_PR} * 2$;
30	for n = size_0-1, ..., 2, 1, 0 do
31	if $T_{\text{idle}} > Q_{\text{NODE}}.\text{at}(n).\text{getExtraCycleTMA}()$ then
32	$T_{\text{idle}} -= Q_{\text{NODE}}.\text{at}(n).\text{getExtraCycleTMA}()$;
33	else
34	break;
35	end
36	$\text{ndata_dep_acc} += Q_{\text{NODE}}.\text{at}(n).\text{get_data_dep}()$;
37	if $Q_{\text{NODE}}.\text{at}(n).\text{is_node_lp_config}()$ then
38	$\text{TMA_node_1} = n$;
39	end
40	end
41	$\text{TMA_node_0} = \text{size_0} - 1$;
42	end
43	if $\text{ndata_dep_acc} > \tau$ then
44	$Q_{\text{NODE}}.\text{at}(\text{TMA_node_0}).\text{insert_TMA}()$;
45	$Q_{\text{NODE}}.\text{at}(\text{TMA_node_1}).\text{insert_TMA}()$;
46	End

For interrupt-based programs, the ERPA attempts to reduce the operating frequency of each node in Q_{NODE} . In lines 1–27 of Table 4.5, the process starts from the last node in Q_{NODE} , which is marked as node_lp_config . When the operating frequency is reduced, T_{EXE} and T_{IDLE} will be updated according to the node type. For CPU nodes, node execution time will increase (due to lowering of the operating frequency), which, in turn, stretches the overall T_{EXE} (and reduces T_{IDLE}). In contrast, when the operating frequency is reduced, this will not affect the IO nodes' execution time, since the latter depend on the IO module baud rate. The algorithm then continues with next-to-last node_lp_config until all nodes are checked or $T_{\text{IDLE}} < 0$.

Next, in lines 28–43, if $T_{\text{IDLE}} > T_{o_PR} * 2$ after the frequency is tuned, then the ERPA will insert two TMAs at the correct locations to toggle the

microarchitecture back and forth. The accumulated data dependencies (ndata_dep_acc) between the two TMA locations need to be greater than τ . The idea is to use up the slack, T_{IDLE} , as much as possible. Like frequency tuning, the process starts with the last node_lp_config node until all nodes in Q_{NODE} are processed or $T_{IDLE} < 0$. Note that we only need TMAs in the following cases:

1. From ME to PE when executing CPU nodes (corresponds to data sampling and processing tasks).
2. From PE to ME when executing IO nodes (corresponds to data transmission).

4.8 Example of EFRA on tuning frequency to optimal value.

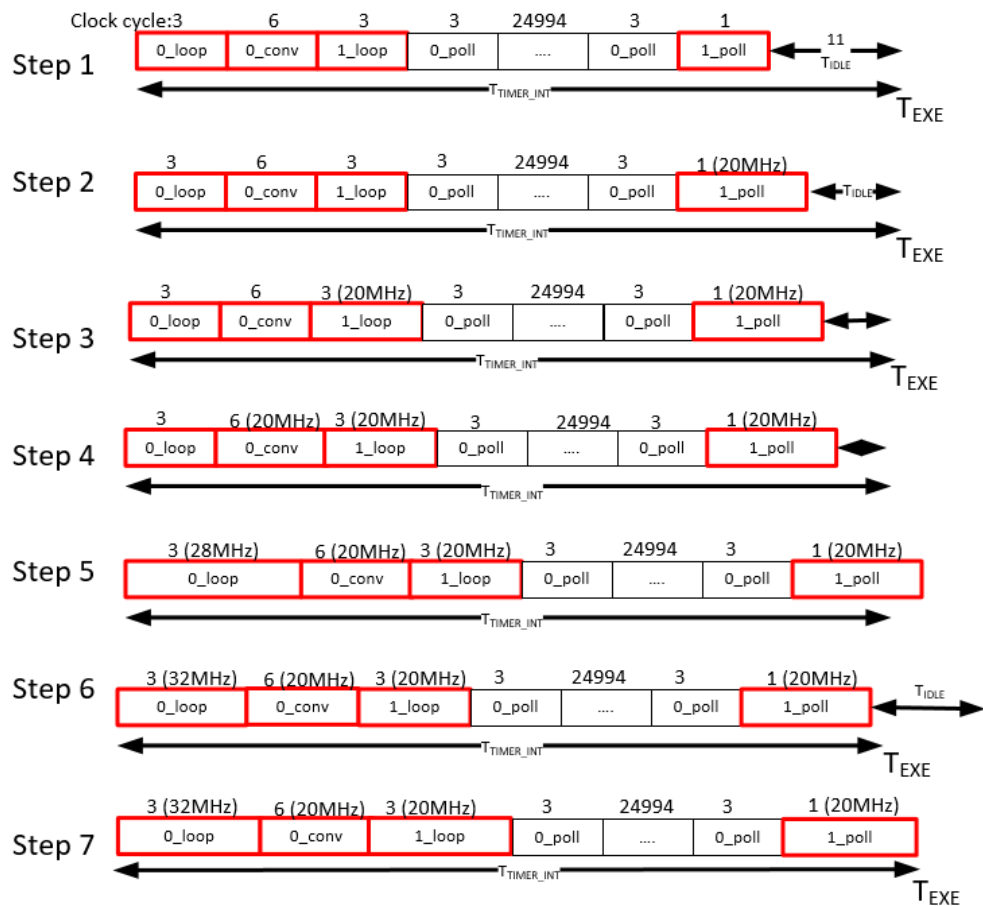


Figure 4.7 ERFA steps on insert SB instruction for DVFS feature in RISC32-LP.

Figure 4.7 shows how the ERFA inserts SB instructions that control the DVFS feature in RISC32-LP. The same example user program is used to explain the steps.

1. The ERFA identifies the idle time, T_{IDLE} , which is located after the user program executed with 40 MHz clock frequency.
2. The ERFA utilizes the T_{IDLE} by reducing the operating frequency of the last `node_lp_config` (i.e., the “1_poll”). ERFA pre-sets the frequency to be as low as possible, which is 20 MHz. T_{IDLE} is reduced by the extra time needed by “1_poll” to execute at 20 MHz, but there is still T_{IDLE} available.
3. ERFA continues to search for `node_lp_config` (starting from the last node). “1_loop” is found to be the next `node_lp_config`. Similar to step 2, the “1_loop” is pre-set to be executed at 20 MHz, and T_{IDLE} further reduces accordingly.
4. ERFA repeats step 3 on the “0_conv” node.
5. ERFA repeats step 3 on “0_loop” node, but at this step, the remaining T_{IDLE} is insufficient for “0_loop” to run at the frequency 28 MHz. ($T_{IDLE} < 0_loop.getExtraTime(28) + T_{o_DVFS}$, refer to Table 4.5).
6. Due to that, “0_loop” is pre-set to execute at 32 MHz, which is the operation frequency that is a step higher than 28 MHz (refer to Table 3.6).
7. After all the nodes have been searched, ERFA will insert SB (32 MHz) and SB (20 MHz) instructions at “0_loop” and “0_conv” nodes, respectively, to tune the operation frequency. There is no need to insert

extra SB instruction at “1_loop” and “1_poll” since they are maintained as 20 MHz after “0_conv” has been tuned to 20 MHz.

CHAPTER 5

ENERGY REDUCTION EXPERIMENT AND RESULT

The energy measurement is performed on Nexys 4 DDR board with Xilinx Artix-7 XC7A100T FPGA chip, 4860 Kbits BRAM, on-chip ADC, and 16 MB serial flash memory.

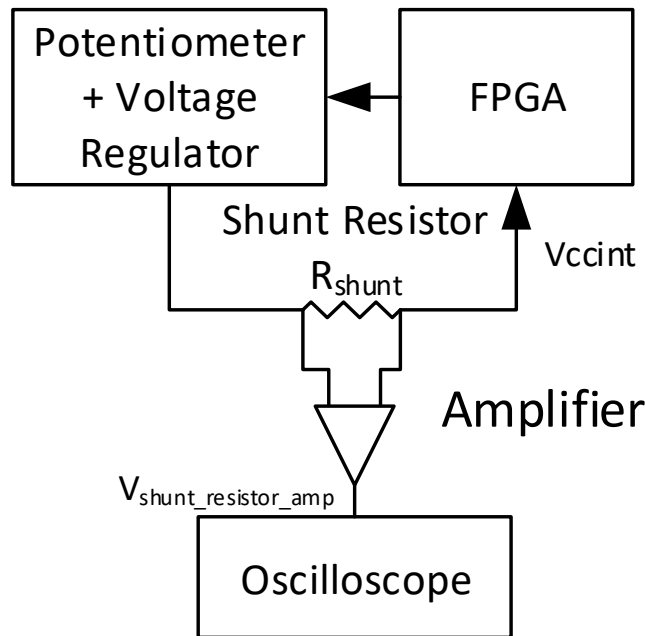


Figure 5.1 Hardware setup for energy measurement during the experiment.

To measure the current supplied to the FPGA chip, a 0.01 Ohm shunt resistor is connected serially in between the external voltage regulator and Xilinx FPGA chip as shown in Figure 5.1. An amplifier is used to amplify voltage of the 0.01 Ohm resistor, and the output of amplifier, $V_{shunt_resistor_amp}$, is recorded. The voltage supply, V_{ccint} and $V_{shunt_resistor_amp}$, are monitored and recorded during the experiments by using an oscilloscope. Since the connected shunt resistor has the same current flow as FPGA chip, the current flow through the FPGA, I_{ccint} , can be obtained by using formula (1). With the obtained

current value, I_{ccint} , the energy consumed by RISC32-LP, $E_{RISC32-LP}$, during the experiment can be calculated by using formula (2).

$$I_{ccint} = \frac{V_{shunt_resistor_amp} / \text{Amplifier gain}}{0.01 \Omega} \quad (1)$$

$$E_{RISC32_LP} = (V_{ccint} * I_{ccint}) * t_{exe} \quad (2)$$

where t_{exe} is test program execution time.

Two types of test program (polling- and interrupt-based) were developed based on typical data transmit operations in IoT sensor nodes. Both programs carry out the same tasks, which are described as follows:

- 1) Data collection: Read N bytes of data from ADC.
- 2) Data processing: Encrypt N bytes of data received using AES-128.
- 3) Data transmission: Send out the encrypted data through UART.

The N-byte data is read from ADC, where N ranges from 64 to 4096. The N cannot be more than 4096 due to the limited 8196-byte RAM memory in RISC32-LP. The RAM is used to store other program's variables besides the N-byte data. Next, the N-byte data is encrypted by using AES-128 algorithm. Finally, the encrypted data is sent out via UART. These three tasks are repeated for 5 minutes, and the total energy consumed during this duration is measured. In practical IoT applications, the sensor node usually accumulates a large amount of data (larger than 1 KB) before transmitting the same to a gateway device. This is done to reduce the energy consumption of the sensor node by avoiding the activation of the IO (which connects to the wireless external module) frequently. For example, sensor nodes in a sensor network take turn to transmit data when they are utilizing the multi-hop communication (Liew et al., 2018) mechanism. In this situation, sensor nodes accumulate the data and store them in local memory (RAM) while waiting for their turn to transmit. These experiments are set up according to this kind of IoT

application settings. The data collection and AES-128 encryption involved large number of arithmetic and logic operations, while data transmission involves IO communication (UART) and memory transfer operations. These program tasks are grouped into two categories:

1) Computationally intensive tasks (CPU-bound tasks): data collection and encryption.

2) Computationally less-intensive tasks (IO-bound task): data transmission.

For computationally intensive tasks, the high-performance setting (40 MHz operating frequency and PE) is used to reduce the energy consumption by completing the tasks faster. On the other hand, for computationally less-intensive tasks (which do not involve many logical and arithmetic operations) low-performance setting (20 MHz operating frequency and ME) is used to achieve low power consumption. However, in some situations, the low-performance setting will not be used when the overhead of PR and DVFS is greater than the energy it can save. Hence, to achieve the optimal energy efficiency in the dynamic scenario discussed above, the ERPA is developed to assist in determining frequency values and the microarchitecture to be used throughout the program execution.

We use two types of test programs (polling- and interrupt-based) to monitor the transmit buffer in UART module.

The first type of test program used polling-based method to monitor the UART's transmit buffers, while the second type of program uses interrupt-based method for the same. In polling test program, the core of RISC32-LP always checks the condition of UART's transmit buffers during IO communication operation. UART needs to send all 4-byte data and make sure

that the transmit buffer is empty before CPU writes another 4-byte of encrypted data. However, in interrupt test program, the CPU core is idle after completing the data collection and processing. The CPU core only writes data into UART's transmit buffers when the UART module interrupt takes place. Owing to this difference, the energy consumption of these two programs varies greatly, which we will discuss in the next section, mainly due to the operating status of CPU core.

To verify that the proposed ERPA can achieve the most energy savings compared to other methods, both types of programs were executed under six different test cases. These cases measured the program's energy savings from using CG, DVFS, PR, a combination of all low-power techniques and ERPA on RISC32-LP. Each test case is described below:

1. TST_{MAX}: Execute the test programs on RISC32 without any low-power techniques.
2. TST_{CG}: Execute the test programs on RISC32-LP with CG enabled throughout the experiment but with no extra low-power techniques used.
3. TST_{DVFS}: Execute the test programs on RISC32-LP with only the DVFS instruction, SB inserted manually. CPU nodes ran at 40 MHz, and IO nodes ran at 20 MHz.
4. TST_{PR}: Execute the test programs on RISC32-LP, but only PR instruction TMA was inserted manually. CPU nodes ran under PE, and IO nodes ran under ME.

5. TST_{COMB} : Execute the test programs on RISC32-LP while CG is enabled and with DVFS and PR instructions inserted manually. CPU nodes ran under PE at 40 MHz, and IO nodes ran under ME at 20MHz.
6. TST_{ERPA} : The test programs were analyzed and modified by ERPA to achieve low power.

TST_{MAX} represents the test programs executed on RISC32 without any low-power techniques applied, which is the base case for comparison. TST_{CG} , TST_{DVFS} , and TST_{PR} represent the test programs executed on RISC32-LP with only a single low-power technique applied (CG, DVFS, or PR). TST_{COMB} represents the test programs executed on RISC32-LP with all the developed low-power techniques (CG, DVFS, and PR) applied. However, TST_{ERPA} represents the test programs executed on RISC32-LP with all developed low-power techniques applied but controlled by the ERPA according to the test programs' behavior. Table 5.1 summarizes the description of each test case.

Table 5.1 Short description for each test cases.

Test Name	Description
TST_{MAX} (No low-power technique applied)	Microarchitecture: Pipeline Clock frequency: 40 MHz
TST_{CG}	Microarchitecture: Pipeline with CG feature Clock frequency: 40 MHz
TST_{DVFS} (manual assign)	Micro-architecture: Pipeline Clock frequency: 40 MHz (CPU-bound), 20MHz (IO-bound)
TST_{PRR} (manual assign)	Microarchitecture: Pipeline (CPU-bound), multi-cycle (IO-bound) Clock frequency: 40 MHz
TST_{COMB} (manual assign)	Microarchitecture: Pipeline (CPU-bound), multi-cycle (IO-bound) with CG feature Clock frequency: 40 MHz (CPU-bound), 20 MHz (IO-bound)
TST_{ERFA}	Microarchitecture with CG feature and clock frequency assign by ERFA.

5.1 Polling-based Test Program

5.1.1 Program Behavior

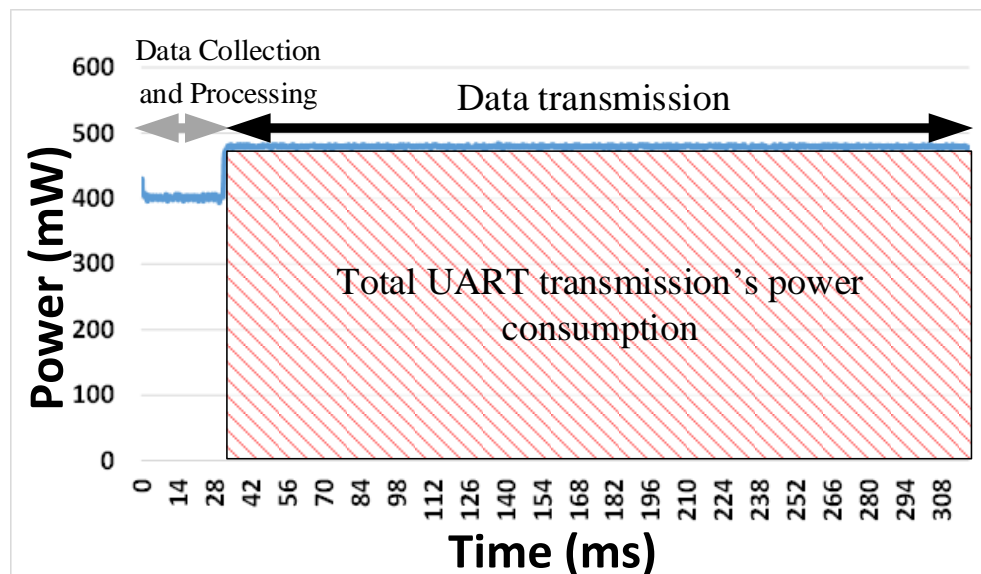


Figure 5.2 Power consumption of polling test program (TST_{MAX}) with 256-byte data size.

The polling-based test program samples and encrypts N bytes of data, then transmits all the encrypted data through UART. Figure 5.2 shows the power consumption of the polling test program for a 256-byte data transfer.

5.1.2 Result based on Polling Test Program

Tables 5.2 and 5.3 show the result for all the test cases. Figures 5.3 and 5.4 are the graphs plotted based on the results.

Table 5.2 Energy consumption of polling test program.

Data Size (Bytes)	Energy Consumption (mJ)					
	TST_{MAX}	TST_{CG}	TST_{PR}	TST_{DVFS}	TST_{COMB}	TST_{ERPA}
64	7.06	6.95	19.70	4.34	7.52	4.26
128	14.90	14.53	16.54	8.98	11.51	8.72
256	30.07	29.44	30.16	18.02	19.54	17.63
512	60.15	58.41	56.89	35.85	35.64	35.94
1024	118.22	114.85	111.07	70.57	65.05	68.51
2048	249.40	243.67	216.65	149.66	134.36	135.12
4096	498.84	487.83	437.63	298.27	263.84	265.03

Table 5.3 Energy saving of polling test program.

Data Size (Bytes)	Energy Saving (%)				
	TST _{CG}	TST _{PR}	TST _{DVFS}	TST _{COMB}	TST _{ERPA}
	vs TST _{MAX}				
64	1.61	-178.87	38.59	-6.49	39.65
128	2.47	-11.06	39.73	22.74	41.47
256	2.08	-0.33	40.07	35.31	41.36
512	2.90	5.43	40.40	40.74	40.24
1024	2.85	6.04	40.31	44.97	42.05
2048	2.30	13.13	39.99	46.13	45.82
4096	2.21	12.27	40.21	47.11	46.87

Data Size (Bytes)	Energy Saving (%)					
	TST _{COMB} vs			TST _{DVFS}	TST _{ERPA}	
	TST _{CG}	TST _{PR}	TST _{DVFS}	vs TST _{COMB}	vs TST _{DVFS}	vs TST _{COMB}
64	-8.23	61.81	-73.41	42.33	1.72	43.32
128	20.78	30.44	-28.18	21.99	2.89	24.24
256	33.94	35.52	-7.94	7.35	2.15	9.35
512	38.97	37.34	0.57	-0.57	-0.27	-0.84
1024	43.36	41.43	7.81	-8.47	2.92	-5.30
2048	44.86	37.99	10.23	-11.39	9.71	-0.57
4096	45.91	39.71	11.54	-13.05	11.14	-0.45

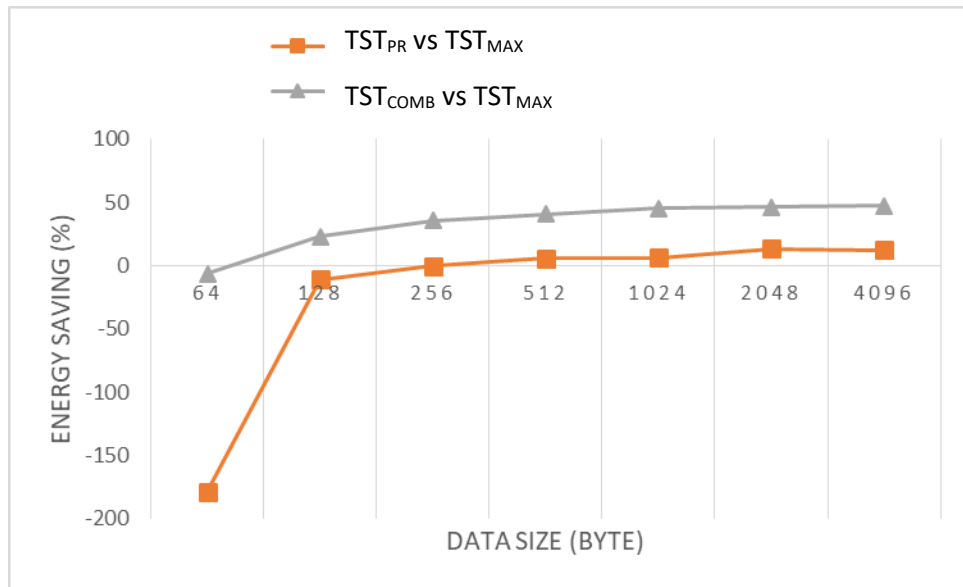


Figure 5.3 Energy saving for (TST_{PR} vs TST_{MAX}) and (TST_{COMB} vs TST_{MAX}). TST_{PR} did not achieve energy saving from 64-byte to 256-byte due to overhead.

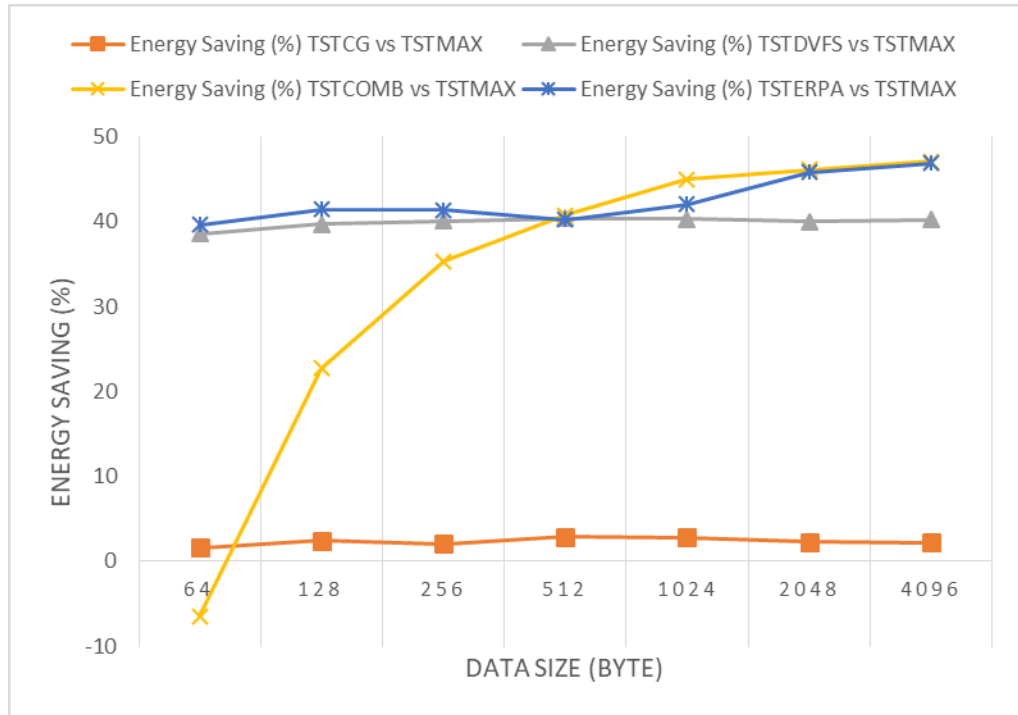


Figure 5.4 Energy saving for (TSTCG vs TSTMAT), (TSTDVFS vs TSTMAT), (TSTCOMB vs TSTMAT) and (TSTERPA vs TSTMAT). Energy saving in TSTCOMB is lower than TSTERPA due to PR overhead shown in Figure 5.3.

In the column “TSTCG” of Table 5.2, it can be seen that CG can save up to 1.6% to 2.2%, which can be averaged out to 2% of energy consumption across different data sizes. Although CG manages to reduce unnecessary switching activities in RISC32-LP, the energy saving achieved is not significant. On the other hand, microarchitectural PR is only able to save energy when the data size is larger than 512 bytes. This implies that microarchitectural PR should be avoided for small data size, because the overhead caused by PR is more significant than the energy saving achieved, for the data size that is lower than 512 bytes. Despite this disadvantage, PR manages to reduce energy around 12% at a 4096-byte data size. DVFS (TSTDVFS) manages to reduce the energy consumption around 40% throughout the experiment, even for small data sizes, because the overhead of configuring DVFS is smaller than the energy saving achieved.

By combining all the low-power techniques (TST_{COMB}), we can achieve 11.5% extra energy saving compared to using DVFS only at 4096-byte data size. However, it consumes extra 42.33% energy compared to TST_{DVFS} at 64-byte due to the additional overhead introduced by PR as shown in Figure 5.3. This shows that even if we combine all the low-power techniques developed, we cannot achieve energy saving when executing tasks with smaller data size. Hence, we proposed to use the ERPA to control PR and DVFS based on the program behavior profile to get the best energy saving. TST_{ERPA} can achieve better energy reduction compared to other TSTs, and comparable to TST_{COMB} when the data size increases more than 512 bytes.

5.2 Interrupt-based Test Program

5.2.1 Program Behavior

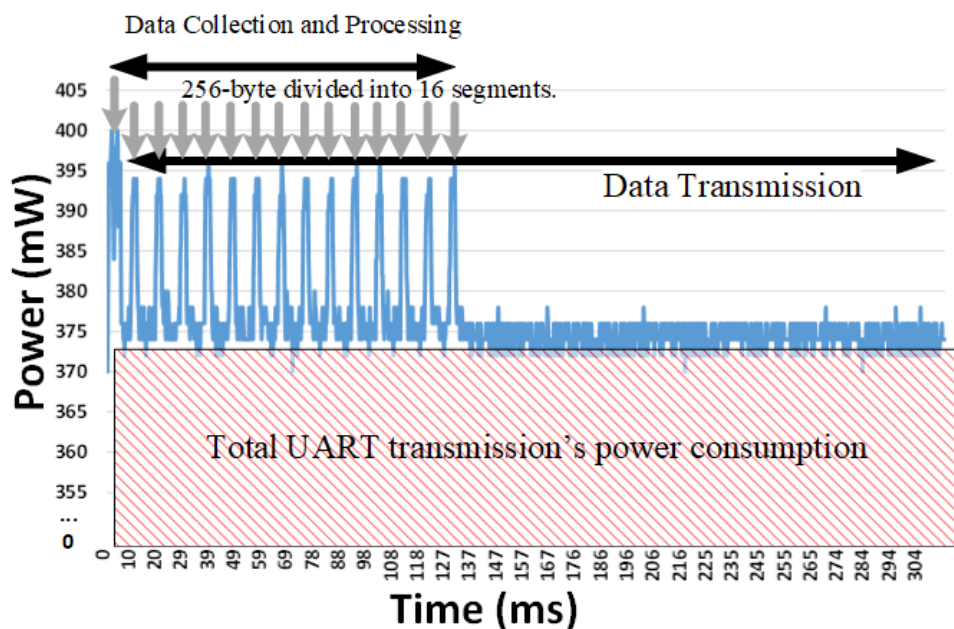


Figure 5.5 Power consumption of interrupt test program (TST_{MAX}) with 256-byte data size

Interrupt test program splits the N-byte data into 16 data segments. The first data segment (N/16 bytes) is sampled, encrypted and then transmitted

through UART. While transmitting the first data segment, the second data segment is sampled and encrypted in parallel. The second data segment will enter the transmission pipeline after the first data segment is transmitted. Referring to Figure 5.5, the following data segments are processed in the similar fashion until all data segments are completely transmitted.

5.2.2 Result based on Interrupt Test Programs

From the polling test program results, we observed that the microarchitectural PR will only start saving energy for data size of 256 bytes onward. Hence, in interrupt test program, the minimum data size was chosen to be 256 bytes. In the interrupt test program, the minimum data size of each data segment must be 16 bytes due to AES-128 encryption, which uses 16-byte data as input. Hence the number of data segments selected is also 16, obtained by dividing minimum data size (256 bytes) with a data segment size (16 bytes).

Referring to Table 5.4, CG manages to save energy up to 1.9%, which is a small number, similar to the result in Table 5.2 for polling-based program. Similarly, DVFS managed to save 41% of energy, which is close to the result in polling-based program.

Table 5.4 Energy consumption of interrupt test program.

Data Size (Bytes)	Energy Consumption (mJ)					
	TST _{MAX}	TST _{CG}	TST _{PR}	TST _{DVFS}	TST _{COMB}	TST _{ERPA}
256	27.95	27.52	28.58	16.61	16.41	16.59
512	42.46	41.07	42.89	24.95	24.48	24.83
1024	69.62	68.77	69.48	40.98	39.31	40.12
2048	126.21	124.07	125.50	74.06	71.11	72.79
4096	235.58	231.03	233.15	138.44	132.20	132.33
8192	461.05	452.53	458.75	270.98	259.70	260.71

Table 5.5 Energy saving of interrupt test program.

Data Size (Bytes)	Energy Saving (%)				
	TST _{CG}	TST _{PR}	TST _{DVFS}	TST _{COMB}	TST _{ERPA}
	vs TST _{MAX}				
256	1.52	-2.27	40.59	41.27	40.64
512	1.79	-1.02	41.23	42.35	41.53
1024	1.23	0.21	41.14	43.54	42.37
2048	1.69	0.57	41.32	43.65	42.32
4096	1.93	1.03	41.23	43.88	43.83
8192	1.85	0.50	41.23	43.67	43.45

Data Size (byte)	Energy Saving (%)					
	TST _{COMB} vs			TST _{DVFS}	TST _{ERPA}	
	TST _{CG}	TST _{PR}	TST _{DVFS}	vs TST _{COMB}	vs TST _{DVFS}	vs TST _{COMB}
256	40.37	42.58	1.15	-1.17	0.08	-1.08
512	41.30	42.93	1.90	-1.93	0.50	-1.43
1024	42.84	43.42	4.07	-4.24	2.09	-2.06
2048	42.68	43.33	3.98	-4.14	1.71	-2..36
4096	42.78	43.30	4.51	-4.72	4.41	-0.10
8192	42.61	43.39	4.16	-4.34	3.79	-0.39

During the UART transmission in polling-based program, the CPU core keeps monitoring the status of UART transmit buffer to see if it is empty and ready to be loaded with the next encrypted data. This process continues until all the encrypted data is transmitted. However, for interrupt-based program, the CPU core only loads the encrypted data to the UART transmit buffer when the UART module interrupt take place; otherwise, the CPU core remains idle. This causes the power consumption in UART transmission to differ greatly between polling- and interrupt-based programs, which appears as a shaded area under the power—time graph in Figure 5.5. Since the power consumption of UART transmission in interrupt-based program is already low, employing PR in this situation does not reduce much energy consumption. Therefore, the TST_{PR} has relatively worse energy saving in interrupt-based program compared to the polling-based program.

TST_{COMB} in interrupt-based program achieves energy saving up to 4.51% compared to TST_{DVFS} due to the additional energy reduction from the CG and PR. TST_{ERPA} achieves similar performance with TST_{COMB} across all data sizes, which is expected. However, TST_{ERPA} allows automatic

configuration of low-power techniques, which saves the program development time and effort

5.3 Comparison with Existing Works.

Table 5.6 Comparison with the existing low power techniques.

Low-Power Techniques	Single/Multiple Cores	Energy reduction	Complexity	Energy Optimization Tool
Sterpone et al. (2011)	Single	Low	Low	No
Bsoul et al. (2015)	Single/Multiple	High	Medium	No
Kiat et al. (2020)	Single	Medium	Medium	No
Wu et al. (2014)	Single	High	Medium	No
Nunez-Yanez et al. (2017)	Multiple	High	High	Yes
Bramdalero (2019)	Multiple	High	High	Yes
RISC-LP (this work)	Single	High	Medium	Yes

Table 5.6 summarizes the existing works on FPGA-based low-power techniques. CG (Sterpone et al., 2011) can be used to reduce the dynamic power consumption, but its contribution is too small for FPGA-based systems. Similarly, the microarchitectural technique proposed in RISC32 (Kiat et al., 2020) is also limited to dynamic energy reduction only. Power gating (Bsoul et al. 2015) can be used to reduce the energy consumption significantly, but this feature is not available in commercial FPGAs. Free razor technique proposed by Wu et al. (2014) requires a hardware module to collect the BER, which is too costly for IoT sensor nodes. Nunez-Yanez et al. (2017) proposed a similar approach that achieved a much higher energy saving with the help of an energy optimization tool. Note that all these techniques only concern a single low-power technique, which may not be the optimal solution for FPGA-based IoT sensor nodes. In this work, the proposed RISC-LP combines the benefits of CG, PR, and DVFS to reduce the energy consumption significantly. Moreover, the proposed energy optimization tool (ERFA) ensures that the energy consumption in RISC32-LP is always optimized based on the given firmware. This is a feature not commonly found in other works. Compared to

Brandalero (2019), our solution is less complex, because we only target single-core processors. The energy optimization tool provided by previous researchers requires additional DBT hardware resources, but ERFA is a completely software-based solution that does not add energy consumption to the hardware system. On the other hand, this work combines a few low-power techniques, which can be a complementary solution for work done by Brandalero (2019) to reduce the energy consumption further.

CHAPTER 6

CONCLUSION

An FPGA-based soft-core IoT System on Chip (Soc), RISC32 has been enhanced in energy saving aspect by implementing known low-power techniques (DVFS and CG). These techniques are applicable to commercial FPGA are employed and combined with the PR technique presented on previous work done by Kiat et al. (2020).

The energy saving of the combined techniques is measured and shown to have saved up to 47.11% (refer to Table 5.3, column TST_{COMB}) and 43.88% (refer to Table 5.5, column TST_{COMB}) on polling and interrupt test program respectively. However, combining techniques did not guarantee the highest energy saving all the time. The PR approach consumed extra (overhead) energy in the microarchitecture configuration. If the energy saving is not more than the overhead energy, the combined technique will be worse than the DVFS alone as seen in Table 5.3. To gain the benefits of the mentioned approaches, the ERPA is proposed to analyze the program and insert low-power-technique-related instructions (SB and TMA) into the program in accordance with program's behavior. ERPA classified the program's tasks into CPU- and IO-bound tasks. While executing an IO-bound task, the RISC32-LP's core is being idle until the IO-bound tasks are completed. Hence, during execution of an IO task, the RISC32-LP's core is switched to the lowest performance (lowest f-v pair and ME), whenever the predicted energy saving done by DVFS and PR must be more than the energy consumed (overhead energy) by DVFS and PR, respectively. As a result, a combination of RISC32-

LP with ERPA-modified program will always achieve minimum energy consumption.

Kiat et al. (2020) showed the PR technique only manages to reduce the dynamic energy. However, in our work, the overall energy (static and dynamic energy) reduction is obtained because DVFS has effect on both. From our result, DVFS (maximum: 41%) contributes most of the energy reduction compared to CG (maximum: 2.21%) and PR (maximum: 12%). This is because the static energy is the dominating component in our design implemented on the Artix-7 FPGA board, and DVFS is the only technique that manages to reduce static energy.

Based on Table 3.13, the total FPGA resource usage for the pipeline structure is around 13.07%. Much of the FPGA areas are not used for logic circuit, thus consuming unnecessary static power. As the technology shrinks, leakage power also increases, making up a larger component of the total power consumption. This can be seen in Kolluri (2015), wherein at 28 nm, the Artix-7 power ratio of the static versus dynamic is around 1:1. This gives a hint that in our current design, the achieved static: dynamic ratio, which is 2.5:1 as shown in Table 6.1, can be further improved. By using a much smaller size commercial reconfigurable FPGA to reduce the unused FPGA areas, the redundant static power consumption can be significantly reduced. This has a significant impact on the values in relation to the dynamic power consumption presented herein. Hence, the dynamic low-power techniques such as the PR and CG will have a larger role to play in FPGA-based designs and worth the time for further development.

Table 6.1: Static and dynamic energy breakdown of 256-byte interrupt-based test program at 40 MHz.

Static energy (mJ)	198.06
Dynamic energy (mJ)	81.44
Static-to-dynamic energy ratio	2.5:1

In future, there is a possibility for RISC32-LP implementing more instructions to support more features. Due to this, the information (e.g., clock cycle needed for new instructions to complete in RISC32-LP multi-cycle mode) is needed to update in ERPA to ensure ERPA functionality. To improve the ERPA, we could explore more task scheduling algorithm and implement a usable algorithm on ERPA to achieve better energy consumption.

REFERENCES

- Peng, S. Y., Huang, T. C., Lee, Y. H., Chiu, C. C., Chen, K. H., Lin, Y. H., ... & Yang, C. C. (2013). Instruction-cycle-based dynamic voltage scaling power management for low-power digital signal processor with 53% power savings. *IEEE Journal of Solid-State Circuits*, 48(11), 2649-2661.
- Sterpone, L., Carro, L., Matos, D., Wong, S., & Fakhar, F. (2011, March). A new reconfigurable clock-gating technique for low power SRAM-based FPGAs. In *2011 Design, Automation & Test in Europe* (pp. 1-6). IEEE.
- Nunez-Yanez, J. L., Hosseinabady, M., & Beldachi, A. (2015). Energy optimization in commercial FPGAs with voltage, frequency and logic scaling. *IEEE Transactions on Computers*, 65(5), 1484-1493
- Nunez-Yanez, J. (2017). Adaptive voltage scaling in a heterogeneous FPGA device with memory and logic in-situ detectors. *Microprocessors and Microsystems*, 51, 227-238.
- Wu, Y., Thomson, S., Sun, H., Krause, D., Yu, S., & Kurio, G. (2014). Free razor: A novel voltage scaling low-power technique for large SoC designs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(11), 2431-2437.
- Kiat, W. P., Mok, K. M., Lee, W. K., Goh, H. G., & Achar, R. (2020). An energy efficient FPGA partial reconfiguration based micro-architectural technique for IoT applications. *Microprocessors and Microsystems*, 73, 102966.
- Hempstead, M., Lyons, M. J., Brooks, D., & Wei, G. Y. (2008). Survey of hardware systems for wireless sensor networks. *Journal of Low Power Electronics*, 4(1), 11-20.
- Bsoul, A. A., Wilton, S. J., Tsoi, K. H., & Luk, W. (2015). An FPGA architecture and CAD flow supporting dynamically controlled power gating. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(1), 178-191.
- Hosseinabady, M., & Nunez-Yanez, J. L. (2014, September). Run-time power gating in hybrid ARM-FPGA devices. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)* (pp. 1-6). IEEE.
- Tamimi, S., Ebrahimi, Z., Khaleghi, B., & Asadi, H. (2018). An efficient SRAM-based reconfigurable architecture for embedded processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(3), 466-479.
- Qin, Y., Zeng, G., Kurachi, R., Li, Y., Matsubara, Y., & Takada, H. (2019). Energy-efficient intra-task dvfs scheduling using linear programming formulation. *IEEE Access*, 7, 30536-30547.

- Tatematsu, T., Takase, H., Zeng, G., Tomiyama, H., & Takada, H. (2011, January). Checkpoint extraction using execution traces for intra-task dvfs in embedded systems. In *2011 Sixth IEEE International Symposium on Electronic Design, Test and Application* (pp. 19-24). IEEE.
- Huang, K., Jiang, X., Zhang, X., Yan, R., Wang, K., Xiong, D., & Yan, X. (2018). Energy-efficient fault-tolerant mapping and scheduling on heterogeneous multiprocessor real-time systems. *IEEE Access*, *6*, 57614-57630.
- Deng, Z., Yan, Z., Huang, H., & Shen, H. (2020). Energy-aware task scheduling on heterogeneous computing systems with time constraint. *IEEE Access*, *8*, 23936-23950.
- Wu, T., Liu, Y., Zhang, D., Li, J., Hu, X. S., Xue, C. J., & Yang, H. (2017). Dvfs-based long-term task scheduling for dual-channel solar-powered sensor nodes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *25*(11), 2981-2994.
- See, J. C., Mok, K. M., Lee, W. K., & Goh, H. G. (2020). RISC32-E: Field programmable gate array based sensor node with queue system to support fast encryption in Industrial Internet of Things applications. *International Journal of Circuit Theory and Applications*, *48*(8), 1209-1226.
- Brandalero, M. (2019). MuTARe: a multi-target, adaptive reconfigurable architecture.
- Datasheet, Xilinx. (2015). Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics v1. 18.
- See, J. C., Lee, W. K., Mok, K. M., & Goh, H. G. (2017, November). Development of LLVM compilation toolchain for IoT processor targeting wireless measurement applications. In *2017 IEEE 4th International Conference on Smart Instrumentation, Measurement and Application (ICSIMA)* (pp. 1-4). IEEE.
- Liew, S. Y., Tan, C. K., Gan, M. L., & Goh, H. G. (2018). A fast, adaptive, and energy-efficient data collection protocol in multi-channel-multi-path wireless sensor networks. *IEEE Computational Intelligence Magazine*, *13*(1), 30-40.
- Kolluri, S. (2015). Leveraging Power Leadership at 28 nm with Xilinx 7 Series FPGAs Process Technology. *vol*, *436*, 1-16.
- MIPS32 (2000). The MIPS32 Instruction Set. In: *MIPS32 Architecture for Programmers Volume II: The MIPS32 Instruction Set*. (pp. 21 - 29) MIPS Technologies, Inc.