THE DESIGN OF AN FPGA-BASED PROCESSOR WITH RECONFIGURABLE PROCESSOR EXECUTION STRUCTURE FOR INTERNET OF THINGS (IoT) APPLICATIONS

KIAT WEI PAU

MASTER OF SCIENCE (COMPUTER SCIENCE)

FACULTYOF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN
DECEMBER 2018

THE DESIGN OF AN FPGA-BASED PROCESSOR WITH RECONFIGURABLE PROCESSOR EXECUTION STRUCTURE FOR INTERNET OF THINGS (IoT) APPLICATIONS

By

KIAT WEI PAU

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)
December 2018

ABSTRACT

THE DESIGN OF AN FPGA-BASED PROCESSOR WITH RECONFIGURABLE MICROARCHITECTURE PROCESSOR EXECUTION STRUCTURE FOR INTERNET OF THINGS (IoT) APPLICATIONS

Kiat Wei Pau

Low power consumption and high computational performance are two important processor design goals for IoT applications. Achieving both design goals in one processor architecture is challenging due to their conflicting nature, whereby low power consumption tends to limit the computational performance and high computational performance tends to consume higher power. This research work introduces a micro-architectural level reconfigurable technique that allows a Reduced Instruction Set Computing (RISC) processor to support IoT applications with different performance power trade-off requirements. The processor can be reconfigured into either multi-cycle execution (low computational speed with low dynamic power consumption) or pipeline execution (high computational speed at the expense of high dynamic power usage), based on dynamic workload characteristics in IoT applications. The switching is made possible through partial reconfiguration (PR) feature offered by FPGAs. A RISC processor was designed based on the proposed micro-architectural level technique and

implemented on FPGA as IoT sensor node. Experimental result demonstrates that the proposed technique is able to reduce dynamic energy consumption by 4.63% and 21.47%, respectively, compared to multi-cycle and pipeline only microarchitecture. In order to improve the dynamic energy consumption without losing too much of computational performance, the energy-delay product metric is used. Our proposed technique shows that the energy-delay product is reduced by 8.81% (compared to multi-cycle) and 18.91% (compared to pipeline) respectively. This implies that the proposed technique can achieve better performance-energy trade-off for IoT applications compared to conventional method that only have single microarchitecture.

ACKNOWLEDGEMENTS

I would like to give a very deep appreciation to my supervisors, Dr. Goh Hock Guan and Mr. Mok Kai Ming, for the guidance, inspiration and enthusiasm that bring towards the completion of the 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 the completion of the experimental work. Last but not least, I like to thank to my family for their full support in order for me to pursue my interest.

APPROVAL SHEET

This dissertation entitled "**THE DESIGN OF AN FPGA-BASED PROCESSOR WITH RECONFIGURABLE MICROARCHITECTURE PROCESSOR EXECUTION STRUCTURE FOR INTERNET OF THINGS (IoT) APPLICATIONS"** was prepared by KIAT WEI PAU 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. Goh Hock Guan)
Date:
Supervisor
Department of Computer and Communication Technology
Faculty of Information and Communication Technology
Universiti Tunku Abdul Rahman

_____
(Mr. Mok Kai Ming)
Date:
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: _____

SUBMISSION OF DISSERTATION

It is hereby certified that **KIAT WEI PAU** (ID No: **16ACM01206**) has completed this dissertation entitled **"THE DESIGN OF AN FPGA-BASED PROCESSOR WITH RECONFIGURABLE MICROARCHITECTURE PROCESSOR EXECUTION STRUCTURE FOR INTERNET OF THINGS (IoT) APPLICATIONS "** under the supervision of **Dr. Goh Hock Guan** (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,

_____

(**KIAT WEI PAU**)

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: KIAT WEI PAU

Date:

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ADC	Analog-to-Digital Converter
AES-128	Advanced Encryption Standard – 128 bytes
ASIC	Application Specific Integrated Circuit
BCH	Bose–Chaudhuri–Hocquenghem
BRAM	Block RAM
CISC	Complex Instruction Sets Computing
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DFF	D Flip-flop
DFS	Fynamic Frequency Scaling
DLL	Delay-Locked Loop
DVFS	Dynamic Voltage and Frequency Scaling
DVS	Dynamic Voltage Scaling
ED	Event Detection
FFT	Fast Fourier Transform
FPGA	Field-Programmable Gate Array
GPIO	General-Purpose Input/Output
HDL	Hardware Description Language
I^2C	Inter-Integrated Circuit
ICAP	Internal Configuration Access Port
IoT	Internet of Things
IP	Internet Protocol
ISA	Instruction Set Architecture
I/O	Input/Output

MIPS	Microprocessor without Interlocked Pipeline Stages
NRE	Non-Recurring Engineering
PLL	Phase-Locked loop
PR	Partial Reconfiguration
RAM	Random Access Memory
RFU	Reconfigurable Function Unit
RISC	Reduced Instruction Set Computing
RTL	Register-Transfer Level
SoC	System-on-Chip
SPE	Spatial Process Estimation
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver-Transmitter
VHDL	Very High-speed Integrated Circuit Hardware Description Language
WSN	Wireless Sensor Network
XADC	Xilinx Analog-to-Digital Converter

CHAPTER 1

INTRODUCTION

1.1 Background

Internet of Things (IoT) enable communication of a wide range of physical objects without human intervention (Lazarescu, M. T., 2013), and nowadays, sensors can be deployed everywhere. Sensor data can be accessed at any time using a remote device, i.e. smartphone, computer etc. The emerging of larger addressing space, i.e. Internet Protocol version 6 (IPv6), allows each sensor node to have a unique Internet Protocol (IP) address and directly access through the Internet. As a result, the physical objects are able "to see, hear, think and perform jobs by having them 'talk' together, to share information and to coordinate decisions." (Al-fuqaha, A. et al., 2015)

IoT, which is evolved from Wireless Sensor Network (WSN), has the advantages of dynamic network size, low devices cost, self-organize without human intervention, querying data and re-tasking capabilities, multihop data aggregation, and multi-environment deployment (Bhattacharyya, D., Kim, T. and Pal, S., 2010; Gungor, V. C., Lu, B. and Hancke, G. P., 2010). WSN consists of a group of sensor nodes. Each sensor node is responsible to collect ambient environmental data, pre-process the data and transmit the data to neighbouring nodes or sink nodes (Akyildiz, I. F. et al., 2002; Stankovic, J. A., 2008). The basic components of a sensor node consist of a sensing unit, processing unit, transceiver unit and power unit. Sensing unit composes of

sensor(s) (can be a module form) where sensor data can be collected through I^2C (Inter-integrated Circuit), SPI (Serial Peripheral Interface), UART (Universal Asynchronous Receiver-Transmitter), GPIO (General-Purpose Input/Output), ADC (analog-to-digital converter), etc. Sensor senses the ambient environment data and the collected data will be forwarded to the processing unit. The processing unit consists of processor and memory units, which used for data processing and storing respectively. Lastly, the transceiver unit is responsible to send the processed data to neighbouring nodes or sink nodes. A power unit is used as the power source of the sensor node. The power source can be from a battery, harvesting unit (collected from renewable energy, e.g. vibrations, solar, heat or electromagnetic energy) or power supply. The role of each node is different depends on the processing capabilities and themselves take on specific functions and behaviors in the network (Kateeb, A. El, Ramesh, A. and Azzawi, L., 2008). A common WSN consists of 2 types of nodes, sensor nodes and sink nodes (Akyildiz, I. F. et al., 2002). Sensor nodes are capable to collect, process sensor data and transmit the data to another sensor node or sink node via wireless (can be Bluetooth, Zigbee etc.), while sink node has additional capability to forward the data to other networks, i.e. Internet or Cellular networks (Buratti, C. et al., 2009). Figure 1.1 shows the WSN architecture.

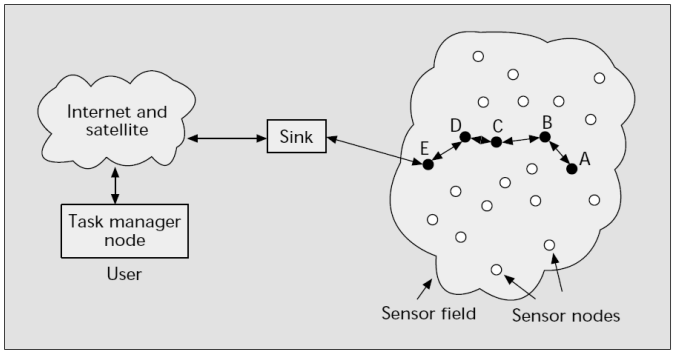


Figure 1.1: WSN architecture
Source: Akyildiz, I. F. et al. (2002) 'A survey on sensor networks', IEEE Communications Magazine, 40(8), pp. 102–114. doi: 10.1109/MCOM.2002.1024422.

For the on-field IoT application, a stringent need for low power is the fundamental requirement. "Low power design is an important topic of wireless sensor network" (Yongjun Xu et al., 2005). The main challenge of WSN is to reduce the power consumption of the sensor node (Jawhar, I., Mohamed, N. and Agrawal, D. P., 2011). From a survey conducted by de la Piedra, A. et al. (2013), most of the IoT deployments require the sensor nodes to operate at least for a few months. To achieve this requirement, the sensor nodes have to be operated in low power mode to minimize the energy consumption. However, reducing the power consumption usually will tend to reduce the performance as well, as the common approach is by reducing the clock frequency. Choi, K., Soma, R. and Pedram, M. (2004) demonstrated energy saving by reducing the clock frequency and voltage, which resulted in 10 - 30% performance loss for CPU-bound applications (bf, crc, djpeg and math) and 10 - 20% performance loss for memory-bound applications (qsort and gzip). Processor with a fixed microarchitecture can cause oversupply of computational speed for processing low computational requirement IoT tasks

and thus, energy is wasted. Furthermore, the operation at low computational speed is able to save power, however it may not process high computational requirement IoT tasks in certain period. Pande, V., Elmannai, W. and Elleithy, K. (2013), Lloret, J. et al. (2009) and Xufeng Wei et al. (2014) showed a fire detection application using temperature and image sensors on a high computational speed processor in WSN. Image sensor was set to sleep mode (Pande, V., Elmannai, W. and Elleithy, K., 2013; Lloret, J. et al., 2009) or with longer sampling interval (Xufeng Wei, Yahui Wang and Yanliang Dong, 2014) for power saving purpose, but temperature sensor has shorter sampling interval. Since temperature sensor is still monitoring the environment frequently, when it detects a rapid increase in temperature, the image sensor is turned to active mode to further verify on such event triggered. In this case, the power consumption can still be reduced, since low computational speed is required to collect temperature sensor data, whereas high computational speed is required on demand.

Violante, M. et al. (2011) had stated that hard macro or hard-core processors, i.e. commercialize off-the-shelf microcontroller chip, for example, ATmega128L inside the MICAz mote, is neither configurable nor modifiable by end user. Slight modification to be made in the manufacturing process could end up costing millions. On the other hand, the soft IP core offers some degrees of customization, which determined the functionalities and peripherals that should be included in the design. This has made a valid issue when de la Piedra, A. et al. (2013) and Qingping Chi et al. (2014) presented that lack of standardized I/O peripherals interface for wireless sensor node as one of the

open issues or limitations for the sensor nodes. The I/O peripherals are used as the communication path between the external chip modules with the processing unit inside the sensor node. Since external chip modules are not always designed with either SPI or I^2C interface (de la Piedra, A. et al., 2013), it will be a limitation when the off-the-shelf microcontroller does not provide a sufficient number of interfaces, for example, off-the-shelf microcontroller provides only UART interface while transceiver module is designed with SPI interface. While struggling with this issue, Johnson, D. (2009) presented a solution by using only the digital GPIO port to imitate the SPI, I^2C and UART communication protocols, and thus solve the unstandardized I/O peripherals issue. However, referring to the experimental result shown in (Mikhaylov, K. and Tervonen, J., 2012), this solution consumes more energy and has lower performance than the real hardware interface protocols. Apart from that, Mikhaylov, K. and Tervonen, J. (2012) also showed that the power consumption of SPI is far lower than UART and I^2C where UART is lower than I^2C. Besides that, Qingping Chi et al. (2014) had also pointed out that the applications are limited by the fixed hardware design and there is still no "one size fits all" kind of solution. Hsieh, C.-M. et al. (2014) on the other hand had experimented the Fast Fourier transform (FFT) function for both software and hardware methods. The result shows that software method consumes 21% more current than the hardware method. Inherently, it is a limitation if an off-the-shelf microcontroller is used, i.e. ATmega128L, since the hardware accelerator is not able to customize or include into the microcontroller.

In our research work, we are motivated to develop a reconfigurable soft-core processor on Field-Programmable Gate Array (FPGA) for the on-field Internet of Things (IoT) application. The processor is developed to be customizable and capable in switching between multi-cycle (to process low computational speed requirement tasks while saving power) and pipeline (to process high computational speed requirement task but consume more power) microarchitectures to satisfy better performance-power tradeoff. Our research has carried out on the processor microarchitecture level, which by experiment the reconfiguration between multi-cycle and pipeline executions. Multi-cycle execution is able to reduce the dynamic power consumption of the processor at the expense of providing lower computational speed. In opposite, pipeline execution provides higher computational speed but consume more dynamic power than multi-cycle execution. The processor is implemented based on FPGA technology, in which FPGA technology provides a key enabling feature for our experiment, the partial reconfiguration (PR) feature. FPGA PR feature allows only reconfiguring a small region, i.e. multi-cycle and pipeline executions, without reconfiguring the whole FPGA chip.

1.2 Problem Statement

A deployed IoT sensor node is expected to perform data aggregation, data processing and data transmission, which require different computational speeds and power consumption. Low power consumption is the fundamental requirement for deploying IoT application because changing device's battery is a difficult task after the sensor nodes were deployed. Various power reduction techniques, refer to Section 2.3, have been proposed to develop energy efficient sensor nodes for IoT deployment, but sacrifice the computational performance. The techniques mentioned were implemented at gate-level or board-level to manipulate the voltage and clock frequency on a fixed microarchitecture processor. Achieving low-power by manipulating the micro-architectural design is, however, has not been well addressed. Reconfigurable microarchitecture of processor offers a new low power technique to be used in IoT sensor nodes. Interestingly, the design of such processor was also accompanied by the following questions: (1) How to tune the processor based on the computational needs from the environment requirement to have the optimum power saving scheme? (2) How to verify the performance of the design in terms of computational speed and power using conventional FPGA chip? Therefore, there is a need to perform a systematic research on the design of an energy efficient processor with reconfigurable microarchitecture for IoT applications.

1.3 Objectives

The main goal of this research is to develop a reconfigurable soft-core processor on FPGA for the on-field IoT application. The developed IoT processor is capable to collect, process and transmit the sensor data to another sensor node. The developed IoT processor is also able to adjust at micro-architectural level, the required computational speed to suit each IoT application and at the same time save power. More specifically, the objective can be further divided into the following sub-objectives:

1) To develop a reconfigurable soft-core IoT processor with essential I/O interfaces (SPI, UART and GPIO) and memory system for on-field IoT application. This work includes the development of a suitable CPU structure, I/Os and firmware, bus system and arbitration, volatile and non-volatile memory controller and memory system arbitration.
2) To develop the microarchitecture that is able to perform PR between multi-cycle and pipeline microarchitectures to satisfy the varying performance-power tradeoff requirements from each IoT application. The developed processor should be able to partial reconfigure itself between multi-cycle and pipeline microarchitectures. This work includes the determination of the CPU components involving in the PR and the development of the PR system.
3) To synthesize the developed processor on a conventional Xilinx Artix-7 XC7A100T FPGA chip. The computational speed and power analysis for pipeline and multi-cycle microarchitectures based on AES-128 encryption will be experimented to identify the performance of the developed processor.

1.4 Contributions

The contributions of this dissertation are:

1) A customizable IoT processor that is able to cope with rapidly changing research functional needs required in IoT. Since the research and development of IoT applications are constantly developing, where extra functionalities may be introduced in the future, customizable offers competitive advantages by shorten the development cycle, lower the development cost and lower manufacturing turn-around time.
2) A reconfigurable soft-core IoT processor to satisfy the varying performance-power tradeoff requirements from each IoT application by PR between multi-cycle and pipeline executions. Multi-cycle execution is used to reduce the dynamic power consumption of the processor at the expense of providing lower computational speed, while pipeline execution provides higher computational speed but consume more dynamic power than multi-cycle execution.
3) An experiment result that highlights the quantitative differences between multi-cycle and pipeline executions. The analysis on computational speed and power consumption for both multi-cycle and pipeline executions are gathered to highlight the strength of each execution.

1.5 Dissertation Organization

This dissertation is organized as follows. Chapter 2 discusses the necessary information prior to conduct our research. Chapter 3 describes the reconfigurable IoT processor developed. Chapter 4 presents the verification flow and compares the computational speed analysis and power analysis for both pipeline and multi-cycle microarchitecture. Chapter 5 concludes the dissertation and provides suggestion for the future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Internet of Things (IoT)

2.1.1 IoT Application

Buratti, C. et al. (2009) had classified the IoT application into 2 categories, namely event detection (ED) and spatial process estimation (SPE). The ED application sensors are deployed to detect an event while SPE application aims to estimate a given physical phenomenon, i.e. estimation of the entire behavior of the spatial process based on the samples taken by the sensor nodes. Borges, L. M. et al. (2014) had further expanded these categories according to the applications area and its applications. Table 2.1 shows the characterization of the applications by Borges, L. M. et al. (2014), with ED represents the event detection and PE represents the process estimation (PE = SPE).

Table 2.1: Characterization of the applications
Source: Borges, L. M., Velez, F. J. and Lebres, A. S. (2014) 'Survey on the Characterization and Classification of Wireless Sensor Network Applications', IEEE Communications Surveys & Tutorials, 16(4), pp. 1860–1890. doi: 10.1109/COMST.2014.2320073.

Applications Area	Applications	Class
Metropolitan Operation	-Highway monitoring [82] [83]	PE
Military	-Condition-based-monitoring [84] [85] [86] [87]	PE
	-Surveillance [88]	ED
	-Borders Monitoring [89]	ED
Civil Engineering	-Structural Integrity Monitoring [90]	ED
Environmental Monitoring	-Monitoring Volcanic Eruptions [91], [92]	PE & ED
	-Habitat Monitoring [93]	PE & ED
	-Water Monitoring [94] [95] [96]	PE & ED
	-Weather Monitoring [97] [98]	PE & ED
	-Forest Fire Detection [99] [100]	PE & ED
	-Precision Agriculture [101] [102]	PE & ED
Logistics	-Target Tracking [103] [104]	ED
	-Warehouse Tracking [105]	PE & ED
Position & Animals Tracking	-Immersive Roam [106]	ED
	-Real-Time Relative Positioning System [107]	ED
	-Wild-Life Tracking [108] [109] [110]	ED
Transportation	-Smart Roads [111] [112]	PE
-Automobile [113]		PE
-Sensor & Robots [114] [115]		PE & ED
-Reconfigurable WSN [116]		ED
-Nanoscopic Sensors [117]		ED

Applications Area	Applications	Class
Industrial Automation	-Commercial Spaces	PE & ED
	-Smart Factory [118]	PE & ED
Health	-Pre-Hospital [119]	PE & ED
	-In-Hospital Emergency Care [120]	PE & ED
	-Telemedicine [121]	PE & ED
	-(Tele)Rehabilitation [122]	PE & ED
Mood-based Services	-Personal Coaching [123]	ED
Entertainment	-Dynamic Spaces [124]	ED
	-Gaming [125]	ED
	-Gesture/body Tracking [126]	ED
-Smart Office [127]		ED
-Sports [128]		ED
-Building Automation [129]		ED
-Home Control [130]		ED

However, the information shown in Table 2.1 is inefficient to identify the computational requirement for each IoT application. Borges, L. M. et al. (2014) and Hempstead, M. et al. (2008) had classified the sampling rates of the sensor nodes into 3 ranges, which are low sampling rate varies between 0.001 Hz and 100 Hz, medium sampling rate varies between 100 Hz and 1 kHz, and high sampling rate which is higher than 1 kHz. Hempstead, M. et al. (2008) had pointed out that the computational requirement is defined by the sampling rate for the measured phenomena and the amount of on-node data filtering required. High performance processor is required to measure and process high sampling data rate of the sensor node, while low sampling data rate sensor node will be idle most of the time. Table 2.2 shows the sensors sampling rate used in different phenomena identified by Hempstead, M. et al. (2008).

Table 2.2: Sensors sampling rate
Source: Hempstead, M. et al. (2008) 'Survey of Hardware Systems for Wireless Sensor Networks', Journal of Low Power Electronics, 4(1), pp. 11–20. doi: 10.1166/jolpe.2008.156.

Phenomena	Sample rate (in Hz)
Very low frequency	
Atmospheric temperature	0.017–1
Barometric pressure	0.017–1
Low frequency	
Heart rate	0.8–3.2
Volcanic infrasound	20–80
Natural seismic vibration	0.2–100
Mid frequency (100 Hz–1000 Hz)	
Earthquake vibrations	100–160 Hz
ECG (heart electrical activity)	100–250
High frequency (>1 kHz)	
Breathing sounds	100–5 k
Industrial vibrations	40 k
Audio (human hearing range)	15–44 k
Audio (muzzle shock-wave)	1 M
Video (digital television)	10 M

Furthermore, Hempstead, M. et al. (2008) further described the desired lifetimes and the computational requirements in each application domain, which is shown in Table 2.3.

Table 2.3: Applications lifetime and computation requirement
Source: Hempstead, M. et al. (2008) 'Survey of Hardware Systems for Wireless Sensor Networks', Journal of Low Power Electronics, 4(1), pp. 11–20. doi: 10.1166/jolpe.2008.156.

Application domain	Desired lifetimes	Computation requirements (Sample rates)
Scientific applications		
Habitat/weather monitoring	Months/decades	very low
Volcanic eruption detection	Months/decades	mid
Military and security applications		
Building/border intrusion detection	Years/decades	low
Structural and earthquake monitoring	Years/decade	low/mid
Active battlefield sensing	Months	mid/high
Medical applications		
Long-term health monitoring (pulse)	Days	low
Untethered medical instruments (ECG)	Days	med
Business applications		
Supply chain management	Months	low
Expired/damaged goods tracking	Months	low
Factory/fab monitoring	Months/years	med/high

Majority of the applications in Table 2.3 require the lifetime of the sensor node to last for at least a few months. The research work conducted by Hempstead, M. et al. (2008) is useful in identifying the lifetime and the computation requirement of the sensor node, especially the targeted application, environmental monitoring application, which would requires low to medium computational speed and expected to last for several months.

In summary, we see an opportunity to save power consumption or provide higher computational speed based on the need of an application as indicated by the sensors sampling rate. For example, for sampling rates between 0.001 Hz to 100 Hz which imply a low-speed processing, a multi-cycle structure can be used to reduce the power consumption. If higher sampling rates (more than 1 kHz) are required by an application, then a pipeline structure can be employed.

2.1.2 Existing IoT Platforms

Borges, L. M. et al. (2014) had presented the sensor node platform used in each IoT application area, which is shown in Table 2.4.

Table 2.4: Application device's current consumption
Source: Borges, L. M., Velez, F. J. and Lebres, A. S. (2014) 'Survey on the Characterization and Classification of Wireless Sensor Network Applications', IEEE Communications Surveys & Tutorials, 16(4), pp. 1860–1890. doi: 10.1109/COMST.2014.2320073.

	Application Area	Metropolitan	Military	Environmental monitoring	Position & animals tracking	Industrial automation	Health
	Project	Traffic Dot [82]	WiBeaM [84]	Reventador volcano [91], [92]	LEMe Room [106]	Anshan [118]	MEDiSN [120]
	Application	Highway monitoring	Condition-based monitoring	Monitoring volcanic eruptions	Immersive roam	Smart factory	Telemedicine
Communication & Traffic	QoS	Collective QoS (Event-Driven)	Collective QoS (Event-Driven)	Collective QoS (Event-Driven)	Collective QoS (Event-Driven)	Collective QoS (Event-Driven & Query-driven)	Collective QoS (Event-Driven)
	Bit rate (kbps)	$\leq$ 57.6	$\leq$ 57.6	$\leq$ 57.6	$\leq$ 57.6	$\leq$ 57.6	$\geq$ 115.2
	Latency	T_{min}= 1 s T_{max}= 2 s	T_{max}= 0.00867 s (per packet)	T_{max}= 63 s (per hop)	T_{max}= 3 ms	T_{min}= 80 ms T_{max}= 250 ms	T_{max}= 7-8 ms
	Synchronization	Sync	Sync	Sync	Sync	Sync	Sync
	Class of service	ISO&CBR	ISO&CBR	ISO&CBR	ISO&RT-VBR	ISO&CBR	ISO&RT-VBR
	Traffic classes	RT&LT&Data	DT&LT&Data	DT&LT&Data	RT&LI&Data	DT&LI&Data	RT&LI&Data
	Modulation	FSK(DSSS)	DSSS-O-QPSK	FSK	FSK(DSSS)	GFSK or MSK/OOK	FSK
	Communication direction	Half-duplex	Half-duplex	Half-duplex	Half-duplex	Half-duplex	Half-duplex
Serv. components	Type of Traffic	STV; LOD MED; HID	LOD	HID	HID	HID	STV; HID
	Packet delivery failure ratio	1% (max.)	9%	4.96%	N.A.	$\leq$10%	6.3%
	Acquisition & dissemination classes	Event-driven	Event-driven	Demand-driven & event-driven	Demand-driven & event-driven	Event-driven & time-driven	Demand-driven & time-driven
Network	Lifetime (h)	$\geq$ 25920	[721; 25920]	[24; 720]	[24; 720]	$\geq$ 25920	N.A.
	Scalability	[51; 400] nodes	$\leq$50 nodes	$\leq$50 nodes	$\leq$50 nodes	$>$ 400 nodes	$\leq$50 nodes
	Density	N.A.	N.A.	3 km^2	450 m^2	3000 m^2	N.A.
	Sensing range	N.A.	Local	Local	2 m^2	$<$ 1 m^2	local
	Self-organization	Entire network	Entire network	Entire network	Entire network	Entire network	Entire network
	Security	Low	Low or none	Low	Medium	None	None
	Addressing	Address-centric: MAC address	Address-centric: MAC address	Address-centric: MAC address	Address-centric: node ID	Address-centric: node ID, group ID, cluster ID	Address-centric: address number
	Programmability	Not programmable	Not programmable	Not programmable	Programmable	Not programmable	Not programmable
	Maintainability	Maintainable	Maintainable	Maintainable	Not maintainable	Maintainable	Not maintainable
	Homogeneity	Heterogeneous	Heterogeneous	Heterogeneous	Heterogeneous	Heterogeneous	Heterogeneous
	Mobility support	No support	No support	No support	Support	No support	Support
Node	Microprocessor	ATMEL (ATMega 128L)	TI (MSP430F1611)	ATMEL (ATMega 128L)	ATMEL & Microchip (ATMega 128L) & (PIC16F872)	ATMEL (ATMega 128L)	TI (MSP430F1611)
	Transceiver	CC1100	CC2420	CC1000 (433 MHz)	CC1000 (900 MHz)	CC1100 (433 MHz)	CC2420
	Overall energy consumption	0.01044 A	0.0612 A	0.038 A	0.0520 A	0.072503 A	0.040955 A
	Sampling rate	[100; 1000] Hz	$>$1 kHz	[100; 1000] Hz	$>$1 kHz	[100; 1000] Hz	[0.001; 100] Hz
	Type of function	Sensor & sink	Sensor & sink	Sensor & sink	Sensor & sink	Sensor & sink + gateway	Sensor & sink
	Communication range	5.5 m	10 m	10.7 m	305 m	$<$ 100 m	$>$ 5 m
	Power supply	Battery	Battery	Battery	Battery & local supply	Battery	Battery

Continued from Table 2.4: Application device's current consumption

	Application Area	Civil engineering	Environmental monitoring	Logistics	Transportation	Automobile	Sports
	Project	Sustainable Bridges [90]	FireBug [99], [100]	VigilNet [103] Trio [104]	AstroRoads [111], [112]	Intelligent Tires [113]	DexterNet [128]
	Application	Structural integrity monitoring	Forest fire detection	Target tracking	Smart roads	Automobile	Sports
Communication & Traffic	QoS	Collective QoS (Event-Driven)	Collective QoS (Event-Driven & Query-driven)	Collective QoS (Event-Driven)	Collective QoS (continuous) individual specific	Hybrid model	Collective QoS (Event-Driven)
	Bit rate (kbps)	$\leq$ 57.6	[57.6; 115.2]	$\geq$ 115.2	$\leq$ 57.6	$\leq$ 57.6	$\geq$ 115.2
	Latency	T_{min}= 400 ms T_{max}= 1500 ms	T_{min}= - T_{max}= 2 s	T_{min}= - T_{max}= -	T_{min}= - T_{max}= 130 s	T_{min}= - T_{max}= 2.9 s	T_{min}= - T_{max}= -
	Synchronization	Async	Sync	Sync	Sync	Sync	Sync and Async
	Class of service	ISO&RT-VBR	ISO&CBR	ISO&RT-VBR	ISO&RT-VBR	ISO&CBR	ISO&RT-VBR
	Traffic classes	RT&LT&Data	DT&LI&Data	RT&LI&Data	RT&LT&Data	DT&LI&Data	RT&LT&Data
	Modulation	FSK(DSSS)	FSK	DSSS-O-QPSK	FSK(DSSS)	FSK(DSSS)	DSSS-O-QPSK
	Communication direction	Half-duplex	Half-duplex	Half-duplex	Half-duplex	Half-duplex	Half-duplex
Serv. components	Type of Traffic	HID	LOD	LOD	HID	LOD	HID
	Packet delivery failure ratio	-	-	1%	0.1%	4%	-
	Acquisition & dissemination classes	Event-driven & Time-driven	Event-driven & Time-driven	Event-driven	Event-driven & event-driven	Event-driven & & time-driven	Event-driven
Network	Lifetime (h)	$\geq$ 25920	[24; 720]	[721; 25920]	N.A.	[24; 720]	N.A.
	Scalability	N.A.	$\leq$ 50 nodes	$>$ 400 nodes	$\leq$ 50 nodes	$\leq$ 50 nodes	$\leq$ 50 nodes
	Density	-	3 km^2	50 km^2	N.A.	0.17 m^2	16 m^2
	Sensing range	10-52 m^2	Local	200 m^2	N.A.	$<$ 1 m^2	$<$ 1 m^2
	Self-organization	Entire network	Entire network	Entire network	Entire network	Entire network	Entire network
	Security	Low	Low	None	None	None	None
	Addressing	Address-centric: MAC address	Address-centric: MAC address	Address-centric: MAC address	Address-centric: node ID	Geographic addressing scheme	Address-centric: node ID
	Programmability	Programmable	Not programmable	Programmable	Programmable	Not programmable	Programmable
	Maintainability	Maintainable	Maintainable	Maintainable	Maintainable	Maintainable	Maintainable
	Homogeneity	Heterogeneous	Heterogeneous	Heterogeneous	Heterogeneous	Heterogeneous	Heterogeneous
	Mobility support	No support	No support	No support	No support	Support	Support
Node	Microprocessor	TI (DSP 8-bit)	ATMEL (ATMega 128L)	TI (MSP430)	ATMEL & Microchip (ATMega 128L)	ATMEL (ATMega 128L)	ATMEL (ATMega 8)
	Transceiver	CC2420	CC1000	CC2420	CC1000	CC1000	CC2420
	Overall energy consumption	0.053 A	0.036 A	0.0351 A	0.045 A	0.0346 A	0.0289 A
	Sampling rate	$>$ 1 kHz	[100; 1000] Hz	[0.001; 100] Hz	N.A.	N.A.	[0.001; 100] Hz
	Type of function	Sensor & sink	Sensor & sink	Sensor & sink + gateway	Sensor & sink + actuator	Sensor & sink	Sensor & sink + gateway
	Communication range	10-30 m	10.7 m	125 m	305 m	140 m	10 m
	Power supply	Battery	Battery	Battery Solar panel	Battery	Battery	Battery

This study shows the sampling rate that supported by each sensor node platform. The sampling rate of the environmental monitoring application falls under medium sampling rate range, i.e. varies from 100 Hz to 1 kHz. ATMega128L microcontroller is used to implement the environmental monitoring application, with consumes 0.036A to 0.038A in overall energy consumption. Besides that, Hempstead, M. et al. (2008) presented the specification of the hardware system used in the WSN in Table 2.5.

Table 2.5: Hardware system for WSN
Source: Hempstead, M. et al. (2008) 'Survey of Hardware Systems for Wireless Sensor Networks', Journal of Low Power Electronics, 4(1), pp. 11–20. doi: 10.1166/jolpe.2008.156.

System	Arch style	Data path width	Event driven (y/n)	Circuit techniques	Accelerators	Memory (KB)	Process	Voltage (V)	Throughput (MIPS)	Energy (pJ/ins)
Atmel ATmega128L	GP Off-the-shelf	8	N	N	N	132 KB	350 nm	3.0 V	7.3 MHz	3200
TI MSP430	GP Off-the-shelf	16	N	N	N	10 KB	NA	3.0	8 MHz	750
SNAP/LE	GP RISC	16	Y	Asyncronous	Timer, message interface	8 KB	180 nm	1.8 0.6	200 23	218 24
BitSNAP	GP RISC. Bit-serial datapath	16	Y	Asyncronous	Timer, message interface	8 KB	180 nm	1.8 0.6	54 6	152 17
Smart Dust	GP RISC	8	N	Syncronous-two clocks	None	3.125 KB	250 nm	1.0	0.5 (500 kHz)	12
Charm	Protocol processor	NA	N	Two power domains	Custom radio stack	68 KB	130 nm	1.0 V (high) 0.3–1.0 V (low)	8 MHz	150 μW 53.6 μW leakage
Michigan 1	GP	8	Y	Subthreshold	None	0.25 KB	130 nm	0.360	833 kHz	2.6
Michigan 2	GP	8	Y	Subthreshold	None	0.3125	130 nm	0.350	354 kHz	3.52
Harvard	Event driven accelerator	8	Y	VDD-gating	Timer, filter, message proc	4 KB	130 nm	0.55–1.2	12.5 MHz	680 pJ/task

ATMega128L microcontroller is used by most of the IoT application area as shown in Table 2.4. However, based on the information shown in Table 2.5, the general purpose off-the-shelf microcontrollers (ATMega128L and TI MSP430) consume the most energy, in which it is not the best solution in implementing a low power IoT application sensor node. By investigate Table 2.5, the reasons for such high power usage is because of the memory size and the process technology (350 nm). Another supporting research work by Gajjar, S. et al. (2014), had made an analysis on the sensor node used in WSN, which shown in Table 2.6.

Table 2.6: Sensor node's analysis
Source: Gajjar, S. et al. (2014) 'Comparative analysis of wireless sensor network motes', in 2014 International Conference on Signal Processing and Integrated Networks (SPIN). IEEE, pp. 426–431. doi: 10.1109/SPIN.2014.6776991.

Parameter	TelosB/Tmote Sky	MICA2/MICAZ	SHIMMER	IRIS	SunSpot	eZ430-F2500T	Waspmotes
Controller	TI MSP430F1611	AT Atmega128L*	TI MSP430F1611	AT Atmega1281*	AT Atmega91RM 9200*	TI MSP430F2274	AT Atmega1281*
BUS size (Bits)	16	8	16	8	32	16	8
Frequency (MHz)	8	16	8	16	180	16	16
Wake-up time (µs)	6	180	6	4300	Pin change wake up	1	16000
FLASH (Bytes)	48K	128K	48K	640K	4M	32K	128K
RAM (Bytes)	10K	4K	10K	8K	512K	1K	8K
EEPROM (Bytes)	1M	512K	No support	4K	No Support	No support	4K
Serial Communication	UART	UART	UART	UART	UART	UART	UART
Current Active mode (mA)	1.8	8	1.8	8	25	0.270	8
Current Sleep (µA)	5.1	<15	5.1	8	500	0.7/0.1(Standby/off)	8
Operating voltage (V)	1.8 to 3.6	2.7 to 3.3	1.8 to 3.6	2.7 to 3.3	5(±10%)	1.8 to 3.6	2.7 to 3.3
Power consumption active (mW)	3	33	5.94	21.6	92.5	0.594	21.6
Power consumption sleep (µW)	2	30	16.83	21.6	1850	1.54/0.22(Standby/off)	21.6
Timer support	Two 16 bit	Two 8-bit Two 16-bit	Two 16 bit	Two 8-bit Four 16-bit	Two 16 bit	Two 16 bit	Two 8-bit Four 16-bit
Watchdog	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ADC	12-bit SAR	10-bit	12-bit SAR	10-bit	10-bit	10-bit SAR	10-bit
ADC channels	8	8	8	8	8	12	8
Operating temperature range (°C)	-40 to +85	-55 to +125	-40 to +85	-55 to +125	-40 to +85	-40 to +105	-55 to +125
Package	64-pin QFN	64-lead TQFP, 64-Pad QFN/MLF	64-pin QFN	64-pad QFN/MLF, 64-lead TQFP	208- PQFP, 256-ball BGA	38TSSOP, 40VQFN, 49DSBGA	64-Pad QFN/MLF, 64-lead TQFP
OS support	*Contiki, TOS, Mantis OS	*TOS, Mantis OS	*TOS	*Mote Runner, TOS, MoteWorks	Squawk VM (Java)	*TOS	N.A.
Programming and IDE	*TOS, CCS, IAR	*TOS	*TOS	*TOS	J2ME, JDK	*TOS, CCS, IAR	C++, Waspmotes

*TOS=Tiny OS Cross development tools with TOSSIM Simulator, AT=Atmel

From Table 2.6, TI MSP430 family series microcontroller consumes the lowest power. Based on the information provided by Texas Instruments (2006), MSP430 family series microcontrollers require multiple clock cycles to execute an instruction, i.e. multi-cycle execution. However, due to the nature of multi-cycle execution, it provides lower computational power as compared to the pipeline execution, which makes the pipeline execution popular in high performance processor design. So far, the discussed microcontrollers are manufactured using ASIC technology. It would be costly and requires longer development cycle to implement both the multi-cycle and pipeline execution using ASIC technology in order to gain the advantage from both design approaches. In the next subsection, we will discuss the benefits of the FPGA technology which can help to achieve this goal.

2.2 FPGA versus ASIC

Most of the soft-core processor design start in Register-Transfer Level (RTL) modeling since it is technology-independent and hence the design can be easily ported from FPGA to ASIC with only a few Hardware Description Language (HDL) code changes (Abid, F. and Izeboudjen, N., 2015a; Abid, F. and Izeboudjen, N., 2015b). In addition, HDL is at the center of modern digital design practices, in which the building blocks or the entire processor can be describe either in Very High-speed Integrated Circuit Hardware Description Language (VHDL) or Verilog, and the overall design is much easier to understand (Harris, D. M. and Harris, S. L., 2013; Tong, J. G., Anderson, I. D. L. and Khalid, M. A. S., 2006). However, when it comes to the selection of the implementation platform, there's always an argument between the 2 major technologies, ASIC or FPGA. FPGA is widely used in various designs and diverse target applications (Abid, F. and Izeboudjen, N., 2015a). It has the benefits of low manufacturing turn-around time, shorter the development cycle, reduce the time-to-market and decrease the Non-Recurring Engineering (NRE) cost. However, it comes with a price in higher power consumption, larger design area and longer circuit delay which reduce the design logics performance (Kuon, I. and Rose, J., 2007) and it only progressively used as the final product platforms for low volume production (Abid, F. and Izeboudjen, N., 2015a). For high volume production, ASIC is often chosen as the implementation technology (Abid, F. and Izeboudjen, N., 2015b). Its benefits are lower power consumption, smaller design area and higher design logics performance compare with FPGAs (Kuon, I. and Rose, J., 2007). However, longer development cycle which leads to delayed time-to-market,

higher NRE cost and high manufacturing turn-around time are the drawbacks of the ASIC implementation (Abid, F. and Izeboudjen, N., 2015b). For our project, FPGA is chosen as our core technology to implement the IoT soft-core processor, in order to take advantage in a shorter development cycle and highly customizable. Since IoT data processing requirements, sensors and data loggers interface and communications medium are not mature in implementation, it is expected that the functional changes to take place through the IoT processor development cycle (de la Piedra, A. et al., 2013). For example, iterative experimental work on processor micro-architectural level to achieve lower power consumption, adding or removing required IOs etc. As stated in the previous section, the switching between multi-cycle and pipeline executions is only possible with the help of partial reconfiguration feature offered by FPGA. Besides that, since FPGA is potential to port to ASIC in the future, we may identify the competitiveness of our design with the existing microprocessor or microcontroller, which mostly fabricated in ASIC. Kuon, I. and Rose, J. (2007) had examined that the FPGA is approximately 35 times larger design area than the ASIC with between 3.4 to 4.6 times slower and consumes 14 times more dynamic power. This statistical data will serve as a reference for us to estimate the design performance when ported to ASIC platform.

FPGA has been constructed in technologies ranging from 2.0 microns in 1985 down to 20 nanometers today (Shannon, L. et al., 2015). Shannon, L. et al. (2015) concluded that the FPGA technology has been closely following Moore's law, where the numbers of transistors on an integrated circuit will

double every two years. Table 2.7 shows the related information gathered by Shannon, L. et al. (2015).

Table 2.7: FPGA chip overall analysis
Source: Shannon, L. et al. (2015) 'Technology Scaling in FPGAs: Trends in Applications and Architectures', in 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines. IEEE, pp. 1–8. doi: 10.1109/FCCM.2015.11.

Year	Feature Size	Xilinx FPGA family		Device	LUTs	DSP/Mult blocks	BRAM Kbits	LUTs /DSP	LUTs /BRAM	Altera FPGA family		Device	ALMs (LEs)	DSP /Mult blocks	BRAM Kbits	LEs/ DSP	LEs/ BRAM
2011	28 nm	Virtex 7	V	XC7V2000T	1,221,600	2,160	46,512	566	26								
			VX	XC7VX1140T	712,000	3,600	67,680	198	11								
			VH	XC7VH870T	547,600	2,520	50,760	217	11								
2010										Stratix V	GT	5SGTC7	622,000	512	50,000	1,215	12
											GX	5SGXBB	952,000	704	52,000	1,352	18
											GS	5SGSD8	695,000	3,926	50,000	177	14
											E	5SEEB	952,000	704	52,000	1,352	18
2009	40 nm	Virtex 6	LX	XC6VLX760	474,240	864	25,920	549	18								
			SX	XC6VSX475T	297,600	2,016	38,304	148	8								
			HX	XC6VHX565T	354,240	864	32,832	410	11								
2008										Stratix IV	GT	EP4S100G5	531,200	1,024	27,376	519	19
											GX	EP4SGX530	531,200	1,024	27,376	519	19
											E	EP4SE820	813,050	960	33,294	847	24
2006	65 nm	Virtex 5	LX	XC5VLX330	207,360	192	10,368	1,080	20								
			SX	XC5VSX240T	149,760	1,056	18,576	142	8								
			FX	XC5VFX200T	122,880	384	16,416	320	7								
										Stratix III	L	EP3SL340	337,500	576	16,272	586	21
											E	EP3SE260	255,000	768	14,688	332	17
2005	90 nm									Stratix II	GX	EP2SGX130/G	132,540	252	6,747	526	20
	130 nm											EP2S180	179,400	384	9,383	467	19
2004	90 nm	Virtex 4	LX	XC4VLX200	178,176	96	6,048	1,856	29								
			SX	XC4VSX55	49,152	512	5,760	96	9								
			FX	XC4VFX140	126,336	192	9,936	658	13								
2002	130 nm									Stratix	GX	EP1SGX40D	41,250	56	3,423	737	12
												EP1S80	79,040	88	7,428	898	11
2001	130 nm	Virtex II	Pro	XC2VP100	88,192	444	7,992	199	11								
			Pro X	XC2VPX70	66,176	308	5,544	215	12								
	0.15 um		V	XC2V8000	93,184	168	3,024	555	31	Mercury		EP1M350	14,400	0	115	-	125
2000	0.18 um									Excalibur		EPXA10	38,400	0	3,146	-	12
1999	0.18 um	Virtex E		XCV3200E	64,896	0	851	-	76								
1998	0.22 um									Flex 10KE		EPF10K200E	9,984	0	98	-	102
	0.25 um	Virtex		XCV1000	24,576	0	131	-	188								
1997	0.35 um	4000 E/XL		XC4085XL	12,544	0	0	-	-								
1996	0.3 um									Flex 10KA		EPF10K250A	12,160	0	41	-	297
1995	0.42 um									Flex 10K		EPF10K100	4,992	0	25	-	200
1992	0.6 um									Flex 8000		EPF81500A	1,296	0	0	-	-
1991	0.8um	4000 series		XC4025	2,048	0	0	-	-								
1985	2 um	2000 series		XC2018	400	0	0	-	-								

As transistor size keeps scaling down, a bigger design can be constructed within FPGA, ranging from small building blocks to a very powerful System-on-Chip (Rodriguez-Andina et al., 2015). Moreover, the maximum frequency achieved in the FPGA technology doubles every 8 years, which offer a trend to design a high performance computing platform using FPGA (Shannon, L. et al., 2015). Power consumption also reduces as transistor size scaling down, which by offering lower operational voltage (de

la Piedra et al., 2012). Table 2.8 and Table 2.9 show the related information gathered by de la Piedra et al. (2012).

Table 2.8: Xilinx FPGA chip analysis
Source: de la Piedra, A., Braeken, A. and Touhafi, A. (2012) 'Sensor Systems Based on FPGAs and Their Applications: A Survey', Sensors, 12(12), pp. 12235–12264. doi: 10.3390/s120912235.

Platform	Model	Core Voltage (V)	Number of LUTs	Block RAM (KB)	Static Power Consumption (mW)
Spartan-3	XC3S200	1.2	4,320	216	41
Spartan-3E	XC3S250E	1.2	5,508	216	51
Spartan-6	XC6SLX100	1.2	101,261	4,824	67
Virtex-4	XC4VLX200	1.2	200,448	6,048	1,278
Virtex-5	XC5VLX220	1	138,240	6,912	1,985
Virtex-6	XC6VLX240T	1	241,‖52	14,976	1,977
Virtex-7	XC7VX330T	1	326,400	27,000	141
Kintex-7	XC7K160T	1	162,240	11,700	74
Artix-7	XC7A100T	1	101,440	4,860	41

Table 2.9: Altera FPGA chip analysis
Source: de la Piedra, A., Braeken, A. and Touhafi, A. (2012) 'Sensor Systems Based on FPGAs and Their Applications: A Survey', Sensors, 12(12), pp. 12235–12264. doi: 10.3390/s120912235.

Platform	Model	Core Voltage (V)	Number of LUTs	Block RAM (KB)	Static Power Consumption (mW)
Cyclone	EP1C6	1.5	5,980	92	60
Cyclone II	EP2C8	1.5	8,256	165	40
Cyclone V	5CEFA9	1.10	301,000	12,200	206
Stratix	EP1S25	1.5	25,660	635	450
Stratix II	EP230	1.5	33,880	663	86
Startix V	5SEE9	1	840,000	12,800	880
Arria V	57GXMA1D	1	75,000	8,000	197

With the rapid evolution of semiconductor technology, FPGA manufacturer often come out with extra hardware resources as competitive advantages among competitor (Rodriguez-Andina, J. J., Valdes-Pena, M. D. and Moure, M. J., 2015; Rodriguez-Andina, J. J., Moure, M. J. and Valdes, M. D., 2007; Kuon, I., Tessier, R. and Rose, J., 2007). One of the useful resources are the memories, either volatile or non-volatile memory or both, where user's

program code or data may reside in the memory. Xilinx Analog-to-Digital Converter (XADC) block, offered by Xilinx, allows high-quality analog-to-digital conversion and customizable signal conditioning, Phase-locked loop (PLL) and Delay-locked loop (DLL) can be used to compensate clock propagation delays throughout the FPGA. A soft IP core (MicroBlaze soft-core by Xilinx, Nios II soft-core by Altera etc.) or hard-core processor was integrated on the FPGA board.

With the improving of power consumption in FPGA, it allows turning existing IoT devices into a low power customizable FPGA-IoT platform (Gomes, T. et al., 2015). Several projects had been completed on FPGA covering multimedia application, industrial control, environmental monitoring and safety and security applications (de la Piedra, A. et al., 2012). An example of the project is the development of a co-processor on FPGA (Garcia, R. et al., 2009). This project implemented the Kalman filter for tracking environmental targets, such as animals. Several Kalman filter configurations can be developed depending on the type of objects and operation stages. Besides that, partial reconfiguration feature offered by FPGAs is used to reduce the power consumption. With this approach, power consumption is reduced by 5 - 25 %. Another project using FPGA soft-core, MicroBlaze processor, is used as the processing unit by Hongzhi Liu and Bergmann, N. W. (2010). This project aimed to develop a platform that performs bird call detection. Besides that, another project used the combination of the microcontroller and the FPGA, which the FPGA serve as the co-processor, had been implemented by Vana Jeličić et al. (2011). An 8-bit AVR microcontroller with an FPGA based co-

processor that is able to perform image processing is used for pest detection in olive groves. This platform consumes 87.12 mW in active mode and 18.4 uW in sleep mode at 3.3 V. The projects mentioned had shown a promising result to convince more research on the development of IoT devices using FPGA as the implementation technology.

2.3 Low power techniques in FPGA

Kuon, I., Tessier, R. and Rose, J. (2007) stated that power consumption in FPGAs is categorized into 2 types: static and dynamic power consumption. Dynamic power is consumed by the transitioning of the signals logic level (either 0 to 1 or 1 to 0). A large amount of energy is used to charge or discharge the load capacitance of the transistors in the circuit. In contrast, static power is consumed when using a relatively smaller amount of energy to maintain the same logic level.

Conventional power reduction technique includes dynamic voltage scaling (DVS), dynamic frequency scaling (DFS), dynamic voltage and frequency scaling (DVFS), clock gating and power gating have been implemented on FPGA-based soft-core design in the past. Power reduction using dynamic voltage scaling (DVS) presented by Chow, C. T. et al. (2005) shows a power saving between 4% to 54% is achieved on a 0.18 um Xilinx Virtex 300E-8 FPGA chip. The internal supply voltage (VCCINT) source is replaced by a voltage controller to dynamically adjust the supply voltage. Two different clock frequencies (66 MHz and 100 MHz) have been used to test the efficiency of the DVS, in which the VCCINT supply voltage is reduced to meet the timing requirements and at the same time saving power. DVS extended with dynamic frequency scaling (DFS) to formed dynamic voltage and frequency scaling (DVFS) with extra capability of adaptive voltage scaling has been implemented by Nunez-Yanez, J. L. (2015). The technique proposed is capable of reducing both static and dynamic power consumptions. The experimental work had been implemented on a Xilinx XUPV5-LX110T

evaluation board, with a 65 nm Virtex-5 XC5VLX110T FPGA chip on board. The author replaced the fixed voltage DC-to-DC module on the FPGA board with a specially designed DC-to-DC module, which is able to scale the VCCINT supplying to FPGA logic resources. The corresponding maximum working frequency for the minimum voltage (0.62V) is 40 MHz and achieves maximum power reduction up to 87% (from 615 mW to 80 mW). DVFS extended with power gating and partial reconfiguration between one (ME1) and six (ME6) execution units of a motion estimation processor applies on Xilinx Zynq board, with a 28 nm Xilinx Virtex-7 FPGA chip on board, has been carried out by Luis Nunez-Yanez, J., Hosseinabady, M. and Beldachi, A. (2016). This study shows a power reduction up to 62% (124 mW to 47 Mw) for ME1 and 52% (285 mW to 137 mW) for ME6. Since ME6 is expensive from the energy usage point of view, the author suggested using the ME6 to complete the job fast while idling using ME1 until a new request is received. Furthermore, both studies (Nunez-Yanez, J. L., 2015; Luis Nunez-Yanez, J., Hosseinabady, M. and Beldachi, A., 2016) show a dramatically reduce in total power consumption that has been achieved by the manufacturer, from 65 nm to 28 nm process technology, which shows a competitive advantage in using FPGAs to implement the design.

On the other hand, power gating also able to reduce the static and dynamic power (Hosseinabady, M. and Nunez-Yanez, J. L., 2014; Hosseinabady, M. and Nunez-Yanez, J. L., 2015). The research work by Hosseinabady, M. and Nunez-Yanez, J. L. (2014) shows that the power gating can reduce the power consumption up to 96%. The authors used the hard-core

processor (Cortex A9) on the Xilinx ZYNQ device to power-off the FPGA chip when it is idle with timing overhead (the time for turn-off, turn-on and reconfiguration of the programmable logic) as low as 42.58ms. The authors had extended their research work by applied a streaming application (MP3 player) on FPGA and perform up to 52.9% energy reduction (Hosseinabady, M. and Nunez-Yanez, J. L., 2014). However, this technique (power gating) requires an extra hard-core processor to serve as the watchdog core which consumes extra power other than FPGA.

In contrast, clock gating technique in RTL modeling does not require a hard-core processor or any physical modifications on hardware. Clock gating (Oklobdzija, V. G. and Krishnamurthy, R. K., 2006) technique is a popular technique that used to reduce the dynamic power consumption of the processor. The design logic of the processor is made up of sequential circuits and combinational logic. The sequential circuits do consume energy on every pulses of the clock, even when it is not affecting the final output. The solution to avoid this situation is by disabling the clock input of the sequential circuits. Figure 2.1 illustrates the implementation of the clock gating technique on the D Flip-flop (DFF).

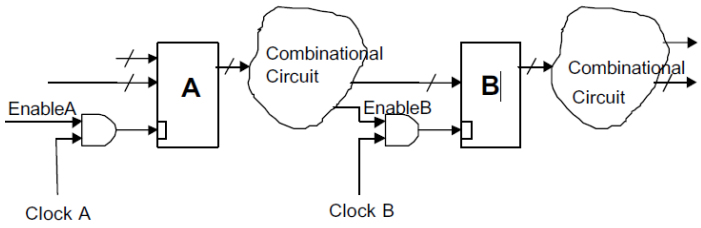


Figure 2.1: Clock gating technique illustration diagram
Source: Oklobdzija, V. G. and Krishnamurthy, R. K. (2006) High-Performance Energy-Efficient Microprocessor Design. Edited by V. G. Oklobdzija and R. K. Krishnamurthy. Boston, MA: Springer US (Series on Integrated Circuits and Systems). doi: 10.1007/978-0-387-34047-0.

From Figure 2.1, input signals EnableA and EnableB perform AND operation with the clock source of the DFF. When both EnableA and EnableB are de-asserted, the clock pulse will not pass into the DFF and thus the DFF stop functioning. Pandey, B. et al. (2013) proposed a Random Access Memory (RAM) unit applied with clock gating technique implemented on a 40 nm Xilinx Virtex-6 FPGA chip. This research work shows a power reduction by 38.89% on the 1 GHz system clock and 41.3% on the 10 GHz system clock, which means the clock gating technique is more beneficial for higher clock frequency. Yan Zhang, Roivainen, J. and Mammela, A. (2006) tested the clock gating technique with several benchmark circuits (CombFilter, EthernetInterface, FrequencyEstimator, Half-bandFilter and I^2C-Interface) and resulting in power saving of 50% to 80% of the dynamic power consumption on a 0.13 um Xilinx Virtex-II FPGA chip.

The discussed power reduction techniques in FPGA had shown an exceptional performance in reducing the power consumption. However, those techniques to achieve low-power consumption or higher computational speed

by manipulating the voltage and operating frequency, are still confined to a fixed microarchitecture. A real-time adaptive microarchitecture for low-power consumption and higher computational speed has yet to be addressed. Our intention is to provide a platform that is able to switch between multi-cycle (low power) and pipeline microarchitectures (high computational power). Thus, we shall adopt the partial reconfiguration (PR) feature offered by FPGA to implement the proposed platform.

2.4 Partial Reconfiguration

One of the noticeable features offer by the FPGA is the reconfiguration, either partial or dynamic run-time self-reconfiguration (Becker, J. et al., 2007). This feature allows the reconfiguration of a certain part of the hardware. Meanwhile, the power constantly fed into the FPGA chip and no hardware reset is required. Thus, increase the adaptation of a system with the actual demands of the applications running on the FPGA chip. By using this feature, it is possible to store part of the hardware functionality to an external non-volatile memory and partial reconfiguration (PR) can be carry out on demand. Thus, power dissipation is reduced since the overall design is smaller. Figure 2.2 shows Xilinx illustration on the partial reconfiguration (Xilinx, 2016a).

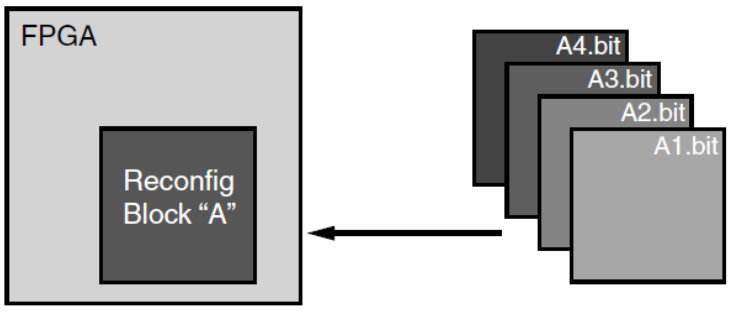


Figure 2.2: Partial reconfiguration Illustration diagram
Source: Xilinx (2016a) 'Vivado Design Suite User Guide Partial Reconfiguration'

Reconfig Block A shown in Figure 2.2 can be replaced by copy over any of the partial bitstream (A1.bit, A2.bit, A3.bit, or A4.bit) on the external non-volatile memory to the FPGA. The partial bitstream is also possible to be transfer from an external smart source through JTAG connection, e.g. a computer (Xilinx, 2016b). Xilinx stated that partial reconfiguration is able to

reduce the FPGA design area that required to implementing a functional hardware, and thus reduce the cost and the power consumption since the cost per unit and the power consumption of the external non-volatile memory is lesser than the FPGA chip. Partial reconfiguration also provides the flexibility in the choices of the algorithms and the protocol for an application, improves FPGA fault tolerance and lastly accelerate configurable computing.

In order to perform PR, a PR controller is required to trigger and control the action of read over the partial bitstream from an external non-volatile memory and write to the FPGA (data loading) through Internal Configuration Access Port (ICAP) based on Xilinx technology (Xilinx, 2016b; Cardona, L. A. and Ferrer, C., 2015). Data loading on the FPGA requires specific timing requirement, generally categorize as continuous data loading and non-continuous data loading. Continuous data loading provides an uninterrupted stream of partial bitstream loading to the FPGA while non-continuous data loading allows an interrupted stream of partial bitstream loading to the FPGA. Continuous data loading requires extra design area and hardware, i.e. FPGA Block RAMs (BRAMs), to use as the temporary buffer to store the partial bistream copied from the external non-volatile memory and write to FPGA in a bunch, in order to reduce the overhead and complete the PR faster (Cardona, L. A. and Ferrer, C., 2015). However, extra design area and hardware used tends to increase the power consumption when the PR takes place. In opposite, non-continuous data loading can reduce the hardware used by directly read the partial bitstream from the external non-volatile memory and write to FPGA through ICAP word by word (32-bits). However,

a lower performance is achieved since the data reading from the external non-volatile memory is usually in serial form. Our experimental work will be based on non-continuous data loading, so that to reduce the design area and the hardware used, which can help to save power when PR takes place.

An example of the reconfigurable system that had been carried out by McDonald, E. (2008), is by constructed a software-defined radio system on the FPGA. The reconfigurable system allows a simplex transceiver to be reconfigured, where either transmit or receive capability is used at any given time and never used at the same time. However, due to the lack of information in power analysis of the proposed reconfigurable system, we cannot predict on how much the improvement of the energy consumption achieved. Krasteva, Y. E. et al. (2008) used the FPGA as a reconfigurable coprocessor that used for sensor data aggregation and data processing. There are 4 partial bitstreams created, which are temperature sensor nodes with the multiplier (TMPS_HW_v2), temperature sensor nodes without the multiplier (TMPS_HW_v1), accelerometer sensor nodes with the multiplier (ACCS_HW_v2), and accelerometer sensor nodes without the multiplier (ACCS_HW_v1). The design areas and the bitstream file size are shown in Table 2.10 and Table 2.11.

Table 2.10: Reconfigurable system hardware resources usage
Source: Krasteva, Y. E. et al. (2008) 'Remote HW-SW reconfigurable Wireless Sensor nodes', in 2008 34th Annual Conference of IEEE Industrial Electronics. IEEE, pp. 2483–2488. doi: 10.1109/IECON.2008.4758346.

Design	Used MULs	Used Slices	Used FFs	Used LUTs	% of Slot Slices
ACCS_HW_v1	0	170	68	311	29
TMPS_HW_v1	0	71	40	127	12
ACCS_HW_v2	2	127	68	232	22
TMPSI_HW_v2	2	63	40	111	10

Table 2.11: Reconfigurable system file size
Source: Krasteva, Y. E. et al. (2008) 'Remote HW-SW reconfigurable Wireless Sensor nodes', in 2008 34th Annual Conference of IEEE Industrial Electronics. IEEE, pp. 2483–2488. doi: 10.1109/IECON.2008.4758346.

Design	.bit (KB)	.xsvf (KB)
ACCS_HW_v1	24,9	25,2
TMPS_HW_v1	24,9	25,2
ACCS_HW_v2	41,3	43,5
TMPSI_HW_v2	41,3	43,5

One of the brilliant features of this project is that the PR bitstreams do not reside in the external non-volatile memory on FPGA board. Instead, the authors using wired or wireless remote to send over the partial reconfiguration bitstream to the 8052 microcontroller to initiate the FPGA PR. The authors had tested with several remote connections, which are cable with 8-bytes packet size (Cable 8B), ZigBee with 8-bytes packet size (ZigBee 8B), cable with 16-bytes packet size (Cable 16B) and ZigBee with 16-bytes packet size (ZigBee 16B). However, the result for the power consumption was not provided by the authors.

Another project by Hinkelmann, H., Zipf, P. and Glesner, M. (2007) used the reconfigurable feature of the FPGA to construct a coarse-grained, domain-specific reconfigurable function unit (RFU). The RFU allows a functional task to perform only certain hardware module to exist on the FPGA. The non-related hardware module will reside in the external non-volatile configuration memory to save power. The RFU is aimed to perform lightweight error detection and correction (CRC-8 checksum calculation and BCH decoding), AES key generation and AES encryption. Table 2.12 shows the energy comparison of the software approach versus the RFU of the given reconfigurable system.

Table 2.12: Reconfigurable system power analysis
Source: Hinkelmann, H., Zipf, P. and Glesner, M. (2007) 'A Domain-Specific Dynamically Reconfigurable Hardware Platform for Wireless Sensor Networks', in 2007 International Conference on Field-Programmable Technology. IEEE, pp. 313–316. doi: 10.1109/FPT.2007.4439274.

Task	Version	Execution [nJ]	Reconf. [nJ]	Gain factors
CRC-8	software	387,6	-	1
CRC-8	RFU	4,8	12,2	81 (23)
AES key gen.	software	234,6	-	1
AES key gen.	RFU	39,2	22,5	6,0 (3,8)
AES encrypt.	software	640,6	-	1
AES encrypt.	RFU	104,9	43,1	6,1 (4,3)
BCH decod.	software	1208,7	-	1
BCH decod.	RFU	245,5	34,4	4,9 (4,3)

From Table 2.12, we can conclude that the reconfigurable system gain more power efficiency compare with the software implementation of a given task. However, the authors only provide the power consumption comparison between the software method and the reduced hardware implementation of a given task, which is insufficient, since our main concern is to identify the

difference of the non-reconfiguration system (i.e. all the hardware module exist on FPGA) versus the reconfigurable system with RFU (i.e. only certain hardware module exist on FPGA). A reconfigurable instruction set extensions has been presented by Koch, D. et al. (2012). They pointed out that the custom instruction, for example, an instruction that used to permuting all bits in a 32-bit operand, can reside as a small reconfigurable slot, which consists of a bunch of CPU instructions, instead of constructing the circuitry for the custom instruction. This allows an expensive algorithm to be executed with only one instruction call. Hansen, S. G., Koch, D. and Torresen, J. (2013) show a case study using a 32-bit Microprocessor without Interlocked Pipeline Stages (MIPS) soft-core processor to implement with the reconfigurable instruction set extensions. Figure 2.3 shows the microarchitecture of the reconfigurable instruction set extensions.

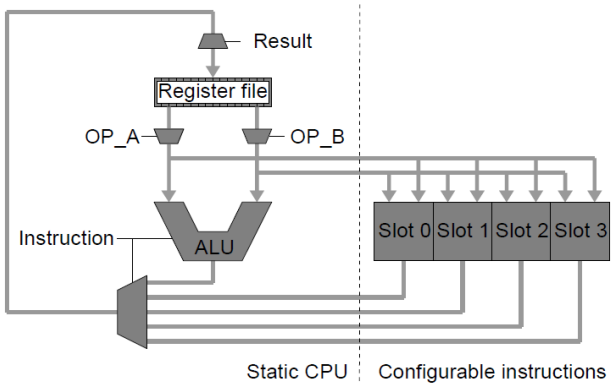


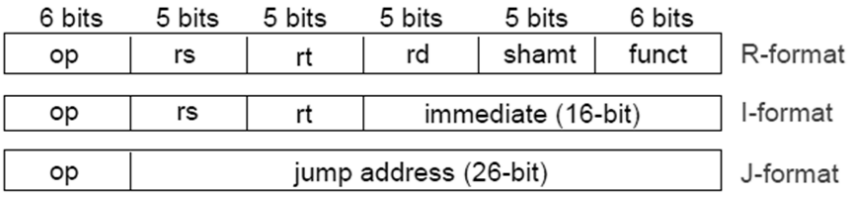
Figure 2.3: Reconfigurable instruction set extension architecture
Source: Hansen, S. G., Koch, D. and Torresen, J. (2013) 'Simulation framework for cycle-accurate RTL modeling of partial run-time reconfiguration in VHDL', in 2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC). IEEE, pp. 1–8. doi: 10.1109/ReCoSoC.2013.6581519.

There are 4 reconfigurable slots, where each slot stores a bunch of CPU instructions to form a function or an algorithm (AES encryption, CRC checksum etc.). This approach can reduce the effort on creating the hardware dedicated to perform special function or algorithm. The design area also reduced, which consequently reduce the power consumption.

We intended to develop a PR system that covers the processor general purpose instructions. Our work is different from the existing work, in which the existing works require the development of the dedicated hardware component (e.g. AES encryption hardware), extra instruction to invoke the usage, and only use for specific purpose (e.g. AES data encryption).

2.5 MIPS ISA

The Microprocessor without Interlocked Pipeline Stages instruction set architecture (MIPS ISA) has been widely used in the research and experiment by many researchers in the past decades (Hennessy, J. L. and Patterson, D. A., 2012; Patterson, D. A. and Hennessy, J. L., 2013). MIPS ISA is developed based on the Reduced Instruction Set Computing (RISC) philosophy, which strongly emphasizes in reduce instruction support and simple hardware structure to increase the processor performance while decreasing the power consumption. In contrast, the Complex Instruction Sets Computing (CISC) processor emphasizes on maximizing the performance by increasing the hardware parallelism and complexity, but increase the power consumption. Thus, a MIPS ISA compatible processor is developed in our project due to its simple hardware structure, which helps to shorten the development cycle while aimed for low power consumption. Each MIPS ISA compatible instruction has 32-bit in data length. The MIPS ISA compatible instructions are classified into 3 instruction format as shown in Figure 2.4. Each instruction format is further divided into several addressing modes, which shown in Table 2.13.



R-format: Instruction operation involving registers only
I-format: Instruction operation involving register and immediate value
J-format: Unconditional branching to 26-bit address specify in machine code

Figure 2.4: MIPS ISA compatible instruction format bit allocation.

Table 2.13: MIPS instruction addressing modes

Instruction format	Addressing Mode
R-format	Register Addressing
I-format	Immediate Addressing
	Base Addressing
	PC-Relative Addressing
J-format	Pseudodirect Addressing

The MIPS ISA compatible processor illustrates in Figure 2.5 consists of 5-stage instruction execution cycle, which corresponds to 5 hardware stages: Instruction Fetch (IF), Instruction Decode and Register File Read (ID), Execution or address calculation (EX), Data Memory Access (MEM) and Write Back (WB).

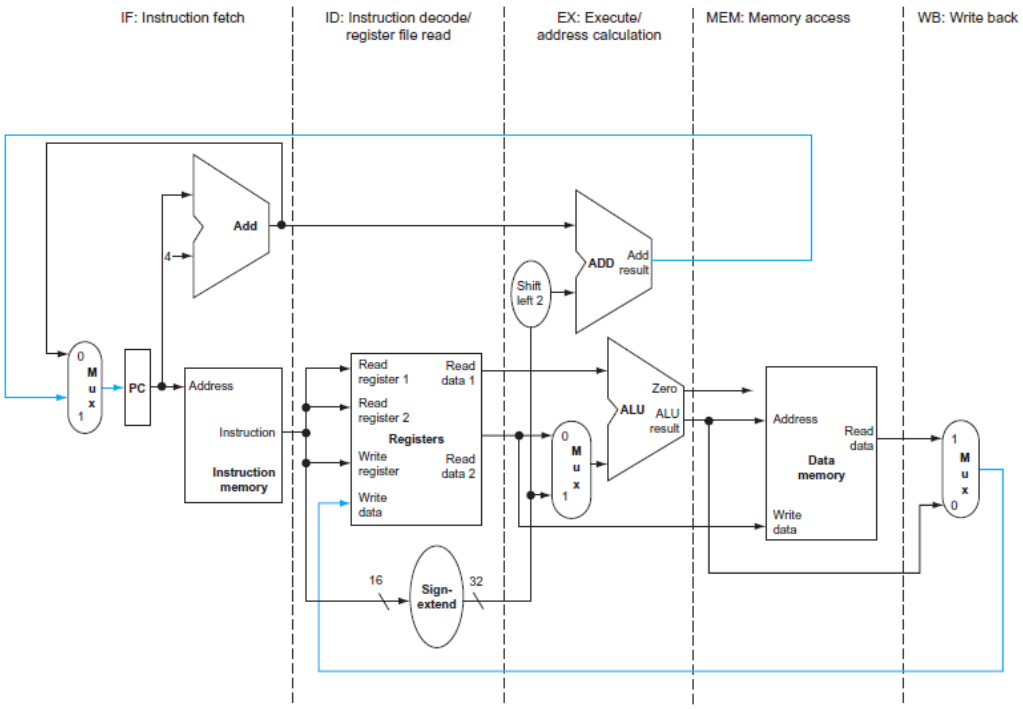


Figure 2.5: Hardware stages of MIPS ISA compatible processor. Source: Hennessy, J. L. and Patterson, D. A. (2012) Computer architecture: a quantitative approach, Elsevier. doi: 10.1.1.115.1881.

IF stage is responsible to fetch an instruction from the instruction memory according to the address stored in the Program Counter (PC) register. ID stage decodes the instruction from IF stage and at the same time fetch the operands from the Register File. EX stage perform the operation or address calculation when load or store instruction is executed. A load or store instruction will access the data memory in the MEM stage. The processed result may store back to the Register File in WB stage.

2.6 Summary

This chapter is summarized as follows:

1) The computational speed required for each IoT application is determined by the sampling rate of the sensor node, where high computational speed is required to measure and process the data for high sampling rate IoT application, while low computational speed is used to measure and process the data for low sampling rate IoT application.
2) General purpose off-the-shelf microcontroller is popular in implementing IoT application. However, several limitations restrict the general purpose off-the-shelf microcontroller in providing the best solution in implementing a low power IoT application.
3) Both ASIC and FPGA show respective advantages in design implementation. However, with the improvement in transistor scaling technology, FPGA provides a better choice in implementing a low power customizable FPGA-IoT platform.
4) Several low power techniques in FPGA are discussed, including DVS, DVFS, power gating and clock gating.
5) Partial Reconfiguration (PR) feature offered by FPGA provides a better energy efficient solution in design implementation and several projects are discussed.
6) MIPS ISA has been widely used in the research. Its simple hardware structure increases the processor performance while decreasing the power consumption.

CHAPTER 3

HARDWARE DEVELOPMENT

3.1 System Overview

The proposed reconfigurable soft-core IoT processor is made up of 3 major parts: Central Processing Unit (CPU), memory system and I/O system. The developed CPU is compatible to the 5-stage 32-bit MIPS Instruction Set Architecture (ISA). It supports up to 50 instructions, covering arithmetic, logical, data transfer, program control and system instruction classes. The memory system consists of a 2-level memory hierarchy. The first level consists of cache, Boot ROM and Data and Stack RAM, and the second level consists of flash memory. The cache (high speed FPGA Block RAM) is used to enhance the speed of instructions and data accessing from the non-volatile memory (low speed flash memory). A duplicated set of data from the flash memory are present in the cache after the processor power up. Flash memory is used to store FPGA configuration bitstream, program code, constant variables and partial reconfiguration bitstream. Data and Stack RAM, implemented from the FPGA BRAM, is used to store runtime variables (e.g. variables generated by function call and dynamic data structures), while Boot ROM store the bootloader program.

I/O system consists of GPIO controller, SPI controller, UART controller, Priority Interrupt controller and General Purpose Register (GPR) unit. They were integrated with CPU through Wishbone B4 standard bus

interface (OpenCores, 2010). GPIO, SPI and UART controllers are used to communicate with external devices, such as, sensors, wireless modules, personal computers etc. The Priority Interrupt controller is used as an external interrupt controller (Co-processor 0 (CP0) is the internal interrupt controller) to handle multiple interrupt sources based on priority. GPR unit is used to store the PR information including current microarchitecture identification bit, PR bitstream size, and PR bitstream start address for both multi-cycle and pipeline executions in the flash memory. Figure 3.1 shows the architecture of the developed reconfigurable soft-core IoT processor.

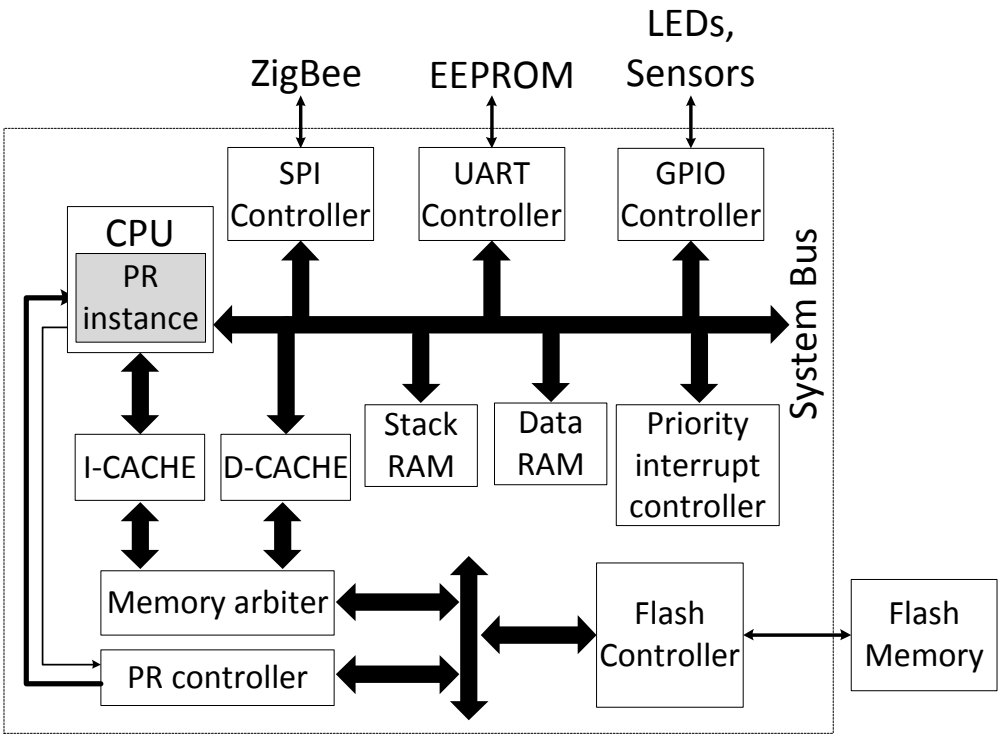


Figure 3.1: Reconfigurable IoT processor architecture

Our research target is to satisfy the varying performance-power tradeoff requirements of the IoT applications. Achieving both design goals

(low power and high computational speed) in one processor architecture is non-trivial due to their conflicting requirements. With the emergence of FPGA technology, PR feature offered by FPGA allows reconfiguration between pipeline (high computational speed at the expense of high power usage) and multi-cycle executions (low computational speed with low dynamic power consumption) according to each IoT application requirement needs.

Figure 3.1 illustrates the categorization of the hardware components in both static and PR regions (a.k.a PR instance) in an overall view. The proposed technique classifies the reconfigurable soft-core IoT processor into 2 regions: static and PR regions. The static region consists of hardware that does not change regardless of the pipeline or multi-cycle microarchitectures. In contrast, the PR region consists of only pipeline or multi-cycle microarchitecture at a time, i.e. multi-cycle microarchitecture executing in the PR region while pipeline microarchitecture is kept in bitstream format which resides in the flash memory, and vice versa. The scope of the PR region limits within the CPU (Memory system and I/O system are not included) since both multi-cycle and pipeline microarchitectures are the implementation technique in modifying the processor execution structure. The detailed view of the CPU components being reconfigured is shown in Figure 3.2.

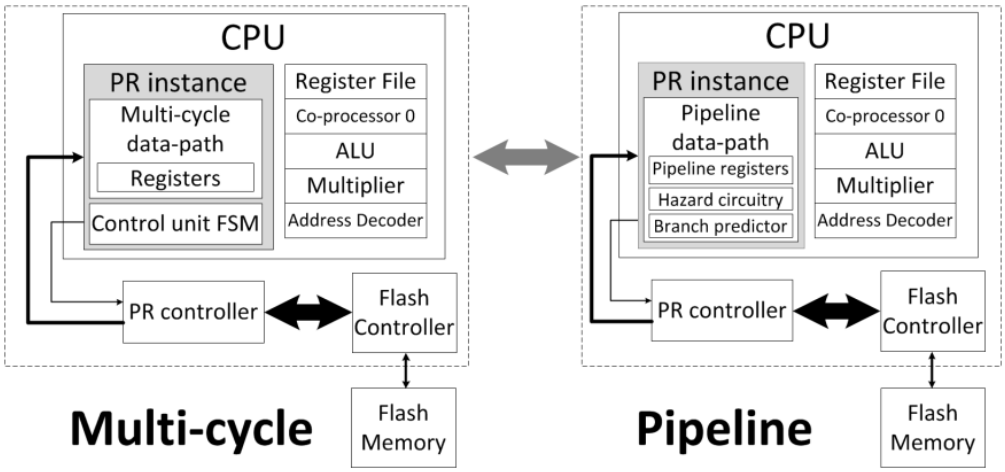


Figure 3.2: Selected reconfigurable components from CPU.

The partial bitstream size is determined by the design area of the PR region, i.e. large design area produces a large size of the partial bitstream, and vice versa. Besides that, large partial bitstream size will incur longer PR time. Although the hazard circuitries (data forwarding block and interlock block) are only used by the pipeline microarchitecture, it is possible to allocate the hazard circuitries to the static region in order to reduce the PR time. We could deactivated the hazard circuitries during multi-cycle execution, by wired the inputs of the hazard circuitries to the ground and the outputs left unconnected and thus, no dynamic power is consumed by the hazard circuitries during multi-cycle execution. Table 3.1 shows the specification of both multi-cycle and pipeline executions.

Table 3.1: Specification of multi-cycle and pipeline executions

		Multi-cycle	Pipeline
Frequency (MHz)		20	20
Instruction's cycle		3 - 5	5, overlapping
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
Common features (Static Region)	Memory	4kBytes Boot ROM, 128kBytes user access flash, 8kBytes RAM (Data & Stack), 1kBytes i-cache, 128Bytes d-cache, 512Bytes Memory Mapped I/O Register	
	Communication interface	UART, SPI, 32 GPIO pins	
Partial Bitstream start address		0x00A0_0000	0x00A8_0000
Bitstream size		1,404,992 bits / 43906 words	
FPGA board		Nexys 4 DDR (XC7A100T)	
FPGA Resources (Overall)	LUT	7643	8561
	LUTRAM	127	311
	FF	5464	5812
	BRAM	3.50	3.50
	IO	45	45
	BUFG	1	1

3.2 CPU

3.2.1 MIPS ISA compatible

The processor is compatible with MIPS ISA. It is fixed to support up to 50 MIPS ISA compatible basic core instructions, including 1 special instruction, Toggle Microarchitecture (*tma*), added for PR purposes. The supported instruction format and addressing modes have been shown in Figure 2.4 and Table 2.13, respectively. Referring to the instruction format and addressing modes supported, the instruction field of the instruction supported is shown in Table 3.2.

Table 3.2: Instruction field information [refer to Patterson, D. A. and Hennessy, J. L. (2013) for the information on the instruction usage]

No	Instruction	$opcode_{[31:26]}$	$rs_{[25:21]}$	$rt_{[20:16]}$	$rd_{[15:11]}$	$shamt_{[10:6]}$	$funct_{[5:0]}$
		$opcode_{[31:26]}$	$rs_{[25:21]}$	$rt_{[20:16]}$	$immediate_{[15:0]}$		
		$opcode_{[31:26]}$	$address_{[25:0]}$				
1	add	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	100000
2	addu	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	100001
3	sub	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	100010
4	subu	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	100011
5	mult	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	011000
6	multu	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	011001
7	mfhi	000000	00000	00000	[xxxxx]	00000	010000
8	mflo	000000	00000	00000	[xxxxx]	00000	010010
9	and	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	100100
10	or	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	100101
11	xor	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	100110
12	nor	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	100111
13	sll	000000	00000	[xxxxx]	[xxxxx]	[xxxxx]	000000
14	srl	000000	00000	[xxxxx]	[xxxxx]	[xxxxx]	000010
15	sra	000000	00000	[xxxxx]	[xxxxx]	[xxxxx]	000011
16	slt	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	101010
17	sltu	000000	[xxxxx]	[xxxxx]	[xxxxx]	00000	101011
18	jr	000000	[xxxxx]	00000	00000	00000	001000
19	jalr	000000	[xxxxx]	00000	[xxxxx]	00000	001001
20	syscall	000000	00000	00000	00000	00000	001100
21	mtc0	010000	00100	[xxxxx]	[xxxxx]	00000	000000
22	mfc0	010000	00000	[xxxxx]	[xxxxx]	00000	000000
23	eret	010000	00001	00000	00000	00000	011000
24	addi	001000	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
25	addiu	001001	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
26	andi	001100	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
27	ori	001101	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		

Continued from Table 3.2

No	Instruction	$opcode_{[31:26]}$	$rs_{[25:21]}$	$rt_{[20:16]}$	$rd_{[15:11]}$	$shamt_{[10:6]}$	$funct_{[5:0]}$
		$opcode_{[31:26]}$	$rs_{[25:21]}$	$rt_{[20:16]}$	$immediate_{[15:0]}$		
		$opcode_{[31:26]}$	$address_{[25:0]}$				
28	xori	001110	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
29	lui	001111	00000	[xxxxx]	[xxxxxxxxxxxxxxxx]		
30	lw	100011	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
31	lwl	100010	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
32	lwr	100110	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
33	lh	100001	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
34	lhu	100101	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
35	lb	100000	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
36	lbu	100100	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
37	sw	101011	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
38	swl	101010	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
39	swr	101110	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
40	sh	101001	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
41	sb	101000	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
42	slti	001010	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
43	sltiu	001011	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
44	beq	000100	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
45	bne	000101	[xxxxx]	[xxxxx]	[xxxxxxxxxxxxxxxx]		
46	blez	000110	[xxxxx]	00000	[xxxxxxxxxxxxxxxx]		
47	bgtz	000111	[xxxxx]	00000	[xxxxxxxxxxxxxxxx]		
48	j	000010	[xxxxxxxxxxxxxxxxxxxxxxxxxx]				
49	jal	000011	[xxxxxxxxxxxxxxxxxxxxxxxxxx]				
50	tma	000000	00000	00000	00000	00000	111111

The developed processor consists of 5 hardware stages, which is illustrated in Figure 2.5. The IF stage is responsible to fetch an instruction from the instruction memory according to the address stored in the Program Counter (PC) register. The instruction memory is a 2-level memory hierarchy memory consists of Boot ROM, I-CACHE and flash memory. The ID stage decodes the instruction from IF stage and at the same time fetch the operands from the Register File (RF). The EX stage performs the execution or address calculation for load and store instructions. A load or store instruction will access the D-CACHE, Data and Stack RAM or I/Os registers in the MEM

stage. The result of the execution will be store back to the Register File (RF) in WB stage.

To allow PR between multi-cycle or pipeline microarchitectures, two versions of processor microarchitectures were developed: pipeline and multi-cycle microarchitectures. In pipeline microarchitecture, every instruction requires 5 clock cycles to complete its execution. Every instruction in the pipeline microarchitecture occupies a single stage for only one clock cycle and compulsory to run through the 5 hardware stages prior to the end of its execution. Multi-cycle microarchitecture also used the same concept of 5 hardware stages structure, except that each instruction takes 2 to 5 clock cycles to execute, which correspond to 2 to 5 hardware stages. Each instruction in the multi-cycle microarchitecture must complete its execution cycle before the consecutive instruction begins to execute.

3.2.2 Pipeline microarchitecture

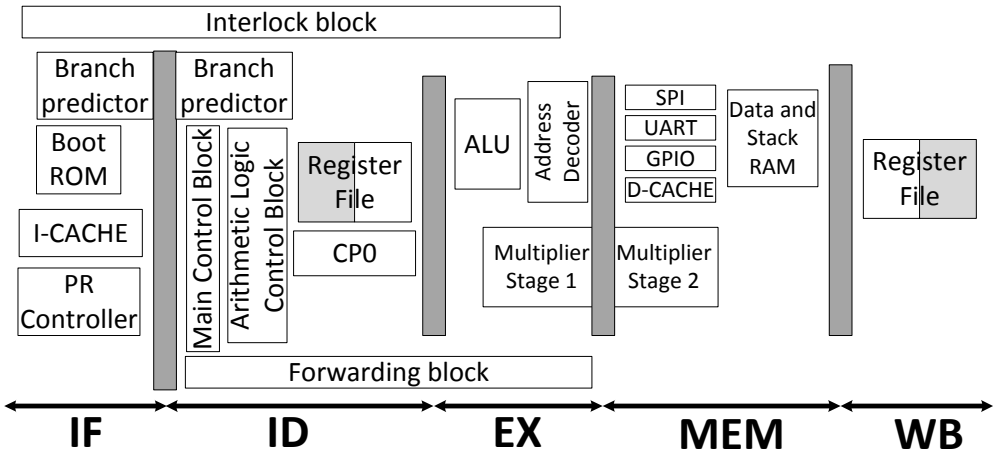


Figure 3.3: Abstract view of 5-stage pipeline processor

Figure 3.3 illustrates the hardware components allocate in each pipeline stages of the 5-stage pipeline processor. At IF stage, an instruction is fetched from the Boot ROM or I-CACHE and registered to the IF/ID pipeline registers. If a cache miss occurs in the IF stage, the I-CACHE will send a signal to stall the processor execution. The execution continues when the respective instruction successfully copied from the flash memory to the I-CACHE. At ID stage, the instruction that registered in the IF/ID pipeline registers will be decoded by the Main Control block and the Arithmetic Logic Control block. Signals output from both hardware components will be registered to the ID/EX pipeline registers and also pass to the remaining hardware components in the ID stage, i.e. Register File block, Forwarding block, CP0 block, Branch Predictor block and Interlock block. At the same time, IF stage continues to fetch the consecutive instruction from the I-CACHE. At EX stage, ALU block covers all the operation except the multiplication operation. Multiplier block starts the multiplication operation at EX stage and requires 2 clock cycles (EX and MEM stages) to perform a

multiplication operation on two 32-bit operands. At the MEM stage, only load and store instructions are permitted to perform the operation, in which other instructions are bypassing this stage. Load or store instruction access the memory components, i.e. D-CACHE, Data and Stack RAM and I/Os registers, at the MEM stage. At WB stage, the result of the operation is updated at the second clock edge (negative edge).

Branch predictor is included in the pipeline microarchitecture to enhance the performance of conditional and unconditional branch instructions. The second reason to include the branch predictor is to reduce the program code size, i.e. no branch delay slot (e.g. nop instruction) is required after every conditional and unconditional jump instructions. Multi-cycle microarchitecture does not need a branch delay slot since the instruction must complete its execution prior to the start of executing the consecutive instruction. Hence, the third reason in using a branch predictor is to allow the same code (without the branch delay slots) to be used in both pipeline and multi-cycle executions which can reduce field work on re-programming the IoT sensor nodes. Adding the unnecessarily branch delay reduces the computational speed of the multi-cycle microarchitecture.

Data hazards always exist in a pipeline processor. It can cause a computational error. Data hazard occurs due to Read-after-Write (RAW) data dependencies, which involve accessing the processor's system registers, i.e. Register File, CP0 registers and HILO register. Extra circuitries (forwarding block and interlock block) are required to resolve the data hazards arise.

However, the high computational speed achieved by pipelined processor still outweighs its counterparts and remains a popular choice in processor design (Kiat, W. P. et al., 2017).

Figure 3.4 shows the microarchitecture of the 5-stage pipeline processor while the design hierarchy is shown in Table 3.3. To enable PR in pipeline microarchitecture, design restructuring is performed, as shown in Figure 3.5, to reduce the PR overhead.

Table 3.3: Pipeline microarchitecture design hierarchy

Chip Level	Unit Level (Microarchitecture Level)	Block Level (Microarchitecture Level)	Sub-block
crisc	Data-path unit (udata_path)	Branch Predictor block (bbp_4way)	
		Register File block (brf)	
		Forwarding block (bfw_ctrl)	
		Interlock block (bitl_ctrl)	
		CP0 block (bcp0)	
		ALU block (balb)	
		Multiplier Block (bmult32)	adder_lvl1_firstrow
			adder_lvl1
			add_lvl1_lastrow
			sub_lvl1_lastrow
			adder_lvl2
			adder_lvl2_lastrow
			adder_lvl3
			adder_lvl4
			adder_lvl5
		Address Decoder block (baddr_decoder)	
	Control-path unit (uctrl_path)	Main Control block (bmain_ctrl)	
		Arithmetic Logic Control block (balb_ctrl)	
	Cache unit (ucache)	Cache Controller block (bcache_ctrl)	
		Cache RAM block (bcache_ram)	
	Flash Controller Unit (ufc)	Flash Controller Clock Generator block (bfc_clk_gen)	
		Flash Controller FSM block (bfc_fsm)	

Continued from Table 3.3

Chip Level	Unit Level (Microarchitecture Level)	Block Level (Microarchitecture Level)	Sub-block
		Flash Controller Transmitter block (bfc_TX)	
		Flash Controller Receiver block (bfc_RX)	
		FIFO block (bfc_FIFO)	
	Data and Stack RAM unit (uram)		
	UART Controller unit (uuart)	UART Baud Clock Generator block (bclkctr)	
		UART Receiver block (brx)	sbrx_ctr
			asynfifo_r1_3
			fifomem_b1_1
			graycntr_r1_3
			synchronizer
		UART Transmitter block (btx)	sbtx_ctr
			asynfifo_r1_3
			fifomem_b1_1
			graycntr_r1_3
			synchronizer
	SPI Controller unit (uspi)	SPI Clock Generator block (bclk_gen)	
		SPI Receiver block (bRX)	
		SPI Transmitter block (bTX)	
		FIFO block (bFIFO)	
		SPI Input Output Control block (bio_ctrl)	
	GPIO Controller unit (ugpio)		
	Priority Interrupt Controller unit (upi_ctrl)	Priority Resolver block (bpic_resolver)	
	General Purpose Register unit (ugpr)		
	Boot ROM unit (uboot_rom)		
	Memory Arbiter unit (umem_arbiter)		
	PR controller unit (upr_ctrl)		
	De-coupler unit (udecoupler)		

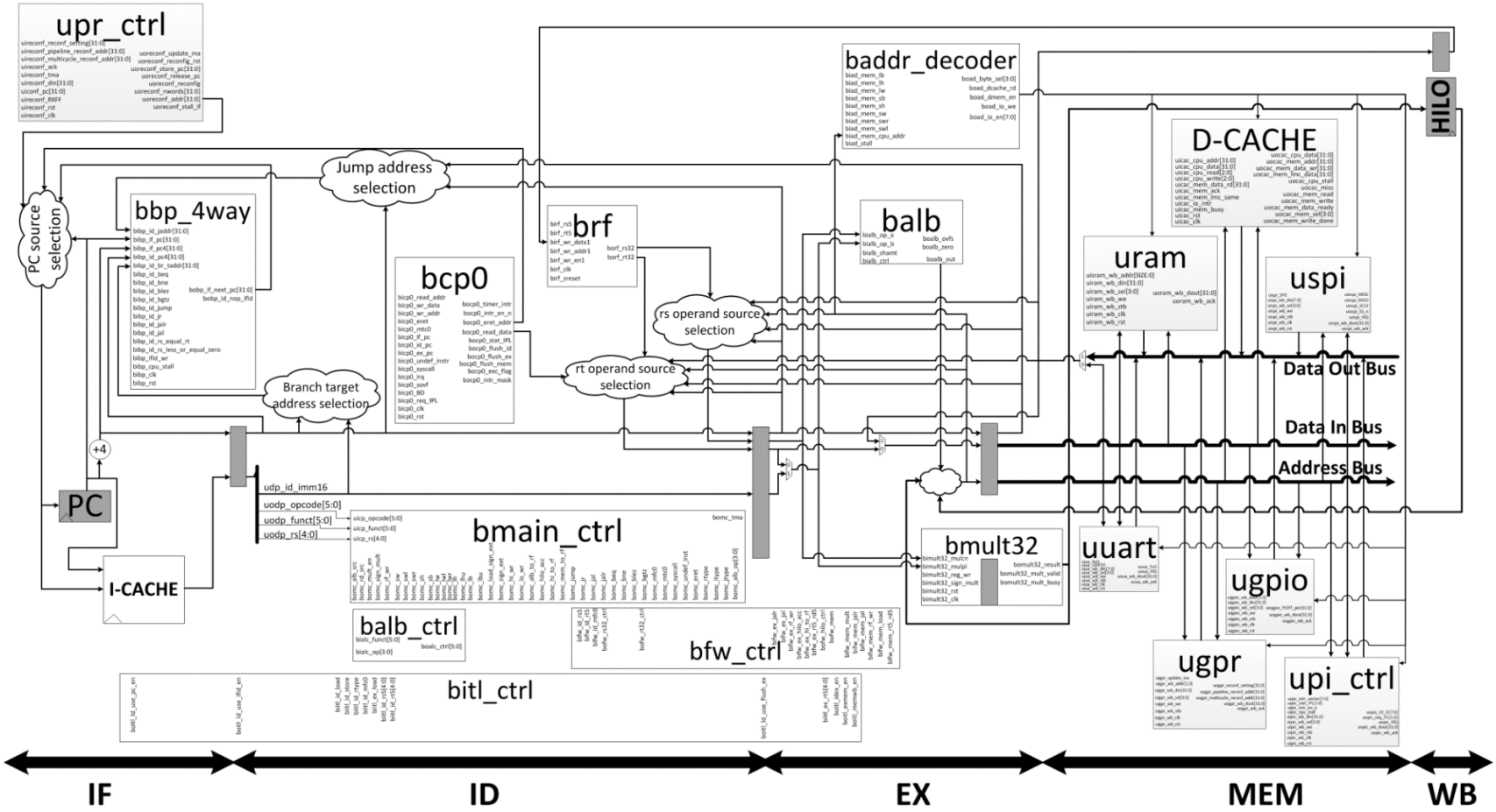


Figure 3.4: 5-stage pipeline processor microarchitecture (functional view)

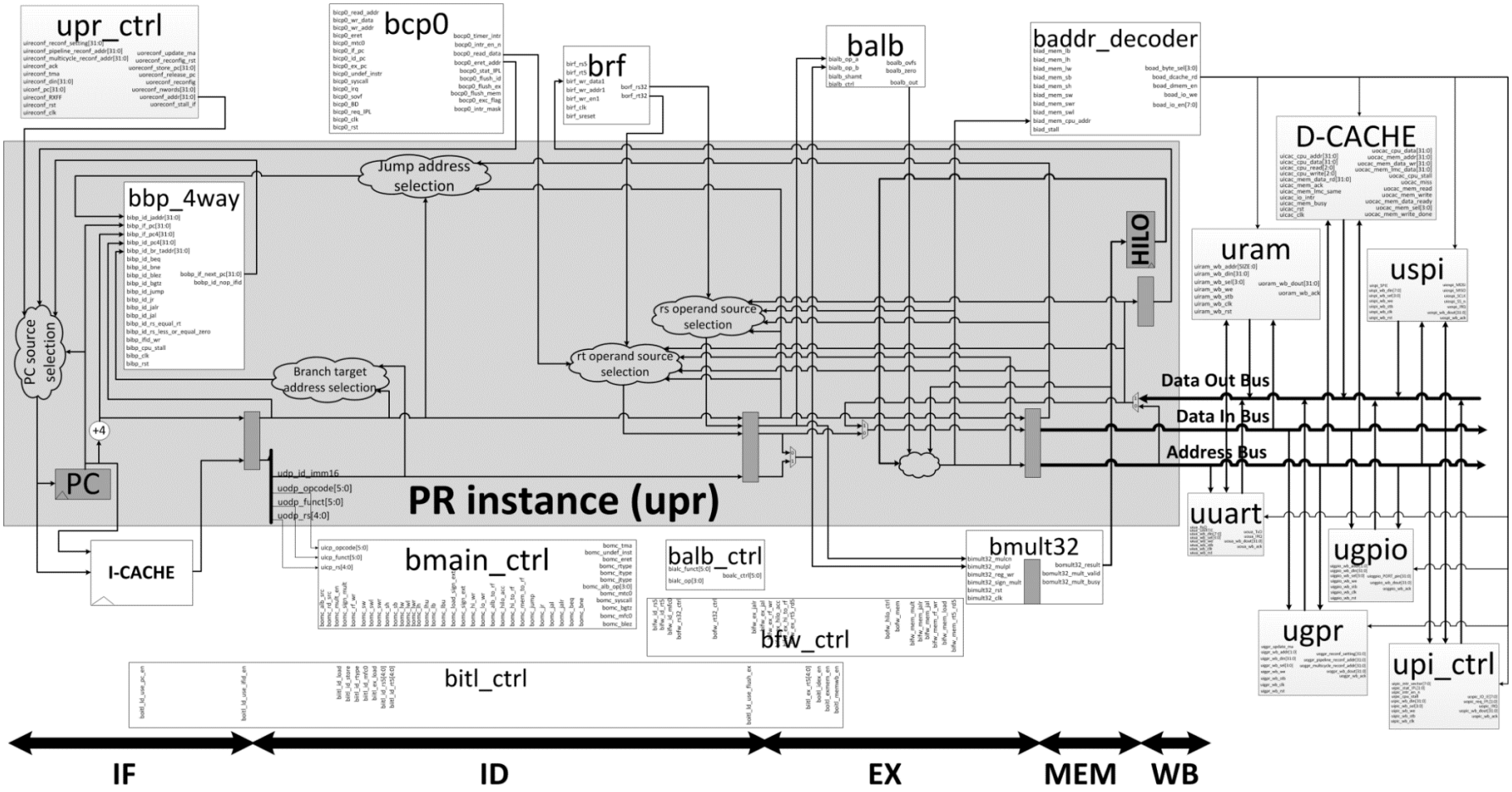


Figure 3.5: Design restructuring of 5-stage pipeline processor microarchitecture for PR purposes

3.2.3 Multi-cycle microarchitecture

Multi-cycle microarchitecture is developed based on the same 5-stage instruction execution cycle as the pipeline microarchitecture. However, it requires an instruction to execute in several clock cycles prior to the end of its execution, which varies from 2 to 5 clock cycles. Each instruction in the multi-cycle execution must complete its execution before the consecutive instruction start its execution, i.e. non-overlapping in execution. Figure 3.6 illustrates the difference between both multi-cycle and pipeline executions.

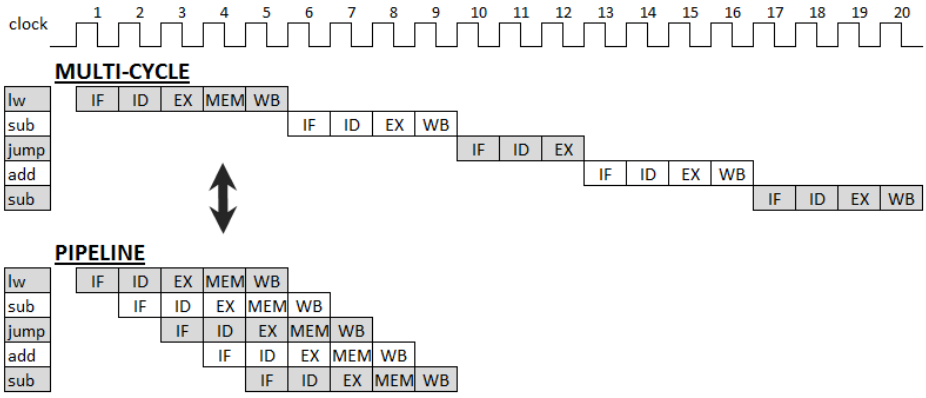


Figure 3.6: Difference between multi-cycle and pipeline executions

From Figure 3.6, multi-cycle execution requires 20 clock cycles to complete a set of 5 instructions, whereas pipeline execution takes only 9 clock cycles. Since instruction execution is non-overlapping, there is no data hazard and hence, no extra circuitries (Forwarding block and Interlock block used in the pipeline microarchitecture) are required. Besides that, the characteristic of the multi-cycle microarchitecture, i.e. only one stage is active at every clock cycle, allows some components to be reused. This leads to lesser components used and hence, lesser signals switching, which dramatically reduces the

design area and dynamic power consumption. To process varying instruction execution stages (from 2 to 5 clock cycles), a Moore Model Finite State Machine (FSM) based Control-path unit is developed, which will be discussed in Section 3.2.3.1. The reason to use a Moore model FSM instead of a Mealy model FSM is due to the design behavior exists of glitches in the Mealy model FSM. Glitches are the unnecessary signal switching that will consume dynamic power. A glitches output will be used by the other hardware modules in the reconfigurable IoT processor and thus, creating more glitches. This is opposite to our intention as to reduce the dynamic power of the multi-cycle microarchitecture. The design hierarchy of the multi-cycle processor is shown in Table 3.4. Figure 3.7 shows the microarchitecture of the multi-cycle processor. The design restructuring of the multi-cycle processor microarchitecture for PR purposes is shown in Figure 3.8.

Table 3.4: Multi-cycle microarchitecture design hierarchy

Chip Level	Unit Level (Microarchitecture Level)	Block Level (Microarchitecture Level)	Sub-block
crisc	Data-path unit (udata_path)	Register File block (brf)	
		Forwarding block (bfw_ctrl)	
		Interlock block (bitl_ctrl)	
		CP0 block (bcp0)	
		ALU block (balb)	
		Multiplier Block (bmult32)	adder_lvl1_firstrow
			adder_lvl1
			add_lvl1_lastrow
			sub_lvl1_lastrow
			adder_lvl2
			adder_lvl2_lastrow
			adder_lvl3
			adder_lvl4
			adder_lvl5
		Address Decoder block (baddr_decoder)	
	Control-path unit FSM (uctrl_path)	Main Control block (bmain_ctrl)	
		Arithmetic Logic Control block (balb_ctrl)	

Continued from Table 3.4

	Cache unit (ucache)	Cache Controller block (bcache_ctrl)	
		Cache RAM block (bcache_ram)	
	Flash Controller Unit (ufc)	Flash Controller Clock Generator block (bfc_clk_gen)	
		Flash Controller FSM block (bfc_fsm)	
		Flash Controller Transmitter block (bfc_TX)	
		Flash Controller Receiver block (bfc_RX)	
		FIFO block (bfc_FIFO)	
	Data and Stack RAM unit (uram)		
	UART Controller unit (uuart)	UART Baud Clock Generator block (bclkctr)	
		UART Receiver block (brx)	sbrx_ctr
			asynfifo_r1_3
			fifomem_b1_1
			graycntr_r1_3
			synchronizer
		UART Transmitter block (btx)	sbtx_ctr
			asynfifo_r1_3
			fifomem_b1_1
			graycntr_r1_3
			synchronizer
	SPI Controller unit (uspi)	SPI Clock Generator block (bclk_gen)	
		SPI Receiver block (bRX)	
		SPI Transmitter block (bTX)	
		FIFO block (bFIFO)	
		SPI Input Output Control block (bio_ctrl)	
	GPIO Controller unit (ugpio)		
	Priority Interrupt Controller unit (upi_ctrl)	Priority Resolver block (bpic_resolver)	
	General Purpose Register unit (ugpr)		
	Boot ROM unit (uboot_rom)		
	Memory Arbiter unit (umem_arbiter)		
	PR controller unit (upr_ctrl)		
	De-coupler unit (udecoupler)		

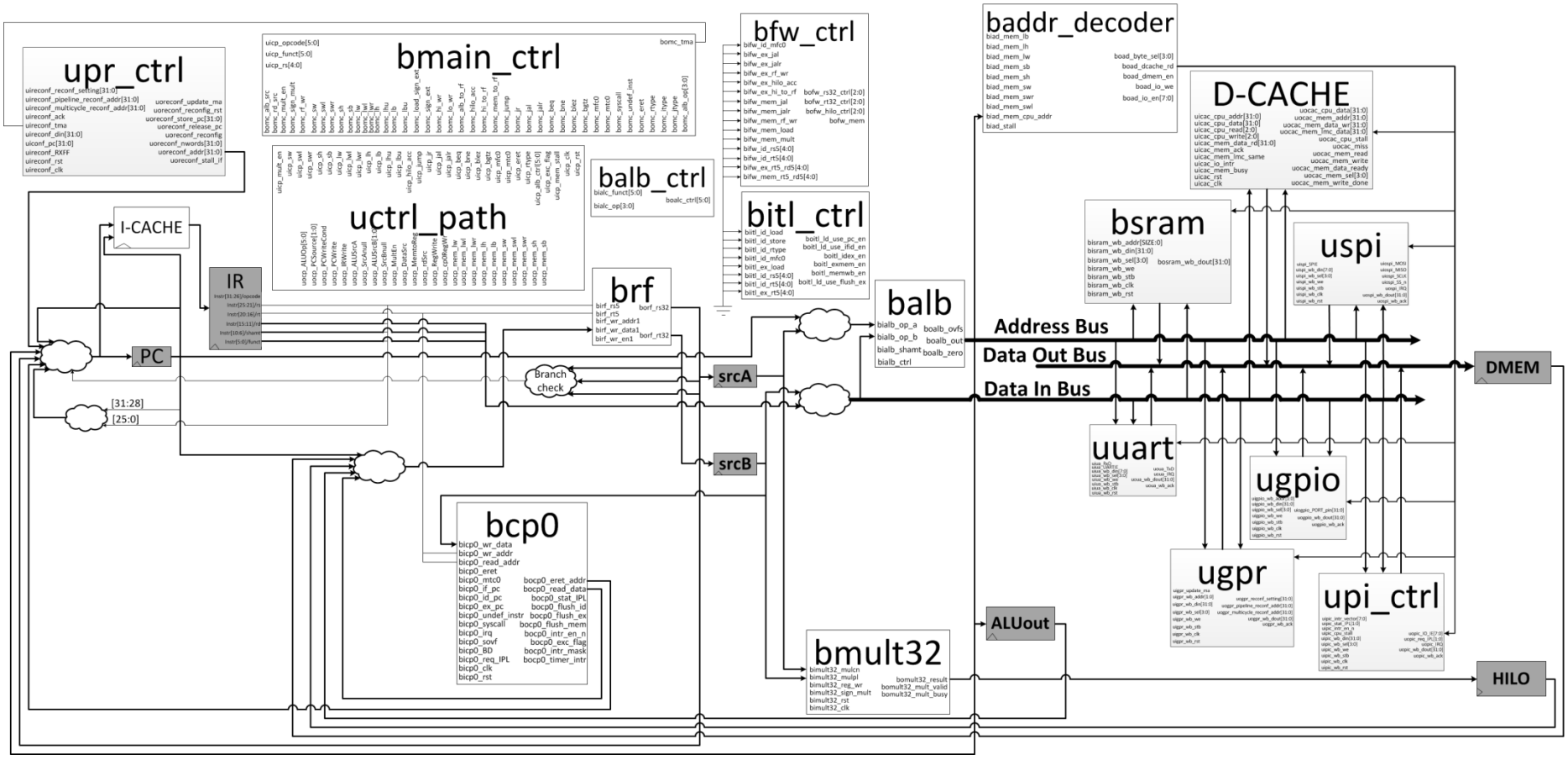


Figure 3.7: Multi-cycle processor microarchitecture

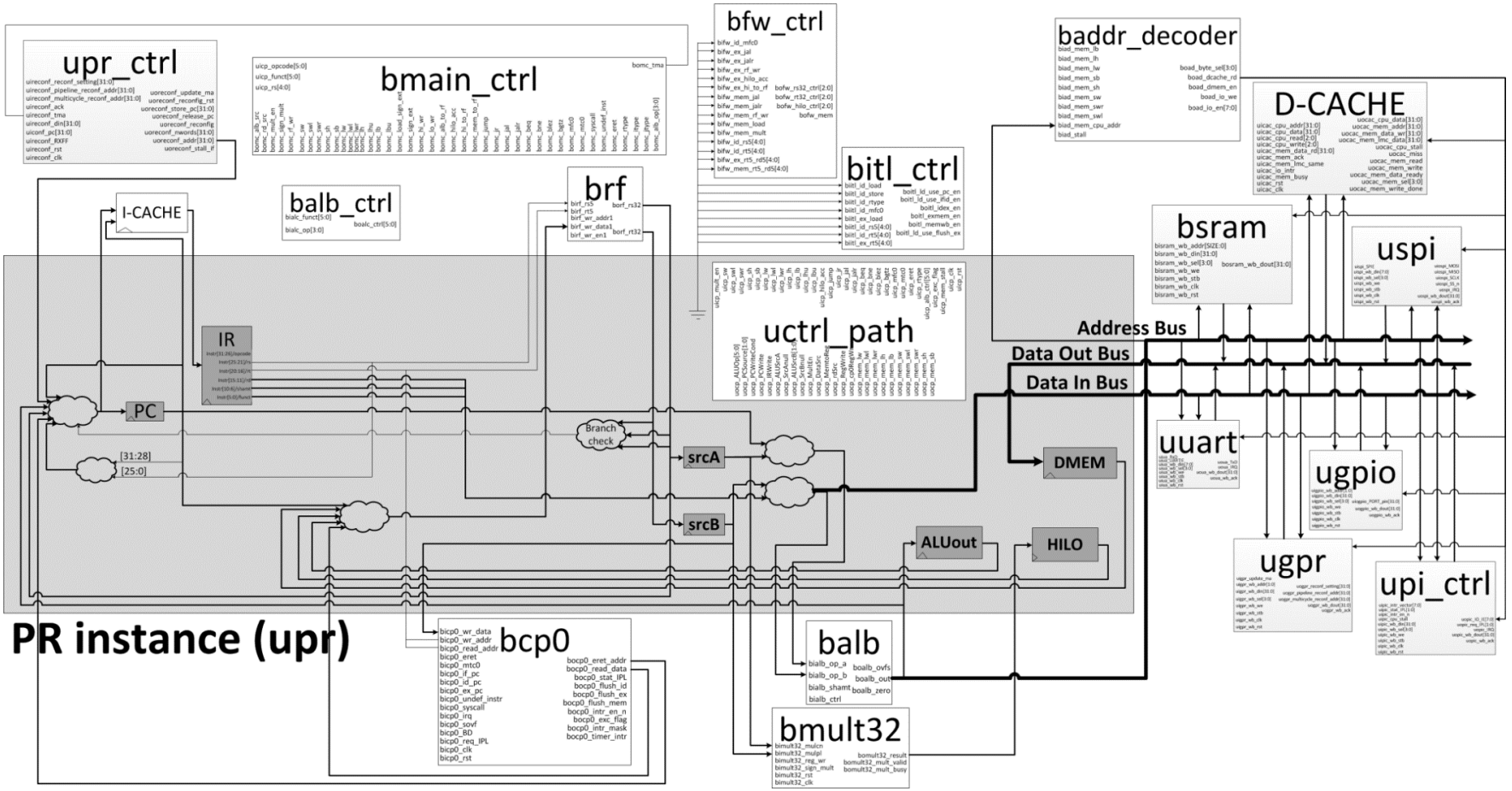


Figure 3.8: Design restructuring of multi-cycle processor microarchitecture for PR purposes

3.2.3.1 Control-path unit FSM (for multi-cycle microarchitecture)

There are a total of 20 states in the Control-path unit FSM as shown in Figure 3.9. Each instruction has its own instruction flow to follow as shown in Table 3.5. The information of each state is described in Table 3.6, while Figure 3.10 illustrates the connection of the Control-path unit FSM with the Main Control Block and the Arithmetic Logic Control Block.

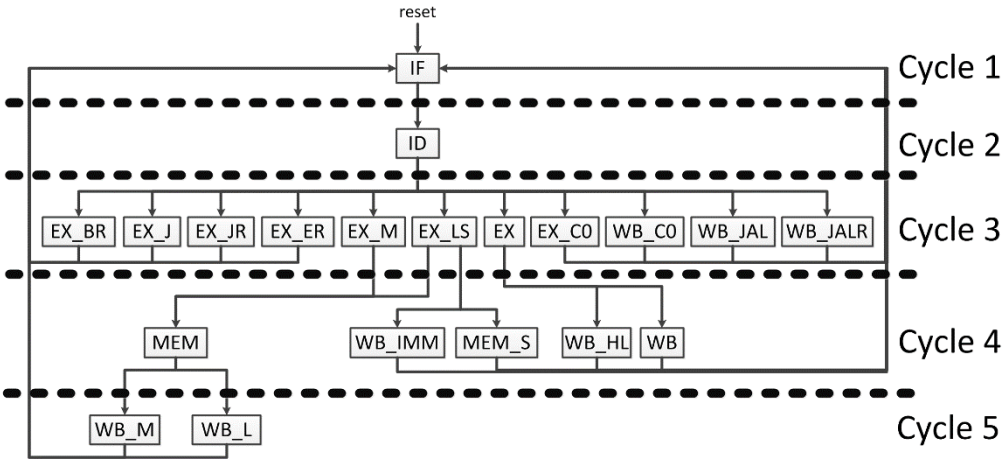


Figure 3.9: 20 states of the multi-cycle microarchitecture Control-path unit FSM

Table 3.5: Instruction cycles and corresponding state required by instruction

No	Instruction	Instruction cycles	State				
			1	2	3	4	5
1	add	4	IF	ID	EX	WB	
2	addu	4	IF	ID	EX	WB	
3	sub	4	IF	ID	EX	WB	
4	subu	4	IF	ID	EX	WB	
5	mult	5	IF	ID	EX_M	MEM	WB_M
6	multu	5	IF	ID	EX_M	MEM	WB_M
7	mfhi	4	IF	ID	EX	WB_HL	
8	mflo	4	IF	ID	EX	WB_HL	
9	and	4	IF	ID	EX	WB	
10	or	4	IF	ID	EX	WB	
11	xor	4	IF	ID	EX	WB	
12	nor	4	IF	ID	EX	WB	
13	sll	4	IF	ID	EX	WB	
14	srl	4	IF	ID	EX	WB	

Continued from Table 3.5

15	sra	4	IF	ID	EX	WB	
16	slt	4	IF	ID	EX	WB	
17	sltu	4	IF	ID	EX	WB	
18	jr	3	IF	ID	EX_JR		
19	jalr	3	IF	ID	WB_JALR		
20	syscall	2	IF	ID			
21	mtc0	3	IF	ID	EX_C0		
22	mfc0	3	IF	ID	WB_C0		
23	eret	3	IF	ID	EX_ER		
24	addi	4	IF	ID	EX_LS		
25	addiu	4	IF	ID	EX_LS		
26	andi	4	IF	ID	EX_LS		
27	ori	4	IF	ID	EX_LS	WB_IMM	
28	xori	4	IF	ID	EX_LS	WB_IMM	
29	lui	4	IF	ID	EX_LS	WB_IMM	
30	lw	5	IF	ID	EX_LS	MEM	WB_L
31	lwl	5	IF	ID	EX_LS	MEM	WB_L
32	lwr	5	IF	ID	EX_LS	MEM	WB_L
33	lh	5	IF	ID	EX_LS	MEM	WB_L
34	lhu	5	IF	ID	EX_LS	MEM	WB_L
35	lb	5	IF	ID	EX_LS	MEM	WB_L
36	lbu	5	IF	ID	EX_LS	MEM	WB_L
37	sw	4	IF	ID	EX_LS	MEM_S	
38	swl	4	IF	ID	EX_LS	MEM_S	
39	swr	4	IF	ID	EX_LS	MEM_S	
40	sh	4	IF	ID	EX_LS	MEM_S	
41	sb	4	IF	ID	EX_LS	MEM_S	
42	slti	4	IF	ID	EX_LS	WB_IMM	
43	sltiu	4	IF	ID	EX_LS	WB_IMM	
44	beq	3	IF	ID	EX_BR		
45	bne	3	IF	ID	EX_BR		
46	blez	3	IF	ID	EX_BR		
47	bgtz	3	IF	ID	EX_BR		
48	j	3	IF	ID	EX_J		
49	jal	3	IF	ID	WB_JAL		
50	tma	2	IF	ID			

Table 3.6: State definition of the multi-cycle microarchitecture Control-path unit FSM

State Name	**Definition**
IF	Instruction fetch from instruction memory
ID	Instruction Decode and Register File Read
EX_BR	Determine branch taken or untaken. Copy branch target address calculated to the PC register if branch taken
EX_J	j instruction detected. Copy jump address to PC register
EX_JR	jr instruction detected. Copy $rs register value to PC register
EX_ER	eret instruction detected. Copy the exception return address, $epc to PC register
EX_C0	mtc0 instruction detected. Move a data from Register File to CP0 register
WB_C0	mfc0 instruction detected. Copy a data from CP0 register to Register File

Continued from Table 3.6

WB_JAL	jal instruction detected. Copy PC register to $ra register in the Register File and copy jump address to PC register
WB_JALR	jal instruction detected. Copy PC register to $ra register in the Register File and copy $rs register data to PC register
EX_M	jal instruction detected. Activate multiplier
EX_LS	I-type instruction detected. Further decode whether load, store or immediate instruction is issued
EX	R-type instruction detected. Activate ALU.
MEM	Load data from data memory (D-CACHE, RAM or I/Os register) or perform as dummy cycle for multiplication operation
WB_IMM	Write back immediate instruction result to Register File
MEM_S	Store data to data memory (D-CACHE, RAM or I/Os register)
WB_HL	Write back HI or LO register to Register File
WB	Write back R-type instruction result to Register File
WB_M	Reserved for write back multiplication result to HILO register
WB_L	Write back data memory (D-CACHE, RAM or I/Os register) data to Register File

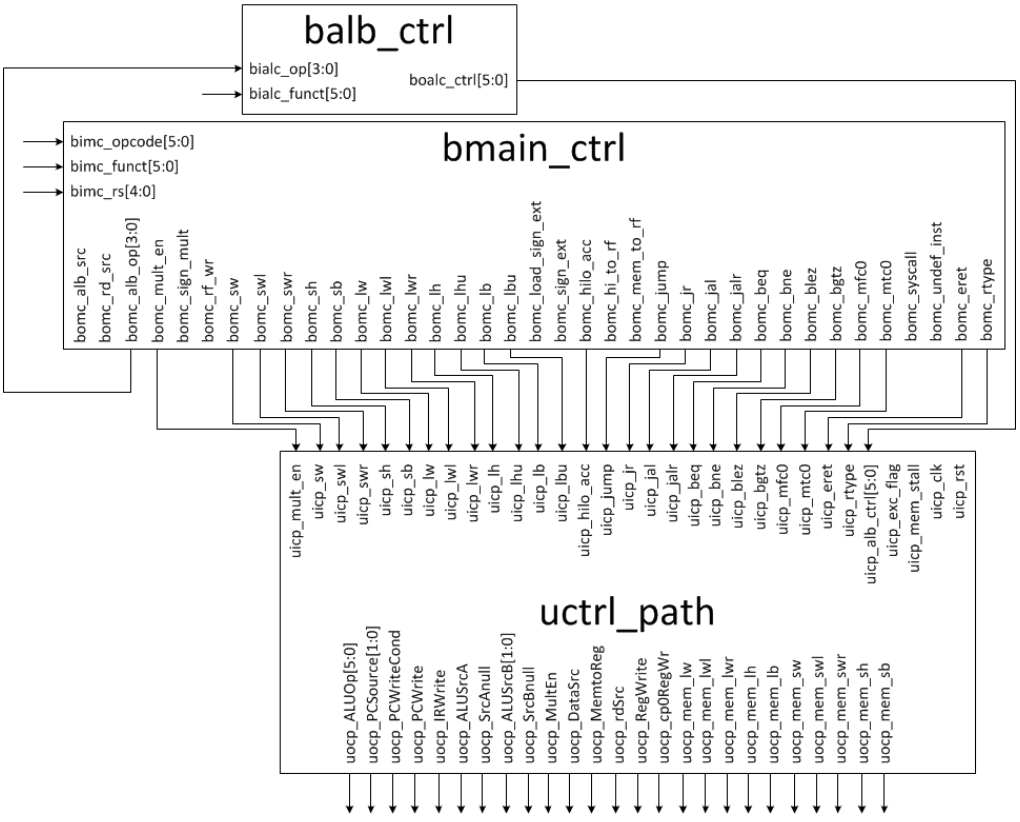


Figure 3.10: Connection of the Control-path unit FSM with the Main Control Block and the Arithmetic Logic Control Block for Multi-cycle microarchitecture

3.2.4 Consistent I/O Interface for Partial Reconfiguration Unit

PR unit (a.k.a PR instance), either multi-cycle or pipeline microarchitecture, is model as a module in RTL modeling. The I/O pins of the PR instance, known as partition pins (Xilinx, 2016a), must be consistent for both multi-cycle and pipeline microarchitectures. The partition pins serve as the static interconnection pins between the static region's logic and the PR region's logic. The partition pins can be: 1) user defined by including the *(* keep = "true" *)* command in the RTL modeling of the PR unit to avoid the optimization of the partition pins; 2) automatically created by Xilinx Vivado, where routing congestion may occur when the PR unit has a large amount of partition pins. For our reconfigurable soft-core IoT processor, the first method is used in order to gain control on the routing and the logic resources placement. Figure 3.11 illustrates the PR unit partition pins that are consistent for both multi-cycle and pipeline microarchitectures. Each partition pin's function is described in Table A.1.

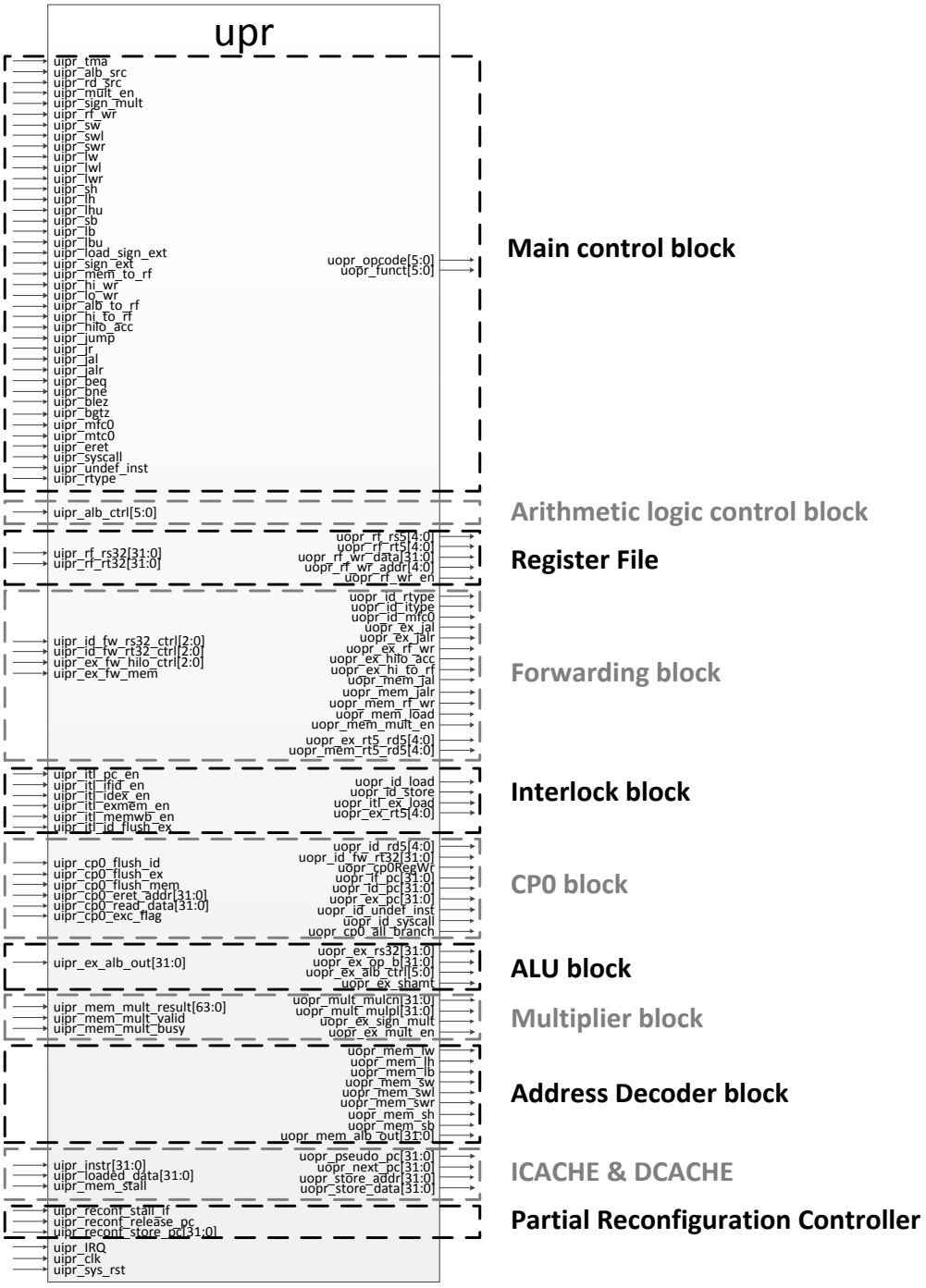


Figure 3.11: Partition pins of Partial Reconfiguration top module

3.2.5 Partial Reconfiguration

PR is initiated when a *tma* instruction is detected in the user program code, which as shown in the program code in Figure 3.12.

```
#define SETTING 0xBFFFFE00
#define P5ADDR  0xBFFFFE04
#define M5ADDR  0xBFFFFE08

void main(){
    SETTING = 0x0A8FD400; 0xC7EA << 9
    P5ADDR = 0x00A80000;
    M5ADDR = 0x00A00000;
    while(1){
        // 1st job - multi-cycle execution
        /*
        data collection: acquire sensor data
         */

        __asm__("tma;");

         //2nd job - pipeline execution
         /*
         data processing:
         band pass filter
         AES encryption
         */

        __asm__("tma;");

        data transmission: Send through wireless module
    }
}
```

Figure 3.12: Sample test program to initiate the PR

PR will reconfigure the processor:

1) When current microarchitecture is multi-cycle microarchitecture, PR will reconfigure the processor to pipeline microarchitecture
2) When current microarchitecture is pipeline microarchitecture, PR will reconfigure the processor to multi-cycle microarchitecture

The currently configured processor will halt its execution and pass the control to the PR controller, which shown in Figure 3.1. The PR controller obtains the partial bitstream which was earlier stored in the flash memory (on the Nexys 4 DDR board). Each 32-bits word of bitstream will be written to the FPGA

through Internal Configuration Access Port (ICAP). The process finishes when the last bitstream word writes into the FPGA, perform a soft reset on the reconfigurable region and continue to fetch the instruction following the *tma* instruction in the user program code. The flow chart of the PR process is shown in Figure 3.13.

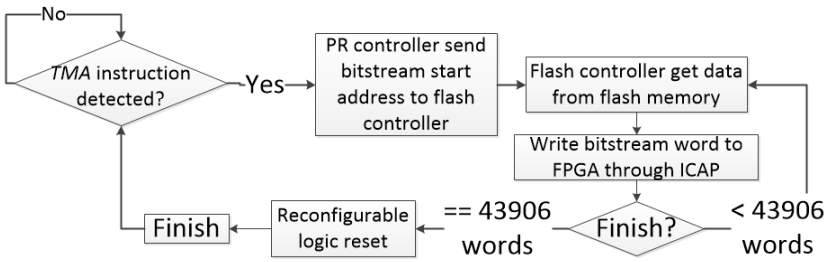


Figure 3.13: PR process flow

In pipeline microarchitecture, when *tma* instruction is detected at the ID stage, the execution of the instruction following *tma* instruction, which is already in IF stage, will be halt by the PR controller. At the same time, the instructions in MEM and WB stages (instructions prior to *tma* instruction) will continue to execute. Once both instructions in MEM and WB stages end its execution, the PR controller proceeds to partial reconfigure the processor. In the multi-cycle microarchitecture, the PR controller starts to partial reconfigure the processor once *tma* instruction is detected in ID state.

One problem arises when PR is reconfiguring the logic circuits: the output signals from the PR unit will be corrupted. The corrupted signals will be passed to the static region's components. The corrupted signals will affect any data stored before the PR could be completed. However, the data stored

should remain consistent when the PR is in progress. One of the root cause of the issue we have identified occurred in the Main Control block, where bimc_rs[4:0], bimc_funct[5:0] and bimc_opcode[5:0] obtained from the instruction is passed from the PR unit to the Main Control block for instruction decoding. There are 2 possible solutions to resolve the issue: 1) deassert the 17-bit inputs of Main Control block (bimc_rs[4:0], bimc_funct[5:0] and bimc_opcode[5:0]); 2) deassert the output signals from the Main Control block that can affect the data stored (data in the Register File, CP0 registers, Data and Stack RAM, I/O controller registers). We used the second solution since lesser signals were identified (bomc_eret and bomc_tma), which shown in Table 3.7. Besides that, several signals (no. 2 to no. 13 in Table 3.7) from the PR unit were identified to have affected the data stored when PR is in progress. This is due to the signals are output from the pipeline registers (pipeline microarchitecture) or produce by the internal circuitry, i.e. uctrl_path (multi-cycle microarchitecture). We have also found that the ALU block will output a corrupted signal (boalb_ovfs) that will trigger an exception. Therefore, a de-coupler block is developed to deassert the corrupted signals to ground when PR is in progress. Table 3.7 shows the corrupted signals that will be connected to the de-coupler block.

Table 3.7: Corrupted signals to be de-coupled when PR is in progress

No	Signal name	Source	Destination	Signal function
1	bomc_eret	Main Control block	CP0 block	Indicate eret instruction is executing
2	bomc_tma	Main Control block	PR controller	Indicate tma instruction is executing, PR will take place to reconfigure the PR unit
3	uopr_mem_lw	PR unit	Address Decoder block	Indicate lw, lwl or lwr instruction in MEM stage

Continued from Table 3.7

4	uopr_mem_lh	PR unit	Address Decoder block	Indicate lh or lhu instruction in MEM stage
5	uopr_mem_lb	PR unit	Address Decoder block	Indicate lb or lbu instruction in MEM stage
6	uopr_mem_sw	PR unit	Address Decoder block	Indicate sw, swl or swr instruction in MEM stage
7	uopr_mem_swl	PR unit	Address Decoder block	Indicate swl instruction in MEM stage
8	uopr_mem_swr	PR unit	Address Decoder block	Indicate swr instruction in MEM stage
9	uopr_mem_sh	PR unit	Address Decoder block	Indicate sh instruction in MEM stage
10	uopr_mem_sb	PR unit	Address Decoder block	Indicate sb instruction in MEM stage
11	uopr_rf_wr_en	PR unit	Register File block	Enable write to Register File
12	uopr_cp0_all_branch	PR unit	CP0 block	Indicate eret, beq, bne, blez, bgtz, j, jr, jal or jalr instruction is executing
13	uopr_cp0RegWr	PR unit	CP0 block	Enable write to CP0 register
14	boalb_ovfs	ALU block	CP0 block	Indicate sign overflow has occur

Since PC register is placed in the PR region to decrease the net delay (longer net routing used by the Xilinx Place and Route tool when PC register is place in the static region), the PC register value (PC register holds the address of the instruction next to *tma* instruction) in the PR unit should not change when the PR is in progress. This will allow the processor to re-execute the consecutive instruction after the PR is done. In order to resolve this issue, when *tma* instruction is detected, the PC register value will be sent to the PR controller. The PR controller holds the PC register value so that no consecutive instruction is fetched when the PR is in progress.

3.3 Memory System

The developed memory system is a 2-level memory hierarchy. First level consists of the caches (I-CACHE with 1-kBytes RAM and D-CACHE with 128-Bytes RAM), Data and Stack RAM and Boot ROM, while second level consists of the flash memory. The flash memory is a non-volatile memory that used to store data (data content include FPGA configuration bitstream, program code, constant variables and PR bitstream) when the power is shut off. In contrast, RAM based cache losses all the data when the power is shut off. Cache benefits in fast memory access, which can be used to enhance the speed of memory access from the flash memory by buffered part of the data in the cache.

Caching the runtime data (*.data* and *.stack* segments) in flash memory is avoided. The flash memory has some limitation, in which a minimum sector size of 4-kBytes must be erase (the data in the respective 4-kBytes sector will be reset to 1 and the data is update by changing the related bit from 1 to 0) prior to data updating. This requires the use of an additional RAM of at least 4-kBytes just only for swapping purpose (to temporary hold the flash memory data before the sector is erased for updating purpose). Also, it consumes additional clock cycles for the data to be read out from the flash memory and write back. To overcome this issue, an 8-kBytes of Data and Stack RAM are created using the FPGA Block RAM to hold the runtime data (refer to *.data*, *.bss*, *.stack* and *.heap* segments in Section 3.3.1) without the need to write back to the flash memory. If needed to store the runtime data, the data will be directed to an external non-volatile memory via UART or SPI. We

employed a D-CACHE with 128-Bytes of RAM to read the *.data* and *.rodata* segments (refer to *.data* and *.rodata* segment in Section 3.3.1) from the flash memory. A Boot ROM (Read-only memory, the bootloader program is pass to the Boot ROM using "$readmemh (`ROM_FILE_PATH, rom_data)" in the Verilog HDL) is integrated with the CPU for bootloading purposes. The architecture and the microarchitecture of the developed memory system are shown in Figure 3.14 and Figure 3.15.

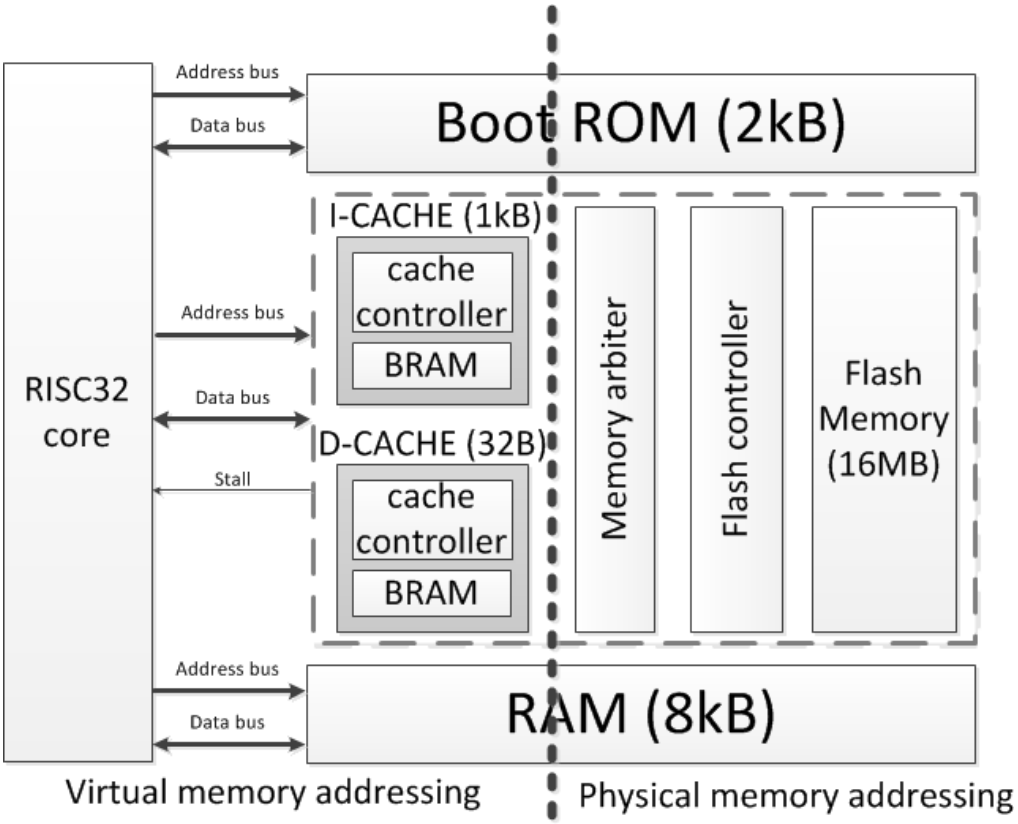


Figure 3.14: Memory system architecture

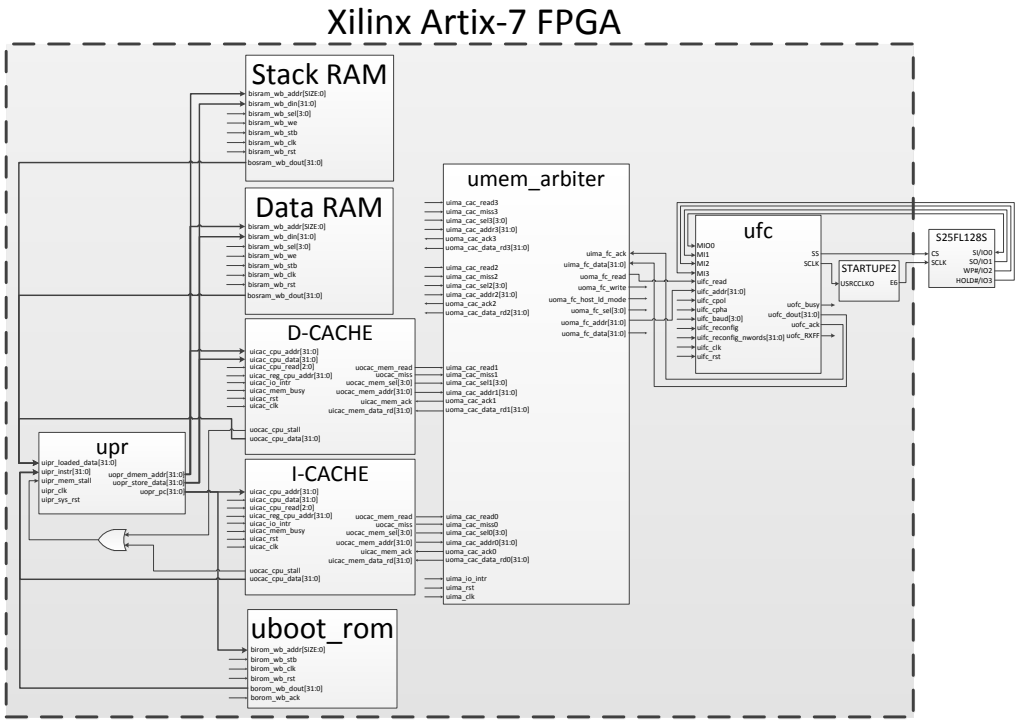


Figure 3.15: Memory system microarchitecture

The development of the memory system covers 5 RTL components, an FPGA component (STARTUPE2) and a main memory (flash memory). The RTL components include the development of Data and Stack RAM, Boot ROM, caches (I-CACHE and DCACHE), memory arbiter and Flash controller. Memory arbiter is used to handle the data communication of the highest priority cache with the flash memory among multiple caches. Flash Controller with quad SPI interface is developed for the ease of flash memory (S25FL128S) data access. The FPGA component (STARTUPE2) in the Xilinx XC7A100T FPGA is hardwired connected to the S6 output pin of the XC7A100T FPGA chip. The user defined SCLK from the Flash Controller is meant to drive the S25FL128S flash memory, indirectly via STARTUPE2 (S6 pin). When the FPGA board is powered up, the S25FL128S flash memory is

automatically controlled by the FPGA for configuration purposes. 3 dummy SCLK cycles are required to send to the STARTUPE2 component before the SCLK, which was earlier controlled by the FPGA, can be passed back to user controlled. Thereby, users can fully control the S25FL128S flash memory.

3.3.1 Memory Map

This section starts with introducing MIPS memory space convention (D. Sweetman, 2006). MIPS with 32-bits addresses, is allowed to support up to 4GB memory space. The MIPS memory address space is implemented in two ways: virtual and physical addresses. Virtual addresses are used by the CPU for the instruction and data accessing. While physical addresses are used to allocate with physical memory such as flash memory, Data and Stack RAM, Boot ROM and I/O registers. Figure 3.16 shows the virtual to physical memory mapping.

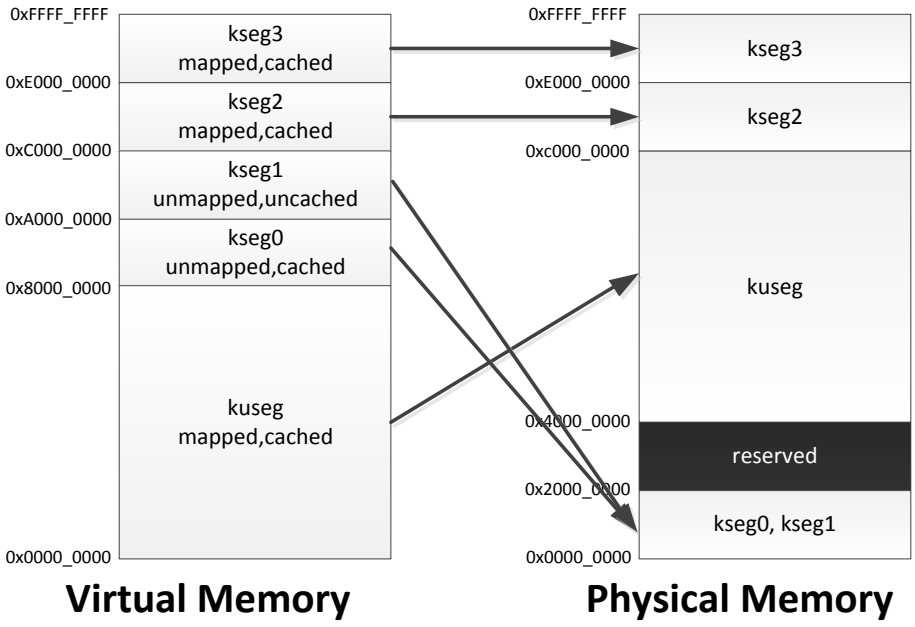


Figure 3.16: Virtual to physical memory mapping based on 32-bit MIPS architecture. The mapped memory segment is mapped to the Memory Management Unit (MMU) while the cached segment used the cache memory to enhance the data accessing speed.

From Figure 3.16, the virtual memory is distributed into 5 segments: kernel user segment (*kuseg*), kernel segment 0 (*kseg0*), kernel segment 1

(*kseg1*), kernel segment 2 (*kseg2*) and kernel segment 3 (*kseg3*). *kuseg*, *kseg2* and *kseg3* are mapped segment, which includes the address translation using Memory Management Unit (MMU). D. Sweetman (2006) suggested avoid using the mapped segment for the processor with no Memory Management Unit (MMU), which left *kseg0* and *kseg1* for our implementation. The *kseg0* and *kseg1* have the same physical addresses with different virtual addresses, except that the *kseg0* is accessed through the cache. Figure 3.17 shows our implementation of memory allocation on *kseg0* and *kseg1*.

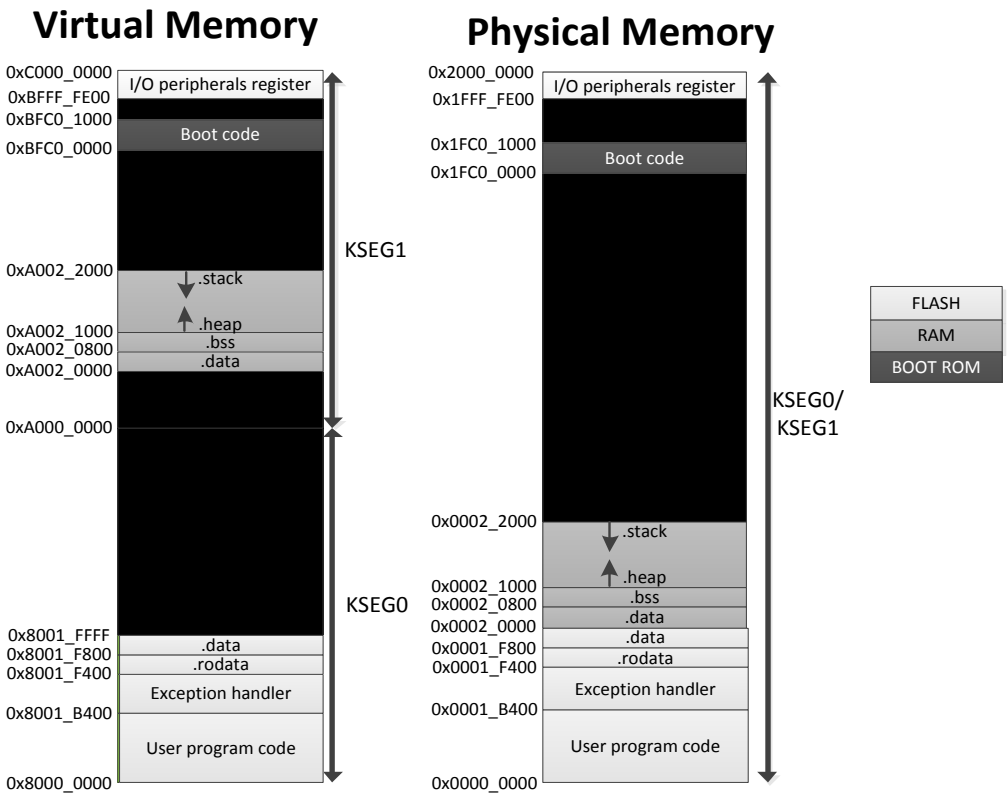


Figure 3.17: Memory allocation on *kseg0* and *kseg1*

A C program memory separates into user program (.*text*), initialized data (.*data*), uninitialized data (.*bss*), read-only constant data (.*rodata*), stack

data (.*stack*) and heap data (.*heap*) (Gu, C., 2016 and I. C. Bertolotti and Tingting Hu., 2015). The .*text* used to store the user program code addressed using the PC register. The .*data* is used to store the initialized data, i.e. the variables that are initialized with values. When CPU power start, the bootloader is responsible to copy the .*data* content in the flash memory to the .*data* segment in the Data RAM. In contrary, .*bss* segment stores the uninitialized data and is allocated only in the Data RAM. The .*stack* is a Last-In-First-Out (LIFO) queue used to store a procedure or function information and local variables. The .*heap* is used as the dynamically allocated memory space requested using *malloc()* function.

When a processor startup, the bootloader program stored in the Boot ROM should perform the following actions:

1) Set up the Register File block registers value
2) Copy .*data* content from flash memory to the Data RAM
3) Jump to user program code located at 0x8001_B400 (virtual address)

Accessing .*data* from the Data RAM instead of the flash memory is to enhance the data accessing performance (flash memory data access is in serial form, while Data RAM is in parallel). The data in .*data*, .*bss*, .*stack*, .*heap* and I/O peripherals registers can be accessed using load and store instructions.

3.3.2 Cache Unit

The developed Cache unit is a direct mapped cache. The cache is used to store the copy of data (virtual memory address location from 0x8001_F400 to 0x8001_F7FF for D-CACHE) or instructions (virtual memory address location from 0x8000_0000 to 0x8001_F3FF for I-CACHE) with the size of 8-words per block from the flash memory. Figure 3.18 shows the chip interface of the Cache unit and Table A.2 describes the function of each pin.

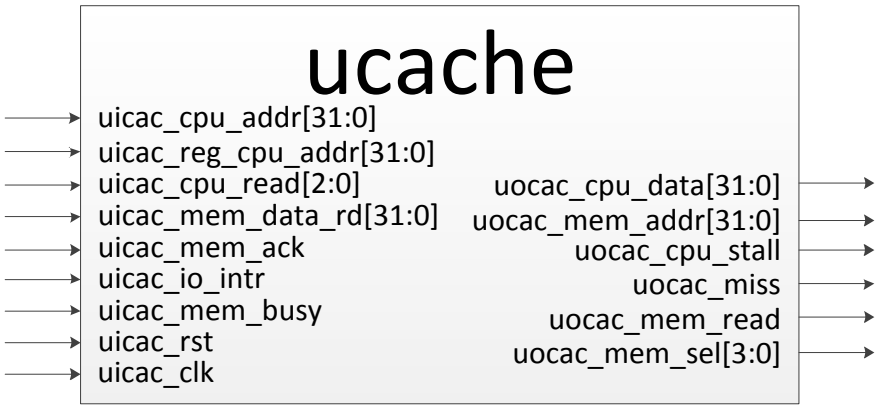


Figure 3.18: Cache unit chip interface

3.3.2.1 Cache protocol

The Cache unit is organized in a multiword block (8-words in one block) as shown in Figure 3.19 (the I-CACHE has 32 indexes while the D-CACHE has 4 index). Each index carries a tag of the address, 1 valid bit and 8 words of data. The tag is used as the unique identifier of the address so that to reduce the size of the comparator circuit instead of comparing 32-bit of the input address. The valid bit is used to identify the validity of the block of data. All the valid bit of the block in the cache is de-asserted when the processor reset and each bit is asserted after performing a read operation. Figure 3.19 illustrates the cache organization, with assuming with 32 indexes.

Index	Valid	Tag	Data	Data	Data	Data	Data	Data	Data	Data
0										
1										
2										
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
30										
31										
5-bit	1-bit	22-bit	32-bit	32-bit	32-bit	32-bit	32-bit	32-bit	32-bit	32-bit

Figure 3.19: Direct mapped cache organization with a cache block size of 8-words

The cache operation is divided into 2, i.e. read hit and read miss. The flow and description for each operation are described in Figure 3.20.

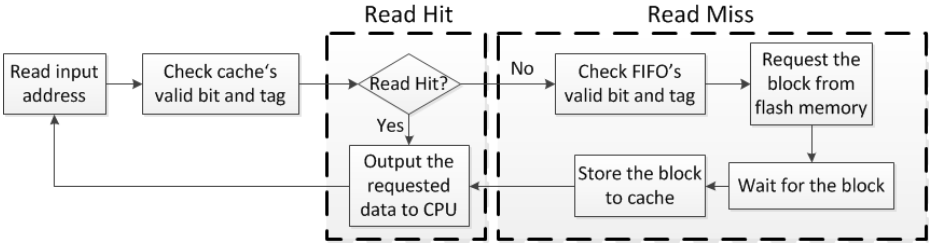


Figure 3.20: Cache read operation

3.3.2.2 Design Partitioning

The Cache unit consists of Cache Controller (bcache_ctrl) and Cache RAM (bcache_ram). The Cache Controller is used to control the data flow and the activity of the Cache unit. Since FPGA is used as the design platform, the FPGA Block RAM is used to implement the memory array (Cache RAM) of the Cache unit. The internal connection of the cache unit is shown in Figure 3.21.

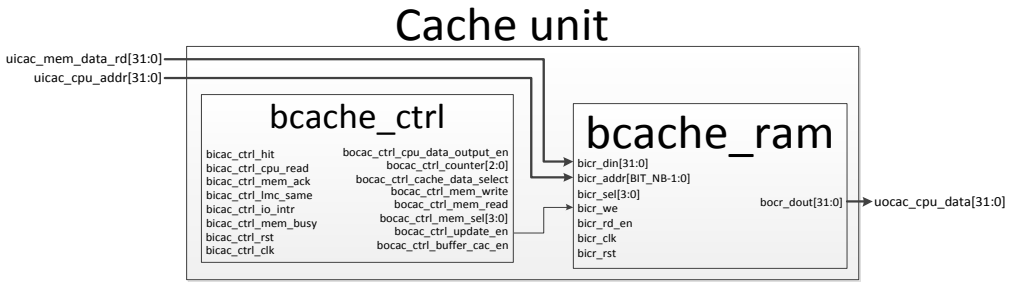


Figure 3.21: Internal connection of the Cache unit

3.3.3 Memory Arbiter Unit

Memory Arbiter unit is used to control the data communication of the multiple caches with the flash memory. Memory Arbiter unit permits the data communication of the highest priority cache with the flash memory while limiting the data communication of the other caches. The chip interface of the Memory Arbiter unit is shown in Figure 3.22 and Table A.3 describes the function of each pin.

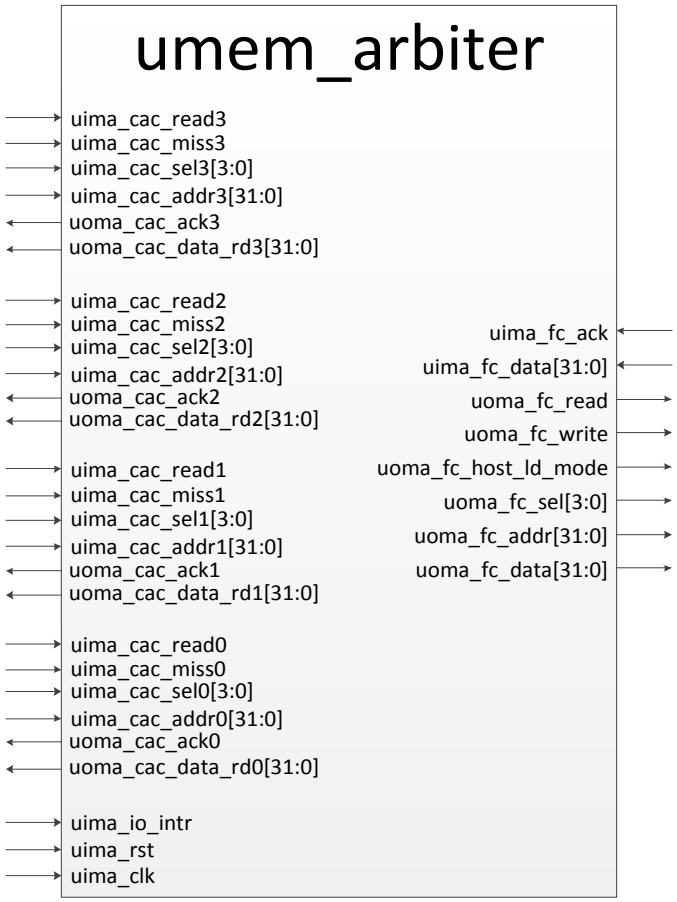


Figure 3.22: Memory Arbiter unit chip interface

The Memory Arbiter Unit is a FSM that consists of 5 states, which is shown in Figure 3.23. The information of each state and the correspondence output of the state are described in Table 3.8.

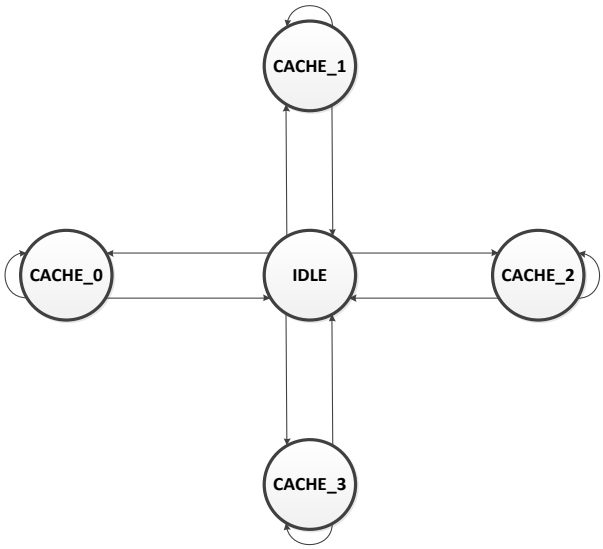


Figure 3.23: Memory Arbiter Unit state diagram

Table 3.8: State definition of the Memory Arbiter Unit

State Name	Definition	Remark
cache3	Highest priority cache given to perform operation	Extra ports for future development on Translation Lookaside Buffer (TLB), Memory Management Unit (MMU) and Operating System (OS)
cache2	Second priority cache given to perform operation	
cache1	Third priority cache given to perform operation	Connected to D-CACHE
cache0	Lowest priority cache given to perform operation	Connected to I-CACHE
idle	No operation	-

3.3.4 Flash Controller Unit

The Flash Controller unit is able to perform the following functionalities:

1) Transmit command instructions to the flash memory serially.
2) 32-bit serial data receiving buffer.
3) 4 SPI modes selectable and 16 speed selectable with up to 10MHz SPI serial communication through RTL/hardware modification. This feature is not software programmable.
4) Act as master device (flash memory as slave device). In the SPI modules connection, master device is responsible to initiate the communication by assert the slave select signal (SS_n), pass the clock signal (SCLK) and transmit the serial data to the slave device (MOSI). Slave device is activated when received the SS_n signal and then pass the serial data to the master device (MISO). The SPI data communication is in full duplex mode, in which data transmission and receiving occur at the same time.

The chip interface of the Memory Arbiter unit is shown in Figure 3.24 and Table A.4 describes the function of each pin.

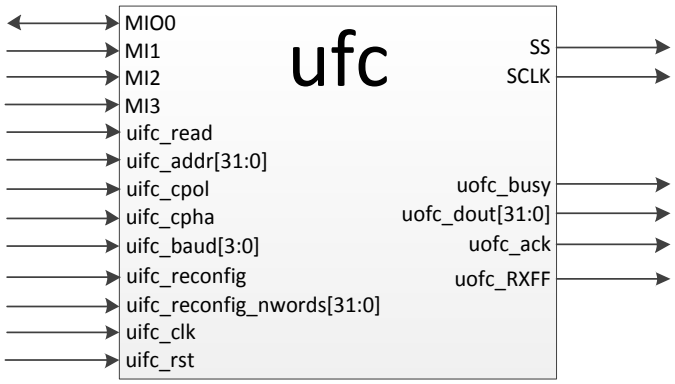


Figure 3.24: Flash Controller unit chip interface

3.3.4.1 Flash Controller Protocol

Typically, flash memory uses SPI interface. Hence, the design of the Flash Controller unit is based on the conventional SPI serial communication protocol shown in Section 3.4.2.1. The Flash Controller unit is developed with quad (one for bi-directional and three for receive) serial data line instead of dual (one for transmit and one for receive) serial data line, to increase the speed of data accessing from the flash memory. The Flash Controller unit is developed to support the flash memory command shown in Table 3.9.

Table 3.9: Supported flash memory command instructions
Source: Cypress (2017) '128 Mbit (16 Mbyte)/256 Mbit (32 Mbyte) 3.0V SPI flash memory'

Command Name	Command Description	Instruction Value (Hex)
RDSR1	Read Status Register-1	05
WRR	Write Register (Status-1, Configuration-1)	01
WREN	Write Enable	06
QOR	Read Quad Out (3- or 4-byte address)	6B

The RDSR1 command is used to read the Status Register-1 of the S25FL128S flash memory. Of particular interest is the Write in Progress (WIP) bit, which shown in Table 3.10. It is used to determine if the user configuration setting has been successfully loaded into the S25FL128S flash memory (to configure the S25FL128S flash memory to quad serial data output mode). The command sequence of the RDSR1 command is shown in Figure 3.25.

Table 3.10: Status Register-1 of S25FL128S flash memory
Source: Cypress (2017) '128 Mbit (16 Mbyte)/256 Mbit (32 Mbyte) 3.0V SPI flash memory'

Bits	Field Name	Function	Type	Default State	Description
7	SRWD	Status Register Write Disable	Non-Volatile	0	1 = Locks state of SRWD, BP, and configuration register bits when WP# is low by ignoring WRR command 0 = No protection, even when WP# is low
6	P_ERR	Programming Error Occurred	Volatile, Read only	0	1 = Error occurred. 0 = No Error
5	E_ERR	Erase Error Occurred	Volatile, Read only	0	1 = Error occurred 0 = No Error
4	BP2	Block Protection	Volatile if CR1[3]=1, Non-Volatile if CR1[3]=0	1 if CR1[3]=1, 0 when shipped from Cypress	Protects selected range of sectors (Block) from Program or Erase
3	BP1				
2	BP0				
1	WEL	Write Enable Latch	Volatile	0	1 = Device accepts Write Registers (WRR), program or erase commands 0 = Device ignores Write Registers (WRR), program or erase commands This bit is not affected by WRR, only WREN and WRDI commands affect this bit
0	WIP	Write in Progress	Volatile, Read only	0	1 = Device Busy, a Write Registers (WRR), program, erase or other operation is in progress 0 = Ready Device is in standby mode and can accept commands

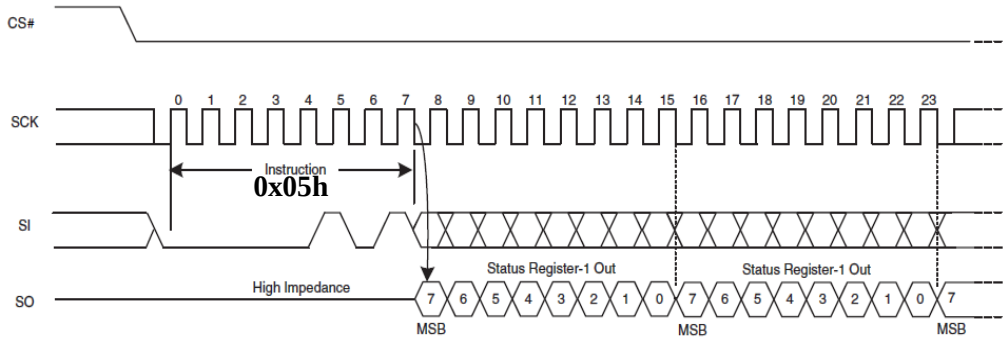


Figure 3.25: RDSR1 command sequence of S25FL128S flash memory
Source: Cypress (2017) '128 Mbit (16 Mbyte)/256 Mbit (32 Mbyte) 3.0V SPI flash memory'

The WRR command is used to writes a new values into the Status Register-1 and the Configuration Register-1 in Table 3.10 and Table 3.11 respectively. In our case, to configure the S25FL128S flash memory to quad serial data output mode, the WRR command follows by the configuration setting, i.e. 0x0002, should be transmit to the S25FL128S flash memory. Table 3.11 shows the register information of the Configuration Register-1 and Figure 3.26 shows the communication sequence of the WRR command.

Table 3.11: Configuration Register-1 of S25FL128S flash memory Source: Cypress (2017) '128 Mbit (16 Mbyte)/256 Mbit (32 Mbyte) 3.0V SPI flash memory'

Bits	Field Name	Function	Type	Default State	Description
7	LC1	Latency Code	Non-Volatile	0	Selects number of initial read latency cycles See Latency Code Tables
6	LC0			0	
5	TBPROT	Configures Start of Block Protection	OTP	0	1 = BP starts at bottom (Low address) 0 = BP starts at top (High address)
4	RFU	RFU	OTP	0	Reserved for Future Use
3	BPNV	Configures BP2-0 in Status Register	OTP	0	1 = Volatile 0 = Non-Volatile
2	TBPARM	Configures Parameter Sectors location	OTP	0	1 = 4-kB physical sectors at top, (high address) 0 = 4-kB physical sectors at bottom (Low address) RFU in uniform sector devices
1	QUAD	Puts the device into Quad I/O operation	Non-Volatile	0	1 = Quad 0 = Dual or Serial
0	FREEZE	Lock current state of BP2-0 bits in Status Register, TBPROT and TBPARM in Configuration Register, and OTP regions	Volatile	0	1 = Block Protection and OTP locked 0 = Block Protection and OTP un-locked

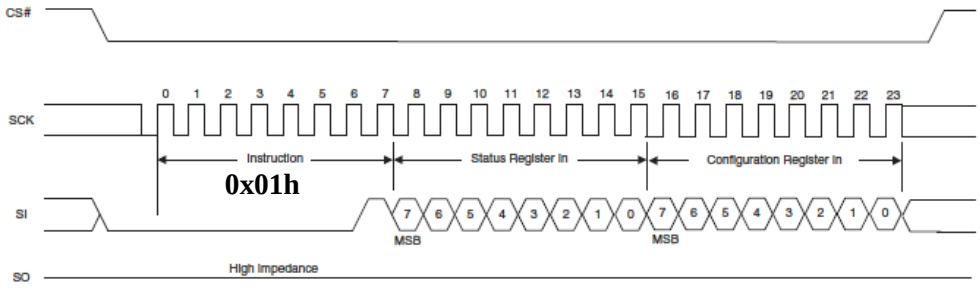


Figure 3.26: WRR command sequence of S25FL128S flash memory Source: Cypress (2017) '128 Mbit (16 Mbyte)/256 Mbit (32 Mbyte) 3.0V SPI flash memory'

Next, the WREN command is used to set the Write Enable Latch (WEL) bit of the Status Register-1. For our case, the WEL bit must be set to 1 prior issuing a WRR command. The communication sequence of the WREN command is shown in Figure 3.27.

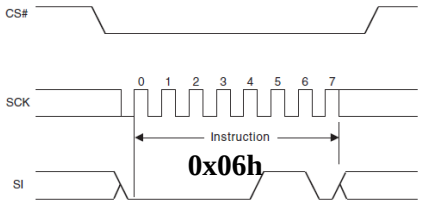


Figure 3.27: WREN command sequence of S25FL128S flash memory Source: Cypress (2017) '128 Mbit (16 Mbyte)/256 Mbit (32 Mbyte) 3.0V SPI flash memory'

Lastly, the QOR command is used to get the data from the S25FL128S flash memory to the cache (I-CACHE or D-CACHE) through IO0-IO3 pins of the S25FL128S flash memory, which is shown in Figure 3.28. The communication sequence of the QOR command is shown in Figure 3.29.

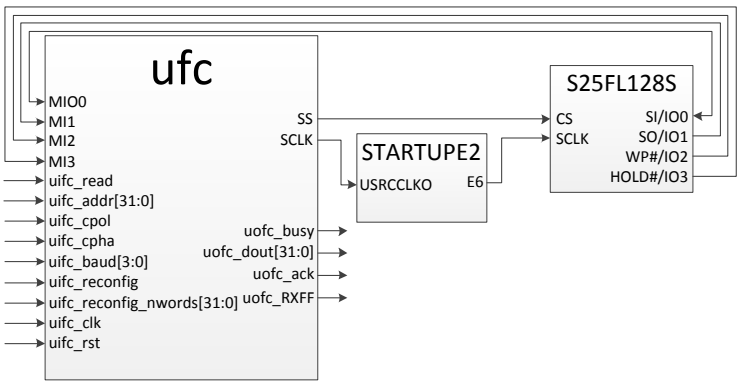


Figure 3.28: Wiring connection of S25FL128S flash memory with Flash Controller Unit

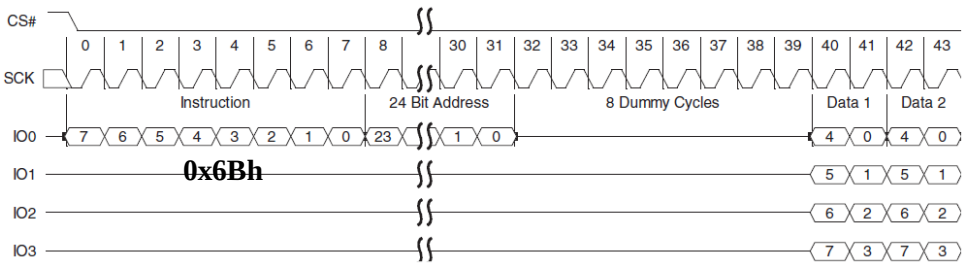


Figure 3.29: QOR command sequence of S25FL128S flash memory Source: Cypress (2017) '128 Mbit (16 Mbyte)/256 Mbit (32 Mbyte) 3.0V SPI flash memory'

3.3.4.2 Design Partitioning

The Flash Controller unit is developed with 5 major blocks: Flash Controller FSM, Flash Controller Receiver, Flash Controller Transmitter, FIFO and Flash Controller Clock Generator. Figure 3.30 shows the microarchitecture of the Flash Controller unit.

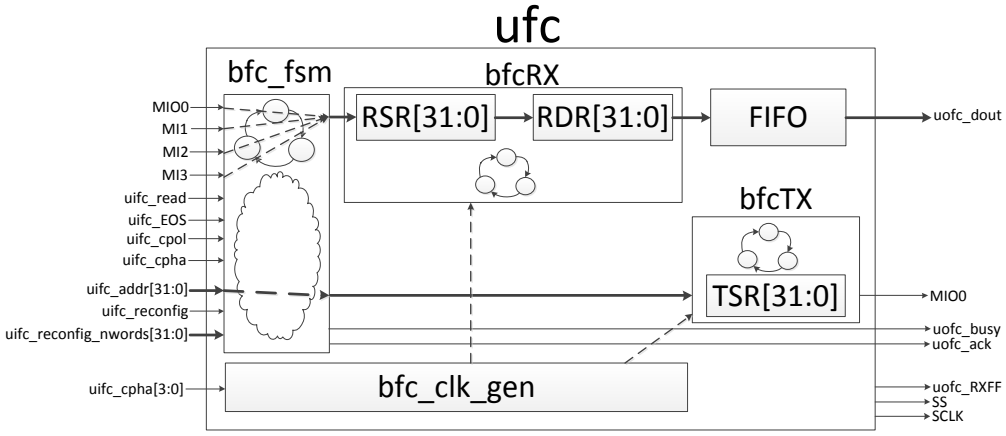


Figure 3.30: Flash Controller unit microarchitecture

3.3.5 Boot ROM Unit

The Boot ROM unit is used to store the bootloader program. The Boot ROM unit is for read-only, i.e. no write data bus, and the bootloader program is pass to the Boot ROM unit using "$readmemh (`ROM_FILE_PATH, rom_data)" in the Verilog HDL. A block wrapper module is designed, as shown in Figure 3.31, for the ease of using the FPGA Block RAM and Table A.5 describes the function of each pin.

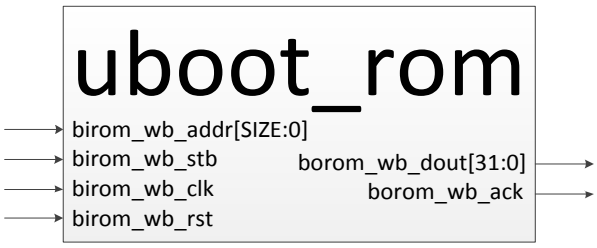


Figure 3.31: Boot ROM Unit chip interface

3.3.6 Data and Stack RAM Unit

The Data and Stack RAM Unit is created to store the runtime data (refer to *.data*, *.bss*, *.stack* and *.heap* segments in Section 3.3.1). The Data and Stack RAM is created using FPGA Block RAM, in which both read and write access of the data must be synchronous to the clock source. A block wrapper module is designed, as shown in Figure 3.32, for the ease of using the FPGA Block RAM and Table A.6 describes the function of each pin.

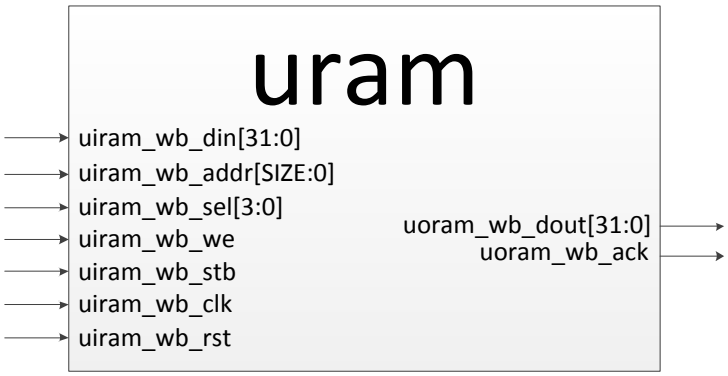


Figure 3.32: Data and Stack RAM Unit chip interface

3.4 I/O System

I/O system components consist of SPI controller, UART controller, GPIO controller, Priority Interrupt Controller and General Purpose Register (GPR) unit, are connected to the CPU through Wishbone B4 standard bus interface (OpenCores, 2010). CPU is treated as the master device while the I/Os connected are treated as the slave devices. Based on the Wishbone standard, Table 3.12 shows the specific signals are required by both master and slave devices while Figure 3.33 shows the I/O system architecture at the MEM stage (remark: Address Decoder Block is in EX stage).

Table 3.12: Wishbone standard signals for master and slave device

	Master Device	Slave Device
Input	clock input (CLK_I), reset input (RST_I), data input array (DAT_I)	clock input (CLK_I), reset input (RST_I), data input array (DAT_I), address input array (ADR_I), select input array (SEL_I), strobe input (STB_I), write enable input (WE_I)
Output	data output array (DAT_O), address output array (ADR_O), select output array (SEL_O), strobe output (STB_O), write enable output (WE_O)	data output array (DAT_O), acknowledge output (ACK_O)

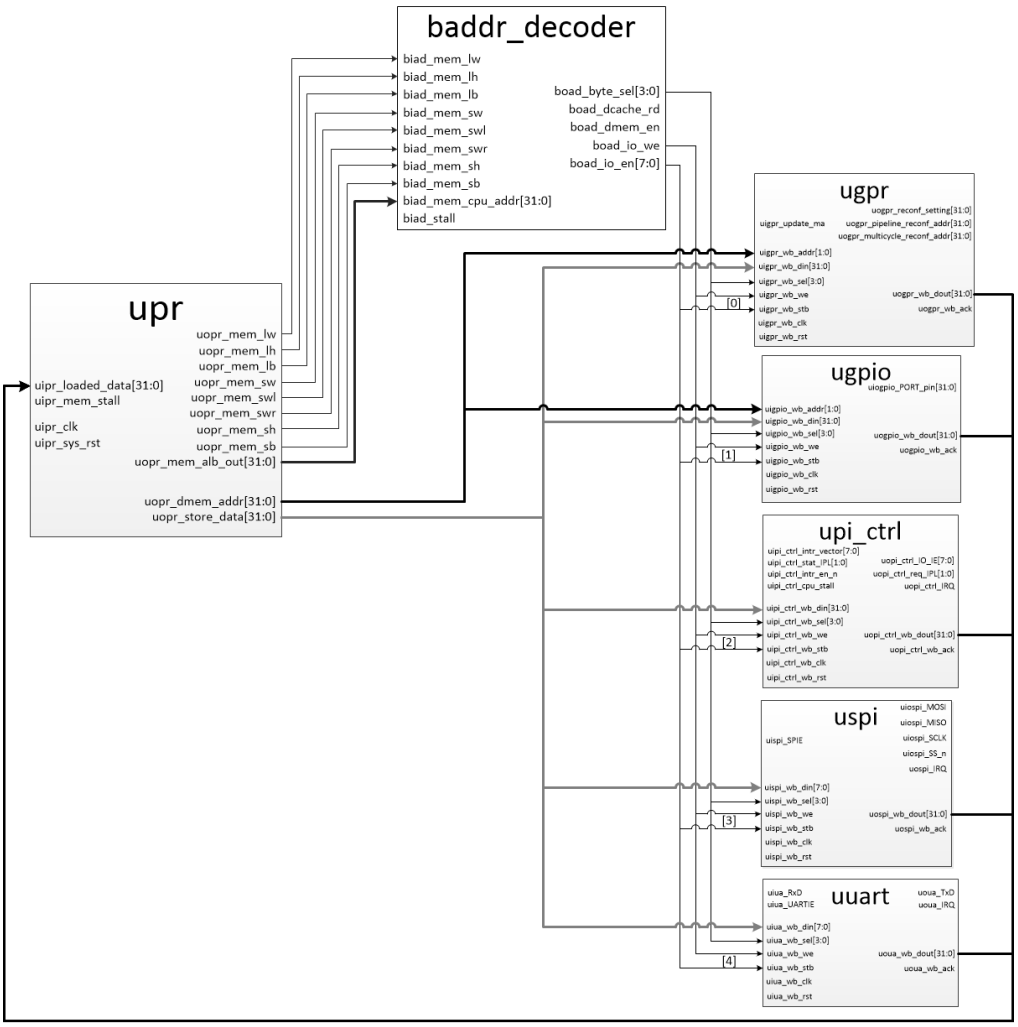


Figure 3.33: I/O system architecture at MEM stage [PR unit (upr) pins is simplified for illustration purpose]

3.4.1 UART controller

The UART controller is used for serial data communication between the UART interface devices. The UART controller is designed to provide the following features:

1) Half-duplex asynchronous transmitting and receiving, where only transmit or receive data can take place at any time
2) Performs serial-to-parallel data conversions on data received from another device via uiua_RxD
3) Performs parallel-to-serial data conversions and transmit the data to another device via uoua_TxD
4) Programmable baud rate with 8 speed selection (300 baud -> 38400 baud)
5) Selectable parity bit
6) Parity error (PE) and framing error (FE) detection
7) Received complete and transmit FIFO empty interrupt support

Figure 3.34 shows the chip interface of the UART controller and Table A.7 describes the function of each pin.

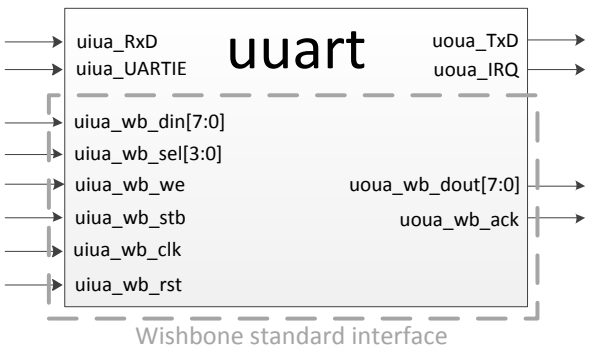


Figure 3.34: UART Controller chip interface

3.4.1.1 UART protocol

UART data communication protocol is divided into 4 parts: start, data, parity and stop bit. Figure 3.35 illustrates the UART data communication protocol.

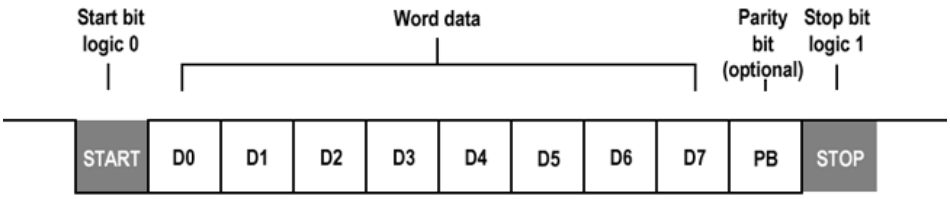


Figure 3.35: UART data communication protocol

1) Start bit: This bit is set to LOW to initiate bit synchronization of the message at the receiver.
2) Data: Represent the data that will be transmitted. The least significant bit (LSB) will be sent out first followed by next bit until the most significant bit (MSB).
3) Parity Bit: This bit represents even or odd parity if parity is enabled. The CPU is in charge of manipulating the even or odd parity.
4) Stop Bit: This bit is set to HIGH to provide the message-framing indication for use in bit synchronization at the receiver.

When receiving the data from the UART controller, the start bit will be sampled for 5 times, where the serial bit at the 5th sample will be indicated as the start bit. When the start bit detected, the UART controller will sample the next serial bit for 10 times and the serial bit at the 10th sample will be

recorded and treated as the LSB of the receiving data. The data receiving process will continue until the stop bit is received. The sequence of the data sampling will be 5 (Start bit), 10 (LSB), 10, 10, 10, 10, 10, 10, 10 (MSB), 10 (Parity bit-optional) and 10 (Stop bit). Figure 3.36 illustrates the process of data sampling when receiving data from the UART controller.

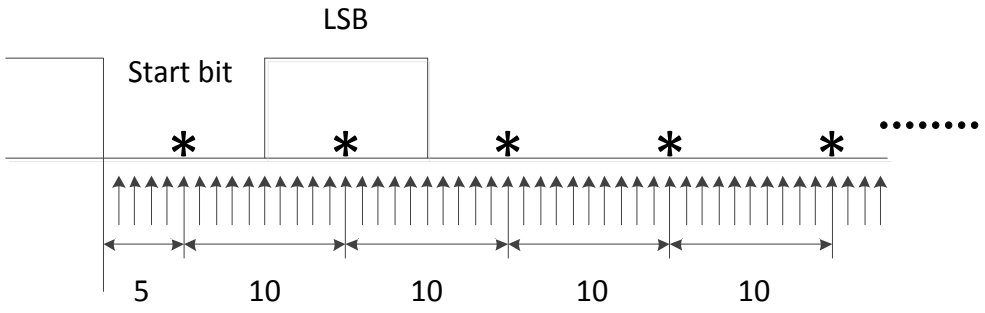


Figure 3.36: Process of data sampling when receiving data through UART controller

A parity bit is used for error detection when in UART serial data transmission. Odd parity create an odd count of 1's in a stream of data (8-bits data and 1 parity bit) while even parity creates an even count of 1's in a stream of data. A parity error occurs when the count of 1's is not as agreed by both UART devices, where even parity will have 0, 2, 4, 6 or 8 of 1's in a stream of data while odd parity will have 1, 3, 5, 7 or 9 of 1's in a stream of data. Parity error may occur due to the transmission line corrupted. However, no correction can be done by the UART controller except notify the user by sending an interrupt request. Such condition can only be resolved by re-sending the data.

Each stream of data consists of a start and a stop bit to indicate the start and the stop of one byte data communication. Start bit initiate the communication while stop bit terminates the communication. The framing error occurs when no stop bit is detected. Such condition may occur due to the transmission line corrupted, different baud rate for both devices and parity setting of both devices. UART controller will send an interrupt request to notify this condition had arisen and the user is required to resolve it by checking the wire connection or the configuration setting of both UART devices.

3.4.1.2 Design Partitioning

UART controller consists of Baud Clock Generator (bclkctr), Receiver (brx) and Transmitter (btx) blocks. 4 registers, UARTCR (UART Configuration Register), UARTSR (UART Status Register), UARTRDR (UART Receive Data Register) and UARTTDR (UART Transmit Data Register), are available for user access while 2 shift registers, Transmitter Shift Register (TSR) and Receiver Shift Register (RSR), are used for parallel-to-serial and serial-to-parallel data conversion respectively. UARTCR and UARTSR are used for configuration setting and status monitoring purposes. UARTTDR holds the data that will be transmitted to another UART device. UARTTDR is a 4 x 1-byte FIFO memory, where up to 4 bytes of data can be inserted by user and queue for data transmitting. UARTRDR is also a 4 x 1-byte FIFO memory, where up to 4 bytes of data can be received and buffered. UART Receiver block used a 9-bit shift register (RSR) to receive each bit serially from another UART device. Once the 9-bit data received (8 data bit and 1 stop bit), the data in the RSR data will be passed to the UARTRDR. UARTRDR holds the data while RSR can continue to receive another byte of data. For data transmitting, UART Transmitter block used a 10-bit shift register (TSR) to transmit the data (8 data bit, 1 parity bit and 1 stop bit) serially to another UART device. Figure 3.37 shows the internal connection of the UART Controller.

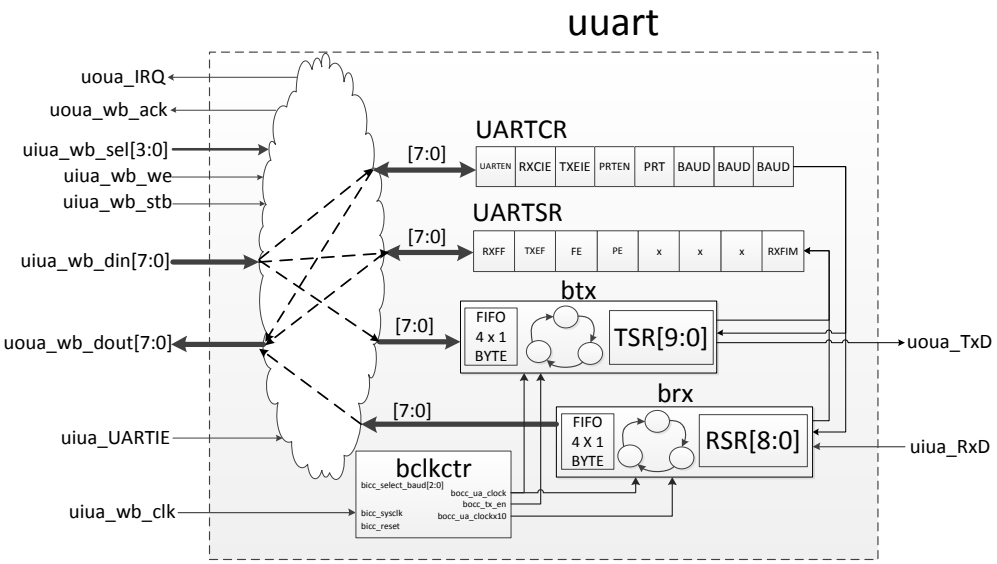


Figure 3.37: Internal connection of the UART Controller

3.4.1.3 Register sets

4 registers are used to allow the data communication between the CPU and the UART controller.

1) UART Configuration Register (UARTCR) – 8-bit (0xBFFFFE28)

UARTEN	RXCIE	TXEIE	PRTEN	PRT	BAUD [2:0]

   a. **UARTEN –** UART enable control. It is used to deactivate the UART controller when it is not in use. When activated, even not transmitting a byte of data, the UART controller is capable to receive a byte of data from another device. To have better control on power consumption, the UART controller is recommended to be deactivated when not in use.

      i. Set to activate the UART controller

      ii. Clear to deactivate the UART controller

   b. **RXCIE –** Receive Complete Interrupt enable control. It can only be used if and only if UARTIE (UART global interrupt enable) bit in PICMASK (0xBFFFFE22) is set. This bit is used for interrupt enable control (to select interrupt method instead of polling) after data has been completely received (as indicated by the RXDF bit in UARTSR (0xBFFFFE29)).

      i. Set to enable Receive Complete Interrupt

      ii. Clear to disable Receive Complete Interrupt

   c. **TXEIE –** Transmit FIFO Empty Interrupt enable control. It can only be used if and only if UARTIE (UART global interrupt enable) bit in PICMASK (0xBFFFFE22) is set. This bit is used for interrupt enable control (to select interrupt method instead

of polling) when the transmitter FIFO (UARTTDR) is empty (as indicated by the TXEF bit in UARTSR (0xBFFFFE29)).

i. Set to enable Transmit Empty Interrupt

ii. Clear to disable Transmit Empty Interrupt

d. **PRTEN** – Parity enable control

e. **PRT** – Parity bit, to select either odd or even parity

f. **BAUD[2:0]** – UART Baud Rate

i. 000: 38400 baud

ii. 001: 19200 baud

iii. 010: 9600 baud

iv. 011: 4800 baud

v. 100: 2400 baud

vi. 101: 1200 baud

vii. 110: 600 baud

viii. 111: 300 baud

2) UART Status Register (UARTSR) – 8-bit (0xBFFFFE29)

RXDF	TXEF	FE	PE	RESERVED [3:1]	RXFM

a. **RXDF** – Receive Done flag. This bit when set by UART, indicates 1-byte or 4-byte of data have been completely received. It is used in conjunction with RXFM bit in UARTSR (0xBFFFFE29) to determine if the receive data is 1-byte (RXFM = 0) or 4-byte (RXFM = 1 i.e. FIFO full).

b. **TXEF** – Transmit FIFO Empty flag. This bit is set to 1 by the UART if the transmit FIFO is empty.

c. **FE** – Framing error. It is set when not detecting a stop bit.

d. **PE** – Parity error. It is set when parity bit mismatch.

e. **RXFM** – Receive FIFO Full Mode. It is part of UARTCR. It is placed in UARTSR (0xBFFFFE29) to avoid creating longer bytes of UARTCR.

   i. Set to 1 by user to indicate a 4-byte (FIFO full) data is expected to be read by CPU.

   ii. Clear to 0 by user to indicate a 1-byte data is expected to be read by CPU.

3) UART Transmit Data Register (UARTTDR) – 8-bit (0xBFFFFE2A)

UARTTDR [7:0]

a. This register holds the data that will transmit to another UART device

4) UART Receive Data Register (UARTRDR) – 8-bit (0xBFFFFE2B)

UARTRDR [7:0]

a. This register holds the data received from another UART device

3.4.2 SPI controller

The SPI controller is used for high speed serial data communication between the SPI interface devices. It is developed with 4 wires, which are Master out Serial in (MOSI), Master in Serial out (MISO), Slave Select (SS) and SPI clock (SCLK), and 4 modes of serial data communication (some of the SPI interface module support only certain mode of serial data communication, e.g. The CC2420 from Texas Instruments only support mode 0 and 3). The SPI controller is able to perform the following functionalities.

1) Full duplex 8-bit data (8 SCLK cycles) communication, which serial data transmission and receiving can take place at the same time
2) Selectable 16 transmission speed (305 Hz -> 10 MHz)
3) Selectable 4 mode of transmission (mode 0, 1, 2 and 3)
4) Mode Fault error detection
5) Received buffer full and transmit buffer empty interrupt support

Figure 3.38 shows the chip interface of the SPI controller and Table A.8 describes the function of each pin.

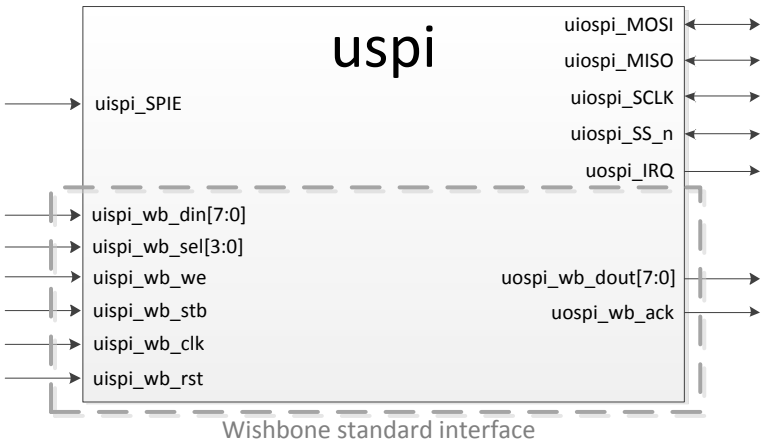


Figure 3.38: SPI Controller chip interface

3.4.2.1 SPI protocol

To allow serial data communication among SPI devices, in which only one master device is allowed to exist in the same connection, whereas other SPI devices must be configured by the user as slave devices. To avoid 2 or more master devices exist in the same connection, the user can check the MSTR bits from all the connected SPI devices to ensure only one MSTR bit is set. The SPI controller also have the capability to check the uiospi_SS_n pin for 2 or more master devices existing in the same connection and will generate an interrupt signal (Mode Fault error) to the CPU to notify the user.

To initiate the SPI serial data communication, master device pull low the uiospi_SS_n pin for 8 uiospi_SCLK cycles. A clock pulse is sent from the master to the slave devices, through the uiospi_SCLK pin, for serial data synchronization purposes. After the uiospi_SS_n pin is pulled low, 8-bit serial data is transmitted from the uiospi_MOSI pin of the master device to the uiospi_MOSI pin of the slave device. At the same time, uiospi_MISO pin of the master device can receive 8-bit data from the uiospi_MISO pin of the slave device. This makes SPI device capable of performing full duplex data communication. The serial 8-bit data communication is completed after 8 uiospi_SCLK cycles followed by pull high the uiospi_SS_n pin. Four communication modes (mode 0, 1, 2 and 3), defined by Clock polarity (CPOL) and Clock phase (CPHA), are available to identify the uiospi_SCLK edges to be used for data transmit and sampling. Table 3.13 shows the SPI mode 0, 1, 2 and 3 with respect to CPOL and CPHA and Figure 3.39 to Figure 3.42 illustrates the SPI serial data communication for each communication mode.

Table 3.13: SPI communication mode information

mode	CPOL	CPHA	Data transmit at	Data sample at (Receive)
0	0	0	half clock before the rising edge	rising edge
1	0	1	rising edge	falling edge
2	1	0	half clock before the falling edge	falling edge
3	1	1	falling edge	rising edge

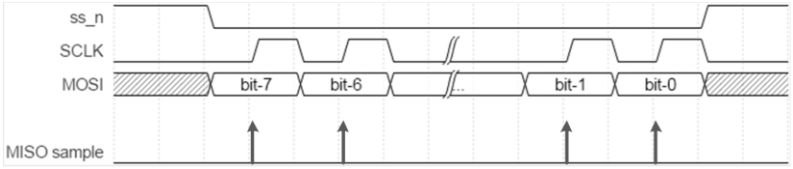


Figure 3.39: Mode 0 serial data communication

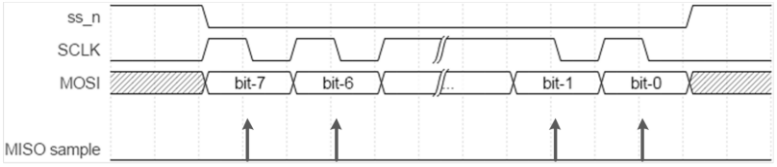


Figure 3.40: Mode 1 serial data communication

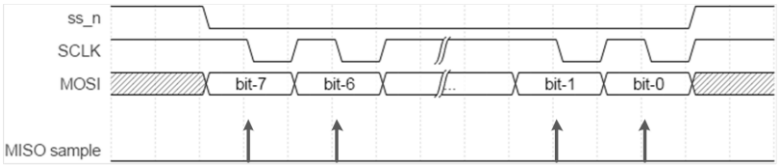


Figure 3.41: Mode 2 serial data communication

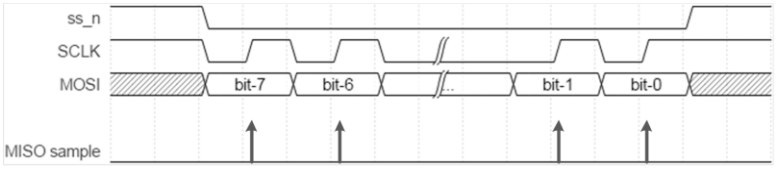


Figure 3.42: Mode 3 serial data communication

3.4.2.2 Design Partitioning

SPI controller consists of Clock Generator (bspiclk_gen), Input Output Control (bspiIO_ctrl), Receiver (bspiRX) and Transmitter (bspiTX) blocks. 4 registers, SPICR (SPI Configuration Register), SPISR (SPI Status Register), SPIRDR (SPI Receive Data Register) and SPITDR (SPI Transmit Data Register), are available for user access while 2 shift registers, Transmitter Shift Register (TSR) and Receiver Shift Register, are used for parallel-to-serial and serial-to-parallel data conversion respectively. SPICR and SPISR are used for configuration setting and status monitoring purposes. SPITDR holds the data that will be transmitted to another SPI device. SPITDR is a 16 x 1-byte FIFO memory, where up to 16 bytes of data can be inserted by user and queue for data transmitting. SPIRDR is also a 16 x 1-byte FIFO memory, where up to 16 bytes of data can be received and buffered. SPI Receiver block used an 8-bit shift register (RSR) to receive each bit serially from another SPI device. Once the 8-bit data received, the data in the RSR data will be passed to the SPIRDR. SPIRDR holds the data while RSR can continue to receive another byte of data. For data transmitting, SPI Transmitter block used an 8-bit shift register (TSR) to transmit the data serially to another SPI device.

3.4.2.3 Register sets

4 registers are used to allow the data communication between the CPU and the SPI controller.

1) SPI Configuration Register (SPICR) – 8-bit (0xBFFFFE24)

SPE	MSTR	CPOL	CPHA	SCR[3:0]

   a. **SPE** – SPI enable control. It is used to deactivate the SPI controller when it is not in use. To have better control on power consumption, the SPI controller is recommended to be deactivated when not in use. Set to activate SPI controller, else otherwise.

   b. **MSTR** – Master/Slave device. Set to indicate as master device, else otherwise.

   c. **CPOL** – Clock Polarity

   a. **CPHA** – Clock Phase

   b. **SCR[3:0]** – SPI Clock Rate (CPU clock speed is 20 MHz)

      i. 0000: 10 MHz

      ii. 0001: 5 MHz

      iii. …

      xv. 1111: 305 Hz

2) SPI Status Register (SPISR) – 8-bit (0xBFFFFE25)

RXDF	TXEF	MODF	RXFM	RXFIE	TXEIE	RXFHE	TXEHE

   a. **RXDF** – Receive Done flag. This bit when set by SPI, indicates 1-byte or 16-byte of data have been completely received. It is used in conjunction with RXFM bit in SPISR (0xBFFFFE25) to

determine if the receive data is 1-byte (RXFM = 0) or 16-byte (RXFM = 1 i.e. FIFO full).

b. **TXEF** – Transmit FIFO Empty flag. This bit is set to 1 by the SPI if the transmit FIFO is empty.

c. **MODF** – Mode Fault error. When SPI unit is set as the master device, the uiospi_SS_n pin must pull high by the master device. If there existed two master devices, any attempt to pull low the uiospi_SS_n pin will trigger the mode fault error. This is to avoid two master devices exist in the same connection and avoid damage to the hardware.

d. **RXFM** – Receive FIFO Full Mode. It is part of SPICR. It is placed in SPISR (0xBFFFFE25) to avoid creating longer bytes of SPICR.

    i. Set to indicate a 16-byte (FIFO full) data is expected to be read by CPU.

    ii. Clear to indicate a 1-byte data is expected to be read by CPU.

e. **RXFIE** – Receive Complete Interrupt enable. It is part of SPICR. It is placed in SPISR (0xBFFFFE25) to avoid creating longer bytes of SPICR. It can only be used if and only if SPIE (SPI global interrupt enable) bit in PICMASK (0xBFFFFE22) is set. This bit is used for interrupt enable control (to select interrupt method instead of polling) after data has been completely received (as indicated by the RXDF bit in SPISR (0xBFFFFE25)). Set to enable Receive Complete Interrupt, else otherwise.

f. **TXEIE** – Transmit FIFO Empty interrupt enable. It is part of SPICR. It is placed in SPISR (0xBFFFFE25) to avoid creating longer bytes of SPICR. It can only be used when SPIE (SPI global interrupt enable) bit in PICMASK (0xBFFFFE22) is set. This bit is used for interrupt enable control (to select interrupt method instead of polling) when the transmitter FIFO (SPITDR) is empty (as indicated by the TXEF bit in SPISR (0xBFFFFE25)). Set to enable Transmit Empty Interrupt, else otherwise.

g. **RXFHE** – Receive-Byte Halt enable. It is part of SPICR. It is placed in SPISR (0xBFFFFE25) to avoid creating longer bytes of SPICR.

   i. Set to enable FSM stall when received one byte of data

   ii. Clear to disable FSM stall

h. **TXEHE** – Transmit FIFO Empty Halt enable. It is part of SPICR. It is placed in SPISR (0xBFFFFE25) to avoid creating longer bytes of SPICR.

   i. Set to enable FSM stall when a full stream of data stored in the FIFO memory is transmitted

   ii. Clear to disable FSM stall

3) SPI Transmit Data Register (SPITDR) – 8-bit (0xBFFFFE26)

SPITDR [7:0]

a. Holds the data that will be transmitted to another SPI device

4) SPI Receive Data Register (SPIRDR) – 8-bit (0xBFFFFE27)

SPIRDR [7:0]

a. Holds the data received from another SPI device

3.4.3 GPIO controller

The General Purpose Input/Output (GPIO) Controller is developed with 32-bits I/O port, where each pin can be set as either input or output. The GPIO Controller can be used for controlling the external devices, blinking LEDs, debugging, digital input reading etc. Figure 3.43 shows the chip interface of the GPIO Controller unit and Table A.9 describes the function of each pin. Figure 3.44 illustrates the internal operation of the GPIO Controller unit.

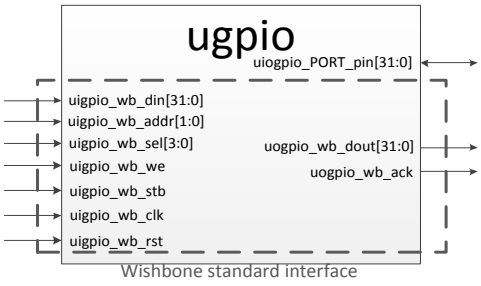


Figure 3.43: GPIO Controller unit chip interface

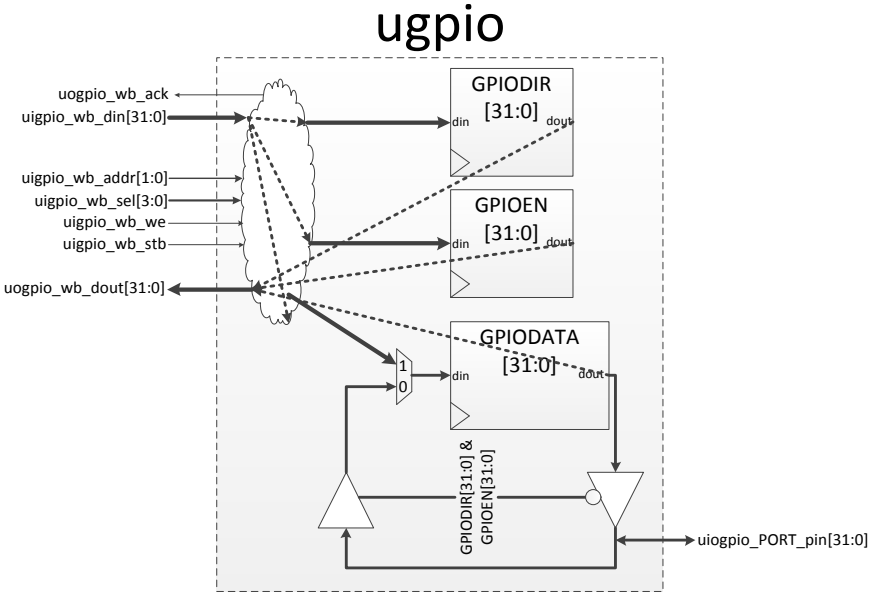


Figure 3.44: Internal operation of GPIO Controller unit

3.4.3.1 Register sets

3 registers are used to allow the data communication between the CPU and the GPIO controller.

1) GPIO Direction Control Register (GPIODIR) – 32-bits (0xBFFFFE10)

GPIODIR[31:0]

   a. **GPIODIR[31:0]** – GPIO Direction. It is used to configure each pin either as input or output. When system restarts, all the pins are preset as output pins.

      i. 1 = input pin; 0 = output pin

2) GPIO Enable Control Register (GPIOEN) – 32-bit (0xBFFFFE14)

GPIOEN [31:0]

   a. **GPIOEN[31:0]** – GPIO pins enable control. It is used to enable or disable each pin.

      i. 1 = pin enable; 0 = pin disable

3) GPIO Data Register (GPIODATA) – 32-bit (0xBFFFFE18)

GPIODATA [31:0]

   a. **GPIODATA[31:0]** – GPIO data. It is used store the data of each pin. Each bit is set by the user if respective bit is defined as an output pin, which user may write 1 into the respective bit in GPIODATA register and respective pins will output logic high. If a GPIO pin is defined as the input pin, the respective bit in GPIODATA register should store the digital data that received from an external device.

3.4.4 Priority Interrupt Controller

The developed Priority Interrupt Controller unit is an external interrupt controller (CP0 is the internal interrupt controller). Priority Interrupt Controller unit assist CP0 block to identify the highest priority interrupt source from 8 interrupt sources. The currently connected interrupt sources are SPI controller, UART controller and CP0 timer. Four interrupt priority levels (IPL) can be set for each interrupt source, where the highest level gains the highest priority. Figure 3.45 shows the chip interface of the Priority Interrupt Controller unit and Table A.10 describes the function of each pin. Figure 3.46 illustrates the internal operation of the Priority Interrupt Controller unit.

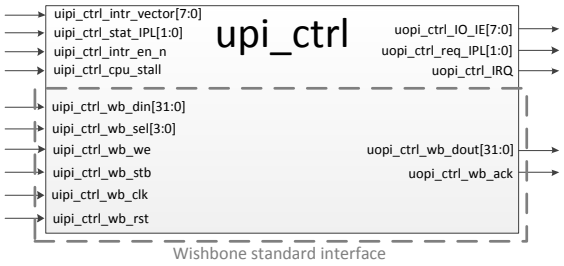


Figure 3.45: Priority Interrupt Controller unit chip interface

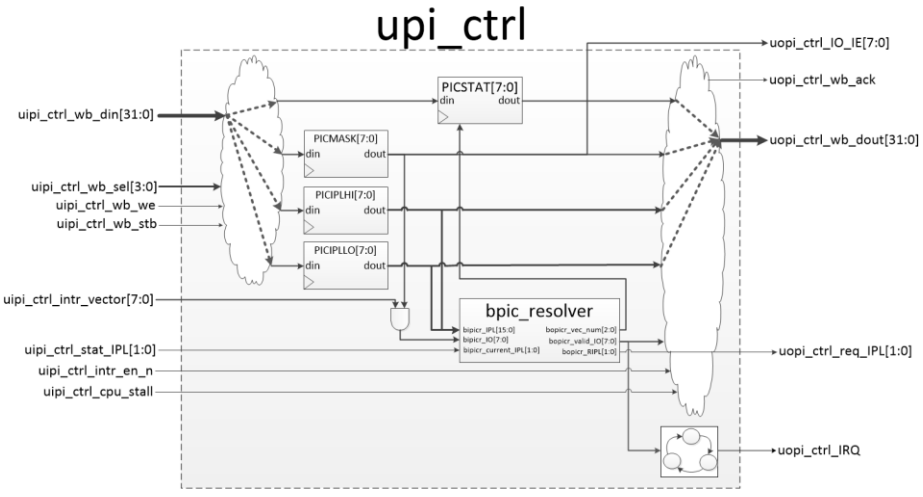


Figure 3.46: Internal operation of Priority Interrupt Controller unit

3.4.4.1 Interrupt protocol

The priority scheme is implemented with respect to two rules:

1) Always serve the highest IPL interrupt source
2) When same IPL, serve the interrupt source with lowest vector number, for example: served uipi_ctrl_intr_vector[2] first followed by uipi_ctrl_intr_vector[3] when both interrupt the CPU at the same time

The uopi_ctrl_IRQ should only assert for 1 clock cycle, in which it will instruct the CPU to jump to the exception handler. Figure 3.47 shows the timing requirement of the Priority Interrupt Controller, in which the condition shown demonstrate the nested interrupt request from 8 interrupt sources.

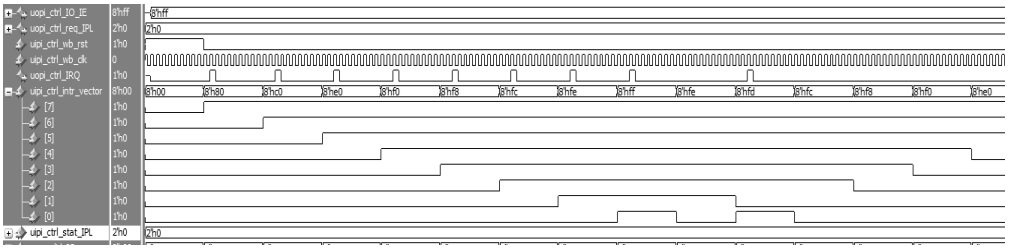


Figure 3.47: Timing requirement of Priority Interrupt Controller unit

3.4.4.2 Register sets

4 registers are used to allow the data communication between the CPU and the Priority Interrupt Controller.

1) Interrupt Priority Level Low-byte Register (PICIPLLO) – 8-bit (0xBFFFFE20)

IPL3	IPL2	IPL1	IPL0
PICIPLLO			

User sets the interrupt priority level of the I/Os. For example, PICIPLLO[7:0] = 8'b00_00_11_11, this means I/O0 has the higher priority than I/O1 because I/O0 has the lowest vector number.

   a. **IPL0[1:0]** – Interrupt Priority Level of interrupt source 0
   b. **IPL1[1:0]** – Interrupt Priority Level of interrupt source 1
   c. **IPL2[1:0]** – Interrupt Priority Level of UART
   d. **IPL3[1:0]** – Interrupt Priority Level of SPI

2) Interrupt Priority Level High-byte Register (PICIPLHI) – 8-bit (0xBFFFFE21)

IPL7	IPL6	IPL5	IPL4
PICIPLHI			

   a. **IPL4[1:0]** – Interrupt Priority Level of interrupt source 4
   b. **IPL5[1:0]** – Interrupt Priority Level of interrupt source 5
   c. **IPL6[1:0]** – Interrupt Priority Level of interrupt source 6
   d. **IPL7[1:0]** – Interrupt Priority Level CP0 timer

3) Interrupt Sources Masking Register (PICMASK) – 8-bits (0xBFFFFE22)

IO7	IO6	IO5	IO4	SPIE	UARTIE	IO1	IO0
PICMASK							

It is used as the global interrupt enable control of each I/O.

a. **IO0** – Interrupt source 0 Interrupt Enable

b. **IO1** – Interrupt source 1 Interrupt Enable

c. **UARTIE** – UART Interrupt Enable

d. **SPIE** – SPI Interrupt Enable

e. **IO4** – Interrupt source 4 Interrupt Enable

f. **IO5** – Interrupt source 5 Interrupt Enable

g. **IO6** – Interrupt source 6 Interrupt Enable

h. **IO7** – Interrupt source 7 Interrupt Enable

4) Status Register (PICSTAT) – 8-bits (0xBFFFFE23)

x	x	x	x	x	vec_num
PICSTAT					

Stored the currently I/O information being served by the CPU. The exception handler reads the PICSTAT to identify which I/O and hence, which Interrupt Service Routine (ISR) to jump to.

a. **vec_num[2:0]** – store the vector number of Interrupt source that currently handles

000 – Interrupt source 0

001 – Interrupt source 1

010 – UART

011 – SPI

100 – Interrupt source 4

101 – Interrupt source 5

110 – Interrupt source 6

111 – CP0 timer

3.4.5 General Purpose Register

The General Purpose Register unit is developed to store the current microarchitecture identification bit, multi-cycle microarchitecture partial bitstream start address, pipeline microarchitecture partial bitstream start address and the partial bitstream size. Figure 3.48 shows the chip interface of the General Purpose Register unit and Table A.11 describes the function of each pin.

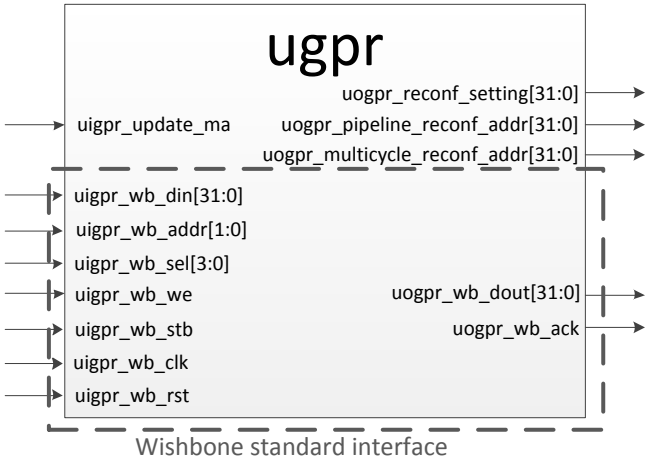


Figure 3.48: General Purpose Register unit chip interface

3.4.5.1 Register sets

3 registers are included in the General Purpose Register unit.

1) Setting Register (SETTING) – 32-bit (0xBFFFFE00)

MABS[31:9]	CMA	SYSCLK[7:0]

   a. **MABS[31:9]** – Microarchitecture PR Bitstream Size (Word). Store the PR bitstream word size. Multi-cycle and pipeline microarchitectures have the same PR bitstream word size. This information is used to prevent fetching the wrong PR bitstream from flash memory

   b. **CMA** – Current Microarchitecture. Set when current microarchitecture is pipeline microarchitecture while reset when it is multi-cycle microarchitecture

   c. **SYSCLK[7:0]** – System clock frequency

2) Pipeline Microarchitecture PR Bitstream Start Address Register (P5CADDR)– 32-bit (0xBFFFFE04)

RESERVED[31:24]	P5CADDR[23:0]

   a. **P5CADDR[23:0]** – Pipeline microarchitecture PR bitstream start address. The 24-bits value stored indicates the address location of pipeline microarchitecture in the flash memory

3) Multi-cycle Microarchitecture PR Bitstream Start Address Register (M5CADDR) – 32-bit (0xBFFFFE08)

RESERVED[31:24]	M5CADDR[23:0]

   a. **M5CADDR[23:0]** – Multi-cycle microarchitecture PR bitstream start address. The 24-bits value stored indicates the address location of multi-cycle microarchitecture in the flash memory

3.5 Polling and Single Vector Nested Interrupt Serving

Interrupt sources include UART, SPI and CP0 timer can be served through polling or interrupt method. Polling access repeatedly checks the interrupt source to determine whether it is ready for data transfer. The following example shows the flow of the UART's transmits FIFO is empty using the polling method:

1) Disable the UART global interrupt through resetting the UARTIE bit in the EXPIC register, of the priority interrupt controller
2) UART configuration through UARTCR
    a. Set baud rate (BAUD[2:0] = 010, 9600 baud)
    b. Start UART by setting the UARTEN bit.
3) Load UARTSR register value to Register File's register to check the TXEF bit of the UARTSR register
    a. When TXEF=1, break the loop and continue with further process
    b. When TXEF=0, repeat step 3 to continue polling

The polling method occupies the processing capability of the CPU. Instead, the interrupt method allows the CPU to proceed with other tasks while waiting for interrupt sources to interrupt the CPU for special attention. The following example shows the flow of the UART's transmit FIFO is empty using interrupt method:

1) Enable the UART global interrupt by setting the UARTIE bit to 1 in the EXPIC register of the Priority Interrupt Controller
2) UART configuration through UARTCR

a. Set baud rate (BAUD[2:0] = 010, 9600 baud) and set the TXEIE bit to enable the transmit FIFO is empty interrupt enable.
b. Start UART by setting the UARTEN bit.

3) Move on to process other tasks
4) When interrupt occurs, jump to exception handler, 0x8001_B400 virtual address (we will discuss this in detailed later)

Even if the interrupt method can achieve higher computation than the polling method, however, it needs higher effort in program flow design. The problem would come when the multiple interrupts occur at the same time or interrupt occurs when the CPU is serving an interrupt. Thus, we had developed a scheme (single vector nested interrupt) that includes the hardware and firmware to overcome the problem.

Single vector interrupt allows every interrupts jump to a single general routine (exception handler) rather than jump to the specific interrupt source's interrupt service routine (ISR). The hardware parts (Priority Interrupt Controller and CP0 block) responsible to send an interrupt signal to the CPU for interruption based on priority (higher priority gets served first), set EXL bit of the $stat register in the CP0 block to disable further exception and reset EXL bit of the $stat when the exception return (*eret* instruction). The related register information of the CP0 block is shown in Figure 3.49 and described in Table 3.14.

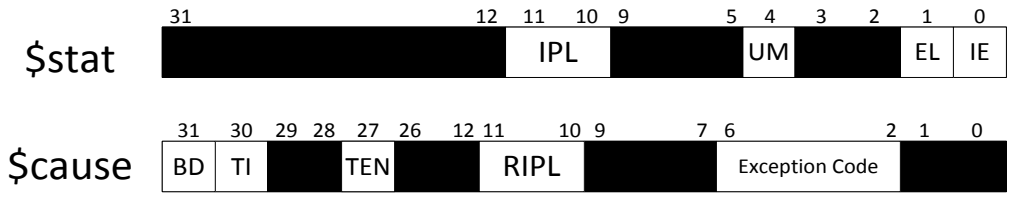


Figure 3.49: Graphical view of CP0 $stat and $cause registers

Table 3.14: $stat and $cause register description

Register	bit	usage
$stat	[31:12]	RESERVED
	IPL[11:10]	store current interrupt priority level
	[9:5]	RESERVED
	UM[4]	1=user mode, 0=kernel mode
	[3:2]	RESERVED
	EL[1]	Exception level 1=exception occurs, disable further exception to occur 0=no exception occurs
	IE[0]	1=Interrupt enable 0=Interrupt disable
$cause	BD[31]	Indicate branch delay
	TI[30]	1=enable timer interrupt 0=disable timer interrupt
	[29:28]	RESERVED
	TEN[27]	CP0 Timer, $count disable control
	[26:12]	RESERVED
	RIPL[11:10]	Request interrupt priority level
	[9:7]	RESERVED
	Exception code [6:2]	encodes reasons for the exception 0=Interrupt 4=AdEL, address error trap (load or instruction fetch) 5= AdES, address error trap (store) 6=lBE, bus error on instruction fetch trap 7=DBE, bus error on data load or store trap 8=Sys, syscall trap 9=Bp, breakpoint trap 10=Rl, undefined instruction trap 12=Ov, arithmetic overflow trap
	[1:0]	RESERVED

The firmware part performs the program flow shown in Figure 3.50 to allow the nested interrupt to occur. The firmware (exception handler program located at 0x8001_B400 virtual address) decodes the exception and jump to the sub-routine accordingly. The nested interrupt program flow makes used of

the stack memory to store the register information so that to allow another interrupt to occurs.

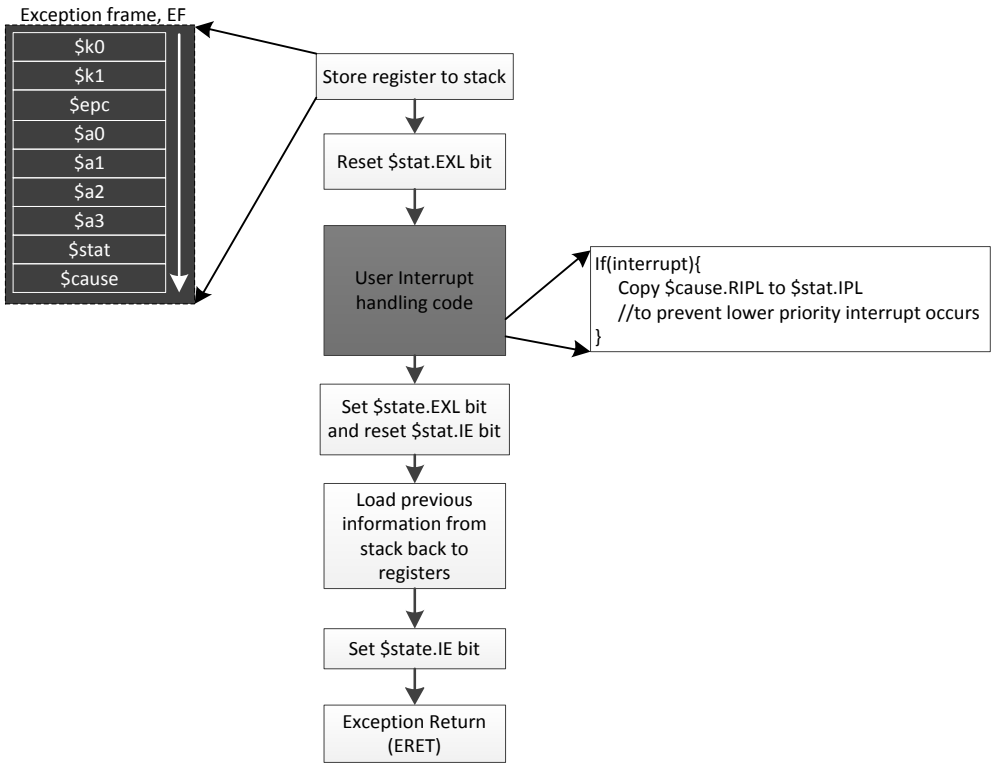


Figure 3.50: Nested interrupt service routine flow

3.6 Summary

This chapter is summarized as follows:

1) MIPS ISA compatible multi-cycle and pipeline processors are developed for experimental purpose.
2) A 2-level memory hierarchy memory system consists of caches, Boot ROM, Data and Stack RAM and flash memory is integrated with the CPU to provide CPU bootloading, fast instruction and data access and runtime variable storage.
3) I/O system consists of SPI controller, UART controller, GPIO controller, Priority Interrupt Controller and GPR unit is connected to the CPU through Wishbone standard bus interface.
4) The PR controller is developed and integrated with CPU to perform the required action for PR. De-coupler unit is used to de-couple the corrupted signals from the static region's logic when PR is in progress.
5) Memory system consists of flash memory, Data and Stack RAM, Boot ROM and I/O registers are map to *kseg0* and *kseg1*.
6) A single vector nested interrupt protocol is developed to allow for nested interrupt support.

CHAPTER 4

SYSTEM VERIFICATION

The developed reconfigurable soft-core IoT processor is synthesized based on Xilinx Artix-7 XC7A100T FPGA chip on Digilent Nexys 4 DDR board. The FPGA resources used in both multi-cycle and pipeline microarchitectures are shown in Table 4.1. The critical path delay of each hardware component in the reconfigurable soft-core IoT processor of both multi-cycle and pipeline microarchitecture is shown in Table 4.2 and Table 4.3.

Table 4.1: FPGA resources used in pipeline and multi-cycle microarchitectures.

FPGA Resources	Microarchitecture	
	Multi-cycle	Pipeline
LUT	7643	8561
LUTRAM	127	311
FF	5464	5812
BRAM	3.50	3.50
IO	45	45
BUFG	1	1

Table 4.2: Critical path delay of each hardware component in multi-cycle microarchitecture (generated from Xilinx Vivado)

Hardware component	Delay (ns)		
	logic	net	total
Partial Reconfiguration			
PR unit (pr_inst)	4.070	8.203	12.273
PR controller unit (upr_ctrl)	3.830	6.478	10.308
CPU			
Data-path unit (udata_path)	4.070	8.203	12.273
Control-path unit FSM (uctrl_path)	4.070	8.203	12.273
Main Control block (bmain_ctrl)	3.616	8.130	11.746
Arithmetic Logic Control block (balb_ctrl)	1.123	2.438	3.561
Register File block (brf)	1.846	2.515	4.361

Continued from Table 4.2

Hardware component	Delay (ns)		
	logic	net	total
CP0 block (bcp0)	4.070	8.203	12.273
Interlock block (bitl_ctrl)	-	-	0.000
Forwarding block (bfw_ctrl)	-	-	0.000
ALU block (balb)	4.070	8.203	12.273
Multiplier Block (bmult32)	6.258	5.957	12.215
Address Decoder block (baddr_decoder)	4.070	7.455	11.525
I/O System			
UART Controller unit (uuart)	3.954	6.993	10.947
- UART Baud Clock Generator block (bclkctr)	1.563	1.736	3.299
- UART Receiver block (brx)	3.946	6.960	10.906
- UART Transmitter block (btx)	3.946	6.993	10.939
SPI Controller unit (uspi)	4.070	7.455	11.525
- SPI Clock Generator block (bclk_gen)	1.801	2.688	4.489
- SPI Receiver block (bRX)	1.261	2.745	4.006
- SPI Transmitter block (bTX)	1.801	2.688	4.489
- SPI Input Output Control block (bio_ctrl)	1.385	2.652	4.037
GPIO Controller unit (ugpio)	3.830	6.524	10.354
Priority Interrupt Controller unit (upi_ctrl)	3.830	6.547	10.377
General Purpose Register unit (ugpr)	3.830	6.483	10.313
Memory System			
Boot ROM unit (uboot_rom)	4.078	8.185	12.263
Data and Stack RAM unit (bsram)	3.706	6.837	10.543
Cache unit (icache)	4.070	8.203	12.273
- Cache Controller block (bcache_ctrl)	3.706	6.802	10.508
- Cache RAM block (bsram)	4.070	8.203	12.273
Cache unit (dcache)	3.830	7.097	10.927
- Cache Controller block (bcache_ctrl)	3.238	6.287	9.525
- Cache RAM block (bsram)	3.830	7.097	10.927
Memory Arbiter unit (umem_arbiter)	2.411	7.031	9.442
Flash Controller Unit (ufc)	2.411	7.031	9.442
- Flash Controller Clock Generator block (bfc_clk_gen)	1.021	2.131	3.152
- Flash Controller FSM block (bfc_fsm)	2.411	7.031	9.442
- Flash Controller Transmitter block (bfc_TX)	1.393	2.811	4.204
- Flash Controller Receiver block (bfc_RX)	1.319	3.385	4.704

Table 4.3: Critical path delay of each hardware component in pipeline microarchitecture (generated from Xilinx Vivado)

Hardware component	Delay (ns)		
	logic	net	total
Partial Reconfiguration			
PR unit (pr_inst)	3.858	10.024	13.882
PR controller unit (upr_ctrl)	3.087	7.255	10.342
CPU			
Data-path unit (udata_path)	3.858	10.024	13.882
Branch Predictor block (bbp_4way)	3.540	6.789	10.329
Main Control block (bmain_ctrl)	3.083	8.374	11.457
Arithmetic Logic Control block (balb_ctrl)	1.123	2.439	3.562
Register File block (brf)	3.540	6.789	10.329
CP0 block (bcp0)	4.068	7.983	12.051
Interlock block (bitl_ctrl)	1.763	6.647	8.410
Forwarding block (bfw_ctrl)	3.357	8.705	12.062
ALU block (balb)	5.398	8.715	14.113
Multiplier Block (bmult32)	7.096	7.325	14.421
Address Decoder block (baddr_decoder)	4.068	6.964	11.032
I/O System			
UART Controller unit (uuart)	3.610	8.269	11.879
- UART Baud Clock Generator block (bclkctr)	1.563	1.736	3.299
- UART Receiver block (brx)	3.944	6.469	10.413
- UART Transmitter block (btx)	3.486	8.183	11.669
SPI Controller unit (uspi)	3.734	8.718	12.452
- SPI Clock Generator block (bclk_gen)	1.801	2.688	4.489
- SPI Receiver block (bRX)	1.261	2.745	4.006
- SPI Transmitter block (bTX)	1.801	2.688	4.489
- SPI Input Output Control block (bio_ctrl)	1.385	2.652	4.037
GPIO Controller unit (ugpio)	3.610	8.582	12.192
Priority Interrupt Controller unit (upi_ctrl)	3.610	8.582	12.192
General Purpose Register unit (ugpr)	3.363	8.400	11.763
Memory System			
Boot ROM unit (uboot_rom)	3.548	6.774	10.322
Data and Stack RAM unit (uram)	5.416	9.044	14.460
Cache unit (icache)	4.076	7.968	12.044
- Cache Controller block (bcache_ctrl)	3.704	6.323	10.027
- Cache RAM block (bcache_ram)	3.540	6.789	10.329
Cache unit (dcache)	5.084	10.864	15.948
- Cache Controller block (bcache_ctrl)	5.084	10.864	15.948
- Cache RAM block (bcache_ram)	4.257	11.649	15.906
Memory Arbiter unit (umem_arbiter)	4.257	11.649	15.906
Flash Controller Unit (ufc)	4.257	11.649	15.906
- Flash Controller Clock Generator block (bfc_clk_gen)	1.021	2.131	3.152

Continued from Table 4.3

Hardware component	Delay (ns)		
	logic	net	total
- Flash Controller FSM block (bfc_fsm)	4.257	11.649	15.906
- Flash Controller Transmitter block (bfc_TX)	1.393	2.811	4.204
- Flash Controller Receiver block (bfc_RX)	1.293	3.422	4.715

Our experiment was conducted in 2 phases: physical functional test and power analysis. The first phase verifies the I/O controller functionality while the second phase performs the power analysis through: 1) the switching activity extracted from the Switching Activity Interchange Format (.saif) file from Xilinx Vivado post-implementation simulation; 2) the physical power analysis.

4.1 Physical Functional Test

The Xilinx Design Constraints (XDC) shown in Table 4.4 has been set for the implementation of our reconfigurable IoT processor on the Xilinx Nexys4 DDR FPGA development board.

Table 4.4: Design pin allocation on Nexys 4 DDR FPGA development board

Group	Design pin	Xilinx Nexys 4 DDR FPGA pin	Remark
Global	uirisc_clk_100mhz	E3	
	uirisc_rst	C12	
Quad SPI Flash Memory	uorisc_fc_sclk	E6	E6 pin hard-wired to Xilinx STARTUPE2 module
	uiorisc_fc_MOSI	K17	
	uirisc_fc_MISO1	K18	
	uirisc_fc_MISO2	L14	
	uirisc_fc_MISO3	M14	
	uorisc_fc_ss	L13	
SPI Controller	uiorisc_spi_miso	F6	
	uiorisc_spi_mosi	K1	
	uiorisc_spi_sclk	J2	
	uiorisc_spi_ss_n	G6	
UART Controller	uorisc_ua_tx_data	D4	
	uirisc_ua_rx_data	C4	
GPIO Controller	urisc_GPIO[0]	C17	PMOD JA
	urisc_GPIO[1]	D18	
	urisc_GPIO[2]	E18	
	urisc_GPIO[3]	G17	
	urisc_GPIO[4]	D17	
	urisc_GPIO[5]	E17	
	urisc_GPIO[6]	F18	
	urisc_GPIO[7]	G18	
	urisc_GPIO[8]	D14	PMOD JB
	urisc_GPIO[9]	F16	
	urisc_GPIO[10]	G16	
	urisc_GPIO[11]	H14	
	urisc_GPIO[12]	E16	
	urisc_GPIO[13]	F13	
	urisc_GPIO[14]	G13	
	urisc_GPIO[15]	H16	
	urisc_GPIO[16]	H17	LEDs
	urisc_GPIO[17]	K15	
	urisc_GPIO[18]	J13	
	urisc_GPIO[19]	N14	
	urisc_GPIO[20]	R18	
	urisc_GPIO[21]	V17	
	urisc_GPIO[22]	U17	
	urisc_GPIO[23]	U16	
	urisc_GPIO[24]	J15	Switches
	urisc_GPIO[25]	L16	

Continued from Table 4.4

Group	Design pin	Xilinx Nexys 4 DDR FPGA pin	Remark
	urisc_GPIO[26]	M13	
	urisc_GPIO[27]	R15	
	urisc_GPIO[28]	R17	
	urisc_GPIO[29]	T18	
	urisc_GPIO[30]	U18	
	urisc_GPIO[31]	R13	

4.1.1 GPIO

A test program was designed to test the functionality of the GPIO controller. Each GPIO pin can be set as either input or output pins. The GPIO test program flow is as follows:

1) Setup GPIO controller
2) Get inputs from switches (urisc_GPIO[24] to urisc_GPIO[31])
3) Turn on LEDs (urisc_GPIO[16] to urisc_GPIO[23]) when switches turn upward, off when switches turn downward
4) Repeat step 2 to 3

The first step setup the GPIO controller by configure bit-24 to bit-31 (urisc_GPIO[24] to urisc_GPIO[31]) as input pins and bit-16 to bit-23 (urisc_GPIO[16] to urisc_GPIO[23]) as output pins through the GPIODIR register located at 0xBFFFFE10. After configure the direction of the GPIO pins, bit-16 to bit-31 in the GPIOEN register located at 0xBFFFFE14 are set. When the GPIO received the input signals as in step 2, the input data is recorded in GPIODATA register located at 0xBFFFFE18. The GPIO test program will copy the input data to its respective output data field in the GPIODATA register, i.e. urisc_GPIO[16] as the output data of urisc_GPIO[24], urisc_GPIO[17] as the output data of urisc_GPIO[25] etc. Figure 4.1 demonstrates the GPIO test set up. The GPIO test program is running in loop, it is repeated until the power source is shut off.

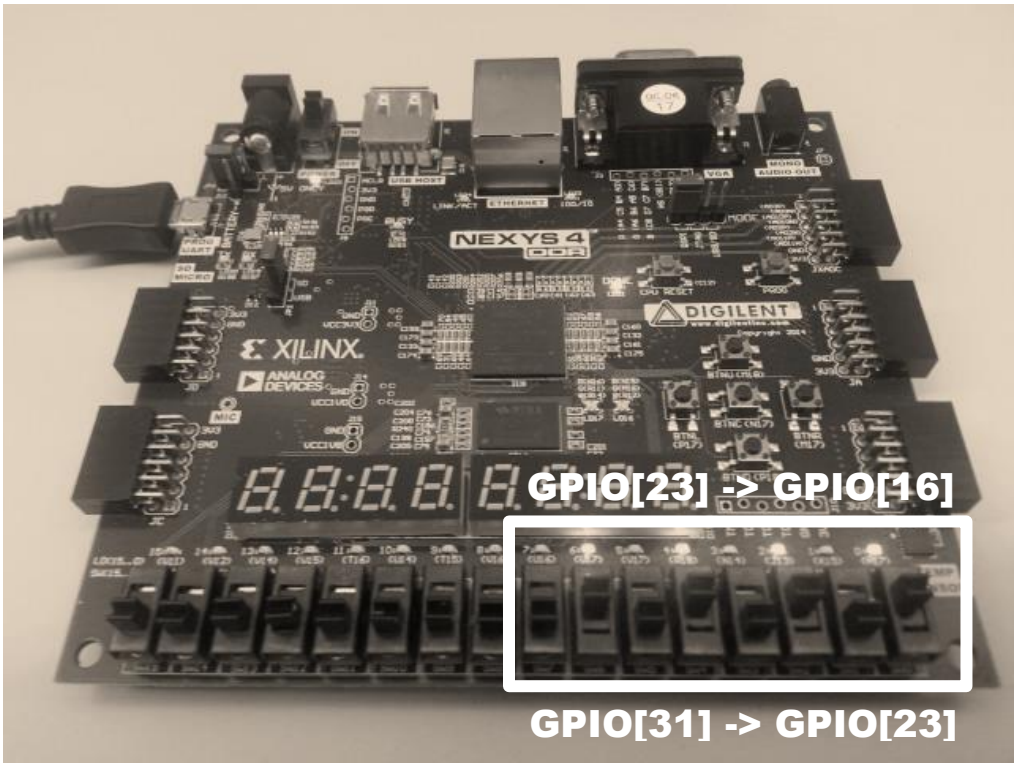


Figure 4.1: Demonstration of GPIO test set up

4.1.2 UART and SPI

A test program was designed to test the functionality of the UART and SPI controllers. Data is being passed from the SPI transmitter to the SPI receiver, then to the computer (via UART). The UART and SPI test program flow are as follows:

1) Create a variable in the Register File which will holds the values of 0x41 (ASCII = “A”) and to 0x5A (ASCII = “Z”). The value is changed in ascending order on every 500 ms.
2) Setup UART controller through UARTCR register
    a. Set baud rate (BAUD[2:0] = 010, 9600 baud)
    b. Start UART by setting the UARTEN bit.
3) Setup SPI controller through SPICR register
    a. Set SPI clock rate (SCR[2:0] = 0100, 625 kHz)
    b. Start SPI by setting the SPE bit
4) Start sending the variable through SPI uiorisc_spi_mosi (uiorisc_spi_miso at pin F6 and uiorisc_spi_mosi at pin K1 are connected together). Refer to Figure 4.2 for the connection.
5) Deactivate SPI by resetting the SPE bit in SPICR register
6) The received data from SPI uiorisc_spi_miso is sent to computer through UART.
7) Variable value counting up and reset the variable’s value to 0x41 when hits the upper bound (0x5A).
8) Delay 500 ms and then repeat step 3 to 8.

The first step set the variable that will be transmitted through SPI controller. Alphabet ‘A’ to ‘Z’ is loaded into the SPITDR register located at

0xBFFFFE26 every 500 ms. The same SPI controller was used for data receiving by connecting MISO pin (data receiving) to MOSI pin (data transmitting). The data received is loaded in the SPIRDR register located at 0xBFFFFE27 and the test program will transfer the respective data to UARTTDR located at 0xBFFFFE2A. UART controller will then transmit the data loaded in the UARTTDR register to the computer for debugging purposes. Figure 4.2 and Figure 4.3 show the board wire connection on Nexys4 DDR board and the value prompt on computer received through UART.

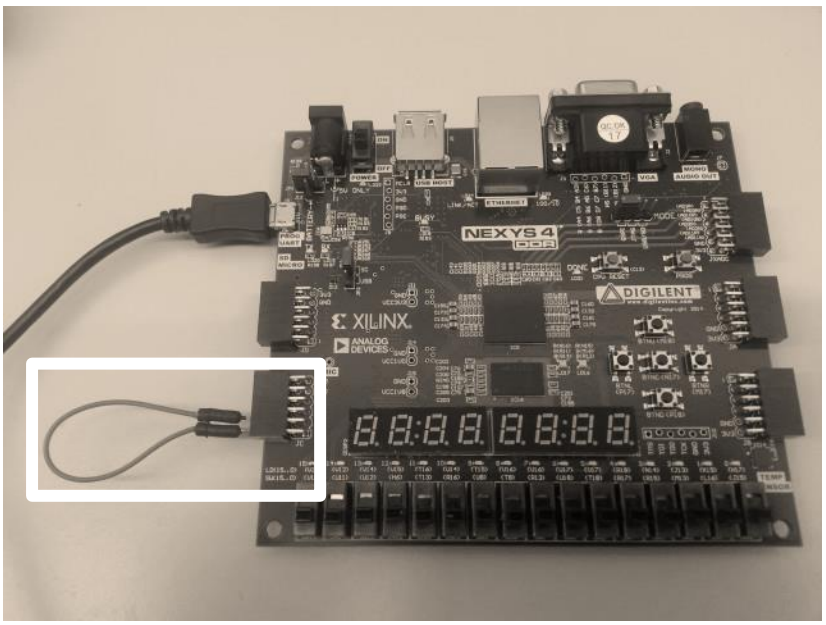


Figure 4.2: SPI uiorisc_spi_miso and uiorisc_spi_mosi connection

Figure 4.3: Data received on the computer through UART. The data is displayed using Putty.

4.1.3 Interrupt Handling

Interrupt-driven I/O access can helps to increase the throughput of the processor, in which the processor can perform extra tasks while waiting for an I/O to request (to interrupt the CPU) for data transfer. On the contrary, constantly checking I/O condition in polling I/O access scheme consumes the CPU cycle that is useful for executing more tasks. However, the program code developed for polling I/O access is relatively simple as compared with interrupt-driven I/O access. The development of the program code for interrupt-driven I/O access requires special handling in I/O configuration. Figure 4.4 illustrates the interrupt handling test program.

```
1  void main(){
2      setupUART();
3      while(1){
4          uart_send("HELLO WELCOME TO NEXYS4 DDR!\n\r");
5          delay_ms(500);
6      }
7  }
8
9  // UART interrupt the CPU when received a byte from external device
10 void UART_RXC_ISR(){
11     uart_send("\n\rCATCH\n\r");
12     clear_RXC_flag();
13 }
```

Figure 4.4: Pseudo code of interrupt handling test program

From Figure 4.4, the test program starts with UART configuration. The UART configuration includes enable UART interrupt (set UARTIE) through PICMASK register located at 0xBFFFFE21, enable UART Receive Complete Interrupt (set RXCIE) through UARTCR register located at 0xBFFFFE28, set the baud rate of the UART controller and activate the UART controller. A

message (“HELLO WELCOME TO NEXYS4 DDR!\n\r”) was transmitted to a computer through UART controller every 500 ms. When user strikes a key on the keyboard, a byte of data in the scan code form is sent from the computer to the IoT processor. The UART controller on our IoT processor will receive the data and interrupts the processor to jump to the exception handler (refer Section 3.5). Exception handler decodes the source of interrupt and then jump to the UART Receive Complete Interrupt Service Routine (RXCISR). The RXCISR send a message (“\n\rCATCH\n\r”) back to computer which will be displayed on the Putty. Then the UART’s ISR will clear the RXFF bit in the UARTSR register located at 0xBFFFFE29 to indicate the interrupt has been served. Figure 4.5 shows the message prompt on the computer screen.

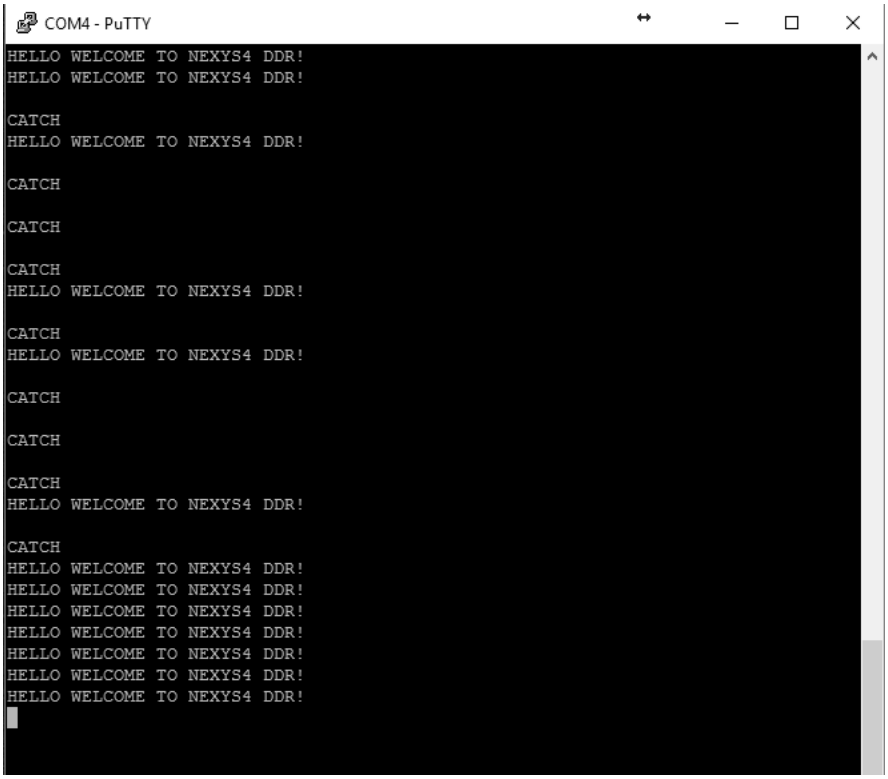


Figure 4.5: Demonstration of interrupt handling

4.2 Power Analysis

4.2.1 Simulation

4.2.1.1 Experiment Setup

The power analysis of both multi-cycle and pipeline microarchitectures is carried out separately from the simulation perspective, where the procedure flow is shown in Figure 4.6.

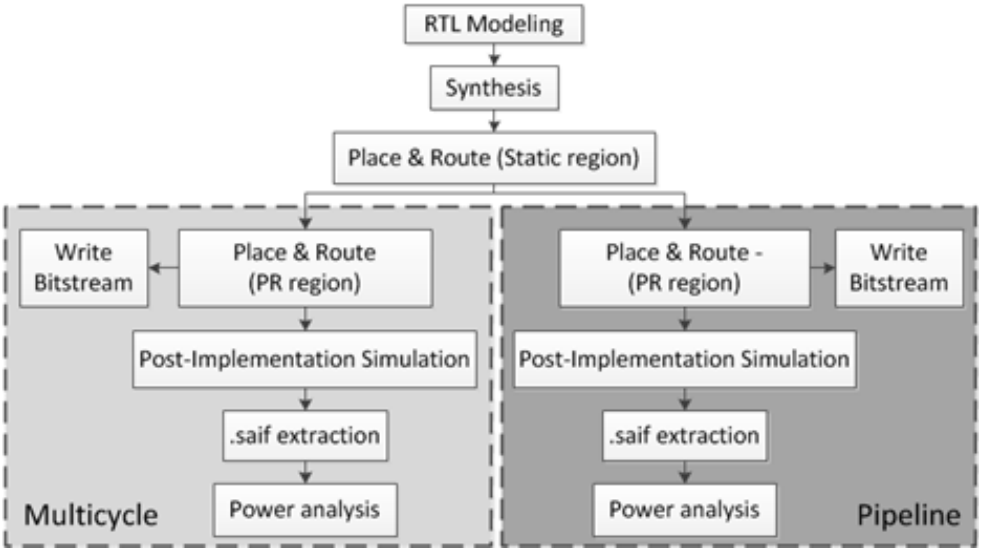


Figure 4.6: Power analysis procedure

Since the hardware components in the static region are shared by the multi-cycle and pipeline microarchitectures, it is only synthesized once to ensure its consistency. The difference comes when perform place and route process in the PR region. Imagine the PR region as a black box that contains the reconfigurable logic, once the place and route process in the static region is completed, the PR region's place and route process fused the PR region's logic with the static region's logic. Since the logic within PR region are

different for both multi-cycle and pipeline microarchitectures and thus, the place and route for the PR region is performed separately.

For a common IoT task, the task time consists of active time (T_{active}) and idle time (T_{idle}). T_{active} defines the performance of the processor while T_{idle} creates delay used to align with the user target task period. The task period will vary among the IoT application. Some of the IoT application may not have the T_{idle} due to high computational power is required to run a heavy workload task. Thus, our experiment focuses on the power analysis based on the full capability of the processor, i.e. the total time of a task consists of only T_{active}.

The post-implementation simulation simulates the program flow as follows with referring to the pseudo code of AES-128 encryption shown in Figure 4.7.

1) Get 16-bytes of data from SPI EEPROM (data collection)
2) Encrypt data received using AES-128 (data processing)
3) Send the encrypted data through UART (data transmission)
4) Toggle urisc_GPIO[0] pin to indicate the end of current loop and start of the new loop
5) Repeat step 1 to 4.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])                  // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                              // See Sec. 5.1.1
      ShiftRows(state)                             // See Sec. 5.1.2
      MixColumns(state)                            // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Figure 4.7: AES128 encryption pseudo code (Nk=4, Nb=4, Nr=10)
Source: National Institute of Standards and Technology (2001) 'FIPS 197: Advanced Encryption Standard'

The post-implementation simulation used an SPI EEPROM simulation model in place of the sensor data storage. The test program was executed in loop for 200 ms, with switching activities and power consumption data collected (through the information gather from *.saif* file) at every 20ms interval. Every loop of the test program will be recorded with the start time and the stop time by monitoring urisc_GPIO[0] pin. The result collected will be discussed in the next subsection.

4.2.1.2 Result & Discussion

Table 4.5 and Table 4.6 are the results collected from the phase 2 experiment.

Table 4.5: Average switching rate (millions of transitions per seconds) based on Artix-7 XC7A100T

Sim. time (ms)	Multi-cycle switching rate (Mtr/s)				Pipeline switching rate (Mtr/s)			
	signal	logic	BRAM	I/O	signal	logic	BRAM	I/O
20	1.09	0.38	0.87	14.01	7.69	1.00	0.96	14.16
40	1.09	0.40	0.93	13.93	8.38	1.08	1.02	14.03
60	1.08	0.41	0.94	13.86	8.60	1.10	1.03	13.99
80	1.09	0.41	0.95	13.85	8.61	1.11	1.04	13.98
100	1.09	0.41	0.95	13.84	8.61	1.11	1.04	13.98
120	1.08	0.41	0.96	13.82	8.62	1.11	1.05	13.98
140	1.08	0.41	0.96	13.83	8.69	1.12	1.05	13.97
160	1.08	0.41	0.96	13.82	8.72	1.12	1.05	13.96
180	1.08	0.41	0.96	13.82	8.77	1.13	1.05	13.95
200	1.08	0.41	0.96	13.82	8.79	1.13	1.05	13.95

Based on the results shown in Table 4.5, the average switching rate of the multi-cycle execution is lesser than the pipeline execution. This condition explained why the dynamic power consumption of the multi-cycle execution is lesser than the pipeline execution as shown in Table 4.6. From the data shown in Table 4.6, the task completed by the multi-cycle and pipeline executions are 1.88 and 2.39 tasks respectively, which consume 85.11 mJ and 108.79 mJ of dynamic energy per task respectively. The multi-cycle execution processes slower by 21.38% but reduces 21.77% dynamic energy than the pipeline execution. At 40 ms simulation time, multi-cycle and pipeline executions have completed 4.04 and 5.13 tasks respectively. The task completed is more than twice as compared to the 20 ms simulation time. The system resetting,

bootloading, flash memory initialization and cache miss consume a start-up overhead of simulated 2.5 ms, which explained the task completed at 40 ms is more than twice than 20 ms. Multi-cycle and pipeline executions at 40 ms simulation time consume 79.21 mJ and 109.16 mJ of dynamic energy per task respectively. The multi-cycle execution is slower than pipeline execution by 21.27%, but consuming 27.44% lesser dynamic energy. The remaining simulations (60 - 200ms) show 32.33% to 33.06% dynamic energy reduction for the pipeline versus multi-cycle execution with a corresponding computational performance reduction of 20.31% - 21.16%.

The developed reconfigurable soft-core IoT processor always starts up with the multi-cycle microarchitecture as the default microarchitecture for power saving purpose. When PR occurs, an overhead of 44 ms (at 20 MHz system clock) is required, which should be taken into consideration when developing program with time-critical tasks. The PR overhead can be reduced by: 1) Increasing the clock frequency up to 100 MHz (based on ICAP requirement); 2) Buffer the partial bitstream in an FPGA Block RAM; 3) Using a Direct Memory Access (DMA) controller when copying the partial bitstream. But all are at the expense of more energy and resources used (Pezzarossa, L., Schoeberl, M. and Sparsø, J., 2017).

The static power consumption (about 90% of the total power) shown in Table 4.6 was not taken into account in our analysis since it is technology dependent, which our implemented technique has no direct relationship with the technology. As the technology in transistor scaling is improving, the static

power will be reduced. This will provide extra benefit to our platform (total power will be reduced when static power reduce). The result from Table 4.5 and Table 4.6 are based on 20MHz operating frequency. The typical operating frequencies used by a variety of sensor nodes cover a wide range from 8MHz to 180MHz (Gajjar, S. et al., 2014). As the operating frequency increased, dynamic power consumption is expected to dominate the total power consumption. Hence, a significant amount of dynamic power reduction result can be observed in Table 4.6. This makes our target focuses on analyzing the dynamic power consumption.

The simulation based power analysis has presented the quantitative differences between multi-cycle and pipeline microarchitectures, in terms of the computational speed and the dynamic energy consumption per task. However, the physical power analysis must be performed based on the following justifications:

1) The simulation experiment is unable to shows the competitive advantages of using the combination of multi-cycle and pipeline microarchitectures to perform an IoT task, i.e. the simulation experiment only shows the IoT task running in each microarchitecture independently. Thus, the energy used for PR cannot be measured.
2) Xilinx Vivado is unable to simulate the behavior of PR (currently unsupported for Xilinx Vivado 2017.2). Thus, the energy used for PR is unknown.

Table 4.6: Power and performance analysis based on Artix-7 XC7A100T

Sim. time (ms)	Multi-cycle				Pipeline				Dynamic energy reduction (%)	Computational performance reduction (%)
	Power		task completed	Dynamic energy/task (mJ)	Power		task completed	Dynamic energy /task (mJ)		
	Static (W)	Dynamic (W)			Static (W)	Dynamic (W)				
20	0.097	0.008	1.88	85.11	0.097	0.013	2.39	138.30	21.77	21.38
40	0.097	0.008	4.04	79.21	0.097	0.014	5.13	138.61	27.44	21.27
60	0.097	0.008	6.21	77.29	0.097	0.015	7.88	144.93	32.33	21.16
80	0.097	0.008	8.37	76.46	0.097	0.015	10.58	143.37	32.59	20.86
100	0.097	0.008	10.54	75.90	0.097	0.015	13.29	142.31	32.75	20.67
120	0.097	0.008	12.71	75.53	0.097	0.015	15.99	141.62	32.90	20.53
140	0.097	0.008	14.88	75.27	0.097	0.015	18.69	141.13	33.01	20.36
160	0.097	0.008	17.04	75.12	0.097	0.015	21.39	140.85	33.05	20.36
180	0.097	0.008	19.19	75.04	0.097	0.015	24.10	140.70	33.02	20.35
200	0.097	0.008	21.36	74.91	0.097	0.015	26.81	140.45	33.06	20.31

Notes: **1) Dynamic energy reduction = (Dynamic energy/task of pipeline - Dynamic energy/task of multi-cycle) / Dynamic energy/task of pipeline x 100%**
2) Computational performance reduction = (Task completed in pipeline - Task completed in multi-cycle) / Task completed in pipeline x 100%

4.2.2 Physical Power Analysis

Due to the limitation of the simulation based power analysis, as mentioned in Section 4.2.1.2, we have carried out the physical power analysis. Our hypothesis in this experiment is that using the combination of pipeline microarchitecture to perform data processing and multi-cycle microarchitectures to perform data transmission part of an IoT task can provide better energy usage. Thus, both multi-cycle and pipeline microarchitectures will be used to test the following conditions:

1) Multi-cycle microarchitecture for data collection and processing
2) Pipeline microarchitecture for data collection and processing
3) Multi-cycle microarchitecture for data transmission
4) Pipeline microarchitecture for data transmission

The physical power analysis experiment used the same IoT program flow as the simulation based power analysis. However, we have used various data sizes (64, 128, 256, 512 and 1024 bytes) for collect, process and transmit. We need to consider the PR overhead (44 ms) since this is directly translated into energy consumption. The PR of the microarchitecture is only advantageous in energy saving if processing and transmitting the data size takes longer time than the PR overhead time. Otherwise, triggering the PR process unnecessarily will waste energy. The combination of the tests is summarized in Table 4.7. MM and PP combinations represents the common practice in designing IoT sensor nodes by using single microarchitecture for fast computational performance (PP combination) or low power (MM combination) design goal. PM combination represents our proposed technique

to reconfigure the microarchitecture based on different workload characteristics to further optimize the energy usage while achieving acceptable computational speed performance.

Table 4.7: Combination of test

Combination of microarchitectures	Data size (bytes)				
	64	128	256	512	1024
MM[1]	MM64	MM128	MM256	MM512	MM1024
MP[2][5]	MP64	MP128	MP256	MP512	MP1024
PP[3]	PP64	PP128	PP256	PP512	PP1024
PM[4][5]	PM64	PM128	PM256	PM512	PM1024

Notes:

[1] MM – Data collection, processing and transmission using multi-cycle microarchitecture

[2] MP – Data collection and processing using multi-cycle microarchitecture and data transmission using pipeline microarchitecture

[3] PP – Data collection and processing using pipeline microarchitecture and data transmission using pipeline microarchitecture

[4] PM – Data collection and processing using pipeline microarchitecture and data transmission using multi-cycle microarchitecture

[5] PR is require, PR overhead take into account in the energy usage

4.2.2.1 Experiment Setup

Since a fixed voltage supply (1V) is used, our intention in this experiment is to measure the current consumption and the power consumption can be calculate using the electric power formula, P=VI. A high side current measurement circuit is constructed to measure the current consumption of the developed soft-core IoT processor running at 20 MHz system clock over a certain period. Figure 4.8 shows the current measurement circuit connection.

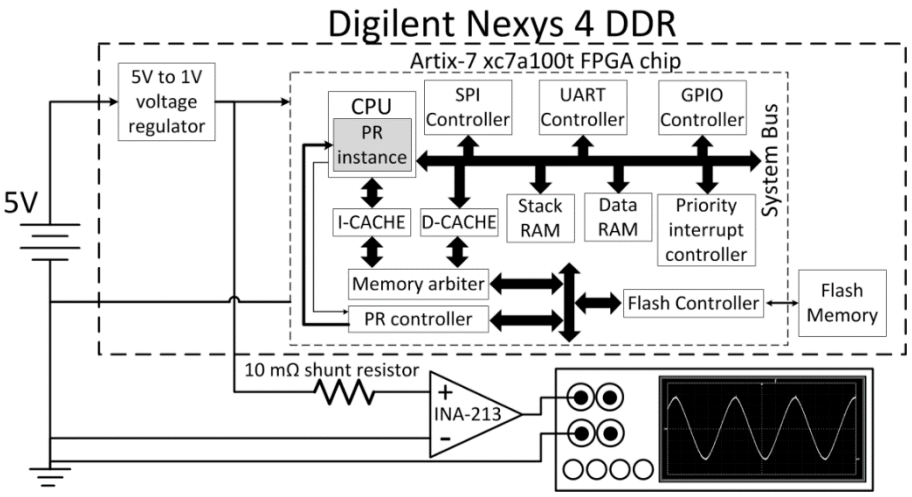


Figure 4.8: High side current measurement circuit

Current consumption is measured through measuring the voltage drop across the 10 mΩ with 1% tolerance shunt resistor. A Tektronix TBS1000B-EDU oscilloscope is used to measure and collect the current consumption sampling data for further analysis. However, based on our investigation, the differences in current consumption measured from both multi-cycle and pipeline microarchitecture is relatively small, which is in terms of mA range. Thus, an extra TI INA-213 current-shunt monitor is used to amplify the measured signal by 50 times before pass into the oscilloscope.

4.2.2.2 Result & Discussion

The dynamic power consumptions of the combination of tests for varying data from 64 Bytes to 1024 Bytes as shown in Table 4.7 are shown in Figure 4.9, Figure 4.10, Figure 4.11, Figure 4.12 and Figure 4.13, with following denotations:

MM: Data collection and processing and data transmission using multi-cycle microarchitecture

MP: Data collection and processing using multi-cycle microarchitecture and data transmission using pipeline microarchitecture

PP: Data collection and processing using pipeline microarchitecture and data transmission using pipeline microarchitecture

PM: Data collection and processing using pipeline microarchitecture and data transmission using multi-cycle microarchitecture

$\mathbf{A_M}$: Data processing using multi-cycle microarchitecture

$\mathbf{T_M}$: Data transmission using multi-cycle microarchitecture

$\mathbf{A_P}$: Data processing using pipeline microarchitecture

$\mathbf{T_P}$: Data transmission using pipeline microarchitecture

PR: Partial Reconfiguration

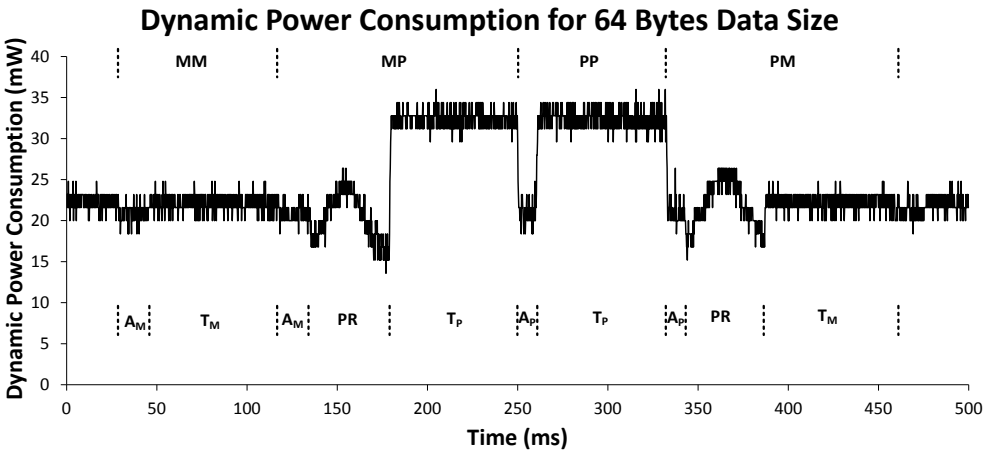


Figure 4.9: Dynamic power consumption for 64 bytes data size.

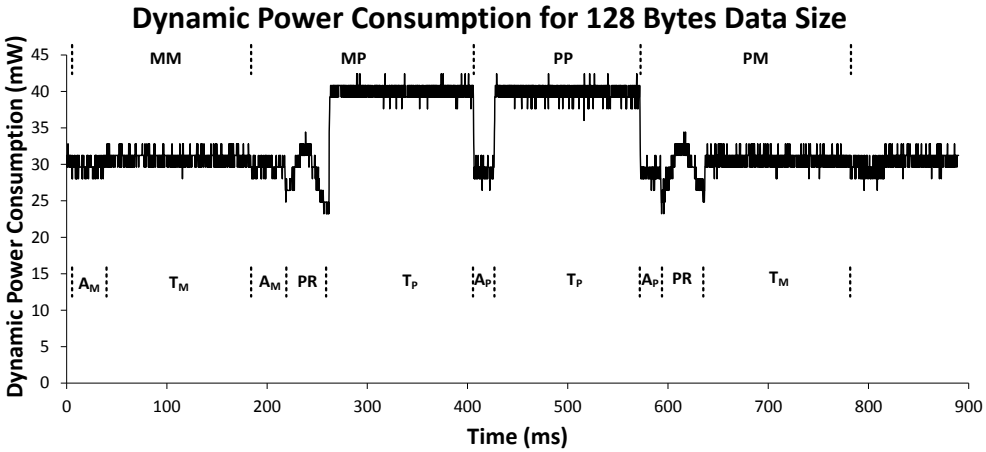


Figure 4.10: Dynamic power consumption for 128 bytes data size.

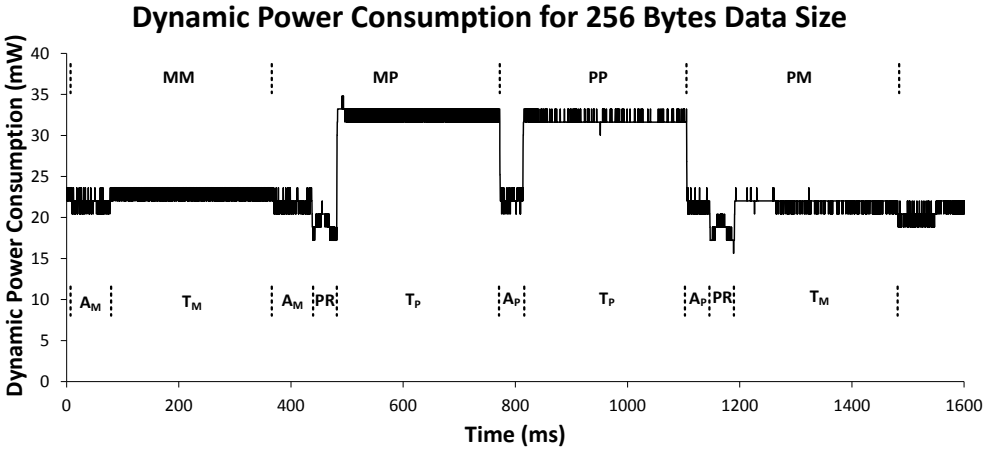


Figure 4.11: Dynamic power consumption for 256 bytes data size.

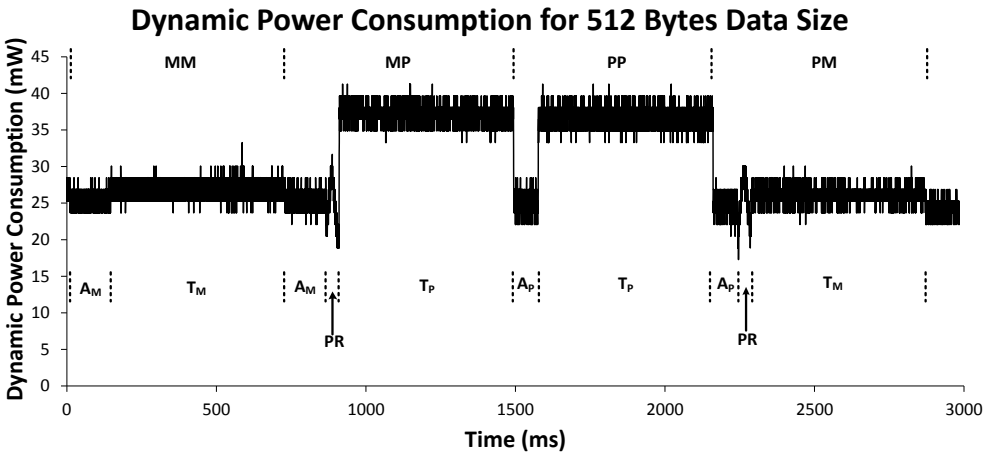


Figure 4.12: Dynamic power consumption for 512 bytes data size.

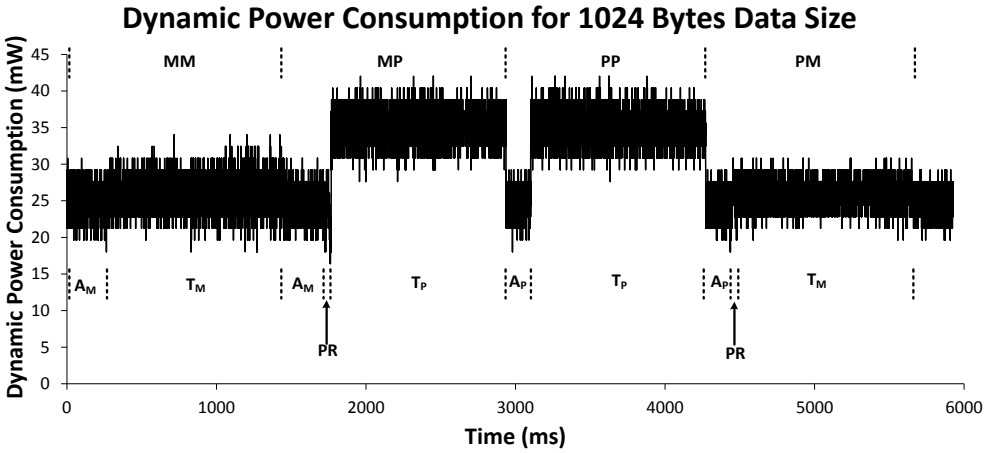


Figure 4.13: Dynamic power consumption for 1024 bytes data size.

From Figure 4.9, Figure 4.10, Figure 4.11, Figure 4.12 and Figure 4.13, we observed that the area occupied on the graph (energy consumption) for PR is decreasing as the data size increases from 64 bytes to 1024 bytes. This is due to the dynamic energy consumed for PR is constant whenever invoking the PR process. PR has a significant effect on the total energy used per task for smaller data size (i.e. 64 bytes and 128 bytes), but relatively less significant in larger data size (i.e. 256 bytes onwards) since energy consumed by the data collection, processing and transmitting works are far greater. The time taken

for the task (task time) to complete for each combination is shown in Figure 4.14, while Figure 4.15 show the dynamic energy consumed by each combination.

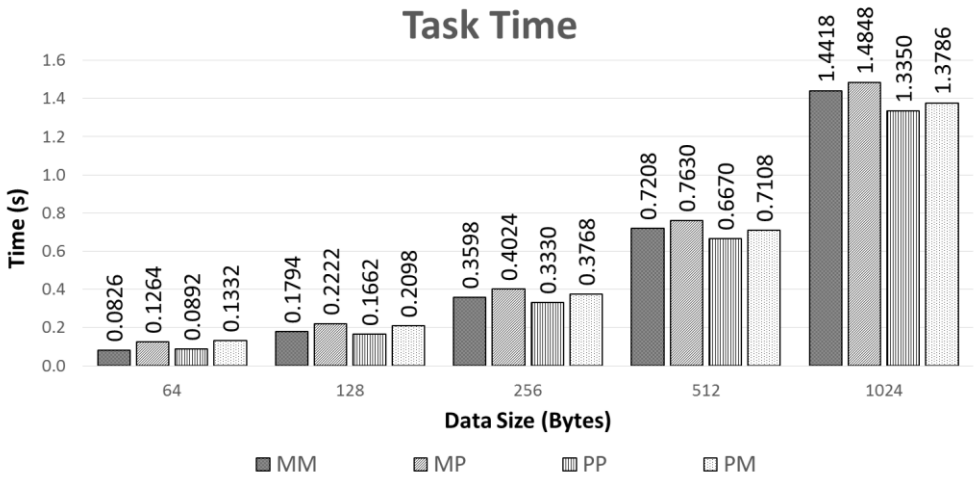


Figure 4.14: Task time used by MM, MP, PP and PM for 64, 128, 256, 512 and 1024 bytes data size.

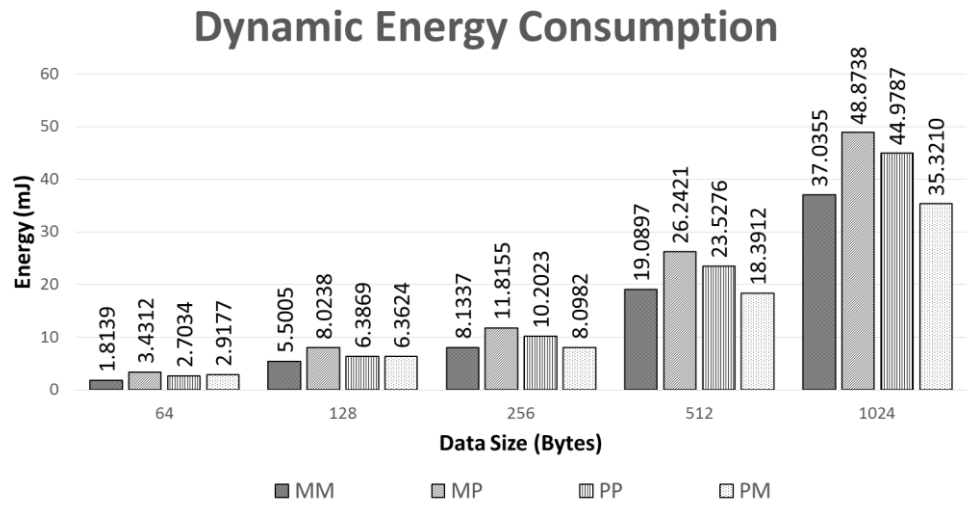


Figure 4.15: Dynamic energy consumption of MM, MP, PP and PM for 64, 128, 256, 512 and 1024 bytes data size.

From Figure 4.14, for 1024 bytes data size, PP combination shows the highest computational performance with the least task time while MP

combination is the least performing combination that requires longer time to compute due to PR overhead. We can safely omit the use of MP combination due to its conflicting usage, whereby low computational multi-cycle microarchitecture is used for data collection and processing while power hungry pipeline microarchitecture is used for low computational requirement data transmission. PM combination is always faster (despite having PR overhead) than MM combination because the most time consuming part (data collection and processing) is executed with pipeline microarchitecture. PM combination is 4.38% faster than MM combination but 3.27% slower than PP combination.

Despite the superiority of PP combination in computational performance, Figure 4.15 shows that PP combination requires significantly more energy to complete the task as compared to PM and MM combinations. Fast speed performance and low dynamic energy consumption are two contradicting design goals that cannot be achieved by utilizing single microarchitecture. However, using PM combination can achieve the lowest energy consumption among the micro-architectural configurations. Considering the case for 1024 bytes data size, PM combination is 4.63% and 21.47% more energy efficient compared to MM and PP combinations respectively. Our proposed technique, PM combination can achieve better energy efficiency when the data size increases. PM combination is preferred over MM combination for data size that is larger than 256 Bytes due to its superiority in terms of energy efficiency and performance. This finding is important as many wireless sensor networks actually employ multi-hop

techniques (S. Y. Liew, C. K. Tan, M. L. Gan, H. G. Goh., 2018) in practical on-field deployment, wherein the sensor nodes collect large amount of data and take turn (based on the designed protocol) to transmit it. Under such scenario, the data size can be much larger than 1024 bytes, which highlights the potential energy reduction of our proposed technique. In order to improve the dynamic energy consumption without losing too much of computational performance, the energy-delay product metric is used, which as shown in Figure 4.16.

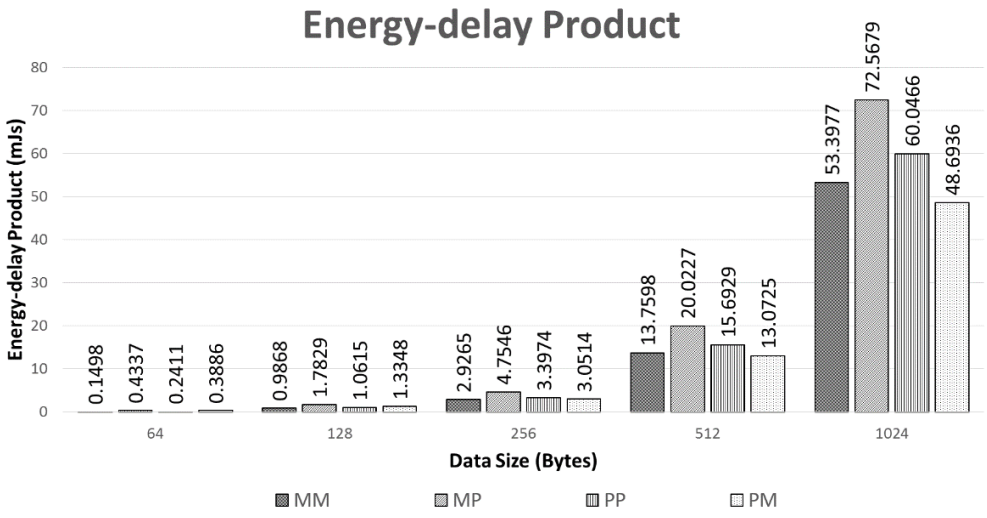


Figure 4.16: Energy-delay product of MM, MP, PP and PM for 64, 128, 256, 512 and 1024 bytes data size.

PM combination only achieves moderate timing performance as compared to PP and MM combinations. But it consumes the least energy, which is 8.81% and 18.91% lesser as compared to MM and PP combinations respectively. Thus, it has the least energy-delay product. This makes PM combination the most optimized option when taking into accounts both energy and computational performance for on-field IoT application. This also implies

that the proposed technique achieved better performance-energy trade-off for IoT applications compared to conventional method (DVS, DFS, DVFS, clock gating and power gating) that only have single microarchitecture. On the other hand, if computational performance is required by an IoT task, then PP combination will be used.

4.3 Summary

In this chapter, we have presented our verification on the developed reconfigurable soft-core IoT processor. The developed reconfigurable soft-core IoT processor is able to perform data aggregation, data processing and data transmission. The experimental result shows that, for 1024 bytes data size, PM combination can achieve the least energy-delay product, which is 8.81% and 18.91% lesser compared to multi-cycle (MM combination) and pipeline (PP combination) microarchitectures respectively. For sensor nodes that process larger data sizes, which larger than 1024 bytes, the energy-delay product can be further reduced.

CHAPTER 5

CONCLUSIONS & FUTURE WORK

5.1 Conclusions

An FPGA-based soft-core IoT System on a Chip (SoC) has been developed to provide rapid customization and reconfigurable based on the need of an IoT application. The development cost for FPGA-based soft-core products are justifiable for small to medium scale production volume. It has the cost advantage over ASIC approach for coping with designs that are still undergoing development.

In summary, the dissertation has provided robust evidence with which to answer the three main research challenges as following:

1) For on-field IoT application, the IoT sensor node is expected to perform data aggregation, data processing and data transmission. Our first issue is to establish the basic requirement (computation speed, power consumption and functionality) of an IoT processor suitable to be used as an IoT sensor node.

    - This dissertation has reviewed the specification of the existing IoT platform required by each IoT application. From the review, an FPGA based soft-core IoT SoC is proposed and developed which has the advantage in highly customizability that is able to cope with the basic requirement needs in each IoT application.

2) In IoT application, low power consumption is the essential issue. Our next issue will be on what is the technique used and how to enable the developed reconfigurable soft-core IoT processor to tune based on the computational needs from the environment requirement to have the optimal power saving scheme?

    - The PR between multi-cycle and pipeline executions has been proved to satisfy the varying performance-power tradeoff requirements from each IoT application. Multi-cycle execution is used to reduce the dynamic power consumption of the processor at the expense of providing lower computational speed, while pipeline execution provides higher computational speed but consume more dynamic power.

3) How to verify the performance of the design in terms of computational speed and power using conventional FPGA chip?

    - An IoT program that consists of intensive data processing requirement, i.e. AES-128, has been used to identify the computational speed and power of the developed FPGA-based soft-core IoT SoC. The experimental result shows that for 1024 bytes data size, PM combination is able to reduce dynamic energy consumption by 4.63% and 21.47% respectively, compare to multi-cycle (MM combination) and pipeline (PP combination) only microarchitectures. Moreover, PM combination can achieve the least energy-delay product, which is 8.81% and 18.91% lesser compared to multi-cycle (MM combination) and pipeline (PP combination) microarchitectures

respectively. For sensor nodes that process larger data sizes, which larger than 1024 bytes, the energy-delay product can be further reduced.

As the technology in transistor scaling is improving steadily over the past few years, the static power consumption of FPGA also reduces dramatically (Tajalli, A. and Leblebici, Y., 2011). Hence, the bottleneck of the low power design has shifted towards the reduction of dynamic power consumption. We have presented a novel technique to further reduce the dynamic power consumption based on micro-architectural level design. This research work showed a proof of concept prototype whereby, with the PR feature offered by FPGAs, multi-cycle and 5-stage pipeline executions are designed to run intermittently in a processor core to achieve better performance-power tradeoff. Other combinations, for example, multi-cycle, 5-stage and 8-stage pipeline executions can also be used for more refined performance-power tradeoff. The proposed technique can be applied to other FPGA platforms as well. Therefore, all objective that stated were met.

5.2 Future work

Currently, the system test programs are developed in assembly language. In fact, it requires extra effort when it comes to debug and trace for the test program code since long test program in assembly form is generated. The existing GNU Compiler Collection (GCC) may help to resolve the issue. High level programming languages help to reduce the programming complexity when developing a test program and increase the program code readability. However, due to the limited MIPS ISA compatible instruction support in our research work, GCC may generates the unsupported MIPS ISA compatible instructions. The developed reconfigurable soft-core IoT SoC will serve the unsupported instruction as reserved instruction, which will trigger an Undefined Instruction exception. Besides that, the pre-built high-level programming language (HLL) test program code has to be manually converted to the assembly form that suits to our system when without the use of a compiler. Thus, an Original Equipment Manufacturer (OEM) compiler is required.

Other useful existing work such as DVFS and clock gating can be integrated with our proposed technique to achieve better power efficiency. DVFS technique scales down the system frequency and voltage level of the processor when low computation is required. Thus, it is expected to have lower static and dynamic power consumption. Clock gating in another side reduces dynamic power consumption by deactivating the clock source of the idle hardware components.

REFERENCES

Abid, F. and Izeboudjen, N. (2015a) 'Technology-independent approach for FPGA and ASIC implementations', in 2015 4th International Conference on Electrical Engineering (ICEE). IEEE, pp. 1–4. doi: 10.1109/INTEE.2015.7416610.

Abid, F. and Izeboudjen, N. (2015b) 'ASIC implementation of an OpenRISC based SoC for VoIP application', in 2015 6th International Conference on Information and Communication Systems (ICICS). IEEE, pp. 64–67. doi: 10.1109/IACS.2015.7103203.

Akyildiz, I. F. et al. (2002) 'A survey on sensor networks', IEEE Communications Magazine, 40(8), pp. 102–114. doi: 10.1109/MCOM.2002.1024422.

Al-Fuqaha, A. et al. (2015) 'Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications', IEEE Communications Surveys & Tutorials, 17(4), pp. 2347–2376. doi: 10.1109/COMST.2015.2444095.

Becker, J. et al. (2007) 'Dynamic and Partial FPGA Exploitation', Proceedings of the IEEE, 95(2), pp. 438–452. doi: 10.1109/JPROC.2006.888404.

Bhattacharyya, D., Kim, T. and Pal, S. (2010) 'A Comparative Study of Wireless Sensor Networks and Their Routing Protocols', Sensors, 10(12), pp. 10506–10523. doi: 10.3390/s101210506.

Borges, L. M., Velez, F. J. and Lebres, A. S. (2014) 'Survey on the Characterization and Classification of Wireless Sensor Network Applications', IEEE Communications Surveys & Tutorials, 16(4), pp. 1860–1890. doi: 10.1109/COMST.2014.2320073.

Buratti, C. et al. (2009) 'An overview on wireless sensor networks technology and evolution', Sensors, 9(9), pp. 6869–6896. doi: 10.3390/s90906869.

Cardona, L. A. and Ferrer, C. (2015) 'AC_ICAP: A Flexible High Speed ICAP Controller', International Journal of Reconfigurable Computing, 2015, pp. 1–15. doi: 10.1155/2015/314358.

Choi, K., Soma, R. and Pedram, M. (2004) 'Dynamic voltage and frequency scaling based on workload decomposition', in Proceedings of the 2004 international symposium on Low power electronics and design - ISLPED '04. New York, New York, USA: ACM Press, p. 174. doi: 10.1145/1013235.1013282.

Chow, C. T. et al. (2005) 'Dynamic voltage scaling for commercial FPGAs', in Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005. IEEE, pp. 173–180. doi: 10.1109/FPT.2005.1568543.

Cypress (2017) ‘128 Mbit (16 Mbyte)/256 Mbit (32 Mbyte) 3.0V SPI Flash Memory’ [Online]. Available: http://www.cypress.com/file/177966/download [Accessed: Nov. 8, 2017] D. Sweetman (2006) ‘See MIPS run 2nd edition’, Elsevier/Morgan Kaufmann. ISBN: 9780080525235

de la Piedra, A., Braeken, A. and Touhafi, A. (2012) ‘Sensor Systems Based on FPGAs and Their Applications: A Survey’, Sensors, 12(12), pp. 12235–12264. doi: 10.3390/s120912235.

de la Piedra, A. et al. (2013) ‘Wireless sensor networks for environmental research: A survey on limitations and challenges’, Eurocon 2013, (July), pp. 267–274. doi: 10.1109/EUROCON.2013.6624996.

Gajjar, S. et al. (2014) ‘Comparative analysis of wireless sensor network motes’, in 2014 International Conference on Signal Processing and Integrated Networks (SPIN). IEEE, pp. 426–431. doi: 10.1109/SPIN.2014.6776991.

Garcia, R., Gordon-Ross, A. and George, A. D. (2009) ‘Exploiting Partially Reconfigurable FPGAs for Situation-Based Reconfiguration in Wireless Sensor Networks’, in 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines. IEEE, pp. 243–246. doi: 10.1109/FCCM.2009.45.

Gomes, T. et al. (2015) ‘Towards an FPGA-based edge device for the Internet of Things’, in 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA). IEEE, pp. 1–4. doi: 10.1109/ETFA.2015.7301601.
Gu, C. (2016) Building Embedded Systems, O’Reilly & Associates. doi: 10.1007/978-1-4842-1919-5.

Gungor, V. C., Lu, B. and Hancke, G. P. (2010) ‘Opportunities and Challenges of Wireless Sensor Networks in Smart Grid’, IEEE Transactions on Industrial Electronics, 57(10), pp. 3557–3564. doi: 10.1109/TIE.2009.2039455.

Hansen, S. G., Koch, D. and Torresen, J. (2013) ‘Simulation framework for cycle-accurate RTL modeling of partial run-time reconfiguration in VHDL’, in 2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC). IEEE, pp. 1–8. doi: 10.1109/ReCoSoC.2013.6581519.

Harris, D. M. and Harris, S. L. (2013) Digital design and computer architecture. doi: 10.1016/B978-0-12-800056-4.00022-4.

Hempstead, M. et al. (2008) ‘Survey of Hardware Systems for Wireless Sensor Networks’, Journal of Low Power Electronics, 4(1), pp. 11–20. doi: 10.1166/jolpe.2008.156.

Hennessy, J. L. and Patterson, D. A. (2012) ‘Computer architecture: a quantitative approach’, Elsevier. doi: 10.1.1.115.1881.

Hinkelmann, H., Zipf, P. and Glesner, M. (2007) ‘A Domain-Specific Dynamically Reconfigurable Hardware Platform for Wireless Sensor Networks’, in 2007 International Conference on Field-Programmable Technology. IEEE, pp. 313–316. doi: 10.1109/FPT.2007.4439274.

Hosseinabady, M. and Nunez-Yanez, J. L. (2014) ‘Run-time power gating in hybrid ARM-FPGA devices’, in 2014 24th International Conference on Field Programmable Logic and Applications (FPL). IEEE, pp. 1–6. doi: 10.1109/FPL.2014.6927503.

Hosseinabady, M. and Nunez-Yanez, J. L. (2015) ‘Energy optimization of FPGA-based stream-oriented computing with power gating’, in 2015 25th International Conference on Field Programmable Logic and Applications (FPL). IEEE, pp. 1–6. doi: 10.1109/FPL.2015.7293946.

Hongzhi Liu and Bergmann, N. W. (2010) ‘An FPGA softcore based implementation of a bird call recognition system for sensor networks’, in 2010 Conference on Design and Architectures for Signal and Image Processing (DASIP). IEEE, pp. 1–6. doi: 10.1109/DASIP.2010.5706238.

Hsieh, C.-M. et al. (2014) ‘Hardware/software co-design for a wireless sensor network platform’, in Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis - CODES ’14. New York, New York, USA: ACM Press, pp. 1–10. doi: 10.1145/2656075.2656086.

I. C. Bertolotti and Tingting Hu. (2015) ‘Embedded Software Development: The Open-Source Approach’, ISBN: 9781466593923

Jawhar, I., Mohamed, N. and Agrawal, D. P. (2011) ‘Linear wireless sensor networks: Classification and applications’, Journal of Network and Computer Applications. Elsevier, 34(5), pp. 1671–1682. doi: 10.1016/j.jnca.2011.05.006.

Johnson, D. (2009) ‘Implementing serial bus interfaces with general purpose digital instrumentation’, in 2009 IEEE AUTOTESTCON. IEEE, pp. 125–129. doi: 10.1109/AUTEST.2009.5314057.

Kateeb, A. El, Ramesh, A. and Azzawi, L. (2008) ‘Wireless Sensor Nodes Processor Architecture and Design’, 22nd International Conference on Advanced Information Networking and Applications - Workshops (aina workshops 2008), pp. 892–897. doi: 10.1109/WAINA.2008.177.

Kiat, W. P. et al. (2017) ‘A Comprehensive Analysis on Data Hazard for RISC32 5-Stage Pipeline Processor’, pp. 2–7. doi: 10.1109/WAINA.2017.20.

Koch, D. et al. (2012) ‘Partial Reconfiguration on FPGAs in Practice Tools and Applications’, in ARCS Workshops.

Krasteva, Y. E. et al. (2008) ‘Remote HW-SW reconfigurable Wireless Sensor nodes’, in 2008 34th Annual Conference of IEEE Industrial Electronics. IEEE, pp. 2483–2488. doi: 10.1109/IECON.2008.4758346.

Kuon, I. and Rose, J. (2007) ‘Measuring the Gap Between FPGAs and ASICs’, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 26(2), pp. 203–215. doi: 10.1109/TCAD.2006.884574.

Kuon, I., Tessier, R. and Rose, J. (2007) ‘FPGA Architecture: Survey and Challenges’, Foundations and Trends® in Electronic Design Automation, 2(2), pp. 135–253. doi: 10.1561/1000000005.

Lazarescu, M. T. (2013) ‘Design of a WSN platform for long-term environmental monitoring for IoT applications’, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 3(1), pp. 45–54. doi: 10.1109/JETCAS.2013.2243032.

Lloret, J. et al. (2009) ‘A wireless sensor network deployment for rural and forest fire detection and verification’, Sensors, 9(11), pp. 8722–8747. doi: 10.3390/s91108722.

McDonald, E. (2008) ‘Runtime FPGA partial reconfiguration’, IEEE Aerospace and Electronic Systems Magazine, 23(7), pp. 10–15. doi: 10.1109/MAES.2008.4579286.

Mikhaylov, K. and Tervonen, J. (2012) ‘Evaluation of Power Efficiency for Digital Serial Interfaces of Microcontrollers’, in 2012 5th International Conference on New Technologies, Mobility and Security (NTMS). IEEE, pp. 1–5. doi: 10.1109/NTMS.2012.6208716.

National Institute of Standards and Technology (2001) ‘FIPS 197: Advanced Encryption Standard’ [Online]. Available: https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf [Accessed: Nov. 8, 2017]

Nunez-Yanez, J. L. (2015) ‘Adaptive Voltage Scaling with In-Situ Detectors in Commercial FPGAs’, IEEE Transactions on Computers, 64(1), pp. 45–53. doi: 10.1109/TC.2014.2365963.

Nunez-Yanez, J.L., Hosseinabady, M. and Beldachi, A. (2016) ‘Energy Optimization in Commercial FPGAs with Voltage, Frequency and Logic Scaling’, IEEE Transactions on Computers, 65(5), pp. 1484–1493. doi: 10.1109/TC.2015.2435771.

Oklobdzija, V. G. and Krishnamurthy, R. K. (2006) High-Performance Energy-Efficient Microprocessor Design. Edited by V. G. Oklobdzija and R. K. Krishnamurthy. Boston, MA: Springer US (Series on Integrated Circuits and Systems). doi: 10.1007/978-0-387-34047-0.

OpenCores (2010) ‘WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores (Revision B4)’ [Online]. Available: http://cdn.opencores.org/downloads/wbspec_b4.pdf [Accessed: Nov. 8, 2017]

Pandey, B. et al. (2013) ‘Clock Gated Low Power Memory Implementation on Virtex-6 FPGA’, in 2013 5th International Conference on Computational Intelligence and Communication Networks. IEEE, pp. 409–412. doi: 10.1109/CICN.2013.90.

Pande, V., Elmannai, W. and Elleithy, K. (2013) ‘Classification and detection of fire on WSN using IMB400 multimedia sensor board’, 9th Annual Conference on Long Island Systems, Applications and Technology, LISAT 2013. doi: 10.1109/LISAT.2013.6578247.

Patterson, D. A. and Hennessy, J. L. (2013) Computer Organization and Design : The Hardware / Software Interface.

Pezzarossa, L., Schoeberl, M. and Sparsø, J. (2017) ‘A Controller for Dynamic Partial Reconfiguration in FPGA-Based Real-Time Systems’, in 2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC). IEEE, pp. 92–100. doi: 10.1109/ISORC.2017.3

Qingping Chi et al. (2014) ‘A Reconfigurable Smart Sensor Interface for Industrial WSN in IoT Environment’, IEEE Transactions on Industrial Informatics, 10(2), pp. 1417–1425. doi: 10.1109/TII.2014.2306798.

Rodriguez-Andina, J. J., Moure, M. J. and Valdes, M. D. (2007) ‘Features, Design Tools, and Application Domains of FPGAs’, IEEE Transactions on Industrial Electronics, 54(4), pp. 1810–1823. doi: 10.1109/TIE.2007.898279.

Rodriguez-Andina, J. J., Valdes-Pena, M. D. and Moure, M. J. (2015) ‘Advanced Features and Industrial Applications of FPGAs - A Review’, IEEE Transactions on Industrial Informatics, 11(4), pp. 853–864. doi: 10.1109/TII.2015.2431223.

Shannon, L. et al. (2015) ‘Technology Scaling in FPGAs: Trends in Applications and Architectures’, in 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines. IEEE, pp. 1–8. doi: 10.1109/FCCM.2015.11.

Stankovic, J. A. (2008) ‘Wireless Sensor Networks’, *Computer*, 41(10), pp. 92–95. doi: 10.1109/MC.2008.441.

S. Y. Liew, C. K. Tan, M. L. Gan, H. G. Goh. (2018) ‘A Fast, Adaptive, and Energy-Efficient Data Collection Protocol in Multi-Channel-Multi-Path Wireless Sensor Networks’, in IEEE Computational Intelligence Magazine, pp. 30-40.

Tajalli, A. and Leblebici, Y. (2011) ‘Design trade-offs in ultra-low-power digital nanoscale CMOS’, IEEE Transactions on Circuits and Systems I: Regular Papers, 58(9), pp. 2189–2200. doi: 10.1109/TCSI.2011.2112595.

Texas Instruments (2006) ‘MSP430x1xx Family User's Guide (Rev. F)’ [Online]. Available: http://www.ti.com/lit/ug/slau049f/slau049f.pdf [Accessed: Nov. 4, 2017]

Tong, J. G., Anderson, I. D. L. and Khalid, M. A. S. (2006) ‘Soft-Core Processors for Embedded Systems’, in 2006 International Conference on Microelectronics. IEEE, pp. 170–173. doi: 10.1109/ICM.2006.373294.

Vana Jeličić et al. (2011) ‘MasliNET : A Wireless Sensor Network based Environmental Monitoring System’, in 2011 Proceedings of the 34th International Convention, pp. 150–155. Available at: http://ieeexplore.ieee.org/document/5967041/.

Violante, M. et al. (2011) ‘A Low-Cost Solution for Deploying Processor Cores in Harsh Environments’, IEEE Transactions on Industrial Electronics, 58(7), pp. 2617–2626. doi: 10.1109/TIE.2011.2134054.

Xilinx (2016a) ‘Vivado Design Suite User Guide Partial Reconfiguration’ [Online].Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_1/ug909-vivado-partial-reconfiguration.pdf [Accessed: Jul. 7, 2016]

Xilinx (2016b) ‘7 Series FPGAs Configuration User Guide’ [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf [Accessed: Jul. 7, 2016]

Xufeng Wei, Yahui Wang and Yanliang Dong (2014) ‘Design of fire detection system in buildings based on wireless multimedia sensor networks’, in Proceeding of the 11th World Congress on Intelligent Control and Automation. IEEE, pp. 3008–3012. doi: 10.1109/WCICA.2014.7053208.

Yan Zhang, Roivainen, J. and Mammela, A. (2006) ‘Clock-Gating in FPGAs: A Novel and Comparative Evaluation’, in 9th EUROMICRO Conference on Digital System Design (DSD’06). IEEE, p. `584-590. doi: 10.1109/DSD.2006.32.

Yongjun Xu et al. (2005) ‘Processor Design Considerations for Wireless Sensor Network’, in 2005 6th International Conference on ASIC. IEEE, pp. 255–257. doi: 10.1109/ICASIC.2005.1611298.

APPENDIX A

Table A.1: PR Unit I/O description

Pin name: uipr_tma Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate tma instruction is executing, PR will take place to reconfigure the PR unit 0: indicate tma instruction is not execute
Pin name: uipr_alb_src Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: Register File data or forwarding data is selected 0: Immediate data is selected
Pin name: uipr_rd_src Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: $rd as destination register 0: $rt as destination register
Pin name: uipr_mult_en Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate mult or multu instruction is executing 0: indicate mult and multu instructions are not execute
Pin name: uipr_sign_mult Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Indicate sign multiplication, MSB of the operands as the sign bit. 1: indicate mult instruction is executing 0: indicate mult instruction is not execute
Pin name: uipr_rf_wr Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: enable write to Register File 0: disable write to Register File
Pin name: uipr_sw Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Store word (32-bit data) 1: indicate sw, swl or swr instruction is executing 0: indicate sw, swl and swr instructions are not execute
Pin name: uipr_swl Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Unaligned store word left (32-bit data) 1: indicate swl instruction is executing 0: indicate swl instruction is not execute
Pin name: uipr_swr Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Unaligned store word right (32-bit data) 1: indicate swr instruction is executing 0: indicate swr instruction is not execute
Pin name: uipr_lw Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Load word (32-bit data) 1: indicate lw, lwl or lwr instruction is executing 0: indicate lw, lwl and lwr instructions are not execute

Continued from Table A.1

Pin name: uipr_lwl Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Unaligned load word left (32-bit data) 1: indicate lwl instruction is executing 0: indicate lwl instruction is not execute
Pin name: uipr_lwr Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Unaligned load word right (32-bit data) 1: indicate lwr instruction is executing 0: indicate lwr instruction is not execute
Pin name: uipr_sh Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Store half-word (16-bit data) 1: indicate sh instruction is executing 0: indicate sh instruction is not execute
Pin name: uipr_lh Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Load half-word (16-bit data), sign extend required (refer uipr_load_sign_ext) 1: indicate lh instruction is executing 0: indicate lh instruction is not execute
Pin name: uipr_lhu Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Load half-word unsigned (16-bit data) 1: indicate lhu instruction is executing 0: indicate lhu instruction is not execute
Pin name: uipr_sb Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Store byte (8-bit data) 1: indicate sb instruction is executing 0: indicate sb instruction is not execute
Pin name: uipr_lb Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Load byte Load byte (8-bit data), sign extend required (refer uipr_load_sign_ext) 1: indicate lb instruction is executing 0: indicate lb instruction is not execute
Pin name: uipr_lbu Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: Load byte unsigned (8-bit data) 1: indicate lbu instruction is executing 0: indicate lbu instruction is not execute
Pin name: uipr_load_sign_ext Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate lh or lb instruction is executing, sign extend 16-bit for lh or 24-bit for lb 0: indicate lbu instruction is not execute
Pin name: uipr_sign_ext Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: Immediate data sign extend 0: Immediate data zero extend
Pin name: uipr_mem_to_rf Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: Data memory data to Register File 0: ALU block result to Register File

Continued from Table A.1

Pin name: uipr_hi_wr Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: reserved for future development
Pin name: uipr_lo_wr Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: reserved for future development
Pin name: uipr_alb_to_rf Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: reserved for future development
Pin name: uipr_hi_to_rf Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate HI register data to Register File 0: indicate LO register data to Register File
Pin name: uipr_hilo_acc Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate mflo or mfhi instruction is executing 0: indicate mflo and mfhi instructions are not execute
Pin name: uipr_jump Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate j instruction is executing 0: j instruction is not execute
Pin name: uipr_jr Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate jr instruction is executing 0: indicate jr instruction is not execute
Pin name: uipr_jal Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate jal instruction is executing 0: indicate jal instruction is not execute
Pin name: uipr_jalr Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate jalr instruction is executing 0: indicate jalr instruction is not execute
Pin name: uipr_beq Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate beq instruction is executing 0: indicate beq instruction is not execute
Pin name: uipr_bne Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate bne instruction is executing 0: indicate bne instruction is not execute
Pin name: uipr_blez Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate blez instruction is executing 0: indicate blez instruction is not execute

Continued from Table A.1

Pin name: uipr_bgtz Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate bgtz instruction is executing 0: indicate bgtz instruction is not execute
Pin name: uipr_mfc0 Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate mfc0 instruction is executing 0: indicate mfc0 instruction is not execute
Pin name: uipr_mtc0 Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate mtc0 instruction is executing 0: indicate mtc0 instruction is not execute
Pin name: uipr_eret Pin direction: input Source -> Destination: Main Control Block -> PR Unit / CP0 Block Pin function: 1: indicate eret instruction is executing 0: indicate eret instruction is not execute
Pin name: uipr_syscall Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate syscall instruction is executing 0: indicate syscall instruction is not execute
Pin name: uipr_undef_inst Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate an undefined instruction is detected 0: indicate supported instruction is detected
Pin name: uipr_rtype Pin direction: input Source -> Destination: Main Control Block -> PR Unit Pin function: 1: indicate R-type instruction is issued 0: indicate I-type or J-type instruction is issued
Pin name: uopr_opcode[5:0] Pin direction: output Source -> Destination: PR Unit -> Main Control Block Pin function: Instruction opcode field
Pin name: uopr_funct[5:0] Pin direction: output Source -> Destination: PR Unit -> Main Control Block / Arithmetic Logic Control Block Pin function: Instruction funct field
Pin name: uipr_alb_ctrl[5:0] Pin direction: input Source -> Destination: Arithmetic Logic Control Block -> PR Unit Pin function: ALU operation to perform
Pin name: uipr_rf_rs32 [31:0] Pin direction: input Source -> Destination: Register File Block -> PR Unit Pin function: 32-bit $rs data from Register File
Pin name: uipr_rf_rt32 [31:0] Pin direction: input Source -> Destination: Register File Block -> PR Unit Pin function: 32-bit $rt data from Register File
Pin name: uopr_rf_rs5[4:0] Pin direction: output Source -> Destination: PR Unit -> Register File Block Pin function: Instruction rs field
Pin name: uopr_rf_rt5[4:0] Pin direction: output Source -> Destination: PR Unit -> Register File Block Pin function: Instruction rt field

Continued from Table A.1

Pin name: uopr_rf_wr_data[31:0] Pin direction: output Source -> Destination: PR Unit -> Register File Block Pin function: Data to be written into Register File
Pin name: uopr_rf_wr_addr[4:0] Pin direction: output Source -> Destination: PR Unit -> Register File Block Pin function: Register to be updated in Register File
Pin name: uopr_rf_wr_en Pin direction: output Source -> Destination: PR Unit -> Register File Block Pin function: 1: enable write to Register File 0: disable write to Register File
Pin name: uipr_id_fw_rs32_ctrl[2:0] Pin direction: input Source -> Destination: Forwarding Block -> PR Unit Pin function: Only use in pipeline microarchitecture, used as the forward control signal for the rs path
Pin name: uipr_id_fw_rt32_ctrl[2:0] Pin direction: input Source -> Destination: Forwarding Block -> PR Unit Pin function: Only use in pipeline microarchitecture, used as the forward control signal for the rt path
Pin name: uipr_ex_fw_hilo_ctrl[2:0] Pin direction: input Source -> Destination: Forwarding Block -> PR Unit Pin function: Only use in pipeline microarchitecture, used as the forward control signal for the HILO path
Pin name: uipr_ex_fw_mem Pin direction: input Source -> Destination: Forwarding Block -> PR Unit Pin function: Only use in pipeline microarchitecture 1: Forward data from MEM stage 0: No data forwarding is require
Pin name: uopr_id_rtype Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture 1: indicate a R-type instruction is in ID stage 0: indicate a J-type or I-type instruction is in ID stage
Pin name: uopr_id_itype Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture 1: indicate an I-type instruction is in ID stage 0: indicate a R-type or J-type instruction is in ID stage
Pin name: uopr_id_mfc0 Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, indicate a mfc0 instruction is in ID stage
Pin name: uopr_ex_jal Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, indicate a jal instruction is in EX stage
Pin name: uopr_ex_jalr Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, indicate a jalr instruction is in EX stage
Pin name: uopr_ex_rf_wr Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, indicate enable write to Register File operation is in EX stage
Pin name: uopr_ex_hilo_acc Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, indicate a mflo or mfhi instruction is in EX stage

Continued from Table A.1

Pin name: uopr_ex_hi_to_rf Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, indicate a mfhi instruction is in EX stage
Pin name: uopr_mem_jal Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, indicate a jal instruction is in MEM stage
Pin name: uopr_mem_jalr Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, indicate a jalr instruction is in MEM stage
Pin name: uopr_mem_rf_wr Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, indicate enable write to Register File operation in MEM stage
Pin name: uopr_mem_load Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, indicate a lw, lwl, lwr, lh, lhu, lb or lbu instruction is in MEM stage
Pin name: uopr_mem_mult_en Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, indicate a mult or multu instruction is in MEM stage
Pin name: uopr_ex_rt5_rd5[4:0] Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function:Only use in pipeline microarchitecture, destination register address is in EX stage
Pin name: uopr_mem_rt5_rd5[4:0] Pin direction: output Source -> Destination: PR Unit -> Forwarding Block Pin function: Only use in pipeline microarchitecture, destination register address is in MEM stage
Pin name: uipr_itl_pc_en Pin direction: input Source -> Destination: Interlock Block -> PR Unit Pin function: Only use in pipeline microarchitecture 1: no stall on PC register 0: stall PC register
Pin name: uipr_itl_ifid_en Pin direction: input Source -> Destination: Interlock Block -> PR Unit Pin function: Only use in pipeline microarchitecture 1: no stall on IF/ID pipeline register 0: stall IF/ID pipeline register
Pin name: uipr_itl_idex_en Pin direction: input Source -> Destination: Interlock Block -> PR Unit Pin function: Reserved for future development, temporary always enable
Pin name: uipr_itl_exmem_en Pin direction: input Source -> Destination: Interlock Block -> PR Unit Pin function: Reserved for future development, temporary always enable
Pin name: uipr_itl_memwb_en Pin direction: input Source -> Destination: Interlock Block -> PR Unit Pin function: Reserved for future development, temporary always enable
Pin name: uipr_itl_id_flush_ex Pin direction: input Source -> Destination: Interlock Block -> PR Unit Pin function: Only use in pipeline microarchitecture 1: flush ID/EX stage pipeline register 0: no pipeline register flush is requires
Pin name: uopr_id_load Pin direction: output Source -> Destination: PR Unit -> Interlock Block Pin function: Only use in pipeline microarchitecture, indicate a lw, lwl, lwr, lh, lhu, lb or lbu instruction is in ID stage

Continued from Table A.1

Pin name: uopr_id_store Pin direction: output Source -> Destination: PR Unit -> Interlock Block Pin function: Only use in pipeline microarchitecture, indicate a sw, swl, swr, sh or sb instruction is in ID stage
Pin name: uopr_itl_ex_load Pin direction: output Source -> Destination: PR Unit -> Interlock Block Pin function: Only use in pipeline microarchitecture, indicate a lw, lwl, lwr, lh, lhu, lb or lbu instruction is in EX stage
Pin name: uopr_ex_rt5[4:0] Pin direction: output Source -> Destination: PR Unit -> Interlock Block Pin function: Only use in pipeline microarchitecture, the $rt address in EX stage
Pin name: uipr_cp0_flush_id Pin direction: input Source -> Destination: CP0 Block -> PR Unit Pin function: 1: Flush IF/ID pipeline registers 0: no pipeline register flush is requires
Pin name: uipr_cp0_flush_ex Pin direction: input Source -> Destination: CP0 Block -> PR Unit Pin function: 1: Flush ID/EX pipeline registers 0: no pipeline register flush is requires
Pin name: uipr_cp0_flush_mem Pin direction: input Source -> Destination: CP0 Block -> PR Unit Pin function: 1: Flush EX/MEM pipeline registers 0: no pipeline register flush is requires
Pin name: uipr_cp0_eret_addr[31:0] Pin direction: input Source -> Destination: CP0 Block -> PR Unit Pin function: Exception return address
Pin name: uipr_cp0_read_data[31:0] Pin direction: input Source -> Destination: CP0 Block -> PR Unit Pin function: CP0 register output data
Pin name: uipr_cp0_exc_flag Pin direction: input Source -> Destination: CP0 Block -> PR Unit Pin function: 1: an exception has occur 0: no exception occur
Pin name: uopr_id_rd5[4:0] Pin direction: output Source -> Destination: PR Unit -> CP0 Block Pin function: $rd address in ID stage, the address to read or write in the CP0 register
Pin name: uopr_id_fw_rt32[31:0] Pin direction: output Source -> Destination: Partial Reconfiguration Unit -> CP0 Block Pin function: Data to be updated in CP0 register
Pin name: uopr_cp0RegWr Pin direction: output Source -> Destination: PR Unit -> CP0 Block Pin function: 1: enable write to CP0 register 0: disable write to CP0 register
Pin name: uopr_if_pc[31:0] Pin direction: output Source -> Destination: PR Unit -> CP0 Block Pin function: pipeline microarchitecture: PC register address in IF stage multi-cycle microarchitecture: Previous PC value when uipr_IRQ=1; current PC value when uipr_IRQ=0

Continued from Table A.1

Pin name: uopr_id_pc[31:0] Pin direction: output Source -> Destination: PR Unit -> CP0 Block Pin function: pipeline microarchitecture: PC register address in ID stage multi-cycle microarchitecture: Previous PC value when uipr_IRQ=1; current PC value when uipr_IRQ=0
Pin name: uopr_ex_pc[31:0] Pin direction: output Source -> Destination: PR Unit -> CP0 Block pipeline microarchitecture: PC register address in EX stage multi-cycle microarchitecture: Previous PC value when uipr_IRQ=1; current PC value when uipr_IRQ=0
Pin name: uopr_id_undef_inst Pin direction: output Source -> Destination: PR Unit -> CP0 Block Pin function: 1: indicate an undefined instruction is detected 0: indicate supported instruction is detected
Pin name: uopr_id_syscall Pin direction: output Source -> Destination: PR Unit -> CP0 Block Pin function: 1: indicate syscall instruction is executing 0: indicate syscall instruction is not execute
Pin name: uopr_cp0_all_branch Pin direction: output Source -> Destination: PR Unit -> CP0 Block Pin function: 1: indicate eret, beq, bne, blez, bgtz, j, jr, jal or jalr instruction is executing 0: indicate eret, beq, bne, blez, bgtz, j, jr, jal and jalr instruction is not execute
Pin name: uipr_ex_alb_out[31:0] Pin direction: input Source -> Destination: ALU Block -> PR Unit Pin function: ALU output result
Pin name: uopr_ex_rs32[31:0] Pin direction: output Source -> Destination: PR Unit -> ALU Block Pin function: ALU operand
Pin name: uopr_ex_op_b[31:0] Pin direction: output Source -> Destination: PR Unit -> ALU Block Pin function: ALU operand
Pin name: uopr_ex_alb_ctrl[5:0] Pin direction: output Source -> Destination: PR Unit -> ALU Block Pin function: ALU operation to perform
Pin name: uopr_ex_shamt[4:0] Pin direction: output Source -> Destination: PR Unit -> ALU Block Pin function: Instruction shamt field in EX stage
Pin name: uipr_mem_mult_result[63:0] Pin direction: input Source -> Destination: Multiplier Block -> PR Unit Pin function: Multiplier output result
Pin name: uipr_mem_mult_valid Pin direction: input Source -> Destination: Multiplier Block -> PR Unit Pin function: 1: indicate multiplier result is valid 0: indicate multiplier result is not ready to use
Pin name: uipr_mem_mult_busy Pin direction: input Source -> Destination: Multiplier Block -> PR Unit Pin function: reserved for future development
Pin name: uopr_mult_mulcn[31:0] Pin direction: output Source -> Destination: PR Unit -> Multiplier Block Pin function: Multiplier operand

Continued from Table A.1

Pin name: uopr_mult_mulpl[31:0] Pin direction: output Source -> Destination: PR Unit -> Multiplier Block Pin function: Multiplier operand
Pin name: uopr_ex_sign_mult Pin direction: output Source -> Destination: PR Unit -> Multiplier Block Pin function: 1: indicate mult instruction in EX stage 0: indicate no mult instruction in EX stage
Pin name: uopr_ex_mult_en Pin direction: output Source -> Destination: PR Unit -> Multiplier Block Pin function: 1: indicate mult or multu instruction in EX stage 0: indicate no mult and multu instructions in EX stage
Pin name: uopr_mem_lw Pin direction: output Source -> Destination: PR Unit -> Address Decoder Block Pin function: 1: indicate lw, lwl or lwr instruction in MEM stage 0: indicate no lw, lwl and lwr instructions in MEM stage
Pin name: uopr_mem_lh Pin direction: output Source -> Destination: PR Unit -> Address Decoder Block Pin function: 1: indicate lh or lhu instruction in MEM stage 0: indicate no lh and lhu instructions in MEM stage
Pin name: uopr_mem_lb Pin direction: output Source -> Destination: PR Unit -> Address Decoder Block Pin function: 1: indicate lb or lbu instruction in MEM stage 0: indicate no lb and lbu instructions in MEM stage
Pin name: uopr_mem_sw Pin direction: output Source -> Destination: PR Unit -> Address Decoder Block Pin function: 1: indicate sw, swl or swr instruction in MEM stage 0: indicate no sw, swl and swr instructions in MEM stage
Pin name: uopr_mem_swl Pin direction: output Source -> Destination: PR Unit -> Address Decoder Block Pin function: 1: indicate swl instruction in MEM stage 0: indicate no swl instruction in MEM stage
Pin name: uopr_mem_swr Pin direction: output Source -> Destination: PR Unit -> Address Decoder Block Pin function: 1: indicate swr instruction in MEM stage 0: indicate no swr instruction in MEM stage
Pin name: uopr_mem_sh Pin direction: output Source -> Destination: PR Unit -> Address Decoder Block Pin function: 1: indicate sh instruction in MEM stage 0: indicate no sh instruction in MEM stage
Pin name: uopr_mem_sb Pin direction: output Source -> Destination: PR Unit -> Address Decoder Block Pin function: 1: indicate sb instruction in MEM stage 0: indicate no sb instruction in MEM stage
Pin name: uopr_mem_alb_out[31:0] Pin direction: output Source -> Destination: PR Unit -> Address Decoder Block Pin function: ALU output result in MEM stage

Continued from Table A.1

Pin name: uipr_instr[31:0] Pin direction: input Source -> Destination: ROM / ICACHE Block -> PR Unit Pin function: Instruction machine code to be decoded
Pin name: uipr_loaded_data[31:0] Pin direction: input Source -> Destination: DCACHE Block / I/O controller -> PR Unit Pin function: Data from memory device or I/O registers
Pin name: uipr_mem_stall Pin direction: input Source -> Destination: ICACHE / DCACHE Block -> PR Unit Pin function: 1: halt operation, processor stalling for cache miss 0: no processor stalling is requires
Pin name: uopr_next_pc[31:0] Pin direction: output Source -> Destination: PR Unit -> ROM / ICACHE Pin function: This bus carries the address of the next instruction to be fetched from the ROM / ICACHE
Pin name: uopr_pseudo_pc[31:0] Pin direction: output Source -> Destination: PR Unit -> ROM / ICACHE / Partial Reconfiguration Controller Pin function: The PC value in IF stage. It is used by I-CACHE to generate the cache hit and cache miss signals.
Pin name: uopr_store_addr[31:0] Pin direction: output Source -> Destination: PR Unit -> DCACHE Block / RAM / I/O Controller Pin function: address of data memory or I/O registers to be access
Pin name: uopr_store_data[31:0] Pin direction: output Source -> Destination: PR Unit -> DCACHE Block / RAM / I/O Controller Pin function: Data to be store in data memory or I/O registers
Pin name: uipr_reconf_stall_if Pin direction: input Source -> Destination: PR Controller -> Partial Reconfiguration Unit Pin function: 1: stall IF/ID pipeline register 0: no processor stalling is requires
Pin name: uipr_reconf_release_pc Pin direction: input Source -> Destination: PR Controller -> Partial Reconfiguration Unit Pin function: 1: indicate to copy the PC register address stored in GPR unit to current PC register 0: No copy require
Pin name: uipr_reconf_store_pc[31:0] Pin direction: input Source -> Destination: PR Controller -> Partial Reconfiguration Unit Pin function: PC register address stored in GPR unit
Pin name: uipr_IRQ Pin direction: input Source -> Destination: Priority Interrupt Controller -> PR Unit / CP0 Block Pin function: 1: Interrupt request from I/O controller 0: No interrupt request
Pin name: uipr_clk Pin direction: input Source -> Destination: Global clock -> PR Unit Pin function: Global clock
Pin name: uipr_sys_rst Pin direction: input Source -> Destination: Global reset -> PR Unit Pin function: 1: reset 0: no reset require

Table A.2: Cache unit I/O description

Pin name: uocac_cpu_data[31:0] Pin direction: output Source -> Destination: Cache Unit -> PR Unit -> Data-path Unit Pin function: 32-bits data to CPU (instruction for I-CACHE, data for D-CACHE)
Pin name: uocac_mem_addr[31:0] Pin direction: output Source -> Destination: Cache Unit -> Memory Arbiter Unit -> Data-path Unit Pin function: 32-bits address that indicates which location in the flash memory to be accessed
Pin name: uocac_cpu_stall Pin direction: output Source -> Destination: Cache Unit -> PR Unit -> Data-path Unit Pin function: Indicate a halt operation, processor stalling for cache miss
Pin name: uocac_miss Pin direction: output Source -> Destination: Cache Unit -> Memory Arbiter Unit Pin function: indicates cache miss.
Pin name: uocac_mem_read Pin direction: output Source -> Destination: Cache Unit -> Memory Arbiter Unit Pin function: 1: request read data from the flash memory 0: read disable from the flash memory
Pin name: uocac_mem_sel[3:0] Pin direction: output Source -> Destination: Cache Unit -> Memory Arbiter Unit Pin function: 4-bit byte select control, to select any one or more bytes to be accessed
Pin name: uicac_cpu_addr[31:0] Pin direction: input Source -> Destination: PR Unit -> Cache Unit Pin function: 32-bit address from CPU
Pin name: uicac_reg_cpu_addr[31:0] Pin direction: input Source -> Destination: PR Unit -> Cache Unit Pin function: The registered 32-bit address from CPU. This address bus is used to generate the cache hit and cache miss signals
Pin name: uicac_cpu_read[2:0] Pin direction: input Source -> Destination: PR Unit -> Cache Unit Pin function: 1xx: read word 01x: read half-word 001: read byte
Pin name: uicac_mem_data_rd[31:0] Pin direction: input Source -> Destination: Memory Arbiter Unit -> Cache Unit Pin function: 32-bits data from the flash memory that will transfer to the Cache Unit
Pin name: uicac_mem_ack Pin direction: input Source -> Destination: Memory Arbiter Unit -> Cache Unit Pin function: Acknowledge signal (HIGH) to indicate read data is ready to be transfer from flash memory to Cache Unit
Pin name: uicac_io_intr Pin direction: input Source -> Destination: CP0 block -> Cache Unit Pin function: 1: I-CACHE stop operation, CPU jump to exception handler 0: run normal
Pin name: uicac_mem_busy Pin direction: input Source -> Destination: Flash Controller unit -> Cache Unit Pin function: 1: flash memory is busy 0: flash memory is ready to use
Pin name: uicac_rst Pin direction: input Source -> Destination: Global reset -> Cache Unit Pin function: 1: reset 0: no reset require
Pin name: uicac_clk Pin direction: input Source -> Destination: Global clock -> Cache Unit Pin function: Global clock

Table A.3: Memory Arbiter Unit I/O description, where x = 0, 1, 2 and 3

Pin name: uoma_cac_ack***x*** Pin direction: output Source -> Destination: Memory Arbiter Unit-> Cache Unit Pin function: Acknowledge signal (HIGH) to indicate read data is ready to be transfer from flash memory to Cache Unit
Pin name: uoma_cac_data_rd***x***[31:0] Pin direction: output Source -> Destination: Memory Arbiter Unit-> Cache Unit Pin function: 32-bits data that from flash memory
Pin name: uima_cac_read***x*** Pin direction: input Source -> Destination: Cache Unit-> Memory Arbiter Unit Pin function: 1: request read data from the flash memory 0: read disable from the flash memory
Pin name: uima_cac_miss***x*** Pin direction: input Source -> Destination: Cache Unit-> Memory Arbiter Unit Pin function: indicates cache miss.
Pin name: uima_cac_sel***x***[3:0] Pin direction: input Source -> Destination: Cache Unit-> Memory Arbiter Unit Pin function: 4-bit byte select control, to select any one or more bytes to be accessed
Pin name: uima_cac_addr***x***[31:0] Pin direction: input Source -> Destination: Cache Unit-> Memory Arbiter Unit Pin function: 32-bits address location to be access in the flash memory (only lower 24-bit is used, higher 8-bit is allocated for future expansion)
Pin name: uoma_fc_data[31:0] Pin direction: output Source -> Destination: Memory Arbiter Unit -> Flash Controller Unit Pin function: 32-bits data to be write to flash memory (RESERVED, all caches are read-only)
Pin name: uoma_fc_addr[31:0] Pin direction: output Source -> Destination: Memory Arbiter Unit -> Flash Controller Unit Pin function: 32-bits address location to be access in the flash memory (only lower 24-bit is used, higher 8-bit is allocated for future expansion)
Pin name: uoma_fc_sel[3:0] Pin direction: output Source -> Destination: Memory Arbiter Unit -> Flash Controller Unit Pin function: 4-bit byte select control, to select any one or more bytes to be accessed
Pin name: uoma_fc_read Pin direction: output Source -> Destination: Memory Arbiter Unit -> Flash Controller Unit Pin function: Request read operation from the flash memory
Pin name: uima_fc_ack Pin direction: input Source -> Destination: Flash Controller Unit -> Memory Arbiter Unit Pin function: Acknowledge signal (HIGH) to indicate read data is ready to be transfer from flash memory to Cache Unit
Pin name: uima_fc_data[31:0] Pin direction: input Source -> Destination: Flash Controller Unit -> Memory Arbiter Unit Pin function: 32-bits data from flash memory
Pin name: uima_io_intr Pin direction: input Source -> Destination: CP0 Block -> Memory Arbiter Unit Pin function: 1: An exception has occurred. CPU will jump to exception handler at the next clock cycle 0: run normal
Pin name: uima_rst Pin direction: input Source -> Destination: Global reset -> Memory Arbiter Unit Pin function: 1: reset 0: no reset require
Pin name: uima_clk Pin direction: input Source -> Destination: Global clock -> Memory Arbiter Unit Pin function: Global clock

Table A.4: Flash Controller Unit I/O description

Pin name: SS Pin direction: output Source -> Destination: Flash Controller Unit -> flash memory Pin function: SPI protocol Slave Select
Pin name: SCLK Pin direction: output Source -> Destination: Flash Controller Unit -> flash memory Pin function: SPI protocol clock signal
Pin name: MIO0 Pin direction: bi-directional Source -> Destination: flash memory -> Flash Controller Unit or Flash Controller Unit -> flash memory Pin function: : SPI protocol serial input output pin
Pin name: MI1 Pin direction: input Source -> Destination: flash memory -> Flash Controller Unit Pin function: SPI protocol serial input pin
Pin name: MI2 Pin direction: input Source -> Destination: flash memory -> Flash Controller Unit Pin function: SPI protocol serial input pin
Pin name: MI3 Pin direction: input Source -> Destination: flash memory -> Flash Controller Unit Pin function: SPI protocol serial input pin
Pin name: uofc_busy Pin direction: output Source -> Destination: Flash Controller Unit -> Cache Unit Pin function: Indicate flash memory is busy, data fetching from the flash memory in operation
Pin name: uofc_dout [31:0] Pin direction: output Source -> Destination: Flash Controller Unit -> Memory Arbiter Unit / PR Controller Unit Pin function: 32-bits data from flash memory
Pin name: uofc_ack Pin direction: output Source -> Destination: Flash Controller Unit -> Memory Arbiter Unit / PR Controller Unit Pin function: Indicate data is ready to be fetched
Pin name: uofc_RXFF Pin direction: output Source -> Destination: Flash Controller Unit -> PR Controller Unit Pin function: 1: 1-word (uifc_reconfig=1) or 8-words (uifc_reconfig=0) of data has been received 0: Data receiving is in progress
Pin name: uifc_read Pin direction: input Source -> Destination: Memory Arbiter Unit -> Flash Controller Unit Pin function: 1: request read data from the flash memory 0: read disable from the flash memory
Pin name: uifc_addr[31:0] Pin direction: input Source -> Destination: Memory Arbiter Unit / PR Controller Unit -> Flash Controller Unit Pin function: flash memory address location to be access
Pin name: uifc_cpol Pin direction: input Pin function: SPI clock polarity, default to 1 (set in HDL) to align with the SPI communication mode supported by the flash memory
Pin name: uifc_cpha Pin direction: input Pin function: SPI clock phase, default to 1 (set in HDL) to align with the SPI communication mode supported by the flash memory

Continued from Table A.4

Pin name: uifc_baud[3:0] Pin direction: input Pin function: the clock speed of the SCLK, default 4'b0000 (set in HDL) 0000: uifc_clk / 2 0001: uifc_clk / 4 0010: uifc_clk / 8 0011: uifc_clk / 16 0100: uifc_clk / 32 0101: uifc_clk / 64 0110: uifc_clk / 128 0111: uifc_clk / 256 1000: uifc_clk / 512 1001: uifc_clk / 1024 1010: uifc_clk / 2048 1011: uifc_clk / 4096 1100: uifc_clk / 8192 1101: uifc_clk / 16384 1110: uifc_clk / 32768 1111: uifc_clk / 65536
Pin name: uifc_reconfig Pin direction: input Source -> Destination: PR Controller Unit -> Flash Controller Unit Pin function: 1: Partial reconfiguration has taken place to read the partial bitstream from the flash memory 0: Normal run
Pin name: uifc_reconfig_nwords[31:0] Pin direction: input Source -> Destination: PR Controller Unit -> Flash Controller Unit Pin function: Partial bitstream size (number of words)
Pin name: uifc_clk Pin direction: input Source -> Destination: Global clock -> Flash Controller Unit Pin function: Global clock
Pin name: uifc_rst Pin direction: input Source -> Destination: Global reset -> Flash Controller Unit Pin function: 1: reset 0: no reset require

Table A.5: Boot ROM Unit I/O description

Pin name: borom_wb_dout[31:0] Pin direction: output Source -> Destination: Boot ROM unit -> PR Unit -> Data-path Unit Pin function: 32-bits data output
Pin name: borom_wb_ack Pin direction: output Source -> Destination: Boot ROM unit -> PR Unit Pin function: Indicate data is ready to be fetched
Pin name: birom_wb_addr[SIZE:0] Pin direction: input Source -> Destination: Data-path Unit -> PR Unit -> Boot ROM unit Pin function: Address location of the data in the Boot ROM unit
Pin name: birom_wb_stb Pin direction: input Source -> Destination: Data-path Unit -> PR Unit -> Boot ROM unit Pin function: Strobe control 1: Boot ROM unit is activated to perform read access for new address location 0: Boot ROM unit is de-activated to perform read access
Pin name: birom_wb_clk Pin direction: input Source -> Destination: Global clock -> Boot ROM unit Pin function: Global clock

Continued from Table A.5

Pin name: birom_wb_rst Pin direction: input Source -> Destination: Global reset -> Boot ROM unit Pin function: 1: reset 0: no reset require

Table A.6: Data and Stack RAM Unit I/O description

Pin name: uoram_wb_dout[31:0] Pin direction: output Source -> Destination: Data and Stack RAM Unit -> PR Unit -> Data-path Unit Pin function: 32-bits data output
Pin name: uoram_wb_ack Pin direction: output Source -> Destination: Data and Stack RAM Unit -> PR Unit Pin function: Indicate data is ready to be fetched
Pin name: uiram_wb_din[31:0] Pin direction: input Source -> Destination: Data-path Unit -> PR Unit -> Data and Stack RAM Unit Pin function: 32-bits data input
Pin name: uiram_wb_addr[SIZE:0] Pin direction: input Source -> Destination: Data-path Unit -> PR Unit -> Data and Stack RAM Unit Pin function: Address location of the data in the Data and Stack RAM Unit
Pin name: uiram_wb_sel[3:0] Pin direction: input Source -> Destination: Address Decoder Block -> Data-path Unit -> Data and Stack RAM Unit Pin function: 4-bit byte select control, to select any one or more bytes to be accessed
Pin name: uiram_wb_we Pin direction: input Source -> Destination: Address Decoder Block -> Data-path Unit -> Data and Stack RAM Unit Pin function: write control 1: Enable to write to the Data and Stack RAM Unit 0: No operation
Pin name: uiram_wb_stb Pin direction: input Source -> Destination: Address Decoder Block -> Data-path Unit -> Data and Stack RAM Unit Pin function: Strobe control 1: Data and Stack RAM Unit is activated to perform read or write access for new address location 0: Data and Stack RAM Unit is de-activated to perform read or write access
Pin name: uiram_wb_clk Pin direction: input Source -> Destination: Global clock -> Data and Stack RAM Unit Pin function: Global clock
Pin name: uiram_wb_rst Pin direction: input Source -> Destination: Global reset -> Data and Stack RAM Unit Pin function: 1: reset 0: no reset require

Table A.7: UART Controller I/O description

Pin name: uoua_TxD Pin direction: output Source -> Destination: device 0's uoua_TxD -> device 1's uiua_RxD Pin function: UART standard pin – transmit serial data

Continued from Table A.7

Pin name: uoua_IRQ Pin direction: output Source -> Destination: UART controller Unit -> Priority Interrupt Controller Unit Pin function: To request an interrupt (uiua_UARTIE must pull high before can send an interrupt) 1: Request to interrupt 0: No interrupt request
Pin name: uoua_wb_dout [7:0] Pin direction: output Source -> Destination: UART controller Unit -> PR Unit -> Data-path Unit Pin function: Wishbone standard data output bus
Pin name: uoua_wb_ack Pin direction: output Source -> Destination: UART controller Unit -> PR Unit -> Data-path unit Pin function: Wishbone standard acknowledge signal - indicates the termination of a normal bus cycle 1: normal bus cycle termination 0: no bus cycle termination
Pin name: uiua_RxD Pin direction: input Source -> Destination: device 0's uiua_RxD <- device 1's uoua_TxD Pin function: UART standard pin – receive serial data
Pin name: uiua_UARTIE Pin direction: input Source -> Destination: Priority Interrupt Controller Unit-> UART controller Unit Pin function: allow UART to interrupt 1: enable UART global interrupt 0: disable UART global interrupt
Pin name: uiua_wb_din [7:0] Pin direction: input Source -> Destination: Data-path unit -> PR Unit -> UART controller Unit Pin function: Wishbone standard data input bus
Pin name: uiua_wb_sel [3:0] Pin direction: input Source -> Destination: Address Decoder Block -> UART controller Unit Pin function: Wishbone standard byte select signal – data granularity control 1111: word selected 1100: upper half-word selected 0011: lower half-word selected 1000: 4th byte selected 0100: 3rd byte selected 0010: 2nd byte selected 0001: 1st byte selected
Pin name: uiua_wb_we Pin direction: input Source -> Destination: Address Decoder Block -> UART controller Unit Pin function: Wishbone standard write enable signal – indicate current bus cycle is for READ or WRITE 1: WRITE cycle – write to UART controller 0: READ cycle – Read from UART controller
Pin name: uiua_wb_stb Pin direction: input Source -> Destination: Address Decoder Block -> UART controller Unit Pin function: Wishbone standard strobe signal – indicate valid data transfer cycle 1: activate UART controller for read or write access 0: deactivate UART controller for read or write access
Pin name: uiua_wb_clk Pin direction: input Source -> Destination: Global clock -> UART controller Unit Pin function: Global clock
Pin name: uiua_wb_rst Pin direction: input Source -> Destination: Global reset -> UART controller Unit Pin function: 1: reset 0: no reset require

Table A.8: SPI Controller I/O description

Pin name: uiospi_MOSI Pin direction: bi-directional Source -> Destination: device 0's uiospi_MOSI <-> device 1's uiospi_MISO Pin function: SPI standard pin – Master out Serial in If the SPI is configure as a master, then uiospi_MOSI will become an output, else otherwise.
Pin name: uiospi_MISO Pin direction: bi-directional Source -> Destination: device 0's uiospi_MISO <-> device 1's uiospi_MOSI Pin function: SPI standard pin – Master in Serial out If the SPI is configure as a master, then uiospi_ MISO will become an input, else otherwise.
Pin name: uiospi_SCLK Pin direction: bi-directional Source -> Destination: device 0's uiospi_SCLK <-> device 1's uiospi_SCLK Pin function: SPI standard pin – SPI clock signal for data synchronization across devices If the SPI is configure as a master, then uiospi_ SCLK will become an output, else otherwise.
Pin name: uiospi_SS_n Pin direction: bi-directional Source -> Destination: device 0's SS_n <-> device 1's SS_n Pin function: SPI standard pin – SPI slave select control signal If the SPI is configure as a master, then uiospi_ SS_n will become an output, else otherwise.
Pin name: uospi_IRQ Pin direction: output Source -> Destination: SPI controller Unit -> Priority Interrupt Controller Unit Pin function: To request an interrupt (uispi_SPIE must pull high before can send an interrupt) 1: Request to interrupt 0: No interrupt request
Pin name: uospi_wb_dout [7:0] Pin direction: output Source -> Destination: SPI controller Unit -> PR Unit -> Data-path Unit Pin function: Wishbone standard data output bus
Pin name: uospi_wb_ack Pin direction: output Source -> Destination: SPI controller Unit -> PR Unit -> Data-path Unit Pin function: Wishbone standard acknowledge signal - indicates the termination of a normal bus cycle 1: normal bus cycle termination 0: no bus cycle termination
Pin name: uispi_SPIE Pin direction: input Source -> Destination: Priority Interrupt Controller Unit-> SPI controller Unit Pin function: allow SPI to interrupt 1: enable SPI global interrupt 0: disable SPI global interrupt
Pin name: uispi_wb_din[7:0] Pin direction: input Source -> Destination: Data-path Unit -> PR Unit -> SPI controller Unit Pin function: Wishbone standard data input bus
Pin name: uispi_wb_sel[3:0] Pin direction: input Source -> Destination: Address Decoder Block -> SPI controller Unit Pin function: Wishbone standard byte select signal – data granularity control 1111: word selected 1100: upper half-word selected 0011: lower half-word selected 1000: 4th byte selected 0100: 3rd byte selected 0010: 2nd byte selected 0001: 1st byte selected
Pin name: uispi_wb_we Pin direction: input Source -> Destination: Address Decoder Block -> SPI controller Unit Pin function: Wishbone standard write enable signal – indicate current bus cycle is for READ or WRITE 1: WRITE cycle – Write to SPI controller 0: READ cycle – Read from SPI controller

Continued from Table A.8

Pin name: uispi_wb_stb Pin direction: input Source -> Destination: Address Decoder Block -> SPI controller Unit Pin function: Wishbone standard strobe signal – indicate valid data transfer cycle 1: activate SPI controller for read or write access 0: deactivate SPI controller for read or write access
Pin name: uispi_wb_clk Pin direction: input Source -> Destination: Global clock -> SPI controller Unit Pin function: Global clock
Pin name: uispi_wb_rst Pin direction: input Source -> Destination: Global reset -> SPI controller Unit Pin function: 1: reset 0: no reset require

Table A.9: GPIO Controller unit I/O description

Pin name: uiogpio_PORT_pin Pin direction: inout Source -> Destination: GPIO Controller Unit <-> External device (LEDs, switches etc.) Pin function: GPIO pins
Pin name: uogpio_wb_dout [31:0] Pin direction: output Source -> Destination: GPIO Controller Unit -> PR Unit -> Data-path Unit Pin function: Wishbone standard data output bus
Pin name: uogpio_wb_ack Pin direction: output Source -> Destination: GPIO Controller Unit -> PR Unit -> Data-path Unit Pin function: Wishbone standard acknowledge signal - indicates the termination of a normal bus cycle 1: normal bus cycle termination 0: no bus cycle termination
Pin name: uigpio_wb_din [31:0] Pin direction: input Source -> Destination: Data-path Unit -> PR Unit -> GPIO Controller Unit Pin function: Wishbone standard data input bus
Pin name: uigpio_wb_addr[1:0] Pin direction: input Source -> Destination: Data-path Unit -> PR Unit -> GPIO Controller Unit Pin function: Used to select which register to be access 00: GPIODIR 01: GPIOEN 10: GPIODATA 11: RESERVED
Pin name: uigpio_wb_sel [3:0] Pin direction: input Source -> Destination: Address Decoder Block -> GPIO Controller Unit Pin function: Wishbone standard write enable signal – data granularity control 1111: word selected 1100: upper half-word selected 0011: lower half-word selected 1000: 4th byte selected 0100: 3rd byte selected 0010: 2nd byte selected 0001: 1st byte selected
Pin name: uigpio_wb_we Pin direction: input Source -> Destination: Address Decoder Block -> GPIO Controller Unit Pin function: Wishbone standard write enable signal – indicate current bus cycle is for READ or WRITE 1: WRITE cycle – write to GPIO controller 0: READ cycle – read from GPIO controller

Continued from Table A.9

Pin name: uigpio_wb_stb Pin direction: input Source -> Destination: Address Decoder Block -> GPIO Controller Unit Pin function: Wishbone standard strobe signal – indicate valid data transfer cycle 1: activate UART controller for read or write access 0: deactivate UART controller for read or write access
Pin name: uigpio_wb_clk Pin direction: input Source -> Destination: Global clock -> GPIO Controller Unit Pin function: Global clock
Pin name: uigpio_wb_rst Pin direction: input Source -> Destination: Global reset -> GPIO Controller Unit Pin function: 1: reset 0: no reset require

Table A.10: Priority Interrupt Controller unit I/O description

Pin name: uopi_ctrl_IO_IE[7:0] Pin direction: output Source -> Destination: Priority Interrupt Controller Unit -> UART / SPI Controller Unit Pin function: Interrupt sources masking bit 1: Enable the interrupt source 0: Disable the interrupt source
Pin name: uopi_ctrl_req_IPL[1:0] Pin direction: output Source -> Destination: Priority Interrupt Controller Unit -> CP0 Block Pin function: indicate the IPL of the interrupt source. This value will be store in the CP0 $cause register to prevent lower IPL interrupt sources to interrupt the CPU
Pin name: uopi_ctrl_IRQ Pin direction: output Source -> Destination: Priority Interrupt Controller Unit -> PR Unit -> Data-path Unit Pin function: Interrupt request signal. Pull high for 1 clock cycle to interrupt the CPU to stop current process and jump to exception handler (0x8001_B140). 1: Interrupt request from one of the interrupt sources 0: No interrupt request
Pin name: uopi_ctrl_wb_dout [31:0] Pin direction: output Source -> Destination: Priority Interrupt Controller Unit -> PR Unit -> Data-path Unit Pin function: Wishbone standard data output bus
Pin name: uopi_ctrl_wb_ack Pin direction: output Source -> Destination: Priority Interrupt Controller Unit -> PR Unit -> Data-path Unit Pin function: Wishbone standard acknowledge signal - indicates the termination of a normal bus cycle 1: normal bus cycle termination 0: no bus cycle termination
Pin name: uipi_ctrl_intr_vector[7:0] Pin direction: input Source -> Destination: CP0 Timer, SPI and UART Controller Unit -> Priority Interrupt Controller Unit Pin function: Connect up to 8 interrupt sources uipi_ctrl_intr_vector[7]: CP0 timer uipi_ctrl_intr_vector[6]: 1’b0 uipi_ctrl_intr_vector[5]: 1’b0 uipi_ctrl_intr_vector[4]: 1’b0 uipi_ctrl_intr_vector[3]: SPI Controller Unit uipi_ctrl_intr_vector[2]: UART Controller Unit uipi_ctrl_intr_vector[1]: 1’b0 uipi_ctrl_intr_vector[0]: 1’b0

Continued from Table A.10

Pin name: uipi_ctrl_stat_IPL Pin direction: input Source -> Destination: CP0 Block -> Priority Interrupt Controller Unit Pin function: Indicate the Interrupt Priority Level that currently handle. This is to prevent the lower IPL interrupt sources to interrupt the CPU.
Pin name: uipi_ctrl_intr_en_n Pin direction: input Source -> Destination: CP0 Block -> Priority Interrupt Controller Unit Pin function: Interrupt enable signal 1: disable interrupt 0: enable interrupt
Pin name: uipi_ctrl_cpu_stall Pin direction: input Source -> Destination: Cache Unit -> Priority Interrupt Controller Unit Pin function: stall the Priority Interrupt Controller when memories (I-CACHE/D-CACHE) stall. This is to ensure that no interrupt request can occur during memories stall
Pin name: uipi_ctrl_wb_din[31:0] Pin direction: input Source -> Destination: Data-path Unit -> PR Unit -> Priority Interrupt Controller Unit Pin function: Wishbone standard data input bus
Pin name: uipi_ctrl_wb_sel[3:0] Pin direction: input Source -> Destination: Address Decoder Block -> Priority Interrupt Controller Unit Pin function: Wishbone standard write enable signal – data granularity control 1111: word selected 1100: upper half-word selected 0011: lower half-word selected 1000: 4th byte selected 0100: 3rd byte selected 0010: 2nd byte selected 0001: 1st byte selected
Pin name: uipi_ctrl_wb_we Pin direction: input Source -> Destination: Address Decoder Block -> Priority Interrupt Controller Unit Pin function: Wishbone standard write enable signal – indicate current bus cycle is for READ or WRITE 1: WRITE cycle – Write to Priority Interrupt Controller 0: READ cycle – Read from Priority Interrupt Controller
Pin name: uipi_ctrl_wb_stb Pin direction: input Source -> Destination: Address Decoder Block -> Priority Interrupt Controller Unit Pin function: Wishbone standard strobe signal – indicate valid data transfer cycle 1: activate Priority Interrupt controller for read or write access 0: deactivate Priority Interrupt controller for read or write access
Pin name: uipi_ctrl_wb_clk Pin direction: input Source -> Destination: Global clock -> Priority Interrupt Controller Unit Pin function: Global clock
Pin name: uipi_ctrl_wb_rst Pin direction: input Source -> Destination: Global reset -> Priority Interrupt Controller Unit Pin function: 1: reset 0: no reset require

Table A.11: General Purpose Register unit I/O description

Pin name: uogpr_reconf_setting[31:0] Pin direction: output Source -> Destination: General Purpose Register Unit -> PR Controller Unit Pin function: Output the SETTING register of the General Purpose Register Unit
Pin name: uogpr_pipeline_reconf_addr[31:0] Pin direction: output Source -> Destination: General Purpose Register Unit -> PR Controller Unit Pin function: Pipeline microarchitecture partial bitstream start address

Continued from Table A.11

Pin name: uogpr_multicycle_reconf_addr[31:0] Pin direction: output Source -> Destination: General Purpose Register Unit -> PR Controller Unit Pin function: Multi-cycle microarchitecture partial bitstream start address
Pin name: uogpr_wb_dout[31:0] Pin direction: output Source -> Destination: General Purpose Register Unit -> PR Unit -> Data-path Unit Pin function: Wishbone standard data output bus
Pin name: uogpr_wb_ack Pin direction: output Source -> Destination: General Purpose Register Unit -> PR Unit -> Data-path Unit Pin function: Wishbone standard acknowledge signal - indicates the termination of a normal bus cycle 1: normal bus cycle termination 0: no bus cycle termination
Pin name: uigpr_update_ma Pin direction: input Source -> Destination: PR Controller Unit -> General Purpose Register Unit Pin function: Indicate to update the current microarchitecture status in SETTING register
Pin name: uigpr_wb_din[31:0] Pin direction: input Source -> Destination: Data-path Unit -> PR Unit -> General Purpose Register Unit Pin function: Wishbone standard data input bus
Pin name: uigpr_wb_addr[1:0] Pin direction: input Source -> Destination: Data-path Unit -> PR Unit -> General Purpose Register Unit Pin function: Used to select which register to be access 00: SETTING 01: P5ADDR 10: M5ADDR 11: RESERVED
Pin name: uigpr_wb_sel[3:0] Pin direction: input Source -> Destination: Address Decoder Block -> General Purpose Register Unit Pin function: Pin function: Wishbone standard write enable signal – data granularity control 1111: word selected 1100: upper half-word selected 0011: lower half-word selected 1000: 4th byte selected 0100: 3rd byte selected 0010: 2nd byte selected 0001: 1st byte selected
Pin name: uigpr_wb_we Pin direction: input Source -> Destination: Address Decoder Block -> General Purpose Register Unit Pin function: Wishbone standard write enable signal – indicate current bus cycle is for READ or WRITE 1: WRITE cycle – write to General Purpose Register Unit 0: READ cycle – read from General Purpose Register Unit
Pin name: uigpr_wb_stb Pin direction: input Source -> Destination: Address Decoder Block -> General Purpose Register Unit Pin function: Wishbone standard strobe signal – indicate valid data transfer cycle 1: activate General Purpose Register for read or write access 0: deactivate General Purpose Register for read or write access
Pin name: uigpr_wb_clk Pin direction: input Source -> Destination: Global clock -> General Purpose Register Unit Pin function: Global clock
Pin name: uigpr_wb_rst Pin direction: input Source -> Destination: Global reset -> General Purpose Register Unit Pin function: 1: reset 0: no reset require