

**DESIGN OF AN ADC CONTROLLER FOR 5-STAGE PIPELINE RISC32
MICROPROCESSOR**

**BY
TAN YAN KAI**

**A REPORT
SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfillment of the requirements
for the degree of
BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMPUTER
ENGINEERING
Faculty of Information and Communication Technology
(Kampar Campus)**

JAN 2022

**DESIGN OF AN ADC CONTROLLER FOR 5-STAGE PIPELINE RISC32
MICROPROCESSOR**

BY
TAN YAN KAI

A REPORT
SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfillment of the requirements
for the degree of
BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMPUTER
ENGINEERING
Faculty of Information and Communication Technology
(Kampar Campus)

JAN 2022

REPORT STATUS DECLARATION FORM

Title: DESIGN OF AN ADC CONTROLLER FOR 5-STAGE
PIPELINE RISC32 MICROPROCESSOR

Academic Session: JAN 2022

I TAN YAN KAI
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.



(Author's signature)

Verified by,



(Supervisor's signature)

Address:

5, LORONG TERKUKUR 4,
TAMAN COWIN, 14300
NIBONG TEBAL, PULAU PINANG

Teoh Shen Khang

Supervisor's name

Date: 15 APRIL 2022

Date: 18 April 2022

Universiti Tunku Abdul Rahman			
Form Title : Sample of Submission Sheet for FYP/Dissertation/Thesis			
Form Number: FM-IAD-004	Rev No.: 0	Effective Date: 21 JUNE 2011	Page No.: 1 of 1

**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN**

Date: 15 April 2022

SUBMISSION OF FINAL YEAR PROJECT

It is hereby certified that Tan Yan Kai (ID No: 18ACB03478) has completed this final year project entitled “ Design of an ADC Controller for 5-stage pipeline RISC32 Microprocessor ” under the supervision of Mr Teoh Shen Khang (Supervisor) from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology.

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

Yours truly,



(TAN YAN KAI)

DECLARATION OF ORIGINALITY

I declare that this report entitled “**DESIGN OF AN ADC CONTROLLER FOR 5-STAGE PIPELINE RISC32 MICROPROCESSOR**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.



Signature : _____

Name : TAN YAN KAI

Date : 15 APRIL 2022

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Mr Mok Kai Ming and Mr Teoh Shen Khang who has given me this bright opportunity to engage in a processor design project. It definitely helps me to get familiar with my future career. I am really thankful to him because I was able to learn a lot of new things and pick up some skills during the project development. Again, thank you for precious time and patience guiding me throughout the project.

Besides I would like to thank all my course mate for their sincerity, kindness, and selfless attitude in this wonderful learning journey. I am grateful that we are able to support each other on internet during the outbreak of Covid-19.

Last but not least, I would also like to express my special gratitude and thanks to my beloved family members. Thanks to my brothers for teaching me the knowledge and skills they had acquired in their university life. Also, thanks to my parents who have been supporting me both mentally and financially over the years. Without their support, I will not have such great opportunity to further my studies in university.

ABSTRACT

This project is about the Analog-to-Digital Converter (ADC) controller unit design and implementation for academic purpose. Throughout the project, XADC from Xilinx is used for simulation purpose. The development of this project will begin with the design of the ADC controller unit. The datasheet of the XADC which is provided by the Xilinx is studied in the beginning of design process in order to match all the requirements and protocols to ensure the functionality of ADC controller unit. The RTL design flow will be used throughout the project development and the micro-architectural level design will be focused more as the ADC controller to be designed is in the unit level. The ADC controller unit and the internal block will be modelled by using Verilog HDL. The XADC with a model name of ug480 is obtained from the IP Catalog of Vivado and it will be instantiated in RISC32. The ug480 module also comes with a constraint and a simple dataset that are needed for simulation purpose. The specifications of the ADC controller unit and the instantiation of ug480 will be functionally verified by writing testbenches in Verilog HDL. Besides, the specific registers that needed for the communication between XADC and ADC controller unit will also be tested for functionality. Some of the functions provided by ug480 are not implemented as they are irrelevant to the learning course.

After the ADC controller unit has been functionally verified, it will be integrated into the existing 5-stage pipeline RISC32 processor which is developed in UTAR. This involves the interfacing between the ADC controller and the RISC32 based on the I/O memory mapping technique. Moreover, a simple Interrupt Service Routine will be specifically developed and implemented on the RISC32 to perform certain instructions whenever there is an interrupt signal sent by XADC.

Lastly, a simple assembly program code will be specifically designed to test the overall functionality. Other than basic function in the ADC controller unit, multiple interrupt and multiple trap case will also be tested since ADC controller unit is one of the IO devices and it will eventually work with other IO devices at the same time after the integration is completed.

Table of Contents

TITLE PAGE	I
REPORT STATUS DECLARATION FORM	II
SUBMISSION OF FINAL YEAR PROJECT	III
DECLARATION OF ORIGINALITY	IV
ACKNOWLEDGEMENTS	V
ABSTRACT	VI
LIST OF FIGURES	XI
LIST OF TABLES	XIII
LIST OF ABBREVIATIONS	XIV
CHAPTER 1: INTRODUCTION	1
1.1 Background Information	1
1.1.1 MIPS	1
1.1.2 ADC	2
1.2 Motivation	2
1.3 Problem Statement	3
1.4 Project Scope	3
1.5 Project Objectives	4
1.6 Impact, Significance, and Contribution	5
CHAPTER 2: LITERATURE REVIEW	6
2.1 Overview of Xilinx XADC	6
2.2 Memory-mapped I/O	10
2.3 ADC controller designed in [2].....	11
CHAPTER 3: PROPOSED METHOD/APPROACH	12
3.1 Methodologies and General Work Procedures	12
3.1.1 RTL Design Flow	12
3.1.2 Micro-architecture Specification	13
3.1.3 RTL Modelling and Verification	13
3.1.4 Logic Synthesis for FPGA	14
3.2 Design Tools	14
3.2.1 ModelSim SE-64 10.5.....	15
3.2.2 Xilinx Vivado Design Suite	15
3.2.3 PCSpim/QtSpim.....	16
3.3 Technologies Involved.....	16
3.3.1 Field Programmable Gate Array (FPGA)	16

3.4 Implementation Issues and Challenges	16
3.5 Timeline	17
3.5.1 Gantt Chart for Project I	17
3.5.2 Gantt Chart for Project II	18
CHAPTER 4: SYSTEM SPECIFICATION	19
4.1 System Overview of the RISC32 Pipeline Processor	19
4.1.1 RISC32 Pipeline Processor Architecture	19
4.1.2 Functional View of the RISC32 Pipeline Processor	21
4.1.3 Memory Map of the RISC32 Pipeline Processor.....	21
4.2 Chip Interface of the RISC32 Pipeline Processor.....	23
4.3 Input Pin Description of the RISC32 Pipeline Processor	23
4.4 Output Pin Description of the RISC32 Pipeline Processor.....	24
4.5 Input Output Pin Description of the RISC32 Pipeline Processor	24
CHAPTER 5: MICRO-ARCHITECTURE SPECIFICATION.....	26
5.1 Functionality/Feature of the ADC Controller Unit.....	26
5.2 Unit Interface of the ADC Controller Unit	26
5.3 Input Pin Description of the ADC Controller unit.....	27
5.4 Output Pin Description of the ADC Controller Unit	29
5.5 Internal Operation of the ADC Controller Unit.....	29
5.6 Example application of XADC with ADC Controller Unit in RISC32.....	33
5.7 Design Partitioning of the ADC Controller Unit	34
5.8 Micro-Architecture of the ADC Controller Unit	35
5.9 ADC Clock Control block	36
5.9.1 Functionality/Feature of ADC Clock Control block.....	36
5.9.2 Block interface of ADC Clock Control block.....	36
5.9.3 Input Pin Description of the ADC Clock Control block.....	36
5.9.4 Output Pin Description of the ADC Clock Control block	37
5.10 Finite State Machine of the ADC Controller Unit.....	38
5.10.1 Flags in FSM.....	38
5.10.2 Internal Operation: UADC to XADC	41
5.10.3 Internal Operation: XADC to UADC	43
5.11 Register Set	45

CHAPTER 6: FIRMWARE DEVELOPMENT	52
6.1 Exception Handler of the RISC32 Pipeline Processor.....	52
6.2 Interrupt Service Routine (ISR) of the ADC Controller Unit.....	53
CHAPTER 7: VERIFICATION SPECIFICATION AND SIMULATION RESULT.....	55
7.1 Unit Level Functional Test Plan	55
7.2 Simulation Result for Unit Level Functional Test.....	59
7.2.1 Test case 1: System Reset.	59
7.2.2 Test case 2: Read Operation.	60
7.2.3 Test case 3: Write Operation.....	60
7.2.4 Test case 4: Interrupt Operation.....	62
7.3 Testbench for Integration Level Functional Test.....	64
7.4 Simulation Result for Integration Level Functional Test	65
CHAPTER 8: MULTIPLE IO SYSTEM FUNCTIONAL TEST	67
8.1 Test Case: Multiple interrupt and Multiple Trap.	67
8.2 Testbench for Multiple Exception Test.....	69
8.3 Simulation result	70
CHAPTER 9: CONCLUSION AND FUTURE WORK	72
9.1 Conclusion	72
9.2 Future Work.....	73
BIBLIOGRAPHY	74
APPENDIX A: XADC DATASHEET	A-1
XADC ports and description.....	A-1
Formula for temperature	A-3
Formula for power-supply sensor	A-3
Formula for auxiliary input.....	A-3
Detail description for XADC Status register	A-3
Detail for XADC Configuration Registers	A-5
The Control Register for Sequence mode in XADC	A-7
XADC Alarms Threshold Registers	A-9

Dynamic Reconfiguration Port (DRP) Timing	A-9
XADC Verilog code from Xilinx	A-10
Testbench code from Xilinx.....	A-16
Original Analog Stimulus File from Xilinx	A-17
APPENDIX B: SIMULATION SOURCE	B-1
Testbench for ADC Controller Unit Level Functional Test	B-1
Testbench for Integration Level Functional Test and Multiple Exception Test with assembly language.	B-5
APPENDIX C: FINAL YEAR PROJECT WEEKLY REPORT.....	C-1
APPENDIX D: POSTER.....	D-1
APPENDIX E: PLAGIARISM CHECK RESULT	E-1
Form iad-FM-IAD-005	E-5
APPENDIX F: FYP 2 CHECKLIST.....	F-1

LIST OF FIGURES

Figure Number	Title	Page
Figure 1-1	MIPS 5 stage pipeline	1
Figure 2-1	XADC Primitive Ports	6
Figure 2-2	XADC Register Interface	8
Figure 2-3	The three Configuration registers in XADC	8
Figure 2-4	Unipolar data format	9
Figure 2-5	Bipolar data format	9
Figure 3-1	RTL design flow	13
Figure 3-2	Comparison among 3 different Verilog simulators	14
Figure 3-3	Gantt Chart for Project 1	17
Figure 3-4	Gantt Chart for Project 1	17
Figure 3-5	Gantt Chart for Project 2	18
Figure 4-1	An overview on the architecture of the RISC32 pipeline processor	20
Figure 4-2	The functional view of the RISC32 pipeline processor	21
Figure 4-3	Memory map of the RISC32 pipeline processor	22
Figure 4-4	Chip interface of the RISC32 pipeline processor	23
Figure 5-1	ADC controller unit interface	26
Figure 5-2	Application of XADC in RISC32	33
Figure 5-3	Partitioning of the ADC controller unit	34
Figure 5-4	ADC Clock Control block interface	36
Figure 5-5	FSM flag register priority	38
Figure 5-6	Schematic diagram for the flag register	40
Figure 5-7	Schematic diagram for FSM writing operation	41
Figure 5-8	Timing diagram for FSM writing operation	42
Figure 5-9	Schematic diagram for FSM reading operation	43
Figure 5-10	Timing diagram for FSM reading operation	43
Figure 6-1	Pseudocode describing exception handler in RISC32	53
Figure 6-2	Pseudocode describing UADC's ISR	54
Figure 7-1	Stimulus text file	55

Figure 7-2	Simulation result for test case 1 using Vivado simulation tool	59
Figure 7-3	Simulation result for test case 2 using Vivado simulation tool	60
Figure 7-4	Simulation result for test case 3(i) using Vivado simulation tool.	60
Figure 7-5	Simulation result for test case 3(ii) using Vivado simulation tool.	61
Figure 7-6	Simulation result for test case 3(iii) using Vivado simulation tool.	61
Figure 7-7	Simulation result for test case 4(i) using Vivado simulation tool.	62
Figure 7-8	Simulation result for test case 4(ii) using Vivado simulation tool.	62
Figure 7-9	Simulation result for test case 4(iii) using Vivado simulation tool.	63

LIST OF TABLES

Table Number	Title	Page
Table 2-1	Three general types of special purpose registers used in MMIO	10
Table 4-1	Specification of the RISC32 pipeline processor	20
Table 4-2	Memory map description of the RISC32 pipeline processor	22
Table 4-3	Input pin description of the RISC32 pipeline processor	23
Table 4-4	Output pin description of the RISC32 pipeline processor	24
Table 4-5	Input output Pin description of the RISC32 pipeline processor	24
Table 5-1	Input pin description of the ADC controller unit	27
Table 5-2	Output pin description of the ADC controller unit	29
Table 5-3	Functional description of the ADC controller unit's read operation	29
Table 5-4	Functional description of the ADC controller unit's write operation	31
Table 5-5	Input Pin description of the ADC Clock Control	36
Table 5-6	Output Pin description of the ADC Clock Control block	37
Table 7-1	Digital value of stimulus file	59

LIST OF ABBREVIATIONS

<i>ADC</i>	Analog-to-Digital Converter
<i>CPO</i>	Coprocessor 0
<i>CPU</i>	Central Processing Unit
<i>DRP</i>	Dynamic Reconfiguration Port
<i>EDA</i>	Electronic Design Automation
<i>EX</i>	Execute
<i>FICT</i>	Faculty of Information and Communication Technology
<i>FPGA</i>	Field Programmable Gate Array
<i>FSM</i>	Finite State Machine
<i>GPIO</i>	General-Purpose Input Output
<i>GPR</i>	General-Purpose Register
<i>HDL</i>	Hardware Description Language
<i>IC</i>	Integrated Circuit
<i>ID</i>	Instruction Decode and Operand Fetch
<i>IF</i>	Instruction Fetch
<i>IO</i>	Input Output
<i>IP</i>	Intellectual Property
<i>ISA</i>	Instruction Set Architecture
<i>ISR</i>	Interrupt Service Routine
<i>JTAG</i>	Joint Test Action Group
<i>MEM</i>	Memory Access
<i>MIPS</i>	Microprocessor without Interlocked Pipeline Stages
<i>MMIO</i>	Memory-mapped Input Output
<i>Msp/s</i>	Mega sample per second
<i>RAM</i>	Random Access Memory
<i>RAW</i>	Read-After-Write
<i>RF</i>	Radio Frequency
<i>RISC</i>	Reduced Instruction Set Computer
<i>ROM</i>	Read Only Memory
<i>RTL</i>	Register Transfer Level
<i>SPI</i>	Serial Peripheral Interface

<i>UADC_CREG</i>	ADC Controller Unit Configuration Register
<i>UADC_SREG</i>	ADC Controller Unit Status Register
<i>UART</i>	Universal Asynchronous Receiver-Transmitter
<i>UTAR</i>	University Tunku Abdul Rahman
<i>VAUXP/VAUXN</i>	Positive / Negative terminal for auxiliary analog input
<i>VP/VN</i>	Positive / Negative terminal for analog input
<i>WB</i>	Write Back

CHAPTER 1: Introduction

1.1 Background Information

An overview of the project fields that matter is provided in the following sections to help identify and understand some facts or knowledge related to this project.

1.1.1 MIPS

MIPS which stands for Microprocessor without Interlocked Pipelined Stage is a microprocessor that is developed based on the Reduced Instruction Set Computer (RISC) architecture. The book published by John L. Hennessy and David A. Patterson as mentioned in [4] gives insight into the MIPS architecture used in this project. MIPS processors design has always aimed to improved instruction throughput during operation which is to complete as many instructions as possible in one clock. To achieve the goal, MIPS processors breaks a single instruction into 5 independent stage and apply pipelining to fully utilize every single clock cycle. Figure 1-1 shows how multiple instructions are overlapped in execution.

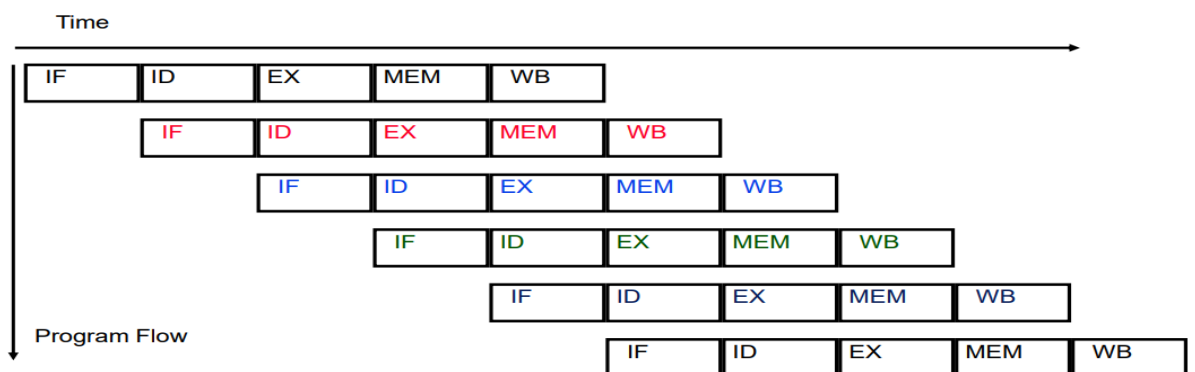


Figure 1-1: MIPS 5 stage pipeline as in [9].

The instruction execution is divided into 5 stages:

- IF (Instruction Fetch): Fetch the new instruction from instruction cache and update program counter (PC).
- ID (Instruction Decode): Decode the instruction and perform operands fetch from register file.
- EX (Execution): Perform arithmetic and logical operation in ALU.

- MEM (Data Memory Access): Read/write the data from/to Data Memory.
- WB (Write Back): Write the result data into the register file.

1.1.2 ADC

ADC as known as Analog-to-Digital Converter is used to convert analog signal to digital signal for processor to use according to [3]. In computer organisation and architecture, processor is like a brain and ADC is like a sensory organ in human. Human can sense and react to everything in this world that have analog signal behaviour like sound, light, heat, humidity, pressure and so on. However, in digital world there is only 1 and 0. Therefore, for computer to sense the environment like human, ADC is needed to convert the continuous signal to digital or discrete signal before computer can make use of it and act accordingly.

1.2 Motivation

There are a few reasons that project on 32-bit RISC pipeline microprocessor is initiated in the Faculty of Information and Communication Technology (FICT) of University Tunku Abdul Rahman (UTAR). The motivations are as follows:

- Getting a microprocessor design with full documentation and information on both front-end design process (modelling and verification) and back-end physical design (synthesis and implementation) for research and academic purposes is not easy. Knowing that semiconductor companies are taking microprocessors as trade secret with commercial purpose, it is hard to have their Intellectual Property (IP) shared just for academic purpose. Even if it could happen, the cost will be unaffordable.
- Some microprocessor cores can be obtained from the Internet and most of them are provided by OpenCores (<http://www.opencores.org/>) with no charge. However, those processors are lacking comprehensive documentation and also not implementing the entire MIPS Instruction Set Architecture (ISA). Hence, they are not suitable for customization and reuse.
- Verification specifications of the RISC microprocessor core on the Internet are incomplete as well and this might slow down the overall design process.

CHAPTER 1: Introduction

- Without a functionality verified front-end design, the physical design phase will also be affected. The back-end design will not be able to carry out smoothly and the processes might need to repeat from time to time whenever the front-end makes any changes.

The RISC32 project that has been initiated in UTAR aims to provide solutions to all the problems mentioned above by creating a 32-bit RISC core-based development environment in order to assist research works in the area of soft-core as well as the application specific hardware modelling. Up to date, the RISC32 project in UTAR has completed the CPU designs that supports basic instructions that is similar to MIPS instructions. The Coprocessor 0 (CP0) is also available to interface with I/O devices and handle interrupts.

Previously in this RISC32 project, three communication interfaces like UART, SPI and GPIO have been designed. However, those interfaces are more to machine-machine interface. While this project will be focused on making human-machine interface possible in the RISC32 processor. Fortunately, with the available microarchitecture design developed in the UTAR FICT as demonstrated in [5],[9],[13] and [14], we can easily gain the software or firmware flexibility advantage without having to rely on and wait for third party community to develop for us.

1.3 Problem Statement

So far, the MIPS ISA compatible pipeline processor has included the Central Processing Unit (CPU), basic memory, flash controller, Coprocessor 0 (CP0), GPIO controller, SPI controller, UART controller with all functionalities verified. However, the current RISC32 pipeline processor still does not have the functionality to handle analog signal that come from various analog sensor such as voltage sensor, temperature sensor, proximity sensor, humidity sensor and light intensity sensor as mentioned in [10]. Therefore, an ADC controller unit and ADC device have to be implemented.

1.4 Project Scope

The project is to design and integrate an ADC controller unit for the RISC32 pipeline processor that has been developed previously. The ADC device that is used in this project is XADC model ug480 and the datasheet provided by Xilinx is going to be used to understand the attribute and properties. The specifications of the ADC controller unit

and the instantiation of XADC will be functionally verified by developing testbenches. Besides, with a new IO device controller unit implemented, there are some changes need to be made on the memory-mapped IO address value for the new IO registers.

Furthermore, an Interrupt Service Routine (ISR) will be developed specifically to handle the interrupt request generated by the ADC controller unit. The ISR will be integrated into the existing IO exception handler of the RISC32 pipeline processor, and the priority interrupt controller's register will have one bit assigned for ADC controller unit interrupt enable. In addition, a simple MIPS programs will also be written to test out the functionality of ADC controller unit after integration and have the execution of ISR verified at the same time.

On top of that, a complete documentation of the ADC controller unit will be produced at the end of project. The documentation will include verification specification, verification methodology, testcase with testbench coding, figures to help with illustrating. It is important to make sure the verification specifications of the ADC controller unit are well-developed before proceeding to physical design phase. Otherwise, the physical design process might not be able to carry out smoothly in the future.

1.5 Project Objectives

The objectives of this project are:

- To develop an ADC controller unit
 - Review the instantiation of XADC using the Vivado IP catalog.
 - Develop the microarchitecture requirements and specifications.
 - Model the ADC controller unit using Verilog Hardware Description Language (HDL).
 - Develop a testbench in Verilog HDL to verify the ADC controller unit's functionality

- To integrate the ADC controller unit into the RISC32.
 - To develop the interface between the ADC controller unit and the RISC32 using I/O memory mapping technique.

- To develop an ADC controller's Interrupt Service Routine in MIPS assembly language and integrated into the exception handler.
- Develop a test program in assembly language to verify the overall functionality.

1.6 Impact, Significance, and Contribution

After this project is done, RISC32 microprocessor can handle analog signal by connecting external analog device to the chip since this project can provide a complete core-based development environment of RISC32 microprocessor and proper interfacing system for connecting the ADC controller unit to the microprocessor. The development environment is referred to the availability of the following:

- A complete design documentation of the chip system specification, the architecture specification, and the micro-architecture specification.
- A fully functional well-developed interfacing system between the CPU and the ADC controller unit in the form of synthesis-ready RTL that is written in Verilog HDL.
- A well-developed verification specification of the ADC controller unit. The verification specification contains suitable verification methodology, verification techniques, testbench architecture, test plan and so on.
- A complete system to handle various type of analog signal from external sources.

This project can contribute to develop an environment that mentioned above by providing support to the hardware modelling research work. With the available well-developed basic RISC RTL model, the verification environment, as well as the design documents, researchers will be able to initiate their own research related to RTL model in the MIPS environment and can have their model functionality verified in a short period. As a result, the research works in the future could be done faster and easier.

Chapter 2: Literature Review

2.1 Overview of Xilinx XADC

Xilinx has invented a dual 12-bit 1 Mega sample per second (MSPS) ADC, namely XADC for its own 7 series FPGAs. The XADC is built into the FPGA including several on-chip sensors. The ADCs and sensors are functionally tested, and the detail specifications are documented in the datasheet. In fact, The XADC is seamlessly designed for the 7 series FPGA to do the data conversion with the JTAG (Joint Test Action Group) hardware interface as mentioned in [15]. There are many functions implemented in the XADC, but only those related function and properties will be discussed in this literature review.

2.1.1 Instantiation and application of XADC

Any FPGA that does not have the JTAG hardware interface must instantiate the XADC during the design process in order to access the status register of the XADC according to [15]. In this case, Xilinx have provided the detail for instantiation with a few examples as a guideline for application. The figure 2-1 below shows the ports on XADC primitive while the ports description and the Verilog instantiation code is attached in appendix.

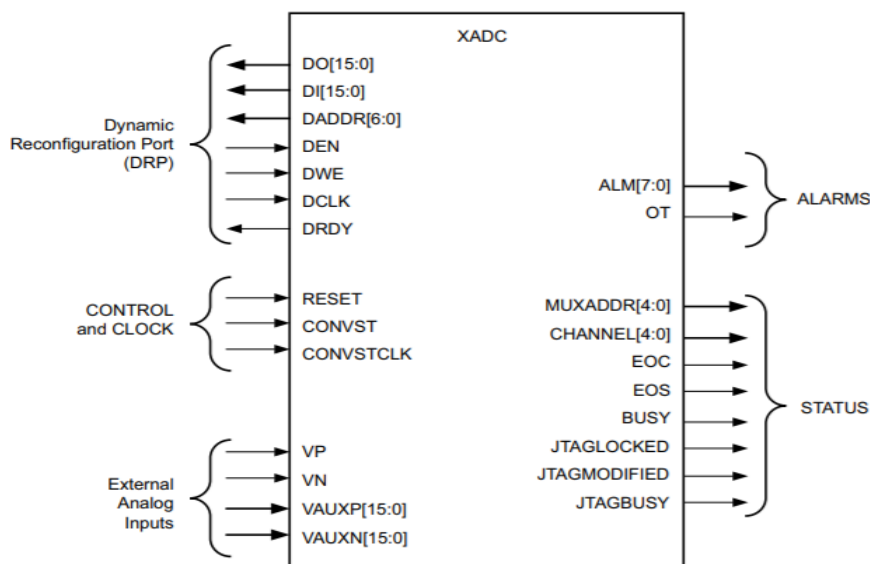


Figure 2-1: XADC Primitive Ports extracted from [15].

In the Verilog code provided by Xilinx, there are initialization made to set the initial value for the XADC register which allows the XADC to function on start up without configuration of registers. Apart from that, Xilinx also provided a sample testbench

code for user to test the functionality of XADC with the aid of Vivado tools, instantiation wizard. The following sections will review the register interface within.

2.1.2 XADC Register Interface

XADC itself contains status register and control register which are all 16-bit. Both registers can be accessed by external device through the dynamic reconfiguration port (DRP). DRP will be the interface to have communications with other FPGA logic port or the RISC32 in this project. In other words, every write and read operation upon XADC will go through DRP and it can only handle one operation at a time as stated in [15].

XADC has used 7-bit address location to allocate Status register and Control register. The last bit is used to differentiate Status register and Control register. First 64 address locations are assigned to Status register with the address value DADDR [6:0] = 00h to 3FH. These addresses in XADC will only be available for read operation. The usage of status register is to store the results of analog-to-digital conversion from all the on-chip sensors and external analog channels. The 12-bit result will be left-aligned in the 16-bit size status register. Each status register has a specific name and function to store the converted value from various channel. For example, Temp, V_{CCINT}, V_{CCAUX} and so on. Within status register, there is one flag register allocated in the last address(3FH) which the function is to monitor interrupt status. In XADC, interrupt is called alarm, which only triggered whenever analog value fall in undesired range set by user according to [15].

While for Control registers, there are 32 of them allocated at addresses DADDR [6:0] = 40H to 5FH and are readable and writable. These registers are used to control all XADC functionality, some functionality that is useful for this report will be discussed in following section.

Figure 2-2 below shows the register interface and the full descriptions for registers are attached in the appendix A.

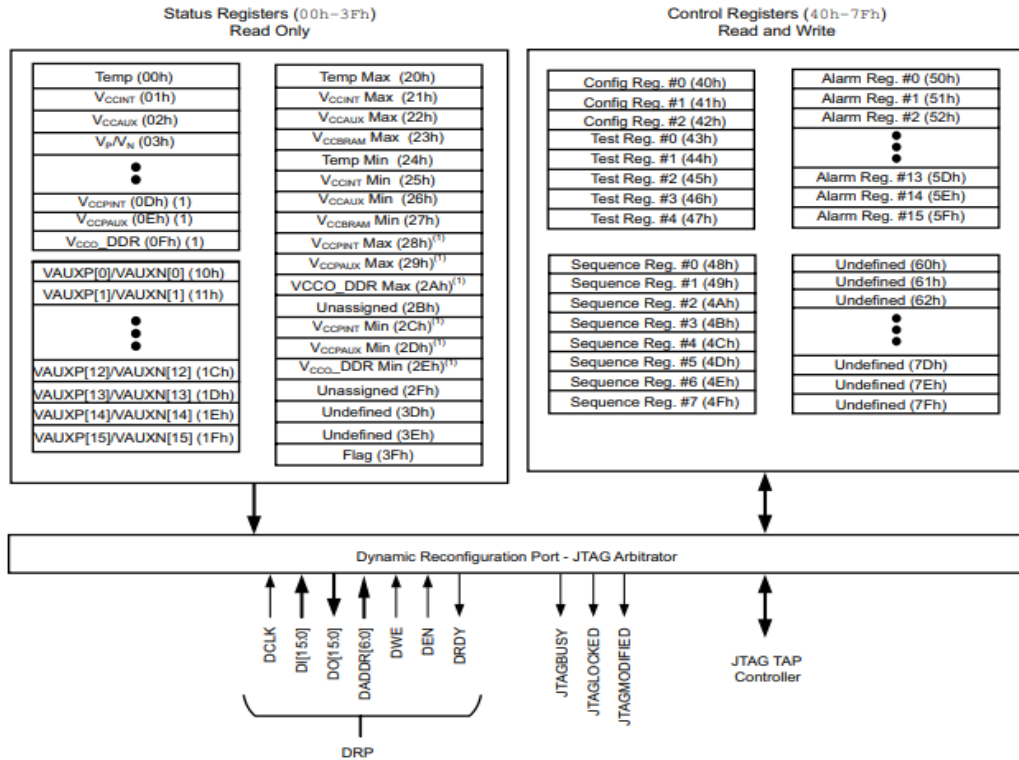


Figure 2-2: XADC Register Interface extracted from [15].

2.1.3 XADC functionality

The first three registers in the Control Register Block are called configuration registers. By changing the value in configuration registers, the XADC operating modes can be changed as mentioned in [15]. Figure 2-3 shows the three configuration registers.

DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	Config Reg #0 DADDR[6:0] = 40h
CAVG	0	AVG1	AVG0	MUX	B $\bar{0}$	E \bar{c}	ACQ	0	0	0	CH4	CH3	CH2	CH1	CH0	
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	Config Reg #1 DADDR[6:0] = 41h
SEQ3	SEQ2	SEQ1	SEQ0	ALM6 (Note1)	ALM5 (Note1)	ALM4 (Note1)	ALM3	CAL3	CAL2	CAL1	CAL0	ALM2	ALM1	ALM0	OT	
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	Config Reg #2 DADDR[6:0] = 42h
CD7	CD6	CD5	CD4	CD3	CD2	CD1	CD0	0	0	PD1	PD0	0	0	0	0	

Figure 2-3: The three Configuration registers in XADC extracted from [15].

There are two operating modes in XADC namely single channel mode and continuous sequence mode.

In single channel mode, there will be one analog input value acquired and converted. While for continuous sequence mode, user can choose to have multiple analog input value monitored and there are some further configurations that can be done to the sequence mode itself.

CHAPTER 2: Literature Review

In fact, both modes have other properties that can be configured to suite the user requirements which are averaging, analog input mode and sampling mode.

Averaging

-User could decide to get an average from a certain amount of number of samples by configuration. In this case, XADC provided 16, 64, 256 averaging sample. For example, if 16 averaging example is chosen, XADC will capture the analog input 16 times and do an average before storing into the respective status register as in [15].

Analog input mode

-There are two analog input mode provided, which are bipolar and unipolar. They are different representation of value. There are certain cases where bipolar would be more prefer because of its two's complement coding especially in calculation case. Figure 2-4 and figure 2-5 below show both modes.

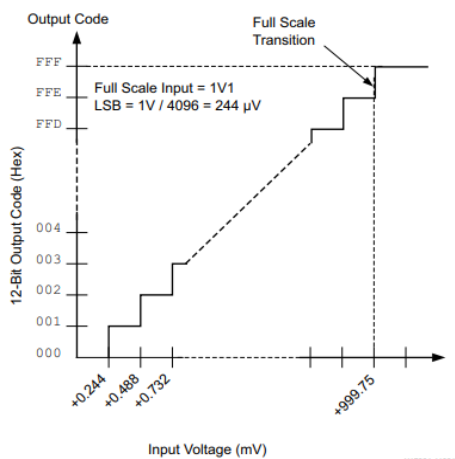


Figure 2-4: unipolar data format

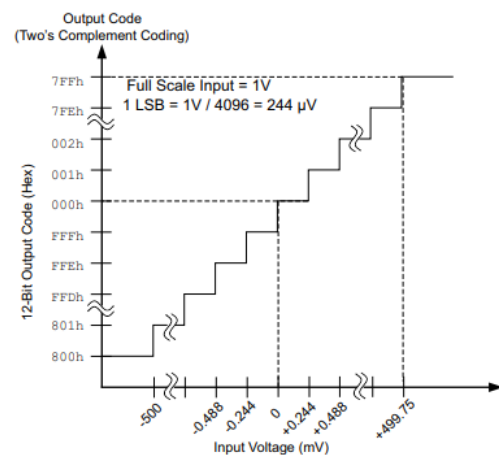


Figure 2-5: bipolar data format

(Figures extracted from [15])

Sampling mode

-There are two sampling mode which are continuous sampling and event-driven sampling. The former will make the XADC to keep capturing and converting the analog value while the latter will only capture and convert whenever user send a signal to the convert start (CONVST) port. The Event-driven sampling mode can allow user to decide when the sampling should start according to [15].

Other than that, there are also registers that are used to configure the alarm as known as interrupt namely alarm threshold registers. These registers are where the threshold values are stored. Whenever the analog input value does not fall within the threshold, it will trigger the alarm as shown in [15].

In order to have full control on XADC, the role of each registers need to be understood and all the detailed information about the registers can be reviewed in appendix.

2.2 Memory-mapped I/O

For CPU to communicate effectively with the I/O devices, IO-memory mapped technique is implemented to have full control on the transmission of data. Memory-map I/O (MMIO) is one of the general methods for assembly language program to address an I/O device. It is the I/O scheme where portions of address space are allocated specifically to I/O devices and reads and writes to those addresses are interpreted as commands to the I/O device as mentioned in [4]. With MMIO, CPU views an I/O device as a set of special-purpose registers. Table 2-1 discusses the three general types of the special-purpose registers used in MMIO.

Table 2-1: Three general types of special purpose registers used in MMIO as in [9].

Register Type	Description
Status register	<ul style="list-style-type: none"> Used to provide status information about the I/O device. Store the converted analog value from the XADC. Usually use to read only.
Configuration/Control register	<ul style="list-style-type: none"> Used to store data that configure and control the I/O device. Both readable and writable.
Data register	<ul style="list-style-type: none"> Used to read data from or send data to the I/O device. Both readable and writable.

By using MMIO method, the addresses of the registers in each of the I/O devices are assigned in a dedicated portion of the kernel's virtual address space. Each of the registers in the I/O controller must have a fixed and unique memory address within the mentioned address space in order for the CPU to access the specific register easily

without crashing with other memory location. Documentation should be made carefully to help clarify the address space allocated to each I/O devices' registers.

The benefits of using MMIO is that it keeps the instructions set small by adhering the design principles of MIPS, that is keeping the hardware simple via regularity as shown in [8]. No new dedicated instructions are required in MMIO to simply read or write those special addresses because it allows the normal load and store instructions to be used for referencing, manipulating, and controlling both memory and I/O devices. The memory address that is being used will determine which type of device (memory or I/O device) to be accessed.

2.3 ADC controller designed in [2]

In [2], Tan Beng Liong has briefly designed the register transfer level (RTL) of ADC controller unit by using Verilog HDL without any documentation. Basically, the unit is integrated in RISC32 by applying the Wishbone interface connection. Within the unit, there is a registered mealy model finite state machine constructed as well as the instantiation of XADC ug480. Although the design was done quite good and simple with all the basic function implemented, there are several problems detected in the RTL. The problems are like:

Unnecessary register and state: After optimization of code, A few registers and state would be redundant.

Incomplete read operation: The unit designed only allow read operation on status registers while there is no way to read on control registers.

Absence of interrupt function: The interrupt function is not designed along, and there is no interrupt request signal send out from the unit. Further adjustment is needed during the integration in RISC32 in order to have a complete interrupt service routine.

Chapter 3: Proposed Method/Approach

3.1 Methodologies and General Work Procedures

Design methodologies is important to make sure the design process goes in plan. It will act as a guideline to ensure the design process flow well. A suitable methodology can also ensure the goals and objectives are being achieved with bugs detection along the design process. At the end of design process, a well-documented project report can be produced as well.

Generally, there are 3 types of design methodologies which are mixed design methodology, top-down design methodology and bottom-up design methodology as mentioned in [6]. In this project, the top-down design methodology will be used for designing and developing the ADC controller unit. By using this methodology, the top-level representation of a unit is first defined, followed by the lower-level representations block and sub-block if there is any.

However, since Xilinx XADC is required in this project, the functionality and applications are studied in advance before the unit's design is started. Applications of the XADC is also demonstrated first to make sure that the IP can be used without any issue.

3.1.1 RTL Design Flow

In the RTL design flow as shown in Figure 3-1 below, physical design will be less likely to be involved. In this report, unit level specification in the micro-architectural level design will be emphasized because the ADC controller to be designed is in the unit level. While for block level, there is only one block which is XADC that is already available. Hence, no modelling and specifications are required, only application is needed. Therefore, the ADC controller unit has to be designed specifically to interface with the XADC Dynamic Reconfiguration Port (DRP) as in the datasheet.

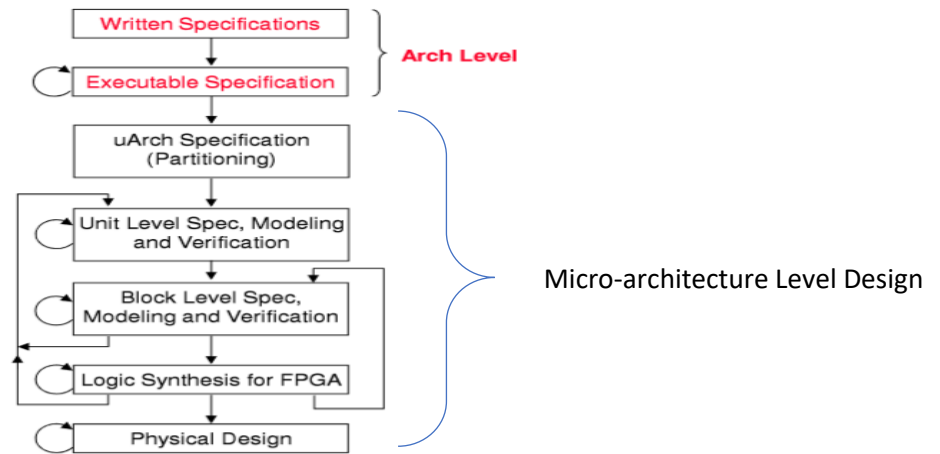


Figure 3-1 RTL design flow from [6]

3.1.2 Micro-architecture Specification

Micro-architecture specification will describe the internal design of the ADC controller unit. The internal design of the ADC controller unit will be described with detailed and design-specific technical information in order for RTL coding to begin. In this project, the unit level of the ADC controller will include the following information:

- Functionality/feature description
- Interface and I/O pin description
- Internal operation: function table, FSM, schematic diagram, timing diagram
- Functional partitioning into block.
 - Block functionality/feature
 - Block interface and I/O pin description
- Test plan (focus on functional test)

3.1.3 RTL Modelling and Verification

With the development of the micro-architecture specification, the RTL coding on the ADC controller can begin. After coding, the RTL models are verified for functional correctness at each level. To further illustrate on this point, each block (RTL model) is verified before they are integrated into unit level. During the development of the project, if the design of the ADC controller unit does not meet all the specified functional requirements, then the design flow should be repeated. After all the RTL models have successfully met the specified functional requirements, then logic synthesis will be carried out on the targeted technology which is the FPGA technology in this project.

3.1.4 Logic Synthesis for FPGA

After the ADC controller unit has been functionally verified, the model is said to be ready for logic synthesis which is the process of converting RTL codes into optimized gate level representation (a netlist). Based on the logic synthesis result, the gate level netlist is verified again for functional correctness. If it can successfully meet all the necessary specifications, the gate level netlist is now ready for physical design. However, if it cannot meet the required specifications, depending on the severity, corrections need to be made accordingly to the gate level netlist, the RTL models, or the architecture.

3.2 Design Tools

Each stage of the design jobs requires the use of appropriate design tools to help automate the design work. Hence, there exist Electronic Design Automation (EDA) tools for design work at each level of abstraction. Since the RTL model of the ADC controller unit is designed by using Verilog Hardware Description Language (HDL), hence a simulator is needed to emulate the Verilog HDL. Figure 3-2 below shows some simulators and the comparison between them.




Simulator	Incisive Enterprise Simulator	ModelSim	VCS
Company			
Language Supported	<ul style="list-style-type: none"> • VHDL-2002 • V2001 • SV2005 	<ul style="list-style-type: none"> • VHDL-2002 • V2001 • SV2005 	<ul style="list-style-type: none"> • VHDL-2002 • V2001 • SV2005
Platform supported	<ul style="list-style-type: none"> • Sun-solaris • Linux 	<ul style="list-style-type: none"> • Windows XP/Vista/7 • Linux 	Linux
Availability for free?	✗	✓ (SE edition only)	✗

Figure 3-2: Comparison among 3 different Verilog simulators from [1],[11] and [12].

Based on the comparison above, it is obvious that the ModelSim from Mentor Graphic is the best choice among others to be used as the design tool for this project as they offer a free license for Student Edition version. Even though there is certain degree of limitations on the ModelSim Student Edition version, it is adequate to be used for this

CHAPTER 3: Proposed Method/Approach

project. In addition, it supports Microsoft Windows platform as well. Although the other two simulators can also offer great features for Verilog stimulation, the price are too expensive and certainly not affordable to be used in this project.

While for the synthesis tools, there are a few logic synthesis tools that can target FPGA. Those logic synthesis tools include Quartus by Altera, Synplify by Synopsys, Vivado Design Suite by Xilinx, Encounter RTL Compiler by Cadence Design System, and so on. Among all the available logic synthesis tools, the Xilinx Vivado Design Suite from [16] is selected for this project as it can support the FPGA that we have in UTAR and only Vivado can provide the XADC that we need in this project.

3.2.1 ModelSim SE-64 10.5

ModelSim is the industry-leading simulation and debugging environment developed by Mentor Graphic specifically for HDL (Hardware Description Language) based design. The student edition of the ModelSim is used for Verilog design simulation. Besides, ModelSim simulator supports both the Verilog and VHDL languages. This simulator can also provide syntax error checking and waveform simulation which play an important part in developing the project. The timing diagrams and the waveforms are very useful in verifying the functionality of the model after writing a testbench. Also, ModelSim has user-friendly interface as well as tutorial provided which helps user to get started easily [11].

3.2.2 Xilinx Vivado Design Suite

Xilinx has designed the software, Vivado Design Suite for synthesis and analysis of HDL designs which enables the developers to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer easily. On top of that, it is a good design environment for FPGA products from Xilinx but it cannot be used with those FPGA products from other vendors [16]. In this project, the XADC IP is obtained from the IP Catalog in Vivado. The XADC Wizard can be found under the "FPGA Features and Design" folder in IP Catalog. During the project set up in Vivado, project part of "xc7a100tcs324-1" from project family "Artix-7" is chosen.

3.2.3 PCSpim/QtSpim

PCSpim is the Window version of spim. It is a software stimulator that loads and executes assembly language program for the MIPS RISC architecture. Besides, it also provides a simple assembler, debugger, and a simple set of operating services. Therefore, it is used in this project for developing the MIPS test program for functional verification. The name has changed to QtSpim for latest Window Version.

3.3 Technologies Involved

3.3.1 Field Programmable Gate Array (FPGA)

As mentioned earlier, the logic synthesis of the ADC controller unit will be eventually carried out on the FPGA technology. The FPGA technology is an integrated circuit (IC) that is programmable in the field after manufacture. FPGAs have been used widely by engineers in the design of specialized integrated circuits that can be later produced hard-wired in large quantities for distribution to computer manufacturers and end users. It is selected for prototype development in this project due to its benefits of cost efficiency, high flexibility and good scalability when compared to the other technologies.

3.4 Implementation Issues and Challenges

The ADC controller unit in this project will be interfacing with the XADC from Xilinx which will be instantiated in RISC 32 also. The XADC has its own documentation about its behaviors and functionality. Hence, in order to make sure the XADC is functioning when it is implemented in RISC32, the datasheet need to be studied thoroughly. Besides, the XADC possesses a lot of functionality, and it is a challenge to identify which functionality and information are useful in this project and are able to fulfill this project's objectives. Although B.L Tan has briefly designed the controller unit with Verilog HDL, it has an issue with lack of documentation and comments in the coding which makes it hard to understand his work. If the ADC controller unit is not designed well, RISC32 will not be able to perform any operation on the analog signal. Hence, a Finite State Machine is needed to organize the data flow in and out of XADC since Xilinx does not allow any developer modification on the XADC primitive.

CHAPTER 3: Proposed Method/Approach

3.5 Timeline

3.5.1 Gantt Chart for Project I

The Design of an ADC Controller for 5-stage Pipeline RISC32 Microprocessor

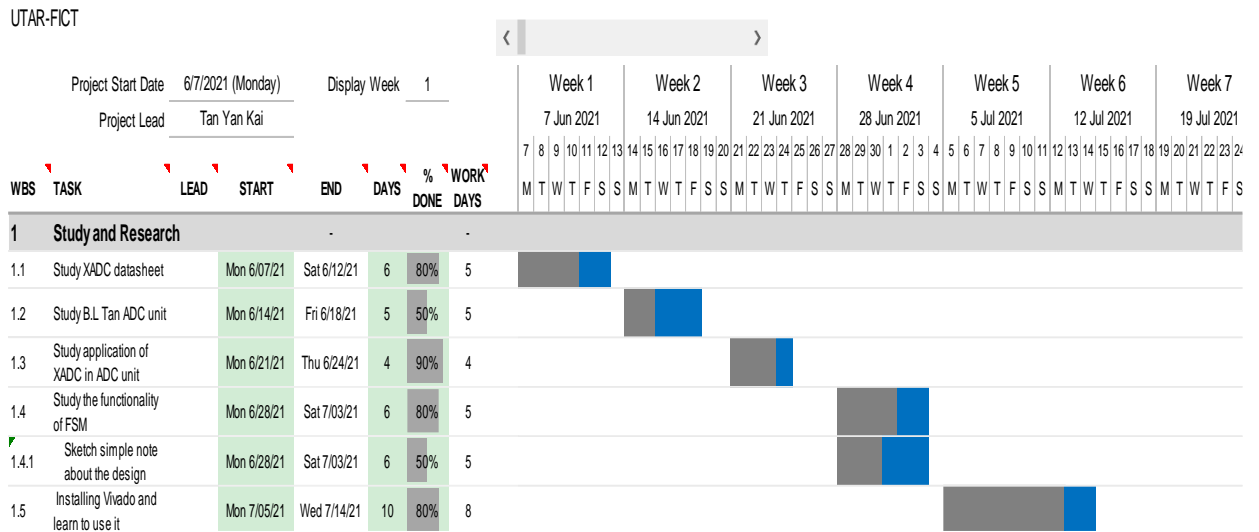


Figure 3-3: Gantt Chart for Project 1

The Design of an ADC Controller for 5-stage Pipeline RISC32 Microprocessor

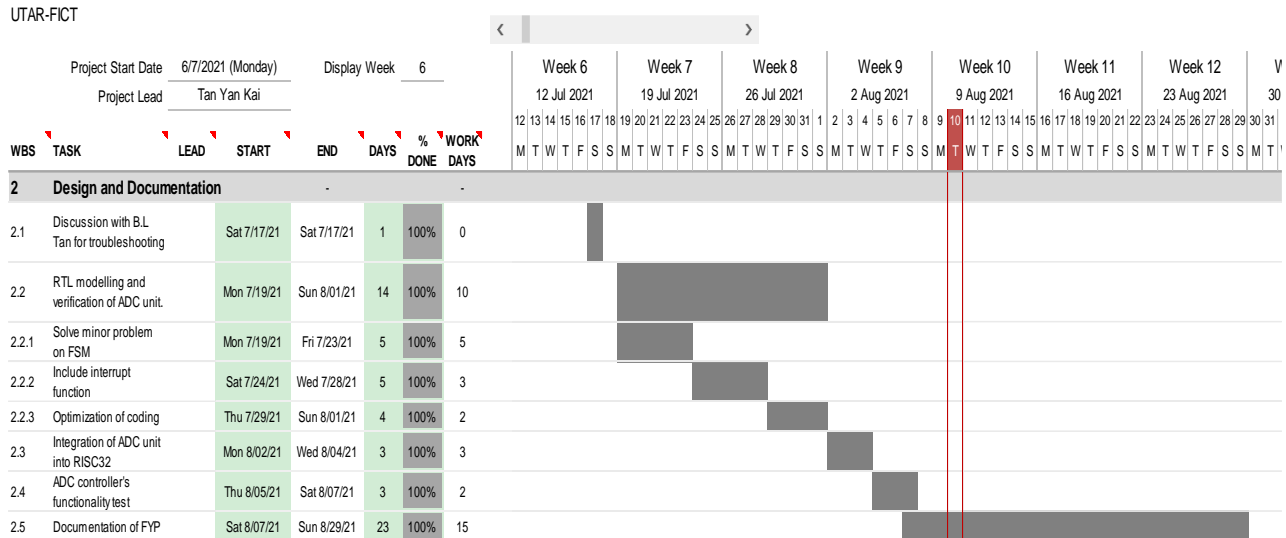


Figure 3-4: Gantt Chart for Project 1

3.5.2 Gantt Chart for Project II

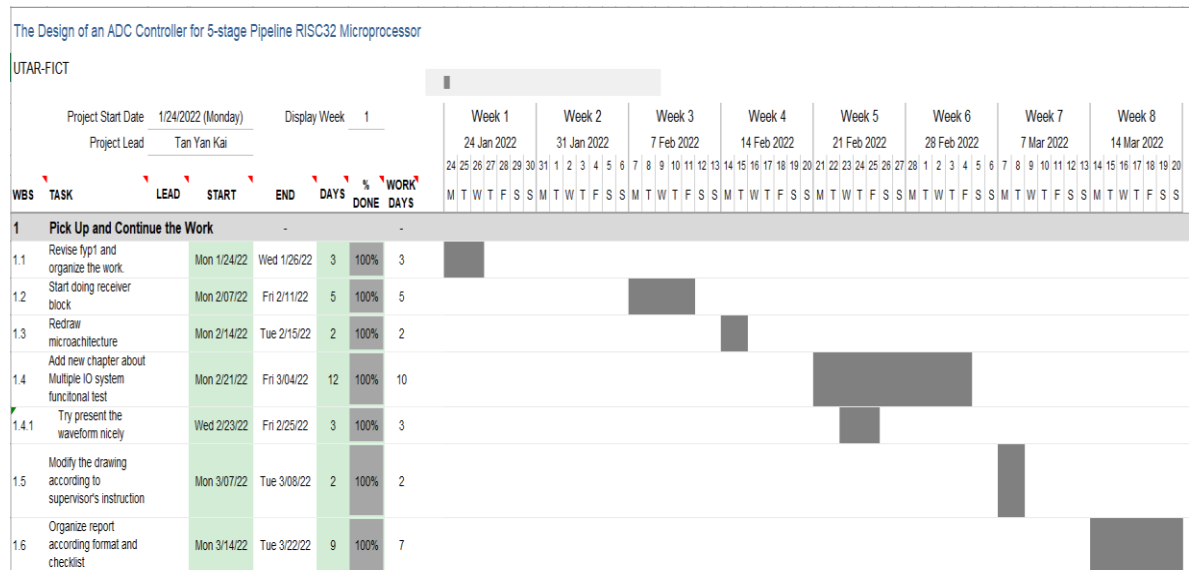


Figure 3-5: Gantt Chart for Project 2

Chapter 4: System Specification

4.1 System Overview of the RISC32 Pipeline Processor

4.1.1 RISC32 Pipeline Processor Architecture

The developed RISC32 pipeline processor is a 32-bit pipeline processor that consists of 3 major components which include Central Processing Unit (CPU), memory system and I/O system as shown in [13] and [14]. The developed CPU is said to be compatible to the 5-stage 32-bit MIPS Instruction Set Architecture (ISA) and it can support up to 49 instructions, covering arithmetic, logical, data transfer, program control, and system instruction classes. In addition, the memory system developed in this processor has a 2-level memory hierarchy with the first level consists of cache, Boot ROM as well as Data and Stack RAM whereas the second level contains a Flash memory. On the other hand, the I/O system of this processor contains GPIO controller, SPI controller, UART controller, ADC controller (about to be designed), Priority Interrupt controller and General-Purpose Register (GPR) unit. In addition, it also has a branch predictor that helps to improve the performance of the RISC32 processor in running program in terms of the number of clock cycle spent. An architectural overview on the RISC32 pipeline processor that has been developed is shown in Figure 4-1. On the other hand, the detailed specification of the RISC32 pipeline processor is also provided in Table 4-1.

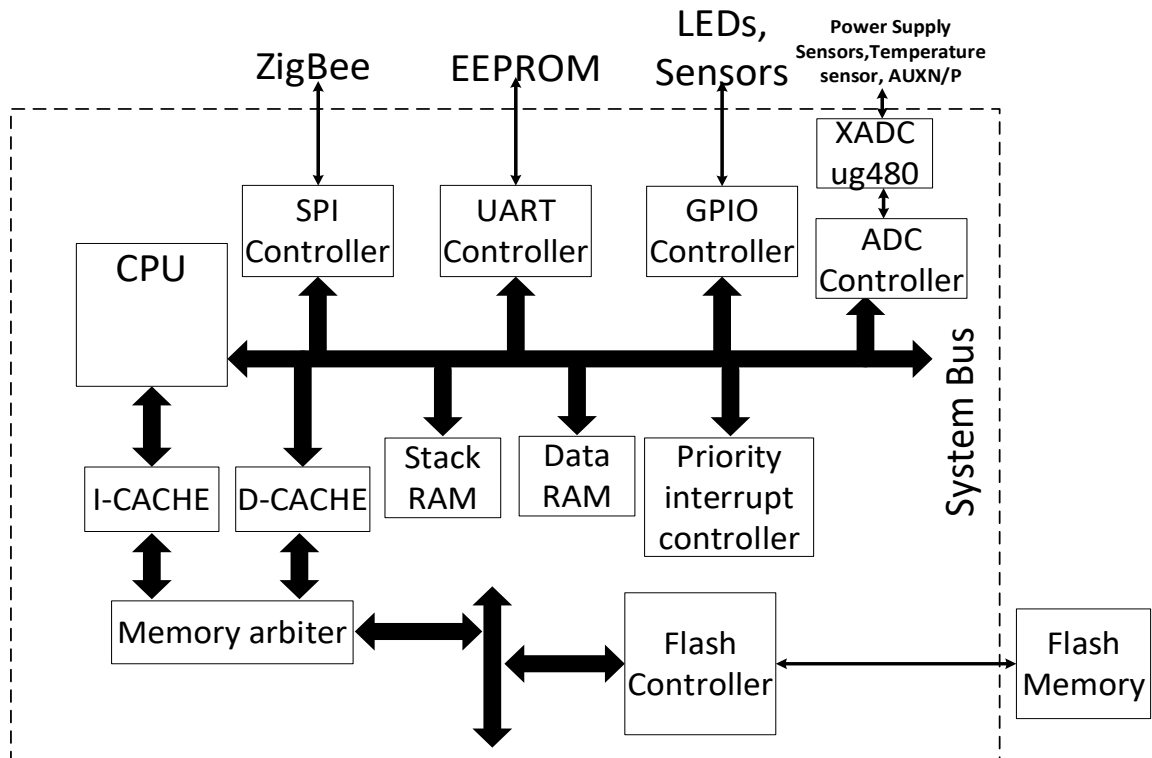


Figure 4-1: An overview on the architecture of the RISC32 pipeline processor as shown in [9] with UADC added.

Table 4-1: Specification of the RISC32 pipeline processor as shown in [9].

		Pipeline
Frequency (MHz)		50
Instruction's cycle		5, overlapping
Branch predictor		64 entries 4 ways associative
Common features (Static)	Memory	4kBytes boot ROM, 128kBytes user access flash, 8kBytes RAM (Data & Stack), 1kBytes i-cache, 32Bytes d-cache, 512Bytes Memory Mapped I/O Register
	Communication interface	UART, SPI, 32 GPIO pins
Partial Bitstream start address		0x00A8_0000
Bitstream size		1,404,992 bits / 43906 words
FPGA board		Nexys 4 DDR (XC7A100T)
FPGA Resources (Overall)	LUT	8266
	LUTRAM	315
	FF	5643
	BRAM	3.50
	IO	46
	BUFG	1

4.1.2 Functional View of the RISC32 Pipeline Processor

The RISC32 pipeline processor that has been developed consists of 5 hardware stages which include Instruction Fetch (IF), Instruction Decode and Operand Fetch (ID), Execution (EX), Memory Access (MEM), and Write Back (WB) stages. Different hardware components are allocated in each of these pipeline stages. Therefore, every instruction will need 5 clock cycles to run through all the 5 stages in order to complete its execution. Since the data hazard issue due to the Read-After-Write (RAW) data dependencies always exist in a pipeline processor, additional circuitries such as the forwarding and interlock block are built for solving the data hazards during the program execution as mentioned in [14]. The functional view of the 5-stage RISC32 pipeline processor is shown in Figure 4.

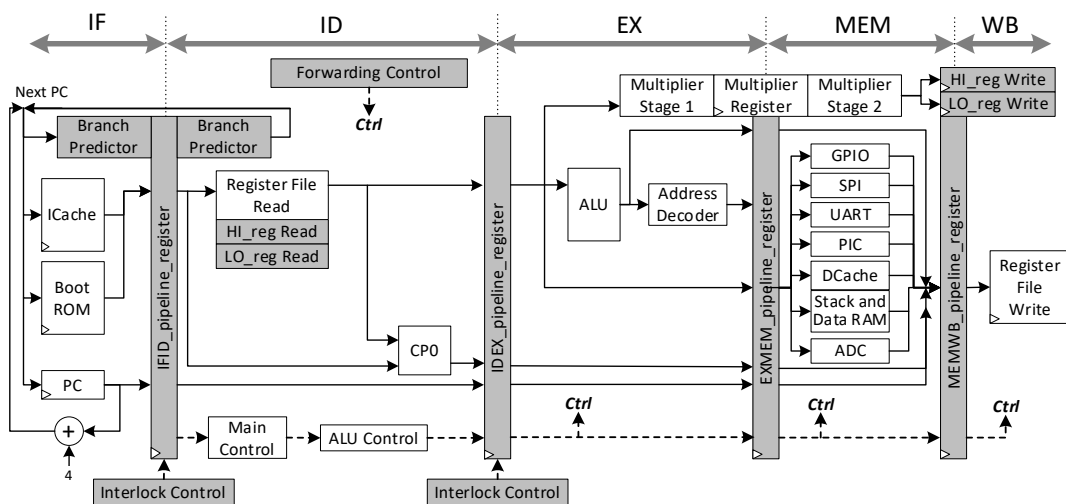


Figure 4-2: The functional view of the RISC32 pipeline processor as shown in [9] with UADC added.

4.1.3 Memory Map of the RISC32 Pipeline Processor

This RISC32 pipeline processor implements the MIPS memory space in two ways, that is by having virtual and physical addresses as mentioned in [14]. The virtual addresses are mainly used to access program instruction and data whereas the physical addresses are used to allocate physical memory such as Flash memory, Data and Stack RAM, boot ROM and I/O registers. The ADC registers in this project will take up the address from 0xBFFFFFFE2C to 0xBFFFFFFE8A in the IO peripherals registers segment. The memory map used in the RISC32 pipeline processor is presented in Figure 4-3 and the purposes of various memory allocation are discussed in Table 4-2.

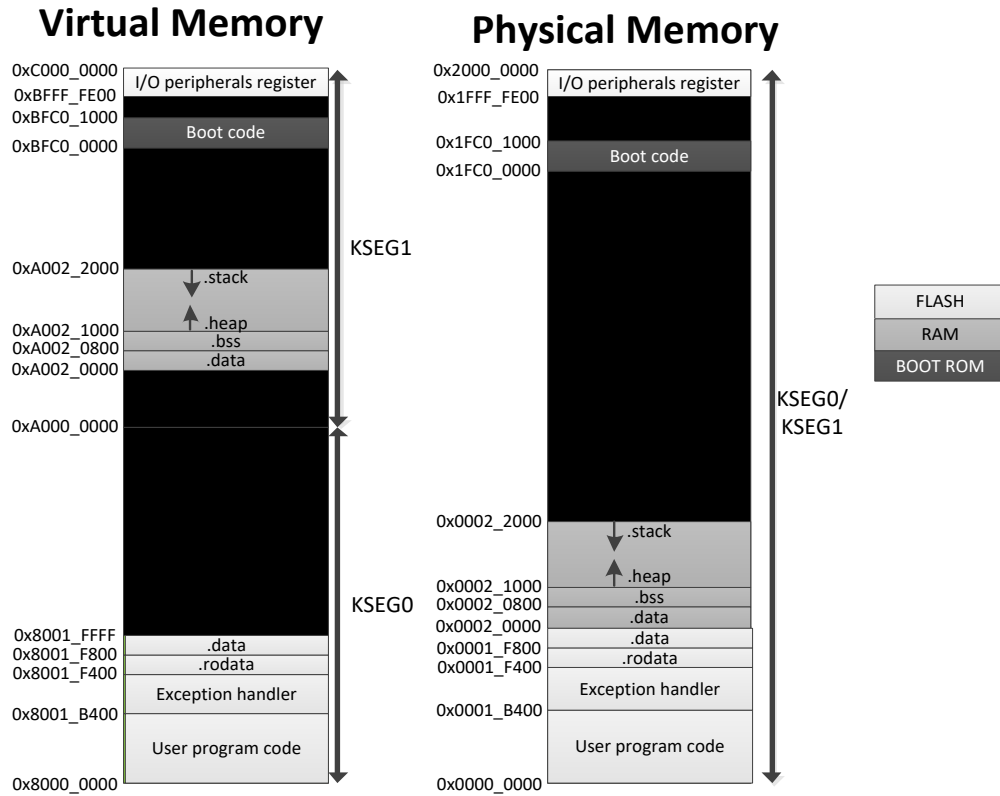


Figure 4-3: Memory map of the RISC32 pipeline processor as shown in [14].

Table 4-2: Memory map description of the RISC32 pipeline processor as in [14].

Memory Usage	Description	Memory Size
I/O peripheral register	Used as the memory-mapped registers for I/O peripheral controllers.	512 bytes
Boot code	Used to store bootloader program code for initial system configuration when powered on.	4k bytes
Stack	Used by procedure during execution to store register values.	8k bytes
Heap	Used to hold variables declared dynamically.	
Exception handler	Used to store the exception handler codes.	16k bytes
User program code	Used to store user program codes	128k bytes

4.2 Chip Interface of the RISC32 Pipeline Processor

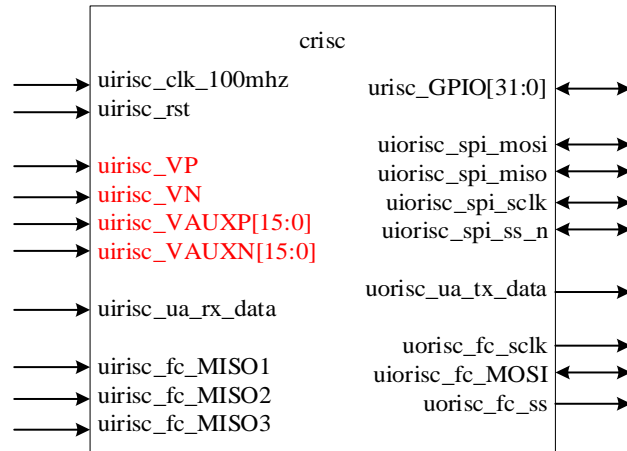


Figure 4-4: Chip interface of the RISC32 pipeline processor as shown in [9] with four inputs added.

4.3 Input Pin Description of the RISC32 Pipeline Processor

Table 4-3: Input pin description of the RISC32 pipeline processor.

Pin name: uirisc_clk_100mhz	Pin class: Global
Source → Destination: External → crisc	
Pin function: To provide a reference signal to synchronize all other signals in a system	
Pin name: uirisc_rst	Pin class: Global
Source → Destination: External → crisc	
Pin function: To reset the whole MIPS ISA compatible pipeline processor	
Pin name: uirisc_VP	Pin class: Data
Source → Destination: External → crisc	
Pin function: External Analog Inputs	
Pin name: uirisc_VN	Pin class: Data
Source → Destination: External → crisc	
Pin function: External Analog Inputs	
Pin name: uirisc_VAUXP [15:0]	Pin class: Data
Source → Destination: External → crisc	
Pin function: External Analog Inputs	
Pin name: uirisc_VAUXN[15:0]	Pin class: Data
Source → Destination: External → crisc	
Pin function: External Analog Inputs	
Pin name: uirisc_ua_rx_data	Pin class: Data
Source → Destination: External device's UART unit → crisc	
Pin function: UART standard pin – Receive Serial Data	

Pin name: uirisc_fc_MISO1	Pin class: Data
Source → Destination: Flash memory → crisc	
Pin function: SPI protocol serial input pin	
Pin name: uirisc_fc_MISO2	Pin class: Data
Source → Destination: Flash memory → crisc	
Pin function: SPI protocol serial input pin	
Pin name: uirisc_fc_MISO3	Pin class: Data
Source → Destination: Flash memory → crisc	
Pin function: SPI protocol serial input pin	

4.4 Output Pin Description of the RISC32 Pipeline Processor

Table 4-4: Output pin description of the RISC32 pipeline processor.

Pin name: uorisc_ua_tx_data	Pin class: Data
Source → Destination: crisc → External device's UART unit	
Pin function: UART standard pin – Transmit Serial Data	
Pin name: uorisc_fc_sclk	Pin class: Data
Source → Destination: crisc → Flash memory	
Pin function: SPI protocol Serial Clock signal	
Pin name: uorisc_fc_ss	Pin class: Control
Source → Destination: crisc → Flash memory	
Pin function: SPI protocol Slave Select	

4.5 Input Output Pin Description of the RISC32 Pipeline Processor

Table 4-5: Input output pin description of the RISC32 pipeline processor

Pin name: urisc_GPIO[31:0]	Pin class: Data
Source → Destination: crisc ↔ External device (LEDs, switch, etc)	
Pin function: 32 GPIO pins	
Pin name: uiorisc_spi_mosi	Pin class: Data
Source → Destination: crisc ↔ External device's SPI unit	
Pin function: SPI standard pin – Master out Serial In (MOSI)	
If the crisc is configured as a master, then uiorisc_spi_mosi will become an output, else otherwise.	
Pin name: uiorisc_spi_miso	Pin class: Data
Source → Destination: crisc ↔ External device's SPI unit	
Pin function: SPI standard pin – Master In Serial Out (MISO)	
If the crisc is configured as a master, then uiorisc_spi_miso will become an input, else otherwise.	
Pin name: uiorisc_spi_sclk	Pin class: Control
Source → Destination: crisc ↔ External device's SPI unit	
Pin function: SPI standard pin – SPI Serial Clock signal for data synchronization across devices.	
If the crisc is configured as a master, then uiorisc_spi_clk will become an output, else otherwise.	
Pin name: uiorisc_spi_ss_n	Pin class: Control
Source → Destination: crisc ↔ External device's SPI unit	

Pin function: SPI standard pin – SPI Slave Select control signal. If the crisc is configured as a master, then uiorisc_spi_ss_n will become an output, else otherwise.	
Pin name: uiorisc_fc_MOSI	Pin class: Data
Source → Destination: crisc ↔ Flash memory	
Pin function: SPI protocol serial input output pin	

Chapter 5: Micro-Architecture Specification

5.1 Functionality/Feature of the ADC Controller Unit

- Have 6-bit register address partially used to accommodate 47 IO register that can support major functionality in XADC.
 - UADC_CREG are registers used to store data temporarily before it is written into the XADC control register.
 - UADC_SREG are registers used to store the converted data from XADC status register.
- Provide interrupt.
 - Handle the alarm signal from the XADC whenever sensor measurement like Temperature, V_{CCINT} , V_{CCAUX} , V_{CCBRAM} exceeds thresholds value defined in the control register.

5.2 Unit Interface of the ADC Controller Unit

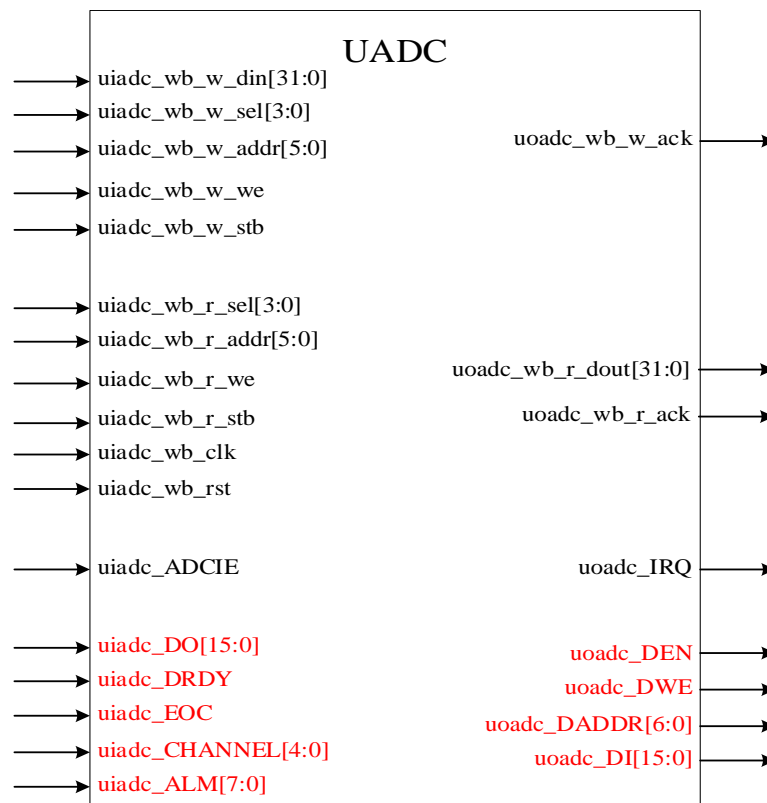


Figure 5-1: ADC controller unit interface.

5.3 Input Pin Description of the ADC Controller unit

Table 5-1: Input pin description of the ADC controller unit

<p>Pin name: uiadc_wb_w_din[31:0] Pin class: Data Source → Destination: Datapath unit → ADC controller unit Pin function: Wishbone standard data input bus (for write operation: 32 bit input data)</p>
<p>Pin name: uiadc_wb_w_sel[3:0] Pin class: Control Source → Destination: Address decoder block → ADC controller unit Pin function: Wishbone standard byte select signal to determine the granularity for writing: 0001: 1st byte selected 0010: 2nd byte selected 0100: 3rd byte selected 1000: 4th byte selected 0011: lower half-word selected 1100: upper half-word selected 0111: 3 bytes from the LSB selected 1110: 3 bytes from the MSB selected 1111: word selected</p>
<p>Pin name: uiadc_wb_w_addr[5:0] Pin class: Control Source → Destination: Datapath unit → ADC controller unit Pin function: Used to select which register to write 00H - 06H for UADC_CREG 08H - 17H for UADC_SREG</p>
<p>Pin name: uiadc_wb_w_we Pin class: Control Source → Destination: Address decoder block → ADC controller unit Pin function: Wishbone standard write enable signal – indicate current bus cycle for write 1: write cycle – write to ADC controller</p>
<p>Pin name: uiadc_wb_w_stb Pin class: Control Source → Destination: Address decoder block → ADC controller unit Pin function: Wishbone standard strobe signal (for write operation) – indicate valid data transfer cycle 1: activate ADC controller for write access 0: de-activate ADC controller for write access</p>
<p>Pin name: uiadc_wb_r_sel[3:0] Pin class: Control Source → Destination: Address decoder block → ADC controller unit Pin function: Wishbone standard byte select signal To determine granularity for reading: 0001: 1st byte selected (applicable for UADC_CREG only) 0010: 2nd byte selected (applicable for UADC_CREG only) 0100: 3rd byte selected (applicable for UADC_CREG only) 1000: 4th byte selected (applicable for UADC_CREG only) 0011: lower half-word selected (applicable for UADC_CREG & UADC_SREG) 1100: upper half-word selected (applicable for UADC_CREG & UADC_SREG) 0111: 3 bytes from the LSB selected (applicable for UADC_CREG only) 1110: 3 bytes from the MSB selected (applicable for UADC_CREG only) 1111: word selected (applicable for UADC_CREG & UADC_SREG)</p>

<p>Pin name: uiadc_wb_r_addr[5:0] Pin class: Control Source → Destination: Datapath unit → ADC controller unit Pin function: Used to select which register to read 00H - 06H for UADC_CREG 08H - 17H for UADC_SREG</p>
<p>Pin name: uiadc_wb_r_we Pin class: Control Source → Destination: Address decoder block → ADC controller unit Pin function: Wishbone standard read enable signal – indicate current bus cycle for read 0: read cycle – read from ADC controller</p>
<p>Pin name: uiadc_wb_r_stb Pin class: Control Source → Destination: Address decoder block → ADC controller unit Pin function: Wishbone standard strobe signal (for read operation) – indicate valid data transfer cycle 1: activate ADC controller for read access 0: de-activate ADC controller for read access</p>
<p>Pin name: uiadc_wb_clk Pin class: Global Source → Destination: Global clock → ADC controller unit Pin function: Global clock</p>
<p>Pin name: uiadc_wb_rst Pin class: Global Source → Destination: Global reset → ADC controller unit Pin function: Global reset 1: reset 0: no reset is required</p>
<p>Pin name: uiadc_ADCIE Pin class: Control Source → Destination: Priority interrupt controller unit → ADC controller unit Pin function: To allow the ADC to interrupt 1: enable ADC global interrupt 0: disable ADC global interrupt</p>
<p>Pin name: uiadc_DO[15:0] Pin class: Data Source → Destination: XADC → ADC controller unit Pin function: Receive 16-bit digital data</p>
<p>Pin name: uiadc_DRDY Pin class: Control Source → Destination: XADC → ADC controller unit Pin function: Receive data ready signal, indicating the new data is ready to be read out from XADC.</p>
<p>Pin name: uiadc_EOC Pin class: Control Source → Destination: XADC → ADC controller unit Pin function: Receive end of conversion signal, indicating the new data finished converted and read operation can be started.</p>
<p>Pin name: uiadc_CHANNEL[4:0] Pin class: Control Source → Destination: XADC → ADC controller unit Pin function: Receive address to store the new converted data in status register</p>
<p>Pin name: uiadc_ALM[7:0] Pin class: Control Source → Destination: XADC → ADC controller unit Pin function: Determine the occurrence of alarm/interrupt</p>

5.4 Output Pin Description of the ADC Controller Unit

Table 5-2: Output pin description of the ADC controller unit.

Pin name: uoadc_wb_w_ack	Pin class: Status
Source → Destination: ADC controller unit → Datapath unit	
Pin function: Wishbone standard acknowledge signal (for write operation) 1: normal bus cycle termination 0: no bus cycle termination	
Pin name: uoadc_wb_r_dout[31:0]	Pin class: Data
Source → Destination: ADC controller unit → Datapath unit	
Pin function: Wishbone standard data output bus	
Pin name: uoadc_wb_r_ack	Pin class: Status
Source → Destination: ADC controller unit → Datapath unit	
Pin function: Wishbone standard acknowledge signal (for read operation) 1: normal bus cycle termination 0: no bus cycle termination	
Pin name: uoadc_IRQ	Pin class: Control
Source → Destination: ADC controller unit → CP0 block & Priority interrupt controller unit	
Pin function: To request an interrupt (The uiadc_ADCIE must be pulled high before an interrupt can be sent) 1: request to interrupt 0: no interrupt request	
Pin name: uoadc_DEN	Pin class: Control
Source → Destination: ADC controller unit → XADC	
Pin function: Enable signal.	
Pin name: uoadc_DWE	Pin class: Control
Source → Destination: ADC controller unit → XADC	
Pin function: Write enable signal.	
Pin name: uoadc_DADDR[6:0]	Pin class: Control
Source → Destination: ADC controller unit → XADC	
Pin function: Address for the 16-bit data to be written.	
Pin name: uoadc_DI[15:0]	Pin class: Control
Source → Destination: ADC controller unit → XADC	
Pin function: 16-bit data to be written.	

5.5 Internal Operation of the ADC Controller Unit

Table 5-3: Functional description of the ADC controller unit's read operation.

uiadc_wb_r_stb	uiadc_wb_r_we	uiadc_wb_r_sel [3:0]	Function
1	0	0001	Enable read operation. Read 1st byte from the address specify by uiadc_wb_r_addr[5:0]

1	0	0010	Enable read operation. Read 2nd byte from the address specify by uiadc_wb_r_addr[5:0]
1	0	0100	Enable read operation. Read 3rd byte from the address specify by uiadc_wb_r_addr[5:0]
1	0	1000	Enable read operation. Read 4th byte from the address specify by uiadc_wb_r_addr[5:0]
1	0	0011	Enable read operation. Read lower half-word from the address specify by uiadc_wb_r_addr[5:0]
1	0	1100	Enable read operation. Read upper half-word from the address specify by uiadc_wb_r_addr[5:0]
1	0	0111	Enable read operation. Read 3 bytes start from the LSB from the address specify by uiadc_wb_r_addr[5:0]
1	0	1110	Enable read operation. Read 3 bytes start from the MSB from the address specify by uiadc_wb_r_addr[5:0]
1	0	1111	Enable read operation. Read word from the address specify by uiadc_wb_r_addr[5:0]

Table 5-4: Functional description of the ADC controller unit's write operation.

uiadc_wb_w_stb	uiadc_wb_w_we	uiadc_wb_w_sel [3:0]	Function
1	1	0001	Enable write operation. Write 1st byte on the address specify by uiadc_wb_w_addr[5:0]
1	1	0010	Enable write operation. Write 2nd byte on the address specify by uiadc_wb_w_addr[5:0]
1	1	0100	Enable write operation. Write 3rd byte on the address specify by uiadc_wb_w_addr[5:0]
1	1	1000	Enable write operation. Write 4th byte on the address specify by uiadc_wb_w_addr[5:0]
1	1	0011	Enable write operation. Write lower half-word on the address specify by uiadc_wb_w_addr[5:0]
1	1	1100	Enable write operation. Write upper half-word from the address specify by uiadc_wb_w_addr[5:0]
1	1	0111	Enable write operation. Write 3 bytes start from the LSB on the address specify by uiadc_wb_w_addr[5:0]
1	1	1110	Enable write operation.

Chapter 5: Micro-Architecture Specification

			Write 3 bytes start from the MSB on the address specify by uiadc_wb_w_addr[5:0]
1	1	1111	Enable write operation. Write word on the address specify by uiadc_wb_w_addr[5:0]

5.6 Example application of XADC with ADC Controller Unit in RISC32

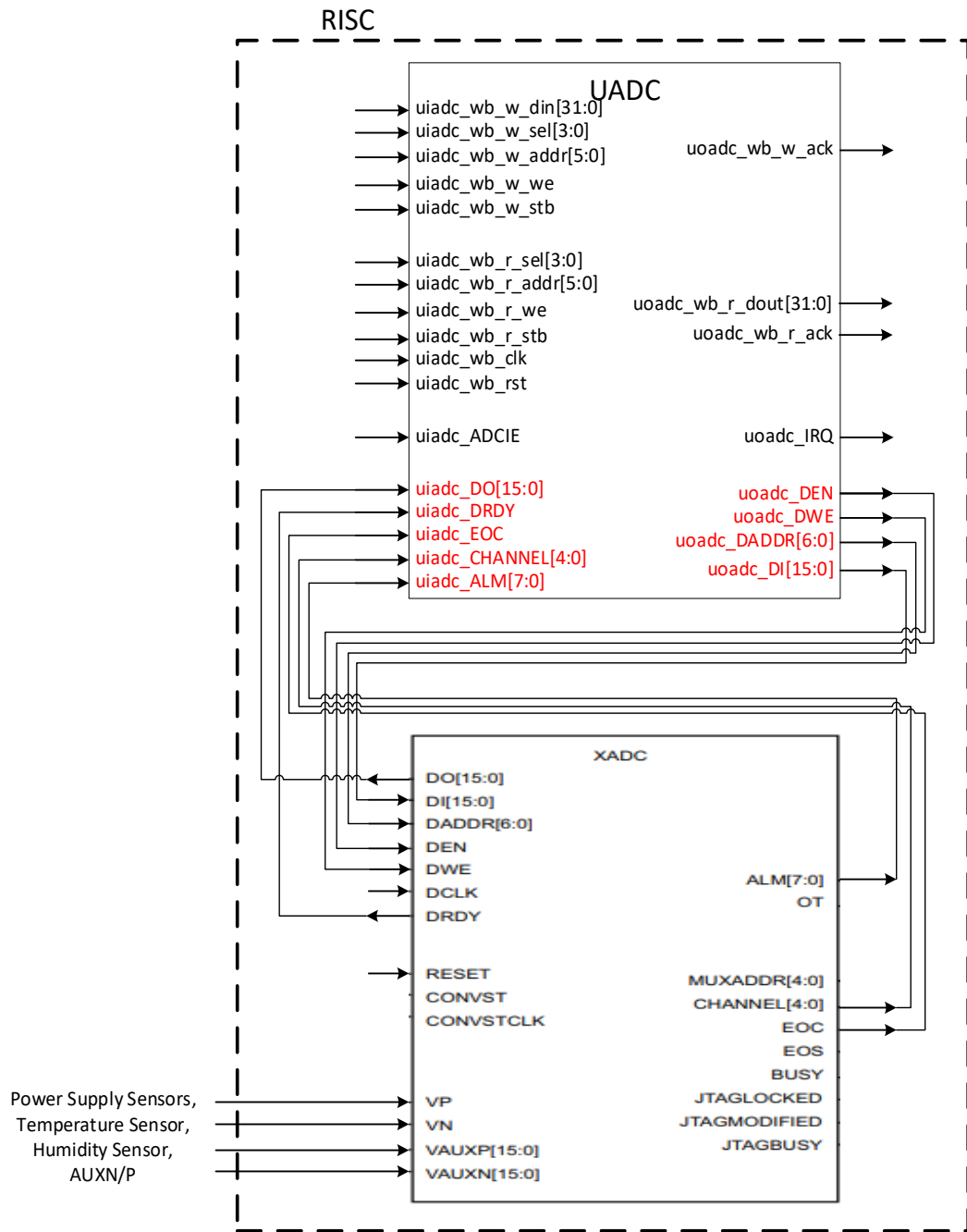


Figure 5-2: Application of XADC in RISC32

5.7 Design Partitioning of the ADC Controller Unit

The ADC controller unit consists of UADC_CREG (ADC Configuration Register) and UADC_SREG (ADC Status Register) and ADC Clock Control (badclk_ctr). The registers are designed to have same bit size and arrangement with the registers in the XADC which have been mentioned in literature review section. UADC_CREG is used to store the data before it is written into the XADC's control register for configuration. While UADC_SREG is used to store the converted data from the XADC's status register. Control FSM will take part in handling the signal during the writing and reading operation as well as exception signal from XADC.

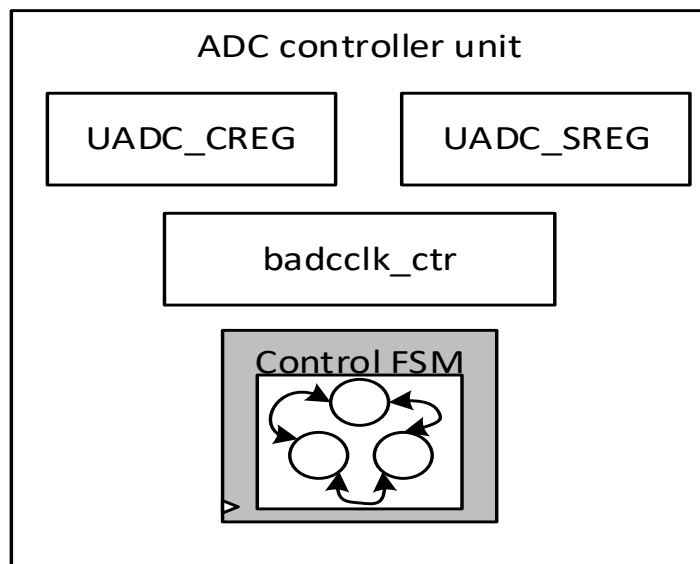
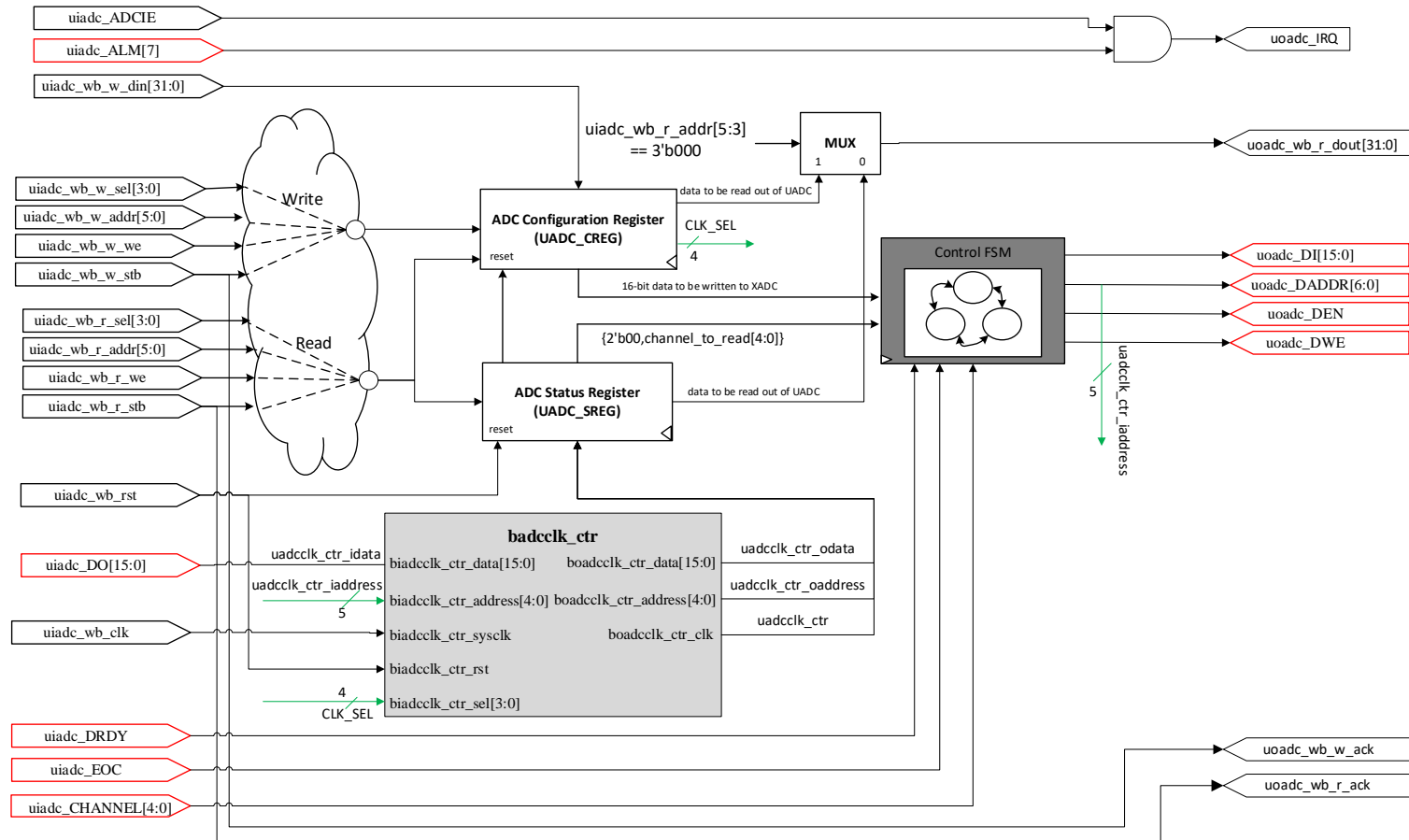


Figure 5-3: Partitioning of the ADC controller unit.

5.8 Micro-Architecture of the ADC Controller Unit



Note: The shaded areas indicate the internal blocks of the designed ADC controller

5.9 ADC Clock Control block

5.9.1 Functionality/Feature of ADC Clock Control block

ADC Clock Control block is responsible to handle the timing on storing the digital data in UADC_SREG. ADC Clock Control block contains user-configurable clock divider that allow user to slow down ADC controller unit in storing the digital data. Figure 5-4 shows the block interface of the ADC Clock Control block and Table 5-5 describes the function of each pin.

5.9.2 Block interface of ADC Clock Control block

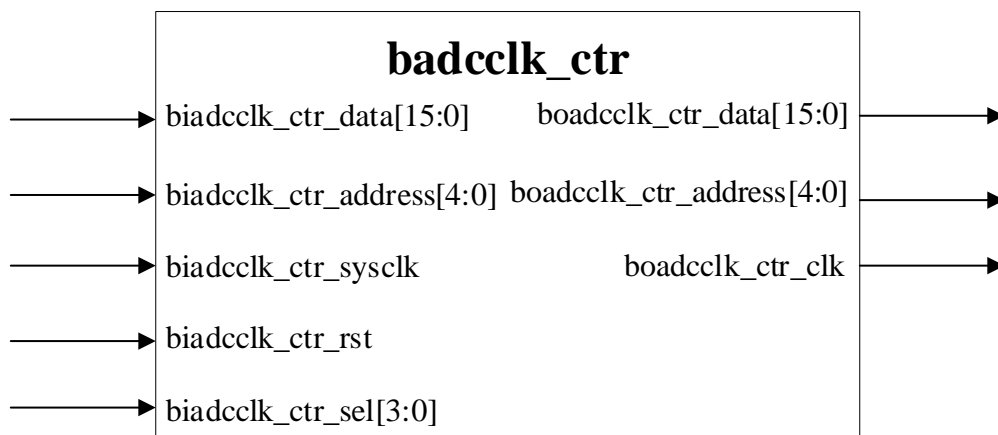


Figure 5-4: ADC Clock Control block interface

5.9.3 Input Pin Description of the ADC Clock Control block

Table 5-5: Input Pin description of the ADC Clock Control block.

Pin name: biadcclk_ctr_data[15:0]	Pin class: Data
Source → Destination: uadc → ADC Clock Control	
Pin function: Receive 16-bit digital data	
Pin name: biadcclk_ctr_address[4:0]	Pin class: Control
Source → Destination: uadc → ADC Clock Control	
Pin function: Address to store 16-bit data in SREG	
Pin name: biadcclk_ctr_sysclk	Pin class: Global
Source → Destination: uadc → ADC Clock Control	
Pin function: global clock	
Pin name: biadcclk_ctr_rst	Pin class: Global
Source → Destination: uadc → ADC Clock Control	
Pin function: Global reset	
1: reset	

0: no reset is required
Pin name: biadcclk_ctr_sel[3:0] Pin class: Control Source → Destination: uadc → ADC Clock Control Pin function: selectable 16-speed 0000: biclk_gen_sysclk / 2 0001: biclk_gen_sysclk / 4 0010: biclk_gen_sysclk / 8 0011: biclk_gen_sysclk / 16 1111: biclk_gen_sysclk / 65536

5.9.4 Output Pin Description of the ADC Clock Control block

Table 5-6: Output Pin description of the ADC Clock Control block.

Pin name: boadcclk_ctr_data[15:0] Pin class: Data Source → Destination: ADC Clock Control → uadc Pin function: Write 16-bit digital data to SREG
Pin name: boadcclk_ctr_address[4:0] Pin class: Control Source → Destination: ADC Clock Control → uadc Pin function: Address to store 16-bit data in SREG
Pin name: boadcclk_ctr_clk Pin class: Control Source → Destination: ADC Clock Control → uadc Pin function: Generate clock for data storage in SREG.

5.10 Finite State Machine of the ADC Controller Unit

5.10.1 Flags in FSM

In the ADC controller unit, a register mealy model FSM is built to control the data flow. Flags generated from address decoder is used to indicate which XADC register to be written to. Each flag signal is meant to write 16-bit data from the ADC controller unit to the XADC except the flag_UADC_READ which is for reading 16-bit data out from XADC. Since the Dynamic Reconfiguration Port (DRP) can only handle 16 bits of data at one clock cycle, a 32-bit data from ADC controller unit need to break into two 16-bit before it can be written into the XADC through DRP. Hence, each configuration of XADC registers will become one of the states in FSM and execute one by one. Same goes to the reading operation on XADC registers. All the configuration states will be executed first before the reading state. There is priority level within the flag registers as shown in Figure 5-5. The flag name is corresponding to the control registers in XADC, indicating which XADC register will be configured in next state except the flag_UADC_READ which only indicate that read operation will be carried out in next state.

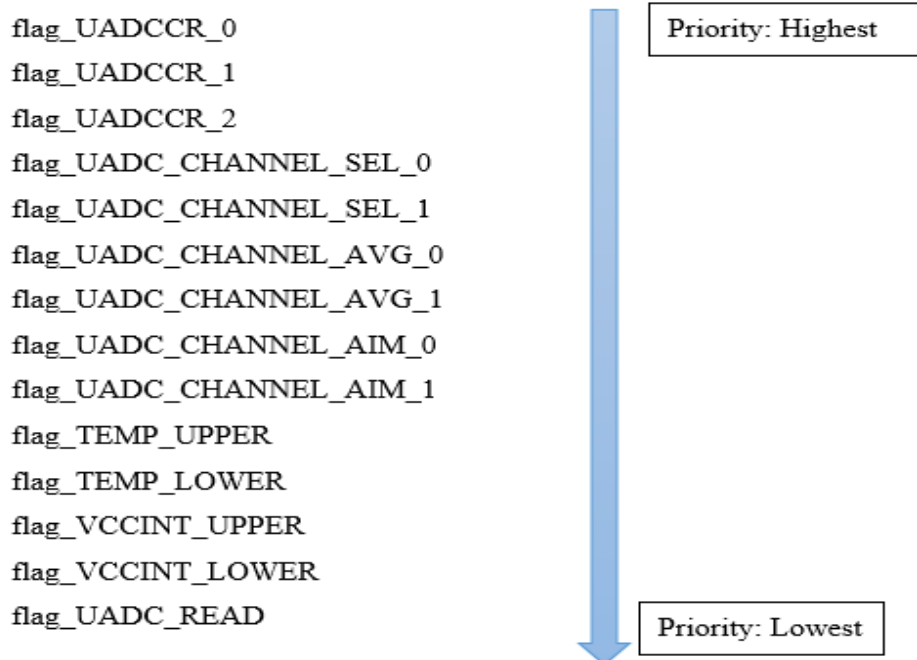


Figure 5-5: FSM flag register priority

Flag registers are purely combinational. The configuration flags like

- flag_UADCCR_0, flag_UADCCR_1, flag_UADCCR_2,
- flag_UADC_CHANNEL_SEL_0, flag_UADC_CHANNEL_SEL_1,
- flag_UADC_CHANNEL_AVG_0, flag_UADC_CHANNEL_AVG_1,
- flag_UADC_CHANNEL_AIM_0, flag_UADC_CHANNEL_AIM_1,
- flag_TEMP_UPPER, flag_TEMP_LOWER,
- flag_VCCINT_UPPER, flag_VCCINT_LOWER

will be raised whenever the wishbone writing wire is targeting the corresponding UADC_CREG. While flag_UADC_READ is dependent on the End of Conversion (EOC) signal output from XADC when there is a new analog signal converted. The mechanism of flag in FSM is further discussed in internal operation section. Figure 5-6 shows the schematic diagram.

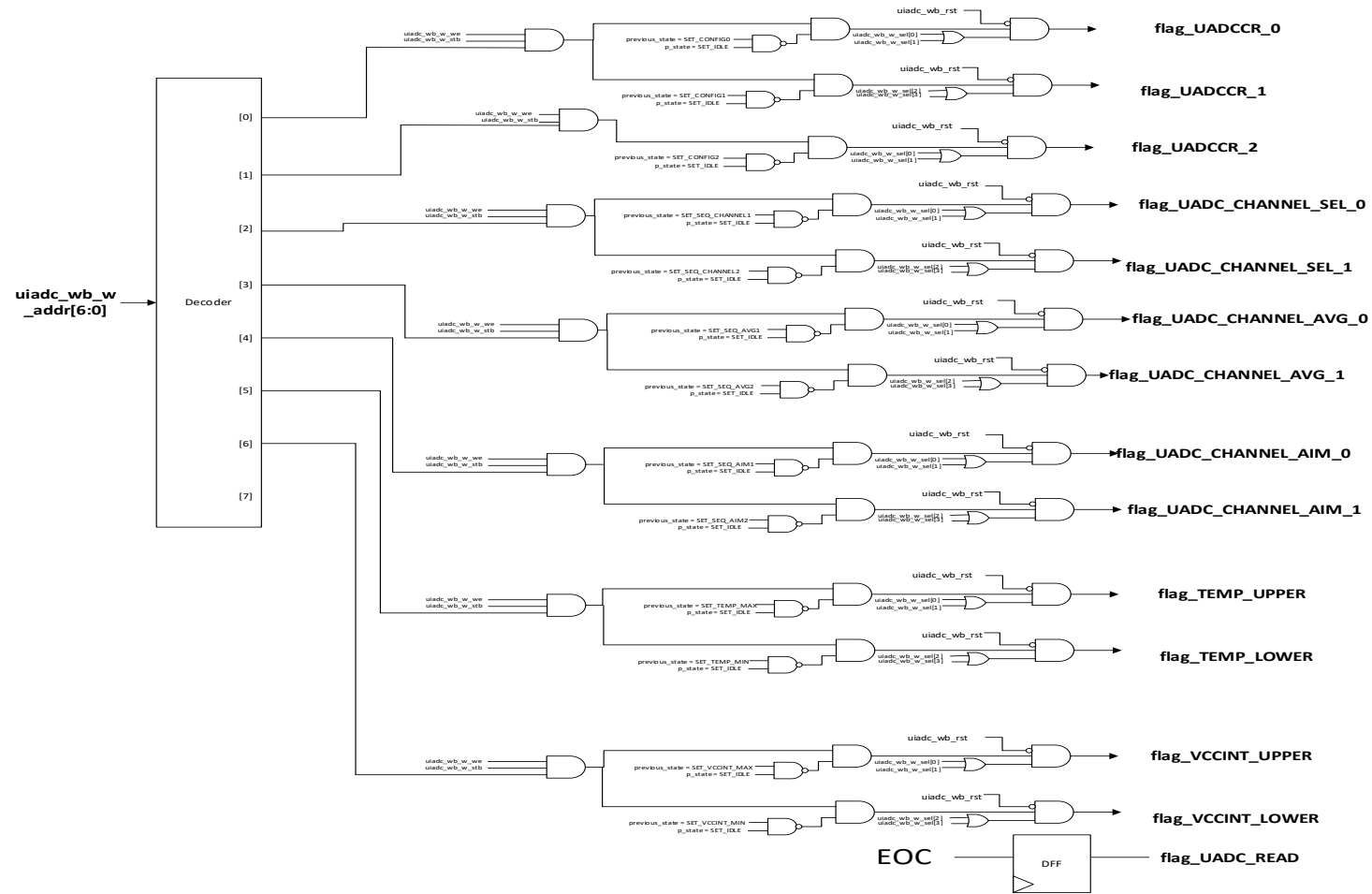


Figure 5-6 Schematic diagram for the flag registers

5.10.2 Internal Operation: UADC to XADC

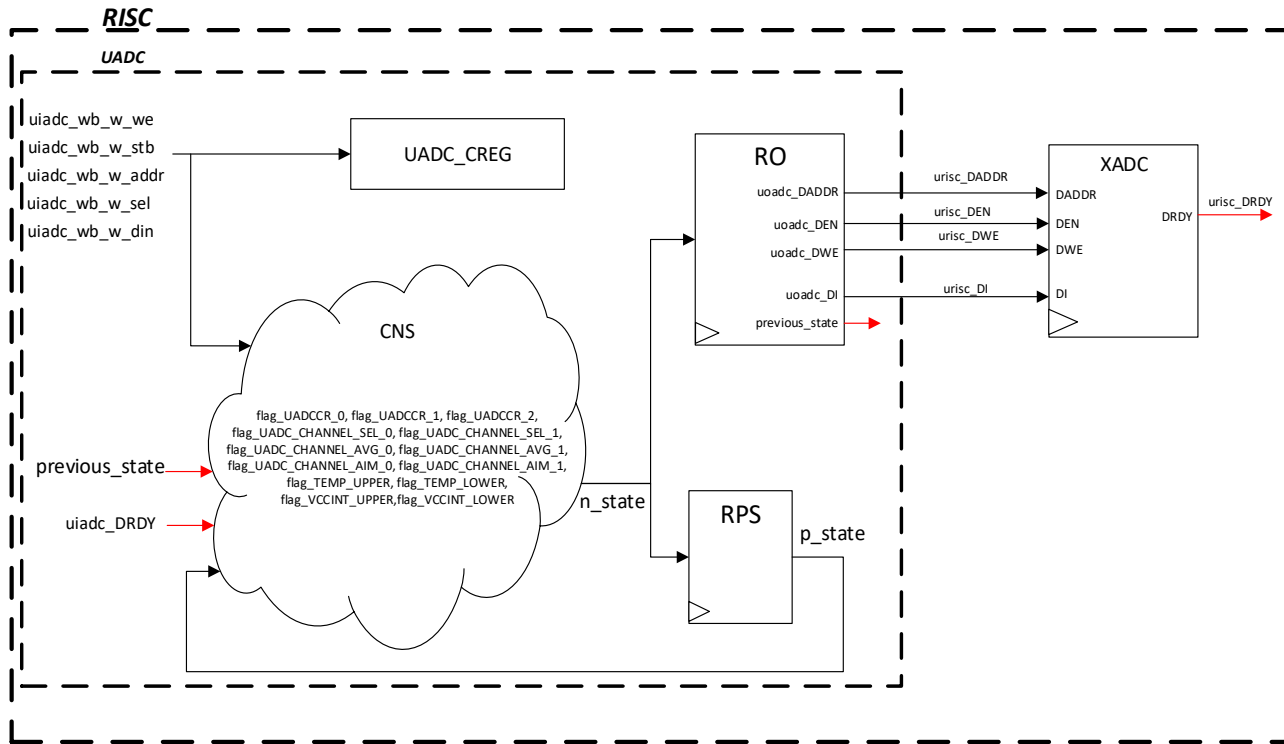


Figure 5-7: Schematic diagram for FSM writing operation

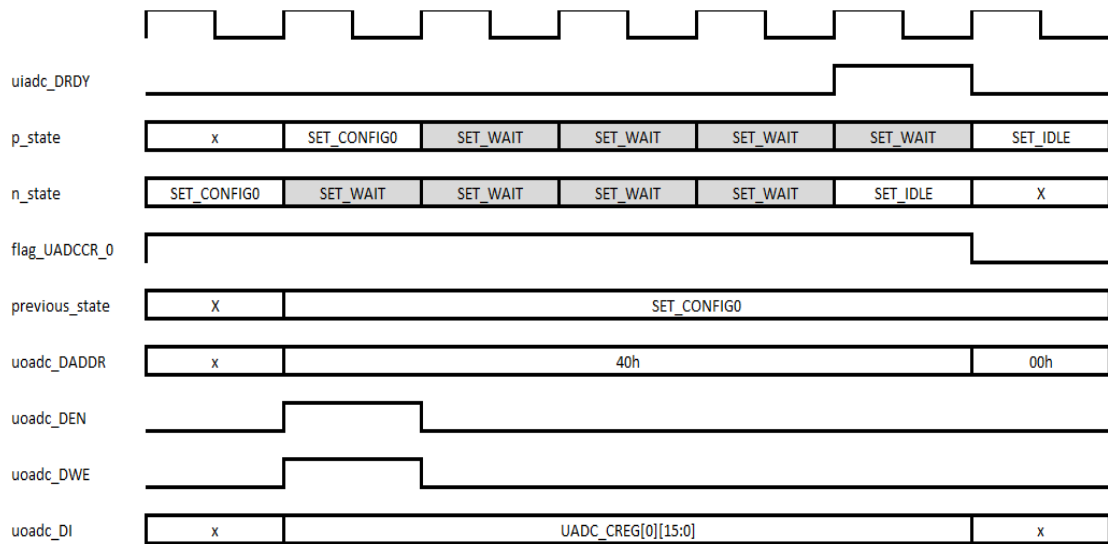


Figure 5-8: Timing diagram for FSM writing operation.

Figure 5-8 shows an example on how FSM is functioning during the configuration of XADC registers, CONFIGURE_REG_0. When the flag is raised due to the wishbone writing input, the next clock cycle will send enable(uoadc_DEN) and write enable (uoadc_DWE) signal to the XADC, to write the data in UADC_CREG [0] [15:0] to the configuration register at address 40h. The SET_WAIT state is required to wait the uiadc_DRDY signal, that indicate data has successfully written into XADC. After that, FSM will back to SET_IDLE state and jump to the next state according to flag register. While the use of previous state is to determine which flag to lower down after the uiadc_DRDY signal. In fact, it is the Dynamic Reconfiguration Port (DRP) in XADC that receive the uoadc_DEN, uoadc_DWE, uoadc_DADDR, uoadc_DI and also send out the uiadc_DRDY signal. The detail of the DRP timing is attached in appendix A.

5.10.3 Internal Operation: XADC to UADC

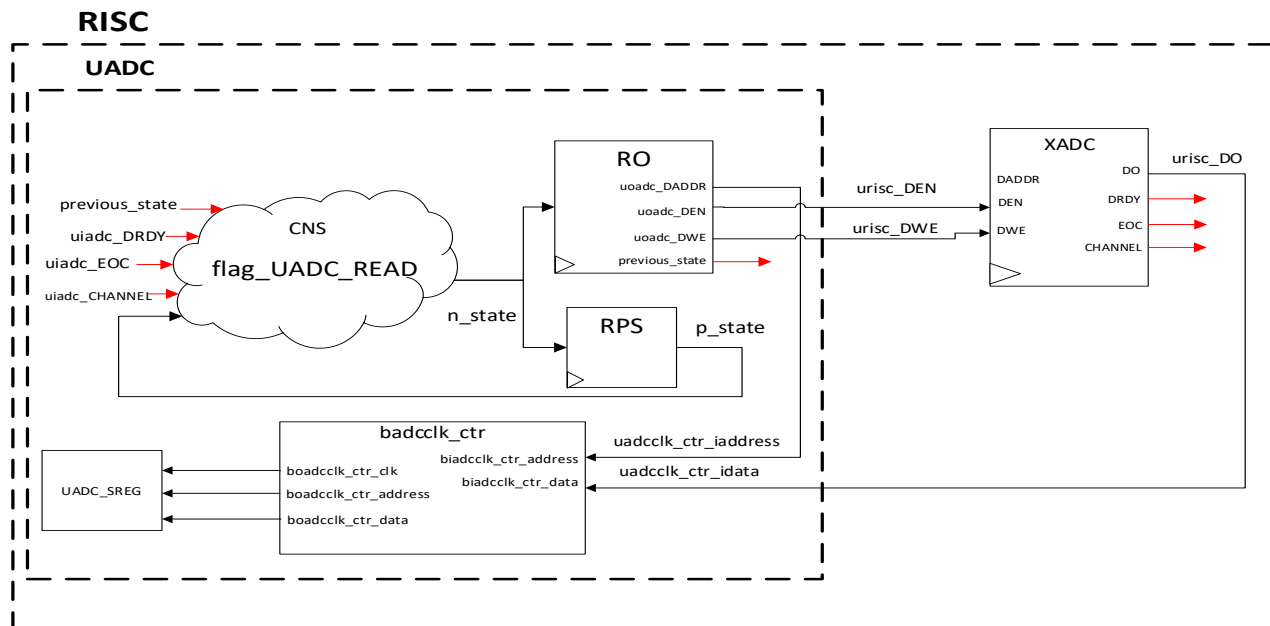


Figure 5-9: Schematic diagram for FSM reading operation.

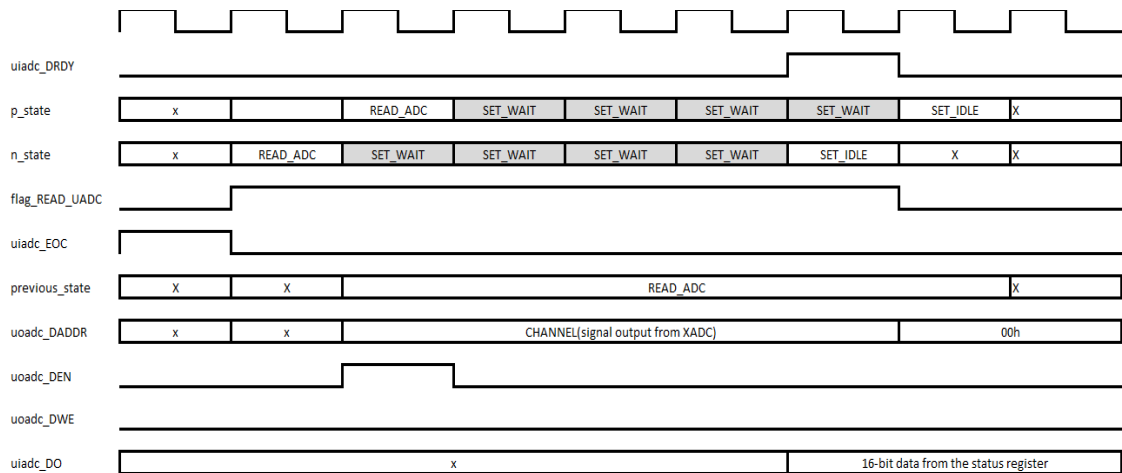


Figure 5-10: Timing diagram for FSM reading operation

The state READ_ADC is to continuously update the UADC_SREG in ADC controller unit whenever there is a new analog value converted. Therefore, flag_UADC_READ will be raised a clock cycle after the End of Conversion (uiadc_EOC) signal asserted, indicating next state will be READ_ADC. During the READ_ADC state, the uoadc_DEN need to be '1' and uoadc_DWE need to be '0' in order for XADC to carry out read operation. The register address(uoadc_DADDR) to read is the same as the CHANNEL output signal from XADC which indicating the address of status register

that having the new value. While the `uiadc_DRDY` in this case will indicate the data has been successfully read out from XADC status register and store temporarily in ADC Clock Control block. The detail can be found in DRP timing attached in appendix A.

5.11 Register Set

UADC_CREG and UADC_SREG are designed to have similar function with the control register and status register in XADC. Some important registers that are related to the simulation test in later section will be briefly discussed here. Complete and detail information about the register can always be found in Appendix.

UADC_CREG

1. CONFIGURE_REG_0 (0XBFFFFE2C)

z	z	AVG1	AVG0	z	BU	z	z	z	z	z	CH4	CH3	CH2	CH1	CH0
CONFIGURE_REG_0															

- CH4 to CH0: When operating in single channel mode, these bits are used to select the ADC input channel. Channel assignments is shown in Appendix A.
- BU: When operating in single channel mode, this bit is used to select either unipolar or bipolar data format of the converted data. Logic 1 for bipolar mode and logic 0 for unipolar mode.
- AVG1, AVG0: These bits are used to select the number of samples needed for averaging operation on selected channel. Averaging function in both single channel and sequence modes will depend on these bits. The bit assignments are shown in Appendix A.

2. CONFIGURE_REG_1 (0XBFFFFE2E)

SEQ3	SEQ2	SEQ1	SEQ0	ALM_EN6	ALM_EN5	ALM_EN4	ALM_EN3	z	z	z	z	ALM_EN2	ALM_EN1	ALM_EN0	z
CONFIGURE_REG_1															

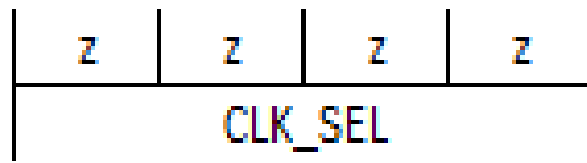
- ALM_EN6 to ALM_EN0: These bits are used to determine which of the channels, V_{CCO_DDR} , V_{CCPAUX} , V_{CCPINT} , V_{CCBRAM} , V_{CCAUX} , V_{CCINT} , temperature can trigger the alarm signal. Logic 1 will disable the alarm while logic 0 will enable it.
- SEQ3 – SEQ0: These bits will determine in which mode will XADC operate. There are six modes in total. The bit assignments are shown in Appendix A.

3. ALM_FLAG (0XBFFFFE32)

z	z	z	z	z	z	z	z
ALM_FLAG							

- These bits used to indicate which ALM has been triggered in XADC. It is supposed to be in UADC_SREG. Normally, MSB is used to use to detect the occurrence of alarm because it is the OR logic for the other 7 bits. The other bits are assigned to indicate different power supply interrupt. The bits assignments are shown in Appendix A.

4. CLK_SEL – ADC Receiver Clock Rate (CPU clock speed is 20MHz)



- 0000: 10 MHz
- 0001: 5 MHz
- ...
- 1110: 610 Hz
- 1111: 305 Hz

- 5. CHANNEL_SEL_REG_0, CHANNEL_SEL_REG_1
 CHANNEL_AVG_REG_0, CHANNEL_AVG_REG_1
 CHANNEL_AIM_REG_0, CHANNEL_AVG_REG_1 (0XBFFFE34 – 3F)

Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z
CHANNEL_SEL_REG_1													CHANNEL_SEL_REG_0														
Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z
CHANNEL_AVG_REG_1													CHANNEL_AVG_REG_0														
Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z
CHANNEL_AIM_REG_1													CHANNEL_AIM_REG_0														

- These registers are used during sequencer mode, they are used to select channel, enable averaging, and select analog input mode respectively. The bit assignments are shown in Appendix A.

6. TEMP_MAX_THRESHOLD, TEMP_MIN_THRESHOLD
 VCCINT_MAX_THRESHOLD, VCCINT_MIN_THRESHOLD
 (0XBFFFFE40 – 47)

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
TEMP_MIN_THRESHOLD													TEMP_MAX_THRESHOLD																
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Vccint_MIN_THRESHOLD													Vccint_MAX_THRESHOLD																

- These registers are used to store the maximum and minimum threshold value for temperature and V_{CCINT} sensor.
- For temperature, whenever the analog value exceeds the maximum threshold, it will trigger alarm, until the analog value is lower than the minimum threshold.
- While for V_{CCINT}, it will trigger alarm only when it exceeds the maximum or lower than the minimum threshold.
- The 12-bit data stored in these registers need to be left aligned. The calculation is discussed in following section. Appendix A attached has the formulas for calculation.

UADC_SREG

1. UADC_SREG_0 - 31 (0XBFFFFE4C – 8B)

z	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z
UADC_SREG_0															
z	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z
UADC_SREG_1															
z	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z
UADC_SREG_2															
z	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z
UADC_SREG_6															
z	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z
UADC_SREG_16															
z	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z
UADC_SREG_17															
z	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z
UADC_SREG_18															
z	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z
UADC_SREG_19															

- UADC_SREG is used to store 12-bit converted value from various sensors. The 12-bit is left aligned in the register.
- These eight highlighted registers will store Temperature, V_{CCINT} , V_{CCAUX} , V_{CCBRAM} , and $V_{AUXP/N}$ [3:0] respectively. They will store the simulation result. Appendix A attached has the formula for calculation.

Chapter 6: Firmware Development

6.1 Exception Handler of the RISC32 Pipeline Processor

Exception can also be known as an interrupt. It is not scheduled in the normal flow of instruction execution. Whenever the exception occurs, it will disrupt the flow of instructions. Besides, the causes for exception to occur can be internal or external of the processor which makes it even unpredictable. In order to handle the unexpected events, an exception handler has already been created. The exception handler is like a routine that define how an interrupt can be served. The exception handler in current processor is able to detect and handle unexpected events such as:

- Interrupt (from IO device)
- address error trap on data load or instruction fetch
- address error trap on data store
- bus error on instruction fetch
- bus error on data load or store
- Syscall trap
- breakpoint trap
- undefined instruction trap
- arithmetic overflow trap

When the CPU detect the exception signal, it will suspend its current program execution. First, the address of the next instruction of PC will be store in a return register called \$sepc for return purpose, and then PC will jump to 0x8001_b400 where exception handler is located. After finishing the interrupt service routine, PC will return to the address stored in return register, where it was interrupted and continue with the normal program execution as mentioned in [9].

A pseudocode that describes the existing exception handler of the RISC32 pipeline processor is given in figure 6-1 below for better understanding.

```

BEGIN
    Push the current state of the user program to stack
    Push the current CP0's status register value to stack
    Push the current CP0's cause register value to stack
    Clear the exception level in the CP0's status register
    Decode the exception code in the CP0's cause register
    CASEOF exception code
        0: Branch to the exception routine of Interrupt
        4: Branch to the exception routine of Address Error Trap LOAD
        5: Branch to the exception routine of Address Error Tap STORE
        6: Branch to the exception routine of Bus Error on IF Trap
        7: Branch to the exception routine of Bus Error on LOAD/STORE Tap
        8: Branch to the exception routine of Syscall
        9: Branch to the exception routine of Breakpoint Trap
        10: Branch to the exception routine of Reserved/Undefined Instruction
        12: Branch to the exception routine of Arithmetic Overflow
    ENDCASE
    Set the exception level in the CP0's status register
    Pop the previous state of user program from stack
    Clear the exception code in the CP0's cause register
    Clear the interrupt priority level in the CP0's cause register
    Return to user program based on the address value in the CP0's EPC register
END

```

Figure 6-1: Pseudocode describing exception handler in RISC32 as mentioned in [9].

6.2 Interrupt Service Routine (ISR) of the ADC Controller Unit

An interrupt is an external event from IO device that interrupts the CPU to inform it that a device needs its service. In this project, the device that interrupts the CPU will be the ADC controller unit. Since interrupt is asynchronous to the program execution, the CPU will simply suspend the normal instruction execution and proceed to execute the corresponding Interrupt Service Routine (ISR). When CPU has finished the routine, it will resume to the place where it was interrupted.

In this project, an ISR specifically for the ADC controller unit is developed by using MIPS assembly language and subsequently integrated into the existing exception handler. This ISR will be invoked by the CPU to handle the interrupt request generated by the ADC controller unit. The interrupt request will be based on the alarm signal generated from XADC's logic outputs namely ALM [7:0]. The XADC are able to monitor up to six internal sensor measurements (Temperature, V_{CCINT} , V_{CCAUX} , V_{CCBRAM} , V_{CCPINT} , V_{CCPAUX} , V_{CCO_DDR}). The MSB ALM [7] is the logic OR of bus ALM

[6:0], and it is used in the ADC controller unit with the `uiadc_ADCIE`(ADC interrupt enable) to generate the Interrupt Request Signal (`uoadc_IRQ`). As mentioned in the previous section, only the Temperature and V_{CCINT} alarm threshold register are implemented. Therefore, in this project, only the Temperature and V_{CCINT} sensors threshold value can be configured in XADC which is enough to observe the simulation result and verify the functionality.

Besides, a simple Interrupt Service Routine is designed for the ADC Controller unit to carry out certain action. In the routine, the action is simply enabling the corresponding GPIO pin. For example, if it is Temperature sensor (ALM [0]) that trigger the Interrupt Request Signal, `GPIOEN [0]` will be set to '1'. If it is V_{CCINT} (ALM [1]), then `GPIOEN [1]` will be set.

Hence, user could apply this function in some home automation project, like enabling the cooling system when the temperature is too high or enable the siren whenever the sensor's value fall in undesired range.

Figure 6-2 below shows the pseudocode of the developed ISR for handling interrupt request from the ADC controller unit. In this case, Each GPIO is assign respectively to the six internal sensor measurements (Temperature, V_{CCINT} , V_{CCAUX} , V_{CCBRAM} , V_{CCPINT} , V_{CCPAUX} , V_{CCO_DDR}).

```
BEGIN
  Load the ALM_FLAG value
  CASEOF ALM_FLAG
    0: Set GPIOEN [0]
    1: Set GPIOEN [1]
    2: Set GPIOEN [2]
    3: Set GPIOEN [3]
    4: Set GPIOEN [4]
    5: Set GPIOEN [5]
    6: Set GPIOEN [6]
  ENDCASE
  Return to the main exception handler
END
```

Figure 6-2: Pseudocode describing UADC's ISR.

Chapter 7: Verification Specification and Simulation Result

In this chapter, there are two simulations carried out. First simulation is for the ADC Controller unit only. The microarchitecture is the older version in which XADC is instantiated in the UADC. As mentioned in the Chapter 3 methodologies, this simulation is needed to test out the application of XADC and validate the FSM of UADC before integration with the RISC32. Hence, it only tests out 4 basic functions, which are reset, read, write, and interrupt. While the second simulation is for the final version of microarchitecture which is after the integration with RISC32. This simulation will be using MIPS assembly code.

The simulation test in this project will use the sample stimulus text file provided by Xilinx with some value altered to suit the test plan. The text file is emulating the real time analog input. There are eight analog input use in this project to carry out functionality test as shown in the figure 7-1.

```
// This analog stimulus file is used to inject analog signals (e.g., volts, temperature) for a simulation.
// Units are as follows:
//      Time:                nanoseconds [ns]
//      Voltage (All rails):  volts [V]
//      Temperature:         degrees C [C]. Please note that the temperature transfer function is in terms of Kelvin
//
// In this example the VCCAUX supply moves outside the 1.89V upper alarm limit at 67 us
// An alarm is generate when the VCCAUX channel is sampled and converted by the ADC
/////////only AUX can change AIM(max is 0.5 for bipolar 1.0 for unipolar)//100us interrupt occur
TIME VAUXP[0] VAUXN[0] VAUXP[1] VAUXN[1] VAUXP[2] VAUXN[2] VAUXP[3] VAUXN[3] TEMP VCCINT VCCAUX VCCBRAM
00000 0.005 0.0 0.2 0.0 0.23 0.0 0.1 0.0 25 1.0 1.8 1.0
67000 0.020 0.0 0.400 0.0 0.40 0.0 0.2 0.0 85 1.05 1.9 1.05
100000 0.049 0.0 0.36 0.0 0.45 0.0 0.23 0.0 105 0.9 1.71 0.95
140000 0.034 0.0 0.900 0.0 0.53 0.0 0.0 0.0 70 1.00 1.8 1.0
167000 0.034 0.0 0.900 0.0 0.53 0.0 -0.034 0.0 70 1.00 1.8 1.0
```

Figure 7-1: Stimulus text file extracted from [15] with some value altered.

7.1 Unit Level Functional Test Plan

Test	Test Vector/Input	Expected Output	Result
Test case 1: System Reset Test Function/Description: -To test if ADC controller unit can be reset. -Hold the reset signal for one clock cycle.	tb_ip_wb_clk <= 1'b1;	UADC_CREG contains the initial value just the same as XADC. UADC_SREG is clear to 0.	Pass

<p>Test case 2: Read operation</p> <p>Test Function/Description: -To test if the UADC_SREG can be read out from ADC controller unit. -The reading operation occur at the time after 167us in which analog input does not change anymore.</p>	<p><i>[constant]</i></p> <pre>tb_ip_wb_r_we <=1'b0; tb_ip_wb_r_stb <=1'b1; tb_ip_wb_r_sel <= 4'b1111;</pre>		
<p>i) Read Temp and Vccint value.</p>	<pre>tb_ip_wb_r_addr <= 6'b001_000;</pre>	<pre>tb_op_wb_r_dout = 32'h 5555_ae4e</pre>	Pass
<p>ii) Read Vccaux value.</p>	<pre>tb_ip_wb_r_addr <= 6'b001_001;</pre>	<pre>tb_op_wb_r_dout = 32'h 0000_9999</pre>	Pass
<p>iii) Read Vccbram value.</p>	<pre>tb_ip_wb_r_addr <= 6'b001_011;</pre>	<pre>tb_op_wb_r_dout = 32'h 0000_5555</pre>	Pass
<p>iv) Read VAUXP/N[1] and VAUXP/N[0] value.</p>	<pre>tb_ip_wb_r_addr <= 6'b010_000;</pre>	<pre>tb_op_wb_r_dout = 32'h e666_08b4</pre>	Pass
<p>v) Read VAUXP/N[3] and VAUXP/N[2] value.</p>	<pre>tb_ip_wb_r_addr <= 6'b010_001;</pre>	<pre>tb_op_wb_r_dout = 32'h 0000_87ae</pre>	Pass
<p>Test case 3: Write operation</p> <p>Test Function/Description: -To test if the data can write into the UADC_CREG and the operating mode of XADC could be changed.</p>	<p><i>[constant]</i></p> <pre>tb_ip_wb_w_we <=1'b1; tb_ip_wb_w_stb <= 1'b1;</pre>		
<p>i) Enable all channel to do average of 16 sample</p>	<pre>tb_ip_wb_w_din <= 32'h000f_4700;</pre>	<p>XADC will only send out the End of Conversion (EOC)</p>	Pass

<p>(Averaging sample is 16 as in initialization) -Write to channel_avg_reg_0 and channel_avg_reg_1.</p>	<pre>tb_ip_wb_w_sel <= 4'b1111; tb_ip_wb_w_addr <= 6'b000_011;</pre>	<p>signal after the channel has finished averaging 16 samples which will then update the UADC_SREG.</p>	
<p>ii)Switch to single channel operating mode with only temperature is sampled. -Write to configure_reg_1, changing the (SEQ3 – 0) to 4'b0011. -default value on (CH4-CH0) is 5'b0 which select the temperature sensor channel to sample.</p>	<pre>tb_ip_wb_w_din <= 32'h0000_3ef0; tb_ip_wb_w_sel <= 4'b1100; tb_ip_wb_w_addr <= 6'b000_000;</pre>	<p>XADC will only sample the temperature sensor value.</p>	<p>Pass</p>
<p>iii)Configure AUXP/N [3:0] to bipolar analog input mode. -Write to channel_aim_reg_0 and channel_aim_reg_1 -Only AUXP/N can switch Analog-Input Mode. Bipolar range: $-0.5 \leq V \leq 0.5$ Unipolar range: $0 \leq V \leq 1.0$</p>	<pre>tb_ip_wb_w_din <= 32'h000f_0000. tb_ip_wb_w_sel <= 4'b1111. tb_ip_wb_w_addr <= 6'b000_100;</pre>	<p>When XADC operate in bipolar mode, the maximum voltage value that an AUXP/N can go is 0.5V instead of 1.0V and minimum voltage value is -0.5V instead of 0V.</p> <p>Around 140us, VAUXP/N [1] and VAUXP/N [2] will have a digital value capped at 0x7ff. (0.5V) when they have value 0.9V and 0.53V.</p> <p>Around 167us, VAUXP/N [3] will have a digital value 0xf74 that indicating a negative</p>	<p>Pass</p>

		voltage value (-0.034V). Previously in Test case 1, the value is 0x000 due to the unipolar analog input mode.	
Test case 4: Interrupt operation Test Function/Description: -to test if ADC controller unit can send out Interrupt request signal, when there is alarm signal generated from XADC.	<i>[constant]</i> tb_ip_ADCIE<=1'b1; tb_ip_wb_w_we <=1'b1; tb_ip_wb_w_stb <= 1'b1;		
i) Write to configure_reg_1 to enable only temp and Vccint alarm signal. -The initialize threshold value for temp and Vccint is $60^{\circ}\text{C} \leq \text{temp} \leq 85^{\circ}\text{C}$ $0.95\text{V} \leq \text{Vccint} \leq 1.05\text{V}$	tb_ip_wb_w_din <= 32'h0000_2ff8; tb_ip_wb_w_sel <= 4'b1100; tb_ip_wb_w_addr <= 6'b000_000;	Around 100us, uoadc_IRQ = 1'b1; and both alarm for temperature and Vccint is asserted. Around 140us, alarm for Vccint will be lower but alarm for temperature will stay high with 70°C	Pass
ii) Write to temp_min_threshold to change the lower threshold value become 70 °C	tb_ip_wb_w_din <= 32'h0000_ae50; tb_ip_wb_w_sel <= 4'b1100; tb_ip_wb_w_addr <= 6'b000_101;	Around 140us, uoadc_IRQ = 1'b0; and both alarm for temperature and Vccint is lower.	Pass
iii) Disable interrupt function of ADC controller unit.	tb_ip_ADCIE<=1'b0;	uoadc_IRQ will be '0', although there is the presence of alarm signal.	Pass

7.2 Simulation Result for Unit Level Functional Test

To prove that the simulation result shown in waveform is correct, the analog value in stimulus file is converted using the formulas attached in Appendix. Table 7-1 shows the digital value for the stimulus file.

Table 7-1: digital value of stimulus file.

Time(us)	VAUXP [0]	VAUXP [1]	VAUXP [2]	VAUXP [3]	TEMP	VCCINT	VCCAUX	VCCBRAM
0	U: 0x014	U: 0x333	U: 0x3ae	U: 0x199	U: 0x977	U: 0x555	U: 0x999	U: 0x555
67	U: 0x051	U: 0x666	U: 0x666	U: 0x333	U: 0xb5e	U: 0x599	U: 0xa22	U: 0x599
100	U: 0x0c8	U: 0x5c2	U: 0x733	U: 0x3ae	U: 0xc01	U: 0x4cc	U: 0x91e	U: 0x511
140	U: 0x08b	U: 0xe66 B: 0x7ff	U: 0x87a B: 0x7ff	U: 0x000	U: 0xae4	U: 0x555	U: 0x999	U: 0x555
167	U: 0x08b	U: 0xe66 B: 0x7ff	U: 0x87a B: 0x7ff	U: 0x000 B: 0xf74	U: 0xae4	U: 0x555	U: 0x999	U: 0x555

U: unipolar, B: Bipolar

7.2.1 Test case 1: System Reset.

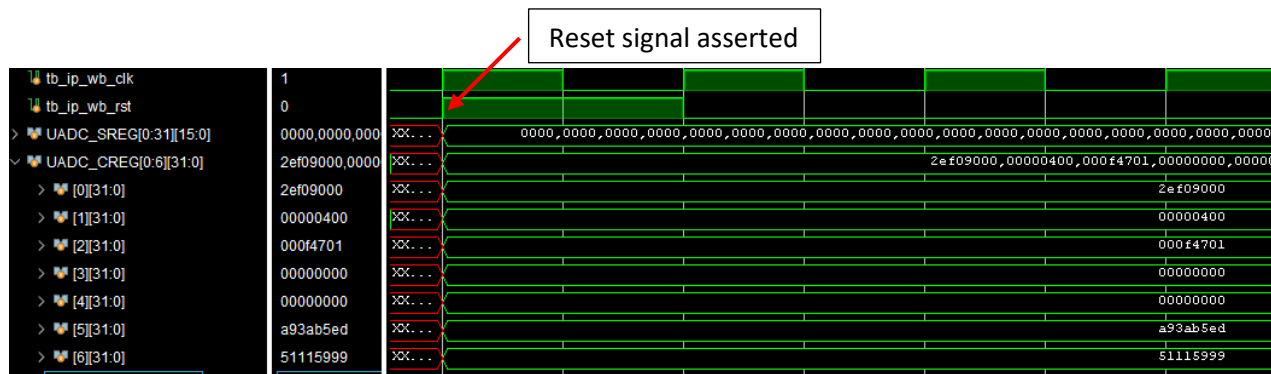


Figure 7-2: Simulation result for test case 1 using Vivado simulation tool.

- When the reset signal is asserted, the UADC_SREG will be clear to '0'
- While UADC_CREG will store the same value as the initialization made on XADC, just in case user did not do configuring at the beginning of program code.

7.2.2 Test case 2: Read Operation.

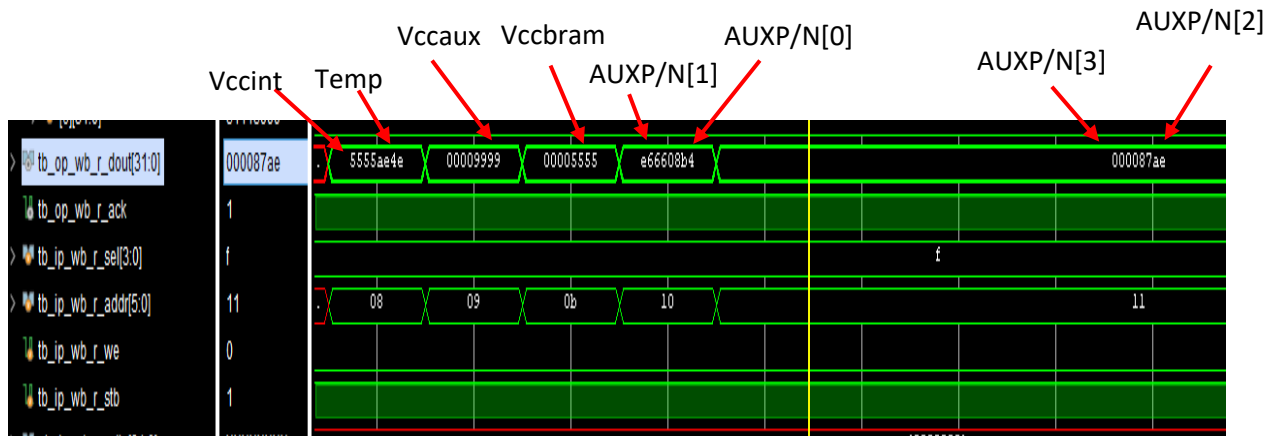
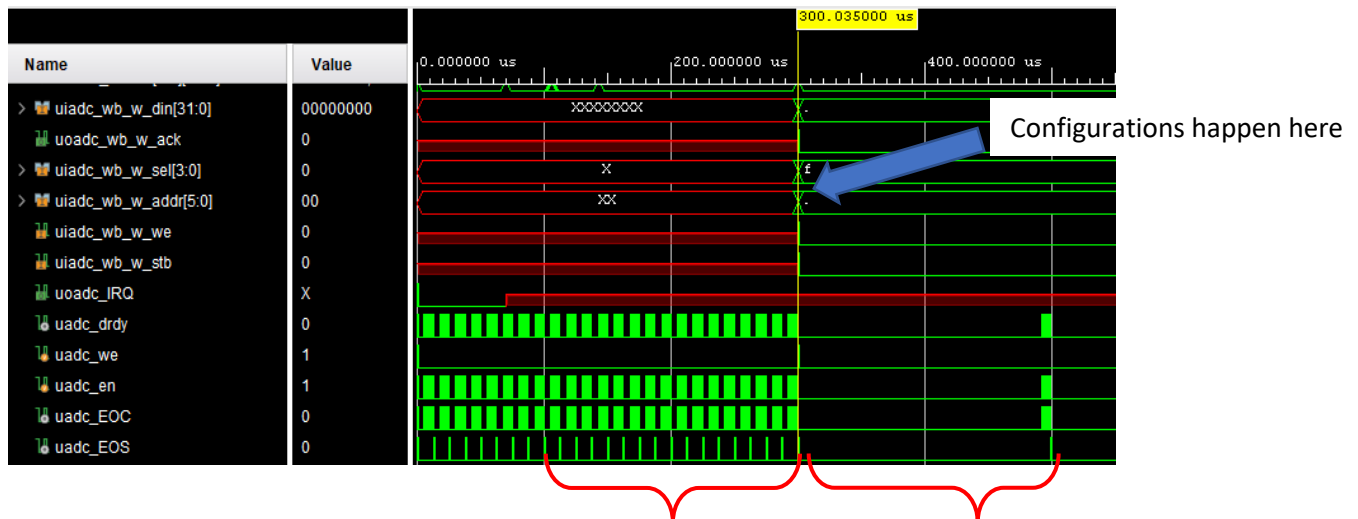


Figure 7-3: Simulation result for test case 2 using Vivado simulation tool.

- The value of Temperature, Vccint, Vccaux, Vccbram, VAUXP/N[3:0] is read out successfully. The values read out are the same as unipolar value shown in Table 7-1 when the analog stimulus has no more changes after 140us.

7.2.3 Test case 3: Write Operation.

- Enable all channel to do average 16 sample.



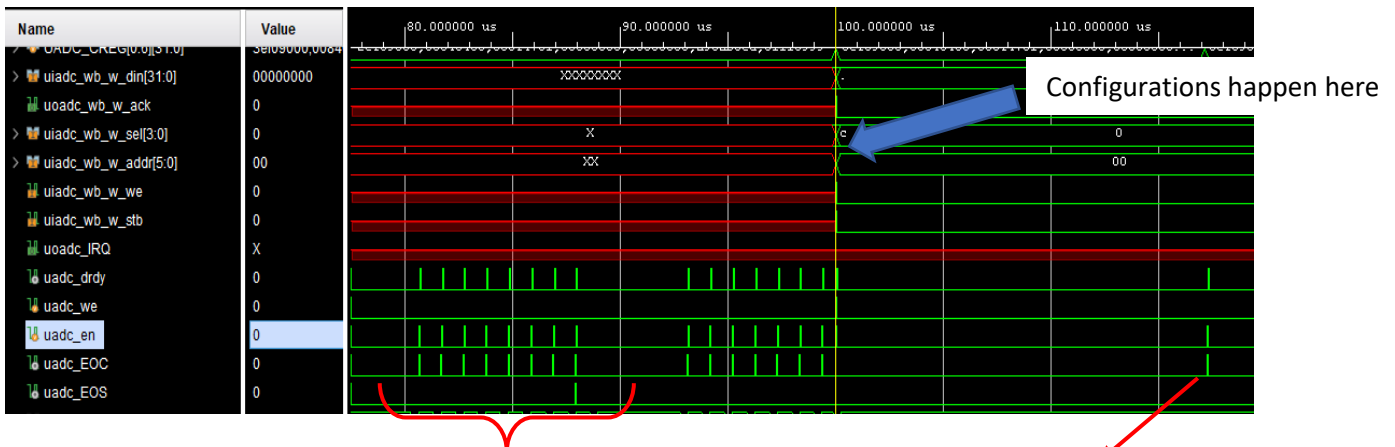
Before: All 16 samples are converted and read out.

After: Only the average of 16 samples value is converted and read out.

Figure 7-4: Simulation result for test case 3(i) using Vivado simulation tool.

- When the XADC has been configured to do 16 sample averaging for all channel, it happens to only EOC and EOS signal when all 16 sample has been used to obtain the average value as shown in figure 7-4.

ii) Switch to single channel mode with only temperature sampled.



Before: All 8 analog input are monitored in sequencer mode.

After: Only temperature sensor is monitored in single channel mode.

Figure 7-5: Simulation result for test case 3(ii) using Vivado simulation tool.

- After the XADC is configured to operate in single channel mode, only temperature sensor value is sampled instead of all eight sensors. Since the CH4-CH0(look for Appendix) has value of 5'b0, it indicates that temperature sensor is the only channel monitored in XADC.

iii) Configure AUXP/N [3:0] to bipolar input mode.

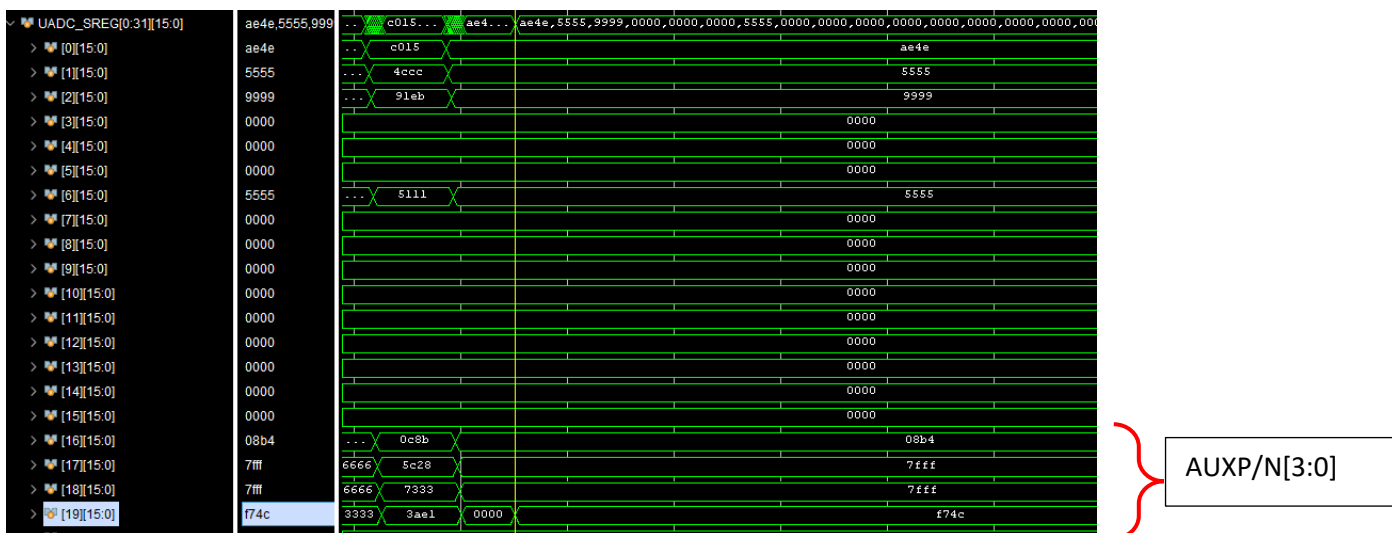


Figure 7-6: Simulation result for test case 3(iii) using Vivado simulation tool.

- After changing the analog input mode for AUXP/N [3:0] to bipolar mode. It now has the same value as shown in Table 7-1 during 167us.

7.2.4 Test case 4: Interrupt Operation.

i) Write to CONFIGURE_REG_1 to enable only temp and Vccint alarm signal.

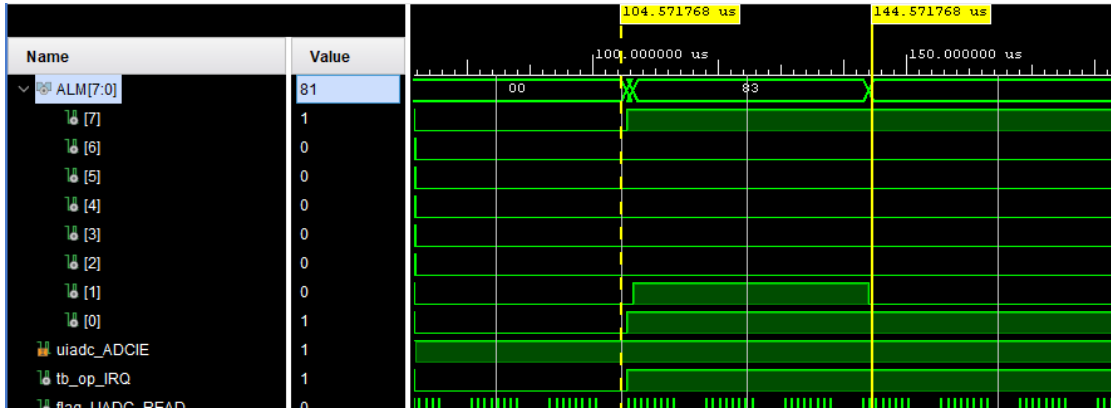


Figure 7-7: Simulation result for test case 4(i) using Vivado simulation tool.

- The initialize threshold value are like:

$$60^{\circ}\text{C} \leq \text{temp} \leq 85^{\circ}\text{C}$$

$$0.95\text{V} \leq \text{Vccint} \leq 1.05\text{V}$$
- Therefore, by referring to the stimulus file, alarm signal will be generated during 100us.
- During 140us, alarm signal for Vccint(ALM[1]) will deasserted for having 1.0 V, while the alarm signal for temperature(ALM[0]) will stay high for having 70°C.

ii) Write to TEMP_MIN_THRESHOLD to change the lower threshold value become 70 °C.

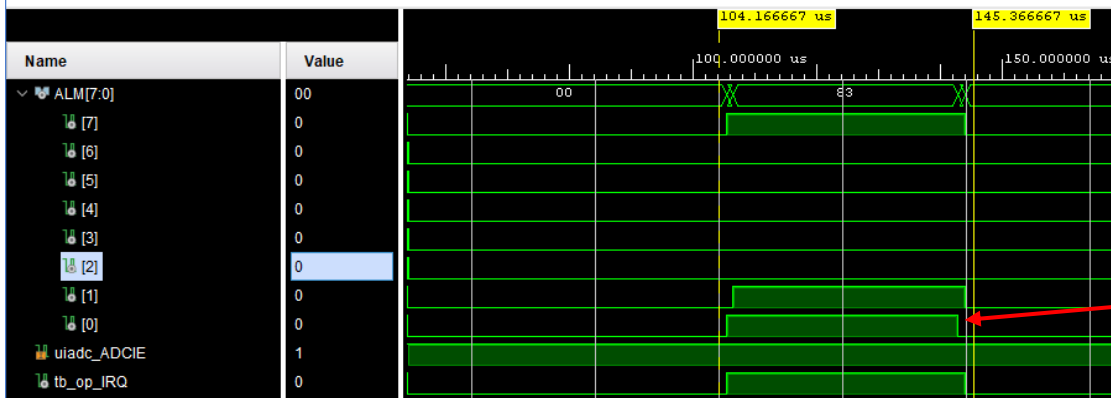


Figure 7-8: Simulation result for test case 4(ii) using Vivado simulation tool.

Chapter 7: Verification Specification and Simulation Result

- By changing the threshold to $70^{\circ}\text{C} \leq \text{temp} \leq 85^{\circ}\text{C}$, alarm signal (ALM[0]) for temperature will now be reset.

iii) Disable interrupt function of ADC controller unit.

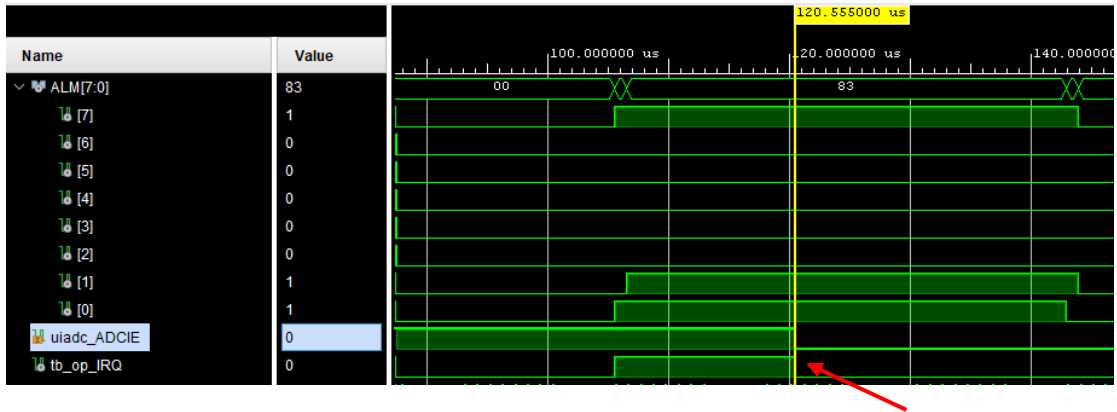


Figure7-9: Simulation result for test case 4(iii) using Vivado simulation tool.

- Interrupt request signal (tb_op_IRQ) will be '0' even there are presence of alarm signal due to the interrupt enable signal (uiadc_ADCIE) is reset.

7.3 Testbench for Integration Level Functional Test

```

`timescale 1ns / 1ps
`default_nettype none
define demo006_ADC
//define multiple_exception
`ifdef demo006_ADC
    `define TEST_CODE_PATH_DUT "demo006_ADC_v2mem_02program.hex"
    `define EXC_HANDLER_DUT "demo006_ADC_mem_03exc_handler.hex"
`endif

`ifdef multiple_exception
    `define TEST_CODE_PATH_DUT "Multiple_Exception+interrupt.hex"
    `define EXC_HANDLER_DUT "exception_handler.hex"
    `define TEST_CODE_PATH_CLIENT "Uart+Spi Transmit to DUT.hex"
    `define EXC_HANDLER_CLIENT "exception_handler.hex"
`endif

module tb_r32_pipeline();
//declaration
//===== INPUT =====
//System signal
reg  tb_u_clk;
reg  tb_u_rst;
//~~~~~
wire tb_u_spi_mosi_dut;
wire tb_u_spi_miso_dut;
wire tb_u_spi_sclk_dut;
wire tb_u_spi_ss_n_dut;

wire tb_u_fc_sclk_dut;
wire tb_u_fc_ss_dut;
wire tb_u_fc_MOSI_dut;
wire tb_u_fc_MISO1_dut;
wire tb_u_fc_MISO2_dut;
wire tb_u_fc_MISO3_dut;
wire tb_u_tx_rx_dut;
wire tb_ua_RTS_dut, tb_ua_CTS_dut;
wire[31:0] tb_u_GPIO_dut;
//~~~~~
wire tb_u_spi_mosi_client;
wire tb_u_spi_miso_client;
wire tb_u_spi_sclk_client;
wire tb_u_spi_ss_n_client;

wire tb_u_fc_sclk_client;
wire tb_u_fc_ss_client;
wire tb_u_fc_MOSI_client;
wire tb_u_fc_MISO1_client;
wire tb_u_fc_MISO2_client;
wire tb_u_fc_MISO3_client;
wire tb_u_tx_rx_client;
wire tb_ua_RTS_client, tb_ua_CTS_client;
wire[31:0] tb_u_GPIO_client;

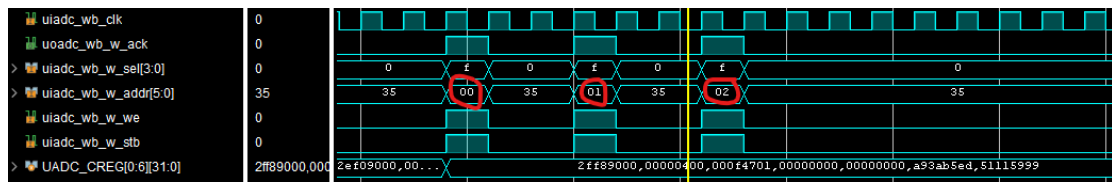
...
...
...

```

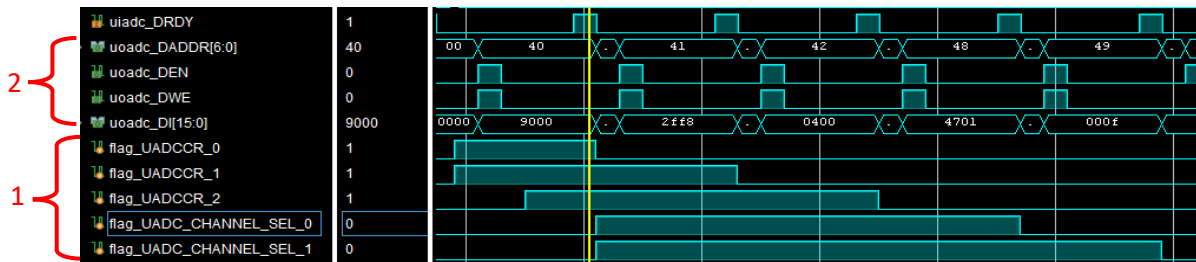
Full Testbench coding is in Appendix B

7.4 Simulation Result for Integration Level Functional Test

Test Case 1: Write Operation



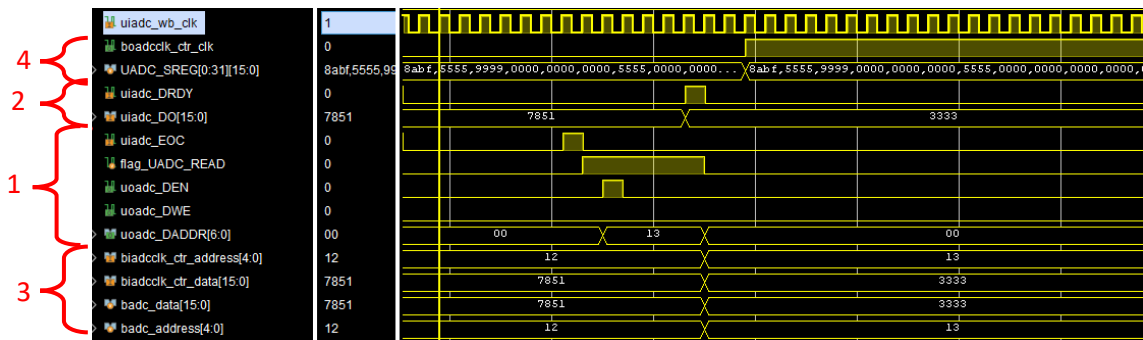
Wishbone input has targeted UADC IO register (BFFFFE2C – 34)



1) Flags that involve in the writing operation will be raised according to the wishbone input and be lowered after the data have successfully written into by monitoring the uiadc_DRDY.

2) uoadc_DEN and uoadc_DWE are raised for one clock cycle to signal XADC that writing operation is on demand. While uoadc_DI and uoadc_DADDR are having the corresponding data and address to be written in XADC.

Test Case 2: Read Operation



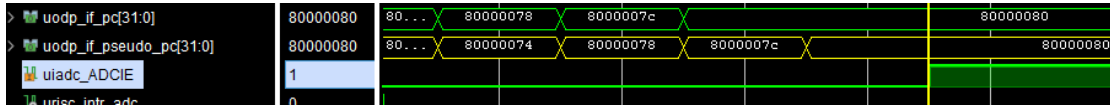
1) When uiadc_EOC asserted, the flag_UADC_READ will be raised in the next clock cycle to indicate that there is a new value converted in XADC and it is ready to be read. After One clock cycle the flag raised, uoadc_DEN and uoadc_DWE are having '1' and '0' value to signal XADC that reading operation is on demand, while uoadc_DADDR is having the address of register in XADC that store the data.

2) uiadc_DRDY is a signal given by XADC to indicate that the intended data is ready to be taken in uiadc_DO.

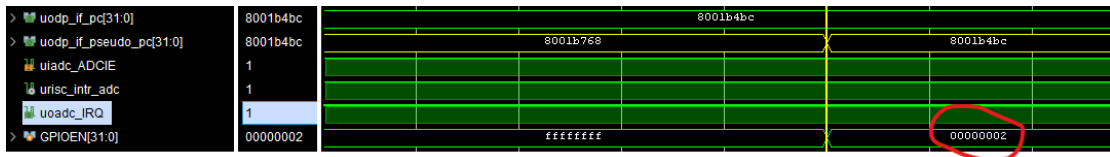
Chapter 7: Verification Specification and Simulation Result

- 3) The new data read is not stored directly in UADC_SREG. Instead, it is stored temporarily in ADC Clock Control block's register.
- 4) The new data is only stored in UADC_SREG on the rising edge of boadclk_ctr_clk.

Test Case 3: Interrupt

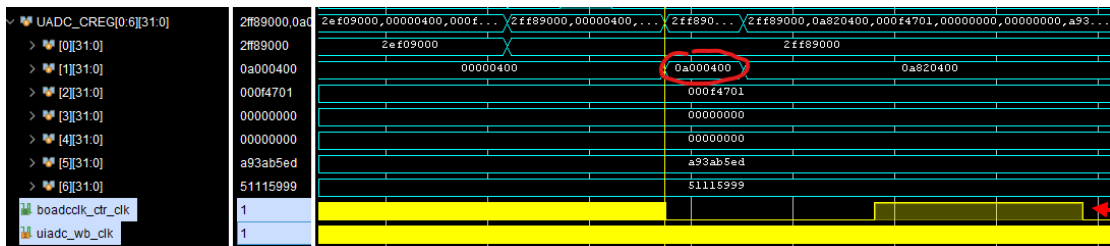


After configuration, uiadc_ADCIE has asserted.



In the ISR, GPIOEN [2] is asserted to indicate that Vccint trigger the interrupt.

Test Case 4: Clock Control for data storing speed



After configuration, CLK_SEL is having value 'a' and boadclk_ctr_clk has decreased in frequency.

Chapter 8: Multiple IO System Functional Test

In previous chapter, we have tested out the individual interrupt test of ADC. However, before we can say the integration of ADC Controller unit is completed, we must make sure the ADC exception event would not clash with the other IO unit's exception and internal exception (trap). Hence, in this chapter, we will develop a test case in which multiple external interrupt and trap are triggered in a single program. The expected output is each exception event will be served, with their respective ISR being executed.

8.1 Test Case: Multiple interrupt and Multiple Trap.

In this test case, all three IO SPI, UART, and ADC interrupts are enabled. To trigger SPI and UART interrupt, connection between Server (DUT) and Client is needed. By connecting both, Client will keep sending data to Server (DUT) through SPI and UART to trigger interrupt in the Server (DUT) side. The connection is established as shown in Figure 8-1.

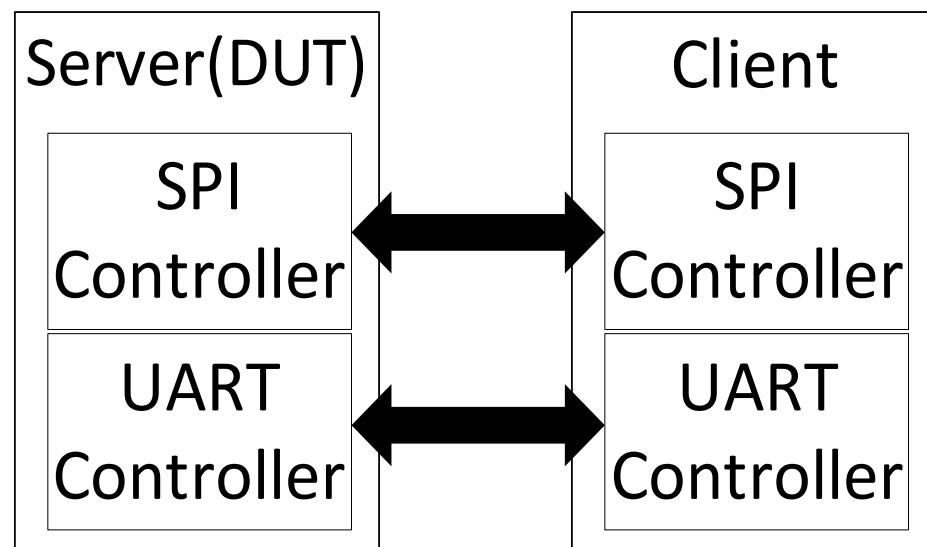


Figure 8-1: Connection between Server (DUT) and Client as shown in [5]

While for ADC interrupt, it is triggered by the power supply V_{CCINT} whenever the value exceeds 1.05V. Figure 8-2 shows roughly when the interrupt will be triggered.

10	TIME	VAUXP[0]	VAUXN[0]	VAUXP[1]	VAUXN[1]	VAUXP[2]	VAUXN[2]	VAUXP[3]	VAUXN[3]	TEMP	VCCINT	VCCAUX	VCCBRAM
11	000000	0.005	0.0	0.2	0.0	0.5	0.0	0.1	0.0	25	1.0	1.8	1.0
12	300000	0.020	0.0	0.400	0.0	0.49	0.0	0.2	0.0	85	1.05	1.9	1.05
13	350000	0.049	0.0	0.600	0.0	0.51	0.0	0.5	0.0	105	0.95	1.71	0.95
14	400000	0.034	0.0	0.900	0.0	0.53	0.0	0.0	0.0	0	1.00	1.8	1.0
15	2338550	0.020	0.0	0.500	0.0	0.47	0.0	0.2	0.0	34	1.10	1.8	1.0
16	2438550	0.020	0.0	0.500	0.0	0.47	0.0	0.2	0.0	34	0.99	1.8	1.0
17	2650000	0.020	0.0	0.500	0.0	0.47	0.0	0.2	0.0	34	1.10	1.8	1.0
18	2800000	0.020	0.0	0.500	0.0	0.47	0.0	0.2	0.0	34	0.99	1.8	1.0

Figure 8-2: The time when the interrupt happens for ADC.

At the same time, multiple trap occurrences have intentionally programmed at the server (DUT) side to see if there were any exception events would clash with each other.

Again, all exception events are expected to be served properly with their respective ISR being executed, meaning if there is IRQ, we should see the initial program counter is halted and stored into a register $\$pc$ before the program counter jump to the respective ISR. When the ISR has run finish, program counter will return to the previous program by reading the value in $\$pc$.

When multiple interrupts happen, all exceptions events should be served according to the priority level without any conflict. External exception that comes from all IO devices have the highest priority over internal exception like Syscall, Undefined Instruction and Sign-overflow. While the priority level between external exception is determined by user by setting register in Priority Interrupt Controller as mentioned in [14]. However, in this test case, the values are all default, and the priority will depend on the vector number of IO device. The lower the vector number, the higher the priority. Hence, priority goes $UART > SPI > ADC$.

8.2 Testbench for Multiple Exception Test

```

`timescale 1ns / 1ps
`default_nettype none
//define demo006_ADC
define multiple_exception
`ifdef demo006_ADC
    `define TEST_CODE_PATH_DUT "demo006_ADC_v2mem_02program.hex"
    `define EXC_HANDLER_DUT "demo006_ADC_mem_03exc_handler.hex"
`endif

`ifdef multiple_exception
    `define TEST_CODE_PATH_DUT "Multiple_Exception+interrupt.hex"
    `define EXC_HANDLER_DUT "exception_handler.hex"
    `define TEST_CODE_PATH_CLIENT "Uart+Spi Transmit to DUT.hex"
    `define EXC_HANDLER_CLIENT "exception_handler.hex"
`endif

module tb_r32_pipeline();
//declaration
//===== INPUT =====
//System signal
reg    tb_u_clk;
reg    tb_u_rst;
//~~~~~
wire   tb_u_spi_mosi_dut;
wire   tb_u_spi_miso_dut;
wire   tb_u_spi_sclk_dut;
wire   tb_u_spi_ss_n_dut;

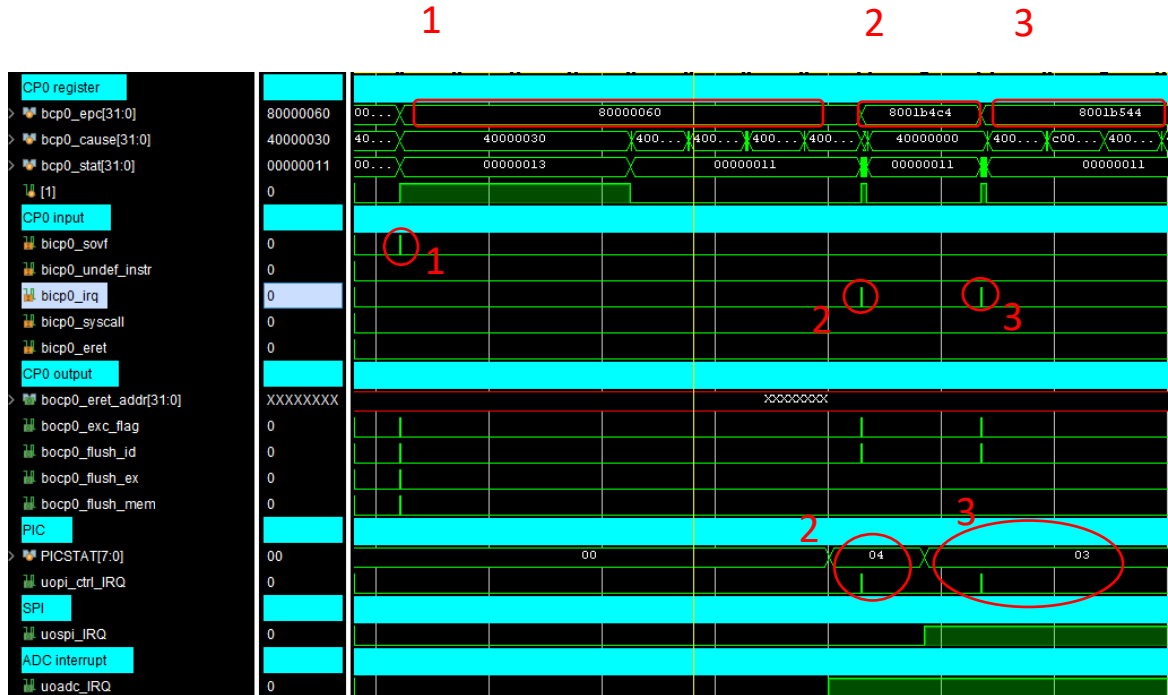
wire   tb_u_fc_sclk_dut;
wire   tb_u_fc_ss_dut;
wire   tb_u_fc_MOSI_dut;
wire   tb_u_fc_MISO1_dut;
wire   tb_u_fc_MISO2_dut;
wire   tb_u_fc_MISO3_dut;
wire   tb_ua_tx_rx_dut;
wire   tb_ua_RTS_dut, tb_ua_CTS_dut;
wire[31:0] tb_u_GPIO_dut;
//~~~~~
wire   tb_u_spi_mosi_client;
wire   tb_u_spi_miso_client;
wire   tb_u_spi_sclk_client;
wire   tb_u_spi_ss_n_client;

wire   tb_u_fc_sclk_client;
wire   tb_u_fc_ss_client;
wire   tb_u_fc_MOSI_client;
wire   tb_u_fc_MISO1_client;
wire   tb_u_fc_MISO2_client;
wire   tb_u_fc_MISO3_client;
wire   tb_ua_tx_rx_client;
wire   tb_ua_RTS_client, tb_ua_CTS_client;
wire[31:0] tb_u_GPIO_client;
...
...
...

```

Full Testbench coding is in Appendix B

8.3 Simulation result

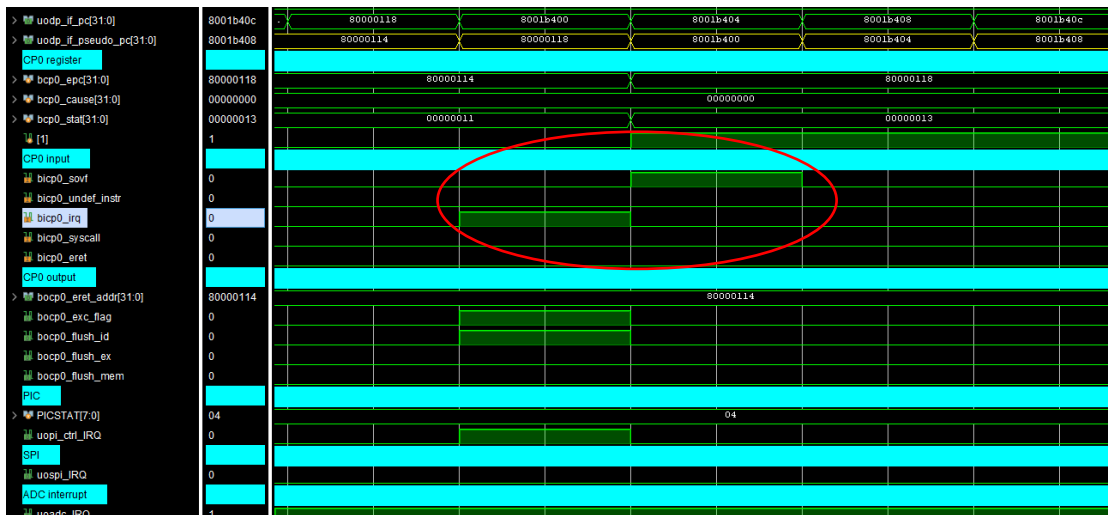


With the help of Co-processor 0(CP0) and Priority Interrupt Controller (PIC), we get to see both internal and external exception event getting served according to their priority.

1) In the figure above, when sign-overflow first happen, it is being served, and the user program's pc value is stored in the \$epc.

2) Before sign-overflow ISR can finish, ADC has fired an IRQ, as showed by the PICSTAT value '04' which is the vector number for ADC interrupt. CP0 will then serve ADC's IRQ first because external exception (UART, SPI, ADC) has higher priority than internal exception (sign-overflow, undefined instruction, syscall) as shown in [5]. PC for sign-overflow ISR is being stored in the \$epc for future return purpose.

3) The same thing happens to ADC, when the SPI fire an IRQ, PC for ADC's ISR is stored in the \$epc. Since no PICIPL value is set in this case, the priority between IO is determine by their vector number. SPI has higher priority over ADC with vector number of '03'



Unlike the UART interrupt as shown in [5], there are no exception conflicting happen between Trap and Interrupt where the two or more IRQ happen in the same clock cycle. The best sequence we can get is ADC interrupt triggered one clock cycle earlier than Trap. In this case, since ADC's IRQ has higher priority, it will continue its ISR until it finishes before servicing sign-overflow's ISR. At this point, the integration of UADC is completed since the CP0 can handle the exceptions with the correct priority.

Chapter 9: Conclusion and Future Work

9.1 Conclusion

The objectives of this project have been achieved. The ADC controller unit is successfully developed by revising and troubleshooting the previously developed ADC Controller Unit. It has been optimized and further enhanced as the ADC Controller Unit is now free of some unnecessary registers and state. Apart from that, the read operation has been completed and an interrupt function is implemented. Lastly, a new Clock Control block is integrated in the ADC controller unit to have control over the data storing speed.

As per the second objective, the ADC Controller Unit has also been successfully integrated into the RISC32 by using the I/O memory mapping technique, UADC_CREG and UADC_SREG are specifically designed to suit the functionality of XADC. Now, XADC can be easily configured, and the converted value can be easily obtained through the ADC Controller Unit. Also, an assembly language Interrupt Service Routine (ISR) has been specifically designed for ADC Controller Unit. The information about the ISR can be found in Chapter 6.

At this point, the documentation of ADC Controller Unit has now completed with the micro-architecture specification clarified in Chapter 5. Also, there are test plans, testbenches and simulation results included in Chapter 7 and 8 to prove that the design and integration are completed. With the availability of well-developed documents, any further research works can be done easier and speed up significantly as the functionality of the ADC Controller Unit has been verified.

9.2 Future Work

In the future, effort might be put into the synthesis and implementation of ADC Controller Unit in real FPGA. By having a physical application of ADC Controller Unit on FPGA, the exception handler can be further modified to carried out more meaningful ISR depend on the use cases. Besides, other XADC's operating mode and functions can also be tested out in the test plan to match more use cases.

Bibliography

- [1] Altera. (2013). *Cadence Incisive Enterprise Simulator Support*. Accessed: July 6, 2021. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/qts/qts_qii53003.pdf
- [2] B. L. Tan, K. M. Mok, J. J. Chang, W. K. Lee and S. O. Hwang, "RISC32-LP: Low-Power FPGA-Based IoT Sensor Nodes With Energy Reduction Program Analyzer," in *IEEE Internet of Things Journal*, vol. 9, no. 6, pp. 4214-4228, 15 March 2022. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9508410>
- [3] CONTEC. no date. *Analog I/O basic knowledge*. Accessed: July 1, 2021. [Online]. Available: <https://www.contec.com/support/basic-knowledge/daq-control/analog-io/>
- [4] D.A. Patterson and J.L. Hennessy, *Computer Organization and Design-The Hardware_Software Interface, Third Edition*. San Francisco: Morgan Kaufmann Publishers.
- [5] J.S. Goh. (2019). *The development of an exception scheme for 5-stage pipeline RISC processor*. Accessed: March 1, 2022. Available: http://eprints.utar.edu.my/3434/1/fyp_CT_2019_GJS_1503470.pdf
- [6] K.M. Mok. (2020). *Digital Systems Designs*. Lecture notes distributed in Faculty of Information and Communication Technology at Universiti Tunku Abdul Rahman.
- [7] K.M. Mok. (2021). *Computer Organization and Architecture*. Lecture notes distributed in Faculty of Information and Communication Technology at Universiti Tunku Abdul Rahman.
- [8] M. Langer. (2016). *Memory Mapped I/O, Polling, DMA*. Accessed: July 6, 2021. Available: <http://www.cim.mcgill.ca/~langer/273/20-notes.pdf>
- [9] M.A. Yong. (2021). *Design and Implementation of a SPI controller for zigbee module*. Accessed: March 1, 2022. [Online]. Available: http://eprints.utar.edu.my/3830/1/16ACB01733_FYP.pdf

BIBLIOGRAPHY

- [10] Maximintegrated. (2012). *Analog Solutions for Xilinx FPGAs*. Accessed: July 1, 2021. [Online]. Available: <https://www.maximintegrated.com/content/dam/files/design/technical-documents/product-guides/fpga-xilinx-product-guide.pdf>
- [11] Mentor Graphic. no date. *ModelSim PE Student Edition*. Accessed: July 6, 2021. [Online]. Available: https://www.mentor.com/company/higher_ed/modelsim-student-edition
- [12] Synopsys. no date. VCS. Accessed: July 6, 2021. [Online]. Available: <https://www.synopsys.com/verification/simulation/vcs.html>
- [13] W.P. Kiat, K.M. Mok, W.K. Lee, H.G. Goh, and R. Achar. (2020). An energy efficient FPGA partial reconfiguration based micro-architectural technique for IoT applications. *Microprocessors and Microsystems*, 73, p.102966. Presented at ELSEVIER 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0141933118304691>
- [14] W.P. Kiat. (2018). *The design of an FPGA-based processor with reconfigurable processor execution structure for internet of things (IoT) applications*. Accessed: April 6, 2021. [Online]. Available: <http://eprints.utar.edu.my/3146/1/CEA-2019-1601206-1.pdf>
- [15] Xilinx.com. (2018). *7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide (UG480)*. Accessed: July 6, 2021. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf
- [16] Xilinx.com. (2021). *Vivado Design Suite- HLx Editions*. Accessed: January 27, 2022. [Online]. Available: <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2020-3.html>

Appendix A: XADC Datasheet

XADC ports and description

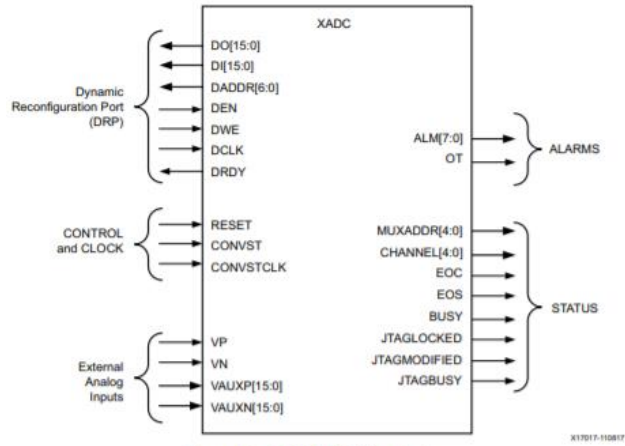


Figure 1-3: XADC Primitive Ports

Table 1-2: XADC Port Descriptions

Port	I/O	Description
DI[15:0]	Inputs	Input data bus for the DRP. ⁽¹⁾
DO[15:0]	Outputs	Output data bus for the DRP. ⁽¹⁾
DADDR[6:0]	Input	Address bus for the DRP. ⁽¹⁾
DEN ⁽²⁾	Input	Enable signal for the DRP. ⁽¹⁾
DWE ⁽²⁾	Input	Write enable for the DRP. ⁽¹⁾
DCLK	Input	Clock input for the DRP. ⁽¹⁾
DRDY ⁽²⁾	Output	Data ready signal for the DRP. ⁽¹⁾
RESET ⁽²⁾	Input	Asynchronous reset signal for the XADC control logic. RESET will be deasserted synchronously to DCLK or the internal configuration clock when DCLK is stopped.
CONVST ⁽³⁾	Input	Convert start input. This input controls the sampling instant on the ADC(s) inputs and is only used in event mode timing (see Event-Driven Sampling, page 73). This input comes from the general-purpose interconnect in the FPGA logic.
CONVSTCLK ⁽³⁾	Input	Convert start clock input. This input is connected to a clock net. Like CONVST, this input controls the sampling instant on the ADC(s) inputs and is only used in event mode timing. This input comes from the local clock distribution network in the FPGA logic. Thus, for the best control over the sampling instant (delay and jitter), a global clock input can be used as the CONVST source.
V _P , V _N	Input	One dedicated analog input pair. The XADC has one pair of dedicated analog input pins that provide a differential analog input. When designing with the XADC feature but not using the dedicated external channel of V _P and V _N , you should connect both V _P and V _N to analog ground.
VAUXP[15:0], VAUXN[15:0]	Inputs	Sixteen auxiliary analog input pairs. In addition to the dedicated differential analog input, the XADC can access 16 differential analog inputs by configuring digital I/O as analog inputs. These inputs can also be enabled pre-configuration through the JTAG port (see DRP JTAG Interface, page 47).
ALM[0] ⁽²⁾	Output	Temperature sensor alarm output.
ALM[1] ⁽²⁾	Output	V _{CCINT} sensor alarm output.
ALM[2] ⁽²⁾	Output	V _{CCAUX} sensor alarm output.
ALM[3] ⁽²⁾	Output	V _{CCBRAM} sensor alarm output.
ALM[4] ⁽⁴⁾	Output	V _{CCPINT} sensor alarm output.
ALM[5] ⁽⁴⁾	Output	V _{CCPAUX} sensor alarm output.
ALM[6] ⁽⁴⁾	Output	V _{CCO_DDR} sensor alarm output.

Table 1-2: XADC Port Descriptions (Cont'd)

Port	I/O	Description
ALM[7] ⁽²⁾	Output	Logic OR of bus ALM[6:0]. Can be used to flag the occurrence of any alarm.
OT ⁽²⁾	Output	Over-Temperature alarm output.
MUXADDR[4:0]	Outputs	These outputs are used in external multiplexer mode. They indicate the address of the next channel in a sequence to be converted. They provide the channel address for an external multiplexer (see External Multiplexer Mode , page 63).
CHANNEL[4:0]	Outputs	Channel selection outputs. The ADC input MUX channel selection for the current ADC conversion is placed on these outputs at the end of an ADC conversion.
EOC ⁽²⁾	Output	End of conversion signal. This signal transitions to active-High at the end of an ADC conversion when the measurement is written to the status registers (see Chapter 5, XADC Timing).
EOS ⁽²⁾	Output	End of sequence. This signal transitions to active-High when the measurement data from the last channel in an automatic channel sequence is written to the status registers (see Chapter 5, XADC Timing).
BUSY ⁽²⁾	Output	ADC busy signal. This signal transitions High during an ADC conversion. This signal also transitions High for an extended period during an ADC or sensor calibration.
JTAGLOCKED ⁽²⁾	Output	Indicates that a DRP port lock request has been made by the JTAG interface (see DRP JTAG Interface , page 47). This signal is also used to indicate that the DRP is ready for access (when Low).
JTAGMODIFIED ⁽²⁾	Output	Used to indicate that a JTAG write to the DRP has occurred.
JTAGBUSY ⁽²⁾	Output	Used to indicate that a JTAG DRP transaction is in progress.

Notes:

1. The DRP is the interface between the XADC and FPGA. All XADC registers can be accessed from the FPGA logic using this interface. For more details on the timing for these DRP signals, see [Figure 5-3](#), page 75.
2. Active-High signal.
3. Rising edge triggered signal.
4. Only available on Zynq-7000 SoC devices.

Formula for temperature

$$\text{Temperature } (^{\circ}\text{C}) = \frac{\text{ADC Code} \times 503.975}{4096} - 273.15$$

Formula for power-supply sensor

$$\text{Voltage} = \frac{\text{ADC Code}}{4096} \times 3\text{V}$$

Formula for auxiliary input

$$\text{Voltage} = \frac{\text{ADC Code}}{4096}$$

Note: ADC Code here represents the left-aligned 12-bit in the status register

Detail description for XADC Status register

Name	Address	Description
Temperature	00h	The result of the on-chip temperature sensor measurement is stored in this location. The data is MSB justified in the 16-bit register. The 12 MSBs correspond to the temperature sensor transfer function shown in Figure 2-9, page 33 .
V _{CCINT}	01h	The result of the on-chip V _{CCINT} supply monitor measurement is stored at this location. The data is MSB justified in the 16-bit register. The 12 MSBs correspond to the supply sensor transfer function shown in Figure 2-10, page 34 .
V _{CCAUX}	02h	The result of the on-chip V _{CCAUX} data supply monitor measurement is stored at this location. The data is MSB justified in the 16 bit register. The 12 MSBs correspond to the supply sensor transfer function shown in Figure 2-10 .
V _P /V _N	03h	The result of a conversion on the dedicated analog input channel is stored in this register. The data is MSB justified in the 16-bit register. The 12 MSBs correspond to the transfer function shown in Figure 2-6, page 31 or Figure 2-7, page 32 depending on analog input mode settings.
V _{REFP}	04h	The result of a conversion on the reference input V _{REFP} is stored in this register. The 12 MSBs correspond to the ADC transfer function shown in Figure 2-10 . The data is MSB justified in the 16-bit register. The supply sensor is used when measuring V _{REFP} .
V _{REFN}	05h	The result of a conversion on the reference input V _{REFN} is stored in this register. This channel is measured in bipolar mode with a two's complement output coding as shown in Figure 2-3, page 27 . By measuring in bipolar mode, small positive and negative offset around 0V (V _{REFN}) can be measured. The supply sensor is also used to measure V _{REFN} , thus 1 LSB = 3V/4096. The data is MSB justified in the 16-bit register.

APPENDIX A

Name	Address	Description
V _{CCBRAM}	06h	The result of the on-chip V _{CCBRAM} supply monitor measurement is stored at this location. The data is MSB justified in the 16-bit register. The 12 MSBs correspond to the supply sensor transfer function shown in Figure 2-10.
Undefined	07h	This location is unused and contains invalid data.
Supply A offset	08h	The calibration coefficient for the supply sensor offset using ADC A is stored at this location.
ADC A offset	09h	The calibration coefficient for the ADC A offset is stored at this location.
ADC A gain	0Ah	The calibration coefficient for the ADC A gain error is stored at this location.
Undefined	0Bh to 0Ch	These locations are unused and contain invalid data.
V _{CCPINT} ⁽¹⁾	0Dh	The result of a conversion on the PS supply, V _{CCPINT} is stored in this register. The 12 MSBs correspond to the ADC transfer function shown in Figure 2-10, page 34. The data is MSB justified in the 16-bit register. The supply sensor is used when measuring V _{CCPINT} .
V _{CCPAUX} ⁽¹⁾	0Eh	The result of a conversion on the PS supply, V _{CCPAUX} is stored in this register. The 12 MSBs correspond to the ADC transfer function shown in Figure 2-10, page 34. The data is MSB justified in the 16-bit register. The supply sensor is used when measuring V _{CCPAUX} .
V _{CCO_DDR} ⁽¹⁾	0Fh	The result of a conversion on the PS supply, V _{CCO_DDR} is stored in this register. The 12 MSBs correspond to the ADC transfer function shown in Figure 2-10, page 34. The data is MSB justified in the 16-bit register. The supply sensor is used when measuring V _{CCO_DDR} .
VAUXP[15:0]/VAUXN[15:0]	10h to 1Fh	The results of the conversions on auxiliary analog input channels are stored in this register. The data is MSB justified in the 16-bit register. The 12 MSBs correspond to the transfer function shown in Figure 2-2, page 26 or Figure 2-3, page 27 depending on analog input mode settings.
Max temp	20h	Maximum temperature measurement recorded since power-up or the last XADC reset.
Max V _{CCINT}	21h	Maximum V _{CCINT} measurement recorded since power-up or the last XADC reset.
Max V _{CCAUX}	22h	Maximum V _{CCAUX} measurement recorded since power-up or the last XADC reset.
Max V _{CCBRAM}	23h	Maximum V _{CCBRAM} measurement recorded since power-up or the last XADC reset.
Min temp	24h	Minimum temperature measurement recorded since power-up or the last XADC reset.

Name	Address	Description
Min V _{CCINT}	25h	Minimum V _{CCINT} measurement recorded since power-up or the last XADC reset.
Min V _{CCAUX}	26h	Minimum V _{CCAUX} measurement recorded since power-up or the last XADC reset.
Min V _{CCBRAM}	27h	Minimum V _{CCBRAM} measurement recorded since power-up or the last XADC reset.
V _{CCPINT} ⁽¹⁾ max	28h	Maximum V _{CCPINT} measurement recorded since power-up or the last XADC reset.
V _{CCPAUX} ⁽¹⁾ max	29h	Maximum V _{CCPAUX} measurement recorded since power-up or the last XADC reset.
V _{CCO_DDR} ⁽¹⁾ max	2Ah	Maximum V _{CCO_DDR} measurement recorded since power-up or the last XADC reset.
Unassigned	2Bh	
V _{CCPINT} ⁽¹⁾ min	2Ch	Minimum V _{CCPINT} measurement recorded since power-up or the last XADC reset.
V _{CCPAUX} ⁽¹⁾ min	2Dh	Minimum V _{CCAUX} measurement recorded since power-up or the last XADC reset.
V _{CCO_DDR} ⁽¹⁾ min	2Eh	Minimum V _{CCO_DDR} measurement recorded since power-up or the last XADC reset.
Unassigned	2Fh	
Supply B offset	30h	The calibration coefficient for the supply sensor offset using ADC B is stored at this location.
ADC B offset	31h	The calibration coefficient for the ADC B offset is stored at this location.
ADC B gain	32h	The calibration coefficient for the ADC B gain error is stored at this location.
Undefined	33h to 3Eh	These locations are unused and contain invalid data.
Flag	3Fh	This register contains general status information (see Flag Register).

Notes:

1. These channels are only available in Zynq-7000 SoC devices.

Detail for XADC Configuration Registers

DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	Config Reg #0 DADDR[6:0] = 40h
CAVG	0	AVG1	AVG0	MUX	B \bar{U}	E \bar{C}	ACQ	0	0	0	CH4	CH3	CH2	CH1	CH0	
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	Config Reg #1 DADDR[6:0] = 41h
SEQ3	SEQ2	SEQ1	SEQ0	ALM6 (Note1)	ALM5 (Note1)	ALM4 (Note1)	ALM3	CAL3	CAL2	CAL1	CAL0	ALM2	ALM1	ALM0	OT	
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	Config Reg #2 DADDR[6:0] = 42h
CD7	CD6	CD5	CD4	CD3	CD2	CD1	CD0	0	0	PD1	PD0	0	0	0	0	

X17030-110817

Table 3-4: Configuration Register 0 Bit Definitions

Bit	Name	Description
DI0 to DI3	CH0 to CH4	When operating in single channel mode or external multiplexer mode, these bits are used to select the ADC input channel. See Table 3-7 for the channel assignments.
DI8	ACQ	When using single channel mode, this bit is used to increase the settling time available on external analog inputs in continuous sampling mode by six ADCCLK cycles (see Chapter 2, Analog-to-Digital Converter and Chapter 5, XADC Timing). The acquisition time is increased by setting this bit to logic 1. ⁽¹⁾
DI9	E \bar{C}	This bit is used to select either continuous or event-driven sampling mode for the ADC (see Chapter 5, XADC Timing). A logic 1 places the ADC in event-driven sampling mode and a logic 0 places the ADC in continuous sampling mode.
DI10	B \bar{U}	This bit is used in single channel mode to select either unipolar or bipolar operating mode for the ADC analog inputs (see Analog Inputs, page 28). A logic 1 places the ADC in bipolar mode and a logic 0 places the ADC in unipolar mode.
DI11	MUX	This bit should be set to a logic 1 to enable external multiplexer mode. See External Multiplexer Mode for more information.

Table 3-4: Configuration Register 0 Bit Definitions (Cont'd)

Bit	Name	Description
DI12, DI13	AVG0, AVG1	These bits are used to set the amount of sample averaging on selected channels in both single channel and sequence modes (see Table 3-8 and Chapter 4, XADC Operating Modes).
DI15	CAVG	This bit is used to disable averaging for the calculation of the calibration coefficients. Averaging is enabled by default (logic 0). To disable averaging, set this bit to logic 1. Averaging is fixed at 16 samples.

Table 3-5: Configuration Register 1 Bit Definitions

Bit	Name	Description
DI0	OT	This bit is used to disable the over-temperature signal. The alarm is disabled by setting this bit to logic 1.
DI1 to DI3, DI8	ALM0 to ALM3	These bits are used to disable individual alarm outputs for temperature, V_{CCINT} , V_{CCAUX} , and V_{CCBRAM} , respectively. A logic 1 disables an alarm output.
DI9 to DI11	ALM4 to ALM6	These bits are used to disable individual alarm outputs for V_{CCPINT} , V_{CCPAUX} , and V_{CCO_DDR} , respectively. A logic 1 disables an alarm output.
DI4 to DI7	CAL0 to CAL3	These bits enable the application of the calibration coefficients to the ADC and on-chip supply sensor measurements. A logic 1 enables calibration and a logic 0 disables calibration. For bit assignments, see Table 3-10.
DI12 to DI15	SEQ0 to SEQ3	These bits enable the channel-sequencer function. For the bit assignments, see Table 3-9. See Chapter 4, XADC Operating Modes, for more information.

Table 3-7: ADC Channel Select

ADC Channel	CH4	CH3	CH2	CH1	CH0	Description
0	0	0	0	0	0	On-chip temperature
1	0	0	0	0	1	V_{CCINT}
2	0	0	0	1	0	V_{CCAUX}
3	0	0	0	1	1	V_P , V_N – Dedicated analog inputs
4	0	0	1	0	0	V_{REFP} (1.25V) ⁽¹⁾
5	0	0	1	0	1	V_{REFN} (0V) ⁽¹⁾
6	0	0	1	1	0	V_{CCBRAM}
7	0	0	1	1	1	Invalid channel selection
8	0	1	0	0	0	Carry out an XADC calibration
9–12	Invalid channel selection
13	0	1	1	0	1	V_{CCPINT} ⁽³⁾
14	0	1	1	1	0	V_{CCPAUX} ⁽³⁾
15	0	1	1	1	1	V_{CCO_DDR} ⁽³⁾
16	1	0	0	0	0	VAUXP[0], VAUXN[0] – Auxiliary channel 0
17	1	0	0	0	1	VAUXP[1], VAUXN[1] – Auxiliary channel 1
18–31	VAUXP[2:15], VAUXN[2:15] – Auxiliary channels 2 to 15 ⁽²⁾

Table 3-8: Averaging Filter Settings

AVG1	AVG0	Function
0	0	No averaging
0	1	Average 16 samples
1	0	Average 64 samples
1	1	Average 256 samples

Table 3-9: Sequencer Operation Settings

SEQ3	SEQ2	SEQ1	SEQ0	Function
0	0	0	0	Default mode
0	0	0	1	Single pass sequence
0	0	1	0	Continuous sequence mode
0	0	1	1	Single channel mode (sequencer off)
0	1	X	X	Simultaneous sampling mode
1	0	X	X	Independent ADC mode
1	1	X	X	Default mode

The Control Register for Sequence mode in XADC

ADC Channel Selection Registers (48h and 49h)

Table 4-1: Sequencer On-Chip Channel Selection (48h)

Sequence Number 7 Series/Zynq-7000	Bit	ADC Channel	Description
1/1	0	8	XADC calibration
-	1	9	Invalid channel selection
	2	10	
	3	11	
	4	12	
-/2	5	13	V _{CCPINT}
-/3	6	14	V _{CCPAUX}
-/4	7	15	V _{CCO_DDR}
2/5	8	0	On-chip temperature
3/6	9	1	V _{CCINT}
4/7	10	2	V _{CCAUX}
5/8	11	3	V _P , V _N – Dedicated analog inputs
6/9	12	4	V _{REFP}
7/10	13	5	V _{REFN}
8/11	14	6	V _{CCBRAM}
-	15	7	Invalid channel selection

Table 4-2: Sequencer Auxiliary Channel Selection (49h)

Sequence Number 7 Series/Zynq-7000	Bit	ADC Channel	Description
9/12	0	16	VAUXP[0], VAUXN[0] – Auxiliary channel 0
10/13	1	17	VAUXP[1], VAUXN[1] – Auxiliary channel 1
11/14	2	18	VAUXP[2], VAUXN[2] – Auxiliary channel 2
12/15	3	19	VAUXP[3], VAUXN[3] – Auxiliary channel 3
13/16	4	20	VAUXP[4], VAUXN[4] – Auxiliary channel 4

Table 4-2: Sequencer Auxiliary Channel Selection (49h) (Cont'd)

Sequence Number 7 Series/Zynq-7000	Bit	ADC Channel	Description
14/17	5	21	VAUXP[5], VAUXN[5] – Auxiliary channel 5
15/18	6	22	VAUXP[6], VAUXN[6] – Auxiliary channel 6 ⁽¹⁾
16/19	7	23	VAUXP[7], VAUXN[7] – Auxiliary channel 7 ⁽¹⁾
17/20	8	24	VAUXP[8], VAUXN[8] – Auxiliary channel 8
18/21	9	25	VAUXP[9], VAUXN[9] – Auxiliary channel 9
19/22	10	26	VAUXP[10], VAUXN[10] – Auxiliary channel 10
20/23	11	27	VAUXP[11], VAUXN[11] – Auxiliary channel 11
21/24	12	28	VAUXP[12], VAUXN[12] – Auxiliary channel 12
22/25	13	29	VAUXP[13], VAUXN[13] – Auxiliary channel 13 ⁽¹⁾
23/26	14	30	VAUXP[14], VAUXN[14] – Auxiliary channel 14 ⁽¹⁾
24/27	15	31	VAUXP[15], VAUXN[15] – Auxiliary channel 15 ⁽¹⁾

ADC Channel Averaging (4Ah and 4Bh)

ADC Channel Analog-Input Mode (4Ch and 4Dh)

Note: ADC Channel Averaging and ADC Channel Analog-Input Mode will also have the same bit assignments to toggle enable/disable averaging and toggle analog-input mode respectively.

XADC Alarms Threshold Registers

Table 4-8: Alarm Threshold Registers

Control Register	Description	Alarm
50h	Temperature upper	ALM[0]
51h	V _{CCINT} upper	ALM[1]
52h	V _{CCAUX} upper	ALM[2]
53h	OT alarm limit ⁽¹⁾	OT
54h	Temperature lower	ALM[0]
55h	V _{CCINT} lower	ALM[1]
56h	V _{CCAUX} lower	ALM[2]
57h	OT alarm reset ⁽¹⁾	OT
58h	V _{CCBRAM} upper	ALM[3]
59h	V _{CCPINT} upper ⁽²⁾	ALM[4]
5Ah	V _{CCPAUX} upper ⁽²⁾	ALM[5]
5Bh	V _{CCO_DDR} upper ⁽²⁾	ALM[6]
5Ch	V _{CCBRAM} lower	ALM[3]
5Dh	V _{CCPINT} lower ⁽²⁾	ALM[4]
5Eh	V _{CCPAUX} lower ⁽²⁾	ALM[5]
5Fh	V _{CCO_DDR} lower ⁽²⁾	ALM[6]

Dynamic Reconfiguration Port (DRP) Timing

Figure 5-3 illustrates a DRP read and write operation. When the DEN is logic High, the DRP address (DADDR) and write enable (DWE) inputs are captured on the next rising edge of DCLK. DEN should only go high for one DCLK period.

If DWE is logic Low, a DRP read operation is carried out. The data for this read operation is valid on the DO bus when DRDY goes high. Thus DRDY should be used to capture the DO bus. For a write operation, the DWE signal is logic High and the DI bus and DRP address (DADDR) is captured on the next rising edge of DCLK. The DRDY signal goes logic High when the data has been successfully written to the DRP register. A new read or write operation cannot be initiated until the DRDY signal has gone low.

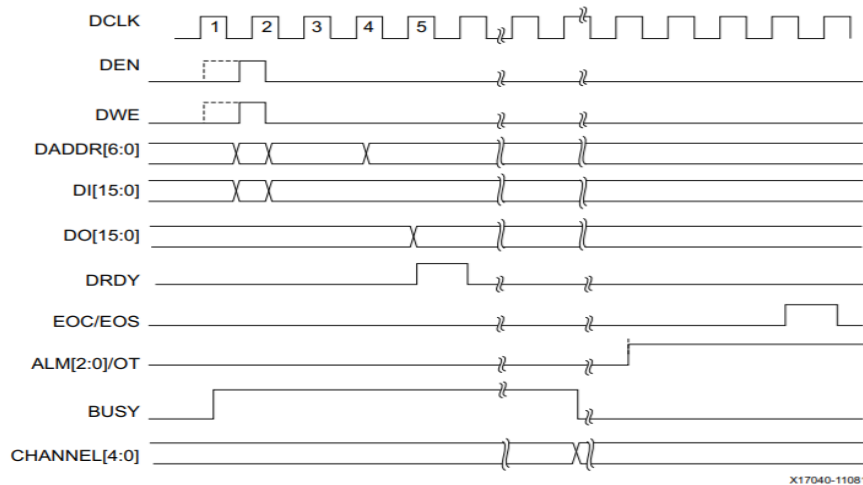


Figure 5-3: DRP Detailed Timing

XADC Verilog code from Xilinx

```

`timescale 1ns / 1ps
module ug480 (
  input DCLK, // Clock input for DRP
  input RESET,
  input [3:0] VAUXP, VAUXN, // Auxiliary analog channel inputs
  input VP, VN, // Dedicated and Hardwired Analog Input Pair

  output reg [15:0] MEASURED_TEMP, MEASURED_VCCINT,
  output reg [15:0] MEASURED_VCCAUX, MEASURED_VCCBRAM,
  output reg [15:0] MEASURED_AUX0, MEASURED_AUX1,

  output reg [15:0] MEASURED_AUX2, MEASURED_AUX3,

  output wire [7:0] ALM,
  output wire [4:0] CHANNEL,
  output wire      OT,
  output wire      EOC,
  output wire      EOS
);
  wire busy;
  wire [5:0] channel;
  wire drdy;
  wire eoc;
  wire eos;
  wire i2c_sclk_in;
  wire i2c_sclk_ts;
  wire i2c_sda_in;
  wire i2c_sda_ts;
  reg [6:0] daddr;
  reg [15:0] di_drp;
  wire [15:0] do_drp;
  wire [15:0] vauxp_active;
  wire [15:0] vauxn_active;
  wire dclk_bufg;

  reg [1:0] den_reg;
  reg [1:0] dwe_reg;
  reg [1:0] den_reg;
  reg [1:0] dwe_reg;

  reg [7:0] state = init_read;
  parameter init_read = 8'h00,
             read_waitdrdy = 8'h01,
             write_waitdrdy = 8'h03,
             read_reg00 = 8'h04,
             reg00_waitdrdy = 8'h05,
             read_reg01 = 8'h06,
             reg01_waitdrdy = 8'h07,
             read_reg02 = 8'h08,
             reg02_waitdrdy = 8'h09,
             read_reg06 = 8'h0a,
             reg06_waitdrdy = 8'h0b,

```

```

        read_reg10    = 8'h0c,
        reg10_waitdrdy = 8'h0d,
        read_reg11    = 8'h0e,
        reg11_waitdrdy = 8'h0f,
        read_reg12    = 8'h10,
        reg12_waitdrdy = 8'h11,
        read_reg13    = 8'h12,
        reg13_waitdrdy = 8'h13;
    BUFG i_bufg (.I(DCLK), .O(dclk_bufg));
    always @(posedge dclk_bufg)
    if (RESET) begin
        state <= init_read;
        den_reg <= 2'h0;
        dwe_reg <= 2'h0;
        di_drp <= 16'h0000;
    end
    else
        case (state)
            init_read : begin
                daddr <= 7'h40;
                den_reg <= 2'h2; // performing read
                if (busy == 0 ) state <= read_waitdrdy;
            end
            read_waitdrdy :
                if (eos ==1) begin
                    di_drp <= do_drp & 16'h03_FF; //Clearing AVG bits for Configreg0
                    daddr <= 7'h40;
                    den_reg <= 2'h2;
                    dwe_reg <= 2'h2; // performing write
                    state <= write_waitdrdy;
                end
                else begin
                    den_reg <= { 1'b0, den_reg[1] } ;
                    dwe_reg <= { 1'b0, dwe_reg[1] } ;
                    state <= state;
                end
            write_waitdrdy :
                if (drdy ==1) begin
                    state <= read_reg00;
                end
                else begin
                    den_reg <= { 1'b0, den_reg[1] } ;
                    dwe_reg <= { 1'b0, dwe_reg[1] } ;
                    state <= state;
                end
            read_reg00 : begin
                daddr <= 7'h00;
                den_reg <= 2'h2; // performing read
                if (eos == 1) state <= reg00_waitdrdy;
            end
            reg00_waitdrdy :
                if (drdy ==1) begin
                    MEASURED_TEMP <= do_drp;
                    state <= read_reg01;
                end
        endcase

```

```

else begin
    den_reg <= { 1'b0, den_reg[1] };
    dwe_reg <= { 1'b0, dwe_reg[1] };
    state <= state;
end
end
read_reg01 : begin
    daddr <= 7'h01;
    den_reg <= 2'h2; // performing read
    state <= reg01_waitdrdy;
end
reg01_waitdrdy :
if (drdy ==1) begin
    MEASURED_VCCINT = do_drp;
    state <= read_reg02;
end
else begin
    den_reg <= { 1'b0, den_reg[1] };
    dwe_reg <= { 1'b0, dwe_reg[1] };
    state <= state;
end
end
read_reg02 : begin
    daddr <= 7'h02;
    den_reg <= 2'h2; // performing read
    state <= reg02_waitdrdy;
end
reg02_waitdrdy :
if (drdy ==1) begin
    MEASURED_VCCAUX <= do_drp;
    state <= read_reg06;
end
else begin
    den_reg <= { 1'b0, den_reg[1] };
    dwe_reg <= { 1'b0, dwe_reg[1] };
    state <= state;
end
end
read_reg06 : begin
    daddr <= 7'h06;
    den_reg <= 2'h2; // performing read
    state <= reg06_waitdrdy;
end
reg06_waitdrdy :
if (drdy ==1) begin
    MEASURED_VCCBRAM <= do_drp;
    state <= read_reg10;
end
end

```

```

else begin
    den_reg <= { 1'b0, den_reg[1] };
    dwe_reg <= { 1'b0, dwe_reg[1] };
    state <= state;
end
read_reg10 : begin
    daddr <= 7'h10;
    den_reg <= 2'h2; // performing read
    state <= reg10_waitdrdy;
end
reg10_waitdrdy :
    if (drdy ==1) begin
        MEASURED_AUX0 <= do_drp;
        state <= read_reg11;
    end
    else begin
        den_reg <= { 1'b0, den_reg[1] };
        dwe_reg <= { 1'b0, dwe_reg[1] };
        state <= state;
    end
read_reg11 : begin
    daddr <= 7'h11;
    den_reg <= 2'h2; // performing read
    state <= reg11_waitdrdy;
end
reg11_waitdrdy :
    if (drdy ==1) begin
        MEASURED_AUX1 <= do_drp;
        state <= read_reg12;
    end
    else begin
        den_reg <= { 1'b0, den_reg[1] };
        dwe_reg <= { 1'b0, dwe_reg[1] };
        state <= state;
    end
read_reg12 : begin
    daddr <= 7'h12;
    den_reg <= 2'h2; // performing read
    state <= reg12_waitdrdy;
end
reg12_waitdrdy :
    if (drdy ==1) begin
        MEASURED_AUX2 <= do_drp;
        state <= read_reg13;
    end
    else begin
        den_reg <= { 1'b0, den_reg[1] };
        dwe_reg <= { 1'b0, dwe_reg[1] };
        state <= state;
    end
read_reg13 : begin
    daddr <= 7'h13;
    den_reg <= 2'h2; // performing read
    state <= reg13_waitdrdy;
end

```



```

reg13_waitdrdy :
  if (drdy ==1)      begin
    MEASURED_AUX3 <= do_drp;
    state <=read_reg00;
    daddr  <= 7'h00;
  end
  else begin
    den_reg <= { 1'b0, den_reg[1] } ;
    dwe_reg <= { 1'b0, dwe_reg[1] } ;
    state <= state;
  end
default : begin
  daddr <= 7'h40;
  den_reg <= 2'h2; // performing read
  state <= init_read;
end
endcase

```

```

XADC # (// Initializing the XADC Control Registers
.INIT_40(16'h9000),// averaging of 16 selected for external channels
.INIT_41(16'h2ef0),// Continuous Seq Mode, Disable unused ALMs, Enable
calibration
.INIT_42(16'h0400),// Set DCLK divides
.INIT_48(16'h4701),// CHSEL1 - enable Temp VCCINT, VCCAUX, VCCBRAM,
and calibration
.INIT_49(16'h000f),// CHSEL2 - enable aux analog channels 0 - 3
.INIT_4A(16'h0000),// SEQAVG1 disabled
.INIT_4B(16'h0000),// SEQAVG2 disabled
.INIT_4C(16'h0000),// SEQINMODE0
.INIT_4D(16'h0000),// SEQINMODE1
.INIT_4E(16'h0000),// SEQACQ0
.INIT_4F(16'h0000),// SEQACQ1
.INIT_50(16'hb5ed),// Temp upper alarm trigger 85°C
.INIT_51(16'h5999),// Vccint upper alarm limit 1.05V
.INIT_52(16'hA147),// Vccaux upper alarm limit 1.89V
.INIT_53(16'hdddd),// OT upper alarm limit 125°C - see Thermal
Management
.INIT_54(16'ha93a),// Temp lower alarm reset 60°C
.INIT_55(16'h5111),// Vccint lower alarm limit 0.95V
.INIT_56(16'h91Eb),// Vccaux lower alarm limit 1.71V
.INIT_57(16'hae4e),// OT lower alarm reset 70°C - see Thermal
Management
.INIT_58(16'h5999),// VCCBRAM upper alarm limit 1.05V
.SIM_MONITOR_FILE("design.txt")// Analog Stimulus file for simulation
)

```

```

XADC_INST (// Connect up instance IO. See UG480 for port descriptions
.CONVST (1'b0), // not used
.CONVSTCLK (1'b0), // not used
.DADDR (daddr),
.DCLK (dclk_bufg),
.DEN (den_reg[0]),
.DI (di_drp),
.DWE (dwe_reg[0]),
.RESET (RESET),
.VAUXN (vauxn_active ),
.VAUXP (vauxp_active ),
.ALM (ALM),
.BUSY (busy),
.CHANNEL(CHANNEL),
.DO (do_drp),
.DRDY (drdy),
.EOC (eoc),
.EOS (eos),
.JTAGBUSY (), // not used
.JTAGLOCKED (), // not used
.JTAGMODIFIED (), // not used
.OT (OT),
.MUXADDR (), // not used
.VP (VP),
.VN (VN)
);

assign vauxp_active = {12'h000, VAUXP[3:0]};
assign vauxn_active = {12'h000, VAUXN[3:0]};

assign EOC = eoc;
assign EOS = eos;

endmodule

```

Testbench code from Xilinx

```
`timescale 1ns / 1ps
module ug480_tb;
  reg [3:0]   VAUXP, VAUXN;
  reg        RESET;
  reg        DCLK;

  wire [15:0] MEASURED_TEMP, MEASURED_VCCINT, MEASURED_VCCAUX;
  wire [15:0] MEASURED_VCCBRAM, MEASURED_AUX0, MEASURED_AUX1;
  wire [15:0] MEASURED_AUX2, MEASURED_AUX3;
  wire [15:0] ALM;
initial
  begin
    DCLK = 0;
    RESET = 0;
  end

always #(10) DCLK= ~DCLK;

// Instantiate the Unit Under Test (UUT)
ug480 uut (
  .VAUXP (VAUXP),
  .VAUXN (VAUXN),
  .RESET (RESET),
  .ALM (ALM),
  .DCLK (DCLK),
  .MEASURED_TEMP (MEASURED_TEMP),
  .MEASURED_VCCINT (MEASURED_VCCINT),
  .MEASURED_VCCAUX (MEASURED_VCCAUX),
  .MEASURED_VCCBRAM (MEASURED_VCCBRAM),
  .MEASURED_AUX0 (MEASURED_AUX0),
  .MEASURED_AUX1 (MEASURED_AUX1),
  .MEASURED_AUX2 (MEASURED_AUX2),
  .MEASURED_AUX3 (MEASURED_AUX3)
);
endmodule
```

APPENDIX A

Original Analog Stimulus File from Xilinx

```
// This analog stimulus file is used to inject analog signals (e.g., volts, temperature) for a simulation.
// Units are as follows:
//      Time:                nanoseconds [ns]
//      Voltage (All rails): volts [V]
//      Temperature:        degrees C [C]. Please note that the temperature transfer function is in terms of Kelvin
//
// In this example the VCCAUX supply moves outside the 1.89V upper alarm limit at 67 us
// An alarm is generate when the VCCAUX channel is sampled and converted by the ADC

TIME VAUXP[0] VAUXN[0] VAUXP[1] VAUXN[1] VAUXP[2] VAUXN[2] VAUXP[3] VAUXN[3] TEMP VCCINT VCCAUX VCCBRAM
00000 0.005 0.0 0.2 0.0 0.5 0.0 0.1 0.0 25 1.0 1.8 1.0
67000 0.020 0.0 0.400 0.0 0.49 0.0 0.2 0.0 85 1.05 1.9 1.05
100000 0.049 0.0 0.600 0.0 0.51 0.0 0.5 0.0 105 0.95 1.71 0.95
134000 0.034 0.0 0.900 0.0 0.53 0.0 0.0 0.0 0 1.00 1.8 1.0
```

Appendix B: Simulation Source

Testbench for ADC Controller Unit Level Functional Test

```

module uadc_v3_tb(
);
#####
//WRITE
//DATA AND ADDRESS BUS
reg [31:0]    tb_ip_wb_w_din;
//CONTROL SIGNALS
wire         tb_op_wb_w_ack; //ACKNOWLEDGE
reg [3:0]    tb_ip_wb_w_sel; //GRANULARITY - SELECT WHICH BYTE /
HALFWORD / WORD
reg [5:0]    tb_ip_wb_w_addr;
reg         tb_ip_wb_w_we;    //WRITE ENABLE
reg         tb_ip_wb_w_stb; //STROBE
//READ
//DATA AND ADDRESS BUS
wire [31:0]  tb_op_wb_r_dout;
//CONTROL SIGNALS
wire         tb_op_wb_r_ack; //ACKNOWLEDGE
reg [3:0]    tb_ip_wb_r_sel; //GRANULARITY - SELECT WHICH
BYTE/HALFWORD/WORD
reg [5:0]    tb_ip_wb_r_addr;
reg         tb_ip_wb_r_we;    //WRITE ENABLE
reg         tb_ip_wb_r_stb; //STROBE
reg         tb_ip_wb_clk;
reg         tb_ip_wb_rst;
#####
reg         tb_ip_ADCIE;///
wire        tb_op_IRQ;///
//analog input to XADC
reg [3:0]    tb_ip_VAUXP, tb_ip_VAUXN;

uadc_v3 uut
(.uiadc_wb_w_din(tb_ip_wb_w_din),
//CONTROL SIGNALS
.uoadc_wb_w_ack(tb_op_wb_w_ack), //ACKNOWLEDGE
.uiadc_wb_w_sel(tb_ip_wb_w_sel), //GRANULARITY - SELECT WHICH
BYTE/HALFWORD/WORD
.uiadc_wb_w_addr(tb_ip_wb_w_addr),
.uiadc_wb_w_we(tb_ip_wb_w_we),    //WRITE ENABLE
.uiadc_wb_w_stb(tb_ip_wb_w_stb), //STROBE
//READ
//DATA AND ADDRESS BUS
.uoadc_wb_r_dout(tb_op_wb_r_dout),
//CONTROL SIGNALS
.uoadc_wb_r_ack(tb_op_wb_r_ack), //ACKNOWLEDGE
.uiadc_wb_r_sel(tb_ip_wb_r_sel), //GRANULARITY - SELECT WHICH
BYTE/HALFWORD/WORD
.uiadc_wb_r_addr(tb_ip_wb_r_addr),

```

APPENDIX B

```

.uiadc_wb_r_we(tb_ip_wb_r_we), //WRITE ENABLE
.uiadc_wb_r_stb(tb_ip_wb_r_stb), //STROBE
.uiadc_wb_clk(tb_ip_wb_clk),
.uiadc_wb_rst(tb_ip_wb_rst),
//#####
.uiadc_ADCIE(tb_ip_ADCIE),/**need to build
.uoadc_IRQ(tb_op_IRQ),/**need to build
//analog input to XADC
.uiadc_VP(),
.uiadc_VN(),
.uiadc_VAUXP(tb_ip_VAUXP),
.uiadc_VAUXN(tb_ip_VAUXN));

initial begin
    tb_ip_wb_clk = 1'b0;
    tb_ip_wb_rst = 1'b0;
end

always #(5) tb_ip_wb_clk = ~tb_ip_wb_clk;

initial begin
    repeat(2) @(posedge tb_ip_wb_clk);
    tb_ip_wb_rst = 1'b1;
    repeat(1) @(posedge tb_ip_wb_clk);
    tb_ip_wb_rst = 1'b0;
//=====RESET=====//
/* repeat(30000) @(posedge tb_ip_wb_clk);
    tb_ip_wb_rst <= 1'b1;
    repeat(1) @(posedge tb_ip_wb_clk);
    tb_ip_wb_rst <= 1'b0;
*/
//=====READ=====//
/* repeat(30000) @(posedge tb_ip_wb_clk);
    tb_ip_wb_r_we <= 1'b0;
    tb_ip_wb_r_stb <= 1'b1;
    tb_ip_wb_r_sel <= 4'b1111;

//////////READ TEMP and Vccint//////////
    @(posedge tb_ip_wb_clk);
    tb_ip_wb_r_addr <= 6'b001_000;

//////////READ Vccaux//////////
    @(posedge tb_ip_wb_clk);
    tb_ip_wb_r_addr = 6'b001_001;
//////////READ Vccbram//////////
    @(posedge tb_ip_wb_clk);
    tb_ip_wb_r_addr = 6'b001_011;
//////////READ VAUXP/N[0] and VAUXP/N[1]//////////
    @(posedge tb_ip_wb_clk);
    tb_ip_wb_r_addr = 6'b010_000;
//////////READ VAUXP/N[2] and VAUXP/N[3]//////////
    @(posedge tb_ip_wb_clk);
    tb_ip_wb_r_addr = 6'b010_001; */

```

```

//=====WRITE=====//

//////////ENABLE all channel to do average//////////
/*  repeat(30000) @(posedge tb_ip_wb_clk);
    tb_ip_wb_w_din <= 32'h000f_4700;//CREG[3]
    tb_ip_wb_w_sel <= 4'b1111;
    tb_ip_wb_w_addr <= 6'b000_011;
    tb_ip_wb_w_we <=1'b1;
    tb_ip_wb_w_stb <= 1'b1;
    @(posedge tb_ip_wb_clk);//clear
    tb_ip_wb_w_din <= 32'h0000_000;
    tb_ip_wb_w_sel <= 4'b0000;
    tb_ip_wb_w_addr <= 6'b000_000;
    tb_ip_wb_w_we <=1'b0;
    tb_ip_wb_w_stb <= 1'b0;*/

////////// CONFIGURE TO SINGLE CHANNEL WITH ONLY TEMP //////////
/*  repeat(10000) @(posedge tb_ip_wb_clk);
    tb_ip_wb_w_din <= 32'h0000_3ef0;//CREG[0] = config 1
    tb_ip_wb_w_sel <= 4'b1100;
    tb_ip_wb_w_addr <= 6'b000_000;
    tb_ip_wb_w_we <=1'b1;
    tb_ip_wb_w_stb <= 1'b1;
    //can clear the unnecessary bit also.
    @(posedge tb_ip_wb_clk);//clear
    tb_ip_wb_w_din <= 32'h0000_000;
    tb_ip_wb_w_sel <= 4'b0000;
    tb_ip_wb_w_addr <= 6'b000_000;
    tb_ip_wb_w_we <=1'b0;
    tb_ip_wb_w_stb <= 1'b0; */

//////////CONFIGURE TO BIPOLAR ANALOG INPUT MODE//////////
/*  repeat(2000) @(posedge tb_ip_wb_clk);
    tb_ip_wb_w_din <= 32'h000f_0000;//CREG[4]= ONLY AUXP/N CAN BE
CONFIGURED THIS WAY
    tb_ip_wb_w_sel <= 4'b1111;
    tb_ip_wb_w_addr <= 6'b000_100;
    tb_ip_wb_w_we <=1'b1;
    tb_ip_wb_w_stb <= 1'b1;
    @(posedge tb_ip_wb_clk);//clear
    tb_ip_wb_w_din <= 32'h0000_000;
    tb_ip_wb_w_sel <= 4'b0000;
    tb_ip_wb_w_addr <= 6'b000_000;
    tb_ip_wb_w_we <=1'b0;
    tb_ip_wb_w_stb <= 1'b0;*/

```

APPENDIX B

```
//=====ENABLE INTERRUPT=====
repeat(50) @(posedge tb_ip_wb_clk);
tb_ip_ADCIE<=1'b1;
///configure to have only temp and vccint alarm signal enable
tb_ip_wb_w_din <= 32'h0000_2ff8;
tb_ip_wb_w_sel <= 4'b1100;
tb_ip_wb_w_addr <= 6'b000_000;
tb_ip_wb_w_we <=1'b1;
tb_ip_wb_w_stb <= 1'b1;
@(posedge tb_ip_wb_clk);//clear
tb_ip_wb_w_din <= 32'h0000_000;
tb_ip_wb_w_sel <= 4'b0000;
tb_ip_wb_w_addr <= 6'b000_000;
tb_ip_wb_w_we <=1'b0;
tb_ip_wb_w_stb <= 1'b0;
@(posedge tb_ip_wb_clk);
///configure temp min threshold to have 70,it can reset the alarm for temp
tb_ip_wb_w_din <= 32'h0000_ae50;
tb_ip_wb_w_sel <= 4'b1100;
tb_ip_wb_w_addr <= 6'b000_101;
tb_ip_wb_w_we <=1'b1;
tb_ip_wb_w_stb <= 1'b1;
@(posedge tb_ip_wb_clk);//clear
tb_ip_wb_w_din <= 32'h0000_000;
tb_ip_wb_w_sel <= 4'b0000;
tb_ip_wb_w_addr <= 6'b000_000;
tb_ip_wb_w_we <=1'b0;
tb_ip_wb_w_stb <= 1'b0;

repeat(12000) @(posedge tb_ip_wb_clk);//can show difference between enable
and disable
tb_ip_ADCIE<=1'b0;

end

endmodule
```


Testbench for Integration Level Functional Test and Multiple Exception Test with assembly language.

```

`timescale 1ns / 1ps
`default_nettype none
define demo006_ADC
//define multiple_exception
`ifndef demo006_ADC
    `define TEST_CODE_PATH_DUT "demo006_ADC_v2mem_02program.hex"
    `define EXC_HANDLER_DUT "demo006_ADC_mem_03exc_handler.hex"
`endif
/*
`ifndef multiple_exception
    `define TEST_CODE_PATH_DUT "Multiple_Exception+interrupt.hex"
    `define EXC_HANDLER_DUT "exception_handler.hex"
    `define TEST_CODE_PATH_CLIENT "Uart+Spi Transmit to DUT.hex"
    `define EXC_HANDLER_CLIENT "exception_handler.hex"
`endif
*/
module tb_r32_pipeline();
//declaration
//===== INPUT =====
//System signal
reg tb_u_clk;
reg tb_u_rst;
//~~~~~
wire tb_u_spi_mosi_dut;
wire tb_u_spi_miso_dut;
wire tb_u_spi_sclk_dut;
wire tb_u_spi_ss_n_dut;

wire tb_u_fc_sclk_dut;
wire tb_u_fc_ss_dut;
wire tb_u_fc_MOSI_dut;
wire tb_u_fc_MISO1_dut;
wire tb_u_fc_MISO2_dut;
wire tb_u_fc_MISO3_dut;
wire tb_ua_tx_rx_dut;
wire tb_ua_RTS_dut, tb_ua_CTS_dut;
wire[31:0] tb_u_GPIO_dut;
//~~~~~
wire tb_u_spi_mosi_client;
wire tb_u_spi_miso_client;
wire tb_u_spi_sclk_client;
wire tb_u_spi_ss_n_client;

wire tb_u_fc_sclk_client;
wire tb_u_fc_ss_client;
wire tb_u_fc_MOSI_client;
wire tb_u_fc_MISO1_client;
wire tb_u_fc_MISO2_client;
wire tb_u_fc_MISO3_client;
wire tb_ua_tx_rx_client;
wire tb_ua_RTS_client, tb_ua_CTS_client;
wire[31:0] tb_u_GPIO_client;

```

APPENDIX B

```
crisc c_risc_dut(
//***** INSTANTIATION *****
//===== INPUT =====
//GPIO
.urisc_GPIO(tb_u_GPIO_dut),

//SPI controller
.uiorisc_spi_mosi(tb_u_spi_mosi_dut),
.uiorisc_spi_miso(tb_u_spi_miso_dut),
.uiorisc_spi_sclk(tb_u_spi_sclk_dut),
.uiorisc_spi_ss_n(tb_u_spi_ss_n_dut),

//UART controller
.uorisc_ua_tx_data(tb_ua_tx_rx_dut),
//.uorisc_ua_rts(tb_ua_RTS_dut),
.uirisc_ua_rx_data(tb_ua_tx_rx_client),
//.uirisc_ua_cts(tb_ua_CTS_dut),

//FLASH controller
.uorisc_fc_sclk(tb_u_fc_sclk_dut),
.uiorisc_fc_MOSI(tb_u_fc_MOSI_dut),
.uirisc_fc_MISO1(tb_u_fc_MISO1_dut),
.uirisc_fc_MISO2(tb_u_fc_MISO2_dut),
.uirisc_fc_MISO3(tb_u_fc_MISO3_dut),
.uorisc_fc_ss(tb_u_fc_ss_dut),

// System signal
.uirisc_clk_100mhz(tb_u_clk),
.uirisc_rst(tb_u_rst));

//-----
s25fl128s
SPI_flash_dut(
.SI(tb_u_fc_MOSI_dut), //IO0
.SO(tb_u_fc_MISO1_dut), //IO1
.SCK(tb_u_fc_sclk_dut),
.CSNeg(tb_u_fc_ss_dut),
.RSTNeg(tb_u_rst),
.WPNeg(tb_u_fc_MISO2_dut), //IO2
.HOLDNeg(tb_u_fc_MISO3_dut));

//=====
=====

crisc
c_risc_client(
//***** INSTANTIATION *****
//===== INPUT =====
//GPIO
.urisc_GPIO(tb_u_GPIO_client),

//SPI controller
.uiorisc_spi_mosi(tb_u_spi_mosi_client),
.uiorisc_spi_miso(tb_u_spi_miso_client),
.uiorisc_spi_sclk(tb_u_spi_sclk_client),
.uiorisc_spi_ss_n(tb_u_spi_ss_n_client),
```

APPENDIX B

```
//UART controller
.uorisc_ua_tx_data(tb_ua_tx_rx_client),
//.uorisc_ua_rts(tb_ua_RTS_client),
.uirisc_ua_rx_data(tb_ua_tx_rx_dut),
//.uirisc_ua_cts(tb_ua_CTS_client),

//FLASH controller
.uorisc_fc_sclk(tb_u_fc_sclk_client),
.uiorisc_fc_MOSI(tb_u_fc_MOSI_client),
.uirisc_fc_MISO1(tb_u_fc_MISO1_client),
.uirisc_fc_MISO2(tb_u_fc_MISO2_client),
.uirisc_fc_MISO3(tb_u_fc_MISO3_client),
.uorisc_fc_ss(tb_u_fc_ss_client),

// System signal
.uirisc_clk_100mhz(tb_u_clk),
.uirisc_rst(tb_u_rst));

//-----
s25fl128s
SPI_flash_client(
.SI(tb_u_fc_MOSI_client), //IO0
.SO(tb_u_fc_MISO1_client), //IO1
.SCK(tb_u_fc_sclk_client),
.CSNeg(tb_u_fc_ss_client),
.RSTNeg(tb_u_rst),
.WPNeg(tb_u_fc_MISO2_client), //IO2
.HOLDNeg(tb_u_fc_MISO3_client));

assign tb_u_spi_mosi_dut = tb_u_spi_mosi_client;
assign tb_u_spi_miso_dut = tb_u_spi_miso_client;
assign tb_u_spi_ss_n_dut = tb_u_spi_ss_n_client;
assign tb_u_spi_sclk_dut = tb_u_spi_sclk_client;

assign tb_ua_CTS_dut = tb_ua_RTS_client;
assign tb_ua_CTS_client = tb_ua_RTS_dut;
```

```

//*****Clock*****
initial tb_u_clk = 1'b1;
always #5 tb_u_clk =~ tb_u_clk;

initial begin
//For client: copy the right test program and exc handler into FPGA flash.
//$readmemh(`EXC_HANDLER_CLIENT, tb_r32_pipeline.SPI_flash_client.Mem);
//$readmemh(`TEST_CODE_PATH_CLIENT, tb_r32_pipeline.SPI_flash_client.Mem);

//For dut: copy the right test program and exc handler into FPGA flash.
$readmemh(`EXC_HANDLER_DUT, tb_r32_pipeline.SPI_flash_dut.Mem);
$readmemh(`TEST_CODE_PATH_DUT, tb_r32_pipeline.SPI_flash_dut.Mem);
//test instruction 1st

//2nd test IO seperately
//SPI
//UART
//GPIO
//ADC without client so comment out
//3rd exception handler

tb_u_rst = 1'b1;
repeat(1)@(posedge tb_u_clk);
tb_u_rst = 1'b0;
repeat(30000)@(posedge tb_u_clk);
tb_u_rst = 1'b1;
repeat(12000000)@(posedge tb_r32_pipeline.c_risc_dut.urisc_clk);
end
endmodule

```

Appendix C: FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Jan, 2022	Study week no.: 2
Student Name & ID: Tan Yan Kai 18ACB03478	
Supervisor: Mr Teoh Shen Khang	
Project Title: Design of an ADC Controller for 5-stage Pipeline RISC32 Microprocessor	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Pick up where I left in FYP 1.
- Revise FYP 1.

2. WORK TO BE DONE

- Make the changes that I have planned to do in FYP1
- Modify microarchitecture and add a new block

3. PROBLEMS ENCOUNTERED

- Not sure exactly where to move(integrate) the XADC

4. SELF EVALUATION OF THE PROGRESS

- Progressing well
- Should be able to figure out after consultation

TEOH

Supervisor's signature

Tan Yan Kai

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Jan, 2022	Study week no.: 4
Student Name & ID: Tan Yan Kai 18ACB03478	
Supervisor: Mr Teoh Shen Khang	
Project Title: Design of an ADC Controller for 5-stage Pipeline RISC32 Microprocessor	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Fix a few issues I didn't notice.
- A new receiver block is designed
- Modifications done on drawing.

2. WORK TO BE DONE

- Check on report to see the information match with recent modifications.

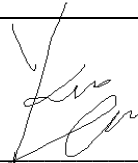
3. PROBLEMS ENCOUNTERED

4. SELF EVALUATION OF THE PROGRESS

- Progressing well

7E07

Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Jan, 2022	Study week no.: 6
Student Name & ID: Tan Yan Kai 18ACB03478	
Supervisor: Mr Teoh Shen Khang	
Project Title: Design of an ADC Controller for 5-stage Pipeline RISC32 Microprocessor	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Update the report to match new modifications

2. WORK TO BE DONE

- Try to add new chapter about multiple IO system functional Test

3. PROBLEMS ENCOUNTERED

- Not sure how to complete the new chapter like what can be mentioned and discussed

4. SELF EVALUATION OF THE PROGRESS

- Progressing well
- Should be able to figure out after consultation.

Teoh

Supervisor's signature

Tan Yan Kai

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Jan, 2022	Study week no.: 8
Student Name & ID: Tan Yan Kai 18ACB03478	
Supervisor: Mr Teoh Shen Khang	
Project Title: Design of an ADC Controller for 5-stage Pipeline RISC32 Microprocessor	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Make the drawing more detail and correct.

2. WORK TO BE DONE

- Check on report format and content with the help of guideline and checklist

3. PROBLEMS ENCOUNTERED

4. SELF EVALUATION OF THE PROGRESS

- Progressing well

7E074

Supervisor's signature

[Handwritten Signature]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Jan, 2022	Study week no.: 10
Student Name & ID: Tan Yan Kai 18ACB03478	
Supervisor: Mr Teoh Shen Khang	
Project Title: Design of an ADC Controller for 5-stage Pipeline RISC32 Microprocessor	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Finish draft report

2. WORK TO BE DONE

- Double check the report format
- Ask supervisor for signature and Turnitin account

3. PROBLEMS ENCOUNTERED

4. SELF EVALUATION OF THE PROGRESS

- Progressing well

Teoh

Supervisor's signature

Tan Yan Kai

Student's signature

Appendix D: POSTER



UNIVERSITI TUNKU ABDUL RAHMAN
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

DESIGN OF AN ADC CONTROLLER FOR 5 STAGE PIPELINE RISC32 PROCESSOR

Introduction

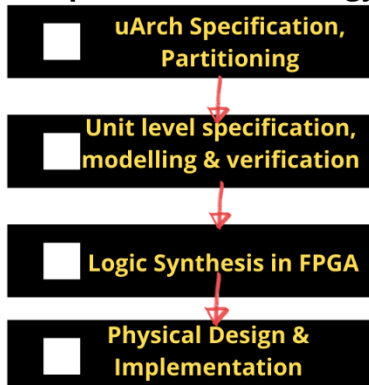
The project is to design and integrate an ADC controller unit for the RISC32 pipeline processor that has been developed previously. The ADC device that is used in this project is XADC model ug480 and the datasheet provided by Xilinx is going to be used to understand the attribute and properties. The specifications of the ADC controller unit and the instantiation of XADC will be functionally verified by developing test benches.

Functionality/Feature

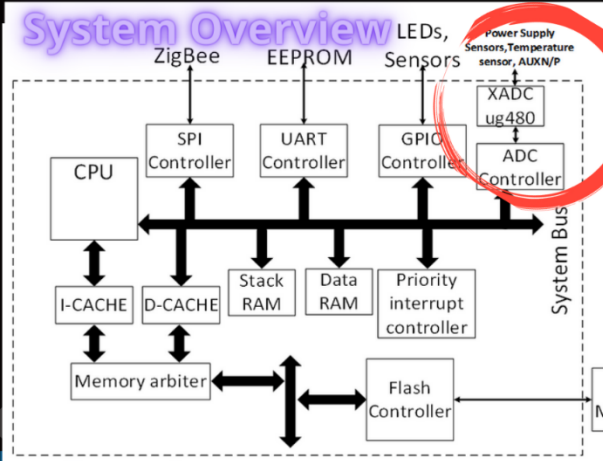
- ADC controller unit has 47 IO registers that can support major functionality in XADC.
- ADC controller unit can handle the alarm signal from the XADC (whenever sensors are having undesired values) by triggering IRQ to run specific ISR.
- With XADC integrated, RISC32 can now support one dedicated analog input (VP/VN) and 16 auxiliary analog inputs (VAUXP/VAUXN) for data conversion.

Method

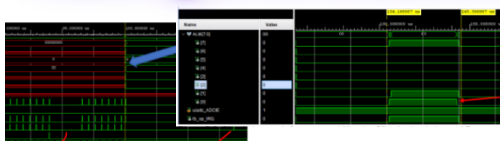
Top-down methodology



System Overview



Testing & Verification



Conclusions

The objectives of this project have been achieved. The ADC controller unit is successfully developed and validated with its functionality verified. Successful multiple IO system functional tests are also carried out to prove that the integration is completed.

Appendix E: PLAGIARISM CHECK RESULT

18ACB03478_FYP2			
ORIGINALITY REPORT			
8%	7%	2%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS
PRIMARY SOURCES			
1	eprints.utar.edu.my Internet Source	3%	
2	dgidb.genome.wustl.edu Internet Source	1%	
3	www.xilinx.com Internet Source	<1%	
4	www.coursehero.com Internet Source	<1%	
5	Submitted to Manchester Metropolitan University Student Paper	<1%	
6	funglee.github.io Internet Source	<1%	
7	Submitted to University of Nebraska, Lincoln Student Paper	<1%	
8	Submitted to University of Wollongong Student Paper	<1%	
9	Wei-Pau Kiat, Kai-Ming Mok, Wai-Kong Lee, Hock-Guan Goh, Ramachandra Achar. "An	<1%	

energy efficient FPGA partial reconfiguration based micro-architectural technique for IoT applications", Microprocessors and Microsystems, 2020

Publication

10	vinoddoriwal.wordpress.com Internet Source	<1 %
11	Submitted to Cyprus Academic Library Consortium Student Paper	<1 %
12	Rothke, Ben. "A Look at the Advanced Encryption Standard (AES)", Information Security Management Handbook Sixth Edition, 2007. Publication	<1 %
13	its1pc19.epfl.ch Internet Source	<1 %
14	Submitted to Chester College of Higher Education Student Paper	<1 %
15	Submitted to Universiti Tunku Abdul Rahman Student Paper	<1 %
16	fict.utar.edu.my Internet Source	<1 %
17	doku.pub Internet Source	<1 %

APPENDIX E

18	Submitted to Punjab Technical University Student Paper	<1 %
19	china.xilinx.com Internet Source	<1 %
20	www.springerprofessional.de Internet Source	<1 %
21	Submitted to UT, Dallas Student Paper	<1 %
22	patentimages.storage.googleapis.com Internet Source	<1 %
23	khaledrefaat.com Internet Source	<1 %
24	Submitted to South Bank University Student Paper	<1 %
25	Submitted to Southern New Hampshire University - Continuing Education Student Paper	<1 %
26	www.jastt.org Internet Source	<1 %
27	www.cyberjournals.com Internet Source	<1 %
28	www.ece.ucf.edu Internet Source	<1 %

APPENDIX E

Exclude quotes	On	Exclude matches	< 8 words
Exclude bibliography	On		

Form iad-FM-IAD-005

Form Title: Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



**FACULTY OF INFORMATION AND COMMUNICATION
TECHNOLOGY**

Full Name(s) of Candidate(s)	TAN YAN KAI
ID Number(s)	18ACB03478
Programme / Course	Bachelor Of Information Technology (Honours) Computer Engineering
Title of Final Year Project	Design of an ADC Controller for 5-stage Pipeline RISC32 Microprocessor

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceed the limits approved by UTAR)
Overall similarity index: <u>8</u> % Similarity by source Internet Sources: <u>7</u> % Publications: <u>2</u> % Student Papers: <u>4</u> %	
Number of individual sources listed of more than 3% similarity: <u>0</u>	
Parameters of originality required, and limits approved by UTAR are as Follows: (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note: Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

TEOH

Signature of Supervisor

Signature of Co-Supervisor

Name: TEOH SHEN KHANG

Name: _____

Date: 18 April 2022

Date: _____

Appendix F: FYP 2 CHECKLIST**UNIVERSITI TUNKU ABDUL RAHMAN****FACULTY OF INFORMATION & COMMUNICATION
TECHNOLOGY (KAMPAR CAMPUS)****CHECKLIST FOR FYP2 THESIS SUBMISSION**

Student Id	18ACB03478
Student Name	Tan Yan Kai
Supervisor Name	Mr. Teoh Shen Khang

TICK (√)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Front Plastic Cover (for hardcopy)
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

***Include this form (checklist) in the thesis (Bind together as the last page)**

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date: 15 April 2022