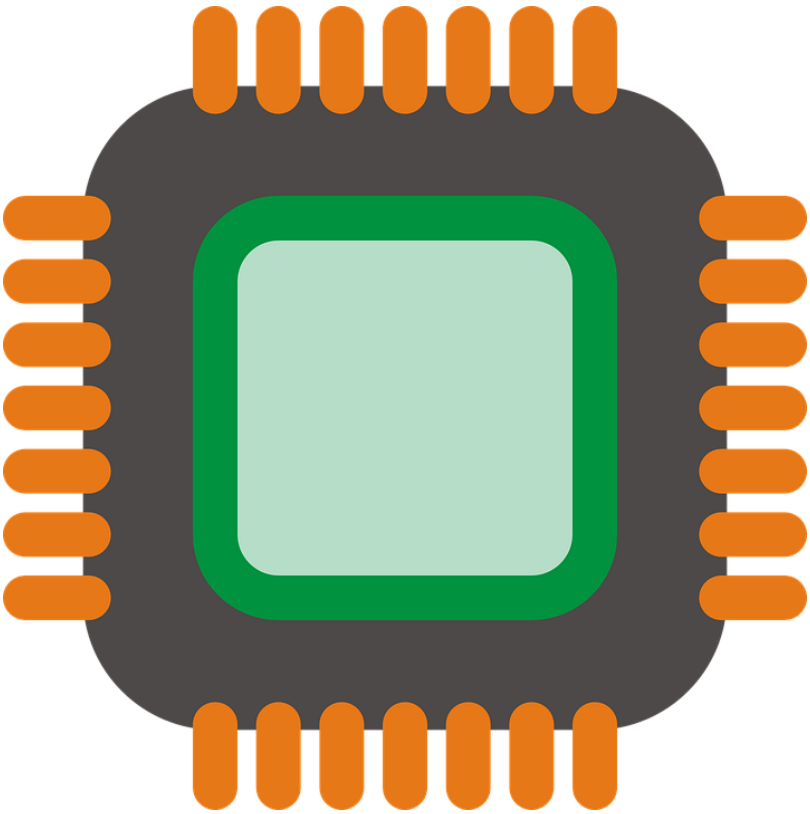
Rowan University

Leg V8 Processor Datasheet

WKMR-PL641

Scrumbody Help Me Think of a Name
Nick Woodward, David Russo, Nick Klein, Kyle McKeown
February 11- May 5, 2018



FEATURES

- Generation One -

- WKMR-PL641 -

- 64 bit Processor -

- Supports 32 unique Instructions-

- GPIO Support -

- PWM Timer -

- 45 MHz Maximum Clock Speed -

- 1.4 KiloBytes SRAM -

TABLE OF CONTENTS

1. Register File…….....5

1.1. Register File Description……5

1.2. Register File Schematic…….. 5

1.3. Register File Inputs/Outputs……... 6

1.4. Register File Testbench…….. 6

2. ALU……. 7

2.1. ALU Description……7

2.2. ALU Schematic…….. 7

2.3. ALU Inputs/Outputs……... 8

2.4. ALU Testbench…….. 9

3. Instruction Set……..... 10

3.1. Instruction Set Summary ……. 10

3.2. Instruction Set Format……. 10

3.3. Instruction Set terminology…….. 11

3.4. Instruction Tables……. 12

4. Memory……. 19

4.1. RAM……. 19

4.1.1. RAM Description……19

4.1.2. RAM Schematic…….. 19

4.1.3. RAM Inputs/Outputs……... 19

4.2. ROM……. 19

4.3. Memory Mapping……. 20

4.3.1. Memory Map Description……... 20

4.3.2. Memory Map Block Diagram……. 20

4.3.3. Peripheral Mapping……. 20

5. Program Counter……... 21

5.1. Program Counter Description……... 21

5.2. Program Counter Schematic……. 21
5.3. Program Counter Inputs/Outputs…….. 22
5.4. Program Counter Testbench…… 22
6. Datapath…… 23
6.1. Datapath Description……... 23
6.2. Datapath Schematic…….. 23
6.3. Datapath Inputs/Outputs……... 23
6.4. Explanation of the Modules Internal to the Datapath……... 26
7. Control Unit……... 27
7.1. Control Unit Description…….. 27
7.2. Control Unit Schematic…….28
7.3. Table of Control Unit Decoders……... 28
7.4. Control Unit Testbench……29
8. CPU……... 35
8.1. Description……....35
8.2. Schematic…… 35
8.3. Testbench…….. 35
9. Peripherals/Programs/Errata…… 37
9.1 PWM Peripheral Code…….....40
10. Appendices…….. 41
11. List of Figures…… 42
12. List of Tables……....43

1. REGISTER FILE

1.1. Register File Description

The Register File contains 32 64-bit registers, and is designed to access and write registers. Each register is accessed by a 5 bit address. Register 31 is hardwired to 64-bit binary zero. Three addresses are accessed at once: the destination address (DA), the select A signal (SA) and the select B signal (SB). 64-bit data is input and stored at the destination address. Since this module is comprised of registers, the Register File also has write, reset, and clock signals. Each of the three registers latch on the positive edge of the clock, or are reset to zero at the positive edge of the reset signal.

1.2. Register File Schematic

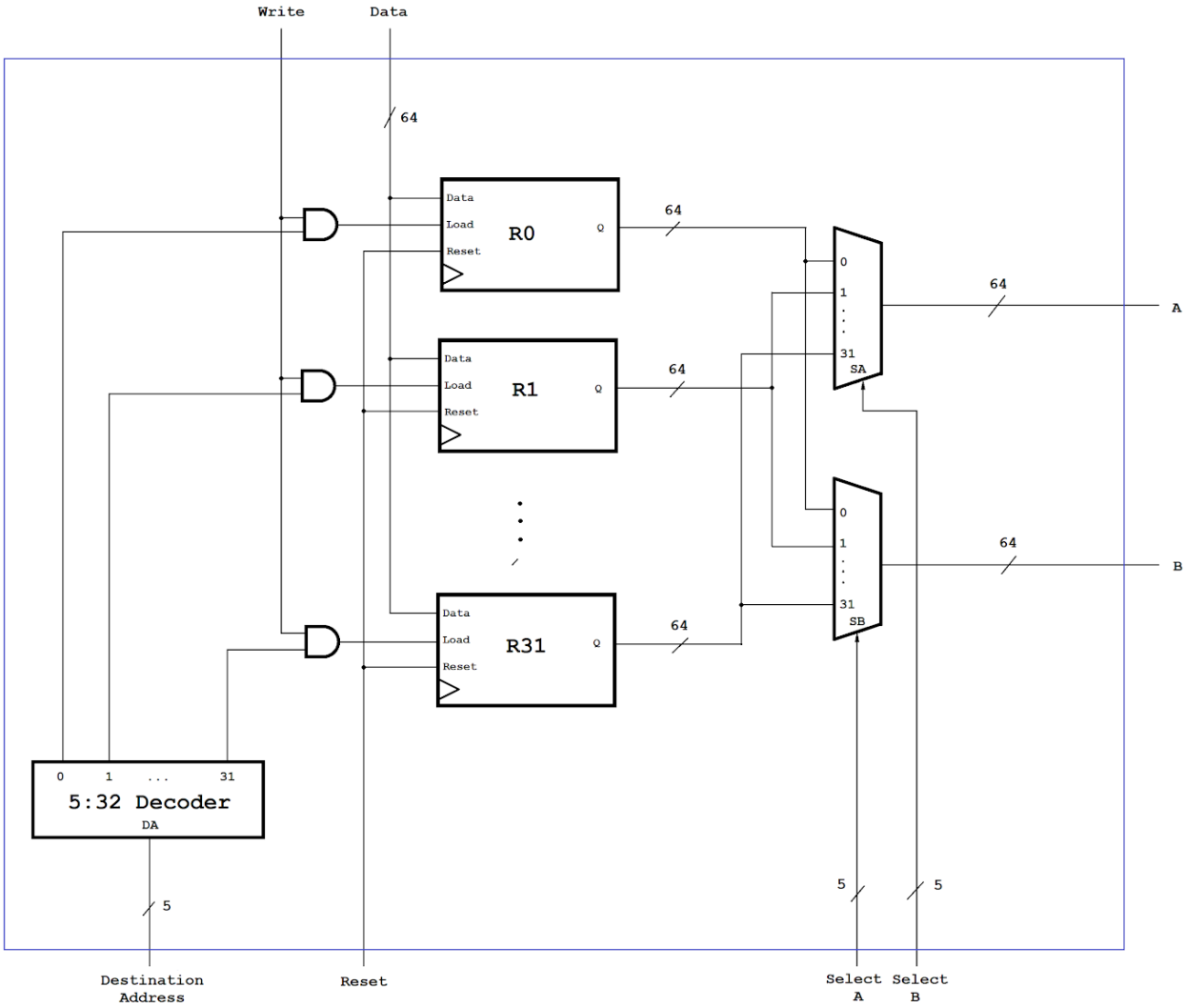


Fig.1.1 - Register Diagram

1.3: Register File Inputs/Outputs

Port type	Name	Bits	Description
Inputs	DA	5	Destination Address: Chooses what register to store the incoming data
	SA	5	Select A: Chooses what register to pull the data going to Output A
	SB	5	Select B: Chooses what register to pull the data going to Output B
	D	64	Data: The incoming data signal
	write	1	Controls whether incoming Data is written to a register
	reset	1	Resets every Register to 0
	clock	1	clock internal to the processor
Outputs	A	64	Output value of A
	B	64	Output value of B

Table 1.1 - Input/Output Table

1.4: Register File Testbench

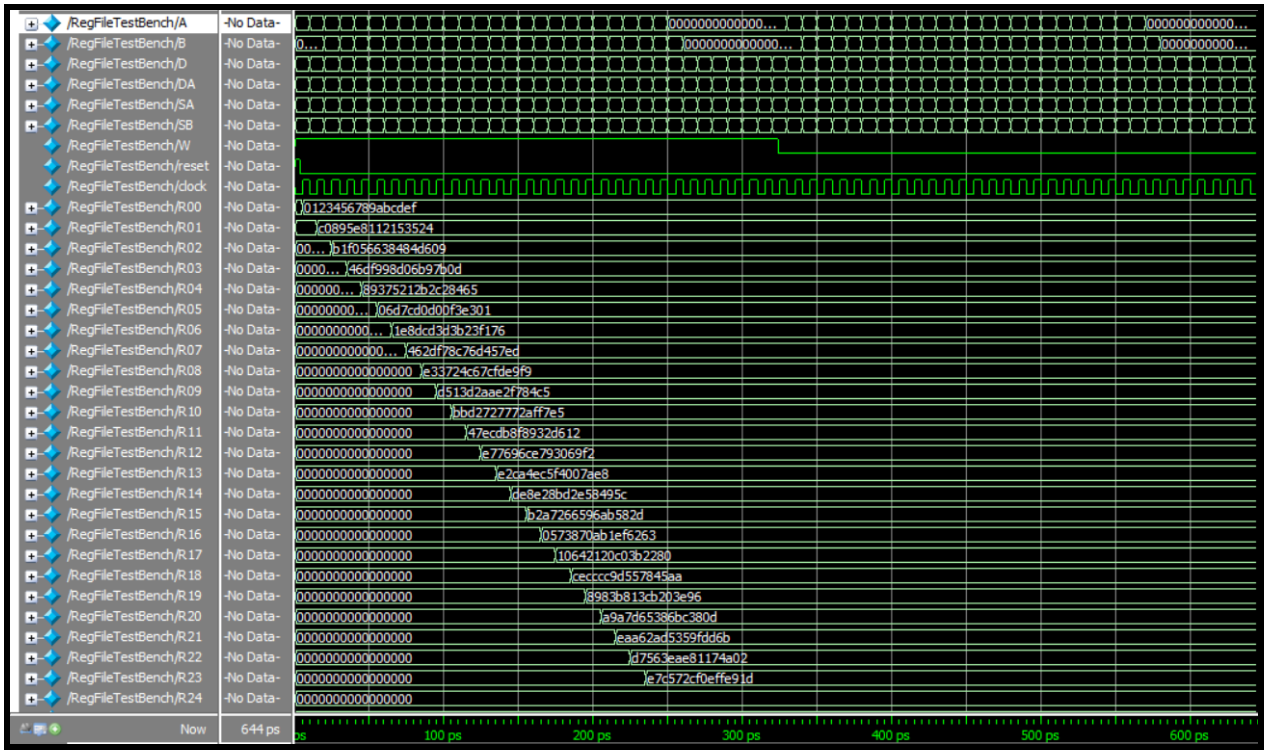


Fig. 1.2 - Register Testbench

The Register Testbench in *Fig 1.2* shows that at each positive clock edge the next successive register location can be written to. A random number was generated each clock cycle to store a different number each time. Once each register location was filled, the write signal was turned off until the test bench was told to stop. This shows that the write function worked perfectly fine.

2. ALU: (ARITHMETIC LOGIC UNIT)

2.1 ALU Description

The Arithmetic Logic Unit is a combinational circuit module that handles logical and arithmetic computations. With inputs A, B, and FS (function select), the ALU can perform a binary logical and arithmetic operations such as bitwise AND, OR, XOR, ADD, SUB, shift left, and shift right. The single bit cell shown in Figure 4 is replicated 64 times. There are four status bits if there is an overflow of bits, or if the output is zero. The test bench is generates random 64-bit hexadecimal values and compares it to an expected value. If the expected and actual outputs do not match, and exception is thrown.

2.2. ALU Schematic

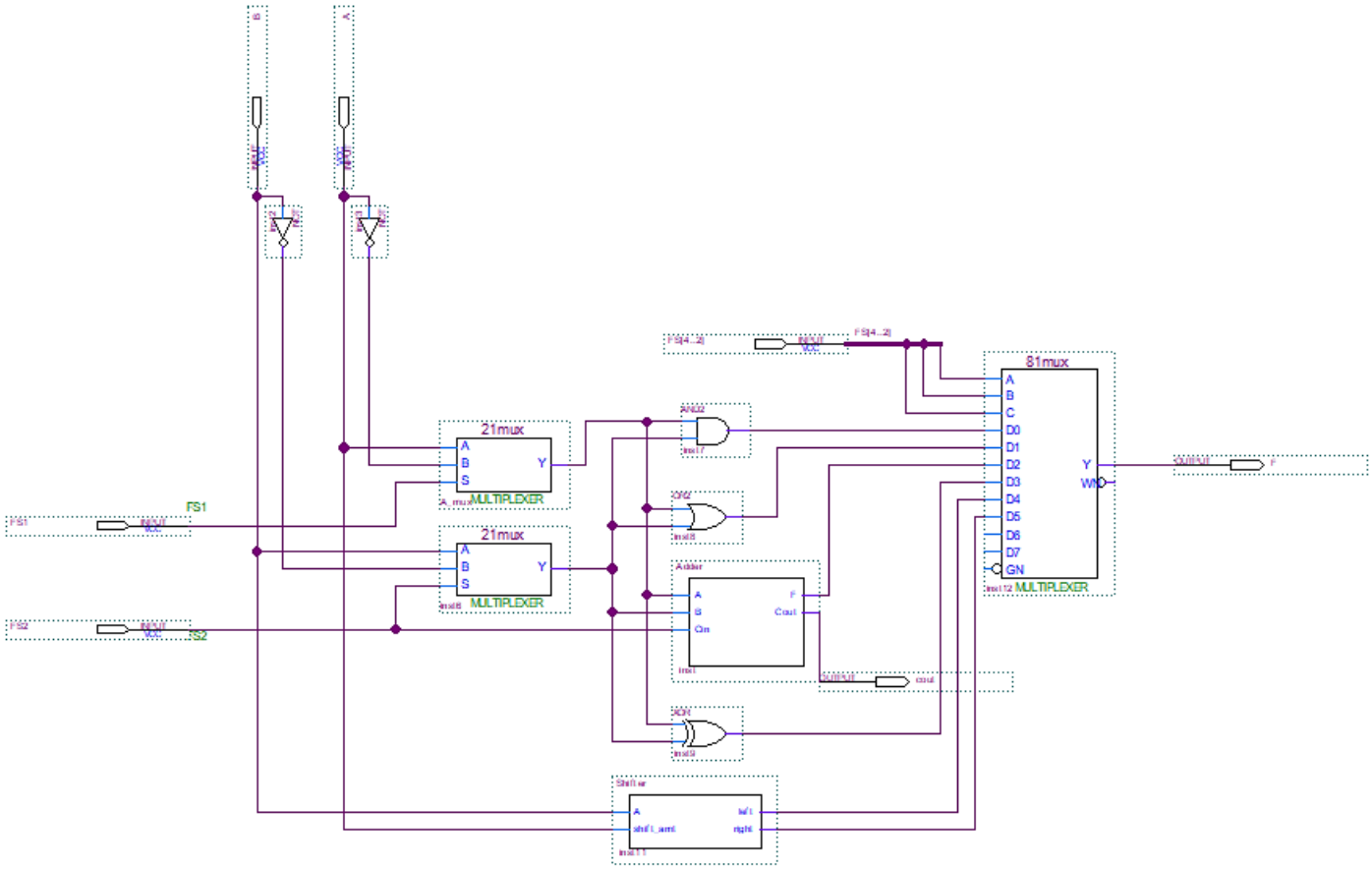


Fig. 2.1 - ALU Diagram

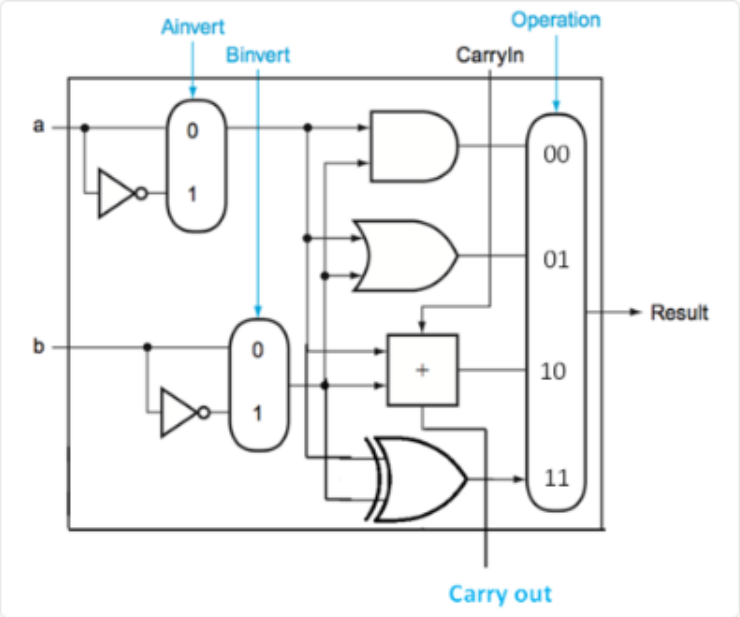


Fig 2.2 - Single ALU Cell

2.3. ALU Inputs/Outputs

Table 2.1 - Inputs and Outputs of the ALU

Port type	Name	Bits	Description
Inputs	a	64	Operand a
	b	64	Operand b
	functionSelect	5	Selects which function to perform on the operands functionSelect[0] is B invert functionSelect[1] is A invert functionSelect[4:2] is operation select including: 000: And 001: OR 010: ADD 011: XOR 100: Shift Left 101: Shift Right 110: not used (ground)
Outputs	F	64	Output of the ALU based on the operands and the function
	Status	4	{V, C, N, Z} status signals to send to the control unit V - Signed overflow C - When the result of the ALU has a carry output N - When the result of the ALU returns a negative value Z - When the resultant is zero

2.4. ALU Testbench

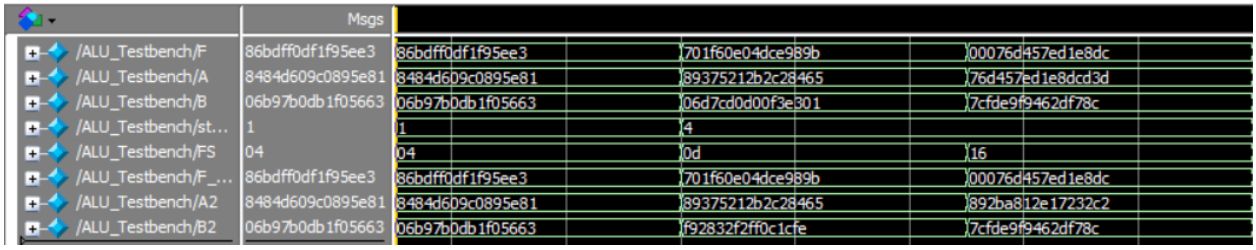


Fig. 2.3 - ALU Testbench

In order to test the ALU, a mock ALU was coded in the testbench to simulate the expected results of the ALU. Random 64-bit numbers numbers were generated and input into both ALU units, with a random function select. The two ALU's processed the same data in parallel and the results of each were compared. If the expected result of the mock ALU differed from the result of the ALU, an exception was thrown.

3. INSTRUCTION SET:

3.1 Instruction Set Summary

The LegV8 instruction set is comprised of two categories:

- Branching- Controlling Program Counter and order of operations.
- Data - Storing, retrieving, calculating, or manipulating data.

Each instruction is 32 bits, utilizing 6 different formats shown in *Table 3.1* . These formats are split into 6 to 11 bit opcodes which specify the actions taken by the processor, register addresses designated by Rx, and literal values which are used during those actions as program counter addresses, memory addresses, shift values, and immediate values. Instructions take one clock cycle to implement with the exception of MOVK.

3.2 Instruction Set Format

Table 3.1 - Instruction Formats

NAME		FIELDS						COMMENTS
FIELD SIZE		6 TO 11 BITS			2 BITS	5 BITS	5 BITS	ALL LEGv8 INSTRUCTIONS ARE 32 BITS
R-FORMAT	R	11 bit opcode	5 bit Rm	6 bit shamt		Rn	Rd	ARITHMETIC INSTRUCTION FORMAT
I-FORMAT	I	10 bit opcode	12 bit immediate (zf)			Rn	Rd	IMMEDIATE FORMAT
D-FORMAT	D	11 bit opcode	9 bit address (zf)		op2 (00)	Rn	Rt	DATA TRANSFER FORMAT
B-FORMAT	B	6 bit opcode	26 bit address (se)					UNCONDITIONAL BRANCH FORMAT
CB-FORMAT	CB	8 bit opcode	19 bits address (se)				Rt / cond	CONDITIONAL BRANCH FORMAT
IW-FORMAT	IW	9 bit opcode	2 bit sh*16	16 bit immediate (i)			Rd	WIDE IMMEDIATE FORMAT

3.3. Instruction Set Terminology

Table 3.2 - Opcode fields & terminology descriptions

Field	Description
Rm	Second Register data address for R
Rn	First Register data address for R, I, & D-Formats
Rd	Register storing address for R and IW-Formats
Rt	Register compared to zero for CBZ & CBNZ operations. Variable is also used to describe destination register for D format.
shamt	Bit spaces shifted left in R-format
zf	Literal value to specify memory location in the D format and a immediate value in I format.
i	Immediate value used in IW
se	Literal value specifying program counter branch address for B and CB-Format
X	Represents the range of the registers (ranging from 0 to 31) available for Rn, Rm, Rd, and Rt.

3.4. Instruction Set Table

Table 3.3 - ADD Instruction Table

ADD	**Add Rm and Rn**
Format	R
Opcode	10001011000
Syntax	ADD Rd, Rn, Rm
Operand(s)	$0 \leq X \leq 31$
Operation	Rd = Rm + Rn
Status Affected	None
Description	Value in register Rn is added together with the value in register Rm and stored in Rd.

Table 3.4 - SUB Instruction Table

SUB	**Subtract Rm from Rn**
Format	R
Opcode	11001011000
Syntax	SUB Rd, Rn, Rm
Operand(s)	$0 \leq X \leq 31$
Operation	Rd = Rn - Rm
Status Affected	None
Description	Value in register Rn is subtracted by the value in register Rm and stored in Rd.

Table 3.5 - ADDI Instruction Table

ADDI	**Add Rm and an immediate value**
Format	I
Opcode	1001000100
Syntax	ADDI Rd, Rm, zf
Operand(s)	$0 \leq X \leq 31$ $0 \leq zf \leq 4,095$
Operation	Rd = Rm + zf
Status Affected	None
Description	Adds immediate value zf to a value in register Rm and stores it in register Rd.

Table 3.6 - SUBI Instruction Table

SUBI	**Subtract an immediate value from Rm**
Format	I
Opcode	1101000100
Syntax	SUBI Rd, Rm, zf
Operand(s)	$0 \leq X \leq 31$ $0 \leq zf \leq 4,095$
Operation	Rd = Rm - zf
Status Affected	None
Description	Subtracts a value in register Rm by immediate value zf and stores it in register Rd.

Table 3.7 - ADDS Instruction Table

ADDS	**Add Rm and Rn, set flags**
Format	R
Opcode	10101011000
Syntax	ADDS Rd, Rn, Rm
Operand(s)	$0 \leq X \leq 31$
Operation	Rd = Rn + Rm
Status Affected	C, N, V, Z
Description	Value in register Rn is added with the value in register Rm. This value is stored in Rd and an appropriate signal (C, N, V, Z) is output from SL.

Table 3.8 - SUBS Instruction Table

SUBS	**Subtract Rn from Rm, set flags**
Format	R
Opcode	11101011000
Syntax	SUBS Rd, Rn, Rm
Operand(s)	$0 \leq X \leq 31$
Operation	Rd = Rn - Rm
Status Affected	C, N, V, Z
Description	Value in register Rn is subtracted by the value in register Rm. This value is stored in Rd and an appropriate signal (N, V, Z) is output from SL.

Table 3.9 - ADDIS Instruction Table

ADDIS	**Add Rm and an immediate value, set flags**
Format	I
Opcode	1011000100
Syntax	ADDIS Rd, Rm, zf
Operand(s)	$0 \leq X \leq 31$ $0 \leq zf \leq 4,095$
Operation	Rd = Rm + zf
Status Affected	C, N, Z
Description	Value in register Rm is added together with immediate value k. This value is stored in Rd and an appropriate signal (C, V, N, Z) is output from SL.

Table 3.10 - SUBIS Instruction Table

SUBIS	**Subtract an immediate value from Rm, set flags**
Function	I
Opcode	1111000100
Syntax	SUBIS Rd, Rm, zf
Operand(s)	$0 \leq X \leq 31$ $0 \leq zf \leq 4,095$
Operation	Rd = Rm - zf
Status Affected	N, V, Z
Description	Value in register Rm is subtracted by immediate value zf. This value is stored in Rd and an appropriate signal (N, V, Z) is output from SL

Table 3.11 - STUR Instruction Table

STUR	**Store data from Register to the Memory**
Format	D
Opcode	11111000000
Syntax	STUR Rt , [Rm, zf]
Operand(s)	$0 \leq X \leq 31$ $0 \leq zf \leq 511$
Operation	Memory[Rm + zf] = Rt
Status Affected	None
Description	Value in register Rm is added together with a constant value zf. This value is then stored in memory location M[Rm + zf].

Table 3.12 - LDUR Instruction Table

LDUR	**Load Data from Register to Memory**
Format	D
Opcode	11111000010
Syntax	LDUR Rt , [Rm, zf]
Operands	$0 \leq X \leq 31$ $0 \leq zf \leq 511$
Operation	Rd = Memory[Rm + zf]
Status Affected	None
Description	Value in memory M[Rm + zf] is loaded into register Rt.

Table 3.13 - MOVZ Instruction Table

MOVZ	**Store 16-bit immediate to Register, zero forward bits**
Format	IW
Opcode	110100101
Syntax	MOVZ Rd, i
Operands	$0 \leq X \leq 31$ $0 \leq i \leq 65535$
Operation	Rd = i
Status Affected	None
Description	Loads a 16 bit constant and pads the remaining bits with 0’s.

Table 3.14 - MOVK Instruction Table

MOVK	**Store 16-bit immediate to Register, leave forward bits unchanged**
Format	IW
Opcode	111100101
Syntax	MOVK Rd, i
Operands	$0 \leq X \leq 31$ $0 \leq i \leq 65535$
Operation	Rd = i
Status Affected	None
Description	Loads a 16 bit constant while keeping the remaining bits unchanged

Table 3.15 - AND Instruction Table

AND	**And Rm and Rn, bit by bit**
Format	R
Opcode	10001010000
Syntax	AND Rd, Rm, Rn
Operands	$0 \leq X \leq 31$
Operation	Rd = Rm & Rn
Status Affected	None
Description	Values in registers Rm and Rn are acted on by a bitwise AND operation and stored in Rd.

Table 3.16: ORR Instruction Table

ORR	**Or Rm and Rn, bit by bit**
Format	R
Opcode	10101010000
Syntax	OR Rd, Rm, Rn
Operands	$0 \leq X \leq 31$
Operation	Rd = Rm | Rn
Status Affected	None
Description	Values in register Rm and Rn are acted on by a bitwise OR operation and stored in Rd.

Table 3.17: EOR Instruction Table

EOR	**Exclusive or Rm and Rn, bit by bit**
Format	R
Opcode	11001010000
Syntax	EOR Rd, Rm, Rn
Operands	$0 \leq X \leq 31$
Operation	Rd = Rm ^ Rn
Status Affected	None
Description	Values in registers Rm and Rn are acted on by a bitwise EOR operation and stored in Rd.

Table 3.18: ANDI Instruction Table

ANDI	**And immediate and Rn, bit by bit**
Format	I
Opcode	1001001000
Syntax	ANDI Rd, Rn, zf
Operands	$0 \leq X \leq 31$ $0 \leq zf \leq 4095$
Operation	Rd = Rn & zf
Status Affected	None
Description	Value in register Rn is added together with immediate value zf and stored in register Rd.

Table 3.19: EORI Instruction Table

EORI	**Exclusive or immediate and Rn, bit by bit**
Format	I
Opcode	1101001000
Syntax	EORI Rd, Rn, zf
Operands	$0 \leq X \leq 31$ $0 \leq zf \leq 4095$
Operation	Rd = Rn ^ zf
Status Affected	None
Description	Value in register Rn and a immediate value zf are acted on by a bitwise XOR operation and stored in register Rd.

Table 3.20 - ORRI Instruction Table

ORRI	**Or immediate and Rn, bit by bit**
Format	I
Opcode	1011001000
Syntax	ORRI Rd, Rn, zf
Operands	$0 \leq X \leq 31$ $0 \leq zf \leq 4095$
Operation	Rd = Rn | zf
Status Affected	None
Description	Value in register Rn and a immediate value zf are acted on by a bitwise OR operation and stored in register Rd.

Table 3.21: ANDS Instruction Table

ANDS	**And Rm and Rn, set flags**
Format	R
Opcode	11101010000
Syntax	ANDS Rd, Rm, Rn
Operands	$0 \leq X \leq 31$
Operation	Rd = Rm & Rn
Status Affected	N, Z
Description	Values in registers Rm and Rn are acted on by a bitwise AND operation and stored in Rd. Condition codes are set after this operation.

Table 3.22 - ANDIS Instruction Table

ANDIS	**And immediate and Rn, bit by bit, set flags**
Format	I
Opcode	1111001000
Syntax	ANDIS Rd, Rn, zf
Operands	$0 \leq X \leq 31$ $0 \leq zf \leq 4095$
Operation	Rd = Rn & zf
Status Affected	N, Z
Description	Value in register Rn and a immediate value zf are acted on by a bitwise AND operation and stored in register Rd.

Table 3.23: LSR Instruction Table

LSR	**Logic Shift Right**
Format	R
Opcode	11010011010
Syntax	LSR Rd, Rn, shamt
Operands	$0 \leq X \leq 31$ $0 \leq shamt \leq 63$
Operation	Rd = Rn >> shamt
Status Affected	None
Description	Value in register Rn is shifted to the right by a literal value shamt and the result is stored in register Rd.

Table 3.24: LSL Instruction Table

LSL	**Logic Shift Left**
Format	R
Opcode	11010011011
Syntax	LSL Rd, Rn, shamt
Operands	$0 \leq X \leq 31$ $0 \leq shamt \leq 63$
Operation	Rd = Rn << shamt
Status Affected	None
Description	Value in register Rn is shifted to the left by a literal value shamt and the result is stored in register Rd.

Table 3.25: CBZ Instruction Table

CBZ	**Control & Branch if Zero**
Format	CB
Opcode	10110100
Syntax	CBZ Rt, se
Operands	$0 \leq X \leq 31$ $0 \leq se \leq 524,287$
Operation	if (Rt==0), go to PC + 4 + 4*se
Status Affected	Z
Description	If the value of Rt is equal to zero, than branch. Else, proceed to next instruction.

Table 3.26: CBNZ Instruction Table

CBNZ	**Control & Branch if not Zero**
Format	CB
Opcode	10110101
Syntax	CBNZ Rt, se
Operands	$0 \leq X \leq 31$ $0 \leq se \leq 524,287$
Operation	if (Rt!=0), go to PC + 4 + 4*se
Status Affected	Z
Description	If the value of Rt is not equal to zero, than branch. Else, proceed to next instruction.

Table 3.27: B.Conditional Table

B.COND	**Branch if condition is true**
Format	CB
Opcode	01010100
Syntax	B.cond se
Operands	$0 \leq se \leq 524,287$
Operation	If (condition true), PC + 4 + 4*se
Status Affected	C, V, N, Z
Description	If the condition code is true, than branch PC + 4 + 4*se. Else, proceed to next instruction.

Table 3.28: B Instruction Table

B	**Branch**
Format	B
Opcode	000101
Syntax	B se
Operands	$0 \leq se \leq 67108863$
Operation	PC + 4 + 4*se
Status Affected	None
Description	Branch to PC + 4 + 4*se.

Table 3.29: BR Instruction Table

BR	**Branch to register**
Format	R
Opcode	11010110000
Syntax	BR Rn
Operands	$0 \leq X \leq 31$
Operation	Branch to Rn
Status Affected	None
Description	Branch to register Rn

Table 3.30: Branch and Link Instruction Table

BL	**Branch and Link**
Format	B
Opcode	000101
Syntax	BL se
Operands	$0 \leq se \leq 67108863$
Operation	PC + 4 + 4*se
Status Affected	None
Description	Branch to specified addresses. Store address of original register.

4. MEMORY

4.1. RAM (Random Access Memory)

4.1.1. RAM Description

The RAM is a 256 x 64 bit volatile module that stores data from the databus through address assignment. Values from RAM can be called upon and placed on the databus to be stored in the Register File.

4.1.2. RAM Schematic

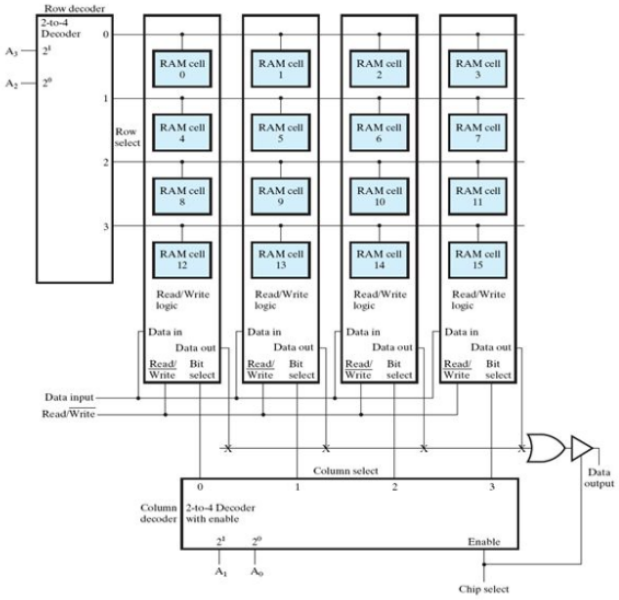


Figure 4.1 - Basic RAM Architecture

4.1.3. RAM Inputs/Outputs

Address: Address is formed as a result of A and some constant K. Dictates the storage location of the In value loaded into the RAM.

Write: Enables the RAM to be written too.

In: Input into the memory module originates from the B output of the Register File.

Out: Output of the memory module directly connects to the 4:1 MUX that controls what is loaded onto the databus.

4.2. ROM (Read Only Memory)

The ROM contains the instructions that the processor will execute. In the current state of the processor, there is no way to update or change the ROM without going into the code and changing it. The ROM takes a 16-bit signal from the Program Counter and outputs a 32-bit instruction to be decoded in the Control Unit and executed in the Datapath. These instructions are hardcoded in machine code, although assemblers can be used to assemble code to make the task of writing new ROM simpler.

4.3. Memory Mapping

4.3.3. Memory Map Description

The memory map above was chosen based on the amount of bits required for each memory based component. RAM was assigned address values 0x00000000 to 0x000000FF because it provides the 256 address locations required for RAM. The unused memory location space was allocated 0x00000100 to 0x000FFFFF. This space provides 1,048,319 unique address locations that can be utilized if the RAM is expanded to be a larger amount. ROM must have at least 67,108,864 unique address locations to perform B-format instructions. To account for this address space, ROM was allocated an address space ranging from 0x00100000 to 0x04FFFFFF, which provides 82,837,503 unique address locations. Lastly, the remaining amount of address spaces ranging from 0x05000000 to 0xFFFFFFFF were allocated to the special function registers.

4.3.2. Block Diagram

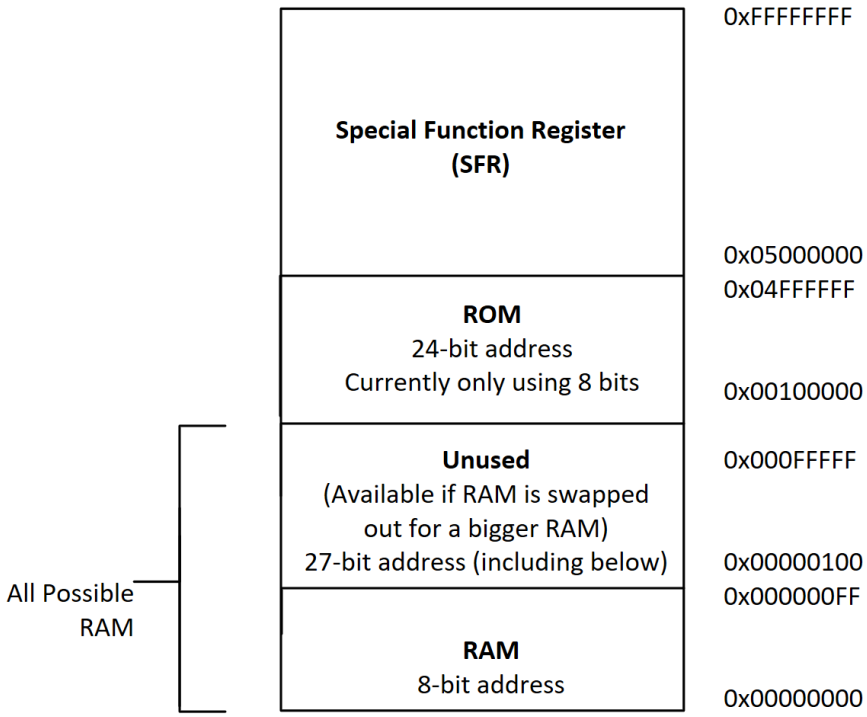


Figure 4.2 - Memory Map

4.3.3 Peripheral Mapping

Due to having only one Peripheral, Memory Addresses 0x05000000 - 0xFFFFFFFF is dedicated to the PWM Peripheral.

5. PROGRAM COUNTER

5.1. Program Counter Description

The Program Counter contains the address of the next instruction as each instruction is completed and selects between its mux inputs based on the instruction it has just completed. PC increments by one each clock cycle unless given a command from the control unit to become a predetermined constant.

5.2. Program Counter Schematic

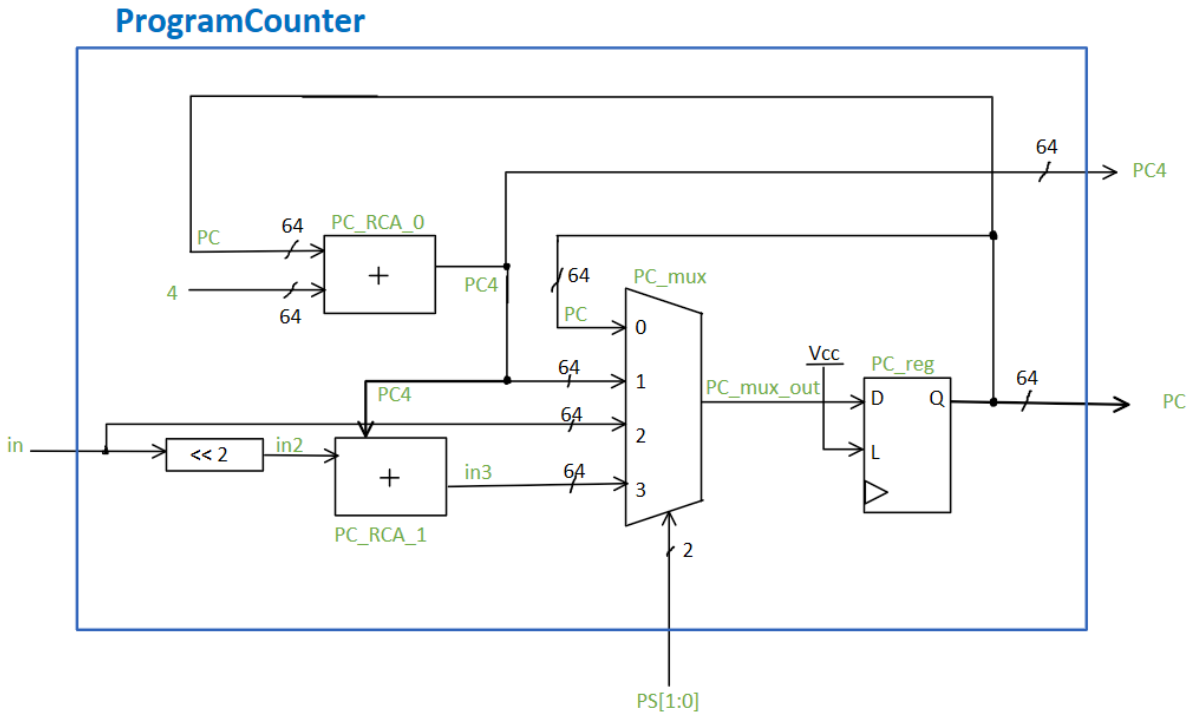


Figure 5.1 - Program Counter Diagram

5.3. Program Counter Inputs/Outputs

Table 5.1: Program Counter Inputs/Outputs

Port type	Name	Bits	Description
Inputs	in	64	Input that is used during branch instructions
	PS	2	Program Counter Select that chooses between four different outputs: 00: PC <= PC 01: In: Uses the input from the A_K Mux. 10: PC <= in 11: PC <= PC + 4 + in*4
Outputs	PC	64	Outputs the least significant 16 bits to the address of the ROM. Determines which instruction to execute.
	PC4	64	Program Counter plus four output; The PC4 output bus goes into the 4:1 mux that selects what goes on to the databus.

5.4. Program Counter Testbench

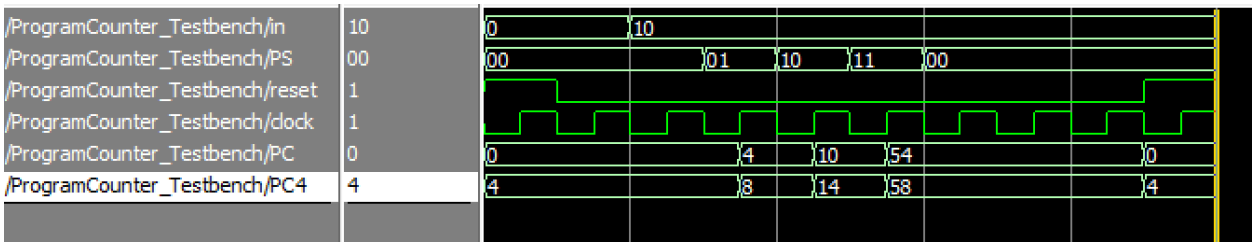


Figure 5.2 - Program Counter Testbench

The Program Counter Testbench was a small program to test the four PS scenarios. At the positive edge of each clock, a new PS value is latched into the Program Counter Register, PC_reg. At first, when reset is PC remains at zero until the positive clock edge after PS gets 2-bit binary 01. PC then gets PC + 4, and PC4 got four more than that value. Next, when PS is 2-bit binary 10 , PC gets 10, because that is the value of the input. Then, when PS is 2-bit binary 11, PC gets 10 + 4 + 10*4, or 54. Finally, when reset is high, the value of PC resets to zero.

6. DATAPATH

6.1. Datapath Description

The 64-bit Datapath benefits from the speed of a Harvard Architecture running pipelined instructions along a separate databus and instruction bus. The instruction set is designed using RISC architecture to allow for more instructions being processed per cycle. The capabilities of the Datapath include various functions such as performing arithmetic, logic, data storage.

6.2. Datapath Schematic

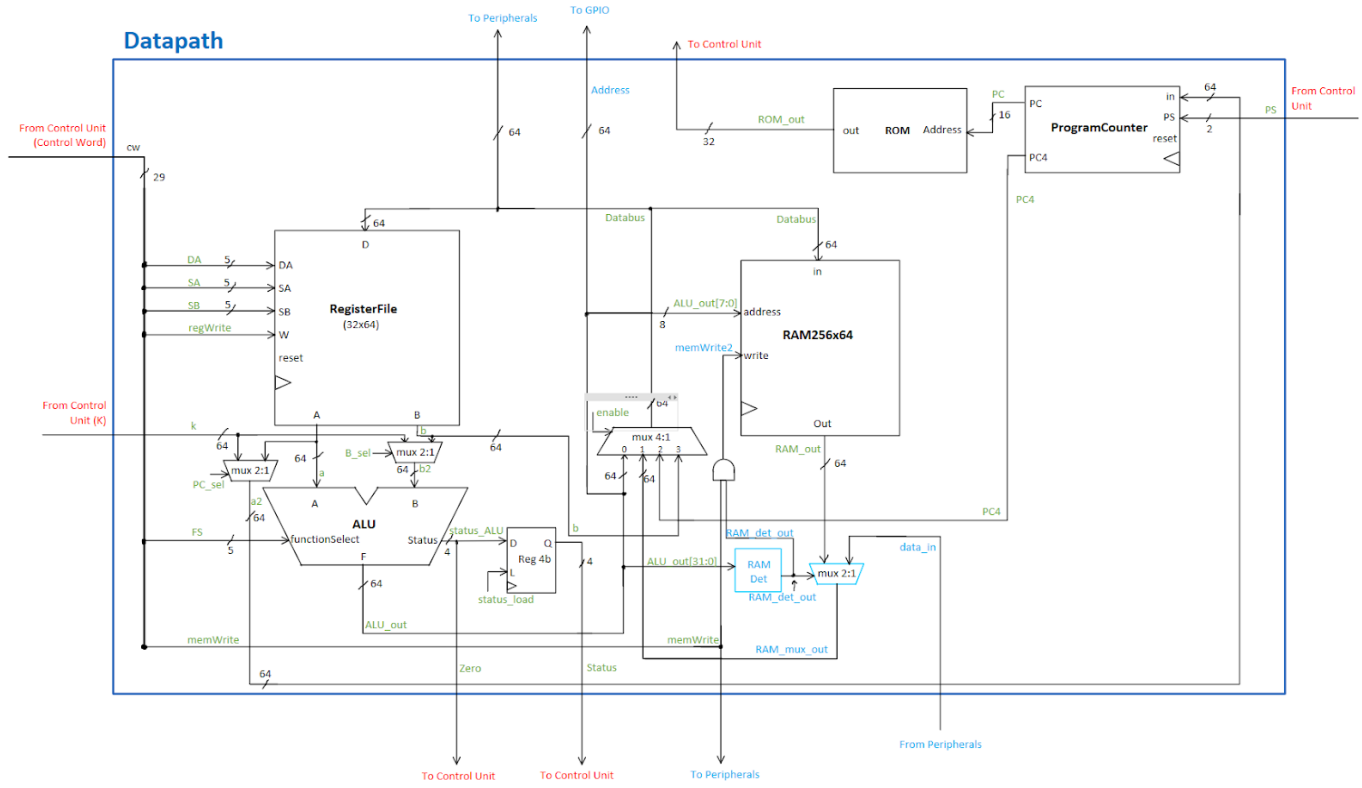


Figure 6.1 - Datapath Diagram

Key:

	Inputs/Outputs
	Internal Wires
	Changes to interface with Peripherals

6.3. Datapath Inputs/Outputs

Table 6.1 - Datapath Inputs/Outputs

Port type	Name	Bits	Description
Inputs	cw	29	Signal from the Control Unit that instructs how each block in the Datapath should behave
	k	64	Constant that is used in the ALU
	data_in	64	Data coming from Peripherals
	reset	1	Processor-wide asynchronous reset
	clock	1	Internal clock of the Processor
Outputs	Databus	64	Data output to the Peripherals
	Status	4	Output of the ALU to the Control unit for B.cond instructions
	I	32	Signal that outputs to the Control Unit and contains instructions for what functions to execute.
	Zero	1	Signal that is high when the output of the ALU is zero and outputs to the CBZ/CBNZ decoder in the Control Unit.
	Write	1	Controls whether or not to write to special function registers outside the processor.
	Address	8	Address bus that branches that is specifically allocated for Special Function Registers.

The following page deconstructs and explains each segment of the Control Word.

Table 6.2 - Control Signals

Control Word Index	Signal	Meaning
[28]	status_load	Loads the status bits into a 4-bit register. Should be high whenever a “set flags” instruction executes and low otherwise
[27]	B_sel	Select B: Used to select between the constant K and the register output B through a mux into the ALU.
[26]	PC_sel	Select PC input: Used to select between the constant K and the register output A through a mux into the Program Counter.
[25]	memWrite	Write to Ram: Determines whether memory will be written to.
[24]	regWrite	Write to Register: Determines whether the register will be written to.
[23:22]	enable	Databus input mux: Determines whether the output of the ALU, memory, PC4, or a ground (64-bit zero) will be put onto the databus. 00: ALU_out 01: RAM_out 10: PC4 11: B
[21:20]	PS	Program counter select: Selects which input is stored in the program counter register for that clock cycle. 00: PC 01: PC+4 10: Input 11: PC4+(Input*4)
[19:15]	FS	ALU Function select: Selects which function the ALU performs on its inputs. FS[4:2] selects the function, FS[1] inverts input A when it’s 1, and FS[0] inverts input B when it’s 1 000: A and B 001: A or B 010: arithmetic/add 011: A xor B 100: shift A, B places left 101: shift A, B places right 110: -not used- 111: -not used-
[14:10]	SB	Select output B: Selects which register location to make output B.
[9:5]	SA	Select output A: Selects which register location to make output A.
[4:0]	DA	Select Register input location: Selects which register location to store the data input if regWrite is on.

6.4. Explanation of the Modules Internal to the Datapath

Table 6.3 - Index of instantiated modules in the datapath

Module	Instance Name	General Purpose
RegisterFile	RegisterFile_inst	32 64-bit registers that functions as a temporary memory with asynchronous reset that is the smallest and most accessible of all memory.
ALU_rev2	ALU_inst	Function unit that takes in two 64-bit operands, a 5-bit function select, and outputs a 4-bit status and a 64-bit signal that represents data to load onto the databus or a memory address, depending on the instruction
RAM_det	RAM_det_inst	32-bit comparator that outputs a single bit signal - 1 if the address is greater than 0xFF and 0 otherwise. This module determines whether or not to write to memory external to the processor.
RAM256x64	RAM_inst	256-slot array of 64-bit registers. This static RAM is internal to the processor and can load and store values to and from the Register File.
RegisterNbit	status_reg	4-bit register that loads status bits from the ALU when a “set flags” instruction is executed
ProgramCounter	PC_inst	Keeps track of which instruction to execute next from ROM.
rom_case_nice	ROM_inst2	Program that loads the least-significant 16-bits of the first 8-registers in the Register File onto the GPIO board extension of the DE0 and displays the word “NICE!”
2:1 Mux - B select	coded with in-line if statement	Selects if the constant, k, or the B output of the Register file should feed into the B input of the ALU and controlled by B_sel signal from the Control Word. If B_sel is 0, then k feeds into the ALU, otherwise B passes through.
2:1 Mux - PC in select	coded with in-line if statement	Selects if the constant, k, or the A output of the Register file should feed into the “in” input of the Program Counter and controlled by the PC_sel signal from the Control Word. If PC_sel is 0, then PC in gets k, otherwise PC in gets A.
2:1 Mux - RAM out select	coded with in-line if statement	Selects if the output of RAM or data external to the processor should pass into an input of the 4:1 databus select mux and controlled by by the output of the RAM_det block. If RAM_det_out is 0, then RAM_mux_out gets RAM_out, and, otherwise gets data external to the processor.
4:1 Mux - Databus select	Coded with nested in-line if statement	Takes in four 64-bit inputs, a 2-bit select called enable, and outputs a one 64-bit based on the select.

7. CONTROL UNIT

7.1. Control Unit Description

The Control Unit consists of 10 different decoders, including D, I-Arithmetic, RI_Logic, IW, R_ALU, B, B_cond, BL, CBZ_CBNZ, and BR. The purpose of each decoder is to provide the correct control word depending on the instruction provided by the ROM. Certain decoders such as I-Arithmetic contain multiple functions that depend on the opcode contained within the input instruction while decoders such as B have a single function. One special case to note is RI_Logic, which combines the logic functions of R-format functions such as AND and ORR with I-format functions such as ANDI and ORRI. These two formats are combined in a single decoder due to their unique property of sharing the same instruction bits I[25:23] for all logic functions in both I-format and R-format. A single 2:1 MUX is implemented to control which set of instructions is chosen through Op[5]. A 1 bit register is implemented to choose which state the control unit is currently in. An output of 1 from this register enables the MOVK function while an output of 0 enables the use of every other function. *Fig 7.1* illustrates the sequential functionality of the state register in the Control Unit. Lastly, the 94 bit CW bus is split such that the most significant bit is the state bit, the following 64 bits represent the output bus k, and the remaining 29 bits make up the ControlWord bus.

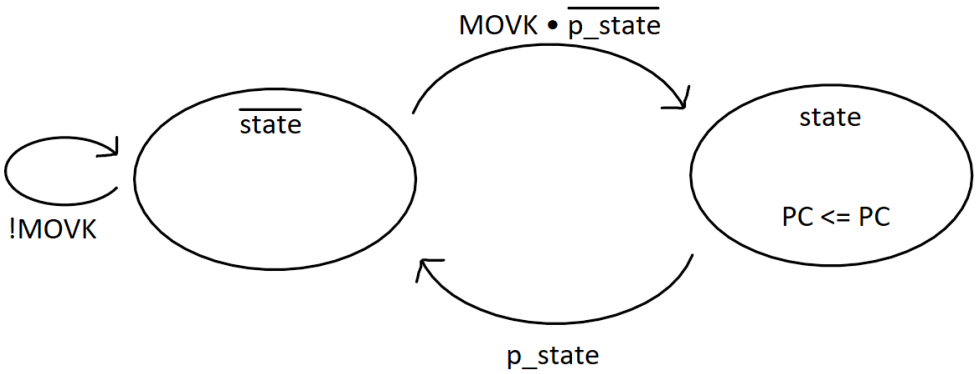


Figure 7.1 - Control Unit State Diagram

7.2. Control Unit Schematic

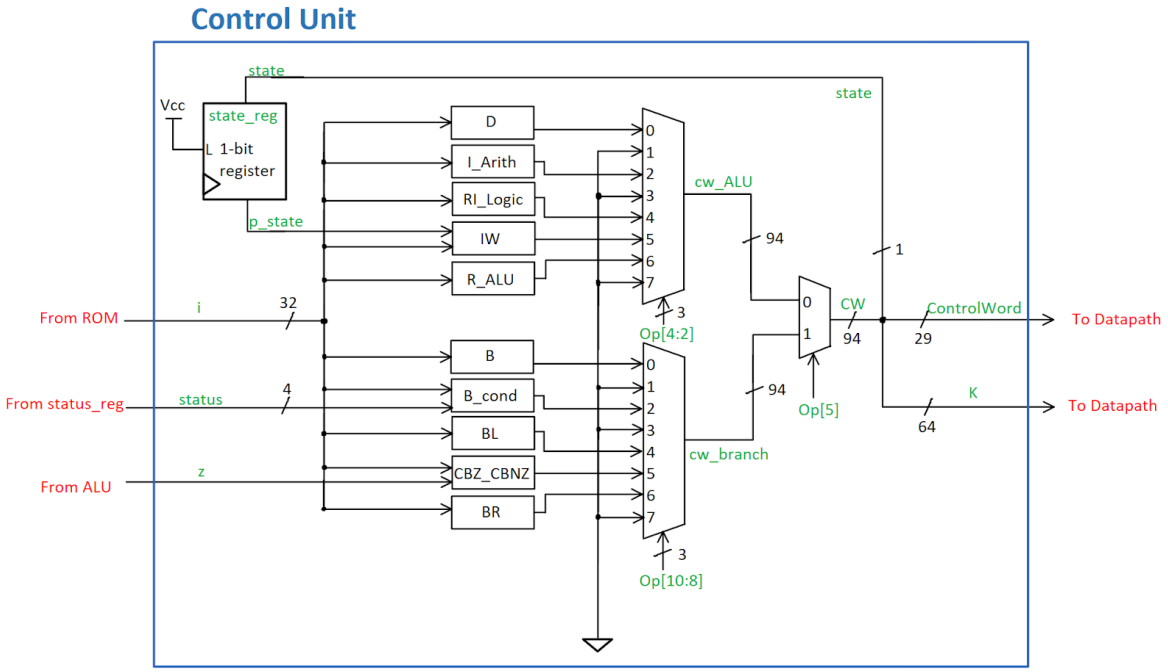


Figure 7.2 - Control Unit Diagram

7.3. Table of Control Unit Decoders

Table 7.1 - Index of Control Unit Decoders

Decoder	Op [4:2]	Functions	Requires Multiple States
D	000	LDUR, STUR	No
I - Arithmetic	010	ADDI, SUBI, ADDIS, SUBIS	No
RI_Logic	100	AND, OR, EOR, ANDI, ORRI, EORI, ANDS, ANDIS	No
IW	101	MOVK, MOVZ	Yes - MOVK
R_ALU	110	ADD, SUB, ADDS, SUBS, LSL, LSR	No
Decoder	**Op [10:8]**	**Functions**	**Requires Multiple States**
B	000	B	No
B_cond	010	B.cond	No
BL	100	BL	No
CBZ_CBNZ	101	CBZ, CBNZ	No
BR	110	BR	No

7.4. Control Unit Testbench

A testbench was written for each individual Control Unit Decoder to test if each of the blocks decoded signals correctly based on a given instruction format. An overall Control Unit Testbench was also coded to demonstrate the control unit output was coming from the correct decoder and the muxes worked correctly. With the exception of the B.cond testbench, the design for each testbench was similar: an instruction was hardcoded, and the control word was checked for accuracy. If a decoder handled more than one instruction, such as the R_ALU Decoder, all instructions that decoder handles were tested in increments of 10 ps.

The B.cond testbench design is similar to the ALU testbench design. A Mock B.cond Decoder was coded in the testbench. Random values are generated for i[3:0] and status[3:0]. i[4] is always 0 because i[4:0] represents the condition code, and no condition codes use i[4]. Values of PS1_exp from the mock B.cond and PS[1] are compared. An exception is thrown when the two values are not equal.

The following figures are screenshots of each Control Unit Testbench:

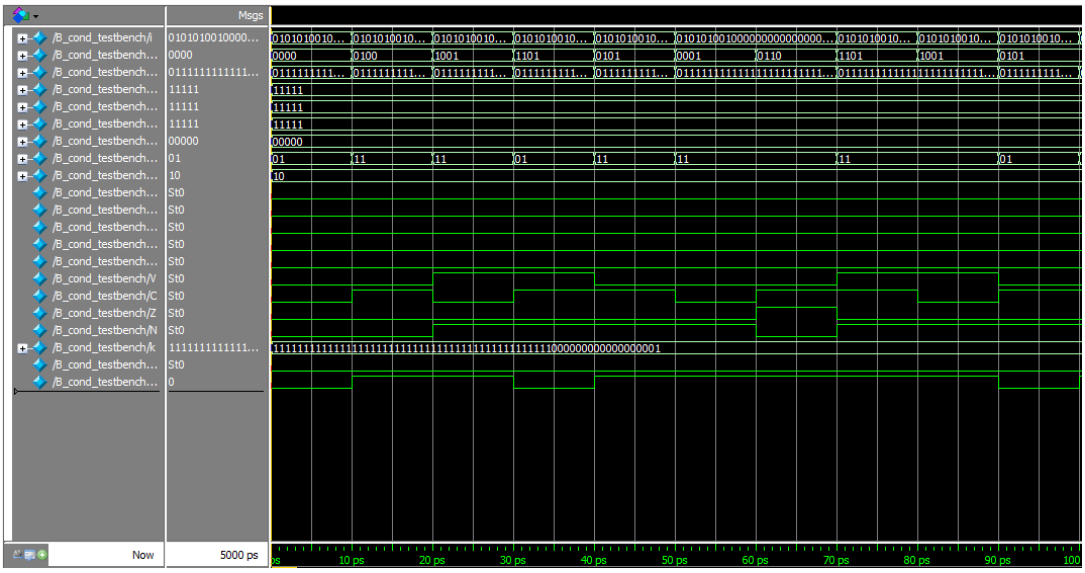


Figure 7.3 - B_Conditional Testbench Results

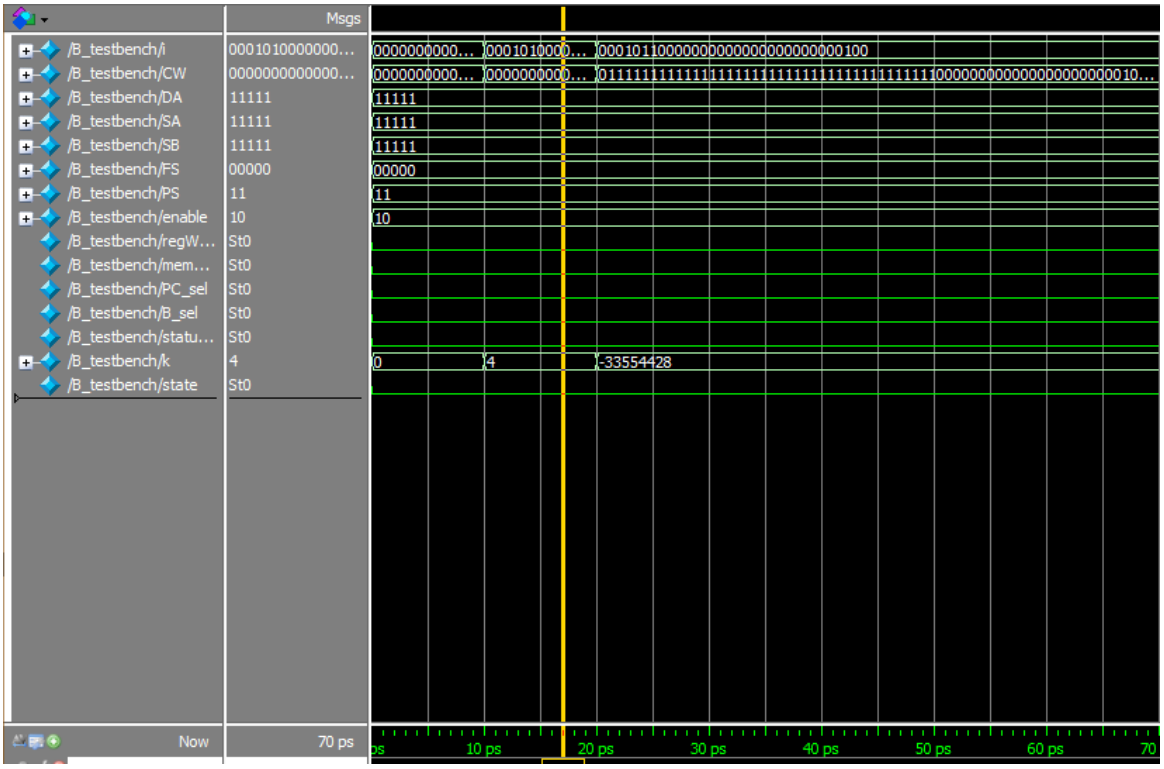


Figure 7.4 - B Testbench

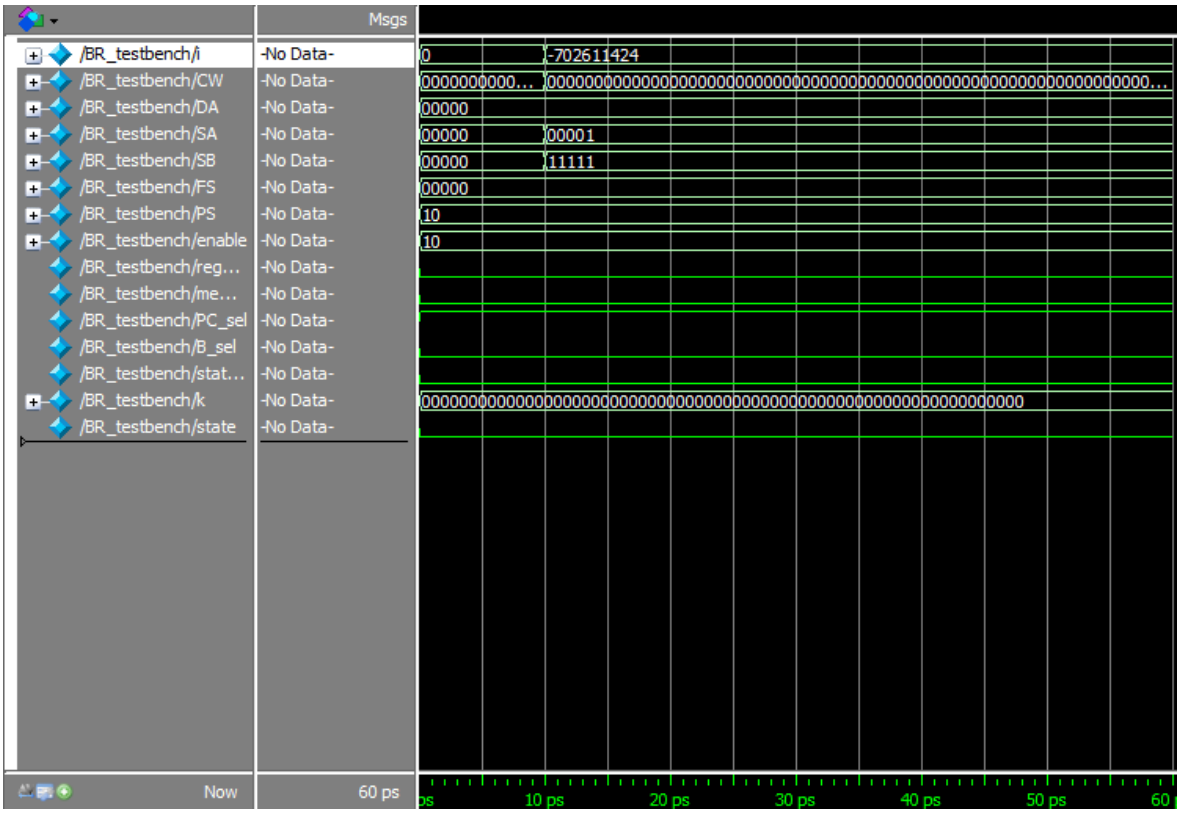


Figure 7.5 - BR Testbench

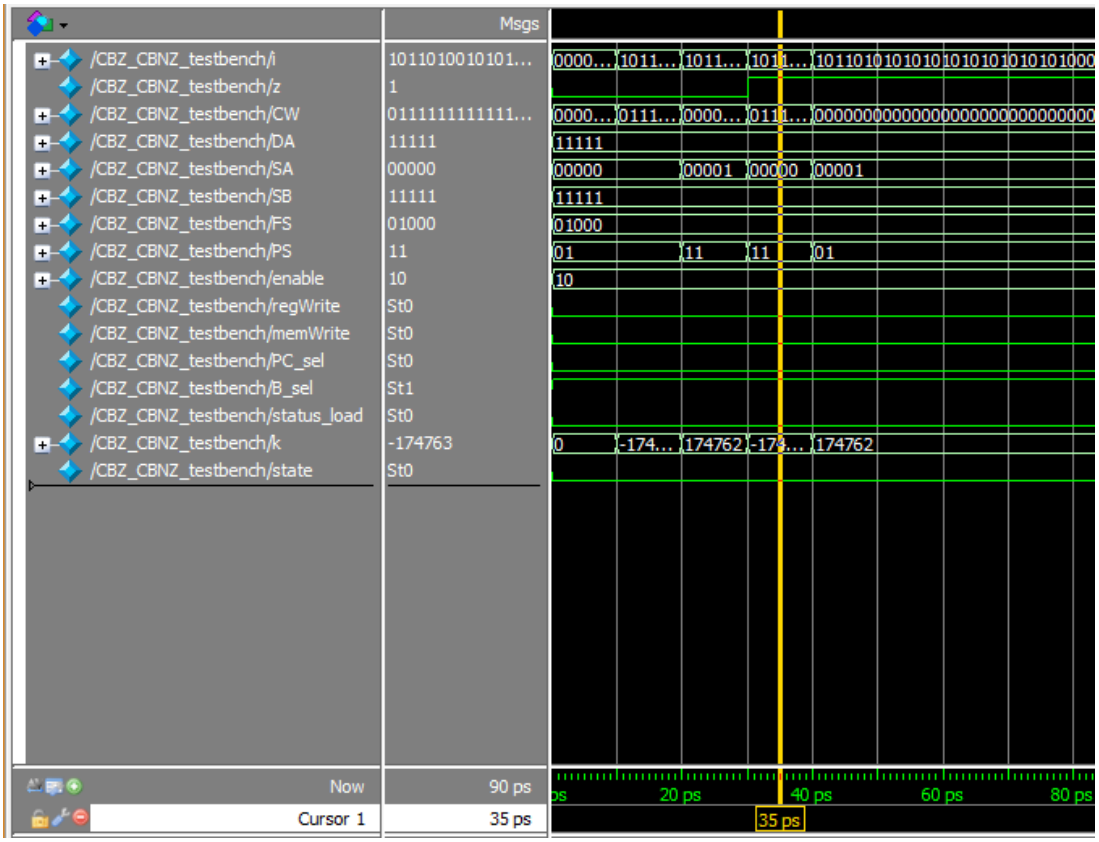


Figure 7.6 - CBZ_CBNZ Testbench

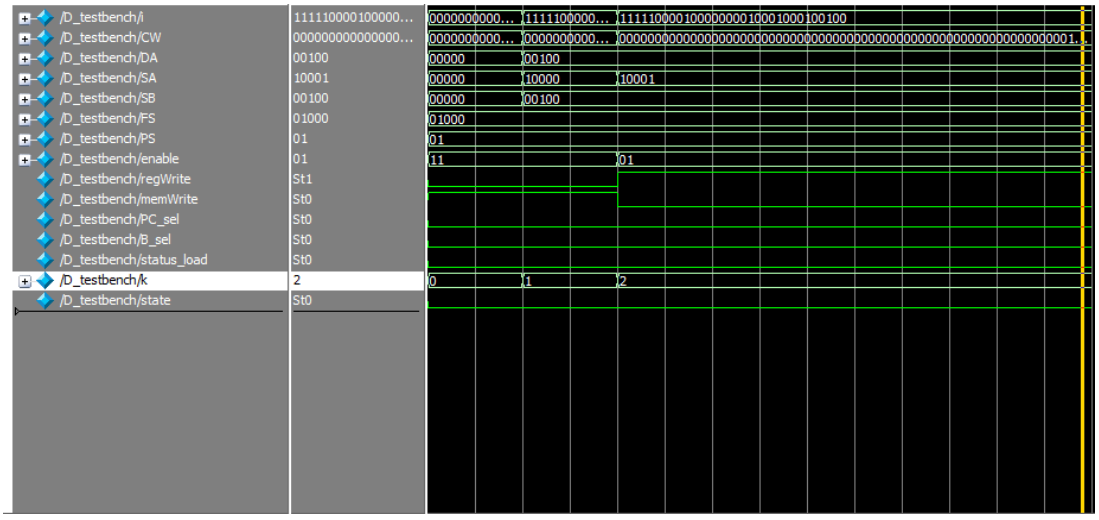


Figure 7.7 - D Testbench

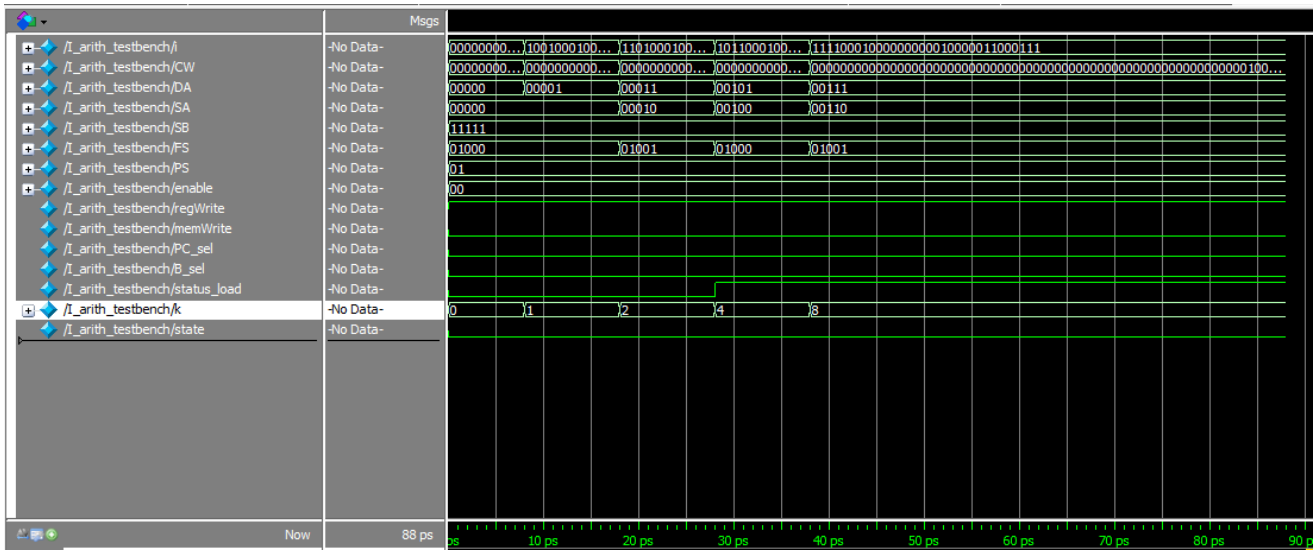


Figure 7.8 - I_arith Testbench

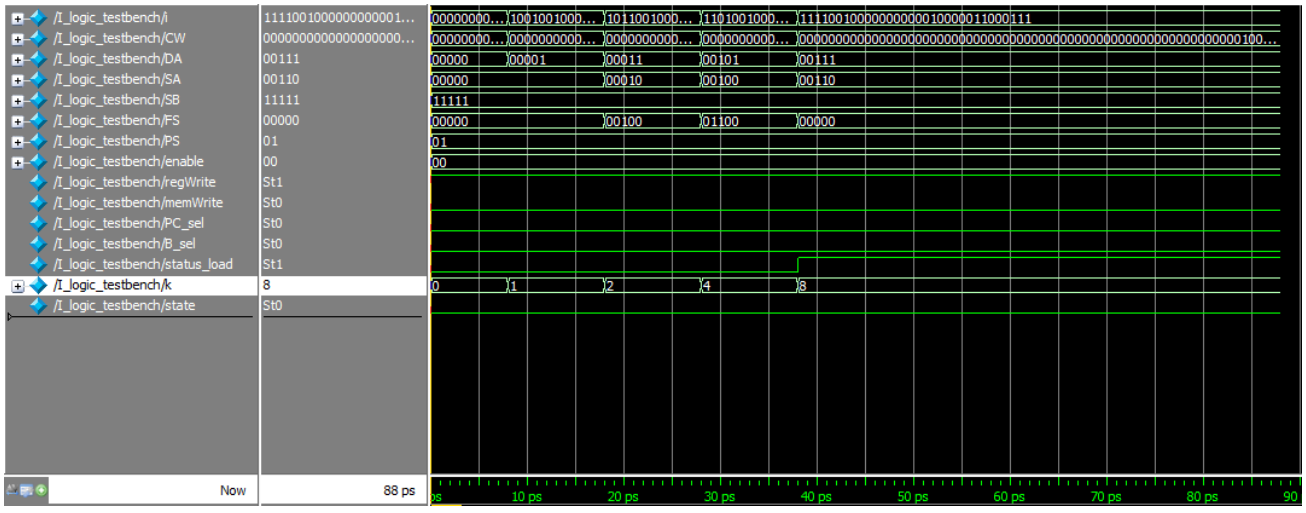


Figure 7.9 - I_logic Testbench

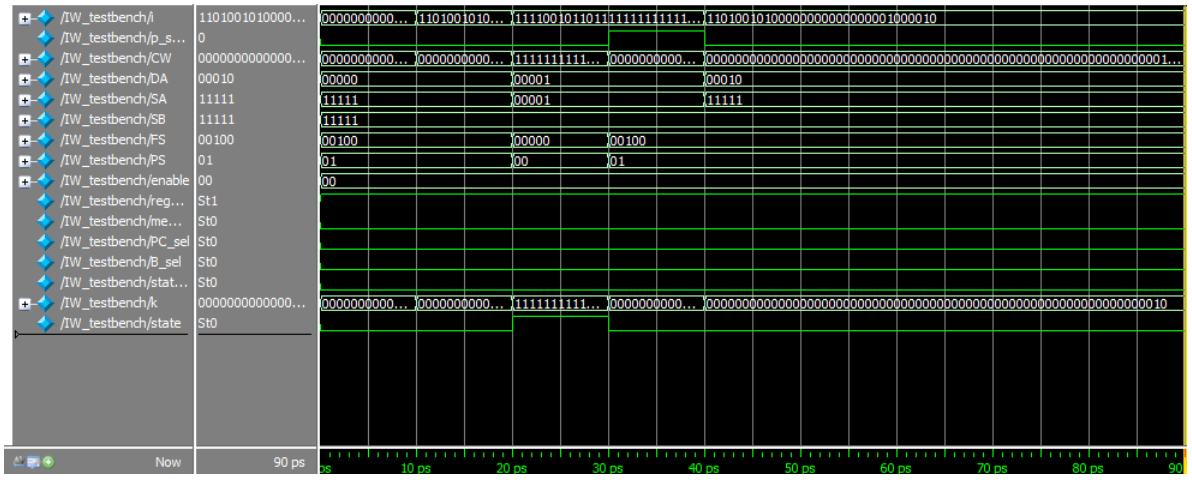


Figure 7.10 - IW Testbench

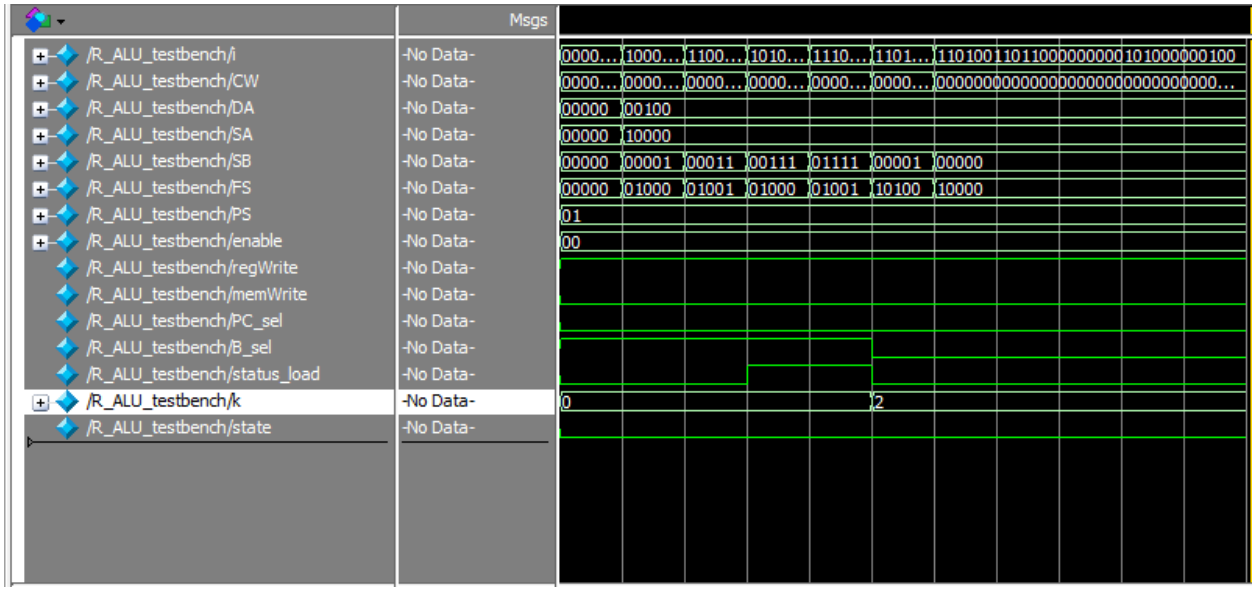


Figure 7.11 - R_ALU Testbench

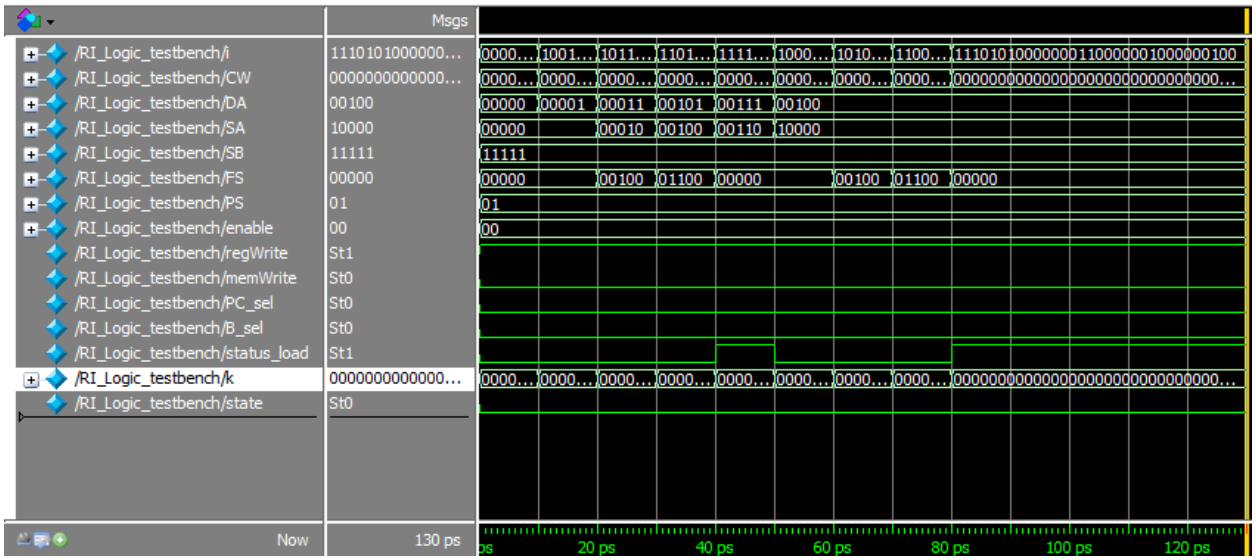


Figure 7.12 - RI_Logic Testbench

8. CPU (CENTRAL PROCESSING UNIT)

8.1. CPU Description

The CPU (Central Processing Unit) is the core of the Processor. The outputs of the CPU consists of an 8 bit address bus, a 64 bit databus, and a 1 bit Write signal. The input of the CPU consist solely of a 64 bit data_in bus, which receives data from any attached peripherals. Alterations such as increasing the address bit size to accommodate more peripherals or pipelining the processor would be lead to an optimized design.

8.2. CPU Schematic

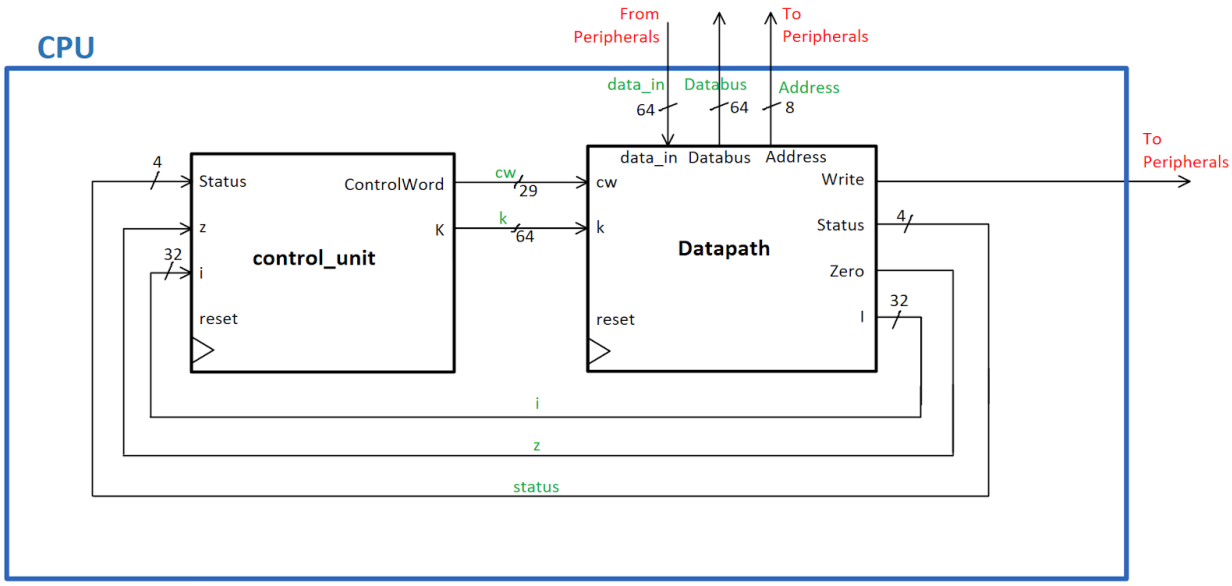


Figure 8.1 - CPU Diagram

8.3. CPU Testbench

In order to validate the CPU, the results that were produced by the testbench were compared to the table of expected results shown below in *Fig 8.3.* To prove the functionality of the CPU, the ADDI X1,X1,2 instruction was randomly chosen and compared to the results provided by the testbench in *Fig 8.2.*

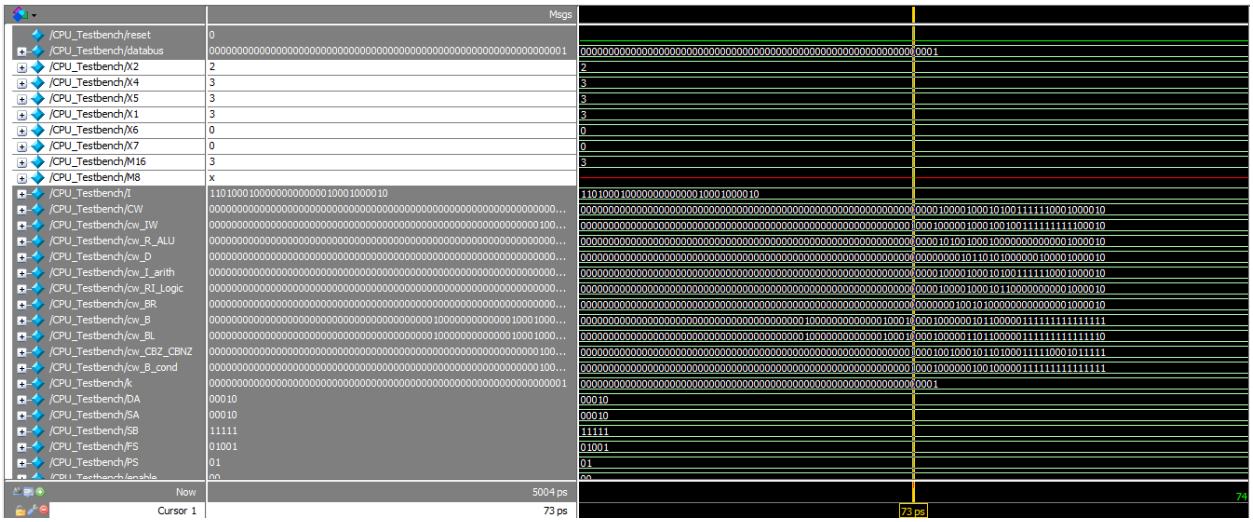


Figure 8.2 - CPU Testbench Results

Clock	Address	Instruction	X1	X2	X4	X5	X6	X7	M[16]	M[8]
1	0	MOVZ X1, 1	1	0	0	0	0	0	x	x
2	4	MOVZ X2, 2	1	2	0	0	0	0	x	x
3	8	ADD X4, X1, X2	1	2	3	0	0	0	x	x
4	12	STUR X4, [XZR, 16]	1	2	3	0	0	0	3	x
5	16	LDUR X5, [XZR, 16]	1	2	3	3	0	0	3	x
6	20	BL 10	1	2	3	3	0	0	3	x
7	64	ADDI X1, X1, 2	3	2	3	3	0	0	3	x
8	68	SUBI X2, X2, 1	3	1	3	3	0	0	3	x
9	72	BR X30	3	1	3	3	0	0	3	x
10	24	CBNZ X2, 1	3	1	3	3	0	0	3	x
11	32	B -7	3	1	3	3	0	0	3	x
12	8	ADD X4, X1, X2	3	1	4	3	0	0	3	x
13	12	STUR X4, [XZR, 16]	3	1	4	3	0	0	4	x
14	16	LDUR X5, [XZR, 16]	3	1	4	4	0	0	4	x
15	20	BL 10	3	1	4	4	0	0	4	x
16	64	ADDI X1, X1, 2	5	1	4	4	0	0	4	x
17	68	SUBI X2, X2, 1	5	0	4	4	0	0	4	x
18	72	BR X30	5	0	4	4	0	0	4	x
19	24	CBNZ X2, 1	5	0	4	4	0	0	4	x
20	28	B 1	5	0	4	4	0	0	4	x
21	36	CBZ X1, 3	5	0	4	4	0	0	4	x
22	40	SUBS XZR, X1, X2	5	0	4	4	0	0	4	x
23	44	B.LO 1	5	0	4	4	0	0	4	x
24	48	STUR X1, [XZR, 8]	5	0	4	4	0	0	4	5
25	52	LDUR X6, [XZR, 8]	5	0	4	4	5	0	4	5
26	56	EORI X7, X7, 1	5	0	4	4	5	1	4	5
27	60	B -2	5	0	4	4	5	1	4	5
28	56	EORI X7, X7, 1	5	0	4	4	5	0	4	5
29	60	B -2	5	0	4	4	5	0	4	5
repeat instruction 56 / 60 forever										

Fig. 8.3 - RomCase expected results

9. Peripherals/Programs/Errata:

Due to time constraints, peripherals and programs for this processor design were not fully implemented in time. However, multiple concepts for peripherals and the idea behind their implementation are included below:

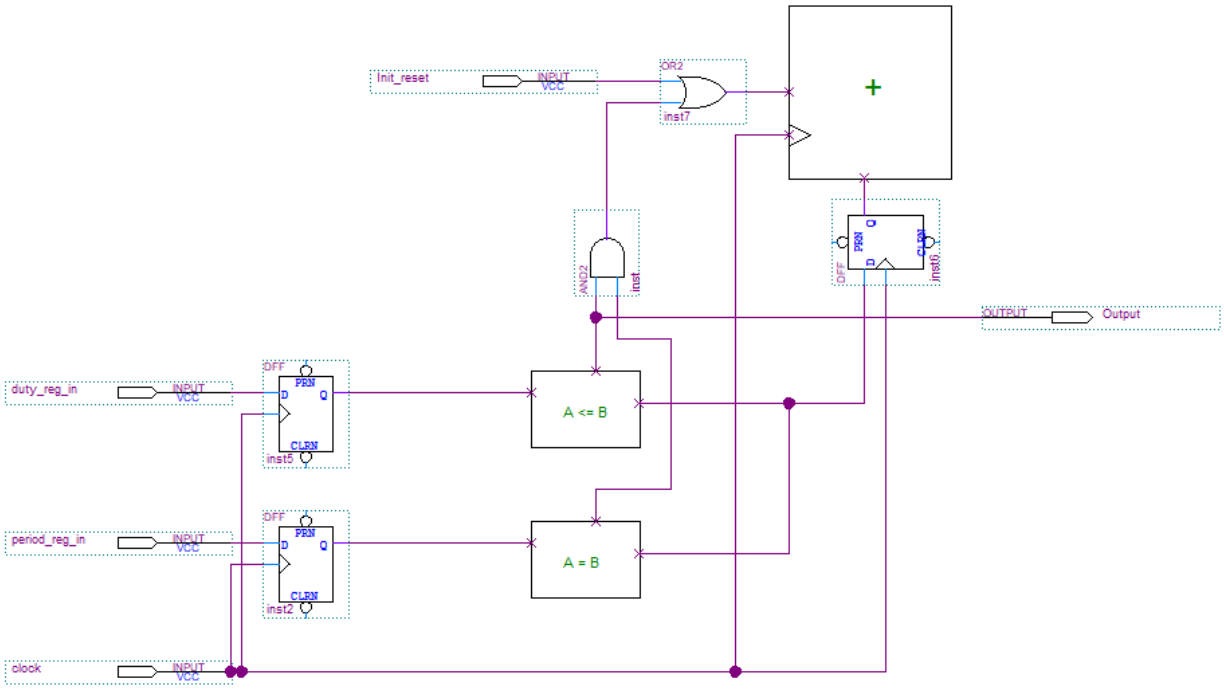


Figure 9.1 Pulse Width Modulation Timer Peripheral

One peripheral design considered for the processor was a PWM wave generator. This design utilizes two registers, including a duty register and a period length register. The purpose of the duty register was to dictate the duty cycle length of the PWM wave while the period length register determined the length of the period. Next, two comparators were utilized to determine the output of the PWM wave generator. One of the comparators checked to see if the period register value was equal to value provided by the incrementor block. The second comparator checks to see if the duty register value was less than or equal to the value provided by the incrementor block. If the value provided by the incrementor equals or exceeds the value stored in the duty register, a signal of 1 will be output of the comparator, causing the wave to go from 0 to 1. When the period value is equal to the incrementor value, a signal of 1 will be generated by the period comparator. With the logic provided through the use of an AND gate, the combination of signals from the comparators will reset the incrementor block, representing the end of the wave and the start of the next. Although this peripheral was not able to be connected to our processor in time, the following code and testbench in *Appendix 1 and Appendix 2* proves the functionality of this peripheral.

```
//Concept: This peripheral is a PWM wave generator that generates a
wave depending on the value loaded from the RAM.
//This peripheral operates by taking a value from a location in the
memory.
//After the value is taken, it will be stored into the Period
Register, in which that value represents the value
//of the period

module PWMPeripheral(periodregin, dutyregin, out, reset, clock);
input [63:0] periodregin;
input [63:0] dutyregin;
input reset, clock;
output out;

wire [63:0] DutyRegWire, PeriodRegWire, IncrementerRegWire;
wire ComparatorOne, ComparatorTwo, ResetCounter;

//Instantiate each component
Incremeneter PWMIncrementer (reset, ResetCounter, clock,
IncrementerRegWire);
ComparatorLessThan c1(DutyRegWire,IncrementerRegWire, ComparatorOne);
ComparatorEqual c2(PeriodRegWire, IncrementerRegWire, ComparatorTwo);
RegisterNbit DutyReg (DutyRegWire, dutyregin, 1'b1, reset, clock);
//Load is always enabled.
RegisterNbit PeriodReg (PeriodRegWire, periodregin, 1'b1, reset,
clock); //datain is value loaded from mem. Load is always enabled.

//The outputs of both comparators are put through an AND gate
assign ResetCounter = ComparatorOne & ComparatorTwo; //ands together
both comparator results into a single wire.
assign out = ComparatorOne; //takes value from A less than B and makes
that the output.
endmodule

//64 bit comparator that will check if A is less than B and output a 1
or 0.
module ComparatorLessThan (A,B,C);
input [63:0] A,B;
output C;
reg D;

always@(*)
begin
```

```
if (A <= B)
D = 1'b1;
else
D = 1'b0;
end
assign C = D;
endmodule

//64 bit comparator that will check if A is equal to B and output a 1
or 0.
module ComparatorEqual (A,B,C);
input [63:0] A,B;
output C;
reg D;

always @(*)
begin
if (A == B)
D = 1'b1;
else
D = 1'b0;
end
assign C = D;
endmodule

// 64 bit incrementer. This constantly adds and will reset to 0 when
reset is on. Will also pause counting when enable is 0.
module Incremeneter (initreset, reset, clock, A);
input reset, initreset, clock;
output [63:0] A;
reg [63:0] count;


always @(posedge clock)
begin
if (reset | initreset)
count <= 64'd0;
else
count <= count + 64'd1;
end
assign A = count;
endmodule
```

9.1 PWM Peripheral Code

In order for the peripheral to interface with the processor, a new ROM program must be created to support this functionality. As seen in *Appendix 3*, the first two instructions load values 10,000 and 5,000 into registers X0 and X1 respectively. Next, the value in X0 is stored to a defined memory location while X1 is stored to another memory location that falls within the range of our SFR address range.

```
module rom_case_PWM(out, address);
      output reg [31:0] out;
      input  [15:0] address; // address- 16 deep memory
      always @(address) begin
            case (address)
                  16'h0000: out =
32'b110100101_00_0010011100010000_00000; //First instruction: Sets X0
                                          //to 10,000
                  16'h0004: out =
32'b110100101_00_0001001110001000_00001; //Second instruction: Sets X1
to 5,000
                  16'h0008: out =
32'b11111000000_100000000_00_11111_00000; //Third instruction: Value
of X0 stored to memory
                  16'h0012: out =
32'b11111000000_100000000_00_11111_00001; //Fourth instruction: Value
of X1 stored to memory
            default: out=32'hD60003E0; //BR XZR
            endcase
      end
endmodule
```

10. APPENDICES

Appendix 1: PWM Peripheral Code

```
module PWMPeripheralTestBench();
reg clock;
reg reset;
wire out;
reg [63:0] dutyregin, periodregin; //loads value into the Period and Duty
Register.
PWMPeripheral dut(periodregin, dutyregin, out, reset, clock);

wire [63:0] DutyRegWire, PeriodRegWire, IncrementerRegWire;
wire ComparatorOne, ComparatorTwo, ResetCounter;

assign out = dut.out;
assign DutyRegWire = dut.DutyRegWire;
assign PeriodRegWire = dut.PeriodRegWire;
assign ComparatorOne = dut.ComparatorOne;
assign ComparatorTwo = dut.ComparatorTwo;
assign ResetCounter = dut.ResetCounter;
assign IncrementerRegWire = dut.IncrementerRegWire;

initial begin
reset <= 1;
clock <= 0;
#10
reset <= 0;
#1000 $stop;
end

always
      begin
            #5 clock <= ~clock;
      end
initial
begin
#20
assign periodregin =
64'b0000000000000000000000000000000000000000000000000000000000001010; //10
assign dutyregin =
64'b0000000000000000000000000000000000000000000000000000000000000101;
//5 (50% duty cycle)
end

endmodule
```

List of Figures

1. Register File

Fig.1.1 - Register Diagram……..….5

Fig. 1.2 - Register Testbench……...…..…6

2. ALU

Fig. 2.1 - ALU Diagram……...…..7

Fig 2.2 - Single ALU Cell ……8

Fig. 2.3: ALU Testbench……9

3. Instruction Set

none

4. RAM

Figure 4.1 - Basic RAM Architecture……...19

Figure 4.2 - Memory Map…….....20

5. Program Counter

Figure 5.1 - Program Counter Diagram…….....21

Figure 5.2 - Program Counter Testbench…….…….22

6. Datapath

Figure 6.1 - Datapath Diagram…….…….23

7. Control Unit

Figure 7.1 - Control Unit State Diagram……..…….....27

Figure 7.2 - Control Unit Diagram……...…....28

Figure 7.3 - B_Conditional Testbench Results…….…....29

Figure 7.4 - B Testbench……...…30

Figure 7.5 - BR Testbench…….....30

Figure 7.6 - CBZ_CBNZ Testbench……..…31

Figure 7.7 - D Testbench……...…....31

Figure 7.8 - I_arith Testbench……..……32

Figure 7.9 - I_logic Testbench…….....32

Figure 7.10 - IW Testbench…….....33

Figure 7.11 - R_ALU Testbench……...…….33

Figure 7.12 - RI_Logic Testbench……..……..34

8. CPU

Figure 8.1 - CPU Diagram…….....35

Figure 8.2 - CPU Testbench Results…….....36

Figure 8.3 - RomCase expected results…….....36

9. Peripherals/Programs/Errata:

Figure 9.1 Pulse Width Modulation Timer Peripheral……...37

List of Tables

1. Register File

Table 1.1 - Input/Output Table……...….8

2. ALU

Table 2.1: Inputs and Outputs of the ALU……..…8

3. Instruction Set

Table 3.1: Instruction Formats……...10

Table 3.2: Opcode fields & terminology descriptions……...…11

Table 3.3 - ADD Instruction Table…….…12

Table 3.4 - SUB Instruction Table……...……..12

Table 3.5 - ADDI Instruction Table……...……...12

Table 3.6 - SUBI Instruction Table……...…….12

Table 3.7 - ADDS Instruction Table…….……13

Table 3.8 - SUBS Instruction Table…….……..13

Table 3.9 - ADDIS Instruction Table……..……..13

Table 3.10 - SUBIS Instruction Table……...…….13

Table 3.11 - STUR Instruction Table……..……...14

Table 3.12 - LDUR Instruction Table……..……..14

Table 3.13 - MOVZ Instruction Table……..…….14

Table 3.14 - MOVK Instruction Table…….…..14

Table 3.15 - AND Instruction Table……..…….15

Table 3.16: ORR Instruction Table……...…….15

Table 3.17: EOR Instruction Table…….……...15

Table 3.18: ANDI Instruction Table……...……..15

Table 3.19: EORI Instruction Table……..…….16

Table 3.20 - ORRI Instruction Table…….…….……16

Table 3.21: ANDS Instruction Table……...…….……..16

Table 3.22 - ANDIS Instruction Table……...…….……...16

Table 3.23: LSR Instruction Table……...…..…….17

Table 3.24: LSL Instruction Table……...……..17

Table 3.25: CBZ Instruction Table……..……...17

Table 3.26: CBNZ Instruction Table…….…….17

Table 3.27: B.Conditional Table……....18

Table 3.28: B Instruction Table…….18

Table 3.29: BR Instruction Table……...……18

Table 3.30: Branch and Link Instruction Table……...……...18

4.RAM

None

5. Program Counter

Table 5.1: Program Counter Inputs/Outputs……...……...22

6. Datapath

Table 6.1 - Datapath Inputs/Outputs……...……..24
Table 6.2 - Control Signals……...…….25
Table 6.3 - Index of instantiated modules in the datapath……...…….26

7. Control Unit

Table 7.1 - Index of Control Unit Decoders……...……..28

8. CPU

None

9. Peripherals/Programs/Errata:

None