EE-224 Project: CPU

Vinay Sutar(21d070078)
Shounak Das(21d070068)
Aditya Anand(21d070007)
Parth Arora(21d070047)
Daksh Pakal(210070023)

November 2022

Contents

1 Instructions

1.1 Types

R Type Instruction format

Opcode	Register A (RA)	Register B (RB)	Register C (RC)	Unused	Condition (CZ)
(4 bit)	(3 bit)	(3-bit)	(3-bit)	(1 bit)	(2 bit)

I Type Instruction format

Opcode	Register A (RA)	Register C (RC)	Immediate
(4 bit)	(3 bit)	(3-bit)	(6 bits signed)

J Type Instruction format

Opcode	Register A (RA)	Immediate
(4 bit)	(3 bit)	(9 bits signed)

Figure 1: Types of Instructions

1.2 Instructions

ADD:	00_00	RA	RB	RC	0	00
ADC:	00_00	RA	RB	RC	0	10
ADZ:	00_00	RA	RB	RC	0	01
ADI:	00_01	RA	RB	6 bit Immediate		
NDU:	00_10	RA	RB	RC	0	00
NDC:	00_10	RA	RB	RC	0	10
NDZ:	00_10	RA	RB	RC	0	01
LHI:	00_11	RA	9 bit Immediate			
LW:	01_00	RA	RB	6 bit Immediate		
SW:	01_01	RA	RB	6 bit Immediate		
LM:	01_10	RA	0 + 8 bits corresponding to Reg R7 to R0			
SM:	01_11	RA	0 + 8 bits corresponding to Reg R7 to R0			
BEQ:	11_00	RA	RB	6 bit Immediate		
JAL:	10_00	RA	9 bit Immediate offset			
JLR:	10_01	RA	RB	000_000		

Figure 2: Instructions

1.3 Instruction Description

Instruction Description

Mnemonic	Name & Format	Assembly	Action
ADD	ADD (R)	add rc, ra, rb	Add content of regB to regA and store result in regC. *It modifies C and Z flags*
ADC	Add if carry set (R)	adc rc, ra, rb	Add content of regB to regA and store result in regC, if carry flag is set. *It modifies C & Z flags*
ADZ	Add if zero set (R)	adz rc, ra, rb	Add content of regB to regA and store result in regC, if zero flag is set. *It modifies C & Z flags*
ADI	Add immediate (I)	adi rb, ra, imm6	Add content of regA with Imm (sign extended) and store result in regB. *It modifies C and Z flags*
NDU	Nand (R)	ndu rc, ra, rb	NAND the content of regB to regA and store result in regC. *It modifies Z flag*
NDC	Nand if carry set (R)	ndc rc, ra, rb	NAND the content of regB to regA and store result in regC if carry flag is set. *It modifies Z flag*
NDZ	Nand if zero set (R)	ndc rc, ra, rb	NAND the content of regB to regA and store result in regC if zero flag is set. *It modifies Z flag*
LHI	Load higher immediate (J)	lhi ra, Imm	Place 9 bits immediate into most significant 9 bits of register A (RA) and lower 7 bits are assigned to zero.
LW	Load (I)	lw ra, rb, Imm	Load value from memory into reg A. Memory address is computed by adding immediate 6 bits with content of reg B. *It modifies flag Z.*

SW	Store (I)	sw ra, rb, Imm	Store value from reg A into memory. Memory address is formed by adding immediate 6 bits with content of red B.
LM	Load multiple (J)	lm ra, Imm	Load multiple registers whose address is given in the immediate field (one bit per register, R7 to R0) in order from right to left, i.e., registers from R0 to R7 if corresponding bit is set. Memory address is given in reg A. Registers are loaded from consecutive addresses.
SM	Store multiple (J)	sm, ra, Imm	Store multiple registers whose address is given in the immediate field (one bit per register, R7 to R0) in order from right to left, i.e., registers from R0 to R7 if corresponding bit is set. Memory address is given in reg A. Registers are stored to consecutive addresses.
BEQ	Branch on Equality (I)	beq ra, rb, Imm	If content of reg A and regB are the same, branch to PC+Imm, where PC is the address of beq instruction
JAL	Jump and Link (I)	jalr ra, Imm	Branch to the address PC+ Imm. Store PC into regA, where PC is the address of the jalr instruction
JLR	Jump and Link to Register (I)	jalr ra, rb	Branch to the address in regB. Store PC into regA, where PC is the address of the jalr instruction

Figure 3: Instruction Description

2 Components

2.1 Instruction Register

Memory from the register is read (*irw='1'*) and stored as a signal *instruction*. The first 4 bits (15 downto 12) of the instruction is the *opcode*, the next 3 bits is the *register A* (11 downto 9), the next 3 bits is the *register B* (8 downto 6), the next 3 bits is the *register C* (5 downto 3), 3rd bit is unused and (1 downto 0) i.e. the last 2 bits correspond to the *carry and zero flag*.

2.2 Memory/RAM

The input variable *init* when goes high, leads to the instruction (15 downto 0) (16 bits) to be read into storage element named $store_ram$ The instruction contains information about opcode, registers and carry and zero flags.
The variable *mread* going high(='1') results in reading of data($store_ram$) as **dout**, while *mwrite* corresponds to writing of data in the variable **din** for further processing.

2.3 Register File

The *registerwrite* variable allows the data to be transfered from the input din_m to the register. Data from 2 registers is required to execute the desired instruction. These 2 registers viz *Register A* and *Register B* (out of 8) are chosen depending on the values of variable *registerselector*. The data from the instruction is sent to the selected registers for further processing in the ALU.

2.4 ALU

Here various processes (like add, subtract, multiply etc) are performed on the 2 inputs (*inp1, inp2*). Selector is a variable that triggers the operation to be performed on the inputs. The output is then used to decide carry and zero flag as shown in the code.

2.5 CPU

This is the main file connecting all the components. Initially the states are initialized in order as constant signals. All the individual components are included here. Further the opcodes of the instructions are initialized as written in the code.

2.6 Sequencer

2.7 Sign Extender

2.8 Sixteen Bit Adder

3 State Flow Diagram

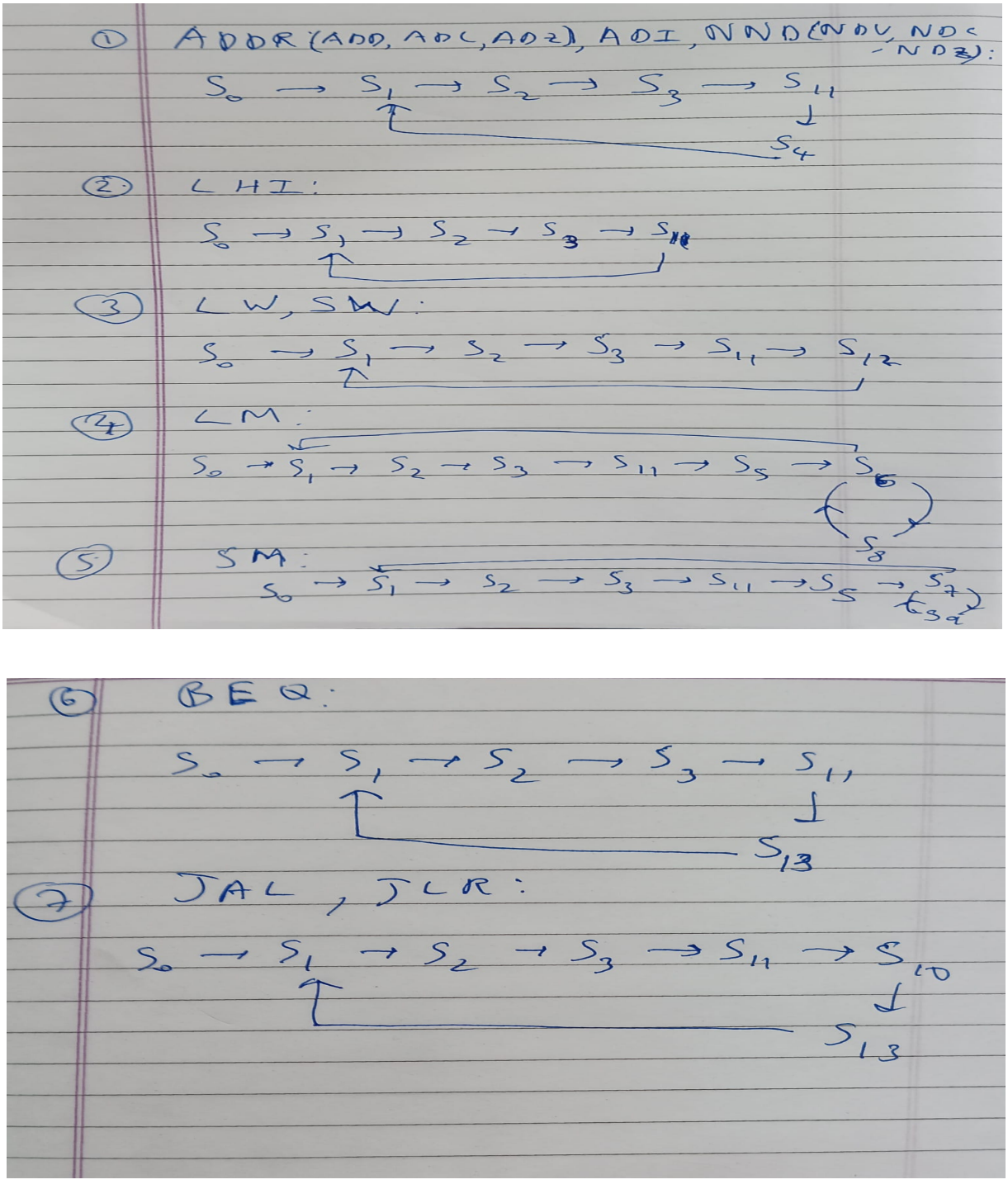

Figure 4: State Flow Diagram

4 Dataflow

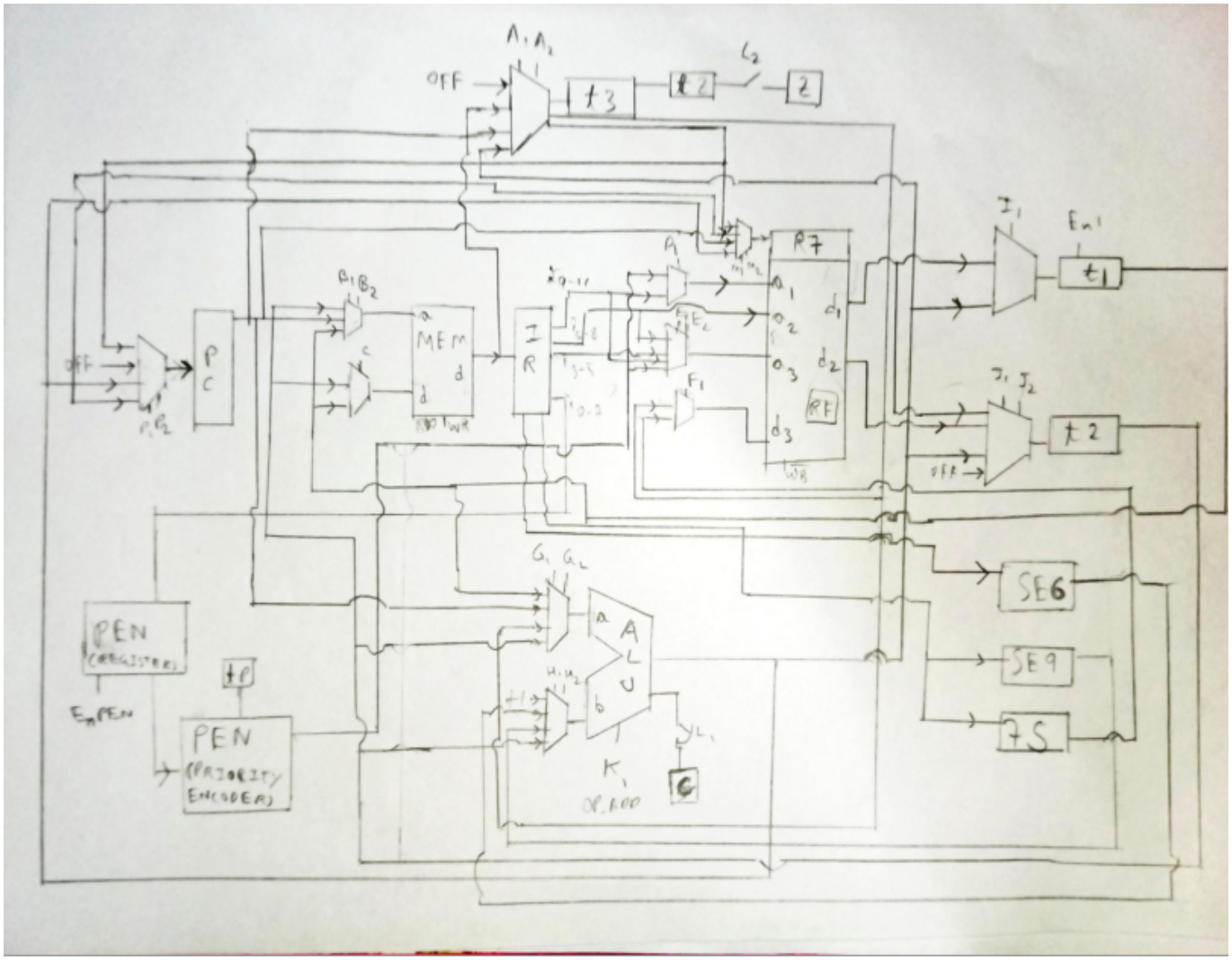

Figure 5: Dataflow

5 Implementation

The memory component has the following 'in' ports:

1. state: denoting the state of the machine
2. init: to store predefined instructions in the memory
3. mread: bit for memory read
4. mwrite: bit for memory write
5. dataPointer: denoting address in the memory
6. din: data in
   It has only one 'out' port:
   (a) dout: data out

6 RTL Diagram

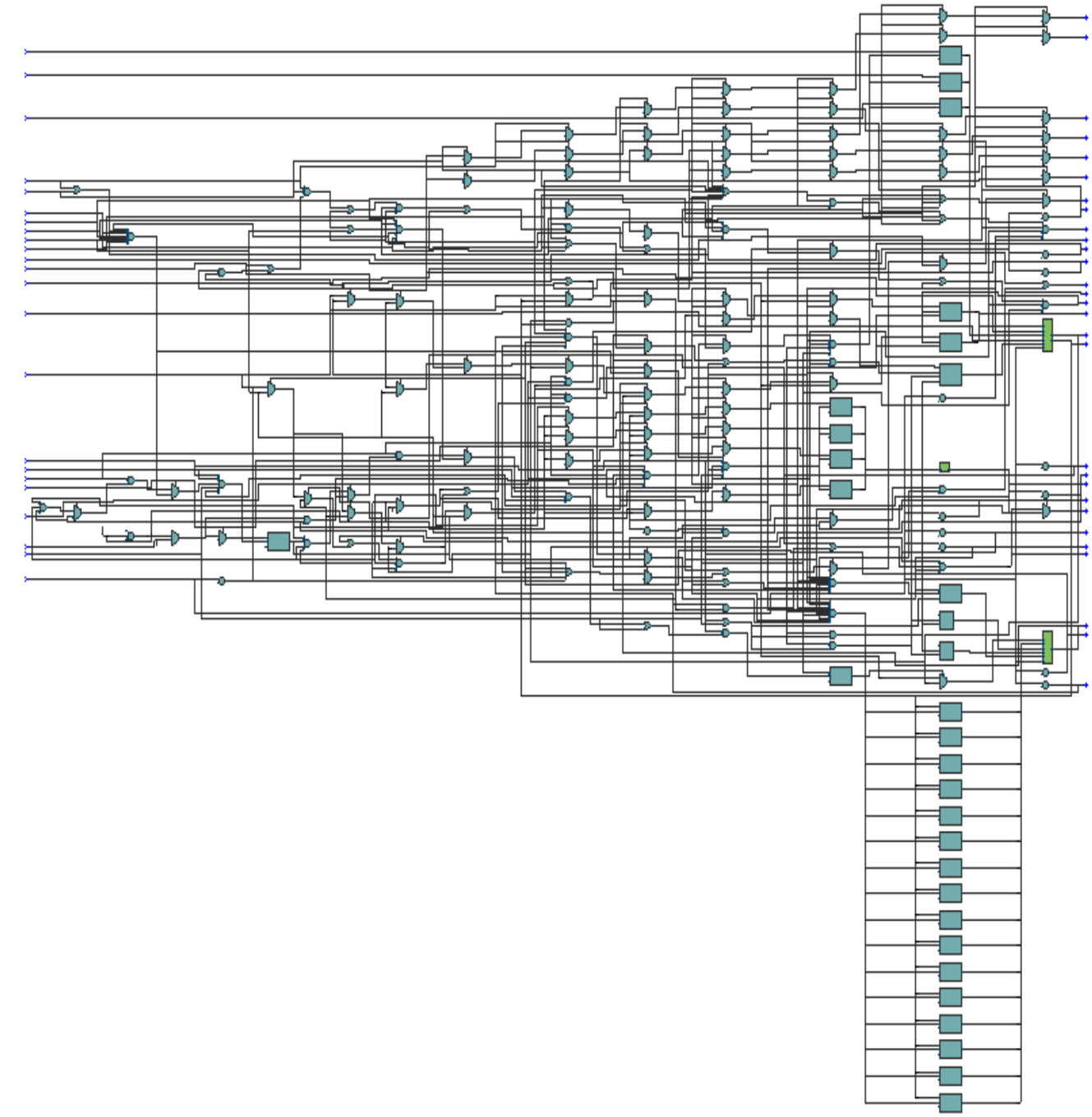

Figure 6: RTL Diagram

7 Technology Viewer Diagram

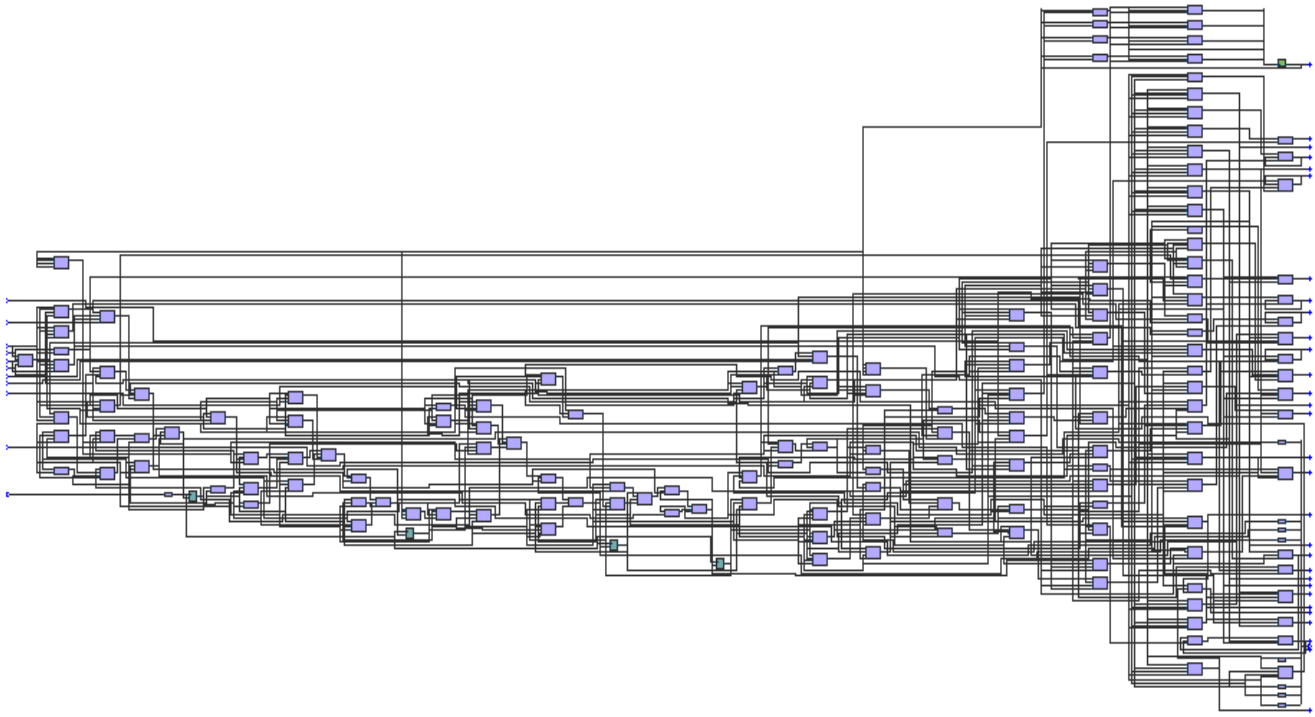

Figure 7: Technology Viewer Diagram

8 Waveform

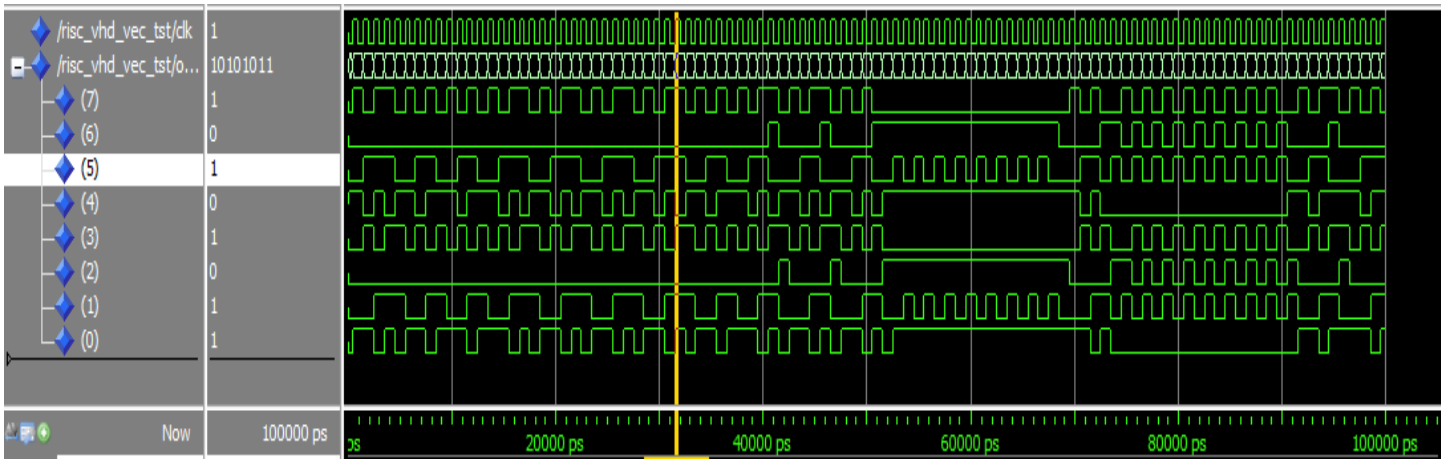

Figure 8: Waveform