ITU VLSI LABS

VLSI 2

Homework #9

KAMİL SAN - ELİF ÖZTÜRK

040160081 - 040170208

RANDOM – ACCESS MEMORY (RAM)

A single port synchronous RAM is designed to be used as data memory and instruction memory in microprocessor design. In the design, ADDR_WIDTH = 8, DATA_WIDTH = 8 and DEPTH = 256. In this way, DEPTH number of data is stored, each of which has a DATA_WIDTH bit number. For the address entry, it is selected such that 2 ^ (ADDR_WIDTH) = DEPTH. Active-high synchronous reset has been added. Since initial begin cannot be made on Cadence, the values to be written to the memory beforehand are written at the time of reset. The size of the memory, namely depth, was kept to a minimum by choosing the amount used. While it is 12 for data memory, it is 80 for instruction memory. One of the important issues is that the data width is 8, because in instructions such as SB, LB, LH, 8-16 bits of data are written to memory. In order for this to work properly, a case-when structure is created using the data from the control signal. In this way, data with 8-16-32 bits is written to memory. 8 bits were used in Instruction memory, so the PC value was increased by 4. Instruction memory is written to always read when there is no reset.

FUNCTION UNIT

It consists of a function unit, ALU and shifter, SLTU block and 4x1 multiplexer. The control signal of the function unit FS is 6 bits. FS [5:4] bits are used to select one of the outputs ALU, shifter, less and lessU required for compare. FS [3: 0] are control signals of ALU. FS [1: 0] is the control signal of the shifter. 4 status signals are given at the FU output: C, V, N, Z. C stands for carry flag, it indicates carry out 1 in adder block V means overflow, it is set when a positive number is obtained from the sum of two negative numbers or a negative number from the sum of two positive numbers. The N flag is a negative flag indicating that the output is less than 0. Z flag means zero flag. It is set when the output is equal to 0. Branch signals have also been added to this block to show that the Instruction was implemented correctly.

After adding the pipeline structure, the functions of the flags disappeared with the control hazard. Branch control signals are generated by adding a comparator block to the output of Register File.

ARITHMETIC LOGIC UNIT (ALU)

In the ALU design, A, B, G data signals, Cin, S as control signals, V and Cout as status signals. Data signals can be immediate, shamt, or values stored in rd and rs registers, depending on the instructions. Control signals, on the other hand, are signals that are used to implement the instructions to the Control Unit and to select the appropriate blocks. ALU has a 4-bit control signal. According to the value of these control signals, ALU performs arithmetic operations with the arithmetic circuit inside or logic operations with the logic circuit. As a result of these operations, ALU creates an output value and status signals to show the correctness of this output value. These status signals are Carry and Overflow. The carry signal indicates the overflow bit in the addition block in the ALU. Overflow shows this situation by taking the value of logic-1 when the sum of two negative numbers is positive or the sum of two positive numbers is negative.

Two separate modules are used as arithmetic circuit and logic circuit. A choice is made between these two blocks with 2x1 MUX.

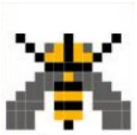
In the internal structure of the logic block according to the 4x1 mux select inputs, it performs AND, OR, XOR, NOT operations respectively.

The arithmetic Circuit module includes two separate modules: B input logic and parallel adder. FA blocks constitute the parallel adder structure, and between the Y input of FA and the B input, the B input logic module.

Parallel adder structure was created with ripple carry adder (RCA). However, the Square Root Carry Select Adder structure is used for better performance. This circuit was taken from the article "Design of 32-bit Carry Select Adder with Reduced Area" written by Ykuntam, Rao and Locharla and used in the design. In the Basic SQRT CSLA structure, the design is first divided into sub-blocks according to the formula N = M + (M + 1) + (M + 2) + ... + (M + P-1). Since this formula does not fit the 32 bits exactly, we used the 2-2-3-4-6-7-8 stages given in the article in our design. Then, in each stage, the addition process is performed against Cin = 0 and Cin = 1, and at the end of the process, the appropriate result is selected according to the previous stage in carry signal. In this way, all 7 stages are made in parallel and the delay created by the carry chain is reduced. In other designs in the article, Cin = 1 state is also calculated by adding 1 (with BEC or add one circuit) after calculating Cin = 0. These designs are more advantageous in terms of using area, but they are more disadvantageous in terms of time since they start after completing the Cin = 0 condition. We decided to use the basic structure as we attach more importance to the use of time in our design.

Function Table for ALU

Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G = A$	Transfer A
0	0	0	1	$G = A + 1$	Increment A
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + B + 1$	Add with carry input of 1
0	1	0	0	$G = A + \overline{B}$	A plus 1s complement of B
0	1	0	1	$G = A + \overline{B} + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement A
0	1	1	1	$G = A$	Transfer A
1	X	0	0	$G = A \wedge B$	AND
1	X	0	1	$G = A \vee B$	OR
1	X	1	0	$G = A \oplus B$	XOR
1	X	1	1	$G = \overline{A}$	NOT (1s complement)



SHIFTER

In shifter design, a 2-bit select signal is used to implement SLL, SRL and SRA instructions. Left shift operation when the select signal was 2'b01, right shift operation when 2'b10 and arithmetic shift operation to the right when 2'b11 were performed. In Shifter, which has 32-bit input and output, a 5-bit input value specifying the shift amount is received. 32-bit shifter is designed behavioral. 2 nested case structures are used in Shifter. The first case structure indicates the shift operation, and the case structure in it indicates the amount of shift. In this structure, instead of using the shift operator, the signals are concatenated.

Select	Output
00	A
01	A << B (sl)
10	A >> B (sr)
11	A >>> B (sra)

REGISTER FILE

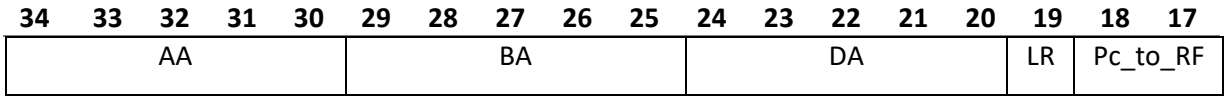
In addition to the memory structures, the register file structure is used to store the data in the designed processor. For example, while ADD r3 r4 operation is performed in assembly language, the data of the r3 and r4 registers are collected and saved to the r3 register. In our structure, there are 32 registers in RF. A 32x1 decoder is used to select the desired register. With the incoming address data, the desired register is selected with the help of the decoder. Then the write signal is activated and write to the register is done. Now that the writing process is complete, we can proceed to the reading process. We have two separate A and B outputs for reading. In order to read the value from the A output, the address value on the select pin of the 32x1 MUX A is selected. For B, similarly, 32x1 MUX B is used.

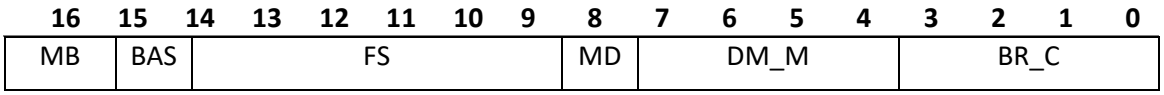
After adding the pipeline, writing to the register for the hazard occurred is done at the time of the negedge clock, so that the new value written to the same register in a clock period can be read.

CONTROL UNIT (INSTRUCTION DECODER)

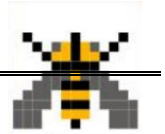
The Instruction Decoder block is responsible for sending the 32-bit instruction data from the instruction memory block to the correct places, in other words, it generates the control word from the instruction to control the whole system. A 33-bit control word is used in the system we designed. Within the Instruction decoder structure, correct control signals are provided for each instruction according to the values of instruction opcode and funct3. The case structure is used for this.

CONTROL WORD



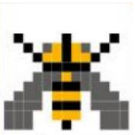


AA	A Address
BA	B Address
DA	Data Address
LR	Register File Load
Pc_to_RF	Used when the value of pc is written to Register File
MB	MuxB select signal
BAS	It is used to transfer immediate value to the A Bus and write it directly to the register.
FS	Function Select
MD	MuxD select signal
DM_M	For LW, LH, LB, LHU, LBU, SW, SH, SB instructions, it controls how data memory is saved and transferred to its output.
BR_C	This is the signal to check the Branch Control block created for the Branch instructions.



AA, BA, DA		LR		Pc_to_RF		MB		BAS		FS	
Function	**Code**	**Function**	**Code**	**Function**	**Code**	**Function**	**Code**	**Function**	**Code**	**Function**	**Code**
R0	00000	No write	0	BusA	00	Register	0	Bus A	0	F <= A	000000
R1	00001	Write	1	PC	01	Constant	1	Constant in	1	F <= A+1	000001
R2	00010			Pc+4	10					F <= A+B	000010
R3	00011			nothing	11					F <= A+B+1	000011
R4	00100									F <= A + ~B	000100
R5	00101									F <= A + ~B +1	000101
R6	00110									F <= A -1	000110
R7	00111									F <= A	000111
R8	01000									F <= A and B	001X00
R9	01001									F <= A or B	001X01
R10	01010									F <= A xor B	001X10
R11	01011									F <= not A	001X11
R12	01100									F <= A << B	01XX00
R13	01101									F <= A >> B	01XX01
R14	01110									F <= A <<< B	01XX10
R15	01111									nothing	01XX11
R16	10000									F <= Less	10XXXX
R17	10001									F <= LessU	11XXXX
R18	10010										
R19	10011										
R20	10100										
R21	10101										
R22	10110										
R23	10111										
R24	11000										
R25	11001										
R26	11010										
R27	11011										
R28	11100										
R29	11101										
R30	11110										
R31	11111										

MD		DM_M		BR_C	
Function	Code	Function	Code	Function	Code
Function Unit	0	nothing	0000	Out <= 00	0000
Data In	1	SW	0001	Out <= 01	0001
		SH	0010	Out <= 11	0010
		SB	0011	Out <= {0,BLT}	0011
		LW	0100	Out <= {0,BGE}	0100
		LH	0101	Out <= {0,BEQ}	0101
		LB	0110	Out <= {0,BNE}	0110
		LHU	0111	Out <= {0,BLTU}	0111
		LBU	1000	Out <= {0,BGEU	1000
		nothing	1001	Out <= 00	1001
		nothing	1010	Out <= 00	1010
		nothing	1100	Out <= 00	1011
		nothing	1110	Out <= 00	1100
				Out <= 00	1101
				Out <= 00	1110
				Out <= 00	1111



PROGRAM COUNTER

There is a PC register to keep the PC value in the Program Counter structure. There is an ADDER circuit that we use in ALU to increase it. In addition, there are 2 separate 2x1 MUX to be able to choose specific to the instructions. The control signals required for selection come from the Branch Control Block. Depending on the control signal to the ADDER entry, 4, immediate, busA and the previous value of the PC are entered. Normally it changes as PC = PC + 4. When the branch values arrive, they are updated as PC = PC + signed extend immediate. Specially for the JALR instruction, it is updated as PC = signed extend immediate + BusA.

5-STAGE PIPELINE

In the pipeline design, we divided the circuit into 5 stages as Instruction Fetch (IF), Instruction decode/register file read (ID), Execute/address calculation (EX), Memory access (MEM) and Write back (WB). In the IF stage, the value of the PC came to the instruction memory and the instruction was read from the instruction memory. In addition, the pc+4 operation and the mux that selects the value of the PC are in this section. In addition, Hazard Detection Unit, one of the blocks created to prevent pipeline hazards, is also included in this section. In the ID stage, the relevant parts of the instruction value are sent to the Register File, the extend out block and Adder that calculates the PC + immediate value. Also, Control Unit is at this stage. There are also comparator module and extra MUXs added for Hazard solution. In the EX stage, there is the ALU and Forwarding Unit. Also, in this section, there is a multiplexer that selects Alu input, B output of Register or immediate. In addition, 2 extra MUX has been added for the forwarding unit. In the MEM stage, there is Data memory and MUXs with forward memory block for data memory hazard solution. In the WB stage, there is a multiplexer that selects the value to be written to the Register File. At this stage, the calculated value or the value to be read from the data memory is selected and written to the Register File. After these stages, the Control Unit, which regulates the control signals, is placed in the ID stage. The control signals of the blocks in the EX stage are delayed by 1 clock cycle, those in the MEM stage by 2 clock cycles, and those in the WB stage by 3 clock cycles. Moreover, ALU flag outputs are delayed by 1 clock cycle and given to the input of branch control block in MEM stage. Branch control block output is given to mux's select on the PC's input.

EXTRA BLOCKS FOR SOLVING HAZARDS

FORWARDING UNITS and MUXs

The forwarding unit is placed in the Execute stage. The addresses Rs1, Rs2, and Rd are passed through the ID/EX pipeline for the forwarding unit. Rs1 and Rs2 values passing through the ID/EX stage are given as input to the forwarding unit. Then the Rd address value is passed through the EX/MEM pipeline and the MEM/WB pipeline. These signals are also given as input for the forwarding unit. Forwarding unit has 2 output signals, ForwardA and ForwardB. These outputs are used as control signals. 4x1 multiplexers are connected to the A and B inputs of the ALU. The 1st input of these multiplexers is the value passed through the ID/EX pipeline from the Register File, the 2nd input is the value from the multiplexer at the WB stage, and the 3rd input is the value passed through the EX/MEM pipeline of the ALU output. 32'b0 signal is connected to its 4th input. One of these inputs is selected using ForwardA and ForwardB control signals. Thus, it is expected that the RAW hazard problem will be solved. If the incoming instructions are reading and writing from the same register, the forwarding unit condition written above will not work correctly. Because in the above condition, it is accepted that there will be no hazard in WB stage. In this case, the result should be transmitted from the MEM stage because the result in the meme stage is the newer result. In order to eliminate this hazard, the delayed versions of the RegWrite signal from the EX/MEM pipeline and the MEM/WB pipeline are given as input to the Forwarding Unit.

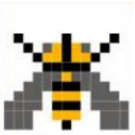
In addition, the forwarding block was added for control hazard. The reason for adding multiplexers is to add forward units similar to data hazards. For example, if add is performed before the beq instruction and the destination register of the add operation is used for beq, the value of the source register should be taken from the alu output of the next stage. Therefore, this situation is solved by adding codes for control hazards into the forward unit.

In addition, the forward structure is required when the load and store instructions use the same destination register in succession. The value in the data memory output is forwarded without being written to the register and enters the data memory. For this, a separate "forward_mem" module has been created and a multiplexer has been added.

HAZARD DETECTION UNIT

When the load instruction comes, it is necessary to stop the 1 clock cycle pipeline. To fulfill this function, a block named Hazard detection unit has been added to the system. The Hazard Detection unit takes inputs Rs1 and Rs2 from the IF/ID pipeline and Rd from the ID/EX pipeline. Since there is no MemRead signal in our system, the

information that Load instructions came from the MemWrite signal is transmitted to the Hazard detection unit. The output of the control block is connected to a multiplexer. 0 is connected to the second input of this multiplexer. Hazard detection unit has 3 outputs: PCWrite, IF/IDWrite and ControlMuxSelect. When the Hazard detection unit detects that a stall needs to be added, it makes the PCWrite and IF/IFWrite signals logic-0, preventing the PC value from being updated and adding new values to the IF/ID pipeline. In addition, by making ControlMuxSelect signal logic-1, 0 input is selected



from the multiplexer so that there is no confusion in the system, thus resetting control signals. Figure 7 shows the stalling condition of the pipeline. Figure 8 shows the entire system with the hazard detection unit added.

Also for control hazard, the next instructions from the branches are flushed according to the branch signals in the hazard detection unit. After this flush instruction, the pc value of the new address to go comes.

COMPARATOR

The following steps were followed for the control hazard solution.

- The adder that collects the immediate value with the PC has been moved to the ID stage.
- The branch control block has also been moved to the ID stage to obtain branch signals.
- However, the register values need to be compared to run the branch control. In the previous design, this process was performed by subtraction within the ALU. However, since this information was required in the previous state, a comparison circuit was added to the output of the register file. The outputs of this comparison circuit may not always come from the register file. For this selection, multiplexers are added to the inputs of the comparison circuit.
- The reason for adding multiplexers is to add forward units similar to data hazards. For example, if add is performed before the beq instruction and the destination register of the add operation is used for beq, the value of the source register should be taken from the alu output of the next stage. Therefore, this situation is solved by adding codes for control hazards into the forward unit.
- Finally, the next instructions from the branches are flushed according to the branch signals in the hazard detection unit. After this flush instruction, the pc value of the new address to go comes.

As mentioned, the stages with some modules were changed and additions were made to the forward unit and hazard detection unit. Differently, COMPARATOR is used here. It generates branch control signals by comparing the output of Register File or forwarded register values.

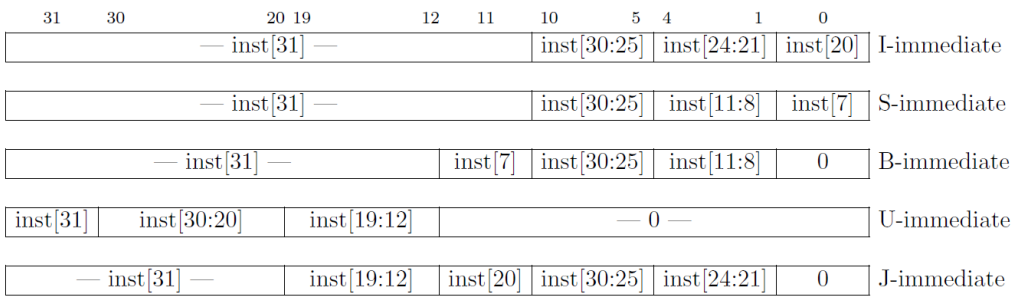
INSTRUCTION SET ARCHITECTURE

RISC-V 32I ISA is used in this processor. The instructions given below are supported. For this, "The RISC-V Instruction Set Manual" file was used, all instructions and operations were examined and a design was made accordingly.

In another image, the reference table we used to create the immediate structure is given. Immediate adjustment has been made according to the type of incoming instructions.

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND



CPI PERFORMANCE

Instructions	Arithmetic and Logic Instructions	Load Instructions	Jump İnstructions	Branch Instructions	Store Instructions
CPI	1 CPI	2 CPI	2 CPI	2 CPI	1 CPI

RESOURCE UTILIZATION AND PERFORMANCE

Firstly, 180 nm technology was used in this assignment. In the synthesis stage, the clock period was first tested as 10 ns. Area report is given in Figure 1, power report is given in Figure 2 and timing report is given in Figure 3.

```
================================================================

           Instance                    Module           Cells  Cell Area  Net Area   Total Area
-------------------------------------------------------------------------------------------------
processor                                               8912     398466    126873       525339
  RF                          Register_File             3512     172872     53006       225878
    MUXB                      MUX32x1_990                692      19697      9660        29357
    MUXA                      MUX32x1                    693      19685      9668        29353
    regx31                    Register_2639               66       4246       402         4649
    regx30                    Register_2640               66       4246       402         4649
```

Figure 1: Area Report

```
                                        Leakage     Dynamic       Total
           Instance            Cells Power(nW)    Power(nW)     Power(nW)
-------------------------------------------------------------------------
processor                       8912  1278.058 358345422.078 358346700.136
  RF                            3512   559.972 184454099.369 184454659.341
    MUXB                         692    54.133    772459.105    772513.238
    MUXA                         693    44.878    632600.864    632645.742
    regx3                         66    14.727   5920064.748   5920079.475
    regx22                        66    14.727   5777926.261   5777940.988
```

Figure 2: Power Report

```
Path 1: VIOLATED (-10379 ps) Setup Check with Pin pc_reg/Rout_reg[4]/CP->D
          Group: sys_clk
     Startpoint: (R) reg_rs2/Rout_reg[0]/CP
          Clock: (R) sys_clk
       Endpoint: (F) pc_reg/Rout_reg[4]/D
          Clock: (R) sys_clk

                    Capture       Launch
      Clock Edge:+   10000            0
      Src Latency:+      0            0
      Net Latency:+      0 (I)        0 (I)
          Arrival:=  10000            0

            Setup:-    101
    Required Time:=   9899
     Launch Clock:-      0
        Data Path:-  20278
            Slack:= -10379
```

Figure 3: Timing report

As can be seen from Figure Z, the slack value is -10379 ps. In the case of negative slack, the system works incorrectly because it cannot meet the timing restrictions. Therefore, the system was retested with different clock periods. It has been seen that the minimum clock period in which the system will operate is 40 ns. Area report in Figure 4, power report in Figure 5 and timing report in Figure 6 are shown for the 40ns clock period.

Instance	Module	Cells	Cell Area	Net Area	Total Area
processor		8618	386405	124136	510541
RF	Register_File	3508	173618	52973	226591
MUXB	MUX32x1_990	691	20089	9651	29740
MUXA	MUX32x1	690	20039	9643	29682

Figure 4: Area Report

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
processor	8618	1232.274	148023122.045	148024354.319
RF	3508	561.392	69553587.257	69554148.649
MUXB	691	54.541	1577983.829	1578038.371
MUXA	690	45.918	1933030.261	1933076.178

Figure 5: Power Report

```
Path 1: MET (54 ps) Setup Check with Pin pc_reg/Rout_reg[0]/CP->D
          Group: sys_clk
     Startpoint: (R) RF/regx27/Rout_reg[1]/CP
          Clock: (F) sys_clk
       Endpoint: (R) pc_reg/Rout_reg[0]/D
          Clock: (R) sys_clk

                    Capture       Launch
      Clock Edge:+   40000        20000
     Src Latency:+       0            0
     Net Latency:+       0 (I)        0 (I)
         Arrival:=   40000        20000

           Setup:-     127
   Required Time:=   39873
    Launch Clock:-   20000
       Data Path:-   19819
           Slack:=      54
```

Figure 6: Timing Report

Then to the TCL file

set_max_delay 30 -from [all_inputs] -to [all_outputs];

command has been added. It was found by trying the 30 ns value, which is the time limit in this command. It has been observed that this system works with a maximum clock period of 30 ns with the set_max_delay restriction instruction. Area report in Figure 7, power report in Figure 8 and timing report in Figure 9 are shown for 30 ns clock period.

Instance	Module	Cells	Cell Area	Net Area	Total Area
processor		8667	388641	124550	513191
RF	Register_File	3535	174641	53199	227840
MUXA	MUX32x1	704	20525	9760	30285
MUXB	MUX32x1_990	704	20462	9760	30222

Figure 7: Area Report

Instance	Cells	Leakage Power(nW)	Dynamic Power(nW)	Total Power(nW)
processor	8667	1243.433	182245693.034	182246936.467
RF	3535	565.057	81458737.537	81459302.594
MUXB	704	56.185	701356.505	701412.690
MUXA	704	47.034	1179017.362	1179064.396

Figure 8: Power Report

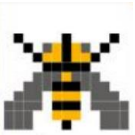
```
Path 1: MET (366 ps) Setup Check with Pin pc_reg/Rout_reg[4]/CP->D
          Group: sys_clk
     Startpoint: (R) RF/regx19/Rout_reg[1]/CP
          Clock: (F) sys_clk
       Endpoint: (F) pc_reg/Rout_reg[4]/D
          Clock: (R) sys_clk

                    Capture       Launch
       Clock Edge:+   30000        15000
      Src Latency:+       0            0
      Net Latency:+       0 (I)        0 (I)
          Arrival:=   30000        15000

            Setup:-     101
     Required Time:=   29899
      Launch Clock:-   15000
         Data Path:-   14534
             Slack:=     366
```

Figure 9: Timing Report

The table below compares cases where the clock period is 40 ns and 30 ns.

Clock Period	Area (μm^2)	Power (mW)
40 ns	510	148
30 ns	513	182

As can be seen from this table, the decrease of the clock period to 30 ns with the addition of a time constraint has a small effect in terms of space usage, but it greatly increased the power consumption.



Comments about what we experienced during the process and to improve the lesson:

By following lecture records, slides and reference books, we came up with a processor. Thus, we learned a lot about computer architecture. We ran it first on Vivado and then on Cadence because it was more efficient for us. Frankly, we think that the course teaches us more computer architecture than VLSI knowledge. We learned to use Cadence, but we couldn't go any further than we did in Vivado for all of the other assignments except for one assignment. And it came to us as a workload. Because the working environment is more difficult, there were cases where the server threw, the commands did not work, or simply because we could not read the instructions from the mem file, we had to write them one by one. Although we were excited to use Cadence at the beginning of the term, Cadence negatively affected us as a separate workload during the term. If we had done the processor design in Vivado, it would have been an easier and more fun process for us.

Another point I would like to mention was that the reference book changed throughout the projects. A different reference was given for the pipeline and hazard part and it seemed very difficult to add the previous reference to the pipeline structure and deal with hazards, so we tried to change the previous single cycle processor structure and bring it to the other reference, which took an extra day.

Another comment is that the time management of the course is poor. As in the last semester's DSDL course, the projects piled up to the end and the correct timing could not be made. In the first weeks of the course, the content is relatively weak and it accumulates to the end, especially because we did not do the final project in this course because of the time. Timing can be provided by giving all of them earlier next year. Verilog courses at the beginning of the semester can be added as an additional course as a resource because most people already know it, so they can all be brought forward.

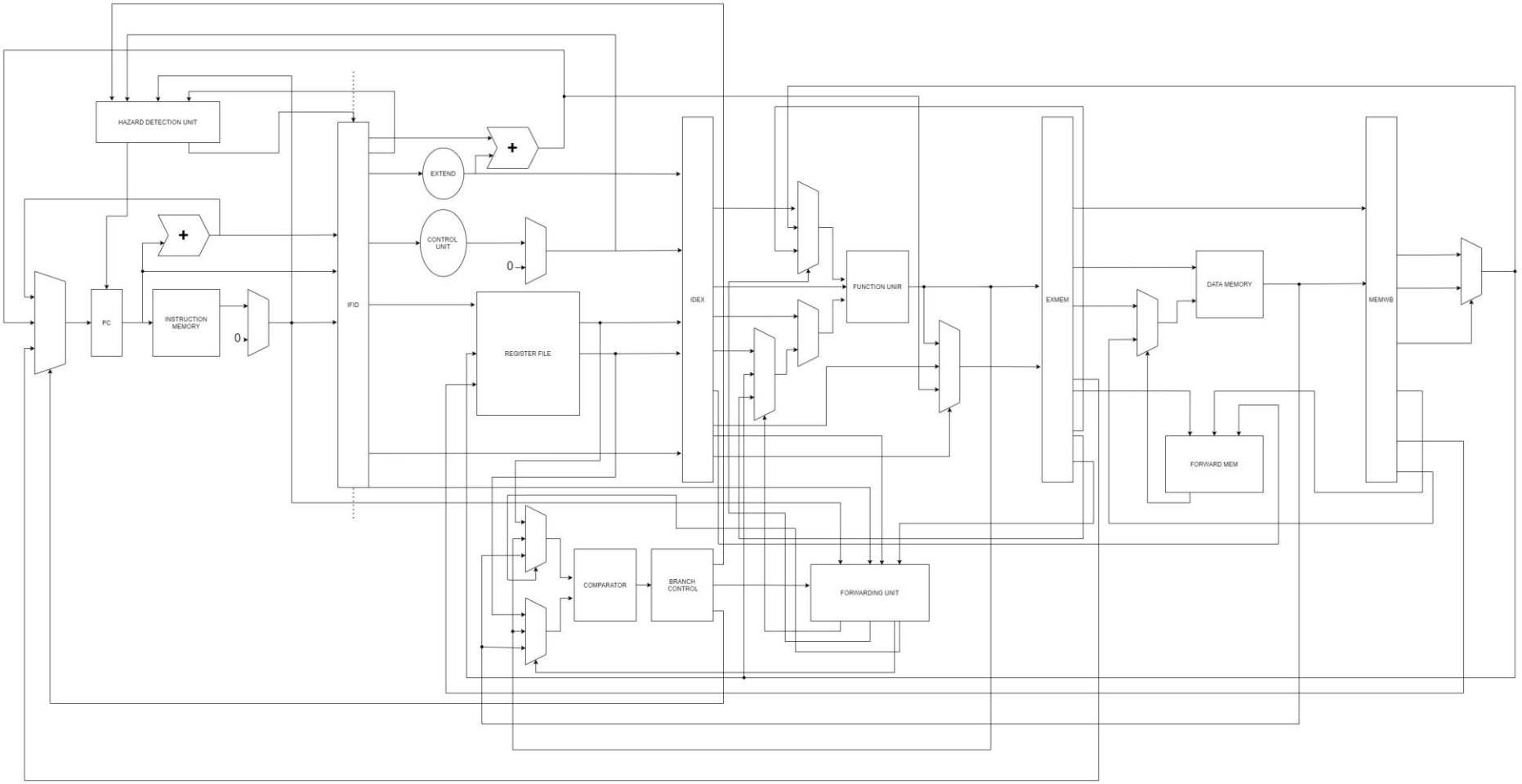


Figure 10: Processor Diagram for Final