Multi Cycle Processor Design with Multiplication accumulate

Rishi Jain

2020CS10373

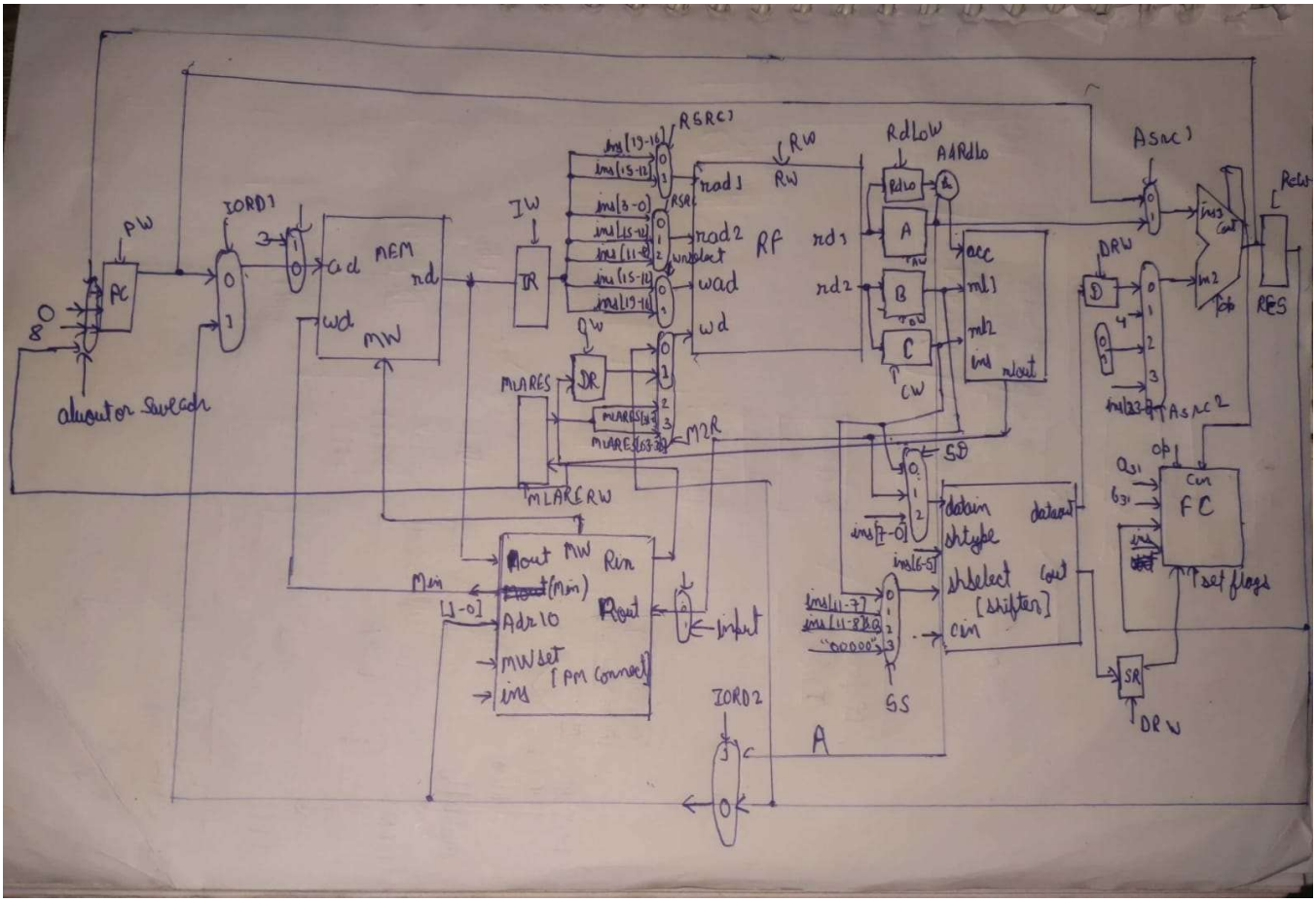
How to run?

To run the processor load all the components except tesbenches(starts with testbench) and the pdf with processor.vhd as the top entity. An external clock must be provided to run the processor. To execute any program, initiate data memory with the code in the dataMem component in datamem.vhd file. The last 64 indices i.e 64*4 bytes are for user instructions and the first 64 indices i.e 64*4 bytes are for priviliged or restricted memory. There are testbenches for testing various instructions with the name testbench.

To check the result, we have to check the values of the register. This can be done on EDA Playground with Aldec Rivera Pro and EP wave. In the EP wave reg1Sel and reg2sel are the signals for the two read signals of the RegisterFile and dataout1 and dataout2 are the data corresponding to these signals. One can also check the datainput, writeadd and write enable in the registerFile signals.

Design: -

Datapath of the processor is: -



The design is based on what has been taught in the lecture 10 and lecture 9. Two components Datapath and Controlpath relate to each other. I've added shifter between ALU input and the register B.

Changes from previous part: -

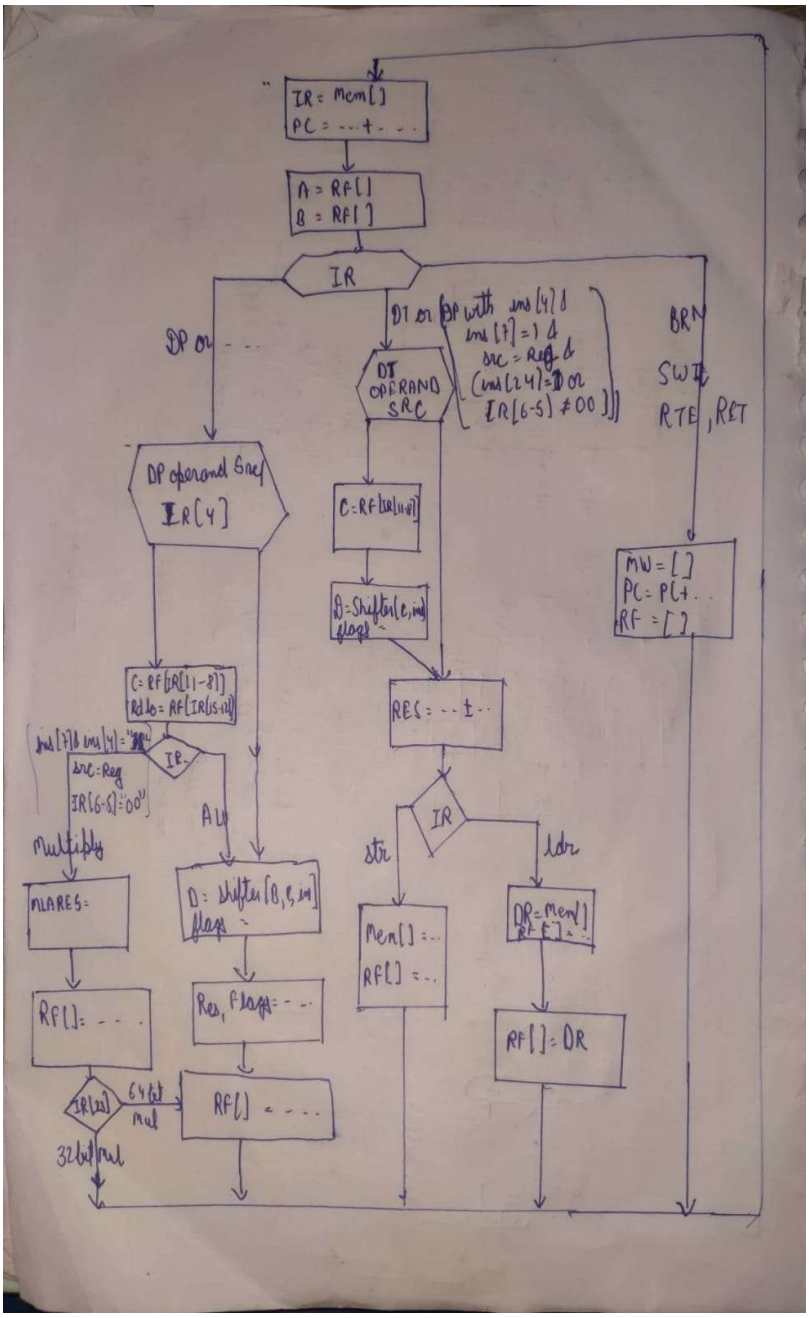
Shifter and multiplier carry is now used to set flags. For multiplier only Z and N carry bits are set when s bit is 1.

Data memory contains two partitions. For system program data can be written from byte 0 to 255 and for user program data can be written from byte 256 to 512. If mode flag is 1 then only, we are able access system part. The 0^{th} location contains branch instruction to 0x20 and on 0x24 we have mov r14,#256 and on 0x28 we have rte instruction and thus program starts again. On 0x08 we have branch instruction to 0x40 and on 0x40 we have mov r0,#12 and on 0x44 we have ldr r0,[r0] and thus r0 takes the stored value from 0x0c which is the input value and then on 0x48 we have rte and the program continues.

PC can take input from 0,8 alu result and register B for various uses. Also, PM connect will take data from user input and from Register.

Control Path: -

The control part takes instruction and flags as input from the datapath and uses decoder to decode various types.



Control part has a 15 state FSM which changes its state on rising clock edge according to the FSM below:

Changes in control path: -

SWI instruction is done through branch instruction in state 8. In state 8 if instruction is of bl type or of SWI type then writing into r14 is done also for input writing into memory is also done in state 8. For RET

and RTE instruction in state 1 where reading for register file is done, the data in r14 is stored in register B and then from state 1, system goes to state 8 and then writing into pc is done. Similar for RTE but here mode changes to 0.

Full predication was already done in previous part. For reset signal I've made PC =0 and then the instructions from the starting follows.

Apart from the FSM the controlpath has a circuit which determines the 29 control signals for every stage. The control part uses the flags and instruction from the datapath to determine these control signals. It uses the conditionchecker component from conditionchecker.vhd file to determine if conditions are satisfied or not.

Testing:-

1st Test Case: - Testing SWI, RTE, Predication, BL and RET instructions.

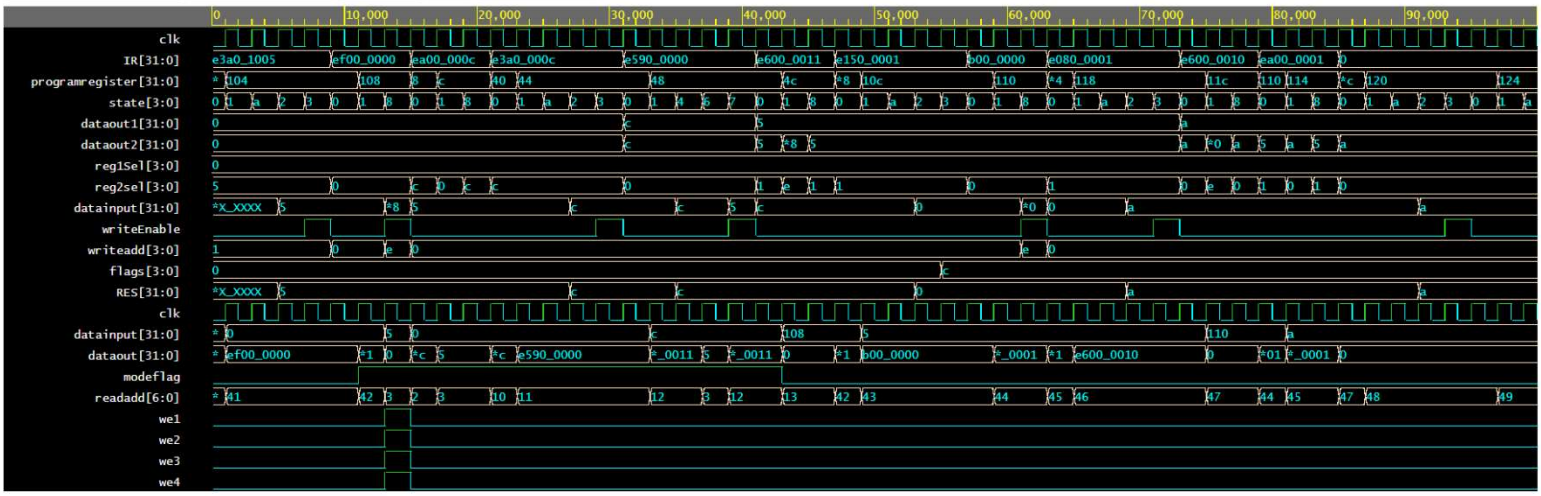
Arm code: -

```
mov r1,#5
swi 0x123456
cmp r0,r1
bleq L
G: b end
L: add r0,r0,r1
mov pc,lr
end :
```

The data memory array is initialized with instructions: -

```
Memory:=(0 => X"EA000006",2 => X"EA00000C",8 => X"E3A0EC01",9 => X"E6000011",16
=> X"E3A0000C",17 => X"E5900000",18 =>
X"E6000011",64=>X"E3A01005",65=>X"EF000000",66=>X"E1500001",67=>X"0B000000",68=>X
"EA000001",69=>X"E0800001",70=>X"E6000010",others => X"00000000");
```

EP wave for the following instruction is:-



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

I've taken input from user and then compared it with 5. If it is 5 then adding 5 to r0 is done else program ends. And as we can see that input was 5, thus r0 gets value 10 at the end.

2nd Test Case: - GCD function which checks all instructions.

Arm code: -

```
.text
   mov r0,#245
   mov r1,#400
   bl gcd
   swi 0x06

gcd:
   cmp r0,r1
   moveq pc,lr
   blt sublt
subgt:
   sub r0,r0,r1
   b gcd
sublt:
   sub r1,r1,r0
```

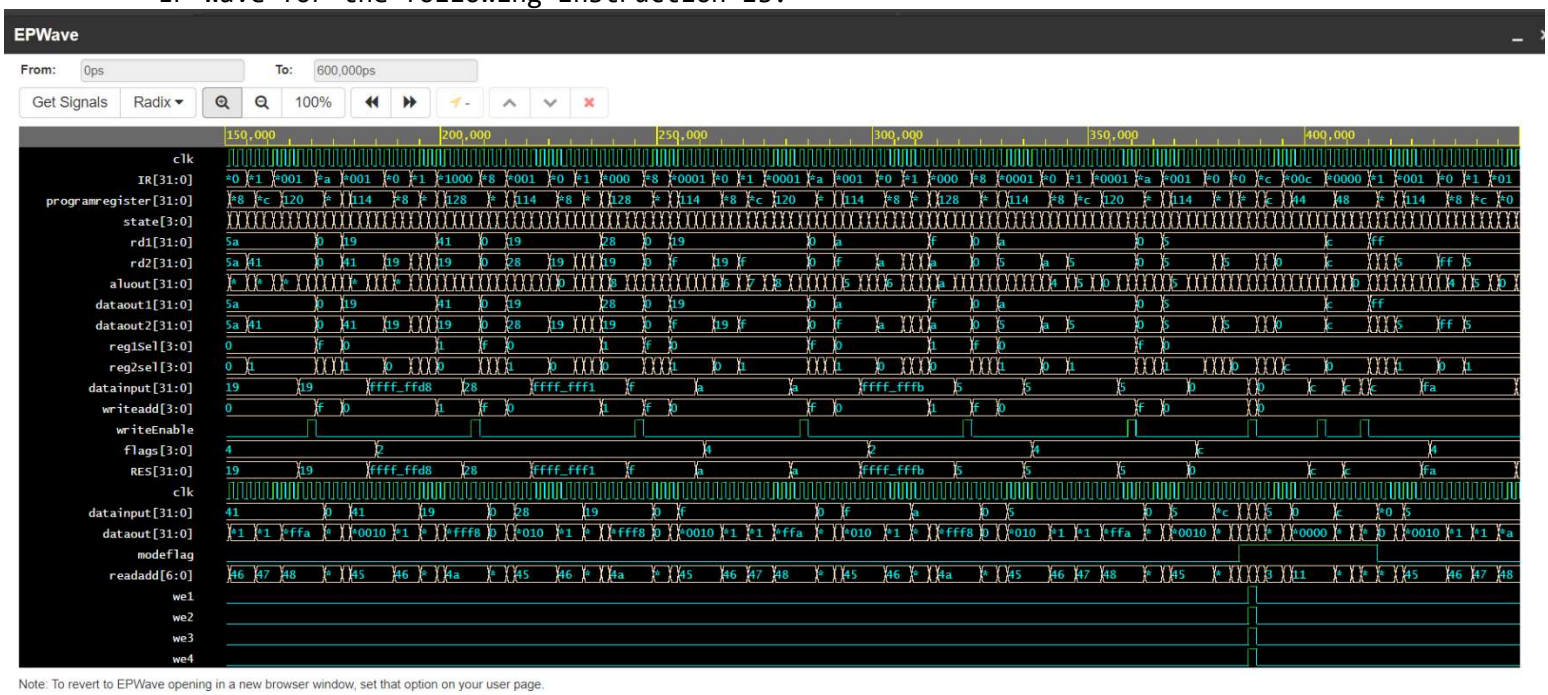
b gcd

.end

The data memory array is initialized with instructions: -

```
(0 => X"EA000006",2 => X"EA00000C",8 => X"E3A0EC01",9 => X"E6000011",16 =>
X"E3A0000C",17 => X"E5900000",18 => X"E6000011",64 => X"E3A000F5",65 =>
X"E3A01E19",66 => X"EB000000",67 => X"EF000000",68 => X"E1500001",69 =>
X"06000010",70 => X"BA000001",71 => X"E0400001",72 => X"EAFFFFFA",73 =>
X"E0411000",74 => X"EAFFFFF8",others => X"00000000");
```

EP wave for the following instruction is:-



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

This functions calculates GCD of 400 and 245 and the final value should be 5. We can see that final both r0 and r1 contains 5 from the ep wave plot.

3st Test Case: - Testing Mul and MLAL instructions

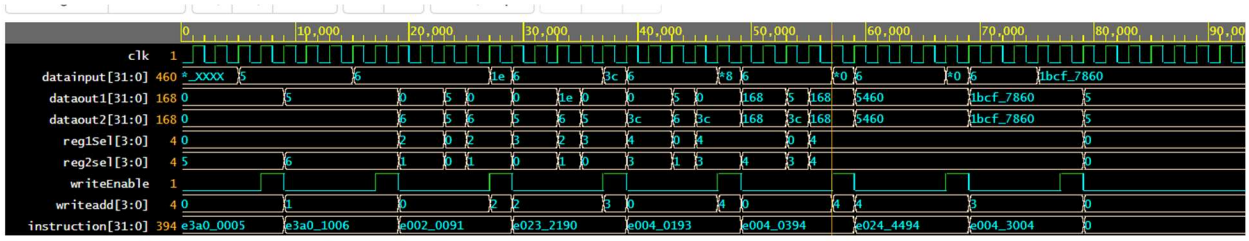
Arm code: -

```
mov r0,#5
mov r1,#6
mul r2,r1,r0
mla r3,r0,r1,r2
mul r4,r3,r1
mul r4,r4,r3
mla r4,r4,r4,r4
and r3,r4,r4
```

The data memory array is initialized with instructions: -

```
 (64=>X"E3A00005",65=>X"E3A01006",66=>X"E0020091",67=>X"E0232190",68=>X"E0040193"
,69=>X"E0040394",70=>X"E0244494",71=>X"E0043004",others => X"00000000" );
```

EP wave for the following instruction is:-



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

I've checked the program signals with the armsim outputs after each instruction. This covers all types of load instructions. 5*6 = 30 i.e. 0x1e then 30 +30 = 60 i.e., 0x3c and then 60 *6 = 360 i.e., 0x1ce.

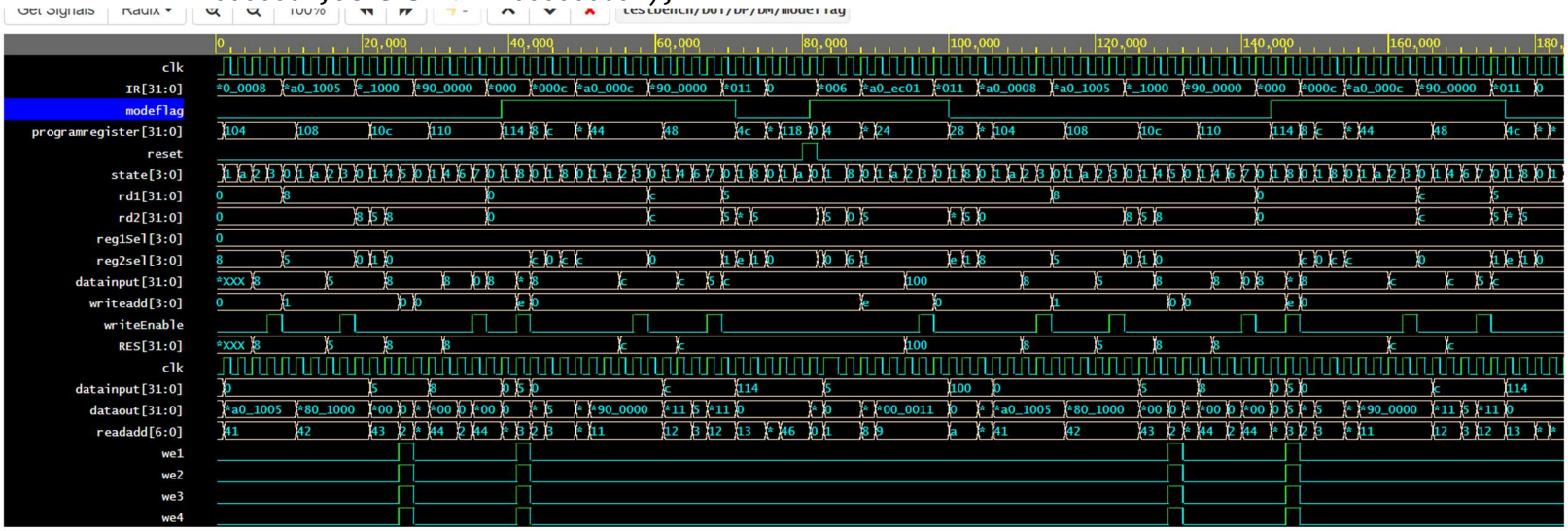
4th Test Case: - Testing of reset instructions and accessing privileged memory

Arm code:-

```
mov r0,#8
mov r1,#5
str r1,[r0]
ldr r0,[r0]
swi 0x06 --read instruction
```

Data memory is initialized with instructions:-

```
Memory:=(0 => X"EA000006",2 => X"EA00000C",8 => X"E3A0EC01",9 => X"E6000011",16
=> X"E3A0000C",17 => X"E5900000",18 =>
X"E6000011",64=>X"E3A00008",65=>X"E3A01005",66=>X"E5801000",67=>X"E5900000",68=>X
"EF000000",others => X"00000000");
```



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

From the EP wave it is clear that when we read from memory position 8 we get 0 and it didn't write in the 8th position as Branch is taken while SWI interrupt. Also, when reset bit is set then program register started from 0 and then instructions are continued from the user area.

5th Test Case: - Testing predication and read instruction if status bit is 0

Arm code :-

```
mov r0,#2
mov r1,#1
cmp r0,r1
moveq r3,#7
addgt r3,r0,r1
subeq r3,r3,r2
muleq r3,r1,r2
swi  0x06 –-wait for input
```

The datamemory array is initialized with instructions:-

```
 Memory:=(0 => X"EA000006",2 => X"EA00000C",8 => X"E3A0EC01",9 => X"E6000011",16
=> X"E3A0000C",17 => X"E5900000",18 =>
X"E6000011",64=>X"E3A00002",65=>X"E3A01001",66=>X"E1500001",67=>X"03A03007",68=>X
"C0803001",69=>X"00433002",70=>X"00030291",71=>X"EF000000",others =>
X"00000000");
```

Note: To revert to EPWave opening in a new browser window, set that option on your user page.

EP wave for the following instruction is shown above

Clearly no write back is done in moveq, subeq and muleq instructions. Write back is only done in addgt instructions. Also while the status bit of input was 0, state didn't changed and system waited for the input.

Synthesis: -

RESOURCE UTILIZATION BY DATAPATH: -

Flow Summary

<<Filter>>

Flow Status	In progress - Sat Apr 2 23:50:32 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	alu
Top-level Entity Name	datapath
Family	Cyclone IV E
Total logic elements	10,108
Total registers	4645
Total pins	8 4645
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	8
Total PLLs	0

RESOURCE UTILIZATION BY CONTROLPATH: -

Flow Status	Successful - Sat Apr 2 23:42:38 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	alu
Top-level Entity Name	controlpath
Family	Cyclone IV E
Total logic elements	114 / 6,272 (2 %)
Total registers	4
Total pins	79 / 180 (44 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP4CE6F17C6
Timing Models	Final

alu

Synthesis of processor

Flow Summary

<<Filter>>

Flow Status	Successful - Sat Apr 2 23:52:49 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	alu
Top-level Entity Name	processor
Family	Cyclone IV E
Total logic elements	0 / 6,272 (0 %)
Total registers	0
Total pins	11 / 92 (12 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP4CE6E22C6
Timing Models	Final

RTL view of whole Processor(Using Quartus):-