Design flow for Multi-Cycle RISC-IITB

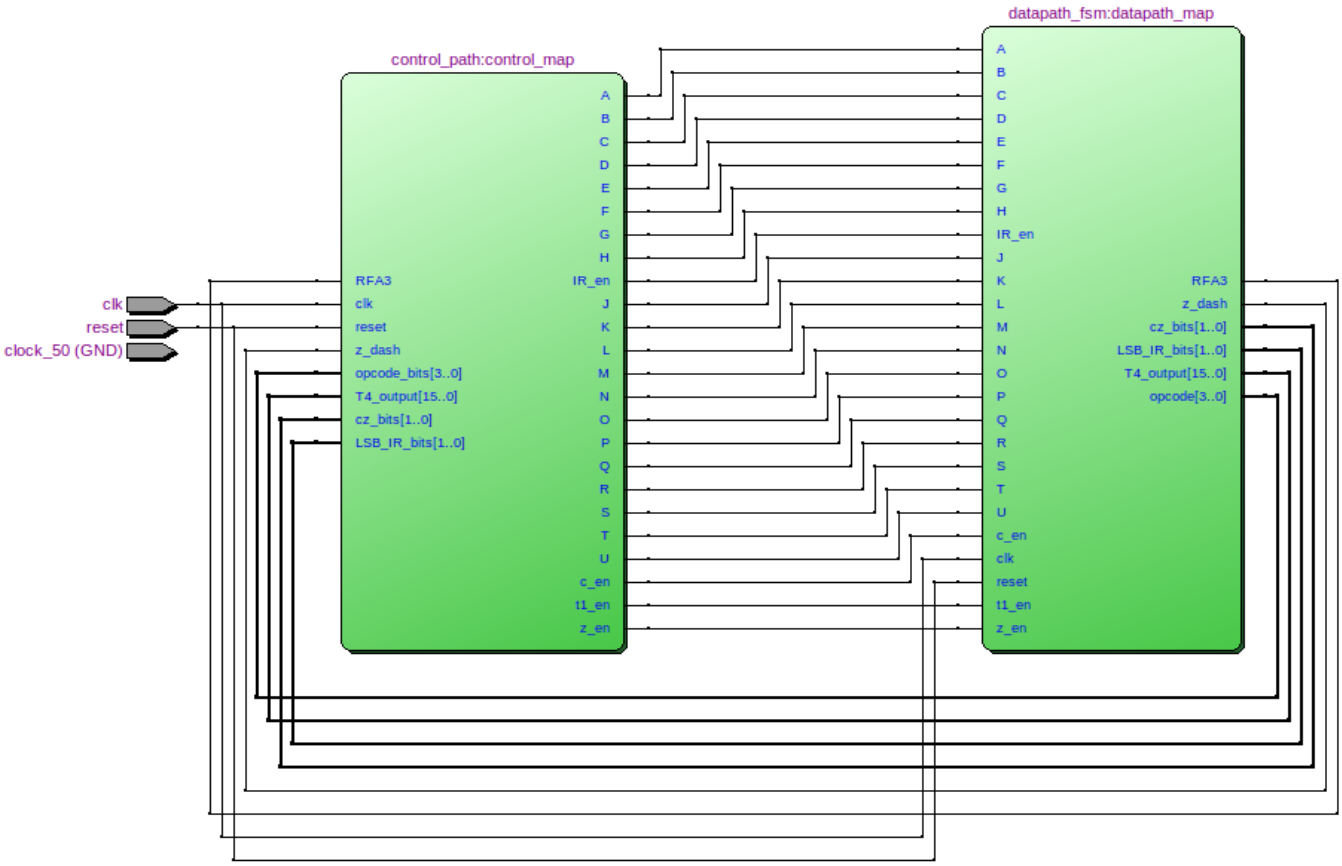
Our entire project includes four files:

1) datapath_components.vhd – This includes all of our entity components for example ALU, MUX, Register Files, registers, Asynchronous Memory, priority Encoder, zero decoder, Left Shifter,
Bit Extender etc. These entities are included in a library package which is further used in the below mentioned files.

2) datapath.vhd – The datapath lays out the entire flowgraph of our multi-cycle RISC design. It includes all the connections between various instances of components.

3) controlpath_fsm.vhd – This file implements the FSM, the decoder, the next state logic as well as generates the values of the control pins which are required for the correct functioning of the datapath

4) DUT.vhd – The DUT is the top-level entity of our entire project which maps the pins of the controlpath_fsm and the datapath.



Implementation details of LM and SM

Initial implementation :- Initially we were storing the last 8 bit immediate data of IR into T4(temporary register which stores the information about which registers need to be loaded or stored(LM/SM) and checking each individual bit whether the register corresponding to that bit needed to be loaded/stored resulting in wastage of cycles in case the bit corresponding to that register was 0. Hence we shifted to a new implementation of LM/SM where we used a **priority encoder** and a **zero_decoder** (which outputs a 8-bit number which is bit-extended and added with T4 to scrap that bit whose register load/store operation has been done.

Final implementation :- We observed that this way we have to check every bit in the 8 bit immediate data, for example in case when only one of the bits is 1(that is only one of registers need to be modified/stored) even then this will run

for 8 loops(24 cycles) checking each bit of T4 individually. So we decided to use a priority encoder which will reduce the number of loops to only the number of registers that need to be modified/stored in memory. The output of the PE tells us which register we need to load/store next and after each loop we scrap that registers entry in T4 using zero_decoder. Also if none of the registers need to be loaded/stored then we have kept a check in the beginning so that it doesn't gets inside the loop and waste clock cycles. Maximum cycles taken by our implementation in case last 8 bits are all 1's = 25 cycles.

Errors/Problems/Difficulties we faced and How we resolved them

Asynchronous memory initialization Error - While implementing the design on FPGA, we were getting compilation error on the board saying

```
Error: Cannot synthesize the initialized RAM logic on board
```

but the same code was running correctly/perfectly on modelsim. Initially we initialized memory like this -

```
signal marray: MemArray(0 to ((2**(addr_width-12))-
1)):=(0=>"0001100000000100",1=>"0001011000000101",2=>"0001010000000011",others=>"0000000000000
000");
```

But later we realized that we need to initialize the memory when reset is pressed and hence we assigned the values to `marray` only when reset is pressed -

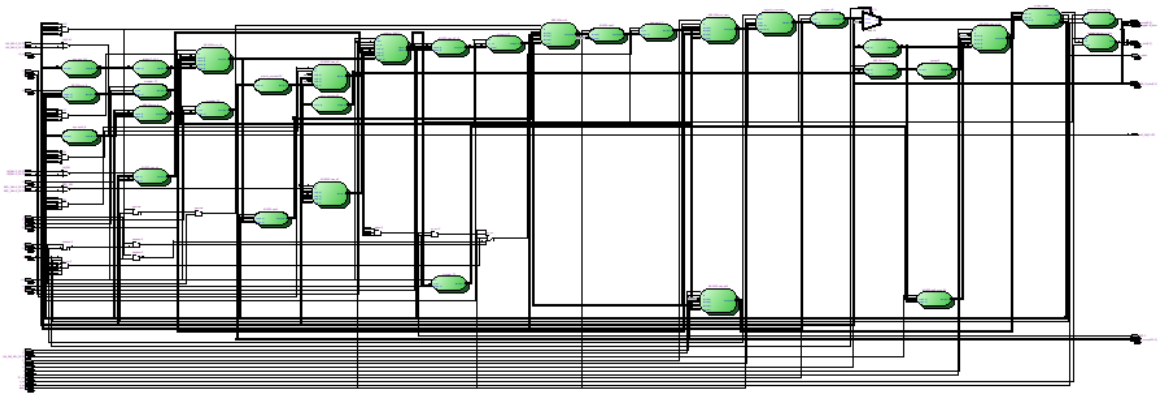
```
if (reset = '1') then
marray<=(0=>"0001100000000100",1=>"0001011000000101",2=>"0001010000000011",others=>"0000000000
000000");
end if;
```

Using of an extra blank state after the fetch state - Initially we were fetching the instruction in the fetch state S1 so that we will get to know the opcode by the end of that state when IR(Instruction Register) will be written but we realized that we need an extra state S0 after S1 where we can decode the opcode from IR which needed for the next state logic implementation.

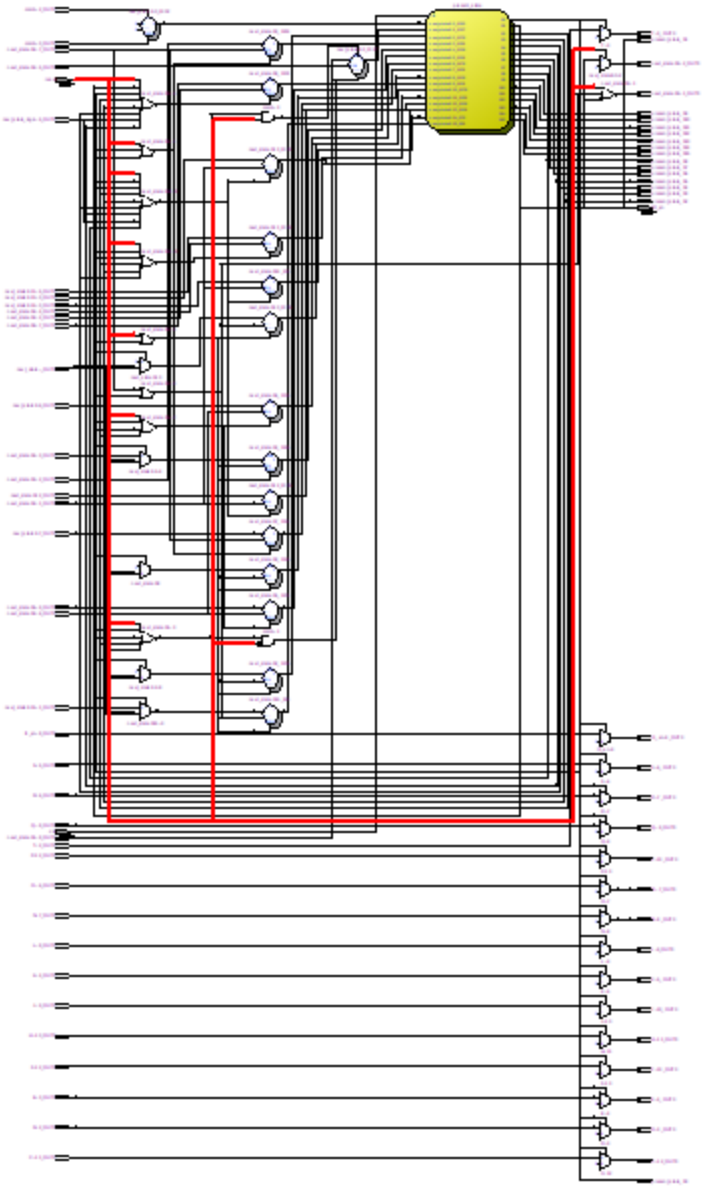
Some precautions/advice which we learnt

1. Inside a process statement instead of using an if/else statement, using case statement is much efficient as it saves time.
2. Never leave an if-else-if without an else(when none of the cases match) otherwise it will lead to formation latch i.e always write a default else statement so that values don't remain undefined

Some more pics from our project -



Entire Datapath Flowgraph



Contolpath_fsm

Github Link to Project Repository
https://github.com/saqib1707/Sem-5/tree/master/Microprocessors-Lab/Multicycle-RISC-IITB

Project Members:
1) Saqib Azim - 150070031
2) Awanish Kumar - 15D070037
3) Anirudh Kumar Yadiki - 150070056