EE337 Project Report

Thariq S M
150070058

Design Files

- ALU.vhd
  The Arithmetic and Logical unit capable of adding and NAND-ing two 16 bit numbers, and gives carry and zero flags along with result as output
- Comparator.vhd
  The comparator used for the BEQ instruction
- Components.vhd
  Has all components consolidated into a package, and small entities like the D Flip Flop defined inside.
- Controller.vhd
  The control bits, next state function and FSM flip flops are defined here.
- Datapath.vhd
  The datapath of the processor. All registers and registerfile, memory, etc are instantiated here.
- Memory.vhd
  The data and instruction memory. A few instructions have been hardcoded into this VHDL file.
- Registerfile.vhd
  The register file which contains eight registers.
- Toplevel.vhd
  The controller and datapath are instantiated and interfaced here.

Difficulties faced

The most difficult error to debug was the one induced by not mentioning all signals used in a process in its sensitivity list. The register file was working as expected when tested independently yet failed to function within the datapath because of this issue.

The reset button in the FPGA is active low - the top level of the design had to be modified slightly to ensure that the processor is not reset in every cycle.

In the initial paper design, the cycle gap between instruction fetch and instruction decode was not considered. After realising the mistake, an additional register has been placed in the datapath between the memory and instruction register, and the Hardware Flowchart has been entirely restructured, so as to incorporate this change in a way which keeps performance maximum.

As a result of these changes, we have been able to run ADD, NAND, ADI, LHI instructions in 3 cycles, while the others except LM,SM in 4 cycles. This has been done without placing any overhead (other than the absolutely necessary delays of combinational components, which is the same as that of any minimal design) on the cycle time.

The next complication came during the FSM design of LM and SM. It was originally thought that even if the corresponding bit of the register was 0, memory location had to be updated by moving to the next one, i.e. we believed that if the memory address starting was 1000 and the last eight bits in the LM instruction are 10010010, the memory locations to be updated are 1000,1003 and 1006.

This had given rise to an immensely large and complicated structure for the FSM. However, once the mistake was realised, that memory location needs to be updated only when when bit is 1, it was a straightforward procedure to design their FSM.

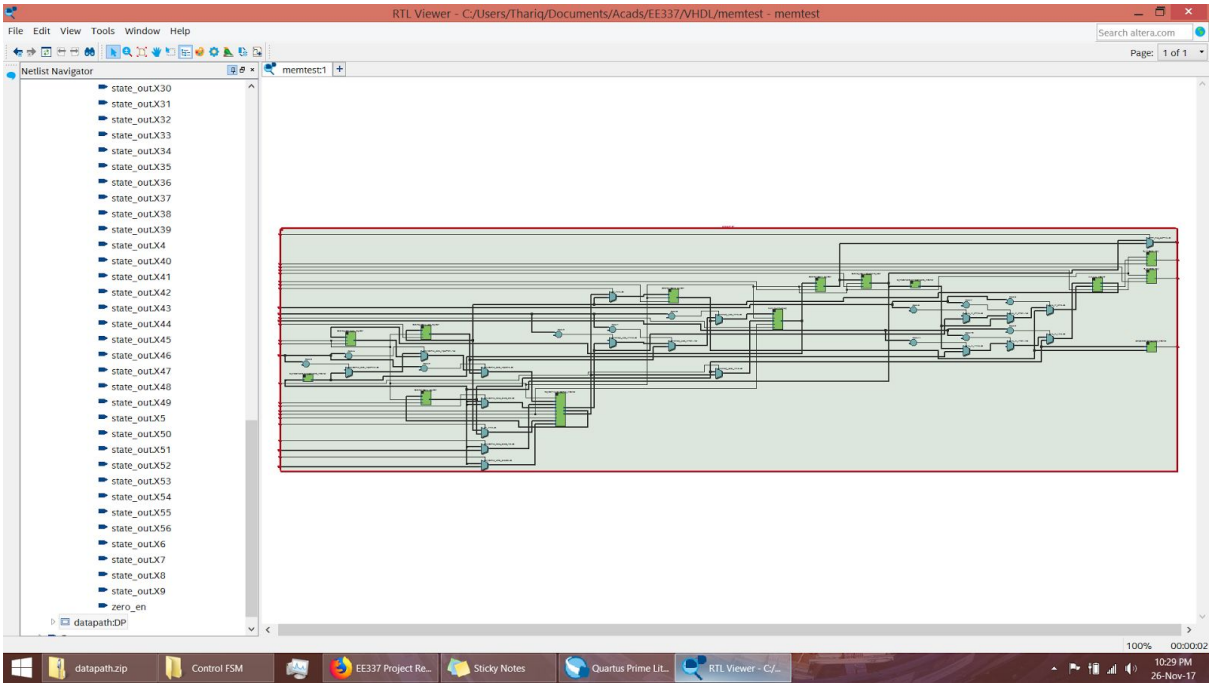
The controller has been designed as a Moore machine. As a result, there are more states than can be expected from a Mealy machine. This has led to the creation of multiple “similar” states, especially for LM and SM. However, there has been absolutely no compromise in performance. Also, we have avoided making a counter, or any other FSM besides the control FSM. This has led to faster execution, but at the cost of high number of states for LM and SM.

Design Flow

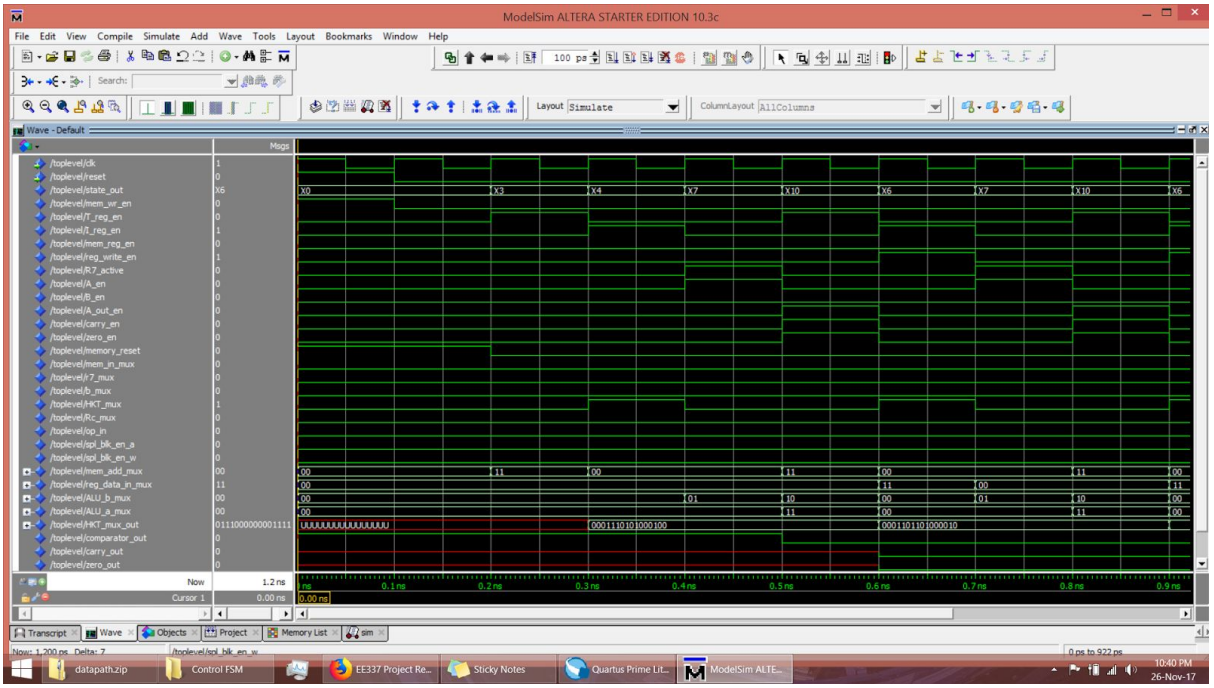
We started the design by drawing a rough datapath, and adding more hardware as required by specific instructions as we wrote the level one flowcharts. We made each of the components in VHDL and tested it by forcing the inputs, and again after everything was put together in the datapath.

LM and SM Instructions

LM and SM have been designed as follows:
Since counters have not been used, corresponding to each Register bit, there is a State. Hence, after the initial setup state, if R0 bit is 1, it goes to state S0. Else if R1 bit is 1, it goes to state S1.And so on. Finally, if R7 is 1, it goes to state S7, else it goes to state S8. This is the conditional next state function corresponding to the initial state.
Next state function for S0 is pretty much the same, except that the if condition starts from checking if R1 bit is 1.
Same is applied to all states except S7 and S8, which have a simple terminating next state.



RTL Netlist of Datapath



RTL Simulation