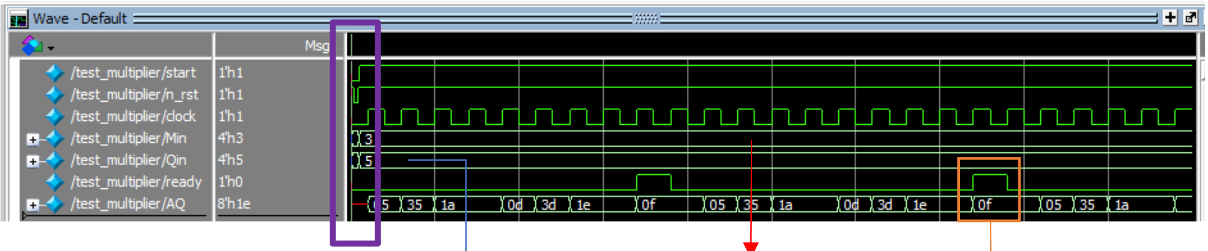
M4 Lab si3g19

SystemVerilog and FPGAs

3 Laboratory Work

3.1 RTL Simulation

Appropriate waveforms from ModelSim:



Wave - Default

	Msg
/test_multiplier/start	1'h1
/test_multiplier/n_rst	1'h1
/test_multiplier/clock	1'h1
/test_multiplier/Min	4'h3
/test_multiplier/Qin	4'h5
/test_multiplier/ready	1'h0
/test_multiplier/AQ	8'h1e

START signal

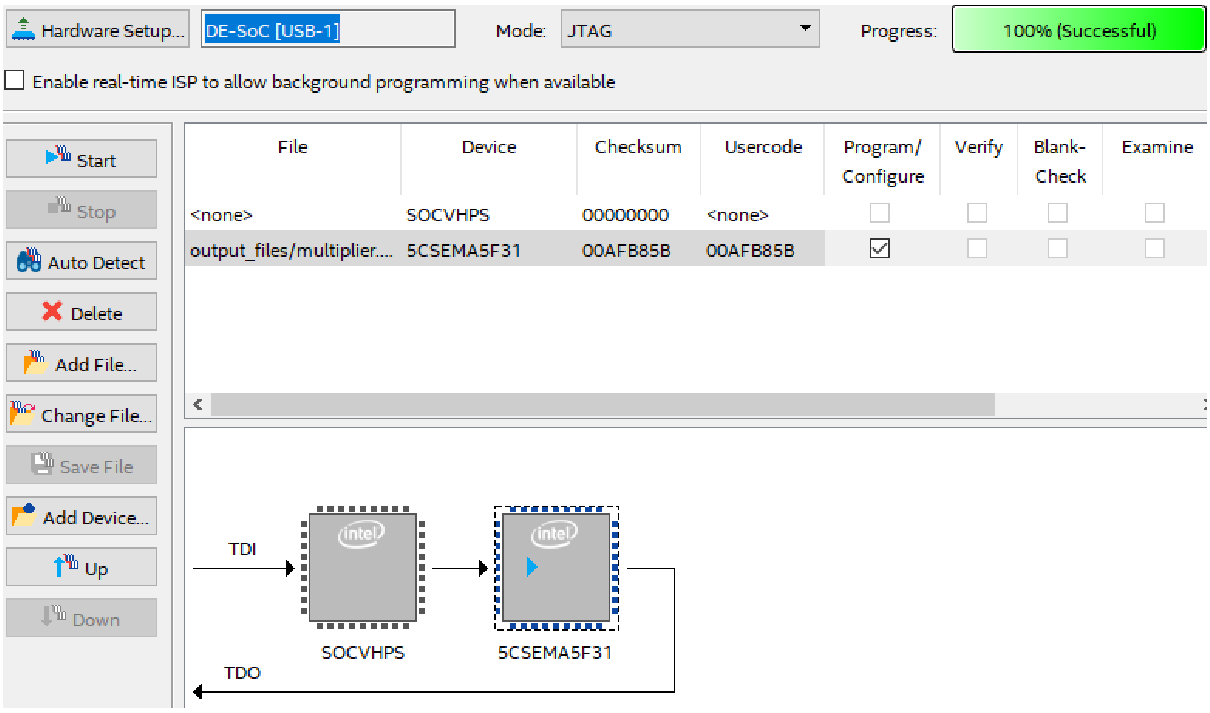
Multiplier Q

Multiplicand M

Product of M and Q, held in registers A and Q, once the active high output READY is asserted

3.2 FPGA Synthesis

1. Design successfully downloaded onto FPGA



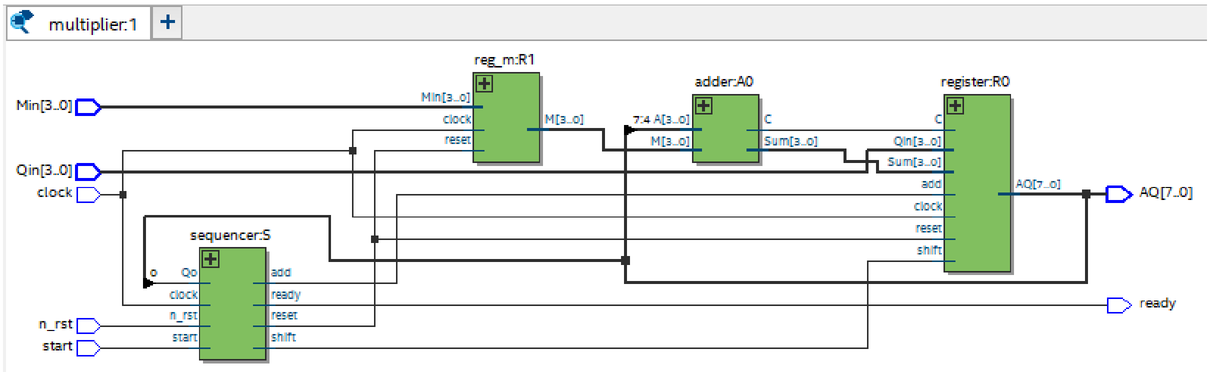
Hardware Setup... DE-SoC [USB-1] Mode: JTAG Progress: 100% (Successful)

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
<none>	SOCVHPS	00000000	<none>	☐	☐	☐	☐
output_files/multiplier....	5CSEMA5F31	00AFB85B	00AFB85B	☑	☐	☐	☐

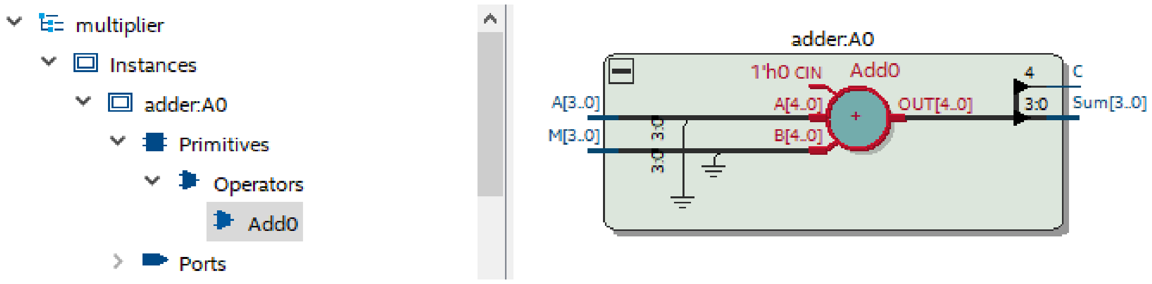
2. Schematic of your design from the RTL viewer in Quartus

Multiplier:

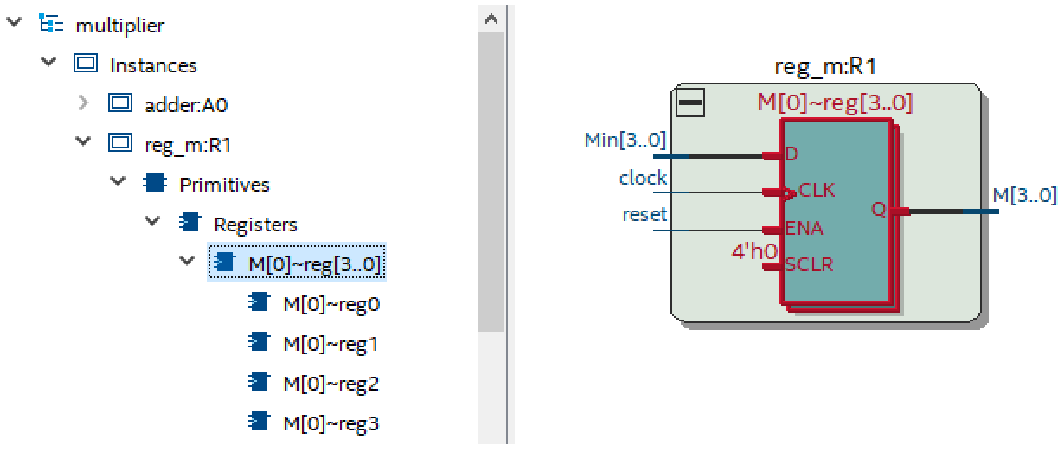


Instances:

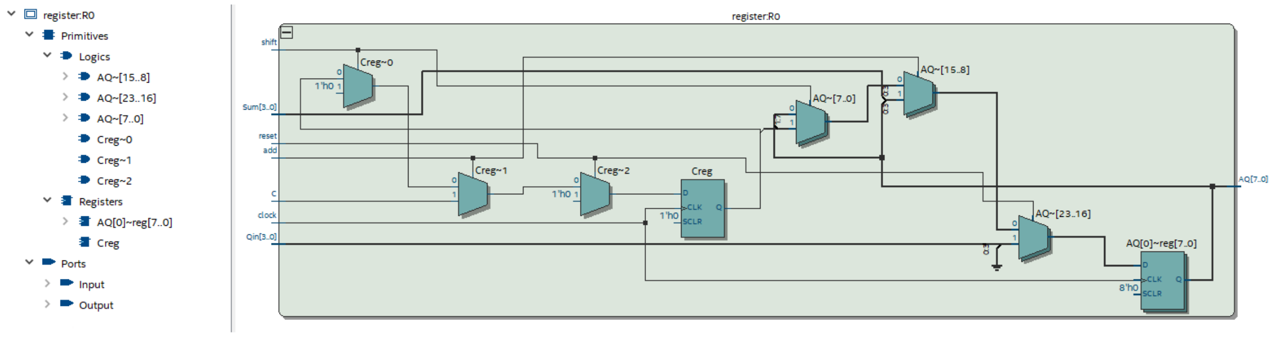
- Adder A0



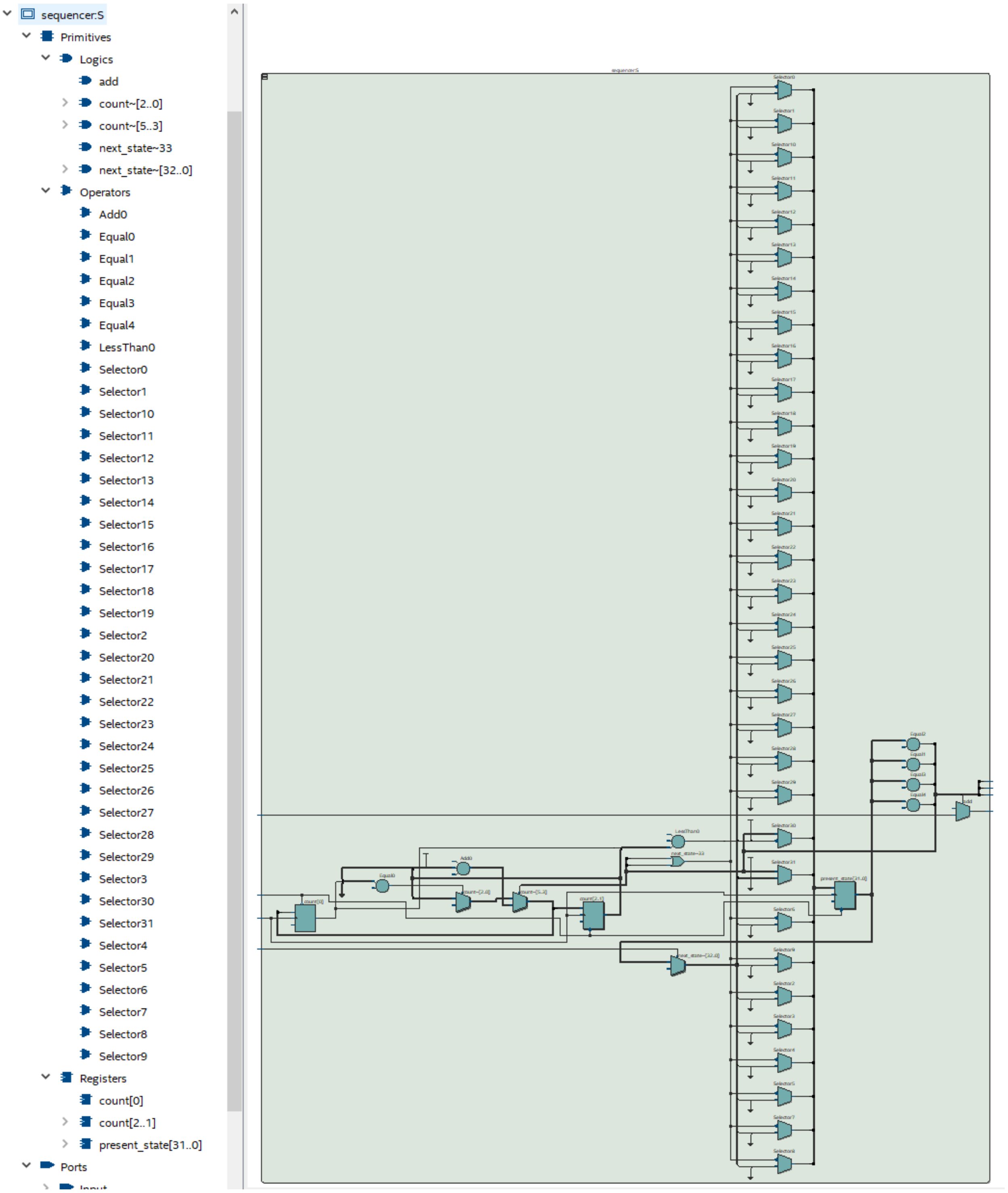
- Reg_m R1



- Register R0

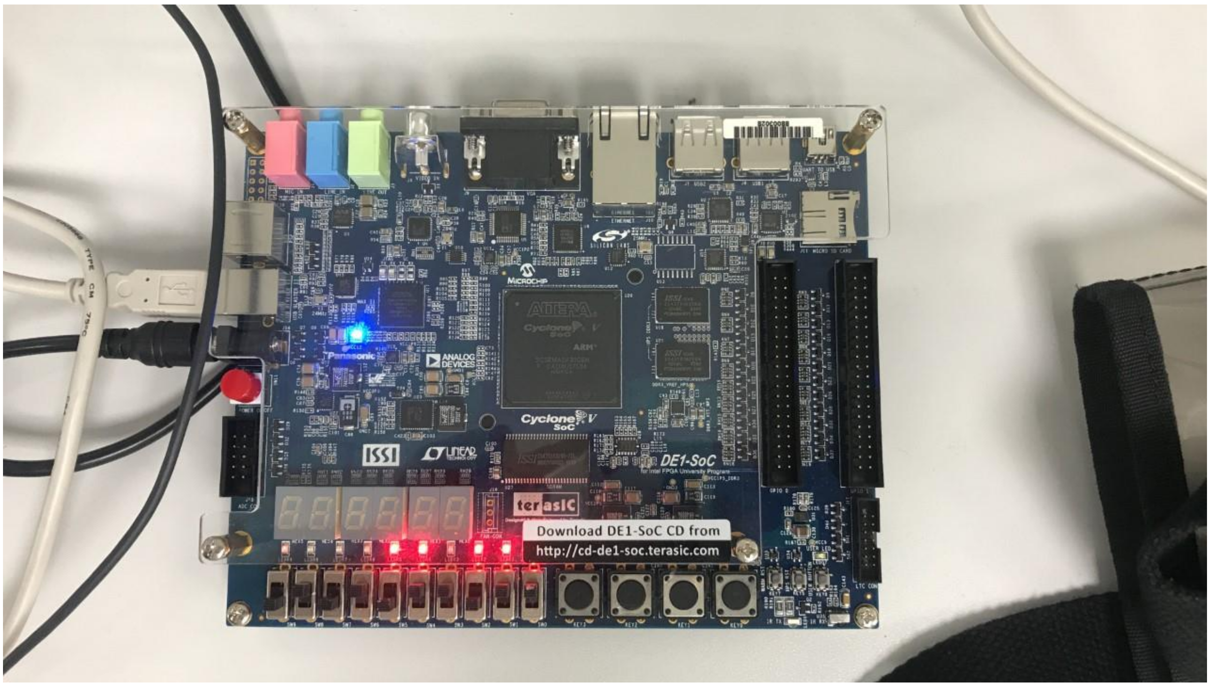


- Sequencer S



- sequencer:S
  - Primitives
    - Logics
      - add
      - count~[2..0]
      - count~[5..3]
      - next_state~33
      - next_state~[32..0]
    - Operators
      - Add0
      - Equal0
      - Equal1
      - Equal2
      - Equal3
      - Equal4
      - LessThan0
      - Selector0
      - Selector1
      - Selector10
      - Selector11
      - Selector12
      - Selector13
      - Selector14
      - Selector15
      - Selector16
      - Selector17
      - Selector18
      - Selector19
      - Selector2
      - Selector20
      - Selector21
      - Selector22
      - Selector23
      - Selector24
      - Selector25
      - Selector26
      - Selector27
      - Selector28
      - Selector29
      - Selector3
      - Selector30
      - Selector31
      - Selector4
      - Selector5
      - Selector6
      - Selector7
      - Selector8
      - Selector9
    - Registers
      - count[0]
      - count[2..1]
      - present_state[31..0]
  - Ports
    - Input

Testing:



Gave a cycle of

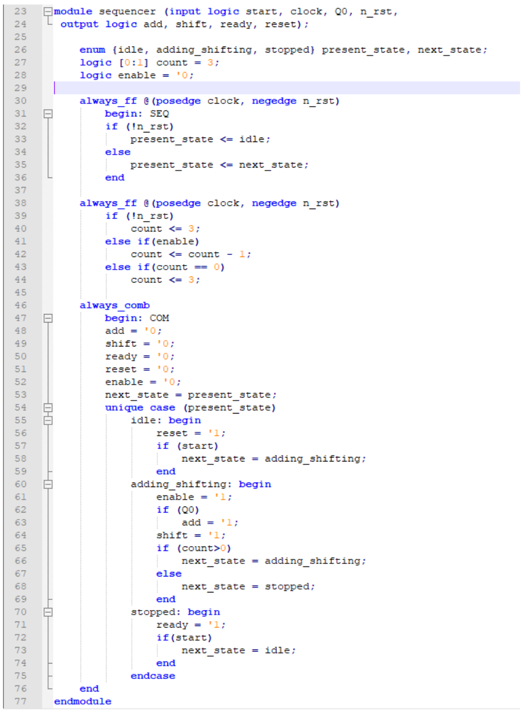
- 0100110111
- 0000110111
- 0000000101
- 0010110101
- 0001011010
- 0001011010
- 0000101101
- 0000101101
- 0011011101
- 0001101110
- 0001101110
- 0100110111

3.3 extended behaviour

Combining the states ‘adding’ and ‘shifting’ into a single state →‘adding_shifting’#

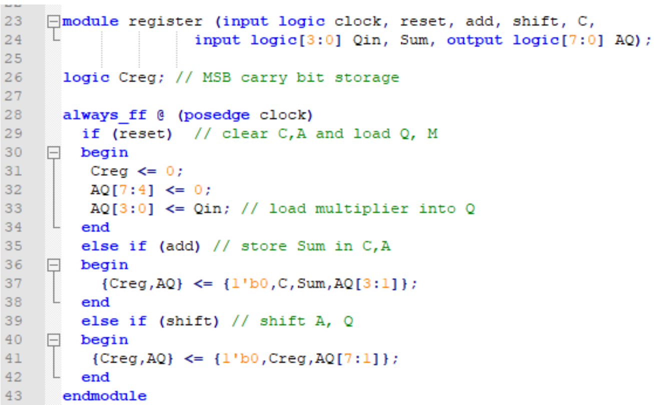
Modifying ‘sequencer.sv’ and ‘register.sv’

Sequencer.sv:



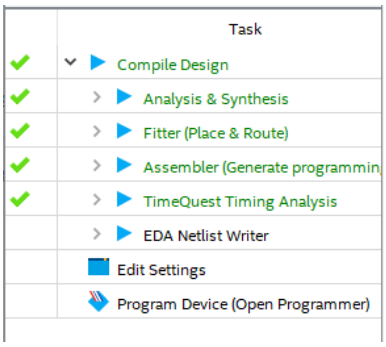
```
module sequencer (input logic start, clock, Q0, n_rst,
 output logic add, shift, ready, reset);

    enum {idle, adding_shifting, stopped} present_state, next_state;
    logic [0:1] count = 3;
    logic enable = '0;

    always_ff @(posedge clock, negedge n_rst)
        begin: SEQ
        if (!n_rst)
            present_state <= idle;
        else
            present_state <= next_state;
        end

    always_ff @(posedge clock, negedge n_rst)
        if (!n_rst)
            count <= 3;
        else if(enable)
            count <= count - 1;
        else if(count == 0)
            count <= 3;

    always_comb
        begin: COM
        add = '0;
        shift = '0;
        ready = '0;
        reset = '0;
        enable = '0;
        next_state = present_state;
        unique case (present_state)
            idle: begin
                reset = '1;
                if (start)
                    next_state = adding_shifting;
                end
            adding_shifting: begin
                enable = '1;
                if (Q0)
                    add = '1;
                shift = '1;
                if (count>0)
                    next_state = adding_shifting;
                else
                    next_state = stopped;
                end
            stopped: begin
                ready = '1;
                if(start)
                    next_state = idle;
                end
            endcase
    end
endmodule
```

Register.sv:



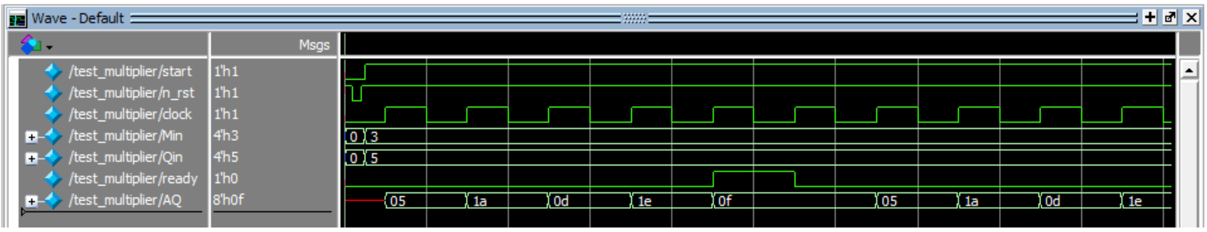
```
module register (input logic clock, reset, add, shift, C,
                 input logic[3:0] Qin, Sum, output logic[7:0] AQ);

logic Creg; // MSB carry bit storage

always_ff @ (posedge clock)
  if (reset)  // clear C,A and load Q, M
  begin
   Creg <= 0;
   AQ[7:4] <= 0;
   AQ[3:0] <= Qin; // load multiplier into Q
  end
  else if (add) // store Sum in C,A
  begin
    {Creg,AQ} <= {1'b0,C,Sum,AQ[3:1]};
  end
  else if (shift) // shift A, Q
  begin
   {Creg,AQ} <= {1'b0,Creg,AQ[7:1]};
  end
endmodule
```

Analysing and compiling code:



	Task
✓	Compile Design
✓	Analysis & Synthesis
✓	Fitter (Place & Route)
✓	Assembler (Generate programmin
✓	TimeQuest Timing Analysis
	EDA Netlist Writer
	Edit Settings
	Program Device (Open Programmer)

Modelsim waveform:



```
/test_multiplier/start   1'h1
/test_multiplier/n_rst   1'h1
/test_multiplier/clock   1'h1
/test_multiplier/Min     4'h3
/test_multiplier/Qin     4'h5
/test_multiplier/ready   1'h0
/test_multiplier/AQ      8'h0f
```

I successfully downloaded the design onto the FPGA as before and confirmed the behaviour by entering numbers and observing the LEDs on the board.