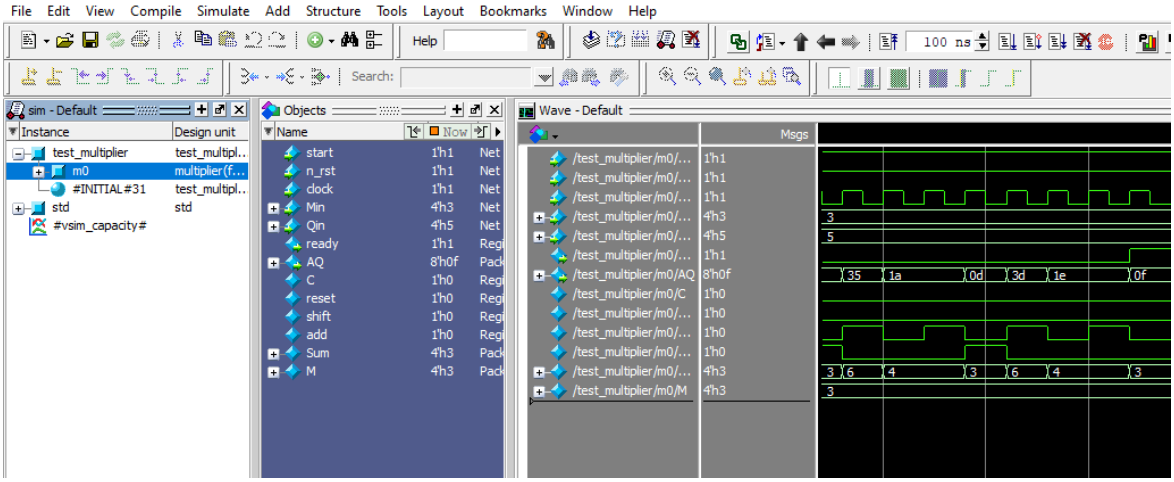
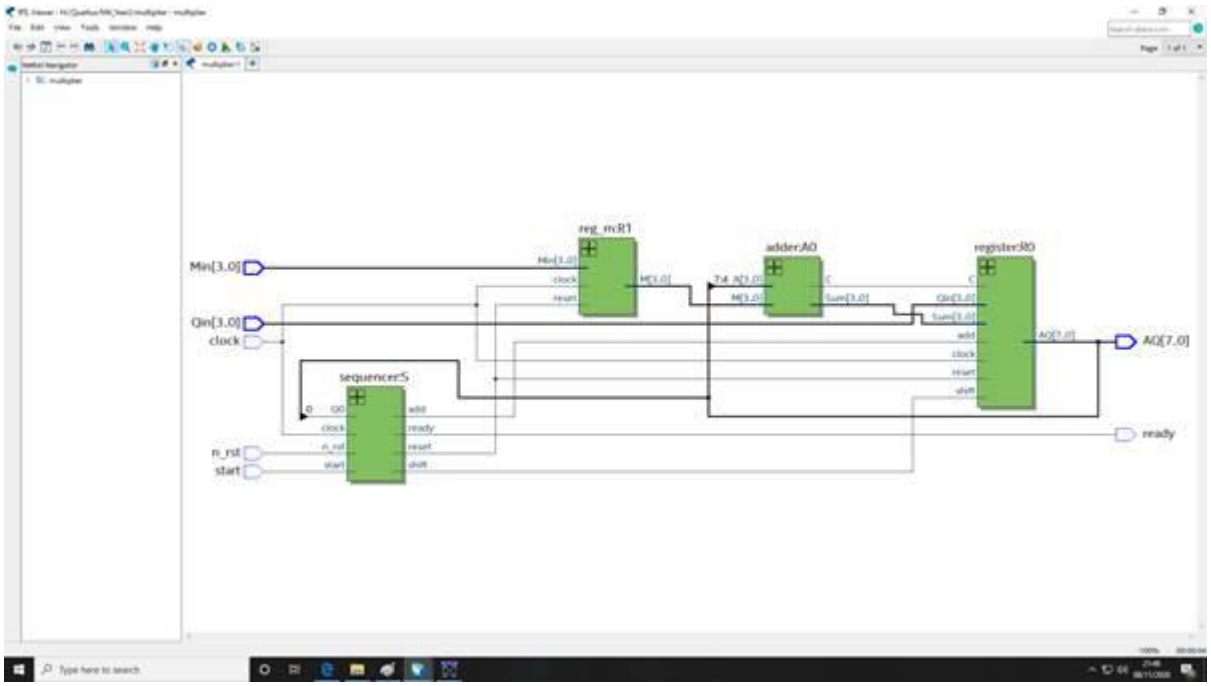
3.1

This waveform shows the correct behaviour, the clock cycle is working. It also shows ready going high after nine clock cycles and it shows the correct result of 15 in hex on the output on the AQ row.

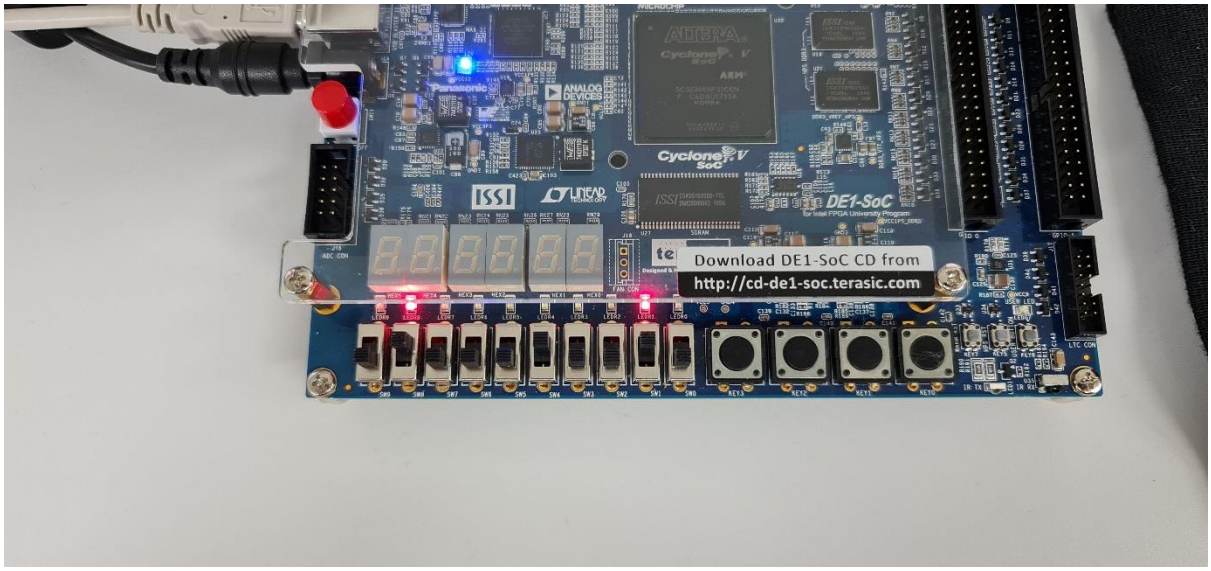


The schematic of the circuit generated by quartus.



3.2

It took 9 clock cycles to get from start to end and as you can see after doing 2x1 it provided the ready sign at the correct position



3.3

Within the always comb block I changed the shifting state combinational logic to the code below, I also added code to the idle to take care of the first iteration of the sequence.

```
idle:   begin
            reset = '1;
            if(start)
                    next_state = shifting;
                    cntN_state = three;       //added code
                    if(Q0)
                                    add='1;
            else
                    next_state = idle;
            end
```

```
shifting:begin
                    shift = '1;
                    unique case(cntP_state)
                            zero:     begin
                                      count = 0;
                                      cntN_state = four;
                                      next_state = stopped;
                                      end
                            one:      begin
                                      cntN_state = zero;
                                      next_state = shifting;
                                      if(Q0)
                                              add='1;
                                      end
                            two:      begin
                                      cntN_state = one;
                                      next_state = shifting;
                                      if(Q0)
                                              add='1;
                                      end
                            three:    begin
                                      cntN_state =two;
                                      next_state = shifting;
                                      if(Q0)
                                              add='1;
                                      end
                    endcase
                    end
```

I also changed the register so that the add output would complete the same action as shift as they would be completing the same action.

```
else if (add) // store Sum in C,A
begin
 Creg <= C;
 {Creg,AQ} <= {1'b0,C,Sum,AQ[3:1]};
end
```

This is a screenshot of it working via modelsim.

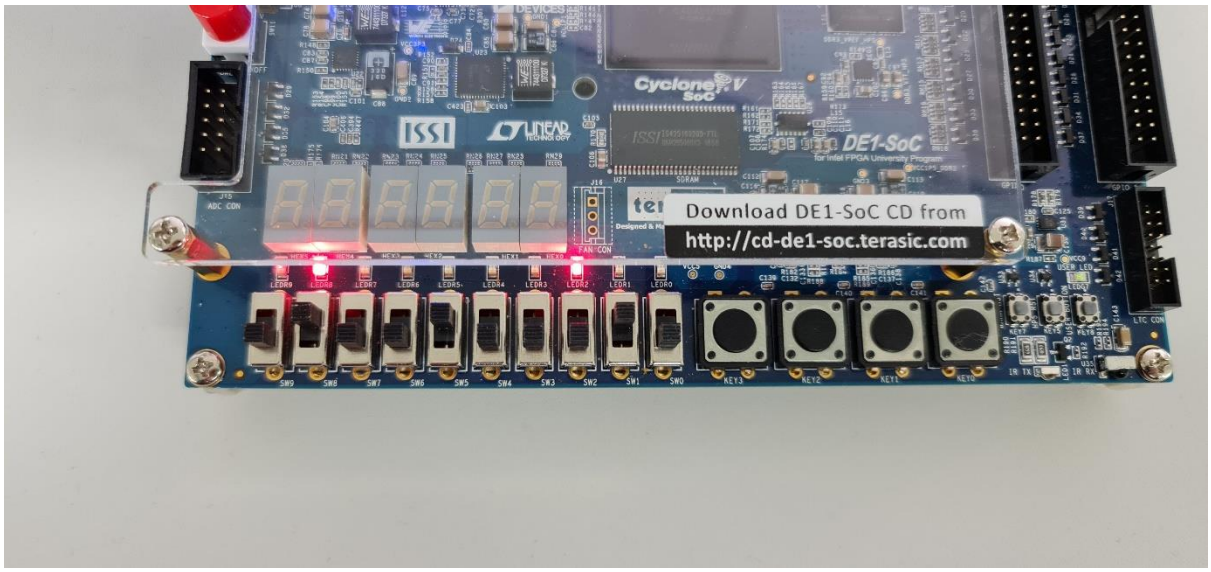
```
VSIM 146> restart -f
# ** Note: (vsim-8009) Loading existing optimized design _opt
# Loading work.test_multiplier(fast)
VSIM 147> run -all
# Test passed: at                  130 Min =  3, Qin =  5, AQ = 15
VSIM 148> quit -sim
# End time: 12:30:25 on Nov 09,2020, Elapsed time: 0:00:36
# Errors: 0, Warnings: 0
ModelSim> vsim M4_part3.test_multiplier
# vsim M4_part3.test_multiplier
# Start time: 12:30:26 on Nov 09,2020
# ** Note: (vsim-8009) Loading existing optimized design _opt
# Loading sv_std.std
# Loading work.test_multiplier(fast)
VSIM 150> run -all
# Test passed: at                  130 Min =  3, Qin =  5, AQ = 15

VSIM 151>
```

Quartus generated no errors and it loaded onto the FPGA and completed the task with no issues, it takes 5 clock cycles to complete the multiplication.

These are the only errors I received after a complete compilation.

```
Type  ID  Message
⚠ 18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate value for best performance.
⚠ 18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate value for best performance.
⚠ 15714 Some pins have incomplete I/O assignments. Refer to the I/O Assignment Warnings report for details
⚠ 16406 1 global input pin(s) will use non-dedicated clock routing
⚠ 18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate value for best performance.
⚠ 18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate value for best performance.
```

This is the picture of 3.3 working on the FPGA.



4.1

I used code from 3.1/3.2 as it was easier for me to imagine and implement. First, I altered the state machine so that it counted from 8 down to 0 and 0 is the off-state holder as N was increased.

```
always_comb
        begin: CNTCOM
        count = 1;
        cntN_state = cntP_state;
        unique case(cntP_state)
        zero:   begin
                count = 0;
                cntN_state = four;
                end
        one:    begin
                if(present_state == adding)
                        cntN_state = zero;
                end
        two:    begin
                if(present_state == adding)
                        cntN_state = one;
                end
        three:  begin
                if(present_state == adding)
                        cntN_state = two;
                end
        four:   begin
                if(present_state == adding)
                        cntN_state = three;
                end
        five:   begin
                if(present_state == adding)
                        cntN_state = four;
                end
        six:    begin
                if(present_state == adding)
                        cntN_state = five;
                end
        seven:  begin
                if(present_state == adding)
                        cntN_state = six;
                end
        eight:  begin
                if(present_state == adding)
                        cntN_state = seven;
                end
        endcase
        end
```

I then went through each file and increased the parameter size from 4 to 7 and adjusted the registers domain. I also adjusted to output logic to be 16 bits e.g.

This is the new register module.

```
module register (input logic clock, reset, add, shift, C,
          input logic[7:0] Qin, Sum, output logic[15:0] AQ);

logic Creg; // MSB carry bit storage

always_ff @ (posedge clock)
  if (reset)  // clear C,A and load Q, M
  begin
   Creg <= 0;
   AQ[15:8] <= 0;
   AQ[7:0] <= Qin; // load multiplier into Q
  end
  else if (add) // store Sum in C,A
  begin
   Creg <= C;
   {Creg,AQ} <= {1'b0,C,Sum,AQ[7:1]};
  end
  else if (shift) // shift A, Q
  begin
   {Creg,AQ} <= {1'b0,Creg,AQ[15:1]};
  end
endmodule
```

This is the new reg_m.

```
module reg_m #(parameter N=8) (input logic clock, reset,
          input logic[N-1:0] Min, output logic[N-1:0] M);

always_ff @ (posedge clock)
  if (reset)  // clear C,A and load Q, M
    M <= Min;

endmodule
```

This is the new adder.

```
module adder #(parameter N=8) (input logic[N-1:0] A,M, output logic C, output logic [N-1:0]
Sum);
 // N+1-bit arithmetic addition allows to extract carry
 always_comb
   {C,Sum} = {1'b0,A} + {1'b0,M};

endmodule
```

This is the new multiplier

```
module multiplier(input logic start, n_rst, clock, input logic [7:0] Min, Qin,
                  output logic ready, output logic[15:0] AQ);

logic C, reset, shift, add;
logic [7:0] Sum, M;


adder A0(.A(AQ[7:4]), .M(M), .C(C), .Sum(Sum));
register R0 (.*);
reg_m R1 (.*);
sequencer S(.start(start), .clock(clock), .reset(reset), .Q0(AQ[0]), .n_rst(n_rst),
 .add(add), .shift(shift), .ready(ready));

endmodule
```

Unfortunately I ran out of time however if given the time I would have edited the sequence to have a new state which was allocated to separating the inputs so that it could be used on the FPGA.