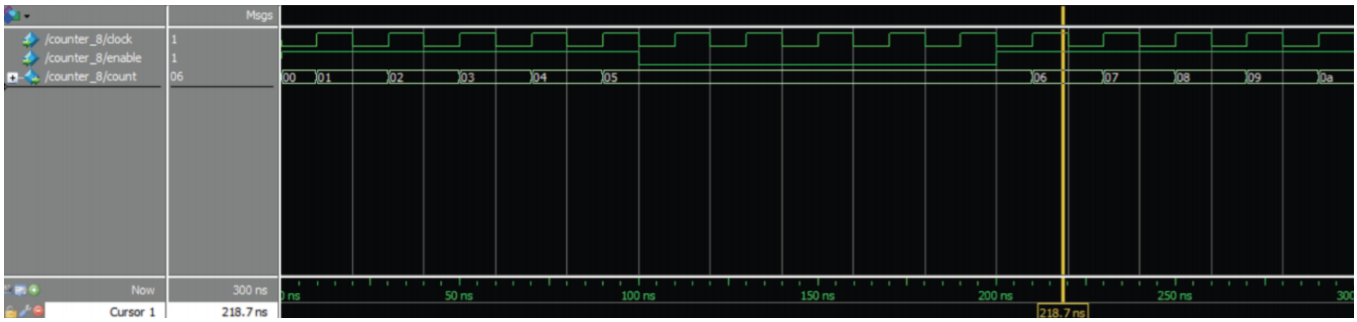
Verilog Experiment - Part 2

Experiment 5

```
1   `timescale 1ns / 100ps
2
3   module counter_8 (
4   clock,
5   enable,
6   count
7   );
8   parameter BIT_SZ=8;
9   input clock;
10  input enable;
11  output [BIT_SZ-1:0] count;
12
13  reg [BIT_SZ-1:0] count;
14
15  initial count=0;
16
17  always @ (posedge clock)
18          if (enable == 1'b1)
19          count <= count + 1'b1;
20
21  endmodule
```

In this experiment we made an 8 bit counter. The first line indicates the unit time is 1ns with a 100ps resolution. The output is registered as its value is changing and we are using blocking assignment inside an always block which is positively edge triggered with clock. Parameters are used to easily change the bit size and thus the count length.

Below is the timing diagram of the counter using modelsim. As can be seen when enable is low the counter holds its value until enable is high again. It is also counting the correct sequence and can count up to 128.

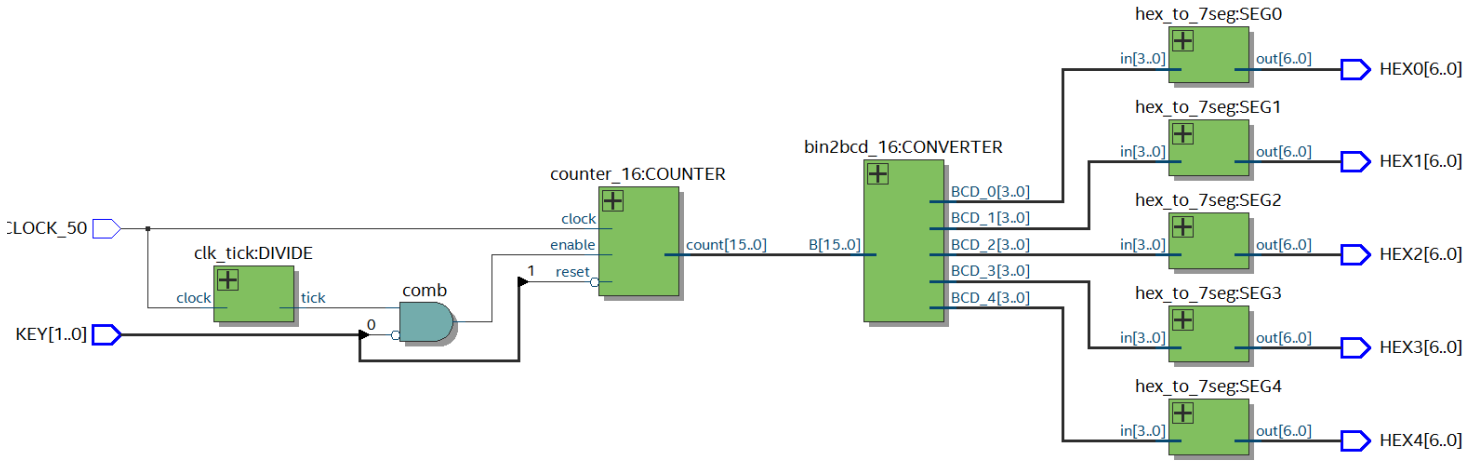



```
1   add wave clock enable
2   add wave -hexadecimal count
3   force clock 0 0, 1 10ns -repeat 20ns
4   force enable 1
5   run 100ns
6   force enable 0
7   run 100ns
8   force enable 1
9   run 1000ns
```

On the left is a do file which essentially allows us to test bench the counter without having to continuously enter commands and easily lets us change our designs. We can also single step through our Verilog code and see how it affects output.

Experiment 6

The 16 -bit converter worked fine and displayed the correct count sequence on the segment screens and reset when reset input was active low as specified. The max Frequency at 85C was 445.04MHz and at 0C was 422.48MHz. The TimeQuest entry was red due to unconstrained ports which are a type of unconstrained path, that are paths without any timing constraints specified to them. The report details the type of unconstrained paths: clocks, input ports, outputs ports. **"Altera recommends that all paths and ports be constrained to achieve optimal placement and fitting results."**



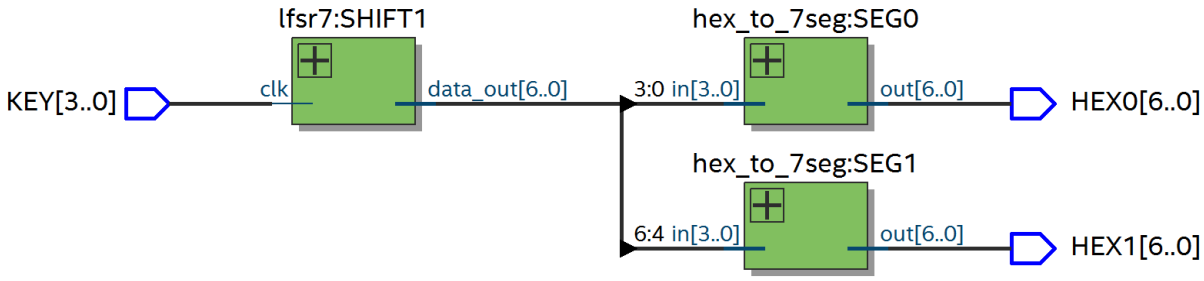
Above is the top level RTL block diagram for the cascaded 16-bit counter, which uses a slower clock of 1KHz so that we can see the counter changing on the display. To do this I created a module called clk_tick shown below, which produces an output every 50000 cycles resembling a clock that is x50000 slower than 50MHz and is fed into the enable input of the counter.

```
1   `timescale 1ns / 100ps
2
3   module clk_tick (
4   clock,
5   tick
6   );
7   parameter BIT_SZ=16;
8   input clock;
9   output tick;
10
11  reg [BIT_SZ-1:0] count;
12  reg tick;
13
14  initial count=0;
15  initial tick = 0;
16
17  always @ (posedge clock)
18          if (count == 50000)
19          begin
20                  tick = 1;
21                  count = 0;
22          end
23          else if (count != 50000)
24          begin
25                  count = count + 1'b1;
26                  tick = 0;
27          end
28
29  endmodule
```

Experiment 7

```
1   module lfsr7 (data_out, clk);
2
3           output [6:0]    data_out;
4           input           clk;
5
6           reg[6:0]        sreg;
7
8           initial sreg = 6'b1;
9
10          always @ (posedge clk)
11                  sreg <= {sreg[6:0], sreg[5] ^ sreg[0]};
12
13          assign data_out = sreg;
14
15  endmodule
```

The LFSR outputs a PRBS sequence which is random. In this case it is 7 bit, so the max length sequence is a 127 cycles. In implements the primitive polynomial 1 + x + x^7. KEY[3] is used as the clock to cycle through the values. x and x^7 are tapped off at registers 1 and 7 and then used as inputs to an XOR gate.

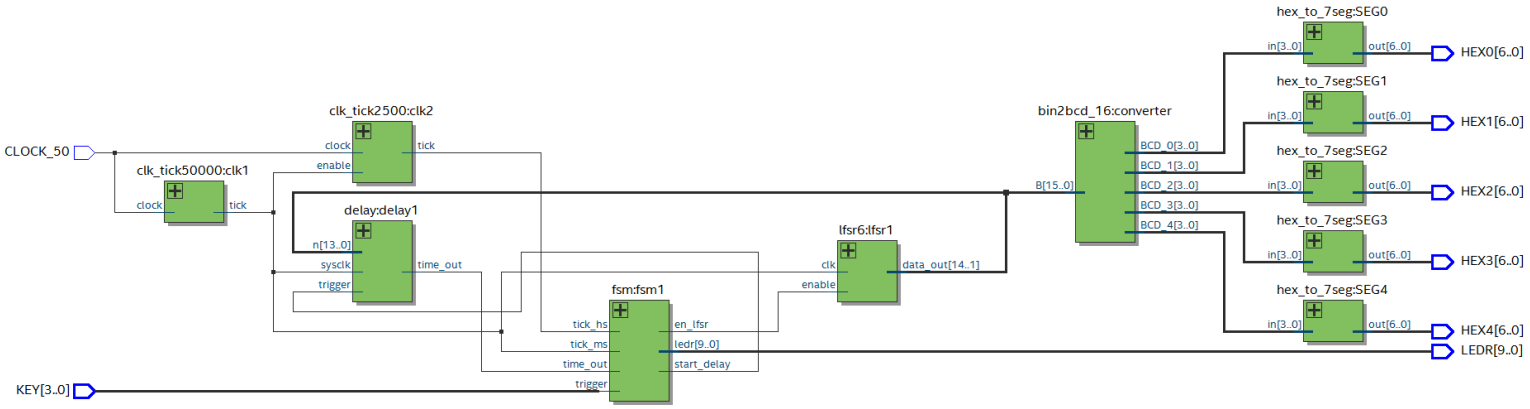


Above is the block diagram for the top – level. As it can be seen that at each clock the lfsr gives one output to 2 segs.

Experiment 8 & 9 (Optional)

The circuit below for experiment 8 has the following behaviour:

1. The circuit is triggered (or started) by pressing KEY[3] (don't forget KEY[3] is low active);
2. The 10 LEDs (below the 7-segment displays) will then start lighting up from left to right at 0.5 second interval, until all LEDs are ON;
3. The circuit then waits for a random period of time between 0.25 and 16 seconds before all LEDs turn OFF;
4. You should also display the random delay period in milliseconds on five 7-segment displays.



Furthermore, more specifically:

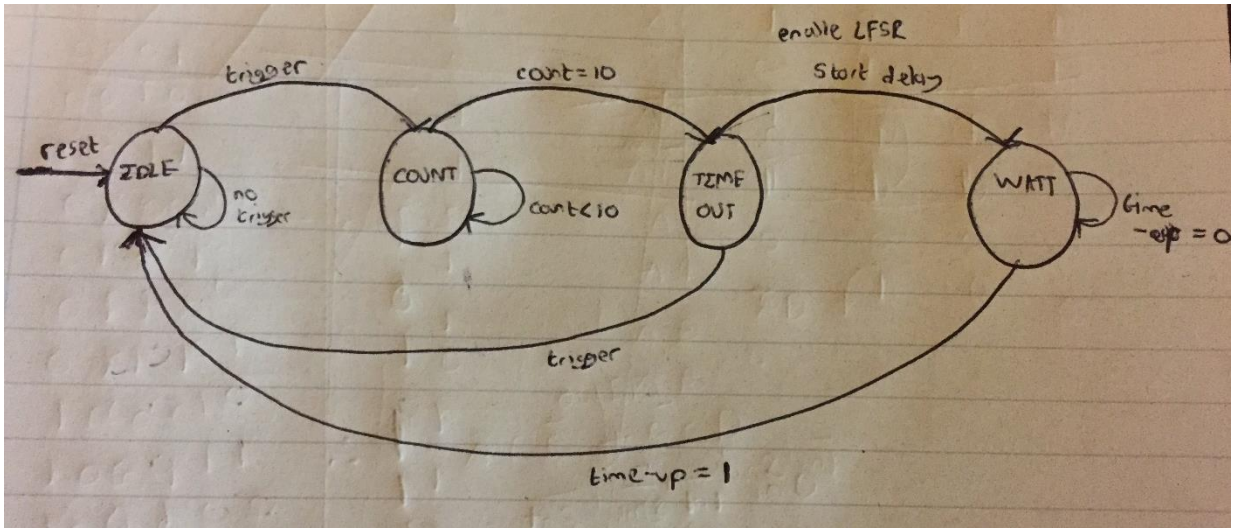
The two clock divider circuits provide clock ticks once every 1ms and 0.5sec respectively. Each clock tick should be a positive pulse lasting one period of **CLOCK_50** (i.e. 20ns). The system then use the **tick_ms** signal as the clock of the remaining circuit.

The **LFSR** module produces a pseudo-random binary sequence (**PRBS**), which is used to determine the random delay required. The **enable** signal to the **LFSR** allows this to cycle through a number of clock cycles before it is stopped at a random value.

The **delay** module is triggered after all 10 LEDs are lid, and then provides a delay of **N** clock cycles (at 1ms period) before asserting the **time_out** signal (for 1ms).

The delay value N is fed to the binary to BCD converter, which then drives the 7-segment displays.

In order for the circuit to work correctly the LFSR must be 6 bits * 250 so truncate to using 14 bits and the value of N in the delay module must also be 14 bits as it counts in milliseconds, so max delay is 16s which is required. Below is the FSM which has been simplified slightly in its representation than from the Verilog code that it describes.



Below is the RTL top-level block diagram from experiment 9. It counts the time from when the LEDs go off and when you press down KEY[0] and then displays your reaction time on the seg displays. All I had to change was to add another output to the FSM which goes high when the count is finished and use that as an enable to a counter that holds its value and outputs it when KEY[0] is held down.