Mark Davis, 5590-3552, 10/16/2018

I implemented the datapath with structural vhdl as shown. Files used here are included in the full lab zip and an exhaustive list is as follows: cmp, genmux, regis, subtr along with the decoder7seg that is called from the top_level.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity datapath1 is
  generic (
    WIDTH : natural := 16);
  port (
          x      : in  std_logic_vector(WIDTH-1 downto 0);
    y      : in  std_logic_vector(WIDTH-1 downto 0);
          clk      : in  std_logic;
          x_sel      : in  std_logic := '0';
          y_sel     : in  std_logic := '0';
          x_en      : in  std_logic := '0';
          y_en      : in  std_logic := '0';
          output_en      : in  std_logic := '0';
          x_lt_y      : inout  std_logic;
          x_ne_y      : out  std_logic;
    output  : out std_logic_vector(WIDTH-1 downto 0));
end entity;


architecture FSM_D1 of datapath1 is

COMPONENT genmux
                generic (
        WIDTH : natural := WIDTH
        );
                port(A, B : in std_logic_vector(WIDTH-1 downto 0);
                        S : in std_logic;
                        Y : out std_logic_vector(WIDTH-1 downto 0));
END COMPONENT ;
COMPONENT cmp
                generic (
        WIDTH : natural := WIDTH
        );
                port(X, Y : in std_logic_vector(WIDTH-1 downto 0);
                        x_lt_y, x_ne_y : out std_logic);
END COMPONENT ;
```

```
COMPONENT regis
		generic (
	WIDTH : natural := WIDTH
	);
		port(i : in std_logic_vector(WIDTH-1 downto 0);
			o : out std_logic_vector(WIDTH-1 downto 0);
			en, clk : in std_logic);
END COMPONENT ;
COMPONENT subtr
		generic (
	WIDTH :natural := WIDTH
	);
		port(A, B : in std_logic_vector(WIDTH-1 downto 0);
			output : out std_logic_vector(WIDTH-1 downto 0));
END COMPONENT ;
signal muxedX : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
signal muxedY : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
signal regX : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
signal regY : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
signal subX : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
signal subY : std_logic_vector(WIDTH-1 downto 0) := (others => '0');

begin  -- FSM_D1
xmux : genmux generic map (WIDTH => WIDTH)
	port map (x, subX, x_sel, muxedX);
ymux : genmux generic map (WIDTH => WIDTH)
	port map (A => y, B=> subY, S=> y_sel, Y=> muxedY);
xreg : regis generic map (WIDTH => WIDTH)
	port map (i => muxedX, en  => x_en,o => regX, clk=> clk);
yreg : regis generic map (WIDTH => WIDTH)
	port map (i => muxedY, en  => y_en,o => regY, clk=> clk);
cmprtr : cmp generic map (WIDTH => WIDTH)
	port map (X => regX, Y => regY, x_lt_y => x_lt_y, x_ne_y  => x_ne_y);
xsub : subtr generic map (WIDTH => WIDTH)
	port map (A => regX, B => regY, output  => subX);
ysub : subtr generic map (WIDTH => WIDTH)
	port map (A => regY, B => regX, output  => subY);
outreg : regis generic map (WIDTH => WIDTH)
	port map (i => regX, o => output, en  => output_en, clk=> clk);
end FSM_D1;
```

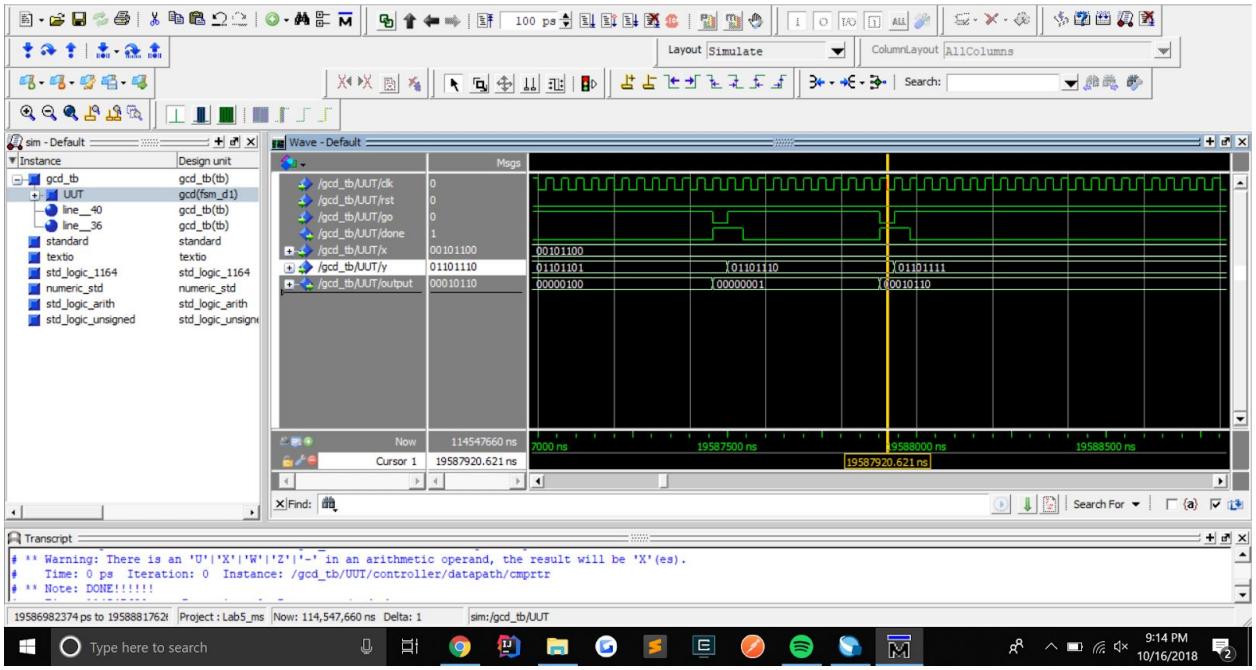
I designed a controller that instantiates this original datapath architecture and implements a finite state machine as follows.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ctrl1 is
  generic (
    WIDTH : positive := 16);
  port (
            clk    : in  std_logic;
    rst    : in  std_logic;
    go     : in  std_logic;
    done   : out std_logic;
    x      : in  std_logic_vector(WIDTH-1 downto 0);
    y      : in  std_logic_vector(WIDTH-1 downto 0);
    output : out std_logic_vector(WIDTH-1 downto 0));
end entity;


architecture FSM of ctrl1 is

COMPONENT datapath1
                    generic (
    WIDTH : positive := 16);
  port (
            x      : in  std_logic_vector(WIDTH-1 downto 0);
    y      : in  std_logic_vector(WIDTH-1 downto 0);
    clk    : in  std_logic;
            x_sel    : in  std_logic;
            y_sel    : in  std_logic;
    x_en    : in  std_logic;
            y_en    : in  std_logic;
            output_en    : in  std_logic;
            x_lt_y    : inout  std_logic;
            x_ne_y    : out  std_logic;
    output : out std_logic_vector(WIDTH-1 downto 0));
END COMPONENT ;
--control signals for the datapath
signal x_sel, y_sel, x_en, y_en, output_en, x_lt_y, x_ne_y : std_logic;
TYPE State_type IS (inld, calc, ldsubx, ldsuby, ldoutput, outs);  -- Define the states
          SIGNAL state : State_Type;    -- Create a signal that uses
```

```
begin

-- Change next line to use the other datapath
datapath : entity work.datapath1(FSM_D1)
        generic map (
           WIDTH => WIDTH) -- 50 MHZ to 1Khz
        port map (
           x => x,
                                        y => y,
                                        clk => clk,
                                        x_sel => x_sel,
                                        y_sel => y_sel,
                                        x_en => x_en,
                                        y_en => y_en,
                                        output_en => output_en,
                                        x_lt_y => x_lt_y,
                                        x_ne_y => x_ne_y,
                                        output => output);

process(clk)
begin
if (rst = '1') then            -- Upon reset
           state <= inld;
elsif(clk'event and clk = '1' and go = '1') then
case state is
                    WHEN inld =>
                            IF go='1' THEN
                                    state <= calc;
                            END IF;

                    WHEN ldsubx =>
                            IF go='1' THEN
                                    state <= calc;
                            END IF;

                    WHEN ldsuby =>
                            IF go='1' THEN
                                    state <= calc;
                            END IF;

                    WHEN calc=>
                            IF go='1' THEN
```

```
                        IF x_lt_y = '1' THEN
                                state <= ldsuby;
                        ELSIF x_ne_y = '1' THEN
                                state <= ldsubx;
                        ELSE
                                state <= ldoutput;
                        END IF;
                END IF;

        WHEN ldoutput =>
                IF go='1' THEN
                        state <= outs;
                END IF;

        WHEN outs =>
                IF go='1' THEN
                        state <= inld;
                END IF;

        WHEN others =>
                state <= inld;
    END CASE;
end if;
end process;
x_en <= '1' WHEN (state=ldsubx or state=inld) ELSE '0';
y_en <= '1' WHEN (state=ldsuby or state=inld) ELSE '0';
y_sel <= '1' WHEN (state=ldsuby or (state=calc and x_lt_y = '1')) ELSE '0';
x_sel <= '1' WHEN (state=ldsubx or (state=calc and x_lt_y = '0' and x_ne_y = '1')) ELSE '0';
output_en <= '1' WHEN (state=ldoutput) ELSE '0';
done <= '1' WHEN state=outs ELSE '0';
end FSM;
```

This is the result with x=44 and y=110 result: 22.



Note: Some warnings ('X' in an arithmetic operand) show up in the console, but these are only from before a "go" is sent and inputs are given by the testbench.

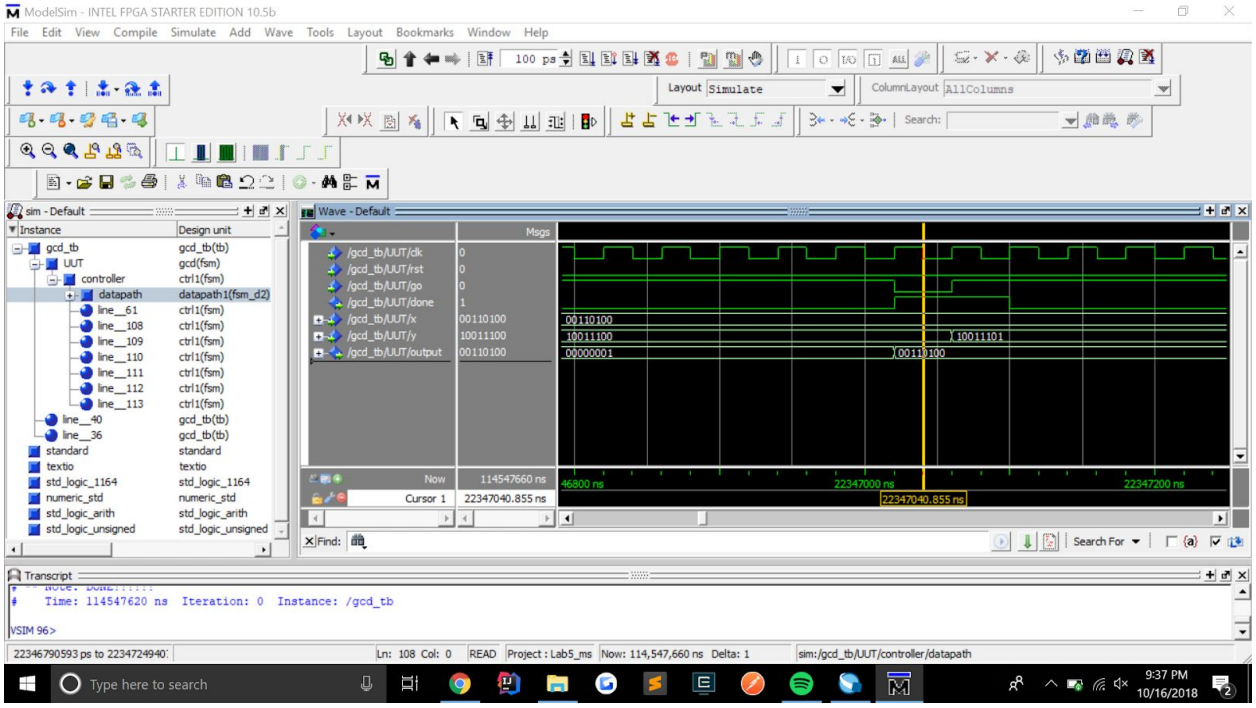
For the second part of the lab, I added an additional architecture to the datapath file (fsm_d2) as follows and changed the controller to instantiate this architecture of the datapath instead.

```
architecture FSM_D2 of datapath1 is
COMPONENT genmux
                generic (
        WIDTH : natural := WIDTH
        );
                port(A, B : in std_logic_vector(WIDTH-1 downto 0);
                        S : in std_logic;
                        Y : out std_logic_vector(WIDTH-1 downto 0));
END COMPONENT ;
COMPONENT cmp
                generic (
        WIDTH : natural := WIDTH
        );
                port(X, Y : in std_logic_vector(WIDTH-1 downto 0);
                        x_lt_y, x_ne_y : out std_logic);
END COMPONENT ;
COMPONENT regis
```

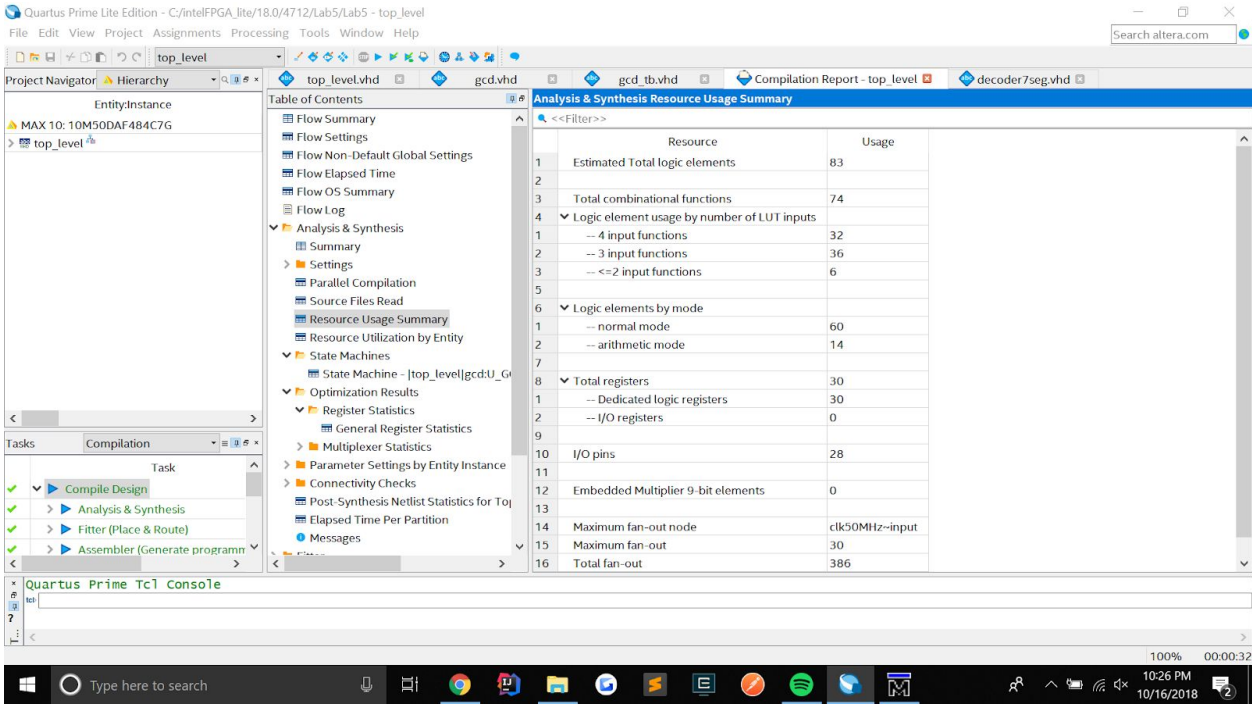
```
		generic (
	WIDTH : natural := WIDTH
	);
		port(i : in std_logic_vector(WIDTH-1 downto 0);
			o : out std_logic_vector(WIDTH-1 downto 0);
			en, clk : in std_logic);
END COMPONENT ;
COMPONENT subtr
		generic (
	WIDTH :natural := WIDTH
	);
		port(A, B : in std_logic_vector(WIDTH-1 downto 0);
			output : out std_logic_vector(WIDTH-1 downto 0));
END COMPONENT ;
signal muxedX : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
signal muxedY : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
signal regX : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
signal regY : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
signal dynamicGreater : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
signal dynamicLess : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
signal subOnly : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
begin  -- FSM_D2
xmux : genmux generic map (WIDTH => WIDTH)
	port map (x, subOnly, x_sel, muxedX);
ymux : genmux generic map (WIDTH => WIDTH)
	port map (A => y, B=> subOnly, S=> y_sel, Y=> muxedY);
xreg : regis generic map (WIDTH => WIDTH)
	port map (i => muxedX, en  => x_en,o => regX, clk=> clk);
yreg : regis generic map (WIDTH => WIDTH)
	port map (i => muxedY, en  => y_en,o => regY, clk=> clk);
cmprtr : cmp generic map (WIDTH => WIDTH)
	port map (X => regX, Y => regY, x_lt_y => x_lt_y, x_ne_y  => x_ne_y);

dynamicGreater <= regY when x_lt_y = '1' else regX;
dynamicLess <= regX when x_lt_y = '1' else regY;

sub : subtr generic map (WIDTH => WIDTH)
	port map (A => dynamicGreater, B => dynamicLess, output  => subOnly);

--We have to dynamically flip the sub inputs depending on x_lt_y

--ysub : subtr generic map (WIDTH => WIDTH)
--	port map (A => regY, B => regX, output  => subY);
```

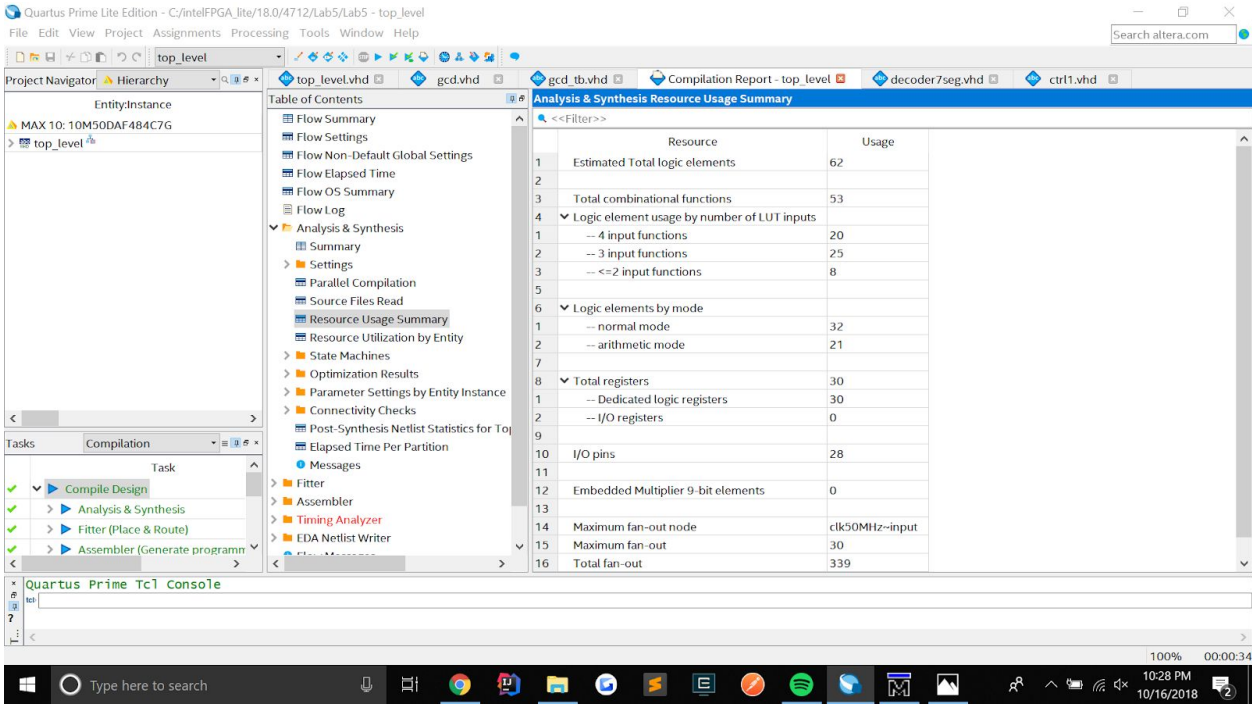
```
outreg : regis generic map (WIDTH => WIDTH)
        port map (i => regX, o => output, en  => output_en, clk=> clk);

end FSM_D2;
```

The change can be observed in the left “instance” column of the screengrab.
This highlighted example is the operation gcd(52, 156) = 52



The difference in total logic elements between the two datapaths is shown below





I wrote a top_level file as follows and assigned the pins by import to avoid retyping the 7Seg LED pins.

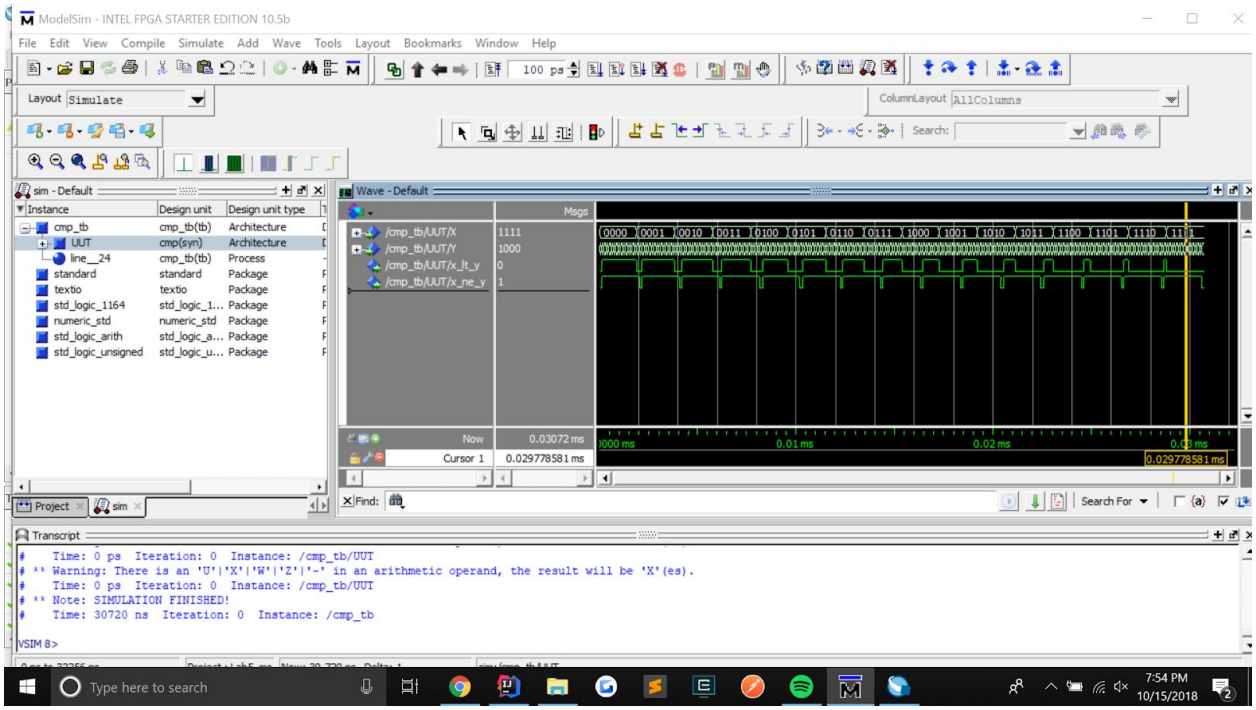
```
-- I/O Explanation (assumes the switches are on side of the
--                  board that is closest to you)
-- switch(9) is the leftmost switch
```

```
-- button(1) is the top button
-- led5 is the leftmost 7-segment LED
-- ledx_dp is the decimal point on the 7-segment LED for LED x

-- Note: this code will cause a harmless synthesis warning because not all
-- the buttons are used and because some output pins are always '0' or '1'

library ieee;
use ieee.std_logic_1164.all;

entity top_level is
   port (
      clk50MHz : in  std_logic;
      switch   : in  std_logic_vector(9 downto 0);
      button   : in  std_logic_vector(1 downto 0);
      led0     : out std_logic_vector(0 to 6);
                  led0_dp     : out std_logic;
      led1     : out std_logic_vector(0 to 6)
      );
end top_level;

architecture STR of top_level is
   component decoder7seg
      port (
         input  : in  std_logic_vector(3 downto 0);
         output : out std_logic_vector(0 to 6));
   end component;

          COMPONENT gcd generic (WIDTH : natural := 16);
          port (
   clk    : in  std_logic;
   rst    : in  std_logic;
   go     : in  std_logic;
   done   : out std_logic;
   x      : in  std_logic_vector(WIDTH-1 downto 0);
   y      : in  std_logic_vector(WIDTH-1 downto 0);
   output : out std_logic_vector(WIDTH-1 downto 0));
         end COMPONENT;

   signal gcdOut    : std_logic_vector(7 downto 0);
          signal catx    : std_logic_vector(7 downto 0);
          signal caty    : std_logic_vector(7 downto 0);
```

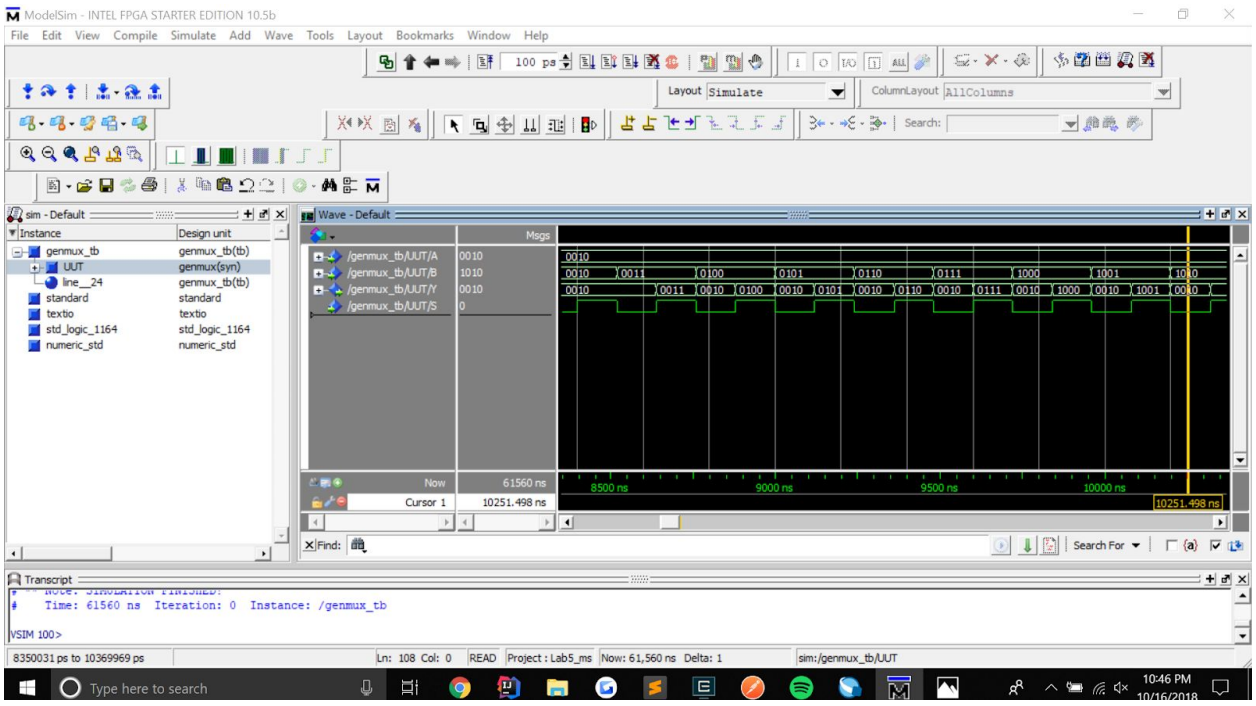
```
begin  -- STR

catx <= "000" & switch(9 downto 5);
caty <= "000" & switch(4 downto 0);

   U_GCD : gcd generic map(WIDTH => 8)
           port map (
      clk    => clk50MHz,
      rst    => not button(0),
                go    => not button(1),
                done => led0_dp,
                x    => catx,
                y    => caty,
      output => gcdOut);

   U_LED1 : decoder7seg port map (
      input  => (gcdOut(7 downto 4)),
      output => led1(0 to 6));

   U_LED0 : decoder7seg port map (
      input  => (gcdOut(3 downto 0)),
      output => led0(0 to 6));
end STR;
```

Some temporary signals were used to concatenate into the portmap.

Other testbenches I wrote are included in the report as well for unit testing/debugging. cmp


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
sim - Default
Instance
Design unit
Design unit type
cmp_tb
cmp_tb(tb)
Architecture
UUT
cmp(syn)
Architecture
line__24
cmp_tb(tb)
Process
standard
Package
textio
Package
std_logic_1164
Package
numeric_std
Package
std_logic_arith
Package
std_logic_unsigned
Package
Wave - Default
Msgs
/cmp_tb/UUT/X
/cmp_tb/UUT/Y
/cmp_tb/UUT/x_lt_y
/cmp_tb/UUT/x_ne_y
Now
0.03072 ms
Cursor 1
0.029778581 ms
Transcript
Time: 0 ps Iteration: 0 Instance: /cmp_tb/UUT
** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
Time: 0 ps Iteration: 0 Instance: /cmp_tb/UUT
** Note: SIMULATION FINISHED!
Time: 30720 ns Iteration: 0 Instance: /cmp_tb
VSIM 8>
Type here to search
7:54 PM
10/15/2018

genmux


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
sim - Default
Instance
Design unit
genmux_tb
genmux_tb(tb)
UUT
genmux(syn)
line__24
genmux_tb(tb)
standard
textio
std_logic_1164
numeric_std
Wave - Default
Msgs
/genmux_tb/UUT/A
/genmux_tb/UUT/B
/genmux_tb/UUT/Y
/genmux_tb/UUT/S
Now
61560 ns
Cursor 1
10251.498 ns
Transcript
Time: 61560 ns Iteration: 0 Instance: /genmux_tb
VSIM 100>
8350031 ps to 10369969 ps
Ln: 108 Col: 0
READ
Project : Lab5_ms
Now: 61,560 ns Delta: 1
sim:/genmux_tb/UUT
Type here to search
10:46 PM
10/16/2018