Report for GCD
Varunesh Goyal 140070006

The vhdl codes used are as follows:

System.vhd :

```
library work;
use work.thepackage.all;

entity System is
       port ( din: in bit_vector(15 downto 0);
       dout: out bit_vector(15 downto 0);
       start: in bit;
       done: out bit;
       erdy: in bit;
       srdy: out bit;
       clk: in bit;
       reset: in bit);
end entity;

architecture behave of System is
       signal T0, T1, T2, T3, T4, T5 : bit;
       signal divDone, compareDone, count_var : bit;
begin
       control : GCD_controlpath port map (T0 => T0,T1 => T1,T2 => T2,T3 => T3,T4 => T4,T5 => T5,
                             divDone => divDone, compareDone => compareDone, count_var =>count_var,
                             start => start, srdy => srdy, erdy => erdy, done => done, reset =>
                             reset, clk => clk);
       data : GCD_datapath port map (T0 => T0,T1 => T1,T2 => T2,T3 => T3,T4 => T4,T5 => T5,
                             divDone => divDone, compareDone => compareDone, count_var =>count_var,
                             input => din, GCD => dout, clk => clk, reset => reset);

end behave;
```

GCD_controlpath.vhd :

```
library work;
use work.thepackage.all;

entity GCD_controlpath is
       port (
              T0,T1,T2,T3,T4,T5: out bit;
              divDone, compareDone, count_var: in bit;
              start: in bit;
              srdy : out bit;
              erdy : in bit;
              done : out bit;
              clk, reset: in bit
            );
end entity GCD_controlpath;

architecture Behave of GCD_controlpath is
       type FsmState is ( phi, input1, input2, doDivide, checkRem, donestate);
       signal fsm_state : FsmState;
begin

   process(fsm_state, start, clk, reset)
      variable next_state: FsmState;
      variable Tvar: bit_vector(0 to 5);
      variable done_var: bit;
      variable srdy_var : bit;
   begin
       -- defaults
       Tvar := (others => '0');
       done_var := '0';
       next_state := fsm_state;
       srdy_var := '0';

       case fsm_state is
          when phi =>
               if(start = '1') then
                  next_state := input1;
                  Tvar(0) := '1';
                 srdy_var := '1';
               end if;
          when input1 =>
               if(erdy = '1') then
                       Tvar(1) := '1';
                       next_state := input2;
                       srdy_var := '1';
               else
                       srdy_var := '1';
                       next_state := input1;
               end if;
         when input2 =>
               if (count_var = '1') then
                       next_state := donestate;
               elsif (erdy = '1') then
                       Tvar(2) := '1';
                       next_state := doDivide;
                       --srdy_var := '1';
               else
                       srdy_var := '1';
                       next_state := input2;
               end if;
         when doDivide =>
               Tvar(3) := '1';                    --start div command
               if (divDone = '1') then
                       next_state := checkRem;
               else
                       next_state := doDivide;
               end if;
         when checkRem =>
               if (compareDone = '1') then
                       next_state := input2;
                       srdy_var := '1';
                       Tvar(5) := '1';
               else
                       Tvar(4) := '1';
                       next_state := doDivide;
               end if;
          when donestate =>
               done_var := '1';
               next_state := phi;
     end case;

     T0 <= Tvar(0); T1 <= Tvar(1); T2 <= Tvar(2); T3 <= Tvar(3); T4 <= Tvar(4); T5 <= Tvar(5);
     done <= done_var;
     srdy <= srdy_var;

     if(clk'event and (clk = '1')) then
       if(reset = '1') then
             fsm_state <= phi;
        else
             fsm_state <= next_state;
        end if;
     end if;
   end process;
end Behave;
```

GCD_datapath.vhd :

```
library work;
use work.thepackage.all;

entity GCD_datapath is
       port (
               T0,T1,T2,T3,T4,T5: in bit;
               divDone, compareDone, count_var : out bit;
               input : in bit_vector(15 downto 0);
               GCD: out bit_vector(15 downto 0);
               clk, reset: in bit
            );
end entity GCD_datapath;

architecture Mixed of GCD_datapath is
    signal DDREG, DRREG: bit_vector(15 downto 0) := "0000000000000000";
    signal CREG: bit_vector(7 downto 0) := "10000000";
    signal RREG, QREG: bit_vector(15 downto 0);
       constant C16 : bit_vector(15 downto 0) := (others => '0');
       constant C7 : bit_vector(7 downto 0) := "00000000";
```

```
        signal DDREG_in, DRREG_in: bit_vector(15 downto 0);
        signal CREG_in : bit_vector(7 downto 0);
        signal rShiftCount : bit_vector(7 downto 0);

        signal compare : bit;
        signal random : bit;
        signal x1, x2, x3 : bit;

        begin

        --writing down the inputs to the controlpath
        count_var <= CREG(0);
        compareDone <= compare;

        --writing down the required registers
        dr1 : dataregister generic map (data_width => 16)
                              port map (Din => DDREG_in, Dout => DDREG, enable => x1, clk => clk);
        x1 <= (T1 or T5 or T4);
        dr2 : dataregister generic map (data_width => 16)
                              port map (Din => DRREG_in, Dout => DRREG, enable => x2, clk => clk);
        x2 <= (T2 or T4);

        dr5 : dataregister generic map (data_width => 8)
                              port map (Din => CREG_in, Dout => CREG, enable => x3, clk => clk);
        x3 <= (T0 or T5);

        --Now writing the logical calculators and calculations
        divider : unsigned_divider port map(clk => clk,
                                             reset => reset,
                                             dividend => DDREG,
                                             divisor  => DRREG,
                                             inputs_ready => T3,
                                             divider_ready => random,
                                             quotient => QREG,
                                             remainder => RREG,
                                             output_ready => divDone,
                                             output_accept => T4);
        comparator : comparer port map(a => C16, b => RREG, c => compare); --returns 1 when RREG = 0
        rShiftCount <= '1' & CREG(7 downto 1);

        --Now writing down all the transfers
        DDREG_in <= input when (T1 = '1') else DRREG when ((T5 or T4) = '1') else  DDREG;
        DRREG_in <= input when (T2 = '1') else RREG when (T4 = '1') else DRREG;
        CREG_in <= C7 when (T0 = '1') else rShiftCount when (T5 = '1') else CREG;
        GCD <= DDREG;

end Mixed;
```

Testbench_GCD.vhd :

```
library std;
use std.textio.all;
library work;
use work.thepackage.all;
entity Testbench_GCD is
end entity;
architecture Behave of Testbench_GCD is
  signal din, dout: bit_vector(15 downto 0);
  signal start, done, srdy, erdy: bit;
  signal clk: bit := '0';
  signal reset: bit := '1';

  component System is
             port ( din: in bit_vector(15 downto 0);
                    dout: out bit_vector(15 downto 0);
                    start: in bit;
                    done: out bit;
                    erdy: in bit;
                    srdy: out bit;
                    clk: in bit;
                    reset: in bit);
  end component System;

  function to_string(x: string) return string is
      variable ret_val: string(1 to x'length);
```

```
    alias lx : string (1 to x'length) is x;
begin
    ret_val := lx;
    return(ret_val);
end to_string;

begin
 clk <= not clk after 5 ns; -- assume 10ns clock.

 process
 begin
   wait until clk = '1';
   reset <= '0';
   wait;
 end process;

 process
   variable err_flag : boolean := false;
   File INFILE: text open read_mode is "TRACEFILE_GCD.txt";
   FILE OUTFILE: text  open write_mode is "OUTPUTS_GCD.txt";
   variable active: bit_vector (15 downto 0);
   variable op: bit_vector (15 downto 0);
   variable INPUT_LINE: Line;
   variable OUTPUT_LINE: Line;
   variable LINE_COUNT: integer := 0;

 begin
   wait until clk = '1';

   while not endfile(INFILE) loop
        wait until clk = '0';

       LINE_COUNT := LINE_COUNT + 1;
            readLine (INFILE, INPUT_LINE);
       start <= '1';
       for I in 1 to 8 loop
         read(INPUT_LINE, active);
         din <= active;
         erdy <= '1';

        --  wait till srdy becomes 1
         while (true) loop
           wait until clk = '1';
           if (srdy = '1') then
             exit;
           end if;
         end loop;
       end loop;

       read(INPUT_LINE, op);
       -- spin waiting for done
       while (true) loop
          wait until clk = '1';
          if(done = '1') then
             exit;
          end if;
       end loop;

        if (dout /= op) then
          write(OUTPUT_LINE,to_string("ERROR: in RESULT, line "));
          write(OUTPUT_LINE, LINE_COUNT);
          writeline(OUTFILE, OUTPUT_LINE);
          err_flag := true;
      else
              write(OUTPUT_LINE,to_string("CORRECT: in RESULT, line "));
          write(OUTPUT_LINE, LINE_COUNT);
          writeline(OUTFILE, OUTPUT_LINE);
       end if;
   end loop;

   assert (err_flag) report "SUCCESS, all tests passed." severity note;
   assert (not err_flag) report "FAILURE, some tests failed." severity error;
   wait;
 end process;
```

```
  dut: System
    port map (
       Din => din,
       Dout => dout,
       start => start,
       done => done,
       erdy => erdy,
       srdy => srdy,
       clk => clk, reset => reset);
end Behave;
```

thepackage.vhd :

```
package thepackage is
       component comparer is
              port(a, b : in bit_vector(15 downto 0);
                     c : out bit);
       end component comparer;

       component controlpath is
              port (
                     T0,T1,T2,T3,T4,T5: out bit;
                     divShiftDone, compareDone, rightShiftDone: in bit;
                     start: in bit;
                     done : out bit;
                     clk, reset: in bit
                   );
       end component controlpath;

       component datapath is
              port (
                     T0,T1,T2,T3,T4,T5: in bit;
                     divShiftDone, compareDone, rightShiftDone : out bit;
                     Dividend , Divisor : in bit_vector(15 downto 0);
                     Quotient, Remainder: out bit_vector(15 downto 0);
                     clk, reset: in bit
                   );
       end component datapath;

       component dataregister is
              generic (data_width:integer);
              port (Din: in bit_vector(data_width-1 downto 0);
                    Dout: out bit_vector(data_width-1 downto 0);
                    clk, enable: in bit);
       end component dataregister;

       component bitAdder is
              port(x, y, cin: in bit;
                     cout, s: out bit);
       end component bitAdder;

       component sixteenBitSubtract is
              port (a,b : in bit_vector(15 downto 0);
                     y : out bit_vector(15 downto 0));
       end component sixteenBitSubtract;

       component unsigned_divider is
              port(   clk: in bit;
                            reset: in bit;
                            dividend: in bit_vector(15 downto 0);
                            divisor : in bit_vector(15 downto 0);
                            inputs_ready: in bit;
                            divider_ready : out bit;
                            quotient : out bit_vector(15 downto 0);
                            remainder : out bit_vector(15 downto 0);
                            output_ready: out bit;
                            output_accept: in bit );
       end component unsigned_divider;

       component GCD_controlpath is
              port (
              T0,T1,T2,T3,T4,T5: out bit;
              divDone, compareDone, count_var: in bit;
              start: in bit;
              srdy : out bit;
              erdy : in bit;
```

```
               done : out bit;
               clk, reset: in bit
          );
     end component GCD_controlpath;

     component GCD_datapath is
               port (
               T0,T1,T2,T3,T4,T5: in bit;
               divDone, compareDone, count_var : out bit;
               input : in bit_vector(15 downto 0);
               GCD: out bit_vector(15 downto 0);
               clk, reset: in bit
          );
     end component GCD_datapath;

     component System is
               port ( din: in bit_vector(15 downto 0);
                      dout: out bit_vector(15 downto 0);
                      start: in bit;
                      done: out bit;
                      erdy: in bit;
                      srdy: out bit;
                      clk: in bit;
                      reset: in bit);
     end component System;

end thepackage;
```

The other codes (unsigned_divider and package related...mostly a repetition of that in the previous report) :

unsigned_divider.vhd:

```
library work;
use work.thepackage.all;

entity unsigned_divider is
       port(   clk: in bit;
                      reset: in bit;
                      -- the two inputs
                      dividend: in bit_vector(15 downto 0);
                      divisor : in bit_vector(15 downto 0);
                      -- the next two implement a ready-ready
                      -- protocol to start the division
                      inputs_ready: in bit;
                      divider_ready : out bit;
                      -- the two outputs
                      quotient : out bit_vector(15 downto 0);
                      remainder : out bit_vector(15 downto 0);
                      -- the output ready-ready handshake
                      output_ready: out bit;
                      output_accept: in bit );
end entity unsigned_divider;

architecture Behave of unsigned_divider is
       signal T0, T1, T2, T3, T4, T5 : bit;
       signal start, done : bit;
       signal divShiftDone, compareDone, rightShiftDone : bit;
       signal random : bit := '0';

begin
       control : controlpath port map(T0 => T0, T1 => T1, T2 => T2, T3 => T3, T4 => T4, T5 => T5,
       divShiftDone => divShiftDone, compareDone=> compareDone, rightShiftDone => rightShiftDone,
       start => inputs_ready, done=> output_ready, clk=>clk, reset=>reset );
       datap : datapath port map(T0 => T0, T1 => T1, T2 => T2, T3 => T3, T4 => T4, T5 => T5,
       divShiftDone => divShiftDone, compareDone=> compareDone, rightShiftDone => rightShiftDone,
              dividend => dividend, divisor =>divisor, quotient =>quotient, remainder=>remainder,
              clk => clk, reset => reset );
       divider_ready <= random;

end Behave;
```

controlpath.vhd:

```
library work;
use work.thepackage.all;
entity controlpath is
       port (
              T0,T1,T2,T3,T4,T5: out bit;
              divShiftDone, compareDone, rightShiftDone: in bit;
              start: in bit;
              done : out bit;
              clk, reset: in bit
            );
end entity controlpath;

architecture Behave of controlpath is
       type FsmState is ( phi, leftshift, comparing, subtracting, doing, donestate);
       signal fsm_state : FsmState;
begin
   process(fsm_state, start, divShiftDone, compareDone, rightShiftDone, clk, reset)
      variable next_state: FsmState;
      variable Tvar: bit_vector(0 to 5);
      variable done_var: bit;
   begin
       Tvar := (others => '0');
       done_var := '0';
       next_state := fsm_state;

       case fsm_state is
          when phi =>
               if(start = '1') then
                  next_state := leftshift;
                  Tvar(0) := '1';
               end if;
          when leftshift =>
               if(divShiftDone = '1') then
               next_state := comparing;
               --Tvar(2) := '1';
               else
               Tvar(1) := '1';
               end if;
          when comparing =>
               Tvar(2) := '1';
               if(compareDone = '1') then
                      next_state := subtracting;
                      Tvar(5) := '1';
               else
                      next_state := doing;
                      Tvar(5) := '1';
               end if;
         when subtracting =>
               Tvar(4) := '1';
               next_state := doing;
          when doing =>
               if(rightShiftDone = '1') then
                      next_state := donestate;
               else
                      next_state := comparing;
                      Tvar(3) := '1';
                      --Tvar(2) := '1';
               end if;
          when donestate =>
               done_var := '1';
               next_state := phi;
     end case;

     T0 <= Tvar(0); T1 <= Tvar(1); T2 <= Tvar(2); T3 <= Tvar(3); T4 <= Tvar(4); T5 <= Tvar(5);
     done <= done_var;

     if(clk'event and (clk = '1')) then
        if(reset = '1') then
             fsm_state <= phi;
        else fsm_state <= next_state;
        end if;
     end if;
   end process;
end Behave;
```

datapath.vhd:

```
library work;
use work.thepackage.all;

entity datapath is
       port (
              T0,T1,T2,T3,T4,T5: in bit;
              divShiftDone, compareDone, rightShiftDone : out bit;
              Dividend , Divisor : in bit_vector(15 downto 0);
              Quotient, Remainder: out bit_vector(15 downto 0);
              clk, reset: in bit
            );
end entity datapath;

architecture Mixed of datapath is
    signal DDREG, DRREG: bit_vector(15 downto 0) := "0000000000000001";
    signal CREG: bit_vector(15 downto 0) := "0000000000000001";
    signal RREG, QREG: bit_vector(15 downto 0);
       constant C16 : bit_vector(15 downto 0) := (others => '0');
       constant C15 : bit_vector(15 downto 0) := "0000000000000001";

       signal DDREG_in, DRREG_in, QREG_in, CREG_in : bit_vector(15 downto 0);
       signal subtractOUT, lShiftDiv, lShiftCount, rShiftDiv, rShiftCount : bit_vector(15 downto 0);

       signal compare : bit;
       --signal arbit1, arbit2 : bit_vector(15 downto 0);
       signal x1, x2, x3 : bit;

       begin

       --writing down the inputs to the controlpath
       divShiftDone <= DRREG(15);
       rightShiftDone <= CREG(0);
       compareDone <= (compare and T2) ;

       --writing down the required registers
       dr1 : dataregister generic map (data_width => 16)
                          port map (Din => DDREG_in, Dout => DDREG, enable => x1, clk => clk);
       x1 <= T0 or T4;
       dr2 : dataregister generic map (data_width => 16)
                          port map (Din => DRREG_in, Dout => DRREG, enable => x2, clk => clk);
       x2 <= T0 or T1 or T3;
       dr3 : dataregister generic map (data_width => 16)
                          port map (Din => QREG_in, Dout => QREG, enable => x3, clk => clk);
       x3 <= T0 or T5;
       dr5 : dataregister generic map (data_width => 16)
                          port map (Din => CREG_in, Dout => CREG, enable => x2, clk => clk);

       --Now writing the logical calculators and calculations
       subtract : sixteenBitSubtract port map(a => DDREG, b => DRREG, y => subtractOUT);
       comparator : comparer port map(a => DDREG, b => DRREG, c => compare);
       lShiftDiv <= DRREG(14 downto 0) & '0';
       rShiftDiv <= '0' & DRREG(15 downto 1);
       lShiftCount <= CREG(14 downto 0) & '0';
       rShiftCount <= '0' & CREG(15 downto 1);

       --Now writing down all the transfers
       DDREG_in <= Dividend when (T0 = '1') else subtractOUT;
       DRREG_in <= Divisor when (T0 = '1') else lShiftDiv when (T1 = '1') else
                                              rShiftDiv when (T3 = '1') else DRREG;
       QREG_in <= C16 when (T0 = '1') else (QREG(14 downto 0) & compare);
       CREG_in <= C15 when (T0 = '1') else lShiftCount when (T1 = '1') else
                                              rShiftCount when (T3 = '1') else CREG;
       Remainder <= DDREG;
       Quotient <= QREG;

end Mixed;
```

dataregister.vhd:

```
library work;
use work.thepackage.all;
entity dataregister is
       generic (data_width:integer);
       port (Din: in bit_vector(data_width-1 downto 0); Dout: out bit_vector(data_width-1 downto 0);
             clk, enable: in bit);
end entity;
architecture Behave of dataregister is
begin
    process(clk)
    begin
       if(clk'event and (clk  = '1')) then
           if(enable = '1') then
               Dout <= Din;
           end if;
       end if;
    end process;
end Behave;
```

comparer.vhd:

```
library work;
use work.thepackage.all;
entity comparer is
       port(a, b : in bit_vector(15 downto 0);
               c : out bit);
end entity comparer;
architecture behave of comparer is
       begin
       c <= '0' when (b > a) else '1';
end architecture behave;
```

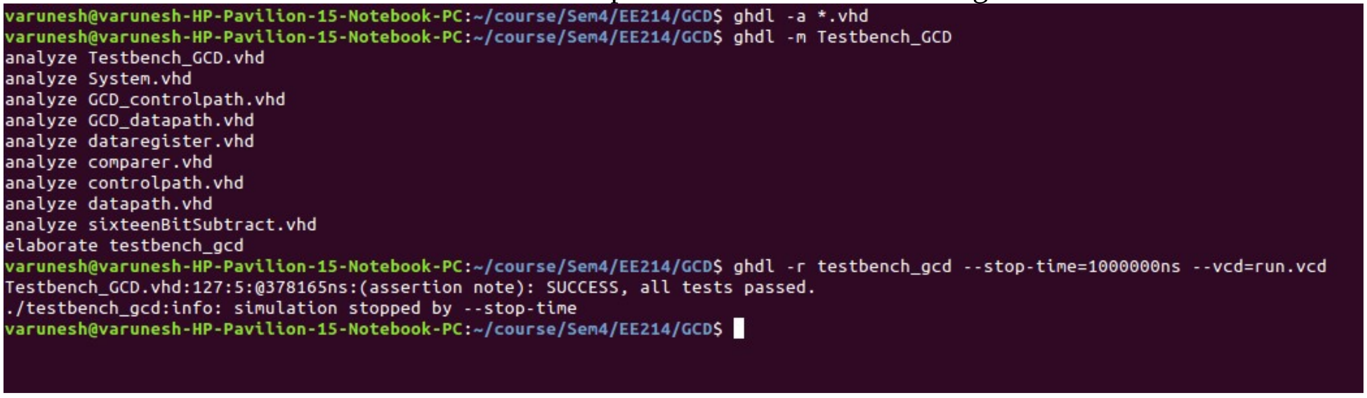
sixteenBitSubtract.vhd:

```
library work;
use work.thepackage.all;
entity sixteenBitSubtract is
       port (a,b : in bit_vector(15 downto 0);
               y : out bit_vector(15 downto 0));
end entity;
architecture basic of sixteenBitSubtract is
       signal c : bit_vector(15 downto 0);
       signal nb : bit_vector(15 downto 0);
begin
       nb <= not b;
       add0 : bitAdder port map ( x=> a(0), y=> nb(0), cin=> '1', s=> y(0), cout=>c(0));
       add1 : bitAdder port map ( x=> a(1), y=> nb(1), cin=> c(0), s=> y(1), cout=>c(1));
       add2 : bitAdder port map ( x=> a(2), y=> nb(2), cin=> c(1), s=> y(2), cout=>c(2));
       add3 : bitAdder port map ( x=> a(3), y=> nb(3), cin=> c(2), s=> y(3), cout=>c(3));
       add4 : bitAdder port map ( x=> a(4), y=> nb(4), cin=> c(3), s=> y(4), cout=>c(4));
       add5 : bitAdder port map ( x=> a(5), y=> nb(5), cin=> c(4), s=> y(5), cout=>c(5));
       add6 : bitAdder port map ( x=> a(6), y=> nb(6), cin=> c(5), s=> y(6), cout=>c(6));
       add7 : bitAdder port map ( x=> a(7), y=> nb(7), cin=> c(6), s=> y(7), cout=>c(7));
       add8 : bitAdder port map ( x=> a(8), y=> nb(8), cin=> c(7), s=> y(8), cout=>c(8));
       add9 : bitAdder port map ( x=> a(9), y=> nb(9), cin=> c(8), s=> y(9), cout=>c(9));
       add10: bitAdder port map ( x=> a(10), y=> nb(10), cin=> c(9), s=> y(10), cout=>c(10));
       add11: bitAdder port map ( x=> a(11), y=> nb(11), cin=> c(10), s=> y(11), cout=>c(11));
       add12: bitAdder port map ( x=> a(12), y=> nb(12), cin=> c(11), s=> y(12), cout=>c(12));
       add13: bitAdder port map ( x=> a(13), y=> nb(13), cin=> c(12), s=> y(13), cout=>c(13));
       add14: bitAdder port map ( x=> a(14), y=> nb(14), cin=> c(13), s=> y(14), cout=>c(14));
       add15: bitAdder port map ( x=> a(15), y=> nb(15), cin=> c(14), s=> y(15), cout=>c(15));
end basic;
```

bitAdder.vhd:

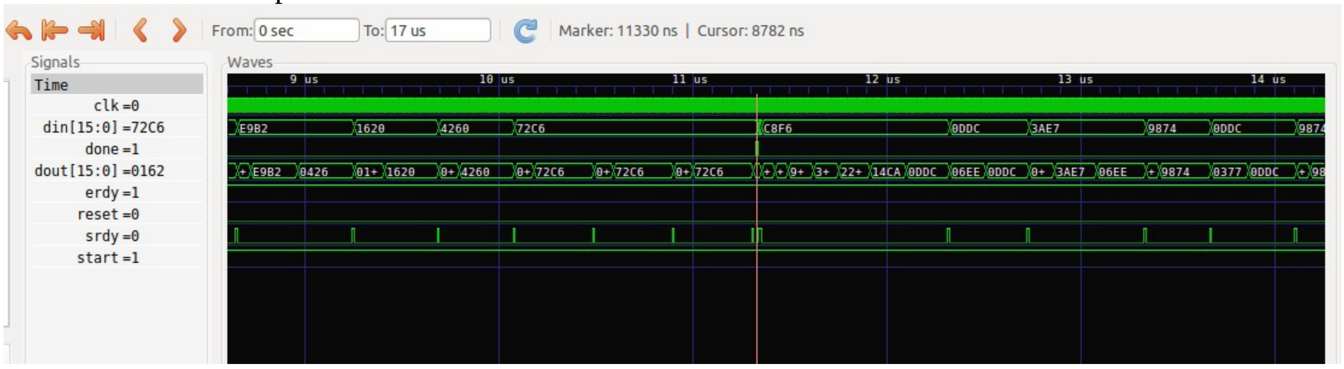
```
entity bitAdder is
       port(x, y, cin: in bit; cout, s: out bit);
end entity;
architecture simple of bitAdder is
begin
       s<= ((not x) and (not y) and cin) or
              ((not x) and y and (not cin)) or
              (x and (not y) and (not cin)) or
              (x and y and cin);
       cout<= (x and y) or (cin and y) or (cin and x);
end simple;
```

The terminal screenshot for succcessful compilation of all .vhd files and gtkwave simulation:

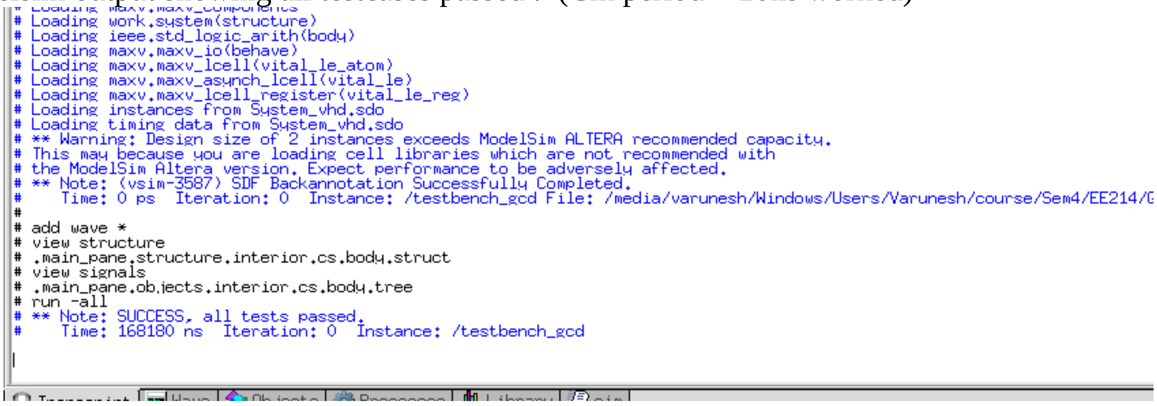


```
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214/GCD$ ghdl -a *.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214/GCD$ ghdl -m Testbench_GCD
analyze Testbench_GCD.vhd
analyze System.vhd
analyze GCD_controlpath.vhd
analyze GCD_datapath.vhd
analyze dataregister.vhd
analyze comparer.vhd
analyze controlpath.vhd
analyze datapath.vhd
analyze sixteenBitSubtract.vhd
elaborate testbench_gcd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214/GCD$ ghdl -r testbench_gcd --stop-time=1000000ns --vcd=run.vcd
Testbench_GCD.vhd:127:5:@378165ns:(assertion note): SUCCESS, all tests passed.
./testbench_gcd:info: simulation stopped by --stop-time
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214/GCD$
```

The GTKWave output screenshot :

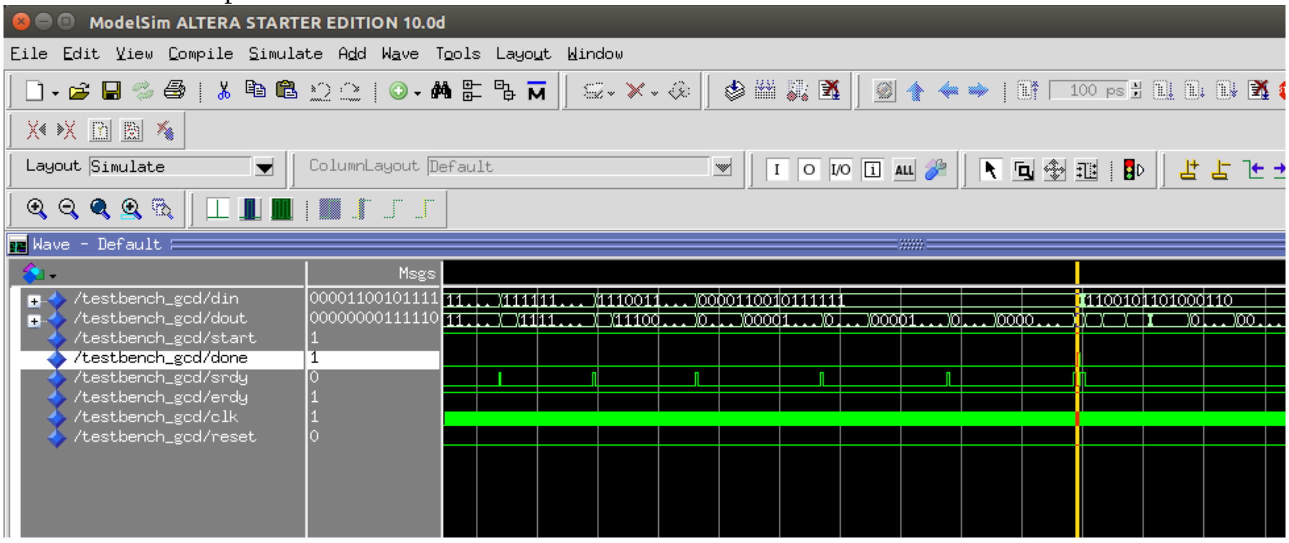


Modelsim output showing all testcases passed : (Clk period = 20ns worked)



```
# Loading maxv.maxv_components
# Loading work.system(structure)
# Loading ieee.std_logic_arith(body)
# Loading maxv.maxv_io(behave)
# Loading maxv.maxv_lcell(vital_le_atom)
# Loading maxv.maxv_asynch_lcell(vital_le)
# Loading maxv.maxv_lcell_register(vital_le_reg)
# Loading instances from System_vhd.sdo
# Loading timing data from System_vhd.sdo
# ** Warning: Design size of 2 instances exceeds ModelSim ALTERA recommended capacity.
# This may because you are loading cell libraries which are not recommended with
# the ModelSim Altera version. Expect performance to be adversely affected.
# ** Note: (vsim-3587) SDF Backannotation Successfully Completed.
#    Time: 0 ps  Iteration: 0  Instance: /testbench_gcd File: /media/varunesh/Windows/Users/Varunesh/course/Sem4/EE214/G
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 168180 ns  Iteration: 0  Instance: /testbench_gcd
```

Modelsim Wave snapshot :



ModelSim ALTERA STARTER EDITION 10.0d