It’s All About the Processors
14:332:439:02 - Embedded Systems Hardware/Software
Lab 5
Raphaelle Marcial
December 04, 2017

Purpose

This lab involved incorporating components from previous labs in designing and implementing a 16-bit GRISC processor utilizing instructions for video and communications. The instruction set architecture used by this processor somewhat mirrored those used by a general MIPS processor. This lab also introduced working in the IP Integrator since a block design interface was used in order to create the top-level design.

Prelab

Theory of Operation

The ALU was modified to perform the corresponding operations in the GRISC instruction set. Along with the traditional arithmetic instructions, this ALU also included instructions for utilizing the UART through the send and recv instructions. Moreover, the ALU performed operations on the pixels through the wpix and rpix instructions.

The pixel_pusher entity was modified to display a 64x64 resolution image by adding a vcount signal to the design. Furthermore, a 16-bit pixel signal was divided between the VGA RGB outputs to have 5, 6, and 5 bits respectively.

Design

myALU.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity myALU is
port (
   clk, en : in std_logic;
   A : in std_logic_vector(15 downto 0);
   B : in std_logic_vector(15 downto 0);
   Opcode : in std_logic_vector(3 downto 0);
   Y : out std_logic_vector(15 downto 0)
);
end myALU;

architecture Behavioral of myALU is

begin

   process(clk) begin
    if (rising_edge(clk) and en = '1') then
      case Opcode is
      when "0000" =>Y<= std_logic_vector(unsigned(A) + unsigned(B));
      when "0001" => Y<= std_logic_vector(unsigned(A) - unsigned(B));
      when "0010" => Y <= std_logic_vector(unsigned(A) sll 1);
      when "0011" => Y<= std_logic_vector(unsigned(A) srl 1);
      when "0100" => Y <= A(15) & A(15 downto 1);
      when "0101" => Y<=  A and B;
      when "0110" =>Y <=  A or B;
      when "0111" => Y <=  A xor B;
      when "1000" =>
```

```
      if (signed(A) < signed(B)) then
                              Y(0) <= '1';
                              Y(15 downto 1) <= (others => '0');
                           else
                              Y(15 downto 0) <= (others => '0');
                           end if;

      when "1001" =>

        if (signed(A) > signed(B)) then
                              Y(0) <= '1';
                              Y(15 downto 1) <= (others => '0');
                           else
                              Y(15 downto 0) <= (others => '0');
                           end if;
      when "1010" =>
      if signed(A) = signed(B) then
                              Y(0) <= '1';
                                    Y(15 downto 1) <= (others => '0');
                                 else
                                    Y(15 downto 0) <= (others => '0');
                           end if;
      when "1011" =>
      if A < B then
                              Y(0) <= '1';
                                    Y(15 downto 1) <= (others => '0');
                                 else
                                    Y(15 downto 0) <= (others => '0');
                           end if;
      when "1100" =>
       if A > B then
                                 Y(0) <= '1';
                                       Y(15 downto 1) <= (others => '0');
                                    else
                                       Y(15 downto 0) <= (others => '0');
                                 end if;
      when "1101" => Y <=  std_logic_vector(unsigned(A) + 1);
      when "1110" => Y<=     std_logic_vector(unsigned(A) - 1);
      when "1111" => Y <= std_logic_vector(0 - unsigned(A));



      when others => Y <= (others => '0');
      end case;
      end if;
   end process;

end Behavioral;
```

pixel_pusher.vhd

```
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use IEEE.NUMERIC_STD.ALL;

    entity pixel_pusher is
    Port (
       clk     : in std_logic;
       en      : in std_logic;
       VS      : in std_logic;
       pixel   : in std_logic_vector(15 downto 0);
```

```
    hcount  : in std_logic_vector(9 downto 0);
         vcount    : in std_logic_vector(9 downto 0);

    vid     : in std_logic;
    R       : out std_logic_vector(4 downto 0) := (others => '0');
    B       : out std_logic_vector(4 downto 0) := (others => '0');
    G       : out std_logic_vector(5 downto 0) := (others => '0');
    addr    : out std_logic_vector(11 downto 0):= (others => '0')

);
end pixel_pusher;

architecture Behavioral of pixel_pusher is

signal counter : std_logic_vector(11 downto 0);

begin

    addr_count: process(clk)
    begin
    if rising_edge(clk) and en = '1' then

        if VS = '0' then
            counter <= (others => '0');

        elsif vid = '1' and hcount < std_logic_vector(to_unsigned(64, 10)) and vcount < std_logic_vector(to_unsigned(64,
10)) then
            counter       <= std_logic_vector( unsigned(counter) + 1 );
            R <= pixel(15 downto 11);
            G <= pixel(10 downto 5);
            B <= pixel(4 downto 0);
            else
                R <= (others => '0');
                G <= (others => '0');
                B <= (others => '0');

        end if;

    end if; --end clock
    end process addr_count;


addr <= counter;

end Behavioral;
```

Part 1

Theory of Operation

This assembly program provided the instructions that were ran through the processor. Two loops were used in this program, the first printed “Hello_World” over the UART. After this string was printed, the second loop received user-input characters from the UART that were utilized in displaying different colored squares.

Design

helloWorld.asm

```
.data
val1: str "Hello_World"

.text

ori $r3 $zero 0
ori $r5 $zero 1
ori $r6 $zero 4095

ori $r8 $zero 0
beg:
// read char, if not zero print it else end
// la $r6 val2
lw $r4 $r3 val1
beq $r4 $zero end
send $r4 //send
add $r3 $r3 $r5 // increment char pointer
j beg

end:
recv $r7
ori $r8 $zero 0
loop:
beq $r8 $r6 end
wpix $r8 $r7
add $r8 $r8 $r5
j loop
```

Simulation

It worked.

Part 2

Theory of Operation

The register entity was created to infer a dual port memory. This memory had both reading and writing capabilities over 32 16-bit words. When designing the memory, an array of 32 indices was used and each index contained a vector of 16 bits. Reading from the memory meant simply assigning the data out signal to its respective register ID. Writing to the memory assigned the input data to the respective register ID. Enables were used so the ports could be used independently.

The framebuffer entity was created to infer a dual port RAM memory. This memory had reading capabilities in both ports and writing in one port over 4096 16-bit words. It was designed similarly as it was an array of 4096 indices where each index contained a 16-bit vector. This entity was different from the ones design so far as there were two clocks utilized. However, they were controlled by the outputs of separate clock divider entities instead of the original clock input.

Design

regs.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity regs is
Port (
        clk, en, reset  : in std_logic;
        id1, id2        : in std_logic_vector(4 downto 0); --addresses
        wr_en1, wr_en2  : in std_logic;
        din1, din2              : in std_logic_vector(15 downto 0);
        dout1, dout2    : out std_logic_vector(15 downto 0)
);
end regs;

architecture Behavioral of regs is


        --Memory consisting of 32 16-bit words
        type memory is array (0 to 31) of std_logic_vector(15 downto 0);
        signal registers    : memory := (others => (others => '0'));

begin

    dout1 <= registers(to_integer(unsigned(id1)));
    dout2 <= registers(to_integer(unsigned(id2)));

  write : process(clk, reset, en)
  begin

   if reset = '1' then
        registers <= (others => (others => '0'));

  elsif rising_edge(clk) and en = '1' then
                registers(0) <= (others => '0');


                if wr_en1 = '1' then
```

```
                registers(to_integer(unsigned(id1))) <= din1;
            end if;

            if wr_en2 = '1' then
                registers(to_integer(unsigned(id2))) <= din2;
            end if;

    end if; --end clock

    end process write;


end Behavioral;
```

framebuffer.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity framebuffer is
Port (
        --clk, en1, en2        : in std_logic;
        clk_a, clk_b        : in std_logic;
        addr1, addr2                    : in std_logic_vector(11 downto 0); --addresses
        wr_en1                          : in std_logic;
        din1                            : in std_logic_vector(15 downto 0);
        dout1, dout2                    : out std_logic_vector(15 downto 0)
);
end framebuffer;

architecture Behavioral of framebuffer is

        --Memory  consisting of 4096 16-bit words
        type memory is array (0 to 4095) of std_logic_vector(15 downto 0);
        signal memSignal  : memory := (others => (others => '0'));

begin

-- Port A
process(clk_a)
begin
    if(rising_edge(clk_a)) then
        if(wr_en1='1') then
            memSignal(to_integer(unsigned(addr1))) <= din1;
        end if;
        dout1 <=  memSignal(to_integer(unsigned(addr1)));
   end if;
end process;

--Port B
process (clk_b)
begin
    if(rising_edge(clk_b)) then

        dout2 <=  memSignal(to_integer(unsigned(addr2)));

    end if;
end process;


end Behavioral;
```

Part 3

Theory of Operation

The controller was a finite state machine that dictated the behavior of each instruction as it was executed in the processor. This behavior basically followed the datapath of a general MIPS processor since the stages involved fetching, decoding, executing, and reading/writing to memory (if applicable). In addition to the usual instructions, there were also states for utilizing the UART and interfacing with pixel data. Some states were broken into multiple clock cycles to account for the latencies in reading from the memory blocks.

Design

controls.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.NUMERIC_STD.ALL;

entity controls is port (
-- Timing Signals
clk, en, rst : in std_logic;
-- Register File IO
rID1, rID2 : out std_logic_vector(4 downto 0);
wr_enR1, wr_enR2 : out std_logic;
regrD1, regrD2 : in std_logic_vector(15 downto 0);
regwD1, regwD2 : out std_logic_vector(15 downto 0);
-- Framebuffer IO
fb_wr_en : out std_logic;
--fbRST : out std_logic;
fbAddr1 : out std_logic_vector(11 downto 0);
fbDin1 : in std_logic_vector(15 downto 0);
fbDout1 : out std_logic_vector(15 downto 0);
-- Instruction Memory IO
irAddr : out std_logic_vector(13 downto 0);
irWord : in std_logic_vector(31 downto 0);
-- Data Memory IO
dAddr : out std_logic_vector(14 downto 0);
d_wr_en : out std_logic;
dOut : out std_logic_vector(15 downto 0);
dIn : in std_logic_vector(15 downto 0);
-- ALU IO
aluA, aluB : out std_logic_vector(15 downto 0);
aluOp : out std_logic_vector(3 downto 0);
aluResult : in std_logic_vector(15 downto 0);
-- UART IO
ready, newChar : in std_logic;
send : out std_logic;
charRec : in std_logic_vector(7 downto 0);
charSend : out std_logic_vector(7 downto 0)
);
end controls;

architecture Behavioral of controls is
        --Signals
        signal instruct : std_logic_vector(31 downto 0) := (others => '0');
        signal pc     : std_logic_vector(15 downto 0) := (others => '0');
        signal result  : std_logic_vector(15 downto 0) := (others => '0');
```

```
        --Rops Signals
        signal opcode_r    : std_logic_vector(4 downto 0) := (others => '0');
        signal source1_r   : std_logic_vector(4 downto 0) := (others => '0');
        signal source2_r   : std_logic_vector(4 downto 0) := (others => '0');
        signal dest_r      : std_logic_vector(4 downto 0) := (others => '0');

        --Iops Signals
        signal opcode_i    : std_logic_vector(4 downto 0) := (others => '0');
        signal source_i    : std_logic_vector(4 downto 0) := (others => '0');
        signal dest_i      : std_logic_vector(4 downto 0) := (others => '0');
        signal imm_i       : std_logic_vector(15 downto 0) := (others => '0');

        --Jops Signals
        signal opcode_j    : std_logic_vector(4 downto 0) := (others => '0');
        signal dest_j      : std_logic_vector(15 downto 0) := (others => '0');

        signal regA, regB, regC : std_logic_vector(15 downto 0);
        signal sum : std_logic_vector(15 downto 0);

   signal count : std_logic := '0';

        --FSM States
        type state is (fetch, decode, Rops, Iops, Jops, calc, store, jr, recv, rpix, wpix, send_state, equals, nequal, ori, lw,
sw, jmp, jal,  finish, getAluResult, getAluResult2, lw2, send_temp); --clrsrc
        signal curr : state := fetch;


        begin

        p: process (clk)
        begin

                if rst = '1' then
                 --- SET EVERYTHING TO 0
                  rID1 <= (others => '0');
                  rID2 <= (others => '0');
                  wr_enR1 <= '0';
                  wr_enR2 <= '0';
                  regwD1 <= (others => '0');
                  regwD2 <= (others => '0');
                  --fbRST <= '0';
                  fbAddr1 <= (others => '0');
                  fbDout1 <= (others => '0');
                  irAddr <= (others => '0');
                  dAddr <= (others => '0');
                  d_wr_en <= '0';
                  aluA <= (others => '0');
                  aluB <= (others => '0');
                  aluOp <= (others => '0');
                  send <= '0';
                  charSend <= (others => '0');

                  instruct <= (others => '0');
                  pc <= (others => '0');
                  result <= (others => '0');
                  opcode_r <= (others => '0');
                  source1_r <= (others => '0');
                  source2_r <= (others => '0');
                  dest_r <= (others => '0');
                  opcode_i <= (others => '0');
                  source_i <= (others => '0');
```

```
            dest_i <= (others => '0');
            imm_i <= (others => '0');
            opcode_j <= (others => '0');
            dest_j <= (others => '0');
            regA <= (others => '0');
            regB <= (others => '0');
            regC <= (others => '0');
            sum <= (others => '0');


            curr <= fetch;


          elsif rising_edge(clk) and en = '1' then

                    case curr is
                              --Fetch
                              when fetch =>
                                 if count = '0' then
                                    rID1 <= "00001";
                                    curr <= fetch;
                                    count <= '1';

                                 else
                                    pc <= regrD1;
                                    irAddr   <= regrD1(13 downto 0);
                                    report integer'image(to_integer(unsigned(pc)));
                                 --NS
                                          curr <= decode;
   count <= '0';
end if;
                              --Decode
                              when decode =>

                                 if count = '0' then
                                    instruct <= irWord;

                                    curr <= decode;
                                    count <= '1';
                                 else
                                 --increment
                                    rID1 <= "00001";
                                    regwD1 <= std_logic_vector( unsigned(pc) + 1 );

                                    wr_enR1 <= '1';

                                 --NS
                                    count <= '0';
                                    if instruct(31 downto 30) = "00" or instruct(31 downto 30) = "01" then
      curr <= rops;
   elsif instruct(31 downto 30) = "10" then
      curr <= iops;
   else
      curr <= jops;
   end if;
end if;


                              --Rops
                              when rops =>
                                 wr_enR1 <= '0';
```

```
                opcode_r    <= instruct(31 downto 27);
                dest_r      <= instruct(26 downto 22);
                source1_r   <= instruct(21 downto 17);
                source2_r   <= instruct(16 downto 12);


                rID1 <= instruct(21 downto 17);
                rID2 <= instruct(16 downto 12);

                regA <= regrD1;
                regB <= regrD2;

                --NS
                if instruct(31 downto 27) = "01101" then
                   curr <= jr;
                elsif instruct(31 downto 27) = "01100" then
                   curr <= recv;
                elsif instruct(31 downto 27) = "01111" then
                   curr <= rpix;
                elsif instruct(31 downto 27) = "01110" then
                   curr <= wpix;
                     rID2 <= instruct(26 downto 22);
                elsif instruct(31 downto 27) = "01011" then
                   curr <= send_state;
                else
                   curr <= calc;
                end if;


             --Iops
             when iops =>
                wr_enR1 <= '0';
                opcode_i    <= instruct(31 downto 27);
                dest_i      <= instruct(26 downto 22);
                source_i    <= instruct(21 downto 17);
                imm_i       <= instruct(16 downto 1);

                rID1 <= instruct(26 downto 22);
                rID2 <= instruct(21 downto 17);

                regA <= regrD1;
                regB <= regrD2;


if instruct(29 downto 27) = "000" then
   curr <= equals;
elsif instruct(29 downto 27) = "001" then
   curr <= nequal;
elsif instruct(29 downto 27) = "010" then
   curr <= ori;
elsif instruct(29 downto 27) = "011" then
   curr <= lw;
else
   curr <= sw;
end if;


             --Jops
             when jops =>
                wr_enR1 <= '0';
                opcode_j <= instruct(31 downto 27);
```

```
                dest_j   <= instruct(26 downto 11);

                --rID1 <= instruct(26 downto 11);
                --regC <= regRD1;

                --NS
                if instruct(31 downto 27) = "11000" then
                   curr <= jmp;
                elsif instruct(31 downto 27) = "11001" then
                   curr <= jal;
                else
                   curr <=  finish;
                end if;


              --calc
              when calc =>

                aluOp <= opcode_r(3 downto 0);
                aluA  <= regrD1;
                aluB  <= regrD2;
                curr <= getAluResult;

               when getAluResult =>

                --result <= aluResult;
                curr <= getAluResult2;

               when getAluResult2 =>

        result <= aluResult;
        curr <= store;
              --store

              when store =>
                if count = '0' then
                   rID1 <= instruct(26 downto 22);

regwD1 <= result;
                   wr_enR1 <= '1';
                   count <= '1';
                   curr <= store;
                 else
                   wr_enR1 <= '0';
                   count <= '0';
                   --NS
                   curr <= finish;
                end if;
              --jr
              when jr =>
                --rID1 <= dest_j;
                --result <= regRD1;
                result <= dest_j;
                --NS
                curr <= store;

              --recv
              when recv =>
                result <= "00000000" & charRec;
                --NS
                if newChar = '0' then
```

```
                                    curr <= recv;
                                  else
                                    curr <= store;
                                  end if;

                                --rpix
                                when rpix =>
                                  --rID2 <= instruct(21 downto 17);
                                  --fbAddr1 <=regrD2(11 downto 0);
                                  fbAddr1 <= regrD2(11 downto 0);

                                  result <= fbDin1;

                                  --NS
                                  curr <= store;


                                --wpix
                                when wpix =>
                                  if count = '0' then

                                    fbAddr1 <= regrD2(11 downto 0);
                                    --fbAddr1 <= regA(11 downto 0);


                                    count <= '1';
                                    curr <= wpix;
                                  else
                                    fb_wr_en <= '1';
  rID1 <= instruct(21 downto 17);
  fbDout1 <= regrD1;
                                    count <= '0';
                                  --NS
                                    curr <= finish;
                                  end if;
                                --send
                                when send_state =>

                                  if count = '0' then

                                    rID1 <= dest_r;

                                    count <= '1';
                                    curr <= send_state;
else
                                  --NS
                                   send <= '1';
                                    charSend <= regrD1(7 downto 0);
                                    --send <= '1';
                                    count <= '0';
                                    curr <= send_temp;
                                 end if;

                                 when send_temp =>
                                  send <= '0';
                                  if ready = '1' then
                    curr <= finish;
                    count <= '0';

                  else
                    count <= '0';
```

```
    curr <= send_temp;
end if;


                --equals
                when equals =>
                    if regRD1 = regRD2 then
                        result <= instruct(16 downto 1);
                        instruct(26 downto 22) <= "00001";
                        curr <= store;
                        --dest_i <= regrD1;
                    else
                        curr <= finish;
                    end if;


                --nequal
                when nequal =>
                    if regRD1 /= regRD2 then
                            result <= imm_i;
                            dest_i <= "00001";
                            curr <= store;
                    else
                        curr <= finish;

                    end if;

                    --NS


                --ori
                when ori =>

                    result <= regB OR imm_i;
                    --NS
                    curr <= store;
                --lw
                when lw =>
                    if count = '0' then
                        sum <= std_logic_vector( unsigned(regrD2) + unsigned(imm_i) );
                        count <= '1';
                        curr <= lw;

                    else
                         dAddr <= sum(14 downto 0);

                    --NS
                        curr <= lw2;
                        count <= '0';
                     end if;

                  when lw2 =>

                        result <= dIn;
                        curr <= store;

                --sw
                when sw =>
                    if count = '0' then
                        sum <= std_logic_vector( unsigned(regrD2) + unsigned(imm_i) );
                        dAddr <= sum(14 downto 0);
```

```
                                        --dOut <= std_logic_vector( unsigned(regB) + unsigned(imm_i) );
                                          d_wr_en <= '1';
                                          rID1 <= dest_i;
                                          dout <= regrD1;
                                          count <= '1';
                                          curr <= sw;
                                        else
                                          d_wr_en <= '0';
                                          count <= '0';
                                        --dAddr <= regrD1;
                                        --NS
                                          curr <= finish;
                                        end if;

                                      --jmp
                                      when jmp =>
                                        if count = '0' then
  rID1 <= "00001";
  count <= '1';
  curr <= jmp;
  wr_enR1 <= '1';
  regWD1 <= instruct(26 downto 11);
else

  wr_enR1 <= '0';
 count <= '0';

                                        --NS
                                         curr <= finish;
                                    end if;
                                      --jal
                                      when jal =>
                                        if count = '0' then
                                          rID1 <= "00001"; --pc
                                          rID2 <= "00010"; --ra
                                          wr_enR1 <= '1';
                                          wr_enR2 <= '1';


                                          regWD2 <=  pc;
                                          regWD1 <= regC;
                                          count <= '1';
                                          curr <= jal;
                                        else
                                          --wr_enR1 <= '0';
    --wr_enR2 <= '0';
                                          count <= '0';
                                        --NS
                                          curr <= finish;
                                        end  if;

                                      --clrscr
--                                    when clrscr =>
--                                      fbRST <= '1';
--                                      --NS
--                                      curr <= finish;

                                      --finish

                                      when finish =>
--                                      fbRST <= '0';
```

```
                    d_wr_en <= '0';
                    wr_enR1 <= '0';
                    wr_enR2 <= '0';
                    count <= '0';
                    fb_wr_en <= '0';
                        curr <= fetch;
                when others =>
                        curr <= fetch;

            end case;

        end if;

    end process p;

end Behavioral;
```

Simulation

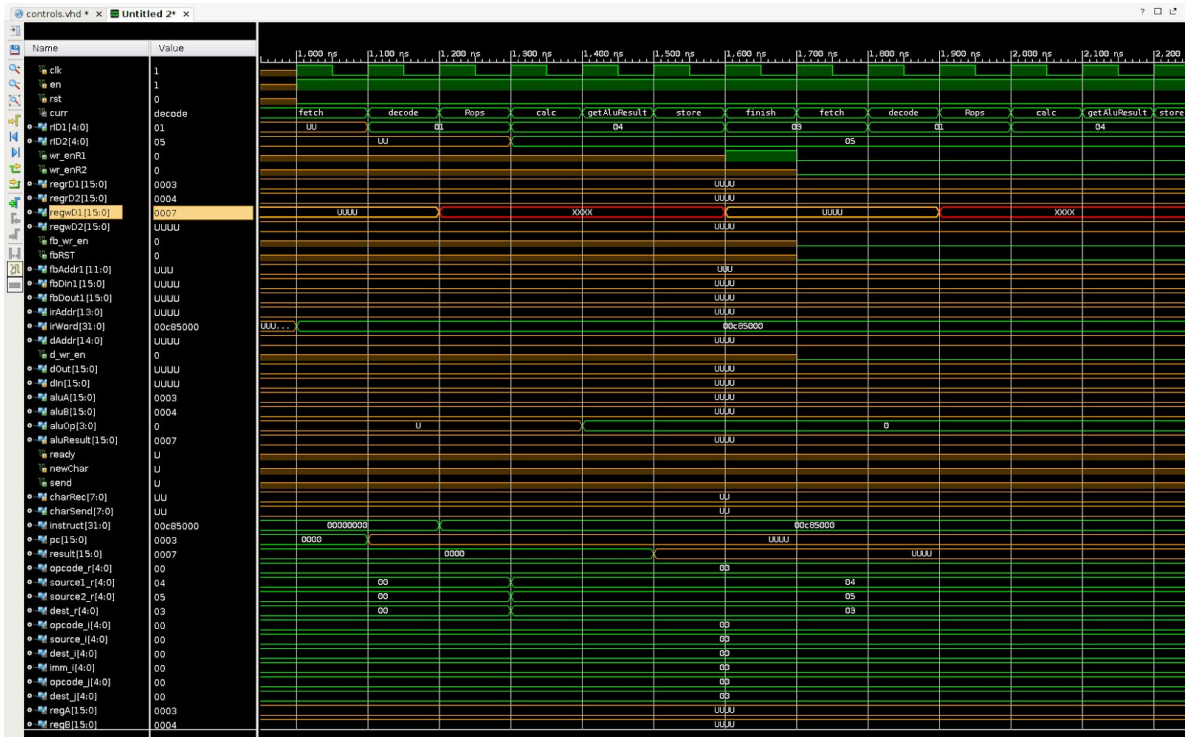

Figure 1: Waveform of “add $r3 $r4 $r5”

Part 4

Theory of Operation

The top-level design combined all the components in the aforementioned parts along with entities used in previous labs to create the processor. Unlike previous labs, a graphical tool was utilized in order to simplify the process of connecting the entities. The inputs for this design were the clock, the txd port of the pmod, and a button that reset the registers, signals in the controllers, and the contents of the UART. The outputs were the vga display signals and the rxd port of the pmod. As soon as the FPGA was programmed, the string “Hello_World” was displayed on the termite terminal. Furthermore, the characters inputted via the terminal were used in displaying different colored squares.

Design

debounce.vhd

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

entity debounce is
port (
  clk   : in std_logic;
  btn   : in std_logic;
  dbnc  : out std_logic
);
end debounce;

architecture Behavioral of debounce is
   signal count     : std_logic_vector (21 downto 0) := (others => '0');
   signal sregister : std_logic_vector (1 downto 0)  := (others => '0');

begin
    process(clk)
    begin

       if rising_edge(clk) then
          sregister(1)    <= sregister(0);
          sregister(0)    <= btn;

          if count =  std_logic_vector(to_unsigned(2500000,22)) then
             dbnc <= '1';
          else
             dbnc <= '0';
          end if;

          if sregister(1) = '1' then
              if count       /=  std_logic_vector(to_unsigned(2500000,22)) then
                      count <= std_logic_vector(unsigned(count)+ 1);
              end if;
           else
            count <= (others => '0');
           end if;

       end if;

    end process;

end Behavioral;
```

clock_1div.vhd (115200 Hz)

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

entity clock_div1 is
port (
  clk : in std_logic;
  div : out std_logic --new clock
);
end clock_div1;

architecture Behavioral of clock_div1 is
  signal count   : std_logic_vector (10 downto 0) := (others => '0');
  signal clk_1 : std_logic := '0';
begin

  process(clk)
  begin

    if rising_edge(clk) then

      if count  =  std_logic_vector(to_unsigned(1084,11)) then
       count    <= (others => '0');
       clk_1  <= '1';
       else
       clk_1  <= '0';
       count    <=  std_logic_vector( unsigned(count) + 1 );
      end if;

    end if;

  end process;

  div <= clk_1;

end Behavioral;
```

clock_2div.vhd (25 MHz)

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

entity clock_div2 is
port (
  clk : in std_logic;
  div : out std_logic --new clock
);
end clock_div2;

architecture Behavioral of clock_div2 is
  signal count   : std_logic_vector (2 downto 0) := (others => '0');
  signal clk_1 : std_logic := '0';
begin

  process(clk)
  begin

    if rising_edge(clk) then
```

```
    if count  =  std_logic_vector(to_unsigned(4,3)) then
     count    <= (others => '0');
     clk_1  <= '1';
     else
     clk_1  <= '0';
     count    <=  std_logic_vector( unsigned(count) + 1 );
    end if;

   end if;

 end process;

 div <= clk_1;

end Behavioral;
```

vga_ctrl.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity vga_ctrl is
   Port ( clk : in STD_LOGIC;
          en : in STD_LOGIC;
          hcount : out STD_LOGIC_VECTOR (9 downto 0);
          vcount : out STD_LOGIC_VECTOR (9 downto 0);
          vid : out STD_LOGIC;
          hs : out STD_LOGIC;
          vs : out STD_LOGIC);
end vga_ctrl;

architecture Behavioral of vga_ctrl is
signal hcount_s : std_logic_vector(9 downto 0) := (others => '0');
signal vcount_s : std_logic_vector(9 downto 0) := (others => '0');

begin

   incrementing_hcount:process(clk)
   begin

      if rising_edge(clk) then
         if en = '1' then
            if hcount_s        = std_logic_vector(to_unsigned(799, 10)) then
               hcount_s       <= (others => '0');
             else
               hcount_s    <= std_logic_vector( unsigned(hcount_s) + 1 );
            end if;
         end if;
      end if;
   end process incrementing_hcount;

   incrementing_vcount:process(clk)
   begin
      if rising_edge(clk) then
         if en = '1' then
            if hcount_s = std_logic_vector(to_unsigned(0, 10)) then
               if vcount_s    = std_logic_vector(to_unsigned(524, 10)) then
                  vcount_s    <= (others => '0');
               else
                  vcount_s       <= std_logic_vector( unsigned(vcount_s) + 1 );
               end if;
```

```
            end if;
         end if;
      end if;
   end process incrementing_vcount;

   display:process(hcount_s, vcount_s)
   begin
      if hcount_s >= std_logic_vector(to_unsigned(0, 10)) and hcount_s <= std_logic_vector(to_unsigned(639, 10)) and
vcount_s >= std_logic_vector(to_unsigned(0, 10)) and vcount_s <= std_logic_vector(to_unsigned(479, 10)) then
         vid <= '1';
      else
         vid <= '0';
      end if; --end display on/off condition
   end process display;

   hs_signal:process(hcount_s)
   begin
      if hcount_s >= std_logic_vector(to_unsigned(656, 10)) and hcount_s <= std_logic_vector(to_unsigned(751, 10)) then
         hs <= '0';
      else
         hs <= '1';
      end if;
   end process hs_signal;

   vs_signal:process(vcount_s)
   begin
      if vcount_s >= std_logic_vector(to_unsigned(490, 10)) and vcount_s <= std_logic_vector(to_unsigned(491, 10)) then
         vs <= '0';
      else
         vs <= '1';
      end if;
   end process vs_signal;



   hcount  <= hcount_s;
   vcount  <= vcount_s;
end Behavioral;
```

uart.vhd (Lab 3 solution)

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

entity uart is port (
            clk, en, send, rx, rst : in std_logic;
   charSend : in std_logic_vector (7 downto 0);
            ready, tx, newChar : out std_logic;
            charRec : out std_logic_vector (7 downto 0)
);
end uart;

architecture structural of uart is
            component uart_tx port
   (
      clk, en, send, rst : in std_logic;
      char : in std_logic_vector (7 downto 0);
      ready, tx : out std_logic
   );
   end component;

   component uart_rx port
```

```
    (
        clk, en, rx, rst : in std_logic;
        newChar : out std_logic;
        char : out std_logic_vector (7 downto 0)
    );
            end component;

begin

            r_x: uart_rx port map(
            clk => clk,
            en => en,
        rx => rx,
        rst => rst,
        newChar => newChar,
        char => charRec);

    t_x: uart_tx port map(
                        clk => clk,
                        en => en,
        send => send,
        rst => rst,
        char => charSend,
        ready => ready,
        tx => tx);

end structural;
```

tx.vhd (Lab 3 solution)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity uart_tx is
    Port ( clk : in STD_LOGIC;
           en : in STD_LOGIC;
           send : in STD_LOGIC;
           rst : in STD_LOGIC;
           char : in STD_LOGIC_VECTOR (7 downto 0);
           ready : out STD_LOGIC;
           tx : out STD_LOGIC);
end uart_tx;

architecture Behavioral of uart_tx is

    type state is (idle, data, stop);
    signal curr : state := idle;

    --shift register
    signal d       : std_logic_vector (7 downto 0) := (others => '0');

    --counter
    signal count    : std_logic_vector (2 downto 0) := (others => '0');


begin

    process (clk) begin

        if rst = '1' then
```

```
      curr    <=     idle;
      d       <=     (others => '0');
      count   <=     (others => '0');
      tx      <=     '1';
      ready   <=     '1';

    elsif rising_edge(clk) and en = '1' then
      case curr is

        when idle =>
          if send = '1' then
            d       <= char;
            ready   <= '0';
            count   <= (others => '0');
            curr    <= data;
            tx      <= '0';

          else
            tx  <= '1';
            ready <= '1';

          end if;
        when data =>
          tx  <= d(0);
          d   <= '0' & d(7 downto 1);
          if unsigned(count) < 7 then
            count <= std_logic_vector(unsigned(count) + 1);
          else
            curr <= stop;
          end if;

        when stop =>
          tx      <= '1';
          ready   <= '0';
          curr    <= idle;

        when others =>
          curr <= idle;
      end case;
    end if;

  end process;
end Behavioral;
```

rx.vhd (Lab 3 solution)

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

entity uart_rx is port
(
        clk, en, rx, rst : in std_logic;
        newChar : out std_logic;
        char : out std_logic_vector (7 downto 0)
);
end uart_rx;

architecture fsm of uart_rx is
```

```
        -- state type enumeration and state variable
        type state is (idle, start, data);
        signal curr : state := idle;

        -- shift register to read data in
        signal d : std_logic_vector (7 downto 0) := (others => '0');

        -- counter for data state
        signal count : std_logic_vector(2 downto 0) := (others => '0');

        -- double flop rx plus 2 extra samples to take majority vote of 3
        -- majority vote of samples helps mitigate noise on line
        signal inshift : std_logic_vector(3 downto 0) := (others => '0');
  signal maj : std_logic := '0';

begin

        -- double flop input to fix potential metastability
  -- plus 2 extra samples to take majority vote of 3 inputs (oversampling)
        -- majority vote of samples helps mitigate noise on line
        process(clk) begin
                if rising_edge(clk) then
                        inshift <= inshift(2 downto 0) & rx;
                end if;
        end process;

  -- majority vote of 3 samples (oversampling)
        -- majority vote of samples helps mitigate noise on line
  process(inshift) begin
        if      (inshift(3) = '1' and inshift(2) = '1' and inshift(1) = '1') or
        (inshift(3) = '1' and inshift(2) = '1') or
        (inshift(2) = '1' and inshift(1) = '1') or
      (inshift(3) = '1' and inshift(1) = '1') then
        maj <= '1';
    else
        maj <= '0';
    end if;
  end process;

        -- FSM process (single process implementation)
  process(clk) begin

                -- Resets the state machine and its outputs
        if rst = '1' then

        curr <= idle;
      d <= (others => '0');
      count <= (others => '0');
      newChar <= '0';

        elsif rising_edge(clk) and en = '1' then
        case curr is

        when idle =>
        newChar <= '0';
        if maj = '0' then
                curr <= start;
           end if;

        when start =>
        d <= maj & d(7 downto 1);
```

```
                count <= (others => '0');
                curr <= data;

            when data =>
            if unsigned(count) < 7 then
                        d <= maj & d(7 downto 1);
                    count <= std_logic_vector(unsigned(count) + 1);
                elsif maj <= '1' then
                        curr <= idle;
                    newChar <= '1';
                    char <= d;
                else
                        curr <= idle;
            end if;

            when others =>
            curr <= idle;

        end case;
    end if;
  end process;

end fsm;
```

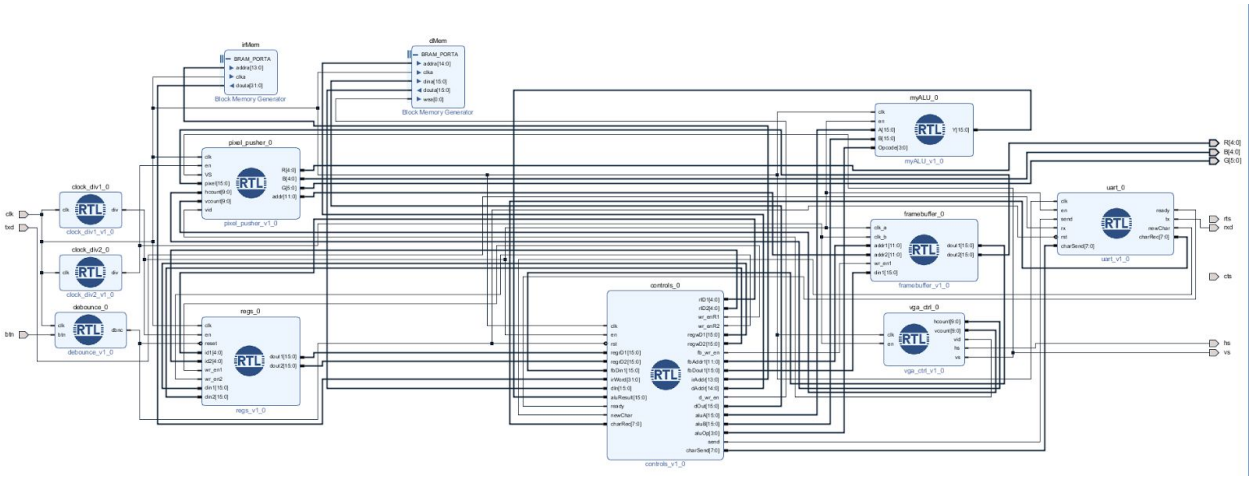

Figure 2: Top-Level Block Design

Simulation

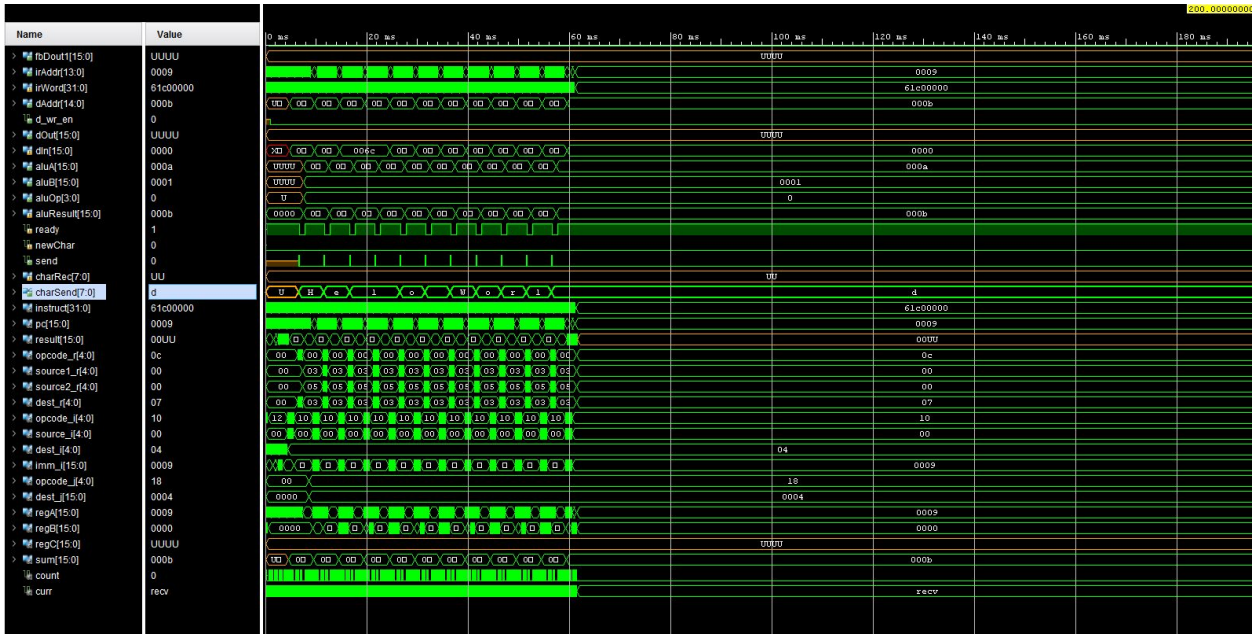

Figure 3: Waveforms when txd = '1' -> Hello_World

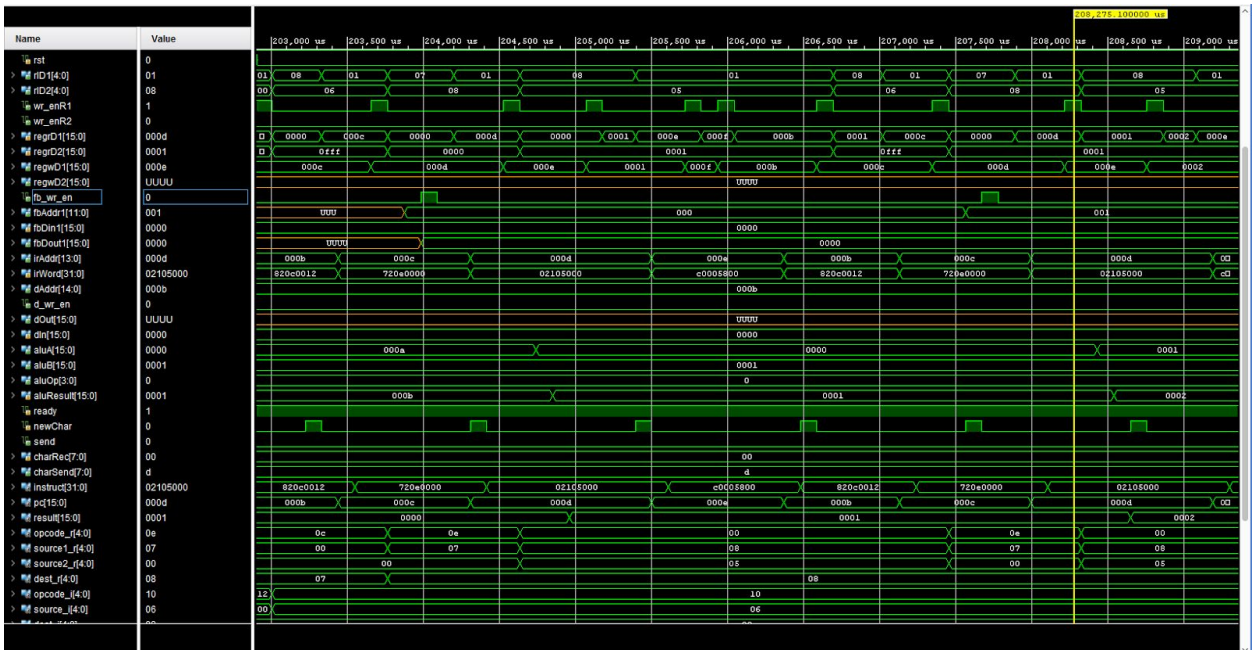

Figure 3: Waveforms when txd = '0' -> wpix

Implementation

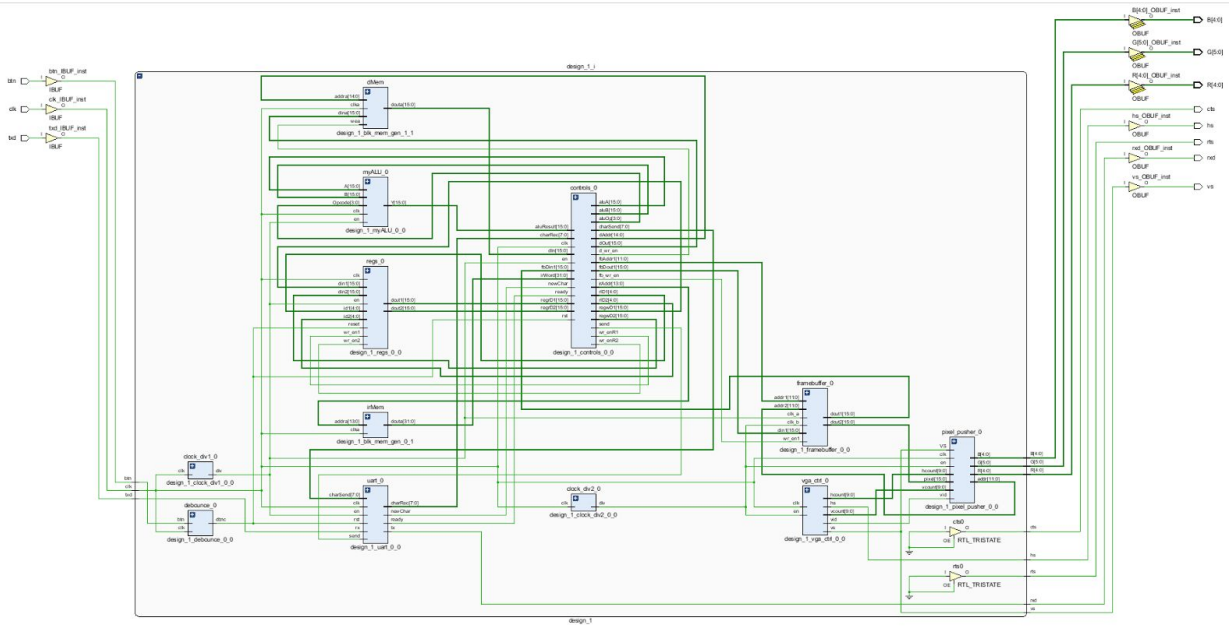

Figure 4: Elaboration Schematic

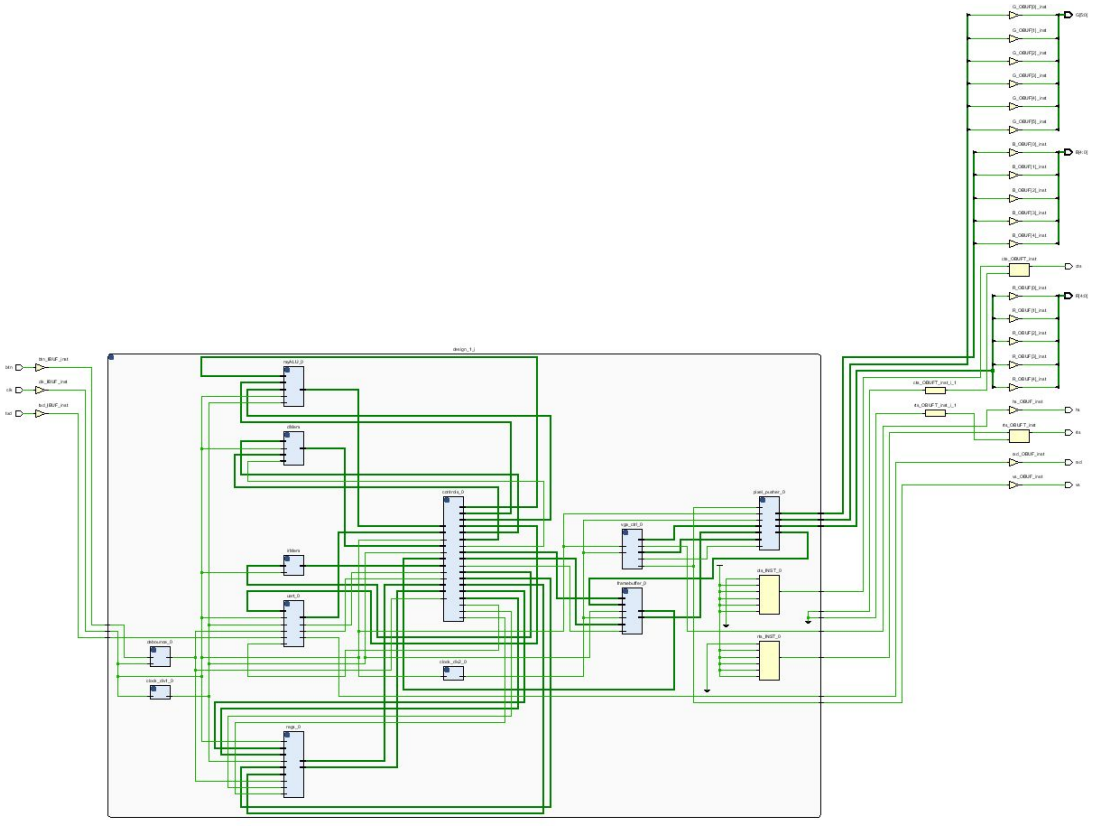

Figure 5: Full Synthesis Schematic

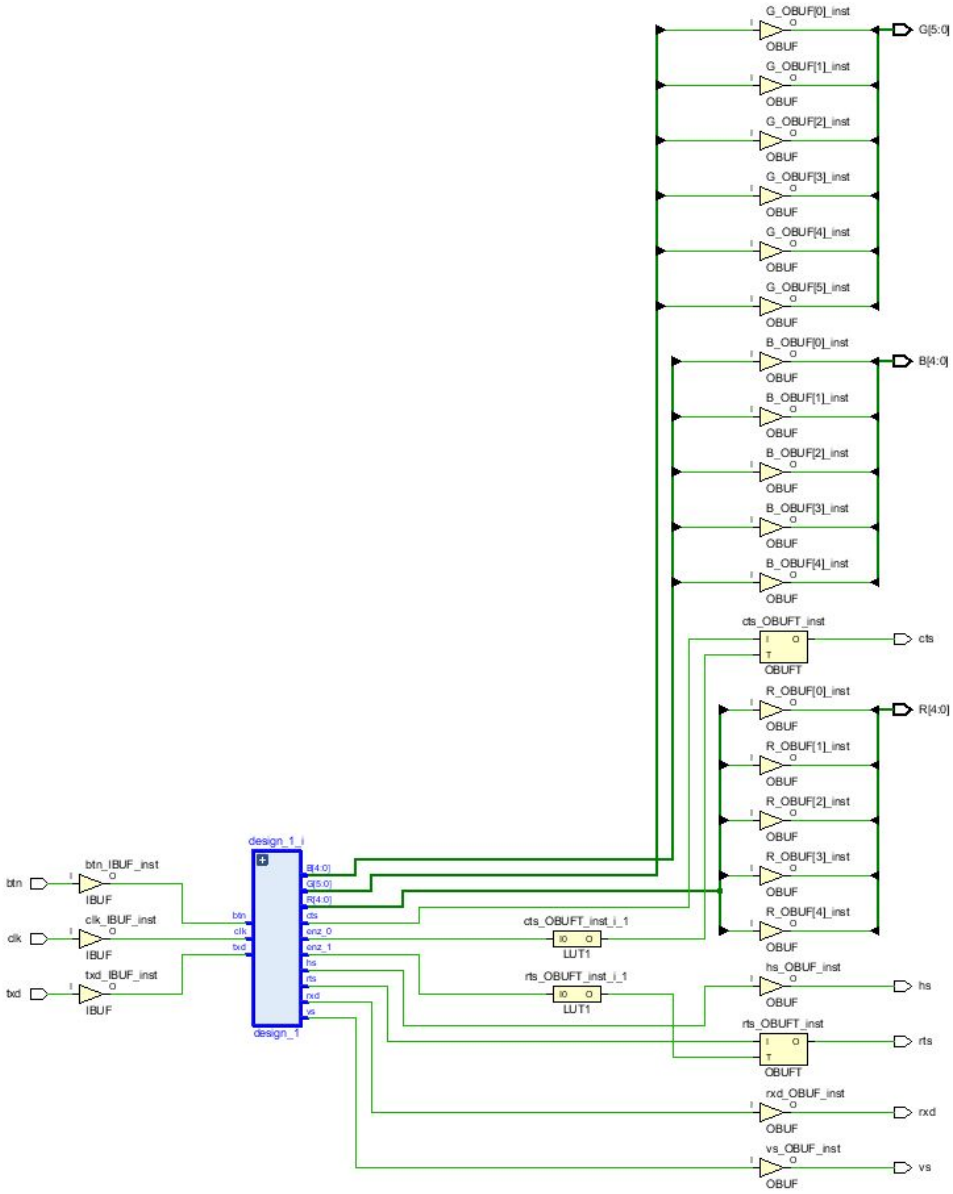

Figure 6: Synthesis Schematic with wrapper minimized

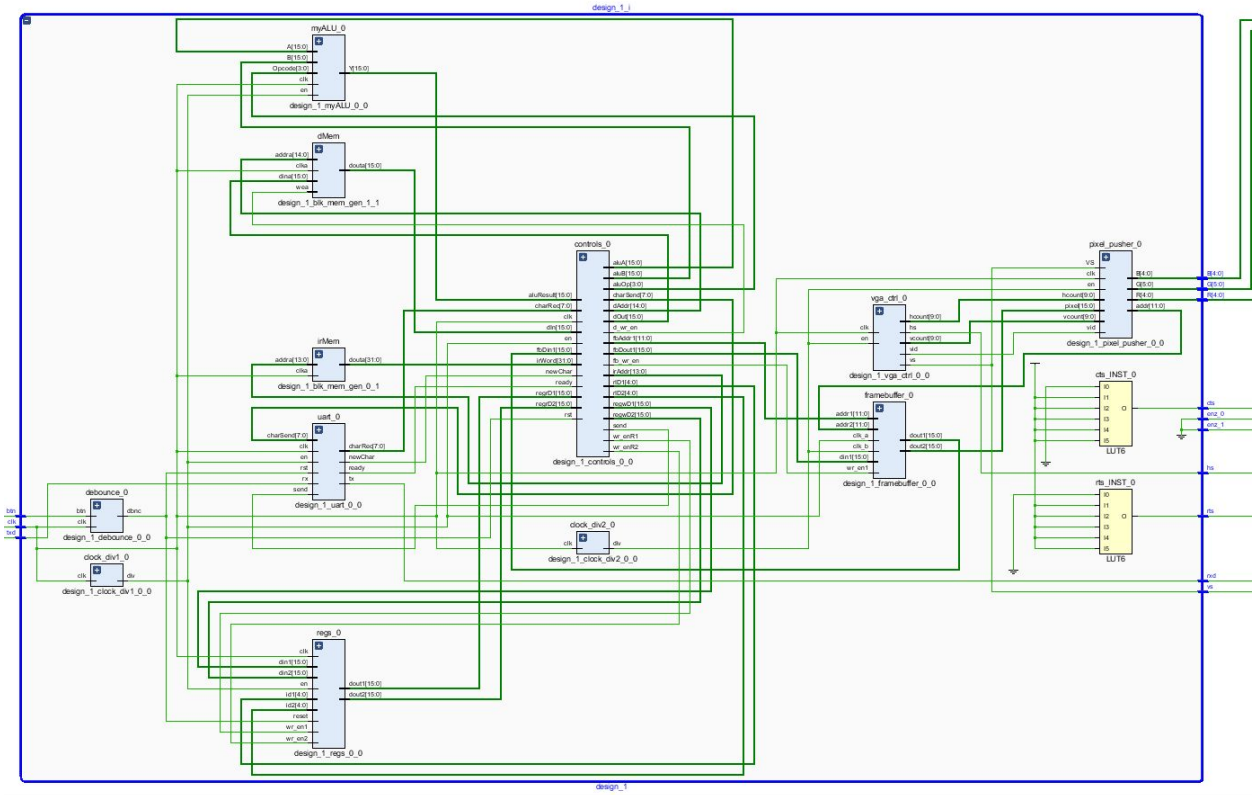

Figure 7: Synthesis Schematic of wrapper

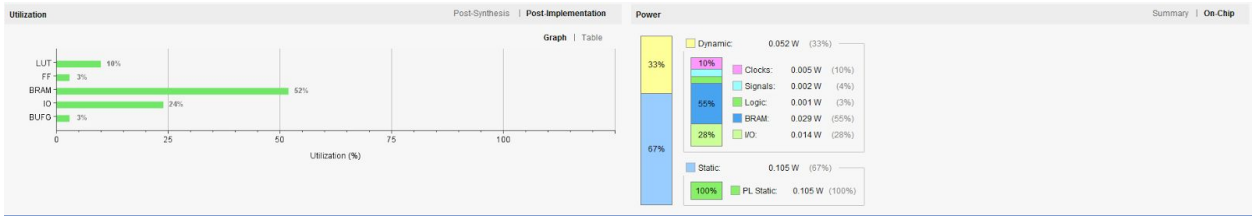

Figure 8: Power Graph and Utilization Graph

XDC File

ZYBO_Master.xdc

```
## This file is a general .xdc for the ZYBO Rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used signals according to the project


##Clock signal
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35
Sch=sysclk
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];


##Switches
#set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }]; #IO_L19N_T3_VREF_35
Sch=SW0
#set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];  #IO_L24P_T3_34 Sch=SW1
#set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L4N_T0_34 Sch=SW2
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L9P_T1_DQS_34 Sch=SW3


##Buttons
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { btn }]; #IO_L20N_T3_34 Sch=BTN0
#set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }]; #IO_L24N_T3_34 Sch=BTN1
##set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }]; #IO_L18P_T2_34 Sch=BTN2
##set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }]; #IO_L7P_T1_34 Sch=BTN3


##LEDs
#set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { led[0] }]; #IO_L23P_T3_35 Sch=LED0
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L23N_T3_35 Sch=LED1
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_0_35=Sch=LED2
#set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L3N_T0_DQS_AD1N_35
Sch=LED3


##I2S Audio Codec
#set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports ac_bclk]; #IO_L12N_T1_MRCC_35
Sch=AC_BCLK
#set_property -dict { PACKAGE_PIN T19   IOSTANDARD LVCMOS33 } [get_ports ac_mclk]; #IO_25_34 Sch=AC_MCLK
#set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports ac_muten]; #IO_L23N_T3_34
Sch=AC_MUTEN
#set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports ac_pbdat]; #IO_L8P_T1_AD10P_35
Sch=AC_PBDAT
#set_property -dict { PACKAGE_PIN L17   IOSTANDARD LVCMOS33 } [get_ports ac_pblrc]; #IO_L11N_T1_SRCC_35
Sch=AC_PBLRC
#set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 } [get_ports ac_recdat]; #IO_L12P_T1_MRCC_35
Sch=AC_RECDAT
#set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports ac_reclrc]; #IO_L8N_T1_AD10N_35
Sch=AC_RECLRC


##Audio Codec/external EEPROM IIC bus
#set_property -dict { PACKAGE_PIN N18   IOSTANDARD LVCMOS33 } [get_ports ac_scl]; #IO_L13P_T2_MRCC_34
Sch=AC_SCL
#set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports ac_sda]; #IO_L23P_T3_34 Sch=AC_SDA
```

```
##Additional Ethernet signals
#set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports eth_int_b]; #IO_L6P_T0_35
Sch=ETH_INT_B
#set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports eth_rst_b]; #IO_L3P_T0_DQS_AD1P_35
Sch=ETH_RST_B


##HDMI Signals
#set_property -dict { PACKAGE_PIN H17   IOSTANDARD TMDS_33 } [get_ports hdmi_clk_n]; #IO_L13N_T2_MRCC_35
Sch=HDMI_CLK_N
#set_property -dict { PACKAGE_PIN H16   IOSTANDARD TMDS_33 } [get_ports hdmi_clk_p]; #IO_L13P_T2_MRCC_35
Sch=HDMI_CLK_P
#set_property -dict { PACKAGE_PIN D20   IOSTANDARD TMDS_33 } [get_ports { hdmi_d_n[0] }]; #IO_L4N_T0_35
Sch=HDMI_D0_N
#set_property -dict { PACKAGE_PIN D19   IOSTANDARD TMDS_33 } [get_ports { hdmi_d_p[0] }]; #IO_L4P_T0_35
Sch=HDMI_D0_P
#set_property -dict { PACKAGE_PIN B20   IOSTANDARD TMDS_33 } [get_ports { hdmi_d_n[1] }]; #IO_L1N_T0_AD0N_35
Sch=HDMI_D1_N
#set_property -dict { PACKAGE_PIN C20   IOSTANDARD TMDS_33 } [get_ports { hdmi_d_p[1] }]; #IO_L1P_T0_AD0P_35
Sch=HDMI_D1_P
#set_property -dict { PACKAGE_PIN A20   IOSTANDARD TMDS_33 } [get_ports { hdmi_d_n[2] }]; #IO_L2N_T0_AD8N_35
Sch=HDMI_D2_N
#set_property -dict { PACKAGE_PIN B19   IOSTANDARD TMDS_33 } [get_ports { hdmi_d_p[2] }]; #IO_L2P_T0_AD8P_35
Sch=HDMI_D2_P
#set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports hdmi_cec]; #IO_L5N_T0_AD9N_35
Sch=HDMI_CEC
#set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 } [get_ports hdmi_hpd]; #IO_L5P_T0_AD9P_35
Sch=HDMI_HPD
#set_property -dict { PACKAGE_PIN F17   IOSTANDARD LVCMOS33 } [get_ports hdmi_out_en]; #IO_L6N_T0_VREF_35
Sch=HDMI_OUT_EN
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports hdmi_scl]; #IO_L16P_T2_35
Sch=HDMI_SCL
#set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports hdmi_sda]; #IO_L16N_T2_35
Sch=HDMI_SDA


##Pmod Header JA (XADC)
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { ja_p[0] }]; #IO_L21P_T3_DQS_AD14P_35
Sch=JA1_R_p
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { ja_p[1] }]; #IO_L22P_T3_AD7P_35
Sch=JA2_R_P
#set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { ja_p[2] }]; #IO_L24P_T3_AD15P_35
Sch=JA3_R_P
#set_property -dict { PACKAGE_PIN K14   IOSTANDARD LVCMOS33 } [get_ports { ja_p[3] }]; #IO_L20P_T3_AD6P_35
Sch=JA4_R_P
#set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { ja_n[0] }];
#IO_L21N_T3_DQS_AD14N_35 Sch=JA1_R_N
#set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS33 } [get_ports { ja_n[1] }]; #IO_L22N_T3_AD7N_35
Sch=JA2_R_N
#set_property -dict { PACKAGE_PIN J16   IOSTANDARD LVCMOS33 } [get_ports { ja_n[2] }]; #IO_L24N_T3_AD15N_35
Sch=JA3_R_N
#set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { ja_n[3] }]; #IO_L20N_T3_AD6N_35
Sch=JA4_R_N


##Pmod Header JB
set_property -dict { PACKAGE_PIN T20   IOSTANDARD LVCMOS33 } [get_ports { rts }]; #IO_L15P_T2_DQS_34 Sch=JB1_p
set_property -dict { PACKAGE_PIN U20   IOSTANDARD LVCMOS33 } [get_ports { rxd }]; #IO_L15N_T2_DQS_34 Sch=JB1_N
set_property -dict { PACKAGE_PIN V20   IOSTANDARD LVCMOS33 } [get_ports { txd }]; #IO_L16P_T2_34 Sch=JB2_P
set_property -dict { PACKAGE_PIN W20   IOSTANDARD LVCMOS33 } [get_ports { cts }]; #IO_L16N_T2_34 Sch=JB2_N
```

```
#set_property -dict { PACKAGE_PIN Y18   IOSTANDARD LVCMOS33 } [get_ports { jb_p[2] }]; #IO_L17P_T2_34 Sch=JB3_P
#set_property -dict { PACKAGE_PIN Y19   IOSTANDARD LVCMOS33 } [get_ports { jb_n[2] }]; #IO_L17N_T2_34 Sch=JB3_N
#set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports { jb_p[3] }]; #IO_L22P_T3_34 Sch=JB4_P
#set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports { jb_n[3] }]; #IO_L22N_T3_34 Sch=JB4_N


##Pmod Header JC
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { jc_p[0] }]; #IO_L10P_T1_34 Sch=JC1_P
#set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports { jc_n[0] }]; #IO_L10N_T1_34 Sch=JC1_N
#set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { jc_p[1] }]; #IO_L1P_T0_34 Sch=JC2_P
#set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { jc_n[1] }]; #IO_L1N_T0_34 Sch=JC2_N
#set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports { jc_p[2] }]; #IO_L8P_T1_34 Sch=JC3_P
#set_property -dict { PACKAGE_PIN Y14   IOSTANDARD LVCMOS33 } [get_ports { jc_n[2] }]; #IO_L8N_T1_34 Sch=JC3_N
#set_property -dict { PACKAGE_PIN T12   IOSTANDARD LVCMOS33 } [get_ports { jc_p[3] }]; #IO_L2P_T0_34 Sch=JC4_P
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { jc_n[3] }]; #IO_L2N_T0_34 Sch=JC4_N


##Pmod Header JD
#set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { jd_p[0] }]; #IO_L5P_T0_34 Sch=JD1_P
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { jd_n[0] }]; #IO_L5N_T0_34 Sch=JD1_N
#set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { jd_p[1] }]; #IO_L6P_T0_34 Sch=JD2_P
#set_property -dict { PACKAGE_PIN R14   IOSTANDARD LVCMOS33 } [get_ports { jd_n[1] }]; #IO_L6N_T0_VREF_34
Sch=JD2_N
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { jd_p[2] }]; #IO_L11P_T1_SRCC_34
Sch=JD3_P
#set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports { jd_n[2] }]; #IO_L11N_T1_SRCC_34
Sch=JD3_N
#set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { jd_p[3] }]; #IO_L21P_T3_DQS_34
Sch=JD4_P
#set_property -dict { PACKAGE_PIN V18   IOSTANDARD LVCMOS33 } [get_ports { jd_n[3] }]; #IO_L21N_T3_DQS_34
Sch=JD4_N


##Pmod Header JE
#set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { je[0] }]; #IO_L4P_T0_34 Sch=JE1
#set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports { je[1] }]; #IO_L18N_T2_34 Sch=JE2
#set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { je[2] }]; #IO_25_35 Sch=JE3
#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { je[3] }]; #IO_L19P_T3_35 Sch=JE4
#set_property -dict { PACKAGE_PIN V13   IOSTANDARD LVCMOS33 } [get_ports { je[4] }]; #IO_L3N_T0_DQS_34 Sch=JE7
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { je[5] }]; #IO_L9N_T1_DQS_34 Sch=JE8
#set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports { je[6] }]; #IO_L20P_T3_34 Sch=JE9
#set_property -dict { PACKAGE_PIN Y17   IOSTANDARD LVCMOS33 } [get_ports { je[7] }]; #IO_L7N_T1_34 Sch=JE10


##USB-OTG overcurrent detect pin
#set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports otg_oc]; #IO_L3P_T0_DQS_PUDC_B_34
Sch=OTG_OC


##VGA Connector
set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports { R[0] }]; #IO_L7P_T1_AD2P_35
Sch=VGA_R1
set_property -dict { PACKAGE_PIN L20   IOSTANDARD LVCMOS33 } [get_ports { R[1] }]; #IO_L9N_T1_DQS_AD3N_35
Sch=VGA_R2
set_property -dict { PACKAGE_PIN J20   IOSTANDARD LVCMOS33 } [get_ports { R[2] }]; #IO_L17P_T2_AD5P_35
Sch=VGA_R3
set_property -dict { PACKAGE_PIN G20   IOSTANDARD LVCMOS33 } [get_ports { R[3] }]; #IO_L18N_T2_AD13N_35
Sch=VGA_R4
set_property -dict { PACKAGE_PIN F19   IOSTANDARD LVCMOS33 } [get_ports { R[4] }]; #IO_L15P_T2_DQS_AD12P_35
Sch=VGA_R5
set_property -dict { PACKAGE_PIN H18   IOSTANDARD LVCMOS33 } [get_ports { G[0] }]; #IO_L14N_T2_AD4N_SRCC_35
```

```
Sch=VGA_G0
set_property -dict { PACKAGE_PIN N20   IOSTANDARD LVCMOS33 } [get_ports { G[1] }]; #IO_L14P_T2_SRCC_34
Sch=VGA_G1
set_property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS33 } [get_ports { G[2] }]; #IO_L9P_T1_DQS_AD3P_35
Sch=VGA_G2
set_property -dict { PACKAGE_PIN J19   IOSTANDARD LVCMOS33 } [get_ports { G[3] }]; #IO_L10N_T1_AD11N_35
Sch=VGA_G3
set_property -dict { PACKAGE_PIN H20   IOSTANDARD LVCMOS33 } [get_ports { G[4] }]; #IO_L17N_T2_AD5N_35
Sch=VGA_G4
set_property -dict { PACKAGE_PIN F20   IOSTANDARD LVCMOS33 } [get_ports { G[5] }]; #IO_L15N_T2_DQS_AD12N_35
Sch=VGA=G5
set_property -dict { PACKAGE_PIN P20   IOSTANDARD LVCMOS33 } [get_ports { B[0] }]; #IO_L14N_T2_SRCC_34
Sch=VGA_B1
set_property -dict { PACKAGE_PIN M20   IOSTANDARD LVCMOS33 } [get_ports { B[1] }]; #IO_L7N_T1_AD2N_35
Sch=VGA_B2
set_property -dict { PACKAGE_PIN K19   IOSTANDARD LVCMOS33 } [get_ports { B[2] }]; #IO_L10P_T1_AD11P_35
Sch=VGA_B3
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { B[3] }]; #IO_L14P_T2_AD4P_SRCC_35
Sch=VGA_B4
set_property -dict { PACKAGE_PIN G19   IOSTANDARD LVCMOS33 } [get_ports { B[4] }]; #IO_L18P_T2_AD13P_35
Sch=VGA_B5
set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS33 } [get_ports hs]; #IO_L13N_T2_MRCC_34 Sch=VGA_HS
set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports vs]; #IO_0_34 Sch=VGA_VS
```

The clock signal was uncommented to use the clock. The first button was uncommented since it was used to reset. The first four pins T20, U20, V20, and W20 for the PMOD connector JB were uncommented and were assigned to rts, rxd, txd, and cts respectively. The VGA connectors were also uncommented for the R, G, B, hs, and vs signals.

Discussion

Observations / Discoveries

This one lab taught me more than an entire semester of lab sessions in other classes. I learned how to infer memory and registers. I learned how to design a controller for a processor. I learned how to use the Vivado IP integrator tool. I learned how to debug simulations with a plethora of waveforms. I feel like the more appropriate question is what didn't I learn?

Questions / Follow Up

I completely understand the importance of simulating your code. I also understand the how to use/ realize the convenience of the IP integrator tool. I am a little unsure of why I had to break up certain states of my controller into multiple parts.