San Jose State University
Department of Electrical Engineering
EE178, Fall 2017, Crabill

# Laboratory Assignment #5

## Objectives

This lab is a continued introduction to the creation of basic constraints files with Xilinx Vivado 2016.2. You have already experienced the specification of pin placements and signaling standards, as well as timing specifications appropriate for designs fully synchronous to a single clock. This lab introduces the coarse placement of logic, also known as floorplanning, in the context of evaluating pipelining as an implementation trade-off.

No new logic design concepts are presented in this lab. The goal of this lab is for you to become more familiar with the tools. Please read carefully, pay attention, and take your time. This lab is not a race to see who gets done first.

In order to receive credit for this lab, you must complete the tasks and submit your project to the instructor.

## Bibliography

This lab draws heavily from documents on the Xilinx website http://www.xilinx.com. I would like to thank Xilinx for making this material available. This lab is effectively a customized introduction to Xilinx Vivado using Verilog-HDL and the Digilent Basys3.

## Baseline Sorting Design

Referring to the previous assignments if necessary, open Vivado and create a new project, but this time select the "xc7a35tcsg324-1" device during project creation. For the purposes of this lab, we need to use an FPGA with more I/O pins. After you have created a new project, create a new design source named bubblesort.v and replace the automatically generated contents of the file with the following:

```
// File: bubblesort.v
// This is the top level design for EE178 Lab #5a.

// The `timescale directive specifies what the
// simulation time units are (1 ns here) and what
// the simulator time step should be (1 ps here).

`timescale 1 ns / 1 ps

// Declare the module and its ports. This is
// using Verilog-2001 syntax.

module sort (
  input  wire clk,
  input  wire [15:0] in1, in2, in3, in4, in5,
  output reg  [15:0] out1, out2, out3, out4, out5
  );

  reg [15:0] dat1, dat2, dat3, dat4, dat5;

  always @(posedge clk)
```

```
  begin
    dat1 <= in1;
    dat2 <= in2;
    dat3 <= in3;
    dat4 <= in4;
    dat5 <= in5;
  end

  integer i, j;
  reg [15:0] temp;
  reg [15:0] array [1:5];

  always @*
  begin
    array[1] = dat1;
    array[2] = dat2;
    array[3] = dat3;
    array[4] = dat4;
    array[5] = dat5;
    for (i = 5; i > 0; i = i - 1)
    begin
      for (j = 1 ; j < i; j = j + 1)
      begin
        if (array[j] < array[j + 1])
        begin
          temp = array[j];
          array[j] = array[j + 1];
          array[j + 1] = temp;
        end
      end
    end
  end

  always @(posedge clk)
  begin
    out1 <= array[1];
    out2 <= array[2];
    out3 <= array[3];
    out4 <= array[4];
    out5 <= array[5];
  end

endmodule
```

This is a bubble sorter that will be discussed in class. After you have saved and closed the new design source, create a new constraint source named bubblesort.xdc. Replace any automatically generated contents of the file with the following:

```
# Constraints for ALL
set_property IOSTANDARD LVCMOS33 [get_ports *]

# Constraints for CLK
set_property PACKAGE_PIN E3 [get_ports clk]
create_clock -period 10.000 -name external_clock [get_ports clk]

# Constraints for IN*
set_property PACKAGE_PIN C2 [get_ports {in1[4]}]
```

```
set_property PACKAGE_PIN E5 [get_ports {in2[2]}]
set_property PACKAGE_PIN H5 [get_ports {in3[0]}]
set_property PACKAGE_PIN H2 [get_ports {in1[2]}]
set_property PACKAGE_PIN A3 [get_ports {in3[12]}]
set_property PACKAGE_PIN H4 [get_ports {in1[13]}]
set_property PACKAGE_PIN C6 [get_ports {in2[11]}]
set_property PACKAGE_PIN D3 [get_ports {in3[5]}]
set_property PACKAGE_PIN F4 [get_ports {in2[14]}]
set_property PACKAGE_PIN E7 [get_ports {in2[1]}]
set_property PACKAGE_PIN J5 [get_ports {in4[15]}]
set_property PACKAGE_PIN G2 [get_ports {in1[1]}]
set_property PACKAGE_PIN B1 [get_ports {in3[11]}]
set_property PACKAGE_PIN F1 [get_ports {in1[8]}]
set_property PACKAGE_PIN B6 [get_ports {in2[8]}]
set_property PACKAGE_PIN J3 [get_ports {in3[4]}]
set_property PACKAGE_PIN J4 [get_ports {in1[14]}]
set_property PACKAGE_PIN D7 [get_ports {in2[0]}]
set_property PACKAGE_PIN F3 [get_ports {in2[13]}]
set_property PACKAGE_PIN A1 [get_ports {in3[10]}]
set_property PACKAGE_PIN D2 [get_ports {in2[15]}]
set_property PACKAGE_PIN A5 [get_ports {in2[6]}]
set_property PACKAGE_PIN K2 [get_ports {in3[3]}]
set_property PACKAGE_PIN E2 [get_ports {in1[0]}]
set_property PACKAGE_PIN F6 [get_ports {in1[9]}]
set_property PACKAGE_PIN B3 [get_ports {in3[9]}]
set_property PACKAGE_PIN H1 [get_ports {in1[6]}]
set_property PACKAGE_PIN D8 [get_ports {in2[5]}]
set_property PACKAGE_PIN C1 [get_ports {in1[3]}]
set_property PACKAGE_PIN C4 [get_ports {in3[15]}]
set_property PACKAGE_PIN G4 [get_ports {in1[12]}]
set_property PACKAGE_PIN F5 [get_ports {in2[12]}]
set_property PACKAGE_PIN B2 [get_ports {in3[8]}]
set_property PACKAGE_PIN E6 [get_ports {in2[3]}]
set_property PACKAGE_PIN E1 [get_ports {in1[7]}]
set_property PACKAGE_PIN A6 [get_ports {in2[7]}]
set_property PACKAGE_PIN K1 [get_ports {in3[2]}]
set_property PACKAGE_PIN G6 [get_ports {in1[10]}]
set_property PACKAGE_PIN B4 [get_ports {in3[14]}]
set_property PACKAGE_PIN C5 [get_ports {in2[10]}]
set_property PACKAGE_PIN D5 [get_ports {in3[7]}]
set_property PACKAGE_PIN G1 [get_ports {in1[5]}]
set_property PACKAGE_PIN C7 [get_ports {in2[4]}]
set_property PACKAGE_PIN H6 [get_ports {in3[1]}]
set_property PACKAGE_PIN G3 [get_ports {in1[11]}]
set_property PACKAGE_PIN A4 [get_ports {in3[13]}]
set_property PACKAGE_PIN D4 [get_ports {in3[6]}]
set_property PACKAGE_PIN J2 [get_ports {in1[15]}]
set_property PACKAGE_PIN B7 [get_ports {in2[9]}]
set_property PACKAGE_PIN L5 [get_ports {in4[3]}]
set_property PACKAGE_PIN V2 [get_ports {in5[13]}]
set_property PACKAGE_PIN K6 [get_ports {in4[8]}]
set_property PACKAGE_PIN N5 [get_ports {in5[4]}]
set_property PACKAGE_PIN P3 [get_ports {in5[3]}]
set_property PACKAGE_PIN U2 [get_ports {in5[12]}]
set_property PACKAGE_PIN M2 [get_ports {in4[7]}]
set_property PACKAGE_PIN L1 [get_ports {in4[10]}]
set_property PACKAGE_PIN L6 [get_ports {in4[2]}]
```

```
set_property PACKAGE_PIN P2 [get_ports {in5[0]}]
set_property PACKAGE_PIN N2 [get_ports {in4[14]}]
set_property PACKAGE_PIN P4 [get_ports {in5[2]}]
set_property PACKAGE_PIN T3 [get_ports {in5[9]}]
set_property PACKAGE_PIN M3 [get_ports {in4[6]}]
set_property PACKAGE_PIN T5 [get_ports {in5[6]}]
set_property PACKAGE_PIN V1 [get_ports {in4[1]}]
set_property PACKAGE_PIN N1 [get_ports {in4[13]}]
set_property PACKAGE_PIN V5 [get_ports {in5[10]}]
set_property PACKAGE_PIN M1 [get_ports {in4[9]}]
set_property PACKAGE_PIN T4 [get_ports {in5[7]}]
set_property PACKAGE_PIN U1 [get_ports {in4[0]}]
set_property PACKAGE_PIN K3 [get_ports {in4[12]}]
set_property PACKAGE_PIN R2 [get_ports {in5[1]}]
set_property PACKAGE_PIN L4 [get_ports {in4[5]}]
set_property PACKAGE_PIN P5 [get_ports {in5[5]}]
set_property PACKAGE_PIN U3 [get_ports {in5[15]}]
set_property PACKAGE_PIN L3 [get_ports {in4[11]}]
set_property PACKAGE_PIN V4 [get_ports {in5[11]}]
set_property PACKAGE_PIN K5 [get_ports {in4[4]}]
set_property PACKAGE_PIN U4 [get_ports {in5[14]}]
set_property PACKAGE_PIN R3 [get_ports {in5[8]}]

# Constraints for OUT*
set_property PACKAGE_PIN G13 [get_ports {out2[10]}]
set_property PACKAGE_PIN H17 [get_ports {out3[11]}]
set_property PACKAGE_PIN E16 [get_ports {out1[15]}]
set_property PACKAGE_PIN H16 [get_ports {out2[5]}]
set_property PACKAGE_PIN C17 [get_ports {out3[8]}]
set_property PACKAGE_PIN G16 [get_ports {out2[6]}]
set_property PACKAGE_PIN F16 [get_ports {out2[4]}]
set_property PACKAGE_PIN C12 [get_ports {out1[0]}]
set_property PACKAGE_PIN J14 [get_ports {out3[9]}]
set_property PACKAGE_PIN K16 [get_ports {out4[14]}]
set_property PACKAGE_PIN E17 [get_ports {out3[15]}]
set_property PACKAGE_PIN G18 [get_ports {out3[3]}]
set_property PACKAGE_PIN G17 [get_ports {out3[12]}]
set_property PACKAGE_PIN F13 [get_ports {out1[4]}]
set_property PACKAGE_PIN K15 [get_ports {out4[15]}]
set_property PACKAGE_PIN D14 [get_ports {out2[12]}]
set_property PACKAGE_PIN B17 [get_ports {out1[7]}]
set_property PACKAGE_PIN F18 [get_ports {out3[4]}]
set_property PACKAGE_PIN D13 [get_ports {out1[5]}]
set_property PACKAGE_PIN C14 [get_ports {out2[11]}]
set_property PACKAGE_PIN B18 [get_ports {out1[14]}]
set_property PACKAGE_PIN B12 [get_ports {out2[15]}]
set_property PACKAGE_PIN H15 [get_ports {out3[10]}]
set_property PACKAGE_PIN J13 [get_ports {out3[14]}]
set_property PACKAGE_PIN B14 [get_ports {out2[13]}]
set_property PACKAGE_PIN A15 [get_ports {out1[10]}]
set_property PACKAGE_PIN E18 [get_ports {out3[5]}]
set_property PACKAGE_PIN D12 [get_ports {out1[6]}]
set_property PACKAGE_PIN A13 [get_ports {out1[12]}]
set_property PACKAGE_PIN E15 [get_ports {out2[9]}]
set_property PACKAGE_PIN A11 [get_ports {out1[1]}]
set_property PACKAGE_PIN D17 [get_ports {out2[0]}]
set_property PACKAGE_PIN B11 [get_ports {out1[2]}]
```

```
set_property PACKAGE_PIN J15 [get_ports {out3[0]}]
set_property PACKAGE_PIN F14 [get_ports {out1[3]}]
set_property PACKAGE_PIN B16 [get_ports {out1[8]}]
set_property PACKAGE_PIN B13 [get_ports {out2[14]}]
set_property PACKAGE_PIN A16 [get_ports {out1[9]}]
set_property PACKAGE_PIN D18 [get_ports {out3[6]}]
set_property PACKAGE_PIN G14 [get_ports {out2[2]}]
set_property PACKAGE_PIN K13 [get_ports {out3[13]}]
set_property PACKAGE_PIN J17 [get_ports {out3[1]}]
set_property PACKAGE_PIN D15 [get_ports {out2[7]}]
set_property PACKAGE_PIN H14 [get_ports {out2[1]}]
set_property PACKAGE_PIN A14 [get_ports {out1[11]}]
set_property PACKAGE_PIN C16 [get_ports {out3[7]}]
set_property PACKAGE_PIN F15 [get_ports {out2[3]}]
set_property PACKAGE_PIN A18 [get_ports {out1[13]}]
set_property PACKAGE_PIN C15 [get_ports {out2[8]}]
set_property PACKAGE_PIN J18 [get_ports {out3[2]}]
set_property PACKAGE_PIN R17 [get_ports {out5[6]}]
set_property PACKAGE_PIN R11 [get_ports {out4[7]}]
set_property PACKAGE_PIN P18 [get_ports {out5[12]}]
set_property PACKAGE_PIN L15 [get_ports {out4[13]}]
set_property PACKAGE_PIN T14 [get_ports {out5[1]}]
set_property PACKAGE_PIN M13 [get_ports {out4[2]}]
set_property PACKAGE_PIN N15 [get_ports {out5[7]}]
set_property PACKAGE_PIN N14 [get_ports {out5[13]}]
set_property PACKAGE_PIN T15 [get_ports {out5[2]}]
set_property PACKAGE_PIN N16 [get_ports {out5[8]}]
set_property PACKAGE_PIN K18 [get_ports {out4[8]}]
set_property PACKAGE_PIN P14 [get_ports {out5[14]}]
set_property PACKAGE_PIN R12 [get_ports {out4[3]}]
set_property PACKAGE_PIN M16 [get_ports {out5[9]}]
set_property PACKAGE_PIN K17 [get_ports {out4[9]}]
set_property PACKAGE_PIN R18 [get_ports {out5[15]}]
set_property PACKAGE_PIN P15 [get_ports {out5[3]}]
set_property PACKAGE_PIN R13 [get_ports {out4[4]}]
set_property PACKAGE_PIN M17 [get_ports {out5[10]}]
set_property PACKAGE_PIN M14 [get_ports {out4[10]}]
set_property PACKAGE_PIN R15 [get_ports {out5[4]}]
set_property PACKAGE_PIN L18 [get_ports {out4[5]}]
set_property PACKAGE_PIN L14 [get_ports {out4[11]}]
set_property PACKAGE_PIN T18 [get_ports {out4[0]}]
set_property PACKAGE_PIN P17 [get_ports {out5[5]}]
set_property PACKAGE_PIN M18 [get_ports {out4[6]}]
set_property PACKAGE_PIN N17 [get_ports {out5[11]}]
set_property PACKAGE_PIN L16 [get_ports {out4[12]}]
set_property PACKAGE_PIN T16 [get_ports {out5[0]}]
set_property PACKAGE_PIN L13 [get_ports {out4[1]}]

# Constraints for CFGBVS
set_property CFGBVS VCCO [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
```

The provided constraints create a 100 MHz clock and, roughly, place all inputs on one side of the device and place all outputs on the other side of the device. After you have saved and closed the new constraint source, create a new simulation source named testbench.v. Replace any automatically generated contents of the file with the following:

```
// File:  testbench.v
// This is the top level testbench for EE178 Lab #5.

// The `timescale directive specifies what the
// simulation time units are (1 ns here) and what
// the simulator time step should be (1 ps here).

`timescale 1 ns / 1 ps

// Declare the module and its ports. This is
// using Verilog-2001 syntax.

module testbench;

  // Generate a free running 100 MHz clock
  // signal to mimic what is on the board
  // provided for prototyping.

  reg clk;

  always
  begin
    clk = 1'b0;
    #5;
    clk = 1'b1;
    #5;
  end

  reg [15:0] in1, in2, in3, in4, in5;

  always @(negedge clk)
  begin
    in1 <= $random;
    in2 <= $random;
    in3 <= $random;
    in4 <= $random;
    in5 <= $random;
  end

  initial
  begin
    $display("If simulation ends before the testbench");
    $display("completes, use the menu option to run all.");
    #400;  // allow it to run
    $display("Simulation is over, check the waveforms.");
    $stop;
  end

  wire [15:0] out1, out2, out3, out4, out5;

  sort my_sort (
    .in1(in1),
    .in2(in2),
    .in3(in3),
    .in4(in4),
    .in5(in5),
```

```
    .out1(out1),
    .out2(out2),
    .out3(out3),
    .out4(out4),
    .out5(out5),
    .clk(clk)
  );

endmodule
```

The provided testbench is very simple. It generates a clock for the sorter and every cycle applies random values to the sorter data inputs. There is no checking of the output results. After you have saved and closed the new simulation source, proceed to evaluate the design.

## Evaluating the Baseline Design

Perform a behavioral simulation of the design. Using the waveform viewer, zoom in to an appropriate level and evaluate the relationship of the input values to the output values. For inputs applied on a given cycle, you should be able to identify the corresponding outputs from the circuit. Although the testbench is far from a comprehensive test of the design behavior, does the design appear to correctly sort the input data?

Screen capture a portion of the waveform and proceed to annotate it. Pick an input data set, and draw a vertical bar at the positive edge of the clock where the input data is sampled by the input flip flops of this sorting design. Then, identify the corresponding outputs from the circuit. Draw another vertical bar at the positive edge of the clock where the output data is presented by the output flip flops of this sorting design. On the image, write the design latency, in clock cycles.

Close the simulator, implement the design, and open the implemented design. Expect the implementation will fail to meet the 100 MHz clock constraint. Generate a timing summary and review the results. Knowing that a 100 MHz clock constraint corresponds to a 10 ns clock period, use the reported WNS to estimate a clock period that is more reasonable for this implementation. You are “hunting” for the best performance you can achieve. Close the implemented design, edit the clock constraint in the constraint file to reflect your estimate, and re-implement the design.

Depending on your estimate, the implementation may or may not meet your revised clock constraint. In any event, it should be close – meaning, the reported WNS, whether positive or negative, should be far closer to zero than it was in the first implementation. Revise your clock constraint a final time and re-implement the design. Take a moment to visually inspect the logic placement shown in the device view of the final implemented design.

On the waveform image from simulation, write your achieved minimum clock period and the corresponding maximum clock frequency. Using the design latency, in clock cycles, and the minimum clock period, in nanoseconds per cycle, compute the design latency in time and write your result on the waveform image.

Finally, generate a utilization report for the design. Identify how many look-up tables (LUTs) and flip flops (FDs) are consumed by the design. Write the LUT and FD counts on the waveform image. After you have obtained this information and recorded it, you have completed the evaluation of the baseline design. Close the project completely, so that Vivado is ready to create a new project.

## Revised Sorting Design

Created a new project, again select the “xc7a35tcsg324-1” device during project creation. Create a new design source named oetsort.v and replace the automatically generated contents of the file with the following:

```
// File: oetsort.v
// This is the top level design for EE178 Lab #5b.

// The `timescale directive specifies what the
// simulation time units are (1 ns here) and what
// the simulator time step should be (1 ps here).

`timescale 1 ns / 1 ps

// Declare the module and its ports. This is
// using Verilog-2001 syntax.

module sort (
  input  wire clk,
  input  wire [15:0] in1, in2, in3, in4, in5,
  output wire [15:0] out1, out2, out3, out4, out5
  );

  wire [15:0] cnx0to1 [1:5];
  wire [15:0] cnx1to2 [1:5];
  wire [15:0] cnx2to3 [1:5];
  wire [15:0] cnx3to4 [1:5];
  wire [15:0] cnx4to5 [1:5];

  // input stage
  pass_thru   #(.REGISTERED(1)) s0_1
(.clk(clk),.pin(in1),.pout(cnx0to1[1]));
  pass_thru   #(.REGISTERED(1)) s0_2
(.clk(clk),.pin(in2),.pout(cnx0to1[2]));
  pass_thru   #(.REGISTERED(1)) s0_3
(.clk(clk),.pin(in3),.pout(cnx0to1[3]));
  pass_thru   #(.REGISTERED(1)) s0_4
(.clk(clk),.pin(in4),.pout(cnx0to1[4]));
  pass_thru   #(.REGISTERED(1)) s0_5
(.clk(clk),.pin(in5),.pout(cnx0to1[5]));

  // first stage
  cmp_and_swp #(.REGISTERED(1)) s1_1n2
(.clk(clk),.xin(cnx0to1[1]),.yin(cnx0to1[2]),.xout(cnx1to2[1]),.yout(cnx
1to2[2]));
  cmp_and_swp #(.REGISTERED(1)) s1_3n4
(.clk(clk),.xin(cnx0to1[3]),.yin(cnx0to1[4]),.xout(cnx1to2[3]),.yout(cnx
1to2[4]));
  pass_thru   #(.REGISTERED(1)) s1_5
(.clk(clk),.pin(cnx0to1[5]),.pout(cnx1to2[5]));

  // second stage
  pass_thru   #(.REGISTERED(1)) s2_1
(.clk(clk),.pin(cnx1to2[1]),.pout(cnx2to3[1]));
  cmp_and_swp #(.REGISTERED(1)) s2_2n3
(.clk(clk),.xin(cnx1to2[2]),.yin(cnx1to2[3]),.xout(cnx2to3[2]),.yout(cnx
2to3[3]));
  cmp_and_swp #(.REGISTERED(1)) s2_4n5
(.clk(clk),.xin(cnx1to2[4]),.yin(cnx1to2[5]),.xout(cnx2to3[4]),.yout(cnx
2to3[5]));
```

```
  // third stage
  cmp_and_swp #(.REGISTERED(1)) s3_1n2
(.clk(clk),.xin(cnx2to3[1]),.yin(cnx2to3[2]),.xout(cnx3to4[1]),.yout(cnx
3to4[2]));
  cmp_and_swp #(.REGISTERED(1)) s3_3n4
(.clk(clk),.xin(cnx2to3[3]),.yin(cnx2to3[4]),.xout(cnx3to4[3]),.yout(cnx
3to4[4]));
  pass_thru   #(.REGISTERED(1)) s3_5
(.clk(clk),.pin(cnx2to3[5]),.pout(cnx3to4[5]));

  // fourth stage
  pass_thru   #(.REGISTERED(1)) s4_1
(.clk(clk),.pin(cnx3to4[1]),.pout(cnx4to5[1]));
  cmp_and_swp #(.REGISTERED(1)) s4_2n3
(.clk(clk),.xin(cnx3to4[2]),.yin(cnx3to4[3]),.xout(cnx4to5[2]),.yout(cnx
4to5[3]));
  cmp_and_swp #(.REGISTERED(1)) s4_4n5
(.clk(clk),.xin(cnx3to4[4]),.yin(cnx3to4[5]),.xout(cnx4to5[4]),.yout(cnx
4to5[5]));

  // fifth stage
  cmp_and_swp #(.REGISTERED(1)) s5_1n2
(.clk(clk),.xin(cnx4to5[1]),.yin(cnx4to5[2]),.xout(out1),.yout(out2));
  cmp_and_swp #(.REGISTERED(1)) s5_3n4
(.clk(clk),.xin(cnx4to5[3]),.yin(cnx4to5[4]),.xout(out3),.yout(out4));
  pass_thru   #(.REGISTERED(1)) s5_5
(.clk(clk),.pin(cnx4to5[5]),.pout(out5));

endmodule


module pass_thru #(parameter REGISTERED = 0) (
  input  wire clk,
  input  wire [15:0] pin,
  output reg  [15:0] pout
  );

  generate
  begin
    if (REGISTERED)
    begin
      always @(posedge clk) pout <= pin;
    end
    else
    begin
      always @* pout = pin;
    end
  end
  endgenerate

endmodule


module cmp_and_swp #(parameter REGISTERED = 0) (
  input  wire clk,
  input  wire [15:0] xin, yin,
  output reg  [15:0] xout, yout
```

```
  );

  generate
  begin
    if (REGISTERED)
    begin
      always @(posedge clk)
      begin
        if (xin < yin)
        begin
          xout <= yin;
          yout <= xin;
        end
        else
        begin
          xout <= xin;
          yout <= yin;
        end
      end
    end
    else
    begin
      always @*
      begin
        if (xin < yin)
        begin
          xout = yin;
          yout = xin;
        end
        else
        begin
          xout = xin;
          yout = yin;
        end
      end
    end
  end
  endgenerate

endmodule
```

This is an odd-even transposition sorter that will be discussed in class, and there are several modules described in one file (generally not recommended, but it cuts down on the number of times you need to step through the new design source creation process). Take a moment to review the code, as it provides an example of how to declare a module parameter, use that parameter in conjunction with the generate statement, and then override the parameter at the place of module instantiations.

After you have saved and closed the new design source, create a new constraint source named oetsort.xdc. Replace any automatically generated contents of the file with the same constraints provided for the bubble sorter.

After you have saved and closed the new constraint source, create a new simulation source named testbench.v. Replace any automatically generated contents of the file with the same testbench provided for the bubble sorter. After you have saved and closed the new simulation source, proceed to evaluate the design.

## Evaluating the Revised Design

Perform a behavioral simulation of the design. Using the waveform viewer, zoom in to an appropriate level and evaluate the relationship of the input values to the output values. For inputs applied on a given cycle, you should be able to identify the corresponding outputs from the circuit. Although the testbench is far from a comprehensive test of the design behavior, does the design appear to correctly sort the input data?

Screen capture a portion of the waveform and proceed to annotate it. Pick an input data set, and draw a vertical bar at the positive edge of the clock where the input data is sampled by the input flip flops of this sorting design. Then, identify the corresponding outputs from the circuit. Draw another vertical bar at the positive edge of the clock where the output data is presented by the output flip flops of this sorting design. On the image, write the design latency, in clock cycles.

Close the simulator, implement the design, and open the implemented design. Expect the implementation will meet the 100 MHz clock constraint. Generate a timing summary and review the results. Knowing that a 100 MHz clock constraint corresponds to a 10 ns clock period, use the reported WNS to estimate a clock period that is pushing the implementation to its performance limit. Close the implemented design, edit the clock constraint in the constraint file to reflect your estimate, and re-implement the design.

Depending on your estimate, the implementation may or may not meet your revised clock constraint. In any event, it should be close – meaning, the reported WNS, whether positive or negative, should be far closer to zero than it was in the first implementation. Revise your clock constraint a final time and re-implement the design. Take a moment to visually inspect the logic placement shown in the device view of the final implemented design.

On the waveform image from simulation, write your achieved minimum clock period and the corresponding maximum clock frequency. Using the design latency, in clock cycles, and the minimum clock period, in nanoseconds per cycle, compute the design latency in time and write your result on the waveform image.

Finally, generate a utilization report for the design. Identify how many look-up tables (LUTs) and flip flops (FDs) are consumed by the design. Write the LUT and FD counts on the waveform image. After you have obtained this information and recorded it, close the implemented design.

## Coarse Placement of Logic (Floorplanning)

The automatic logic placement performed during implementation is fast and generates a reasonable result. There are often reasons, however, to provide placement constraints. Placement constraints may yield a higher performance design, increased timing closure repeatability, or enable advanced design techniques such as modular design and partial reconfiguration.

Placement constraints may be coarse, used to assign portions of a design into a region of the device. They may also be very fine, used to assign an individual element of a design to a specific location on the device. In this portion of the assignment, we will learn how to perform the coarse placement of logic.

To begin, select Open Elaborated Design, under RTL Analysis in the Flow Navigator. You will recognize this from the pin placement exercise in a previous assignment. Once the elaborated design is open, select the Floorplanning layout, using the selection box on the toolbar as shown in Figure 1.

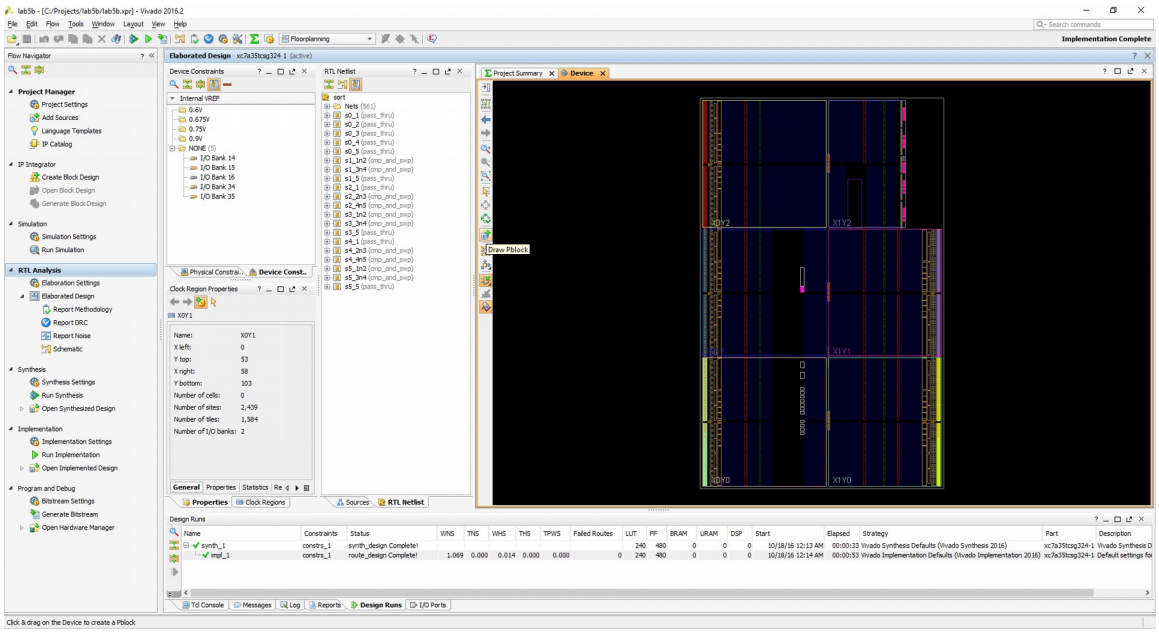

**Figure 1: Open Elaborated Design, Floorplanning Layout**

The device view shown in Figure 1 represents a top view of the resources and their physical arrangement on the device. At this initial level of zoom, it is difficult to identify individual resources. Zoom in around the center of the device, until you can clearly distinguish small gray squares. The small gray squares represent logic slices which contain LUTs and FDs. In this FPGA, logic slices come in pairs, and you will see them grouped in pairs by dark blue rectangles.

Locate the "Draw Pblock" button on the device view toolbar. When you click this button, you may then outline a rectangular region on the device and give the region a name. While you are outlining the region, the number of resources in the region are displayed. Referring to Figure 2, draw a pblock to outline a 2x25 array of slices. Vivado has a strong preference that pblock boundaries should not divide logic slice pairs, meaning that you should position the 2x25 pblock boundary to align with the grid of dark blue rectangles.

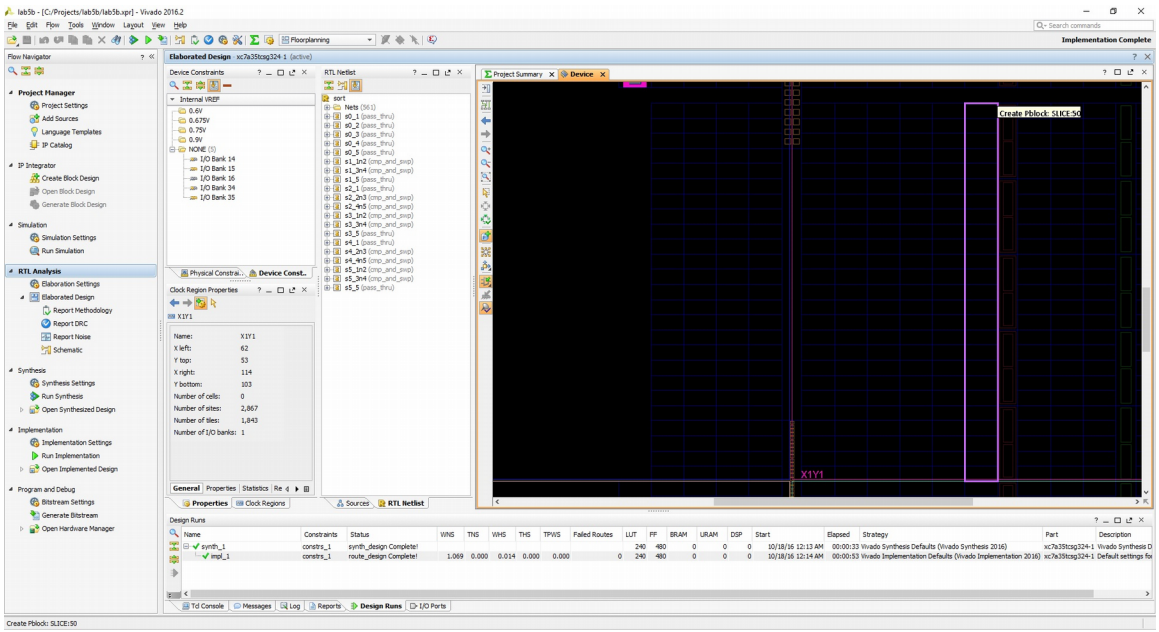

**Figure 2: Drawing a Box to Define a Pblock**

After you draw the pblock, Vivado will prompt you to give the pblock a name, as shown in Figure 3. Name the pblock you have created "pblock_5". This dialog box also serves to confirm what resources are bounded by the pblock. In this example, we have only outlined logic slices, but other resources exist in the device and may be included when drawing a pblock. The checkboxes in the dialog box enable you to remove specific resource types from the pblock if you wish.

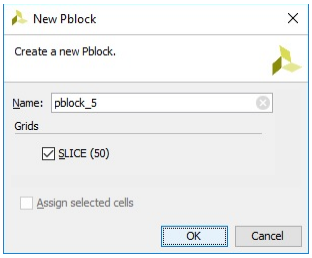


**Figure 3: Naming a Pblock and Selecting Resources**

At this point, you should see "pblock_5" in the device view. If you have made an error in the definition of a pblock, you can select its outline in the device view and delete it. Proceed to add five more pblocks of the same size, from right to left, naming them "pblock_4" down to "pblock_0". When you have completed this, you should have a result similar to that shown in Figure 4.

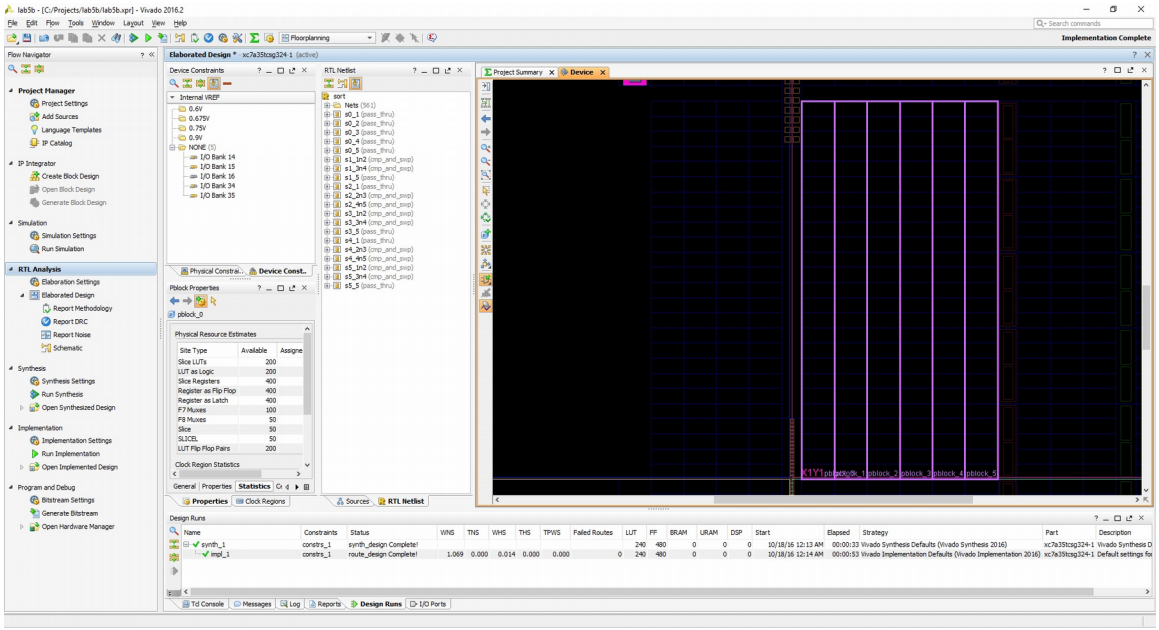

**Figure 4: Multiple Pblocks Defined**

The next step is to assign design objects to each pblock. Unlike the bubble sorter design, the odd-even transposition sorter design has hierarchy (resulting from instantiation of sub-modules, each of which has a unique name) making it easy to identify pieces of the design. The RTL Netlist window shown in Figure 4 contains a tree representation of the design hierarchy. The tree, in its unexpanded state, has expandable branches for the nets (wires) and sub-modules that exist at the top level of the design.

Notice how the sub-modules have names that begin with "s0_" through "s5_". This is not accidental or random – these names are the sub-module instance names chosen in the design source, and they were selected to indicate their relative position in the pipeline stage sequence.

As shown in Figure 5, start assigning the sub-modules to pblocks. You can do this by dragging and dropping. Simply select one or more sub-modules in the RTL Netlist and drag the selection onto a pblock in the Device View. Begin by assigning all sub-modules that begin with "s0_" to pblock_0.

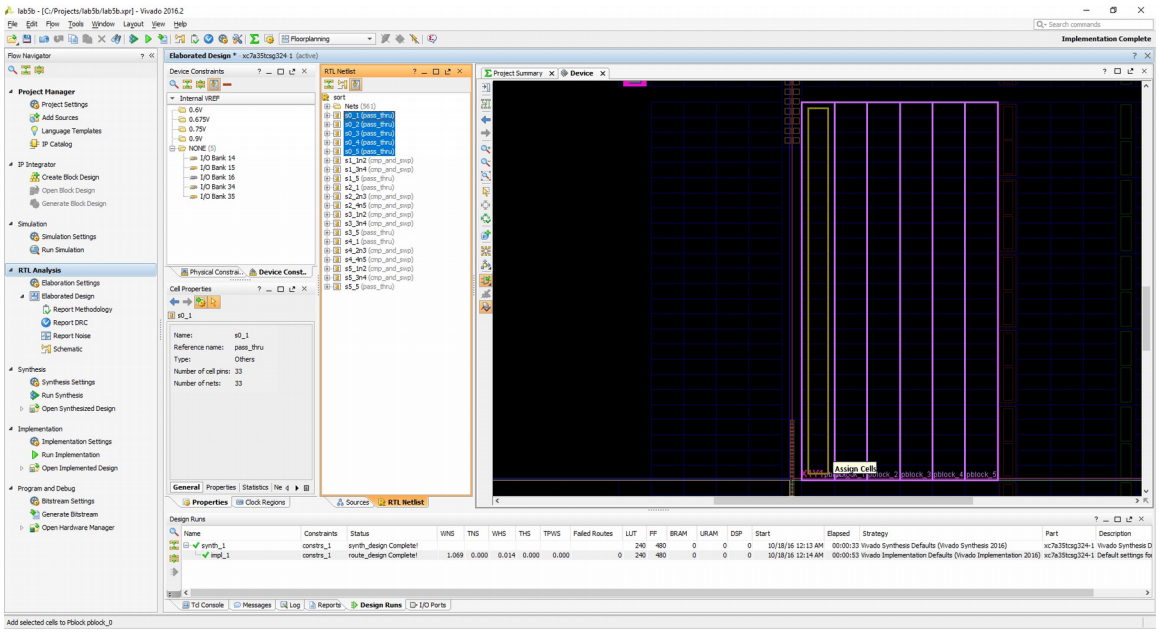

**Figure 5: Assigning Design Objects to a Pblock**

Then, assign all sub-modules that begin with "s1_" to pblock_1. Continue in this manner until all sub-modules have been assigned to their corresponding pblock. When you have completed this, you have established a floorplan for your design.

When you are done, click on the Save Constraints button in the toolbar, also shown in Figure 5. A dialog box will appear, asking if you want to save the constraints to a new XDC file, or to an existing XDC file in the project. Save the constraints to the oetsort.xdc file that's already in the project.

At this time, close the elaborated design by clicking on the close button (a black X) shown in Figure 5, at the very right edge of the light blue "Elaborated Design" title bar.

Open the oetsort.xdc constraint file in the text editor, simply to review the contents. You originally added a file to which you copied a clock constraint and pin placement constraints; now it is populated with the logic placement constraints you applied and should look similar to Figure 6. Close the file without editing it.

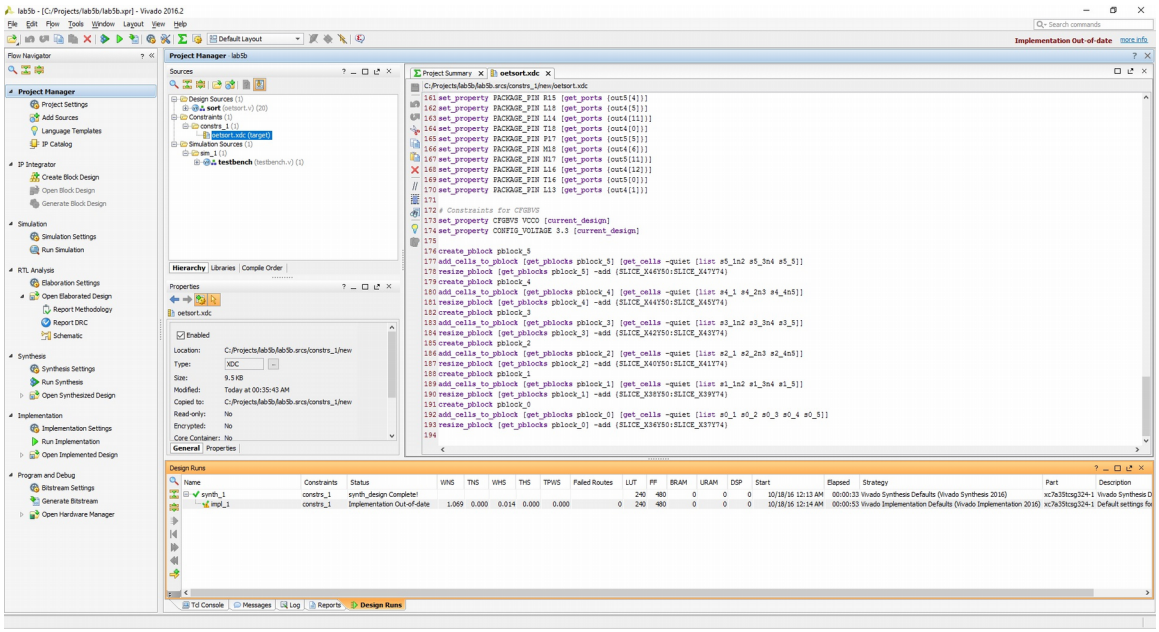

**Figure 6: Reviewing Generated Pblock Constraints in XDC**

In the bottom window, select Design Runs; right click on the synthesis run. The context menu provides an option to Reset Runs. Reset the synthesis run. Resetting the synthesis run allows you to re-run it from scratch. As you have seen in a previous assignment, direct Vivado to remove all the previous results to start with a clean slate.

Now re-run synthesis and implementation, and then open the implemented design so that you can report a timing summary. From the timing summary, expand the timing check category tree and select the path with the least slack in the intra-clock setup check. This path may or may not be failing – it doesn't matter. What we want to observe is the timing path in the Device View, which should be similar to what is shown in Figure 7. You can toggle the show "Routing Resources" button to show or hide the routes.

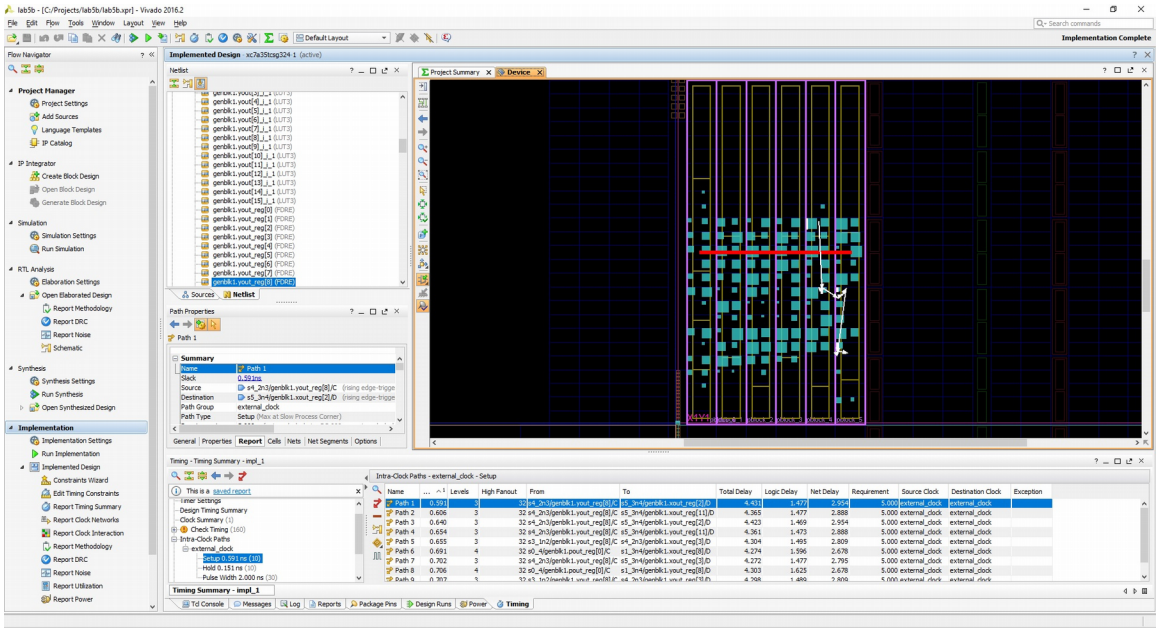

**Figure 7: Review Timing Paths in Floorplanned Design**

Although your results may vary from those shown, you will have achieved new timing results that differ from your earlier exploration of the odd-even transposition sorter. In my own exploration, the floorplan in Figure 5 yielded a maximum clock frequency that was actually lower than the results I achieved with no floorplan at all. On the waveform image from simulation, write your achieved minimum clock period and the corresponding maximum clock frequency after the application of a floorplan.

In the Device View, click on one of the pblocks so that it is selected. Then, look at the properties window below the netlist window. If a pblock is selected, one of the tabs available at the bottom of the properties window is the Statistics tab. In the Statistics tab, you can discover the pblock utilization – that is, of the resources you allocated to the pblock, how many of them were actually used.

If you were going to iterate on this floorplan, the pblock utilization suggests the pblock size could be reduced substantially. However, by studying the failing timing paths, there is a trend that suggests an improvement by changing what logic is put into each pblock.

## Laboratory Hand-In Requirements

Once you have completed this exercise, prepare for the submission process. Within four hours of the scheduled class time on the due date, you are required to submit your entire project directory in the form of a compressed ZIP archive. You must include the screen captures showing the requested annotations. Use WinZIP to archive the entire project directory, and name the archive lab5_yourlastname_yourfirstname.zip. For example, if I were to make a submission, it would be lab5_crabill_eric.zip. Then email the archive to the instructor. Only WinZIP archives will be accepted.