MiniZed: Creating a Zynq Hardware Platform in Vivado

1. Download the MiniZed board definition files from the following link:

- https://github.com/Avnet/bdf

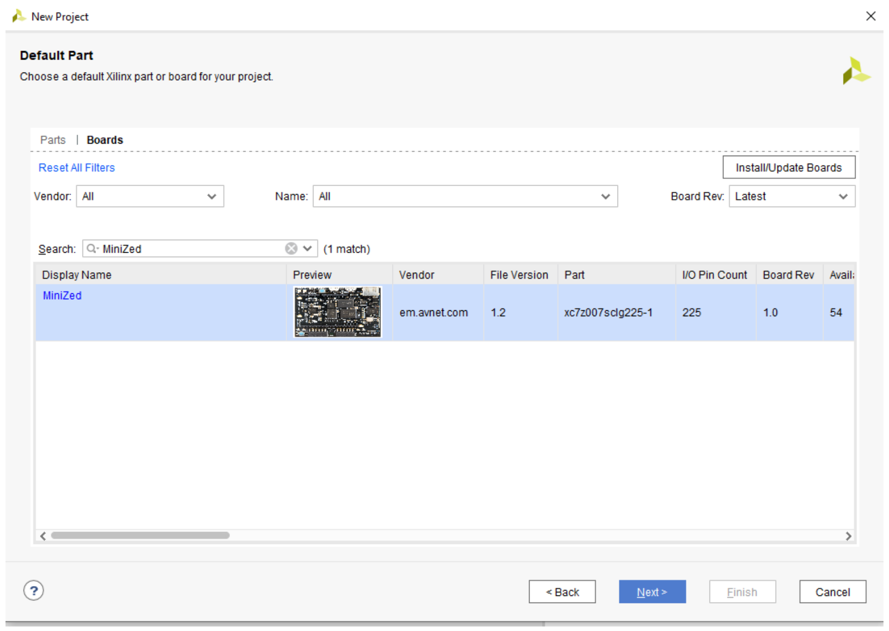
2. Follow the instructions at the bottom of the page to download them to the correct location.

3. Launch Vivado and select **Create Project**.

4. Change the name and location of the project as desired and click **Next**.

5. Leave **RTL Project** and **Do not specify sources at this time** checked and click **Next**.

6. Click **Boards** and search for the MiniZed. If the board definition files were downloaded correctly you will see it.



Vendor: All
Name: All
Board Rev: Latest
Search: MiniZed (1 match)

Display Name	Preview	Vendor	File Version	Part	I/O Pin Count	Board Rev	Avail
MiniZed		em.avnet.com	1.2	xc7z007sclg225-1	225	1.0	54

Figure 1 – Select the Target Board

7. Select the MiniZed board and click **Next**. Then click **Finish**.

8. The current project is blank. To access the ARM processing system, we will add an embedded source to the Vivado project using IP Integrator. Select **Create Block Design**.

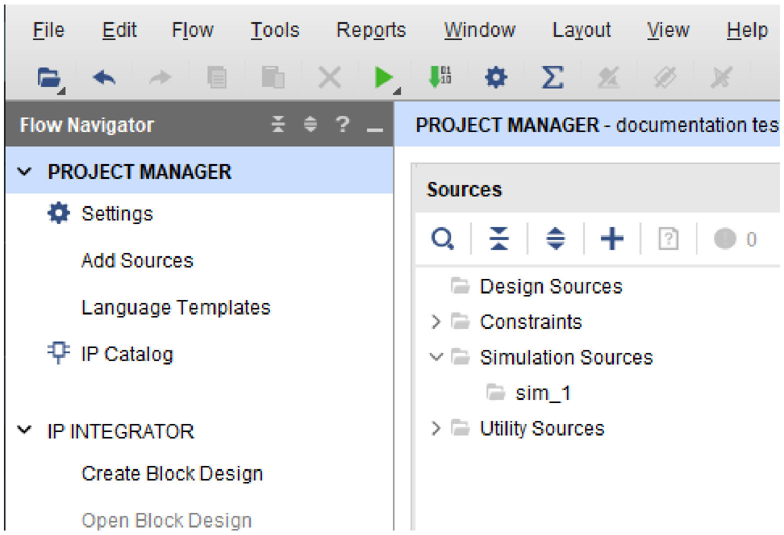
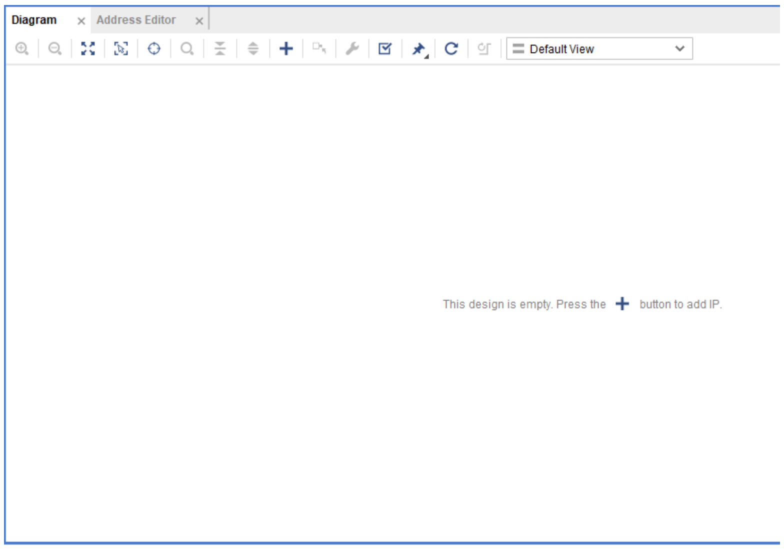


Figure 2 – Create Block Design

9. Give the Block Design a name. *System* is commonly used. Click **Ok**.

10. Click the **Add IP** icon + in either location.



This design is empty. Press the + button to add IP.

Figure 3 – Add IP to the Block Design

11. Search for "Zynq" and find **ZYNQ7 Processing System**. Double-click this or drag and drop to the *Diagram* window.

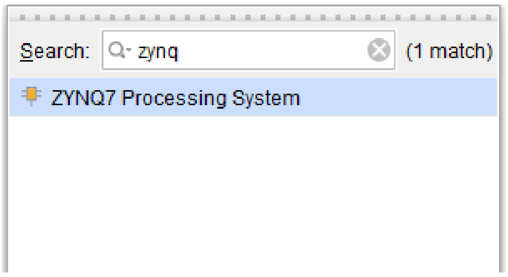


Figure 4 – Add IP Window

12. Select **Run Block Automation** at the top of the Diagram Window.

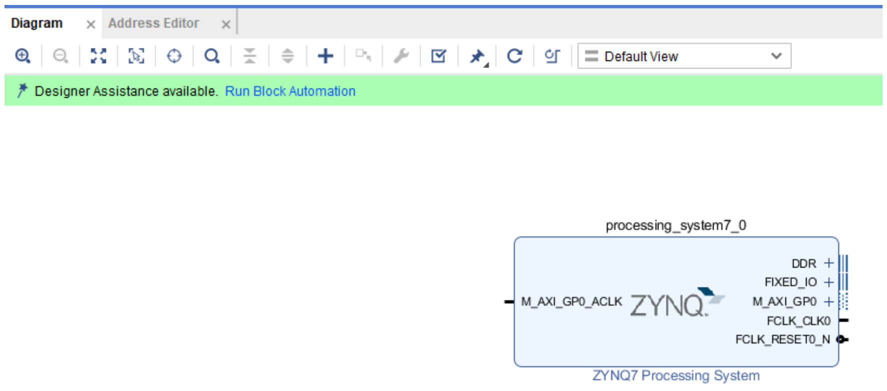


Figure 5 – Run Block Automation

13. Notice the block automation wizard has identified two sources of I/O that need to be made external. One is obvious, the **DDR** interface. The other is labeled **FIXED_IO**. FIXED_IO is basically the MIO pin connections. They are labeled FIXED_IO because you cannot change their assignments in this window.

The *Apply Board Preset* checkbox applies the Preset TCL that was included as part of the board definition archive. Leave this checked.

The Cross Trigger options may be left Disabled.

Click **OK** to connect these external signals.

14. You should now see the ZYNQ block with external I/O.

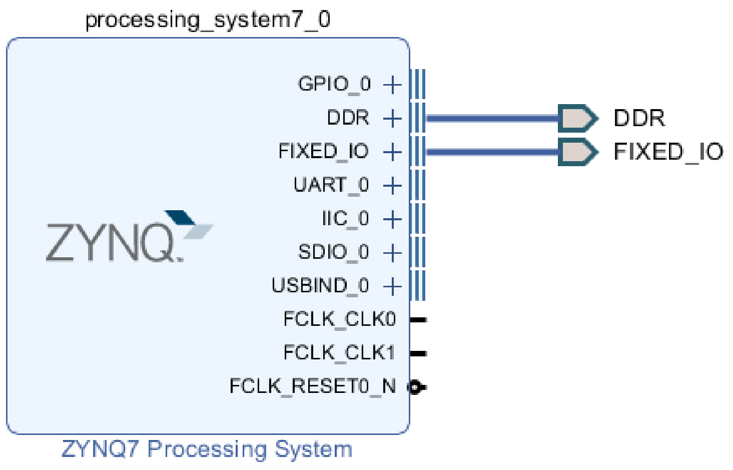


Figure 6 – Zynq Block Diagram for MiniZed with External I/O

15. We can now validate our design. Click the *Validate Design* icon . A successful validation window will appear. Click **OK**.

16. Click **Save Block Design** icon, , to save the project.

17. Switch to the *Sources* tab by clicking on it.

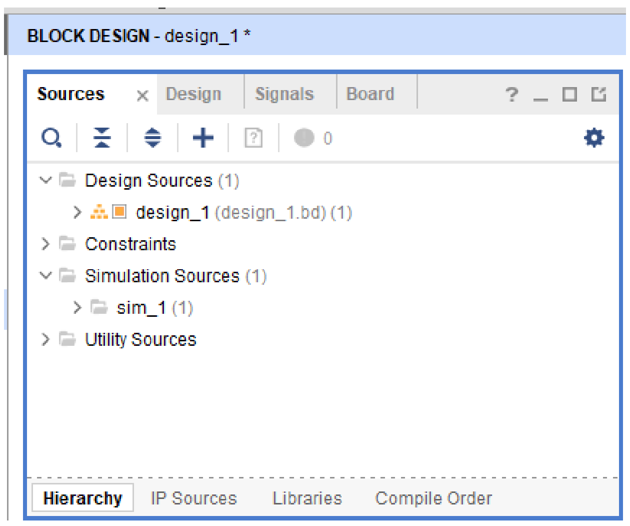
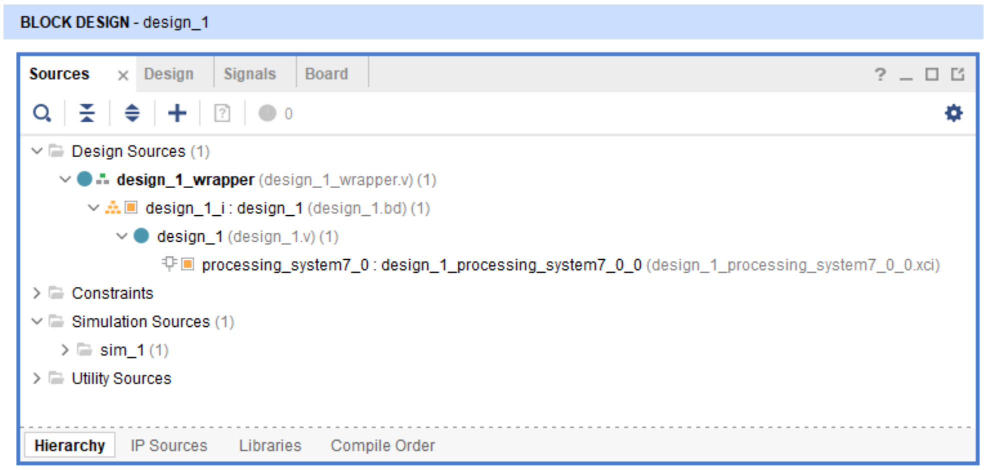


Figure 7 – Sources

18. Right-click on **design_1 (design_1.bd)** and select **Create HDL wrapper.**

19. Leave the option selected to *Let Vivado manage wrapper and auto-update*. Click **OK.**

20. Once the top-level wrapper is created, you can see the design hierarchy in the *Sources* tab. Notice that **System_wrapper.vhd** is the top-level HDL wrapper that was created. **System.bd** is the Block Design.


BLOCK DESIGN - design_1

Sources | Design | Signals | Board

- Design Sources (1)
  - design_1_wrapper (design_1_wrapper.v) (1)
    - design_1_i : design_1 (design_1.bd) (1)
      - design_1 (design_1.v) (1)
        - processing_system7_0 : design_1_processing_system7_0_0 (design_1_processing_system7_0_0.xci)
- Constraints
- Simulation Sources (1)
  - sim_1 (1)
- Utility Sources

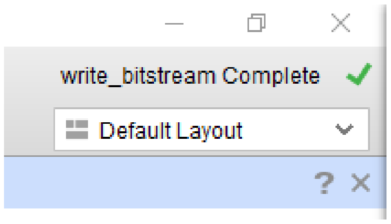
Hierarchy | IP Sources | Libraries | Compile Order

Figure 8 – System_wrapper.vhd Generated

21. Click **Generate Bitstream** in the *Flow Navigator* window.

22. Click **Yes** to start Synthesis and Implementation flows.

23. This may take several minutes. Wait until you see the **write_bitstream Complete** message in the top-right corner.

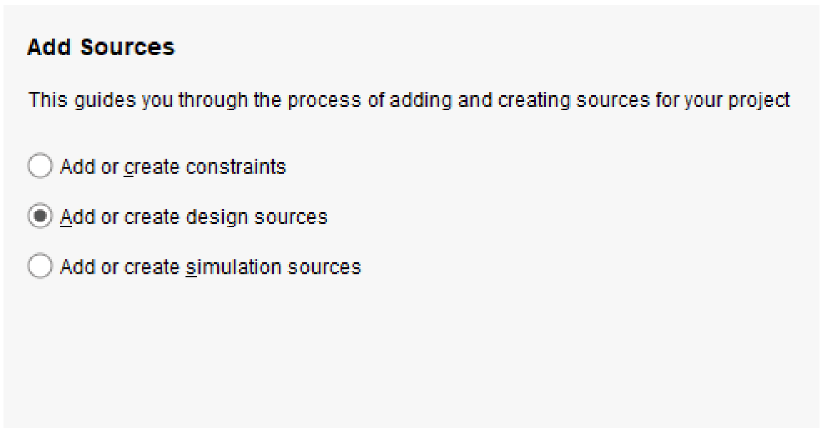

write_bitstream Complete ✓

Default Layout

Figure 9 – write_bitstream Complete

Now that we have created a ZYNQ Hardware Platform for the MiniZed in Vivado, we can create our own RTL module and begin to interface external I/O with the Zynq7 Processing System.

1. Under the *Sources* tab, click the + icon to **Add Sources**. Select **Add or create design sources** and click **Next.**



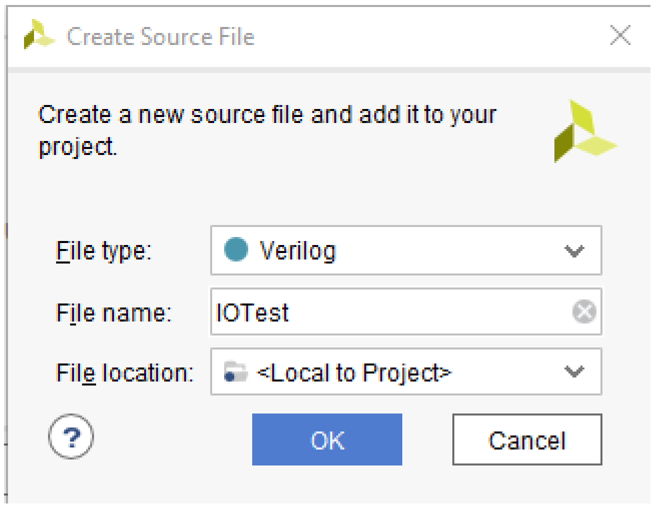
Add Sources

This guides you through the process of adding and creating sources for your project

( ) Add or create constraints
(●) Add or create design sources
( ) Add or create simulation sources

Figure 10 – Add Sources

2. Next, click **Create File**. Name the file accordingly. We are going to name it IOTest for now. Click **OK** and then click **Finish**.



Create Source File

Create a new source file and add it to your project.

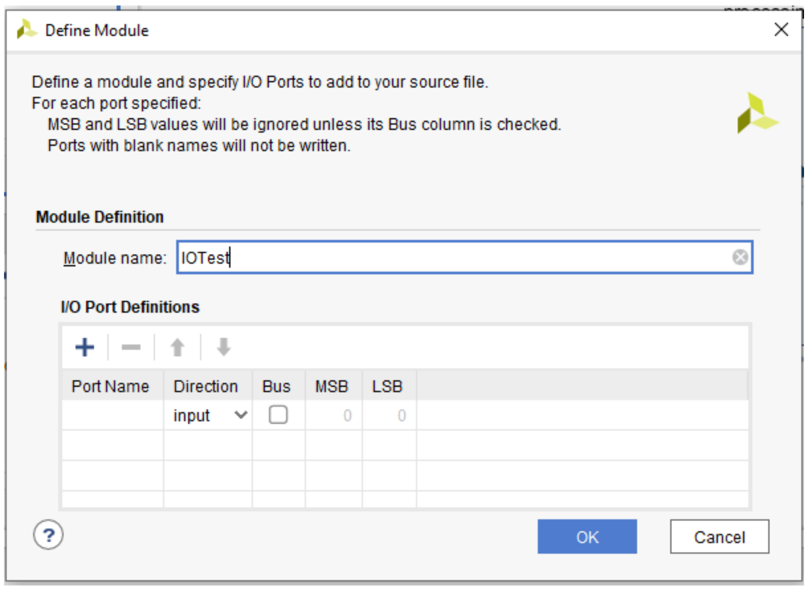
File type: Verilog

File name: IOTest

File location: <Local to Project>

Figure 11 – Create Source File

3. A window will appear telling us to specify I/O ports to add to your source file. Ignore this for now as we will do this manually later. Leave it blank and click **OK**. Ignore the next window and click **Yes**.



Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name: IOTest

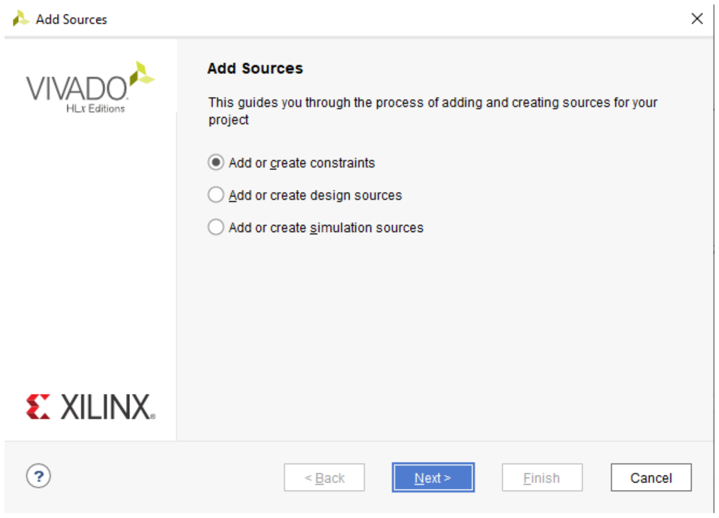
I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	input	☐	0	0

Figure 12 – Define Module

4. Now we need to add our constraints file. Click on the following link and download the .xdc file: https://github.com/Avnet/hdl/blob/master/Boards/MINIZED/minized_pins.xdc.

5. Click on **Add Sources**. Select **Add or create constraints** and click **Next**.



Add Sources

This guides you through the process of adding and creating sources for your project

● Add or create constraints
○ Add or create design sources
○ Add or create simulation sources

Figure 13 – Add Sources

6. Select **Add Files**. Navigate to where you saved the **minized_pins.xdc** file and select it. Click **Finish**.

7. Under *Sources*, double-click **IOTest.v** to begin editing the file.

8. Copy the following code into your source file. This will simply turn on the LEDs corresponding to the respective switch input. Note that the inputs and outputs in the port list must match in the constraints file. We will fix this next.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: SDSU Senior Design
// Engineer: Armaan Kafaipour
//
// Create Date: 03/19/2021 04:25:01 PM
// Design Name:
// Module Name: IOTest
// Project Name: MyDeskBenchSDSU
// Target Devices: MiniZed with I/O PCB
// Tool Versions:
// Description: Sample code to control I/O PCB with MiniZed
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module IOTest(
   input clk,
   input [7:0] sw,
   input [3:0] btn,
   output reg ser,
   output reg latch,
   output reg srclk,
   output reg ser2,
   output reg latch2,
   output reg srclk2,
   output reg [3:0] dig
   );

reg [7:0] led; // 8 LEDs

reg [6:0] LED_out = 7'b0000001; // 7-segment display value

reg [3:0] nextState = 0; // State machine logic for shift register

reg [1:0] two_bit_counter; // To actively drive all 4 digits
reg [31:0] counter = 0; // Clock divider for shift register signals

reg [3:0] LED_BCD; // Sets the value of the 7-segment display


//Clock timing kept above 100ns to meet shift register setup time
//See Datasheet - SN74HC595N
always @(posedge clk) begin
  if(counter < 2_500_000) begin
    counter <= counter + 1;
  end
  else begin
    counter <= 0;
    nextState = nextState + 1;
  end
end


//Counter to cycle between digits
always @(posedge clk) begin
  two_bit_counter = two_bit_counter + 1;
end

//Cycles between each 7-segment digit
//All digits will appear to be on at the same time
always @(*) begin
  case(two_bit_counter)
    0: begin
      dig = 4'b1110;
      end
    1: begin
      dig = 4'b1101;
      end
    2: begin
      dig = 4'b1011;
      end
    3: begin
      dig = 4'b0111;
      end
  endcase
end

//Each switch toggles a single LED
always @(sw) begin
  case(sw)
    8'b00000001: led = 8'b00000001;
    8'b00000010: led = 8'b00000010;
    8'b00000100: led = 8'b00000100;
```

```
        8'b00001000: led = 8'b00001000;
        8'b00010000: led = 8'b00010000;
        8'b00100000: led = 8'b00100000;
        8'b01000000: led = 8'b01000000;
        8'b10000000: led = 8'b10000000;
    endcase
end


//7-segment decoder
always @(*) begin
    case(LED_BCD)
        4'b0000: LED_out = 7'b0000001; // "0"
        4'b0001: LED_out = 7'b1001111; // "1"
        4'b0010: LED_out = 7'b0010010; // "2"
        4'b0011: LED_out = 7'b0000110; // "3"
        4'b0100: LED_out = 7'b1001100; // "4"
        4'b0101: LED_out = 7'b0100100; // "5"
        4'b0110: LED_out = 7'b0100000; // "6"
        4'b0111: LED_out = 7'b0001111; // "7"
        4'b1000: LED_out = 7'b0000000; // "8"
        4'b1001: LED_out = 7'b0000100; // "9"
        4'b1010: LED_out = 7'b0001000; // "A"
        4'b1011: LED_out = 7'b1100000; // "b"
        4'b1100: LED_out = 7'b0110001; // "C"
        4'b1101: LED_out = 7'b1000010; // "d"
        4'b1110: LED_out = 7'b0110000; // "e"
        4'b1111: LED_out = 7'b0111000; // "F"
        default: LED_out = 7'b0000001; // "0"
   endcase
end

// State machine to load 8 bits into shift register
// Outputs to 7-segment display
// Data is loaded in ser, and shifted in on pos edge of srclk
// Latch is set high to shift to output register
always @(nextState) begin
    case(nextState)
        0: begin
                srclk2 = 0;
                latch2 = 1;
           end
        1: begin
                srclk2 = 0;
                latch2 = 0;
                //DECIMAL POINT
                ser2 = 0;
           end
```

```
2: begin
       srclk2 = 1;
   end
3: begin
       srclk2 = 0;
       ser2 = LED_out[0];
   end
 4: begin
       srclk2 = 1;
   end
 5: begin
       srclk2 = 0;
       ser2 = LED_out[1];
   end
 6: begin
       srclk2 = 1;
   end
 7: begin
       srclk2 = 0;
       ser2 = LED_out[2];
   end
 8: begin
       srclk2 = 1;
   end
 9: begin
       srclk2 = 0;
       ser2 = LED_out[3];
    end
 10: begin
       srclk2 = 1;
     end
 11: begin
       srclk2 = 0;
       ser2 = LED_out[4];
     end
 12: begin
       srclk2 = 1;
     end
 13: begin
       srclk2 = 0;
       ser2 = LED_out[5];
     end
 14: begin
       srclk2 = 1;
     end
 15: begin
       srclk2 = 0;
       ser2 = LED_out[6];
```

```
            end
    endcase
end

// State machine to load 8 bits into shift register
// Outputs to 8 LEDs
// Data is loaded in ser, and shifted in on pos edge of srclk
// Latch is set high to shift to output register
always @(nextState) begin
    case(nextState)
        0: begin
                srclk = 1;
                latch = 1;
           end
        1: begin
                srclk = 0;
                latch = 0;
                ser = led[0];
           end
        2: begin
                srclk = 1;
           end
        3: begin
                srclk = 0;
                ser = led[1];
           end
         4: begin
                srclk = 1;
           end
         5: begin
                srclk = 0;
                ser = led[2];
           end
         6: begin
                srclk = 1;
           end
         7: begin
                srclk = 0;
                ser = led[3];
           end
         8: begin
                srclk = 1;
           end
         9: begin
                srclk = 0;
                ser = led[4];
            end
         10: begin
```

```
            srclk = 1;
          end
        11: begin
            srclk = 0;
            ser = led[5];
          end
        12: begin
            srclk = 1;
          end
        13: begin
            srclk = 0;
            ser = led[6];
          end
        14: begin
            srclk = 1;
          end
        15: begin
            srclk = 0;
            ser = led[7];
          end
    endcase
end

endmodule
```

9. Navigate back to the *Sources* tab, open the **Constraints** drop-down menu, and open the previously added constraints file (ending in .xdc). The switches are connected to the J4 Arduino I/O bank, which is an 8-pin connector. Find the section headed by **Arduino 8-pin connector**. We are going to be replacing these constraints with our own to make them more user friendly. For instance, instead of "[get_ports ARD_DAT0]", we will use "[get_ports sw[0]]" and so on. The name in the constraints must match the name in the RTL module. Copy the text from below to replace this section of the constraints file:

```
#ON-BOARD PL LED
set_property PACKAGE_PIN E13 [get_ports PL_LED_G ];
set_property IOSTANDARD LVCMOS33 [get_ports PL_LED_G]

# To ARD_D0 on Arduino 8-pin  Pin 1
set_property PACKAGE_PIN R8 [get_ports sw[0]]
set_property IOSTANDARD LVCMOS33 [get_ports sw[0]]

# To ARD_D1 on Arduino 8-pin  Pin 2
set_property PACKAGE_PIN P8 [get_ports sw[1]]
set_property IOSTANDARD LVCMOS33 [get_ports sw[1]]

# To ARD_D2 on Arduino 8-pin  Pin 3
```

```
set_property PACKAGE_PIN P9 [get_ports sw[2]]
set_property IOSTANDARD LVCMOS33 [get_ports sw[2]]

# To ARD_D3 on Arduino 8-pin  Pin 4
set_property PACKAGE_PIN R7 [get_ports sw[3]]
set_property IOSTANDARD LVCMOS33 [get_ports sw[3]]

# To ARD_D4 on Arduino 8-pin  Pin 5
set_property PACKAGE_PIN N7 [get_ports sw[4]]
set_property IOSTANDARD LVCMOS33 [get_ports  sw[4]]

# To ARD_D5 on Arduino 8-pin  Pin 6
set_property PACKAGE_PIN R10 [get_ports sw[5]]
set_property IOSTANDARD LVCMOS33 [get_ports sw[5]]

# To ARD_D6 on Arduino 8-pin  Pin 7
set_property PACKAGE_PIN P10 [get_ports sw[6]]
set_property IOSTANDARD LVCMOS33 [get_ports sw[6]]

# To ARD_D7 on Arduino 8-pin  Pin 8
set_property PACKAGE_PIN N8 [get_ports sw[7]]
set_property IOSTANDARD LVCMOS33 [get_ports sw[7]]
```

10. Open the Block Design again (design_1.bd). Right-click anywhere on the diagram and select **Add Module**. Alternatively, drag your RTL module from the *Sources* tab and drop it into the diagram.

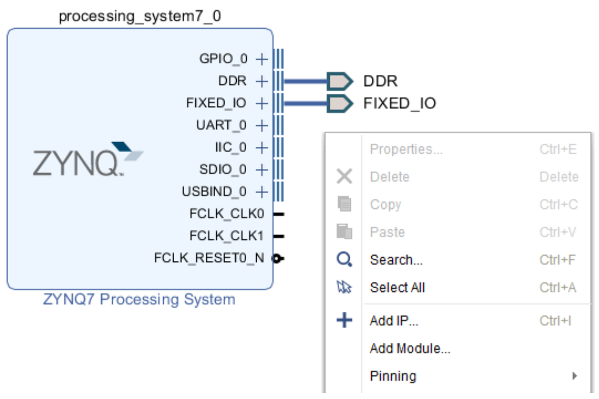


Figure 14 – Add Module

11. Select your RTL module, IOTest.v. Your diagram should like similar to the figure below.

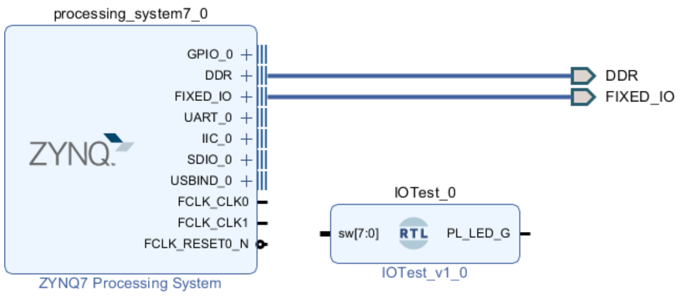


Figure 15 – RTL Module

12. Select the **Regenerate Layout** icon.

13. Right-click the IOTest module, and select **Make External**. Alternatively, click on it once to highlight it and press **CTLR + T**.

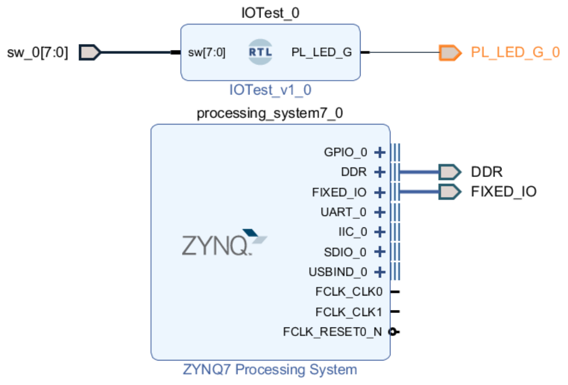
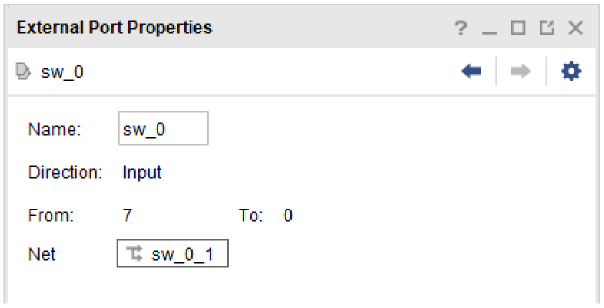


Figure 16 – Block Diagram

14. Click on the external nodes, **sw_0[7:0]** and **PL_LED_G_0**. Look at the External Port Properties. Remove the "_0" from the name "sw_0" so that it simply reads "sw". Do the same for "PL_LED_G_0".


External Port Properties

sw_0

Name: sw_0

Direction: Input

From: 7 To: 0

Net sw_0_1

Figure 17 – External Port Properties

15. Click **Generate Bitstream**.

16. Click **Open Hardware Manager**.

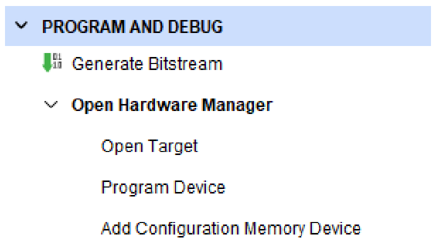
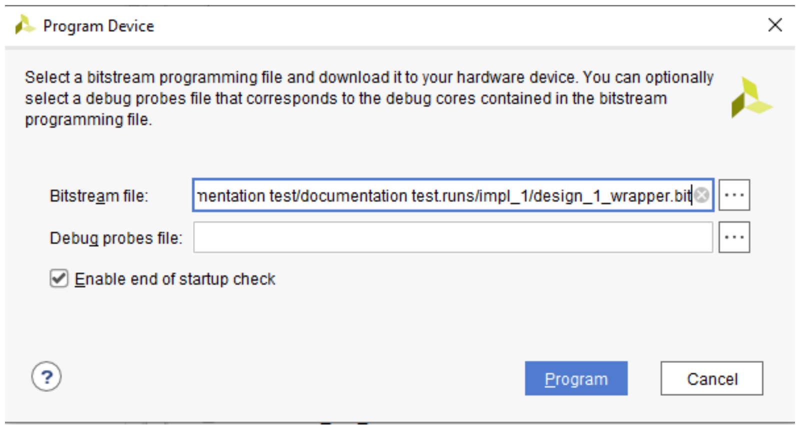


Figure 18 – Hardware Manager

17. Ensure that the MiniZed is properly connected via USB and set in JTAG mode. Then, click **Open Target** and **Auto Connect**.

18. Once connected, click **Program Device**. Select **xc7z007s_1**. The Bitstream File should be automatically selected. Leave *Enable end of startup check* checked. Click **Program**.



Program Device

Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: nentation test/documentation test.runs/impl_1/design_1_wrapper.bit

Debug probes file:

☑ Enable end of startup check

Figure 19 – Program Device

***IMPORTANT NOTE –** Anytime a change is made to the Verilog code in the RTL module, you must navigate to the block diagram, right-click on the RTL module block, and select **Refresh Module**. This must be done to generate an updated bitstream.