JOINT INSTITUTE
交大密西根学院

ECE4810J System-on-Chip (SoC) Design

Fall 2021

Lab #5 HDMI Demo on Arty Z7 and Xilinx Design Constraints

Due: November 11th 11:59pm, 2021

Logistics:

- This lab is a team exercise.
- Please use the discussion board on Piazza for Q&A.
- All reports and code (if available) MUST be submitted to the assignment of Canvas.
- Internet usage is allowed and encouraged.
- No late submission is allowed for this lab.
- All necessary files for this lab have been uploaded to canvas.

1. Overview

This lab is divided into two parts. The first part (part 2) is a demo how to use the USB-UART Bridge, HDMI Sink and HDMI Source with the ZYNQ processor. The second lab is about the Xilinx Design Constraints, after completing this lab, you will be able to:

- Create a I/O Planning project
- Enter the pin locations and IO standards via Device view, Package Pins tab, and Tcl commands
- Create Period, Input Setup, and Output Setup delays
- Perform timing analysis

2. Arty HDMI Demo

2.1 Arty Z7 HDMI In Demo

The Arty Z7 HDMI In project demonstrates the usage of the HDMI in and out ports on the Arty Z7 board. The behavior is as follows:

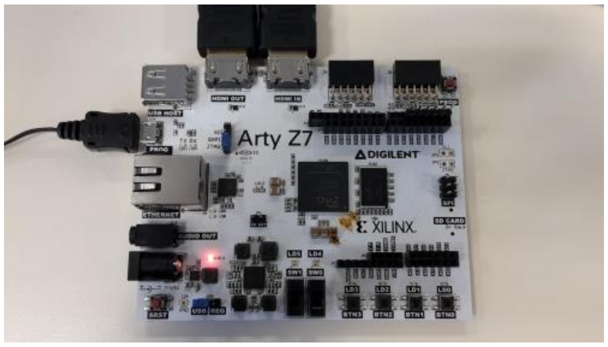
- Video data streams in through the HDMI in port and out through the HDMI out port.
- A UART interface is available to configure what is output through HDMI.
- There are 3 display frame buffers that the user can choose to display or write to.

Downloads

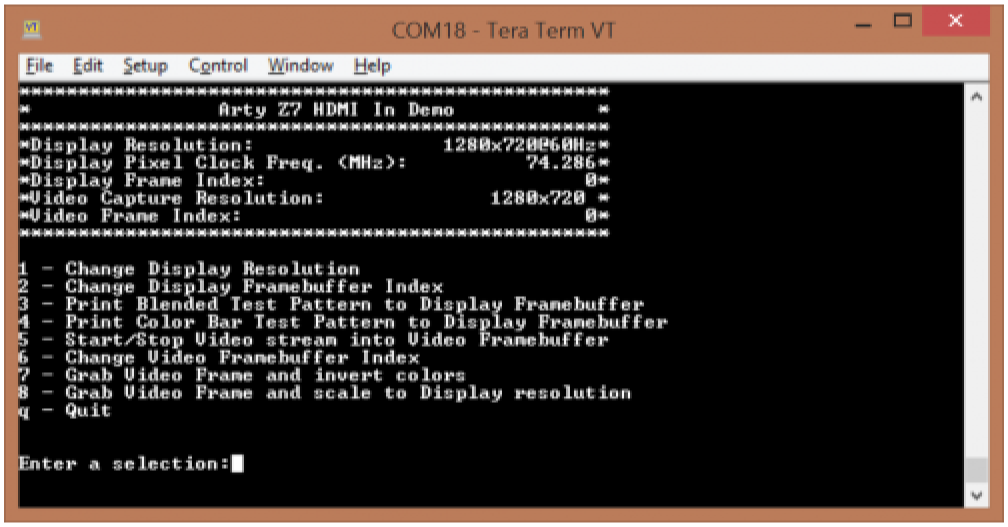
Arty Z7-20 Hdmi In Project Repository – ZIP Git Repo

Steps

1) Follow the Using Digilent Github Demo Projects Tutorial. Since this is a Vivado SDK Project, you can either directly launch SDK and import the hardware handoff, or you can generate a bitstream in Vivado before launching SDK. Select the hardware handoff options in the tutorial if you don't want to modify the project block design later. Return to this guide when prompted to check for additional hardware requirements and setup.
2) Plug one end of the HDMI cable into a video monitor and the other into the Arty Z7 HDMI out port. Do the same for an HDMI source, likely your computer, and the HDMI in port.



3) Turn on your board and open a serial terminal (such as TeraTerm) on your computer to receive status messages. Setup the serial port to connect to the appropriate port for your board, with a baud rate of 115200, 8 data bits, no parity bit and 1 stop bit. Then return to the Github Project Tutorial to finish programming and running the demo.



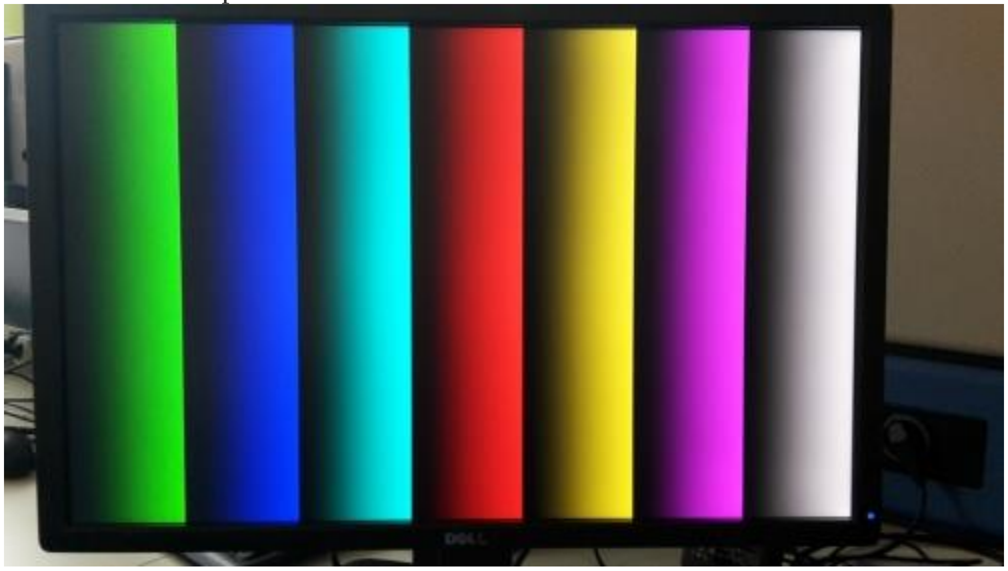
```
**************************************************
*                Arty Z7 HDMI In Demo            *
**************************************************
*Display Resolution:                  1280x720@60Hz*
*Display Pixel Clock Freq. (MHz):           74.286*
*Display Frame Index:                           0*
*Video Capture Resolution:             1280x720 *
*Video Frame Index:                             0*
**************************************************

1 - Change Display Resolution
2 - Change Display Framebuffer Index
3 - Print Blended Test Pattern to Display Framebuffer
4 - Print Color Bar Test Pattern to Display Framebuffer
5 - Start/Stop Video stream into Video Framebuffer
6 - Change Video Framebuffer Index
7 - Grab Video Frame and invert colors
8 - Grab Video Frame and scale to Display resolution
q - Quit


Enter a selection:
```

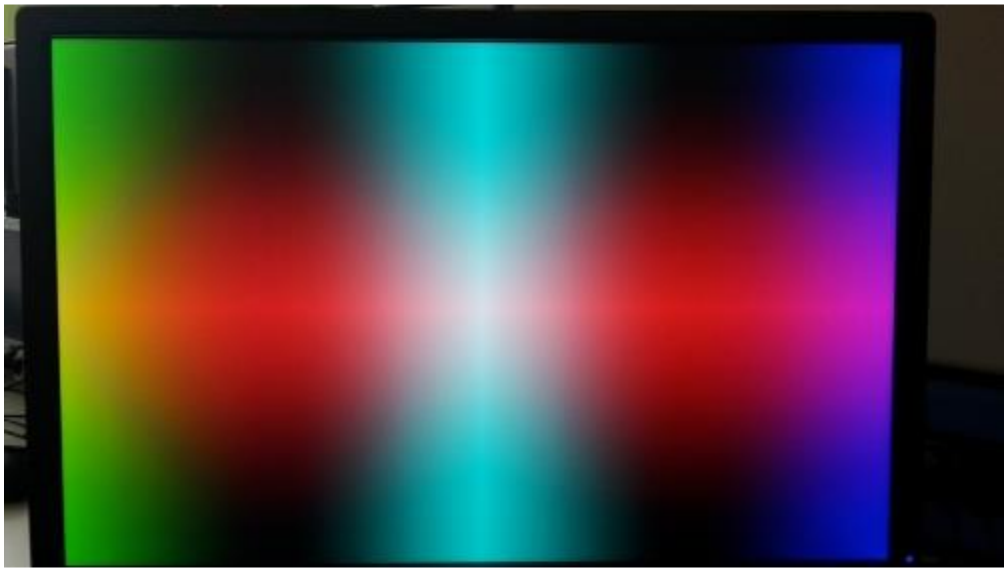
Using the Arty Z7 HDMI Input Demo

HDMI Display Options

1 - Change the resolution of the HDMI output to the monitor.
2 - Change the frame buffer to display on the HDMI monitor.

3/4 - Store a test pattern in the chosen video frame buffer - color bar or blended.





5 - Start/Stop streaming video data from HDMI to the chosen video frame buffer.
6 - Change the video frame buffer that HDMI data is streamed into.
7 - Invert and store the current video frame into the next video frame buffer and display it.

8 - Scale the current video frame to the display resolution, store it into the next video frame buffer, and then display it.

2.2 Arty Z7 HDMI Out Demo

The Arty Z7 HDMI Out project demonstrates the usage of the HDMI out port on the Arty Z7 board. The behavior is as follows:

- Frames stream out through the HDMI port.
- A UART interface is available to configure what is output through HDMI.
- There are 3 display frame buffers that the user can choose to display.

Downloads

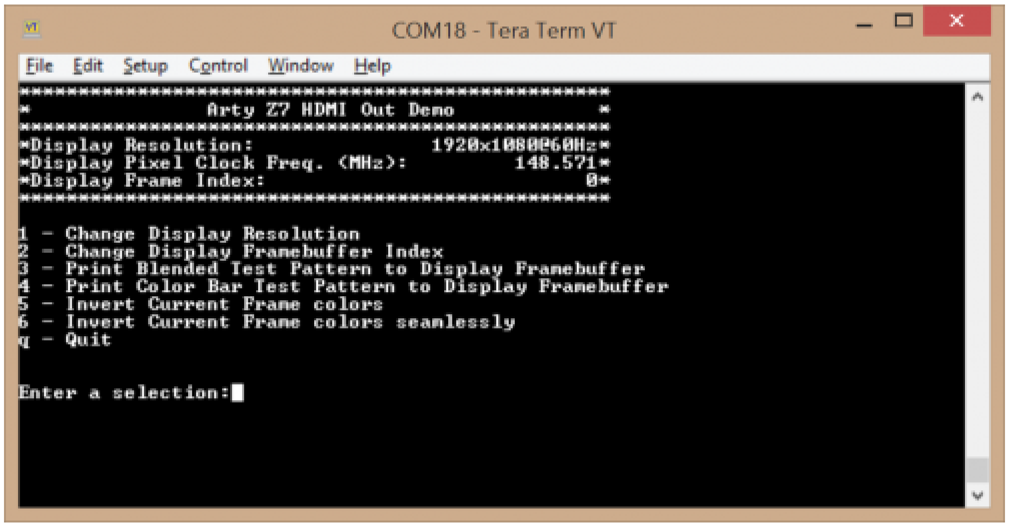
Arty Z7-20 Hdmi Out Project Repository – ZIP Git Repo

Download and Launch the Arty Z7 HDMI Demo

1) Follow the Using Digilent Github Demo Projects Tutorial. Since this is a Vivado SDK Project, you can either directly launch SDK and import the hardware handoff, or you can generate a bitstream in Vivado before launching SDK. Select the hardware handoff options in the tutorial if you don't want to modify the project block design later. Return to this guide when prompted to check for additional hardware requirements and setup.

2) Plug one end of the HDMI cable into a video monitor and the other into the Arty Z7 HDMI out port.

3) Turn on your board and open a serial terminal (such as TeraTerm) on your computer to receive status messages. Setup the serial port to connect to the appropriate port for your board, with a baud rate of 115200, 8 data bits, no parity bit and 1 stop bit. Then return to the Github Project Tutorial to finish programming and running the demo.



```
**************************************************
*                Arty Z7 HDMI Out Demo           *
**************************************************
*Display Resolution:                  1920x1080@60Hz*
*Display Pixel Clock Freq. (MHz):           148.571*
*Display Frame Index:                             0*
**************************************************

1 - Change Display Resolution
2 - Change Display Framebuffer Index
3 - Print Blended Test Pattern to Display Framebuffer
4 - Print Color Bar Test Pattern to Display Framebuffer
5 - Invert Current Frame colors
6 - Invert Current Frame colors seamlessly
q - Quit

Enter a selection:
```

Using the Arty Z7 HDMI Demo

HDMI Display Options

1 - Changes the resolution of the HDMI output to the monitor.

2 - Changes the frame buffer index.

3/4 - Prints a test pattern in the chosen frame buffer: blended or color bar.

5 - Inverts current frame colors.

6 - Inverts current frame colors seamlessly.

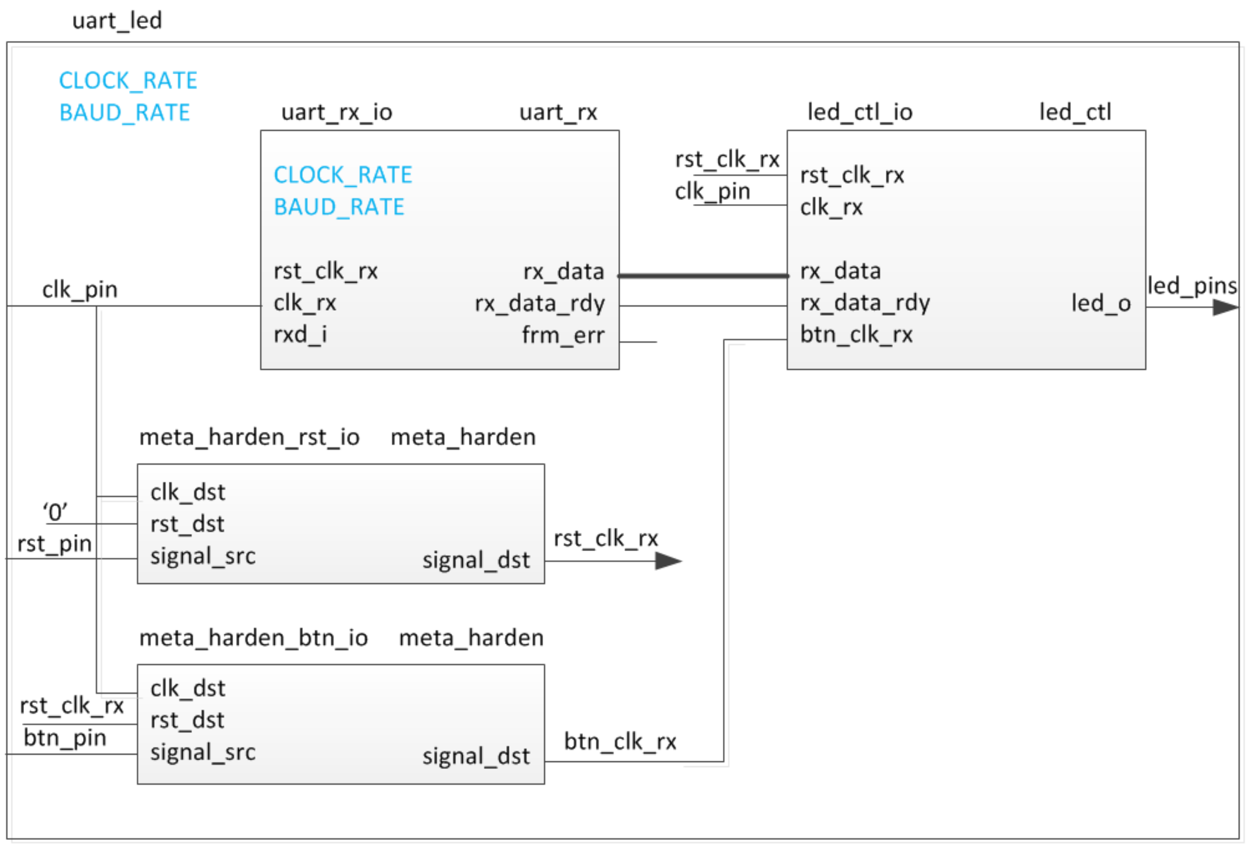
3. Xilinx Design Constraints

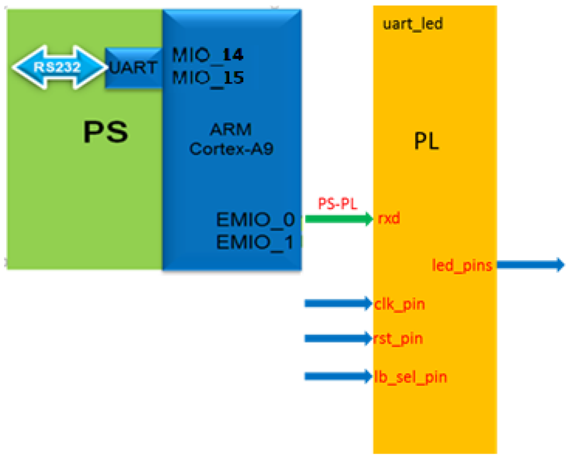
To learn more about XDC, please go to canvas>Lab>Lab#5> 16_Xilinx Design Constraints.pdf. The source file required for this lab has also been uploaded to the lab 5 folder.

The design consists of a uart receiver receiving the input typed on a keyboard and displaying the binary equivalent of the typed character on the 8 LEDs. When a push button is pressed, the lower and upper nibbles are swapped.

In this design we will use board's USB-UART which is controlled by the Zynq's ARM Cortex-A9 processor. Our PL design needs access to this USB-UART. So first thing we will do is to create a Processing System design which will put the USB-UART connections in a simple GPIO-style and make it available to the PL section.

The provided design places the UART (RX) pin of the PS (Processing System) on the Cortex-A9 in a simple GPIO mode to allow the UART to be connected (passed through) to the Programmable Logic. The processor samples the RX signal and sends it to the EMIO channel 0 which is connected to Rx input of the HDL module provided in the Static directory. This is done through a software application provided in the lab5.sdk folder hierarchy.



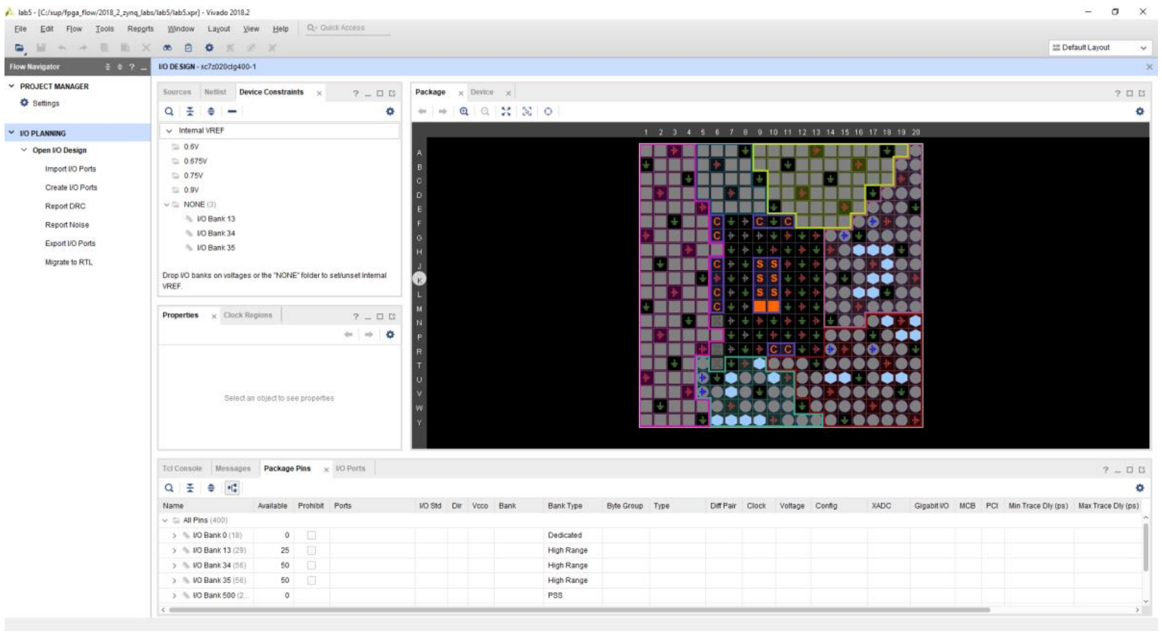


Steps

Create a Vivado I/O Planning Project

Launch Vivado and create a project targeting the XC7Z020clg400-1 device, and use the provided tcl script file (ps7_create_pynq.tcl) to generate the block design for the PS subsystem. Also, add the Verilog HDL files, uart_led_pins_pynq.xdc and uart_led_timing_pynq.xdc files from the <2018_2_zynq_sources>\lab5 *directory*.

1. Open Vivado by selecting **Start > Xilinx Design Tools > Vivado 2018.2**
2. Click **Create New Project** to start the wizard. You will see *Create A New Vivado Project* dialog box. Click **Next**.
3. Click the Browse button of the *Project location* field of the **New Project** form, browse to **<2018_2_zynq_labs>**, and click **Select**.
4. Enter **lab5** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.
5. Select the **I/O Planning Project** option in the *Project Type* form, and click **Next**.
6. Select **Do not import I/O ports at this time**, and click **Next**.
7. In the *Default Part* form, Use the **Boards** option, you may select the **PYNQ-Z1** or the **PYNQ-Z2** depending on your board from the Display Name drop down field. You may also use the **Parts** option and various drop-down fields of the **Filter** section, select the **XC7Z020clg400-1 part**.
8. Click **Next**.
9. Click **Finish** to create the Vivado project.
   The device view window and package pins tab will be displayed.



Internal VREF
- 0.6V
- 0.675V
- 0.75V
- 0.9V
- NONE (3)
  - I/O Bank 13
  - I/O Bank 34
  - I/O Bank 35

Drop I/O banks on voltages or the "NONE" folder to set/unset internal VREF.

Select an object to see properties

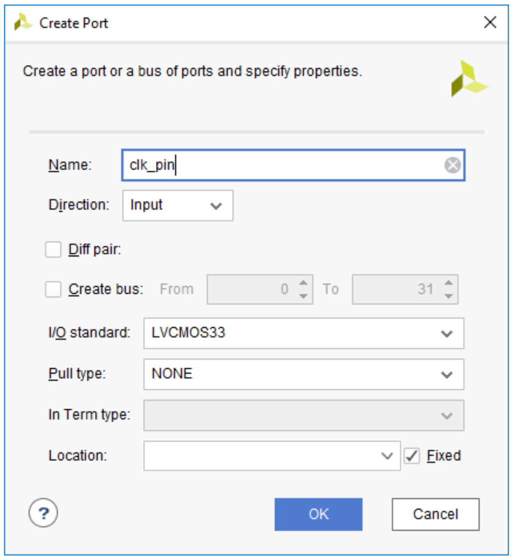
Name	Available	Prohibit	Ports	I/O Std	Dir	Vcco	Bank	Bank Type	Byte Group	Type	Diff Pair	Clock	Voltage	Config	XADC	Gigabit I/O	MCB	PCI	Min Trace Dly (ps)	Max Trace Dly (ps)
All Pins (400)																				
I/O Bank 0 (18)	0	☐						Dedicated												
I/O Bank 13 (29)	25	☐						High Range												
I/O Bank 34 (56)	50	☐						High Range												
I/O Bank 35 (56)	50	☐						High Range												
I/O Bank 500 (2...	0							PSS												

I/O Planning project's default windows and views

Create I/O Ports, Assign Various Pins and Add Source Files

Create input ports clk_pin, btn_pin and rst_pin.

1. Select **Flow Navigator > I/O PLANNING > Open I/O Design > Create I/O Parts**. The Create I/O Ports form will be displayed.
2. Type **clk_pin** in the *Name* field, select **Input** for the *Direction* and select **LVCMOS33** as the *I/O Standard*, and click **OK**.



Create Port

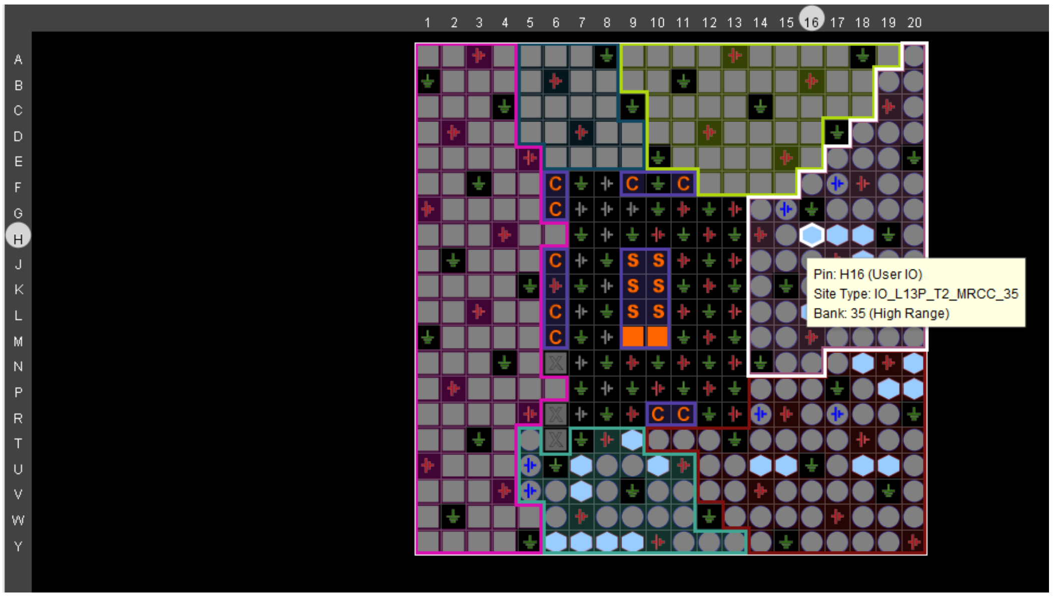
Create a port or a bus of ports and specify properties.

Name:	clk_pin
Direction:	Input
☐ Diff pair:	
☐ Create bus:	From 0 To 31
I/O standard:	LVCMOS33
Pull type:	NONE
In Term type:	
Location:	☑ Fixed

Creating I/O Port for clk_pin input

3. Similarly, create the **btn_pin** and **rst_pin** input ports.

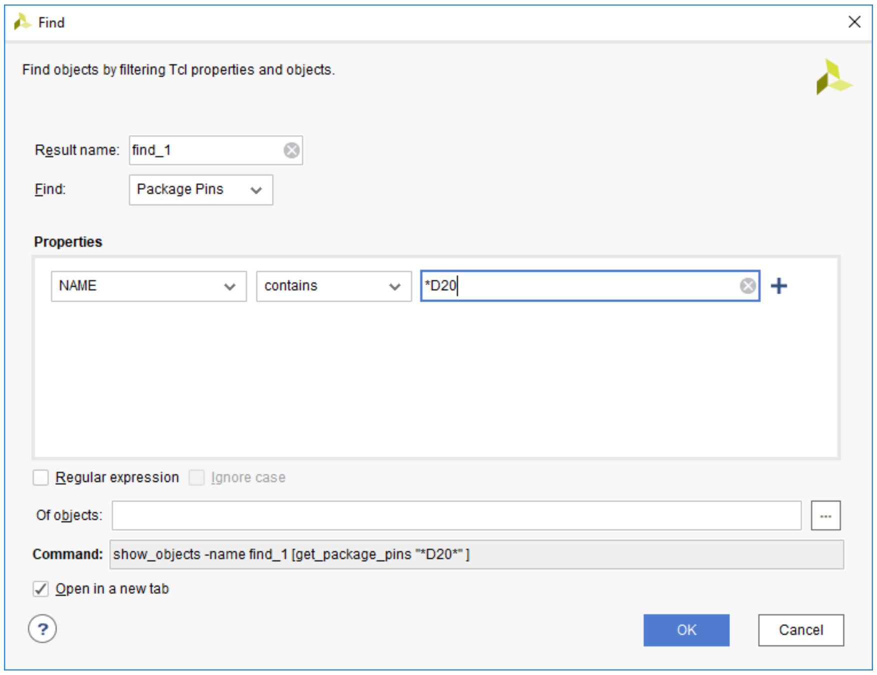
Assign input pins clk_pin, btn_pin and rst_pin to H16, D19 and D20 locations using the Device view and package pins.
Hover the mouse over **H16** in the Device view window.



Pin: H16 (User IO)
Site Type: IO_L13P_T2_MRCC_35
Bank: 35 (High Range)

Locating H16 pin in the Device view

1. When located, click on it.
   The pin entry will be highlighted and displayed in the Package Pins tab.
2. In the *Package Pins* pane, click in the *Ports* column of **H16** pin's row, and select **clk_pin**.
3. Similarly, add the **btn_pin** input port at **D19**.
4. Select **Edit > Find** or Ctrl-F to open the Find form. Select **Package Pins** in the *Find* drop-down field, type **D20** in the match criteria field, and click on **OK**.


Find

Find objects by filtering Tcl properties and objects.

Result name: find_1

Find: Package Pins

Properties

NAME | contains | *D20

Regular expression Ignore case

Of objects:

Command: show_objects -name find_1 [get_package_pins "*D20*"]

✓ Open in a new tab

Finding a package pin

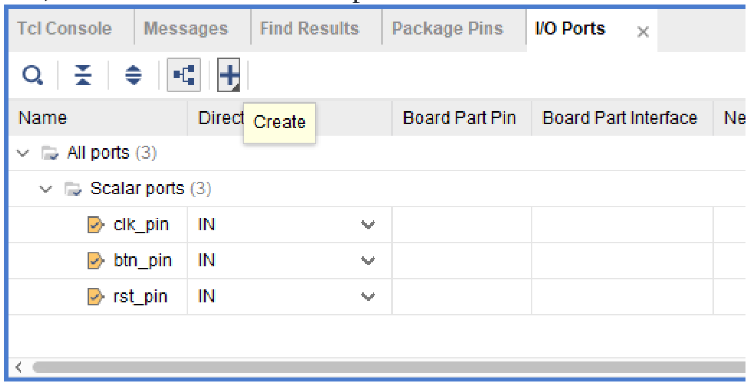
Notice that the Find Results tab is opened, and the corresponding entry is shown in the tab.

5. Assign the **rst_pin** input to the pin.

Assign output pins led_pins[0] to led_pins[7] to locations R14, P14, N16, M14, W14, Y14, T11, T10. Create them as a vector and assign them using the Tcl command set_property. *They all will be LVCMOS33.*

Note: Notice that PYNQ has four LEDs hence we assign led_pins[3:0] to LEDs and led_pins[7:4] are assigned to PMODB.

1. In the I/O Ports tab, click on the create I/O port button on the left vertical ribbon.

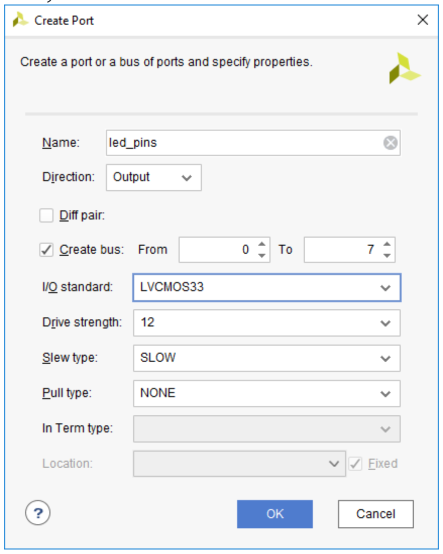


Name	Direct	Board Part Pin	Board Part Interface	Ne
All ports (3)				
Scalar ports (3)				
clk_pin	IN			
btn_pin	IN			
rst_pin	IN			

Create I/O Ports button

The Create I/O Ports form will be displayed.

2. Type **led_pins** in the *Name* field, select *Output* direction, click on the check-box of **Create bus**, set the msb to **7**, and select **LVCMOS33** I/O standard and click **OK**.



Create Port

Create a port or a bus of ports and specify properties.

Name: led_pins
Direction: Output
Diff pair: ☐
Create bus: ✓ From 0 To 7
I/O standard: LVCMOS33
Drive strength: 12
Slew type: SLOW
Pull type: NONE
In Term type:
Location: ✓ Fixed

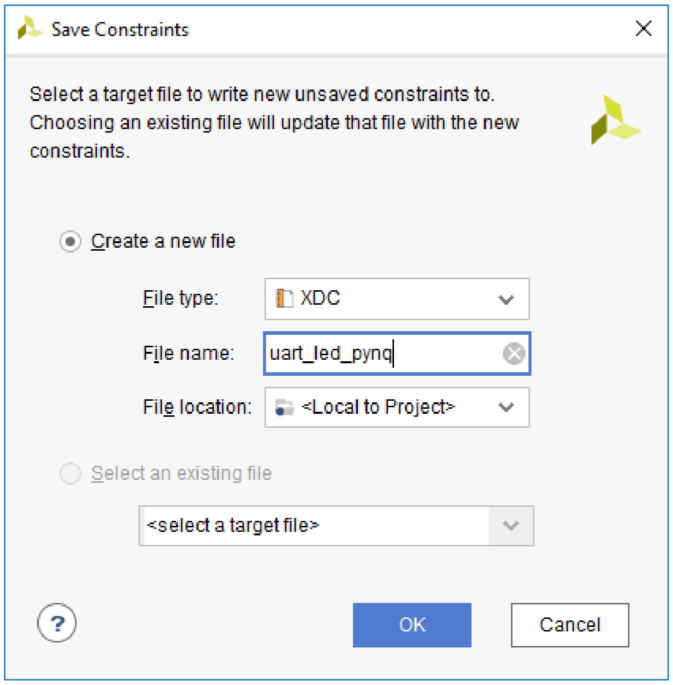
Creating I/O ports for the led_pins output

The led_pins entries will be created and displayed in the I/O Ports tab. Notice that the I/O standard and directions are already set, leaving only the pin locations to be assigned.

3. Type the following commands in the console to assign the pin locations.

set_property -dict { PACKAGE_PIN R14 IOSTANDARD LVCMOS33 } [get_ports { led_pins[0] }];
set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { led_pins[1] }];
set_property -dict { PACKAGE_PIN N16 IOSTANDARD LVCMOS33 } [get_ports { led_pins[2] }];
set_property -dict { PACKAGE_PIN M14 IOSTANDARD LVCMOS33 } [get_ports { led_pins[3] }];
set_property -dict { PACKAGE_PIN W14 IOSTANDARD LVCMOS33 } [get_ports { led_pins[4] }];
set_property -dict { PACKAGE_PIN Y14 IOSTANDARD LVCMOS33 } [get_ports { led_pins[5] }];
set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { led_pins[6] }];
set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { led_pins[7] }];

4. Select **File > Save Constraints**.
   The Save Constraints form will be displayed.
5. Enter **uart_led_pynq** in the *File name* field, and click **OK**.


Save Constraints

Select a target file to write new unsaved constraints to. Choosing an existing file will update that file with the new constraints.

[✓ Create a new file]

File type: XDC

File name: uart_led_pynq

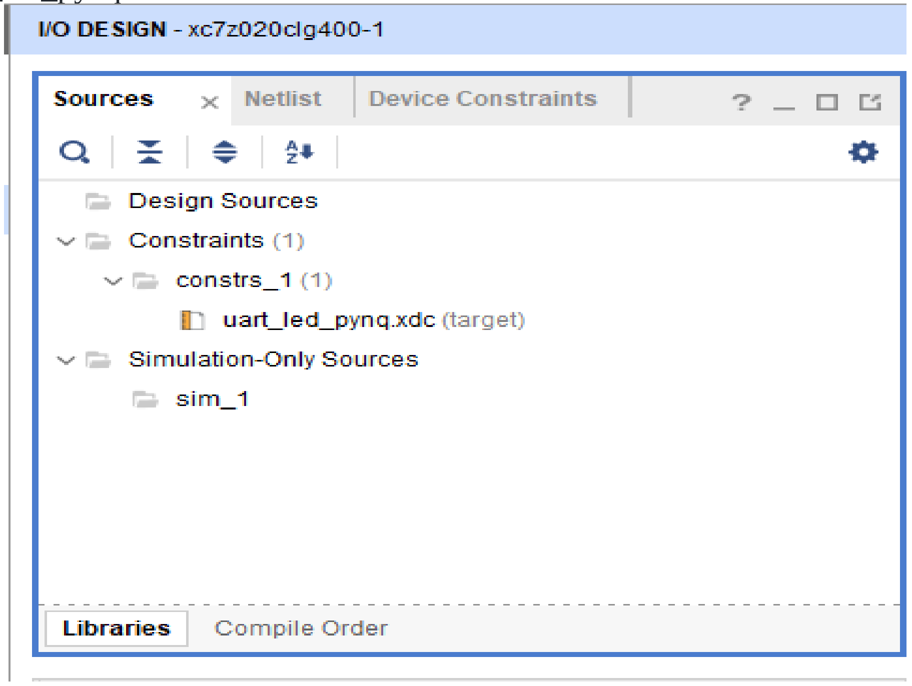
File location: <Local to Project>

Select an existing file

<select a target file>

Saving constraints

The uart_led_pynq.xdc file will be created and added to the Sources tab.


I/O DESIGN - xc7z020clg400-1

```
Design Sources
Constraints (1)
  constrs_1 (1)
    uart_led_pynq.xdc (target)
Simulation-Only Sources
  sim_1
```

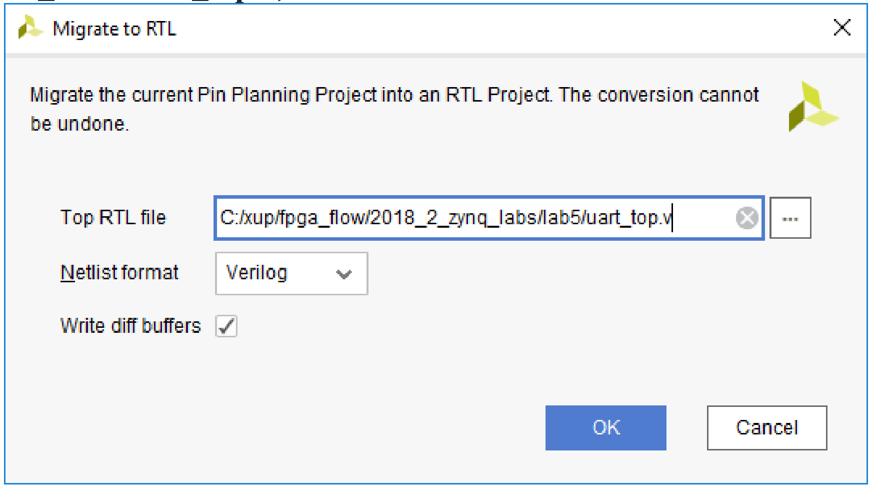
The uart_led_pynq.xdc file added to the source tree

6. Expand the **Flow Navigator** > **I/O PLANNING** > **Open I/O Design** > **Report DRC**.

7. Click **OK**. Notice the design rules checker is run warnings is reported. Ignore the warnings.
8. Expand the **Flow Navigator** > **I/O PLANNING** > **Open I/O Design** > **Report Noise** and click **OK**. Notice the noise analysis is done on the output pins only (led_pins) and the results are displayed.
9. Click on **Migrate to RTL**.

The *Migrate to RTL* form will be displayed with Top RTL file field showing *c:/xup/fpga_flow/2018_2_zynq_labs/lab5/io_1.v* entry.

10. Change *io_1.v* to **uart_top.v**, and click **OK**



Migrate the current Pin Planning Project into an RTL Project. The conversion cannot be undone.

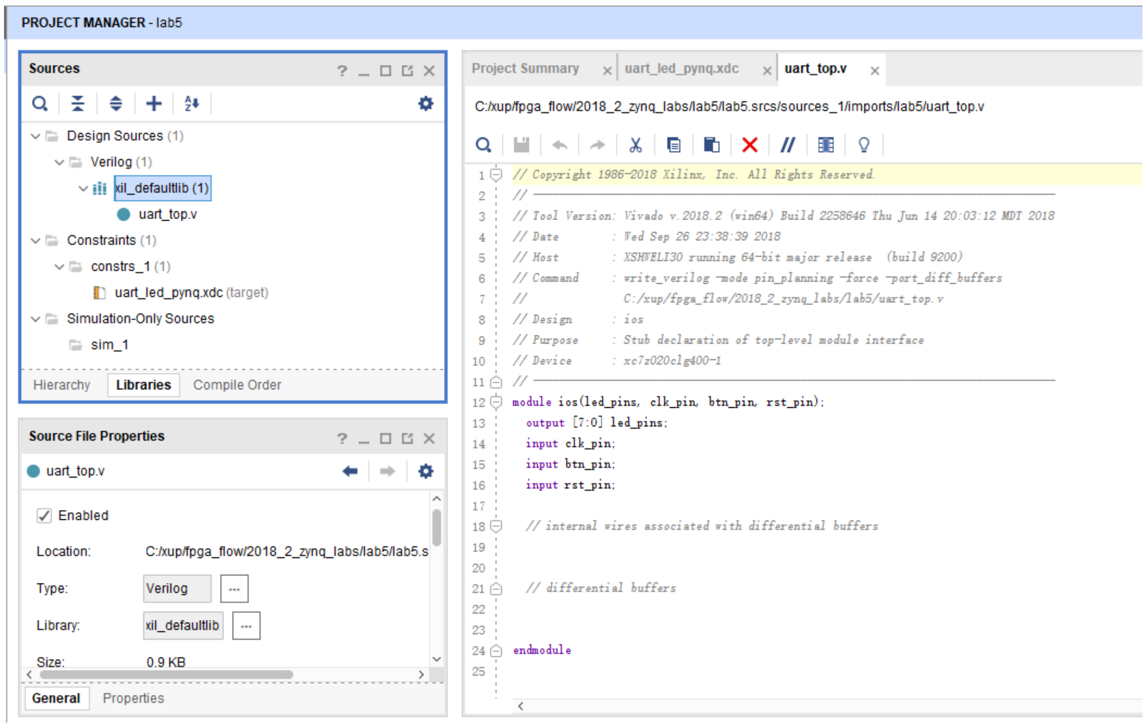
Top RTL file: C:/xup/fpga_flow/2018_2_zynq_labs/lab5/uart_top.v

Netlist format: Verilog

Write diff buffers: ✓

Assigning top-level file name

11. Select the **Hierarchy** tab and notice that the *uart_top.v* file has been added to the project with top-level module name as **ios**. If you double-click the entry, you will see the module name with the ports listing.


PROJECT MANAGER - lab5

Sources
- Design Sources (1)
  - Verilog (1)
    - xil_defaultlib (1)
      - uart_top.v
- Constraints (1)
  - constrs_1 (1)
    - uart_led_pynq.xdc (target)
- Simulation-Only Sources
  - sim_1

Source File Properties: uart_top.v

Enabled	✓
Location:	C:/xup/fpga_flow/2018_2_zynq_labs/lab5/lab5.s
Type:	Verilog
Library:	xil_defaultlib
Size:	0.9 KB

C:/xup/fpga_flow/2018_2_zynq_labs/lab5/lab5.srcs/sources_1/imports/lab5/uart_top.v

```
1  // Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
2  // --------------------------------------------------------------------------------
3  // Tool Version: Vivado v.2018.2 (win64) Build 2258646 Thu Jun 14 20:03:12 MDT 2018
4  // Date         : Wed Sep 26 23:38:39 2018
5  // Host         : XSHWELI30 running 64-bit major release  (build 9200)
6  // Command      : write_verilog -mode pin_planning -force -port_diff_buffers
7  //                C:/xup/fpga_flow/2018_2_zynq_labs/lab5/uart_top.v
8  // Design       : ios
9  // Purpose      : Stub declaration of top-level module interface
10 // Device       : xc7z020clg400-1
11 // --------------------------------------------------------------------------------
12 module ios(led_pins, clk_pin, btn_pin, rst_pin);
13   output [7:0] led_pins;
14   input clk_pin;
15   input btn_pin;
16   input rst_pin;
17
18   // internal wires associated with differential buffers
19
20
21   // differential buffers
22
23
24 endmodule
25
```

The top-level module content and the design hierarchy after migrating to RTL

Add the provided source files (from <2018_2_zynq_sources>\lab5) to the project. Copy the uart_top.txt (located in the <2018_2_zynq_sources >\lab5) content into the top-level source file.

1. Click **Flow Navigator > Add Sources**.
2. In the *Add Sources* form, select *Add or Create Design Sources*, and click **Next**.
3. Click on the **Blue Plus** button, then the **Add Files…**
4. Browse to **<2018_2_zynq_sources>\lab5** and select all .v(led_ctl.v, meta_harden.v, uart_baud_gen.v, uart_led.v, uart_rx.v uart_rx_ctl.v) files and click **OK**.
5. Click **Finish**.
6. Using Windows Explorer, browse to **<2018_2_zynq_sources>\lab5** and open uart_top.txt using any text editor. Copy the content of it and paste it in uart_top.v (around line 22) in the Vivado project.
7. In the Tcl Shell window enter the following command to change to the lab directory and hit the Enter key.
   cd C:/xup/fpga_flow/2018_2_zynq_sources/lab5
8. Generate the PS design by executing the provided Tcl script.
   source ps7_create_pynq.tcl
   This script will create a block design called *system*, instantiate ZYNQ PS with one GPIO channel (GPIO14) and one EMIO channel. It will then create a top-level wrapper file called system_wrapper.v which will instantiate the system.bd (the block design). You can check the contents of the tcl files to confirm the commands that are being run.
9. Double-click on the **uart_led** entry to view its content.

Notice in the Verilog code, the BAUD_RATE and CLOCK_RATE parameters are defined to be 115200 and 125 MHz respectively.

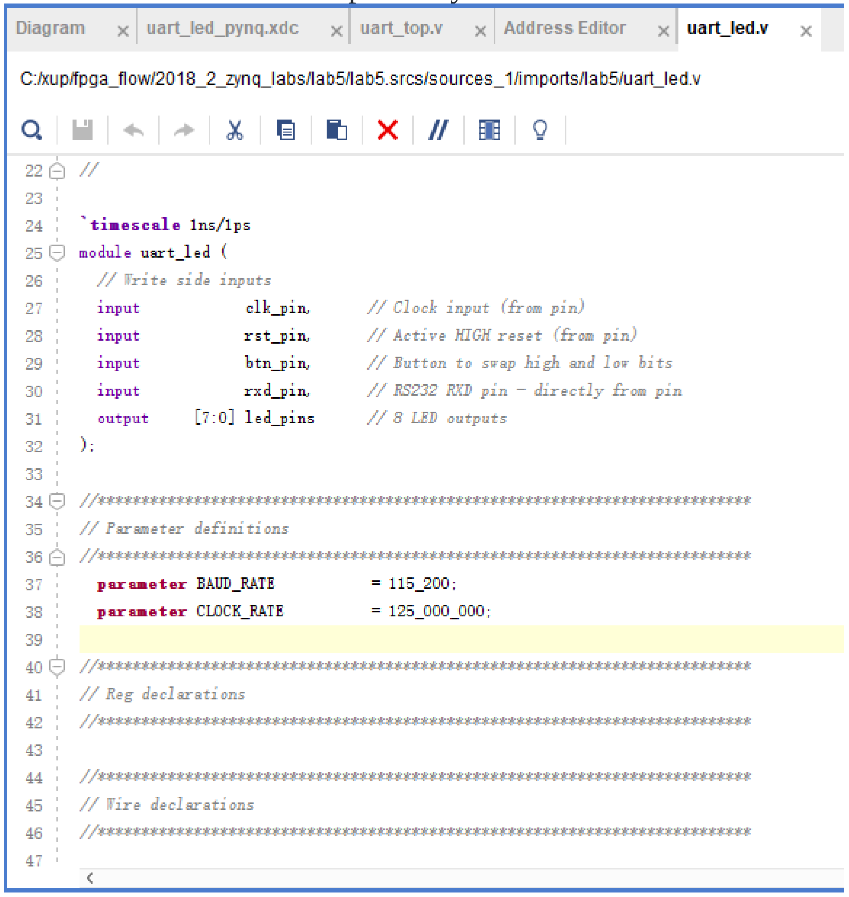


Diagram
uart_led_pynq.xdc
uart_top.v
Address Editor
uart_led.v
C:/xup/fpga_flow/2018_2_zynq_labs/lab5/lab5.srcs/sources_1/imports/lab5/uart_led.v

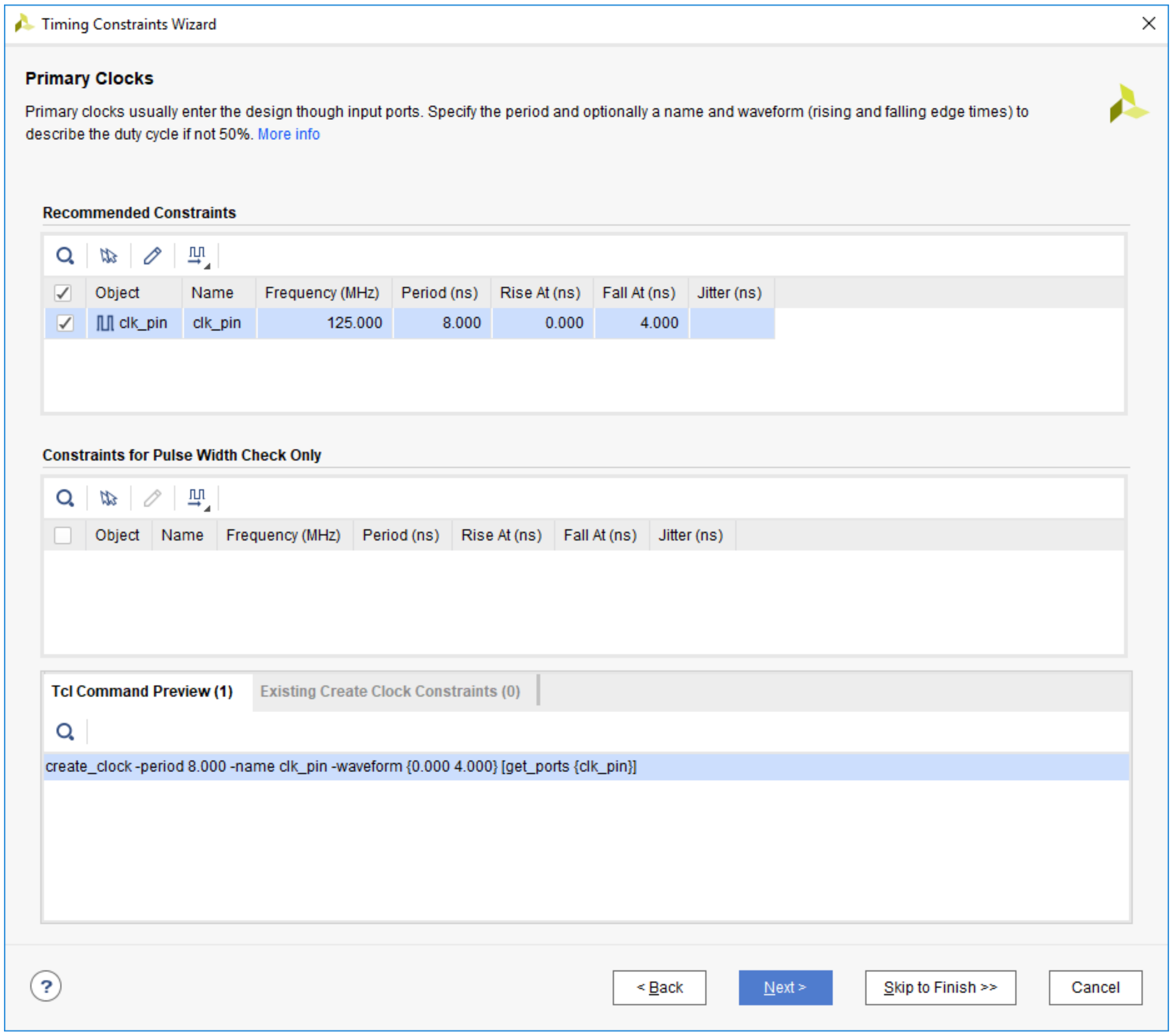
```
//

`timescale 1ns/1ps
module uart_led (
  // Write side inputs
  input            clk_pin,     // Clock input (from pin)
  input            rst_pin,     // Active HIGH reset (from pin)
  input            btn_pin,     // Button to swap high and low bits
  input            rxd_pin,     // RS232 RXD pin - directly from pin
  output     [7:0] led_pins     // 8 LED outputs
);

//***************************************************************************
// Parameter definitions
//***************************************************************************
  parameter BAUD_RATE           = 115_200;
  parameter CLOCK_RATE          = 125_000_000;

//***************************************************************************
// Reg declarations
//***************************************************************************

//***************************************************************************
// Wire declarations
//***************************************************************************
```

CLOCK_RATE parameter of uart_led

Synthesize and Enter Timing Constraints

Synthesize the design. Use the Constraints Wizard to specify a clock frequency, and input and output delay constraints.

1. Click on the **Run Synthesis** in the *Flow Navigator* pane.
   Click on the **Save** if Save project window appears.
   When synthesis is completed a form with three options will be displayed.
2. Select *Open Synthesized Design* and click **OK**.
3. In the *Flow Navigator* pane (under Open Synthesized Design), click on the **Constraints Wizard**. This will open up the Constraints Wizard.
4. Read the *Identify and Recommend Missing Timing Constraints* screen of the wizard to understand what the wizard does and click **Next**.
5. Specify the frequency of the object *clk_pin* to be **125 MHz,** notice the Period, Rise At and Fall At are automatically populated. Also notice the Tcl command that can be previewed at the bottom of the wizard. Click **Next** to proceed.


Timing Constraints Wizard

Primary Clocks

Primary clocks usually enter the design though input ports. Specify the period and optionally a name and waveform (rising and falling edge times) to describe the duty cycle if not 50%. More info

Recommended Constraints

✓	Object	Name	Frequency (MHz)	Period (ns)	Rise At (ns)	Fall At (ns)	Jitter (ns)
✓	clk_pin	clk_pin	125.000	8.000	0.000	4.000	

Constraints for Pulse Width Check Only

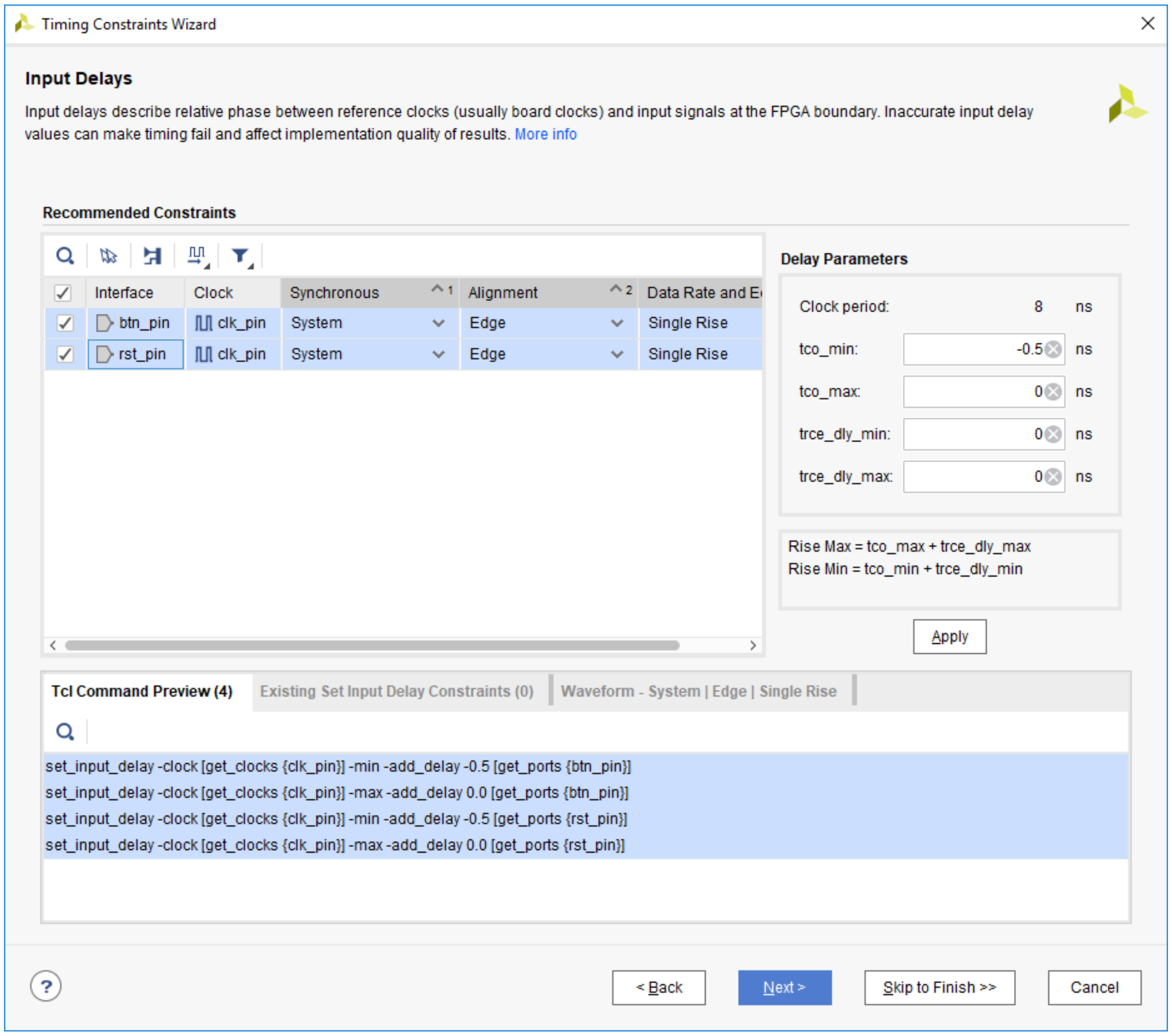
	Object	Name	Frequency (MHz)	Period (ns)	Rise At (ns)	Fall At (ns)	Jitter (ns)

Tcl Command Preview (1) | Existing Create Clock Constraints (0)

```
create_clock -period 8.000 -name clk_pin -waveform {0.000 4.000} [get_ports {clk_pin}]
```

Constraints Wizard clk_pin parameters and Tcl command

6. There are no missing Generated Clocks, click **Next** to proceed.
7. There are no missing Forwarded Clocks, click **Next** to proceed.
8. There are no missing External Feedback Delays, click **Next** to proceed.
9. The wizard identifies Input Delays needed for the *btn_pin* and *rst_pin* pins. Do the following:
    - Press Ctrl and select the two rows.
    - Enter the **tco_min** value to be **-0.5 ns** and everything else as **0 ns**. Click **Apply**.
    - Notice that under the Tcl Command Preview tab, 4 Tcl commands have been generated.
    - Click **Next**.



Input Delays

Input delays describe relative phase between reference clocks (usually board clocks) and input signals at the FPGA boundary. Inaccurate input delay values can make timing fail and affect implementation quality of results. More info

Recommended Constraints

✓	Interface	Clock	Synchronous	Alignment	Data Rate and E
✓	btn_pin	clk_pin	System	Edge	Single Rise
✓	rst_pin	clk_pin	System	Edge	Single Rise

Delay Parameters

Clock period:	8	ns
tco_min:	-0.5	ns
tco_max:	0	ns
trce_dly_min:	0	ns
trce_dly_max:	0	ns

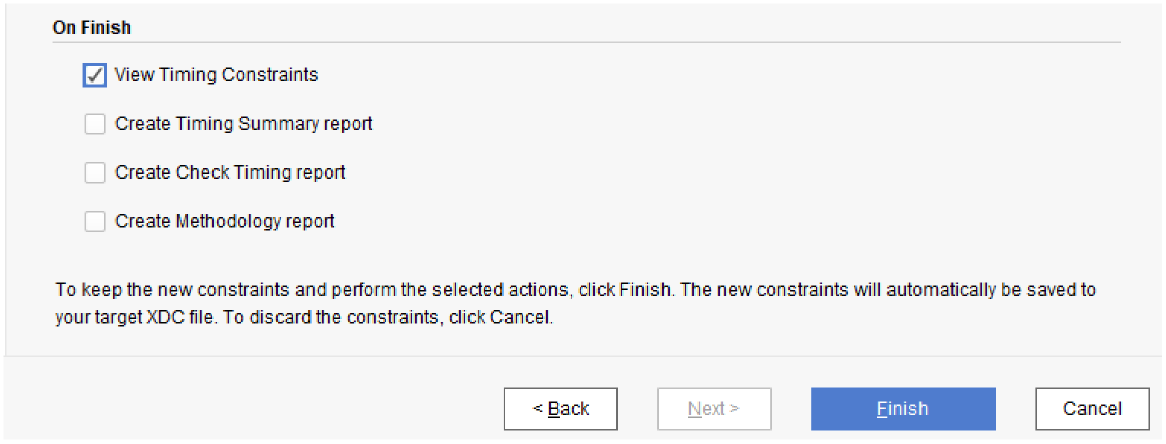
Rise Max = tco_max + trce_dly_max
Rise Min = tco_min + trce_dly_min

Apply
Tcl Command Preview (4)
Existing Set Input Delay Constraints (0)
Waveform - System | Edge | Single Rise

```
set_input_delay -clock [get_clocks {clk_pin}] -min -add_delay -0.5 [get_ports {btn_pin}]
set_input_delay -clock [get_clocks {clk_pin}] -max -add_delay 0.0 [get_ports {btn_pin}]
set_input_delay -clock [get_clocks {clk_pin}] -min -add_delay -0.5 [get_ports {rst_pin}]
set_input_delay -clock [get_clocks {clk_pin}] -max -add_delay 0.0 [get_ports {rst_pin}]
```

Specifying Input Delays for btn_pin and rst_pin

10. Enter the tsu and thd as **0 ns** and Enter the trce_dly_max and trce_dly_min as **-2.20 ns.** Click **Apply** and then click **Next**.
11. There are no Combinatorial Delays identified, click **Next** to proceed.
12. Click **Skip to Finish** to skip to the final Constraints Summary page. Read the description of each page.
13. **Check** *On Finish* – **View Timing Constraints** and click **Finish** to close the wizard. The option will open the Timing Constraints Editor to show you the generated timing constraint.



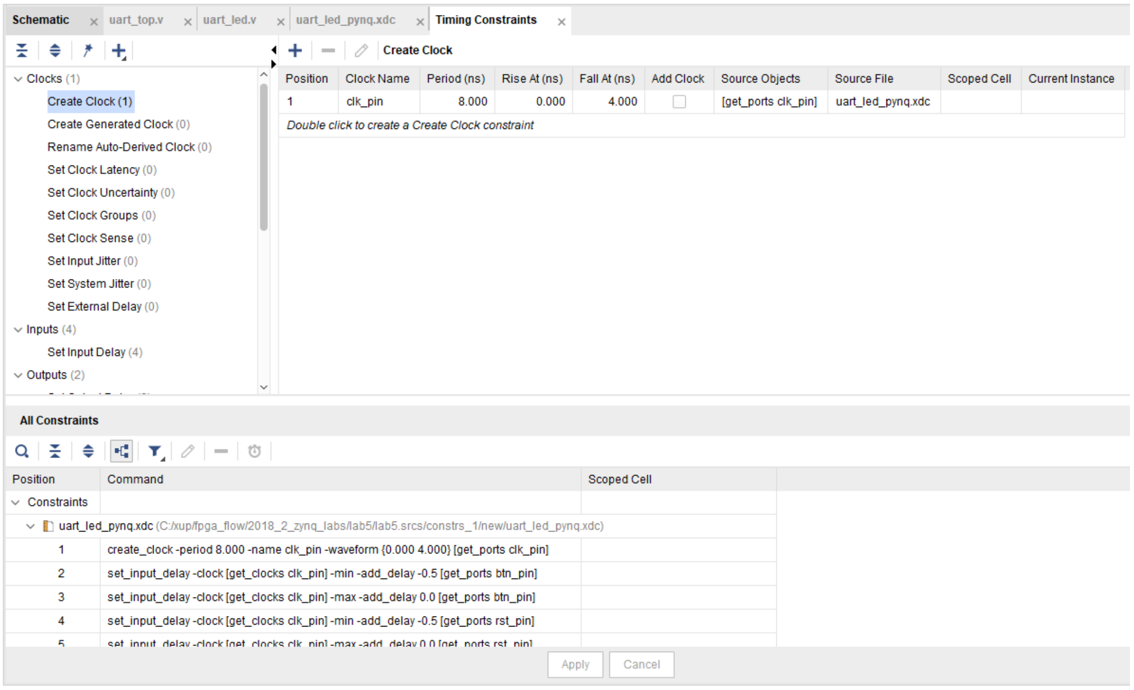
On Finish

- [✓] View Timing Constraints
- [ ] Create Timing Summary report
- [ ] Create Check Timing report
- [ ] Create Methodology report

To keep the new constraints and perform the selected actions, click Finish. The new constraints will automatically be saved to your target XDC file. To discard the constraints, click Cancel.

Selecting View Timing constraints

14. Note the wizard generated the clk_pin constraint for a 8 ns period (or 125 MHz). Notice in the All Constraints window, 7 constraints will be created.
    There is no need to click Apply since the constraints have already been applied in the Constraints Wizard.



Create Clock

Position	Clock Name	Period (ns)	Rise At (ns)	Fall At (ns)	Add Clock	Source Objects	Source File	Scoped Cell	Current Instance
1	clk_pin	8.000	0.000	4.000	☐	[get_ports clk_pin]	uart_led_pynq.xdc		
Double click to create a Create Clock constraint									

- Clocks (1)
  - Create Clock (1)
  - Create Generated Clock (0)
  - Rename Auto-Derived Clock (0)
  - Set Clock Latency (0)
  - Set Clock Uncertainty (0)
  - Set Clock Groups (0)
  - Set Clock Sense (0)
  - Set Input Jitter (0)
  - Set System Jitter (0)
  - Set External Delay (0)
- Inputs (4)
  - Set Input Delay (4)
- Outputs (2)

All Constraints

Position	Command	Scoped Cell
Constraints		
uart_led_pynq.xdc (C:/xup/fpga_flow/2018_2_zynq_labs/lab5/lab5.srcs/constrs_1/new/uart_led_pynq.xdc)		
1	create_clock -period 8.000 -name clk_pin -waveform {0.000 4.000} [get_ports clk_pin]	
2	set_input_delay -clock [get_clocks clk_pin] -min -add_delay -0.5 [get_ports btn_pin]	
3	set_input_delay -clock [get_clocks clk_pin] -max -add_delay 0.0 [get_ports btn_pin]	
4	set_input_delay -clock [get_clocks clk_pin] -min -add_delay -0.5 [get_ports rst_pin]	
5	set_input_delay -clock [get_clocks clk_pin] -max -add_delay 0.0 [get_ports rst_pin]	

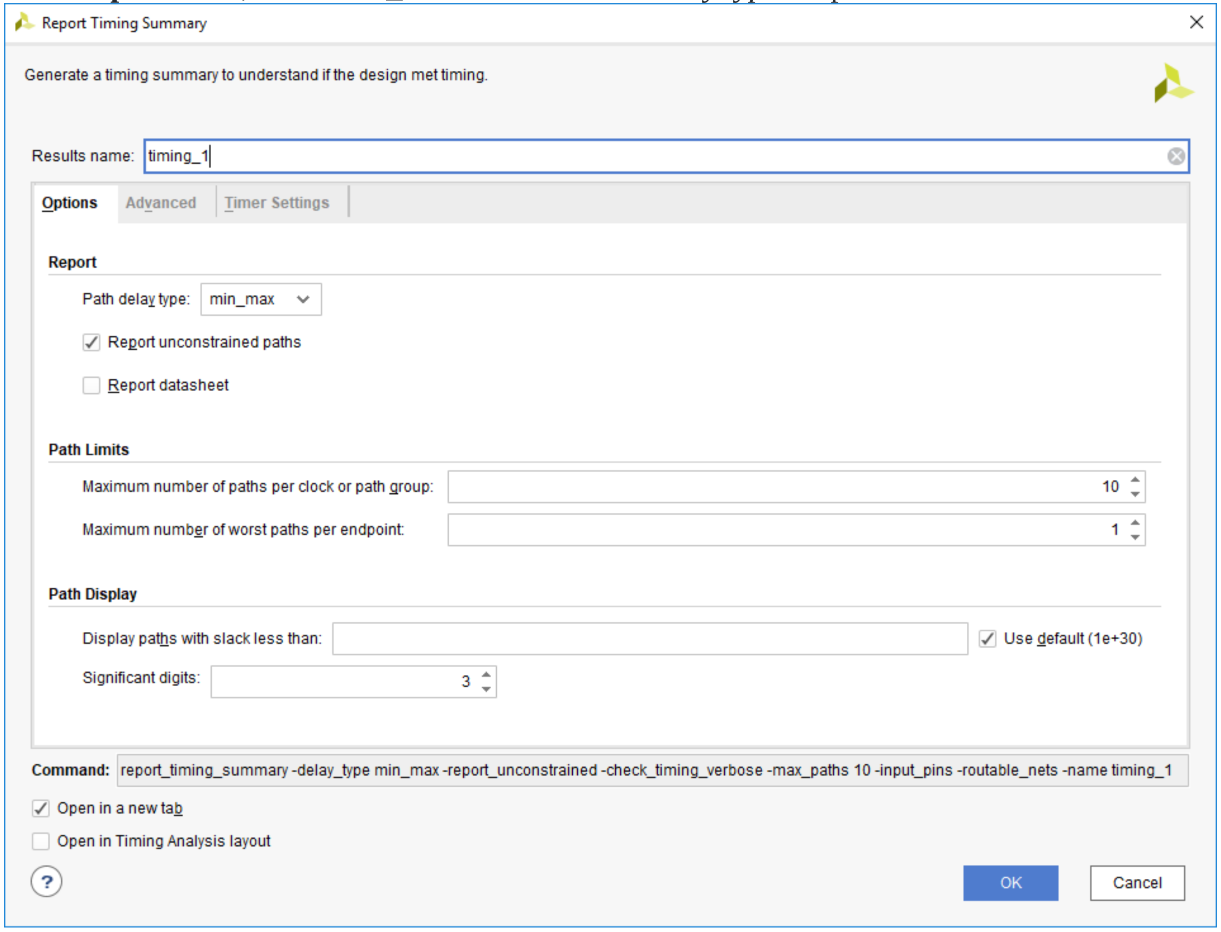
The constraints added after using the Constraints Wizard

15. Open uart_led_pynq.xdc (if it was already opened, click Reload in the yellow status bar) and notice additional constraints were added to the last line of the file.

Generate an estimated Timing Report showing both the setup and hold paths in the design.

1. Select **Flow Navigator > SYNTHESIS > Open Synthesized Design > Report Timing Summary**.

2. In the **Options** tab, select *min_max* from the *Path delay type* drop-down list.



Report Timing Summary

Generate a timing summary to understand if the design met timing.

Results name: timing_1

Options | Advanced | Timer Settings

Report

Path delay type: min_max

[✓] Report unconstrained paths

[ ] Report datasheet

Path Limits

Maximum number of paths per clock or path group: 10

Maximum number of worst paths per endpoint: 1

Path Display

Display paths with slack less than: [✓] Use default (1e+30)

Significant digits: 3

Command:

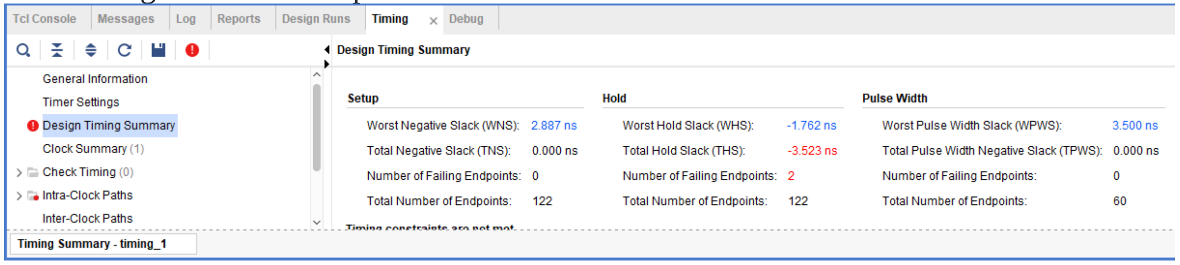
```
report_timing_summary -delay_type min_max -report_unconstrained -check_timing_verbose -max_paths 10 -input_pins -routable_nets -name timing_1
```

[✓] Open in a new tab

[ ] Open in Timing Analysis layout

Performing timing analysis

3. Click **OK** to run the analysis.
   The Timing Results view opens at the bottom of the Vivado IDE.



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.887 ns	Worst Hold Slack (WHS):	-1.762 ns	Worst Pulse Width Slack (WPWS):	3.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	-3.523 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	2	Number of Failing Endpoints:	0
Total Number of Endpoints:	122	Total Number of Endpoints:	122	Total Number of Endpoints:	60

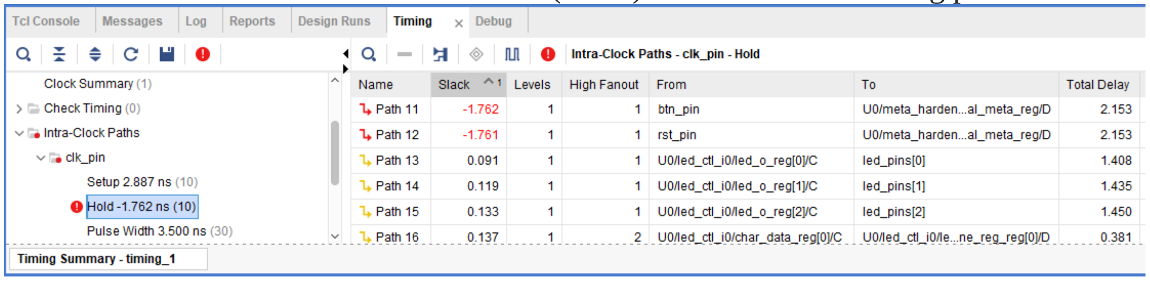
Timing constraints are not met.

Timing summary

The *Design Timing Summary* report provides a brief worst Setup and Hold slack information and Number of failing endpoints to indicate whether the design has met timing or not.

Note that there are three timing failures under the hold check.

4. Click on the link next to *Worst Hold Slack* (WHS) to see the list of failing paths.



Tcl Console | Messages | Log | Reports | Design Runs | Timing | Debug

Clock Summary (1)
Check Timing (0)
Intra-Clock Paths
clk_pin
Setup 2.887 ns (10)
Hold -1.762 ns (10)
Pulse Width 3.500 ns (30)

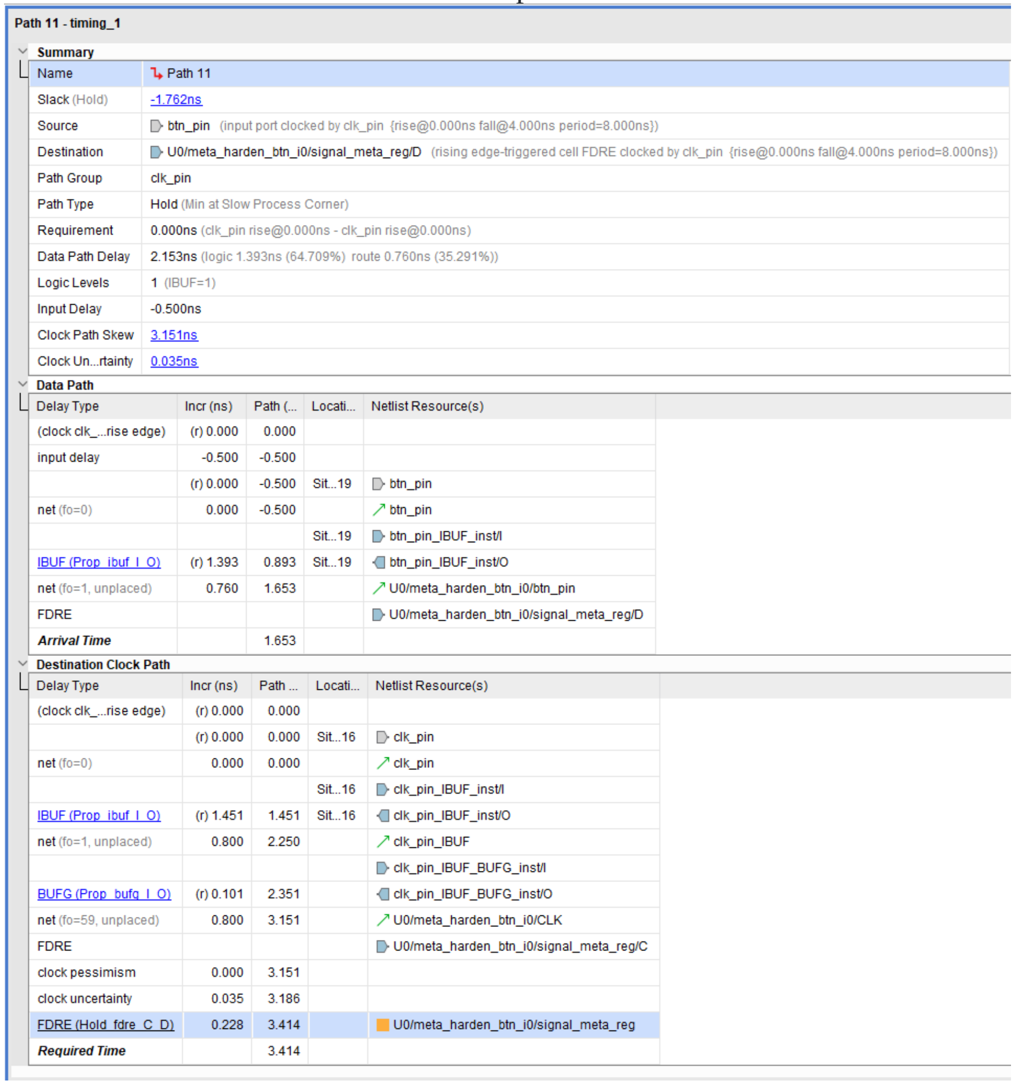
Timing Summary - timing_1

Intra-Clock Paths - clk_pin - Hold

Name	Slack ^1	Levels	High Fanout	From	To	Total Delay
Path 11	-1.762	1	1	btn_pin	U0/meta_harden...al_meta_reg/D	2.153
Path 12	-1.761	1	1	rst_pin	U0/meta_harden...al_meta_reg/D	2.153
Path 13	0.091	1	1	U0/led_ctl_i0/led_o_reg[0]/C	led_pins[0]	1.408
Path 14	0.119	1	1	U0/led_ctl_i0/led_o_reg[1]/C	led_pins[1]	1.435
Path 15	0.133	1	1	U0/led_ctl_i0/led_o_reg[2]/C	led_pins[2]	1.450
Path 16	0.137	1	2	U0/led_ctl_i0/char_data_reg[0]/C	U0/led_ctl_i0/le...ne_reg_reg[0]/D	0.381

The list of paths showing hold violations

5. Double-click on the *Path 11* to see the actual path detail.



Path 11 - timing_1

Summary

Name	Path 11
Slack (Hold)	-1.762ns
Source	btn_pin (input port clocked by clk_pin {rise@0.000ns fall@4.000ns period=8.000ns})
Destination	U0/meta_harden_btn_i0/signal_meta_reg/D (rising edge-triggered cell FDRE clocked by clk_pin {rise@0.000ns fall@4.000ns period=8.000ns})
Path Group	clk_pin
Path Type	Hold (Min at Slow Process Corner)
Requirement	0.000ns (clk_pin rise@0.000ns - clk_pin rise@0.000ns)
Data Path Delay	2.153ns (logic 1.393ns (64.709%) route 0.760ns (35.291%))
Logic Levels	1 (IBUF=1)
Input Delay	-0.500ns
Clock Path Skew	3.151ns
Clock Un...rtainty	0.035ns

Data Path

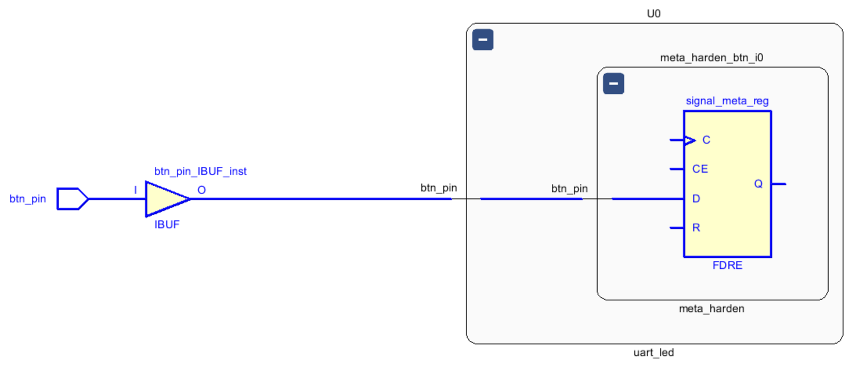
Delay Type	Incr (ns)	Path (...	Locati...	Netlist Resource(s)
(clock clk_...rise edge)	(r) 0.000	0.000		
input delay	-0.500	-0.500		
	(r) 0.000	-0.500	Sit...19	btn_pin
net (fo=0)	0.000	-0.500		btn_pin
			Sit...19	btn_pin_IBUF_inst/I
IBUF (Prop_ibuf_I_O)	(r) 1.393	0.893	Sit...19	btn_pin_IBUF_inst/O
net (fo=1, unplaced)	0.760	1.653		U0/meta_harden_btn_i0/btn_pin
FDRE				U0/meta_harden_btn_i0/signal_meta_reg/D
Arrival Time		1.653		

Destination Clock Path

Delay Type	Incr (ns)	Path ...	Locati...	Netlist Resource(s)
(clock clk_...rise edge)	(r) 0.000	0.000		
	(r) 0.000	0.000	Sit...16	clk_pin
net (fo=0)	0.000	0.000		clk_pin
			Sit...16	clk_pin_IBUF_inst/I
IBUF (Prop_ibuf_I_O)	(r) 1.451	1.451	Sit...16	clk_pin_IBUF_inst/O
net (fo=1, unplaced)	0.800	2.250		clk_pin_IBUF
				clk_pin_IBUF_BUFG_inst/I
BUFG (Prop_bufg_I_O)	(r) 0.101	2.351		clk_pin_IBUF_BUFG_inst/O
net (fo=59, unplaced)	0.800	3.151		U0/meta_harden_btn_i0/CLK
FDRE				U0/meta_harden_btn_i0/signal_meta_reg/C
clock pessimism	0.000	3.151		
clock uncertainty	0.035	3.186		
FDRE (Hold_fdre_C_D)	0.228	3.414		U0/meta_harden_btn_i0/signal_meta_reg
Required Time		3.414		

Failing hold path

6. Select *Path 11*, right-click and select **Schematic**.



The schematic of the failing path

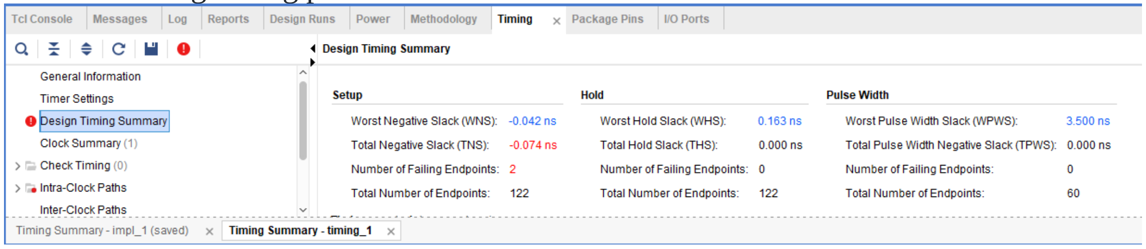
Implement and Analyze Timing Summary

Implement the design.

1. Click on the **Run Implementation**.
2. Click **Yes** to run the synthesis first before running the implementation process. When the implementation is completed, a dialog box will appear with three options.
3. Select the *Open Implemented Design* option and click **OK**.
4. Click *Yes* if you are prompted to close the synthesized design.

Generate a timing summary report

1. Select **Flow Navigator > IMPLEMENTATION > Open Implemented Design > Report Timing Summary**.
2. Click **OK** to generate the report using the default settings. The *Design Timing Summary* window opens at the bottom in the Timing tab. Note that failing timing paths are indicated in red.



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-0.042 ns	Worst Hold Slack (WHS):	0.163 ns	Worst Pulse Width Slack (WPWS):	3.500 ns
Total Negative Slack (TNS):	-0.074 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	2	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	122	Total Number of Endpoints:	122	Total Number of Endpoints:	60

Failing setup paths

3. Click on the *WNS* to see the failing paths.
4. Double-click on the first failing path from the top and see the detailed analysis. The output path delay can be reduced by placing the register in IOB.
5. Apply the constraint by typing the following two commands in the Tcl console.
   set_output_delay -clock [get_clocks clk_pin] -min -add_delay -2.250 [get_ports {led_pins[*]}]
   set_output_delay -clock [get_clocks clk_pin] -max -add_delay -2.250 [get_ports {led_pins[*]}]

6. Select **File > Constraints > Save**. Click **OK** at the warning message. Click **Yes** to save the project.
7. Click on **Run Implementation**.
8. Click **Yes** to reset the synthesis run, perform the synthesis, and run implementation.
9. Open the implemented design and observe that the number of failing paths in the Design Runs tab reported is 0.
10. Click Report Timing Summary, and observe that there are no failing paths.

Generate the Bitstream and Verify the Functionality

Generate the bitstream.

1. Click **low Navigator > PROGRAM AND DEBUG > Generate Bitstream**.
2. The write_bitstream command will be executed (you can verify it by looking in the Tcl console).
3. Click **Cancel** when the bitstream generation is completed.

Connect the board and power it ON. Open a hardware session, and program the FPGA.

1. Make sure that the Micro-USB cable is connected to the JTAG PROG connector.
2. Turn ON the power.
3. Select the *Open Hardware Manager* option.
   The Hardware Manager window will open indicating "unconnected" status.
4. Click on the **Open target** link, then **Auto Connect** from the dropdown menu.
   You can also click on the **Open recent target** link if the board was already targeted before.
5. The Hardware Manager status changes from Unconnected to the server name and the device is highlighted. Also notice that the Status indicates that it is not programmed.
6. Select the device and verify that the **ios.bit** is selected as the programming file in the General tab.

Start a terminal emulator program such as TeraTerm or HyperTerminal. Select an appropriate COM port (you can find the correct COM number using the Control Panel). Set the COM port for 115200 baud rate communication. Program the FPGA and verify the functionality.

1. Start a terminal emulator program such as TeraTerm or HyperTerminal.
2. Select an appropriate COM port (you can find the correct COM number using the Control Panel).
3. Set the COM port for 115200 baud rate communication.
4. Right-click on the FPGA entry in the Hardware window and select Programming Device…
5. Click on the **Program** button.
   The programming bit file be downloaded and the DONE light will be turned ON indicating the FPGA has been programmed.

Start a SDK session, point it to the c:/xup/fpga_flow/2018_2_zynq_sources/lab5/pynq/lab5.sdk workspace.

1. Open **SDK** by selecting **Start > Xilinx Design Tools > Xilinx SDK 2018.2**

2. In the **Select a workspace** window, click on the browse button, browse to *c:/xup/fpga_flow/2018_2_zynq_sources/lab5/pynq/lab5.sdk* and click **OK**.
3. Click **OK**.
   In the *Project Explorer*, right-click on the uart_led_zynq, select *Run As*, and then **Launch on Hardware (System Debugger)**
4. Verify the functionality as you did in the previous lab, by typing some characters into the terminal, and watching the corresponding values appear on the LEDs.
5. When satisfied, close the terminal emulator program and power OFF the board.
6. Select **File > Close Hardware Manager**. Click **OK** to close it.
7. When done, close the **Vivado** program by selecting **File > Exit** and click **OK**.
8. Close the **SDK** program by selecting **File > Exit** and click **OK**.

In this part, you learned how to create an I/O Planning project and assign the pins via the Device view, Package Pins tab, and the Tcl commands. You then exported to the rtl project where you added the provided source files. Next you created timing constraints and performed post-synthesis and post-implementation timing analysis.

References:

1. Arty Z7 Reference Manual https://digilent.com/reference/programmable-logic/arty-z7/reference-manual
2. https://xilinx-wiki.atlassian.net/wiki/spaces/A/overview
3. https://pynq.readthedocs.io/en/v2.4/pynq_overlays/partial_reconfiguration.html
4. https://digilent.com/reference/learn/programmable-logic/tutorials/arty-z7-hdmi-in-demo
5. https://digilent.com/reference/learn/programmable-logic/tutorials/arty-z7-hdmi-demo/start

Acknowledgement:

- Digilent
- Xilinx PYNQ project
- Xilinx University Program

Deliverables:

- Group Deliverables (Compile everything as a single pdf report besides the code):
  - Based on **part 2**, please submit
    - In **2.1**, please take a picture of **color bar and blended color** showing on the monitor.
    - In 2.1, please take a picture of the monitor screen out of step 7 and 8.
    - In 2.2, please take a picture of **color bar and blended color** showing on the monitor.
  - Based on part 3, please submit
    - your uart_led_pynq.xdc
    - Screenshot of the Timing Results view
    - Screenshot of the *Worst Hold Slack* (WHS) results
    - Screenshot of the Failing setup paths
    - Your Bitstream
  - Answer the following questions (based on your understanding, feel free to use the internet):
    - Why do we need to have XDC? How is it used by the tool?
    - If you push the clock frequency to very high, what will happen?
    - What can you do in the case of hold violations?
    - In the case of setup violation, please list at least 3 solutions to fix the violation.
- Individual Deliverables
  - Complete the peer-evaluation form appended in the end of this document

Grading Policy

Factor	**Percentage**
Part 2	20%
Part 3	55%
Questions	25%

ECE4810J System-on-Chip (SoC) Design

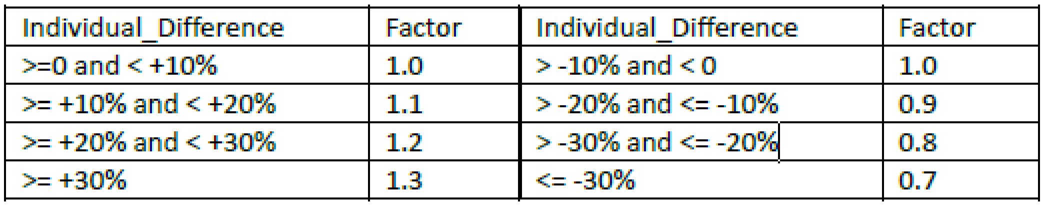
Fall 2021

Lab #5 Peer Evaluation Form

Each team member is required to provide a peer evaluation for the team effort of the lab. The score of the peer evaluation should be integers ranging between 0 to 5, inclusively, with 5 indicating the biggest contribution. A score should be given to each team member including yourself according the team member's contribution based on your observation. A brief description of specific contribution of each team member should also be provided. Note this form needs to be finished without discussing with your teammate, if all forms are found the same, then you have to justify how each member makes exactly the same contribution (through demonstration to the instructor).

Name	Level of contributions (0 – 5)	Description of contributions
(yourself)		
Team member 1		
Team member 2		
Team member 3		

Your lab grade is calculated based on the following:
Individual_Average = (sum of all team member's marks) / (size of the student team)
Group_Average = sum of Individual_Average of all team members / size of student team
Individual_Difference = Individual_Average / Group_Average – 1.0
Using the calculated Individual_Difference, we find a factor from the following lookup table.



Individual_Difference	Factor	Individual_Difference	Factor
>=0 and < +10%	1.0	> -10% and < 0	1.0
>= +10% and < +20%	1.1	> -20% and <= -10%	0.9
>= +20% and < +30%	1.2	> -30% and <= -20%	0.8
>= +30%	1.3	<= -30%	0.7

Your final lab grade is calculated by:
Final grade of lab = points for team effort * factor

Note that the final grade of the lab will not exceed the maximum points assigned to the project. If the peer-evaluation form is not submitted, then your individual_difference will be based on the available data (without yours) only, and your final score will be:

Final grade of lab = points for team effort * factor * 95%