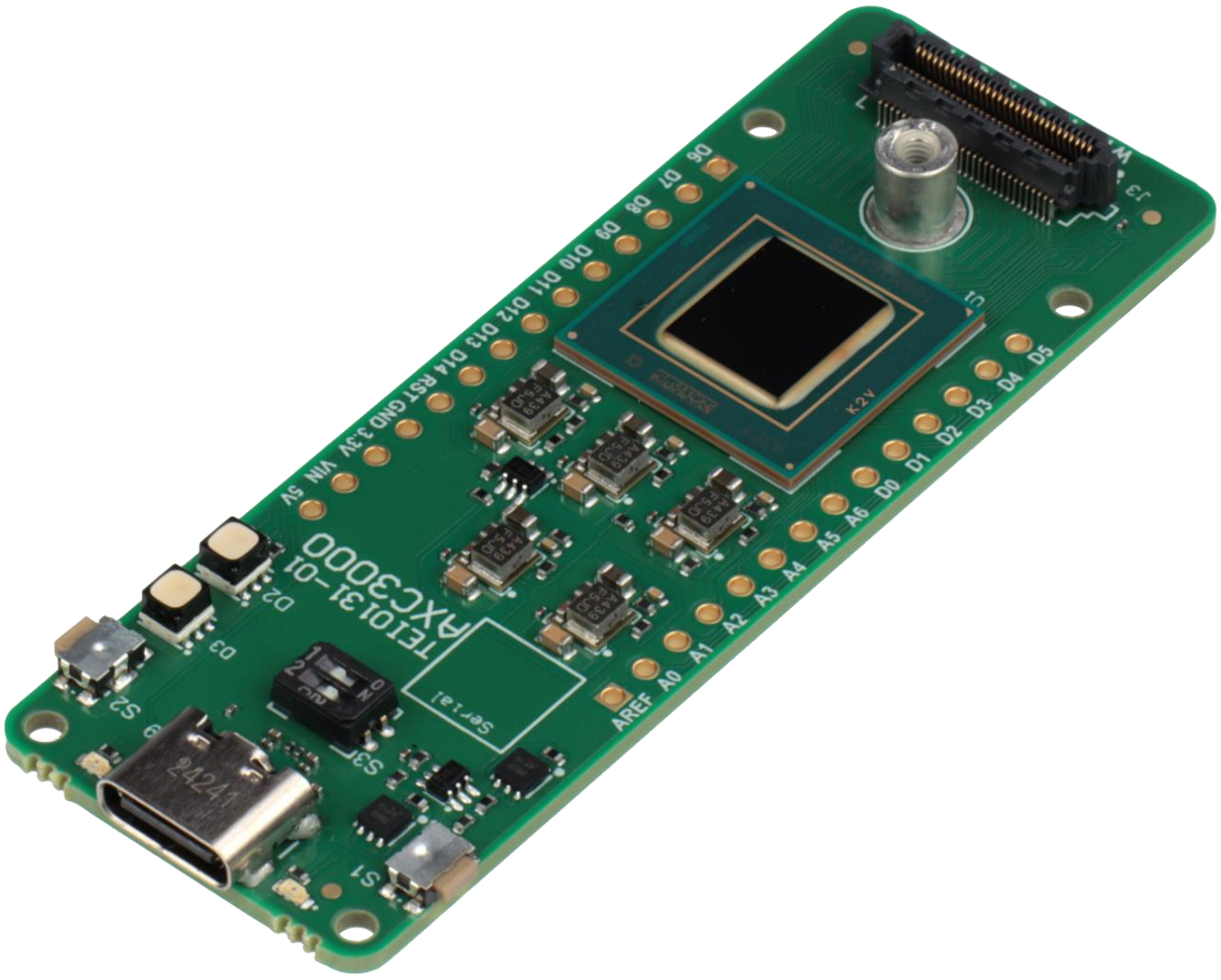
AXC3000 Signal Tap Logic Analyzer



Software and hardware requirements to complete all exercises.

Software Requirements: Quartus® Prime Pro version 25.1.1

Hardware Requirements: AXC3000 Agilex™ 3C Development Kit

Document Control

Document Version:	Version 1
Document Date:	09/04/2025
Document Author(s):	Steven Kravatsky, Erika Peter, Naji Naufel, , ESC Team
Document Classification:	Released
Document Distribution:	This document is still under development. All specifications, procedures, and processes described in this document are subject to change without prior notice
Prior Version History:	0.1

Please read the legal disclaimer at the end of this document.

Contents

1 Introduction

This tutorial provides comprehensive information to help you understand how to create and run the Signal Tap Logic Analyzer on the Arrow AXC3000 development board. The Signal Tap Logic Analyzer is a powerful tool that lets you capture and display signals in real time without requiring external equipment or adding extra I/O pins in the design. The signals that can be captured and displayed, are internal nodes and pins.

This lab will not make you an expert, but at the end, you will understand how to set and use the Signal Tap Logic analyzer within Quartus Prime Pro.

1.1 Objective

1. Create a simple Design
    a. Create IPs and use a top-level file
    b. Assign timing constraints
    c. Assign pins
2. Configure the Signal Tap Logic Analyzer GUI.
    a. Configure the Signal Tap Analyzer
    b. Compile the design
3. Download and Run the Signal Tap configuration on the Arrow AX3000 dev board.

Lab Notes: Many of the names that the lab asks you to use for files, components, and other objects in this exercise must be spelled exactly as directed. This nomenclature is necessary because the pre-written files includes variables that use certain names. Naming the components differently can cause errors.

This lab can be compiled on a local machine or on a remote machine hosted by CloudLabs. Both options allow for the lab to be tested on a local AXC3000 board.

In addition, when using CloudLabs, testing can also be performed using a remote board in the Arrow board farm. Both options are documented.

2 Getting Started

The first objective is to ensure that you have all the necessary software installed so that the lab can be completed successfully. Below is a list of items required to complete this lab. A number of options are provided.

Option 1: Completing the lab using CloudLabs tools and the board farm.

- Personal computer or laptop running 64-bit Linux / Windows 10 or later with at least an Intel i3 core (or equivalent), 8GB RAM.

Option 2: Completing the lab using CloudLabs tools and local hardware.

- AXC3000 Board
- USB C cable
- Quartus Prime 25.1.1 Pro Standalone Programmer.
- Personal computer or laptop running 64-bit Linux / Windows 10 or later with at least an Intel i3 core (or equivalent), 8GB RAM.
- Terminal emulator (Putty or Tera term)

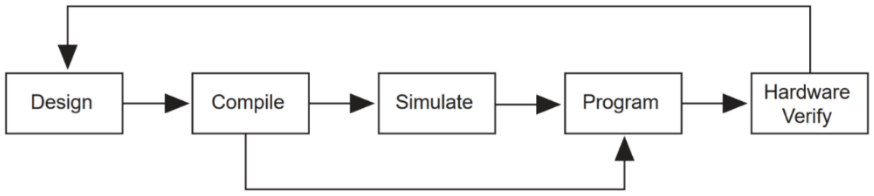
Option 3: Completing the lab using a laptop and local hardware.

- AXC3000 Board
- USB C cable
- Quartus Prime 25.1.1 Pro
- Ashling RiscFree IDE
- Personal computer or laptop running 64-bit Linux / Windows 10 or later with at least an Intel i3 core (or equivalent), 8GB RAM.
- Terminal emulator (Putty or Tera term)

A desire to learn!

3 Design Flow

The design flow for FPGA is



This workshop will focus on the Hardware Verify (Debug) Step, where the SignalTap Logic Analyzer is covered.

In this workshop, before the debugging can occur, Design Entry including Constraints, Compile and Program need to be done first. Simulation will not be covered in this workshop.

Details on the above flow chart are:

Design Specification, which is needed for every FPGA design, is pre-defined for this workshop. In this case, it is an UART TX design.

Design Entry includes Hardware Description Language (HDL), such as Verilog HDL, System Verilog, and VHDL, as well as Platform Designer files that contain IP (intellectual property) components and/or a combination of the files. In the following steps, you will create a portion of the digital design by implementing and configuring various IPs.

Constraints, which are also part of the Design Entry stage, are where pin assignments (pin assignments need to match to the FPGA on the board that the design is being programmed into), I/O standards and timing constraints are assigned.

Compilation is where the Quartus software checks the design for syntax errors, illegal assignments and/or other issues. It creates and uses an internal database to generate programming files.

Programming is where an output file is generated and downloaded into the FPGA. Agilex 3 devices are SRAM based devices, which means that when the power is removed, the device will need to be re-programmed. The programming in this workshop is done via JTAG but for production designs, the device can be programmed via an external memory device.

Debugging (Verification) are the steps where the design is checked to see if it works as expected. If the design does not work as expected an investigation needs to occur i.e. looking at timing, using the built-in Signal Tap Logic Analyzer etc. The SignalTap Logic Analyzer is the focus of this workshop.

The Signal Tap Embedded Logic Analyzer, which is provided in the Quartus Prime Pro software, allows you to analyze and verify designs. It is a powerful tool that lets you capture signals from internal FPGA nodes while the device is running in system. This gives you non-intrusive access to signals from internal device nodes. Neither extra I/O pins on the FPGA nor external lab equipment are needed.

The USB Blaster III, which is the onboard programming interface of the AXC3000, provides analysis support and allows you to transfer signal data to the Quartus Prime Pro software. Using the Signal Tap GUI, you can select signals, set up triggering events, configure memory, and display waveforms in real time. You can also use data retrieved by Signal Tap to debug and verify your design.

Overall, Quartus Prime Pro design software provides a complete FPGA design environment because it supports design, constraints, compilation, simulation, programming and debugging.

4 Hardware Project with AXC3000 board

This section gives instructions for creating the hardware project in Quartus® Prime Pro 25.1.1

4.1 New Quartus Prime project

The hardware implementation starts with a Quartus® Prime project. A Quartus project defines the location for the HDL and IP files used, target family, target device, and initial settings. After the initial creation of the project, changes can be made. Quartus uses the project location for the generated files.

4.1.1 New project creation

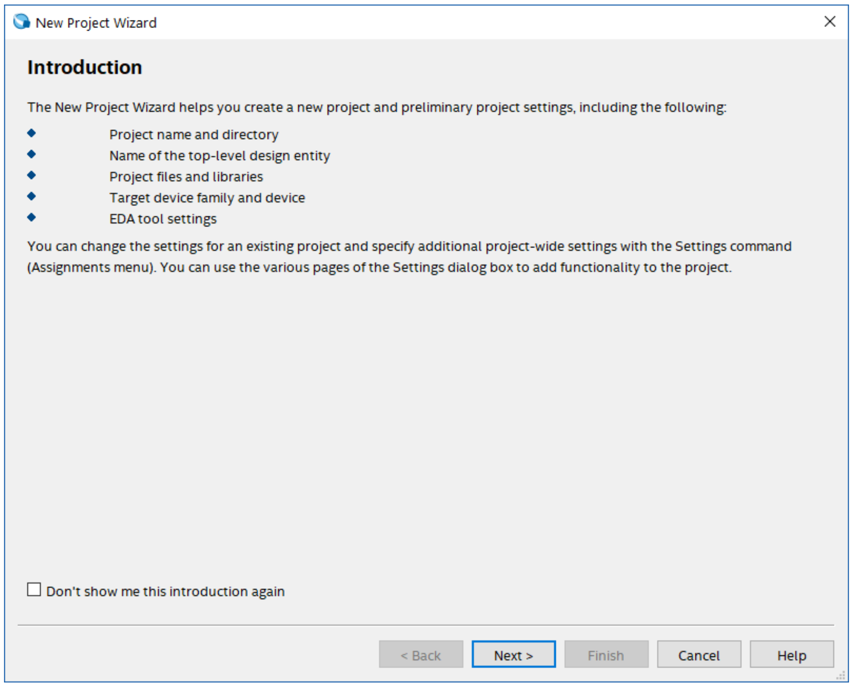
There are some initial settings that will need to be filled in when creating a new project. Quartus will use these settings for the initial setup. The following steps will take you through the Project setup.

4.1.1.1 Open Quartus Pro 25.1.1

If not already open, from the Start menu or the Desktop, open **Quartus Prime Pro 25.1.1** Quartus Prime Pro is needed because it supports Agilex 3 devices.

4.1.1.2 Create a new project

Using the New Project Wizard: **File → New Project Wizard**. This screen shows an introduction window. Click **Next.**


New Project Wizard

Introduction

The New Project Wizard helps you create a new project and preliminary project settings, including the following:

- Project name and directory
- Name of the top-level design entity
- Project files and libraries
- Target device family and device
- EDA tool settings

You can change the settings for an existing project and specify additional project-wide settings with the Settings command (Assignments menu). You can use the various pages of the Settings dialog box to add functionality to the project.

☐ Don't show me this introduction again

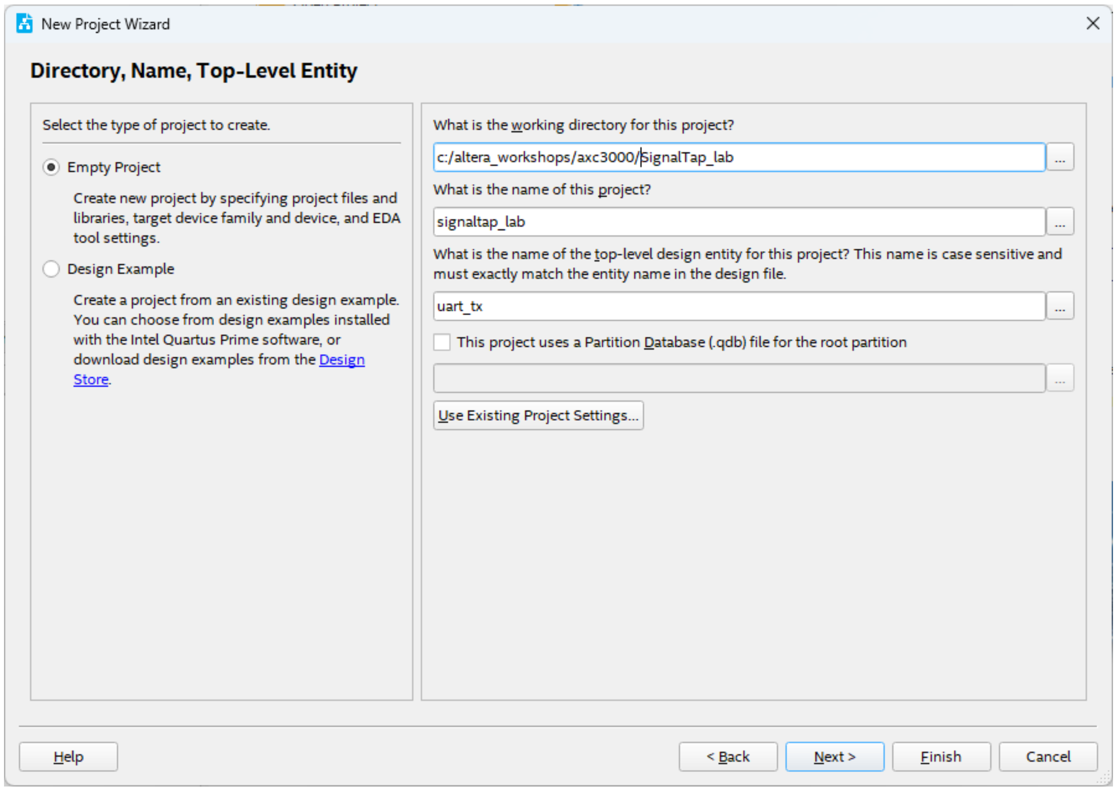
4.1.1.3 Configure the New Project Wizard

Specify project directory, name, and top-level entity information:

Note: No spaces are allowed in path or filenames.

- Ensure that the **Empty Project** radio button is selected on the left side of the window. This will be a new project.
- Type a directory in which you will store your Quartus project files for this design **c:/altera_workshops/axc3000/AXC3_SignalTap_lab**

  The directory will be created for you in the next step.
- Specify the name of the project: **signaltap_lab**
- Specify the name of the top-level entity: **uart_tx**


New Project Wizard

Directory, Name, Top-Level Entity

Select the type of project to create.

(●) Empty Project

Create new project by specifying project files and libraries, target device family and device, and EDA tool settings.

( ) Design Example

Create a project from an existing design example. You can choose from design examples installed with the Intel Quartus Prime software, or download design examples from the Design Store.

What is the working directory for this project?
c:/altera_workshops/axc3000/SignalTap_lab

What is the name of this project?
signaltap_lab

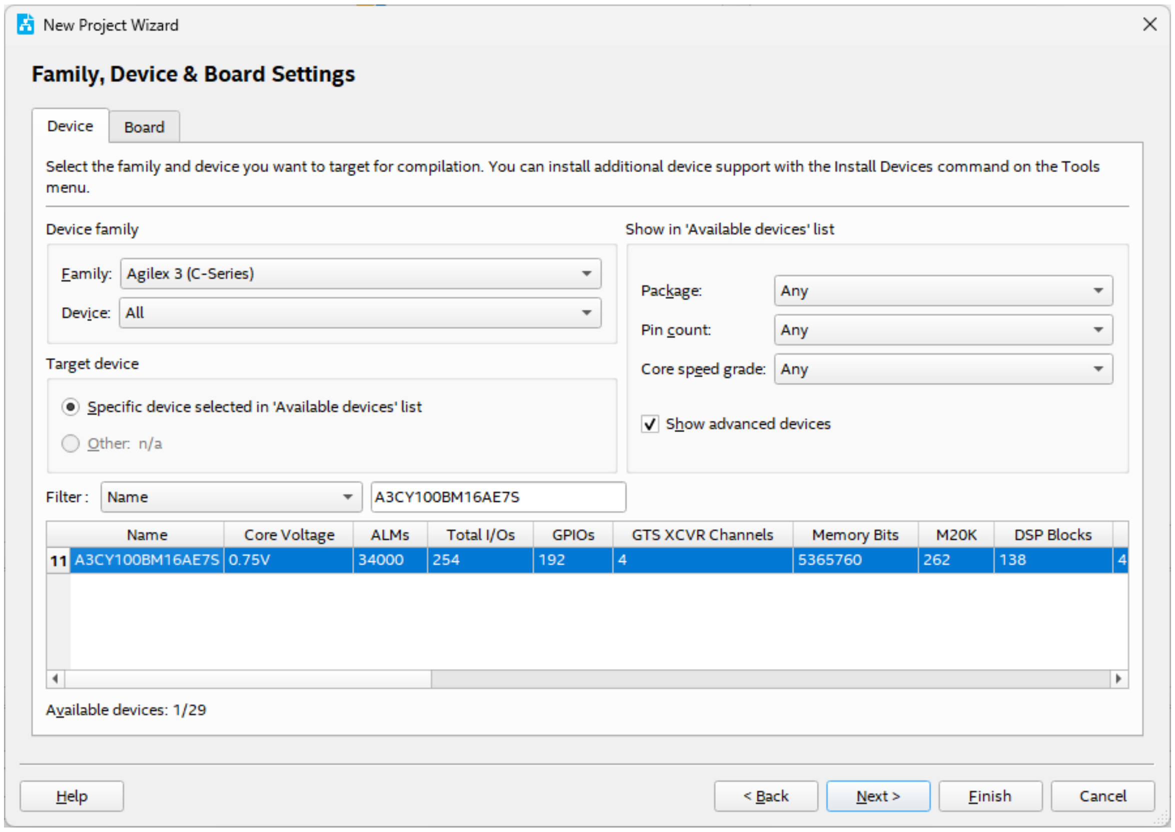
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.
uart_tx

[ ] This project uses a Partition Database (.qdb) file for the root partition

Click **Next**.

4.1.1.4 Specify Family and Device Settings

In the Family field, select ***Agilex 3 (C-series)*** and enter the part number (**A3CY100BM16AE7S**) in the Name Filter text box and select it. The selected family and part number in Quartus need to correspond to the device on the AXC3000 board to avoid programming incompatibility. The part number, with extension of R0, is the engineering sample FPGA that is used for the workshop. It is very important to ensure that part number is highlighted (in blue) or else another (default) part number will be used.


New Project Wizard

Family, Device & Board Settings

Device | Board

Select the family and device you want to target for compilation. You can install additional device support with the Install Devices command on the Tools menu.

Device family

Family: Agilex 3 (C-Series)

Device: All

Show in 'Available devices' list

Package: Any

Pin count: Any

Core speed grade: Any

✓ Show advanced devices

Target device

● Specific device selected in 'Available devices' list

○ Other: n/a

Filter: Name — A3CY100BM16AE7S

	Name	Core Voltage	ALMs	Total I/Os	GPIOs	GTS XCVR Channels	Memory Bits	M20K	DSP Blocks
11	A3CY100BM16AE7S	0.75V	34000	254	192	4	5365760	262	138

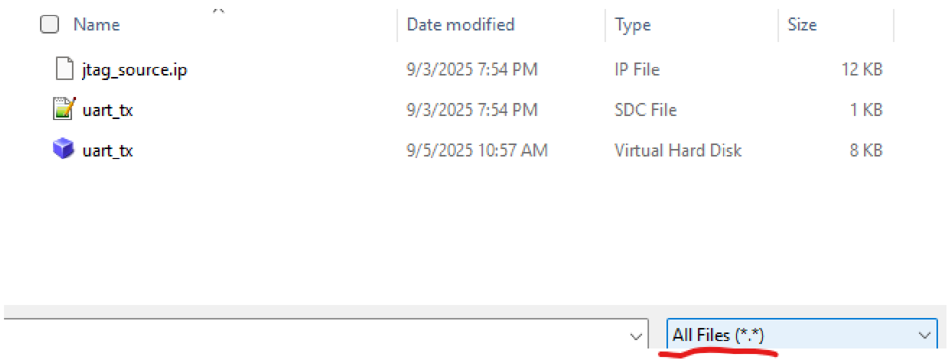
Available devices: 1/29

4.1.1.5 Click Next.

4.1.1.6 Add Files to the Project

Go back to the Quartus -> Add Files section and select the … by the file name, change to the C:/altera_workshops/axc3000/SignalTap_lab and select the files called uart_tx.sdc (timing constraints), uart_tx.vhd (RTL file), and jtag_source.ip

Note by default, you will not all the files unless you select all Files bottom right of the window

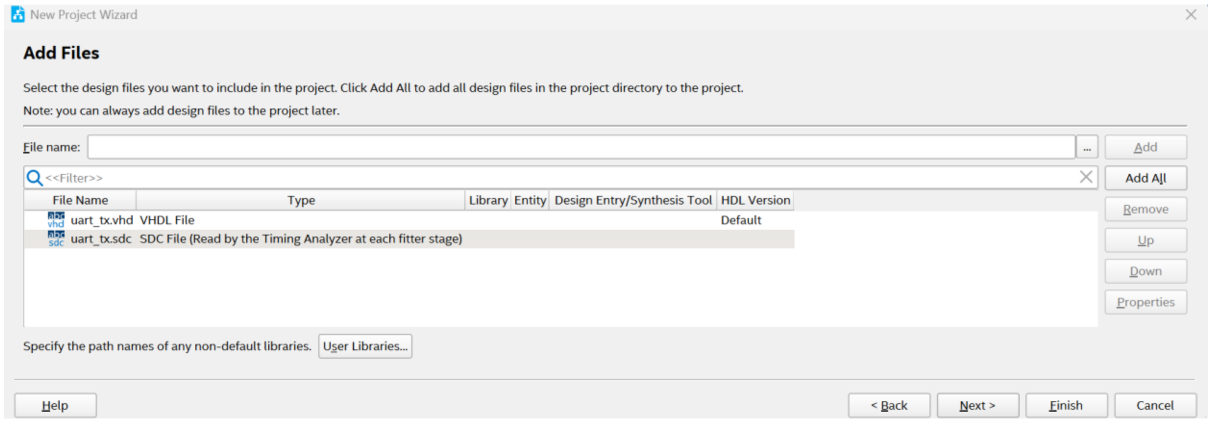


Name	Date modified	Type	Size
jtag_source.ip	9/3/2025 7:54 PM	IP File	12 KB
uart_tx	9/3/2025 7:54 PM	SDC File	1 KB
uart_tx	9/5/2025 10:57 AM	Virtual Hard Disk	8 KB

All Files (*.*)

Select **Open.**

The files will then be seen as



New Project Wizard

Add Files

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.
Note: you can always add design files to the project later.

File name:

File Name	Type	Library	Entity	Design Entry/Synthesis Tool	HDL Version
uart_tx.vhd	VHDL File				Default
uart_tx.sdc	SDC File (Read by the Timing Analyzer at each fitter stage)				

Specify the path names of any non-default libraries.

4.1.1.7 Click Next.

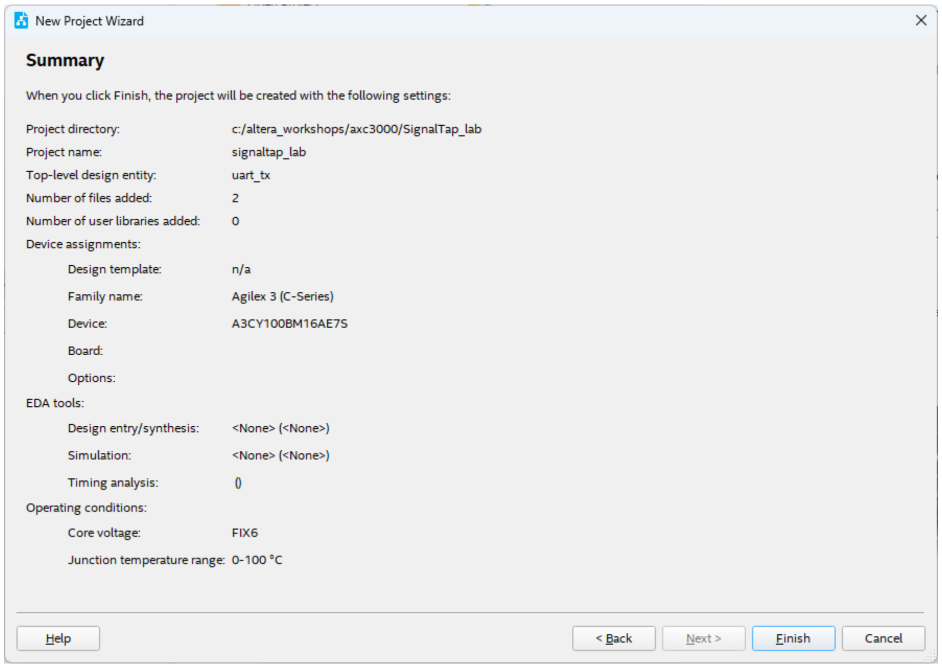
4.1.1.8 EDA Tools Settings

On the EDA Tools Settings page, select **Next**. EDA tools are third party software such as simulation, board level, etc and they will not be used here.

4.1.1.9 Click Next.

4.1.1.10 Summary Screen

The Summary screen will show the project and file locations. These settings can be changed if needed, later.



New Project Wizard

Summary

When you click Finish, the project will be created with the following settings:

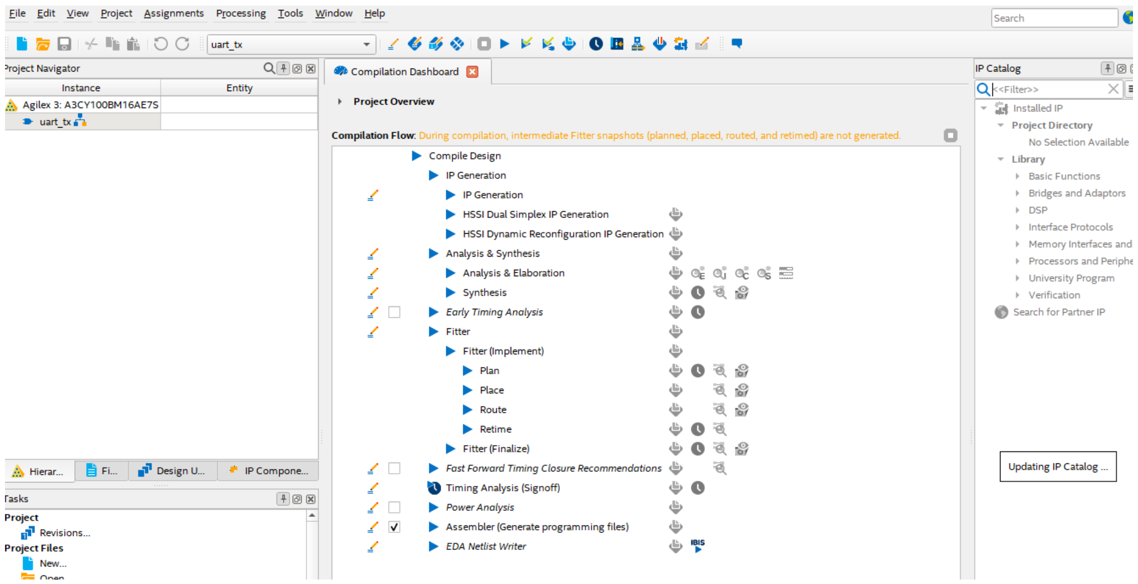
Project directory:	c:/altera_workshops/axc3000/SignalTap_lab
Project name:	signaltap_lab
Top-level design entity:	uart_tx
Number of files added:	2
Number of user libraries added:	0
Device assignments:	
Design template:	n/a
Family name:	Agilex 3 (C-Series)
Device:	A3CY100BM16AE7S
Board:	
Options:	
EDA tools:	
Design entry/synthesis:	<None> (<None>)
Simulation:	<None> (<None>)
Timing analysis:	0
Operating conditions:	
Core voltage:	FIX6
Junction temperature range:	0-100 °C

4.1.1.11 Click Finish.

4.2 Design Entry

In this module you will use IP Catalog to create various files.

4.2.1 Project Top Level



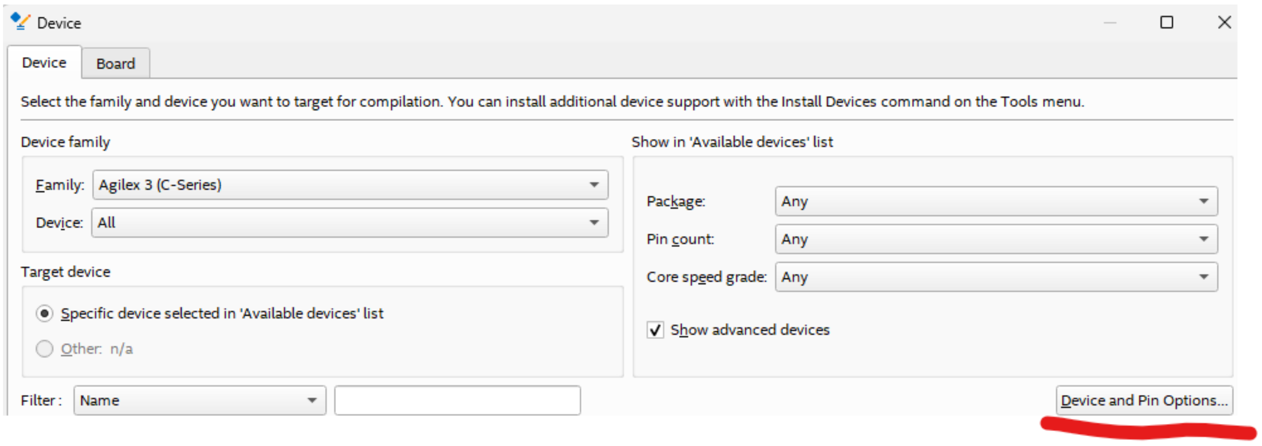
Compilation Flow: During compilation, intermediate Fitter snapshots (planned, placed, routed, and retimed) are not generated.

The top-level (uart_tx) file is seen in the **Project Navigator** pane (top left). The middle section (**Compilation Flow**) shows the FPGA steps which will be taken by the compiler.

These steps include analysis and synthesis, fitter, etc. The far right of the Quartus window shows the IP Catalog. The IP Catalog contains parameterizable functions which will be covered in the next steps.

The AXC3000 board has an LED indicator on the **CONFIG_DONE** pin. **CONFIG_DONE,** being a Dual-Purpose pin, needs to be enabled to drive the LED once the FPGA enters user mode. The Init_done pin also needs to be set for a Reset Release IP.

To do so, select the menu **Assignments -> Devices...** then click the **Device and Pin Options** button.


Device

Select the family and device you want to target for compilation. You can install additional device support with the Install Devices command on the Tools menu.

Device family
Family: Agilex 3 (C-Series)
Device: All

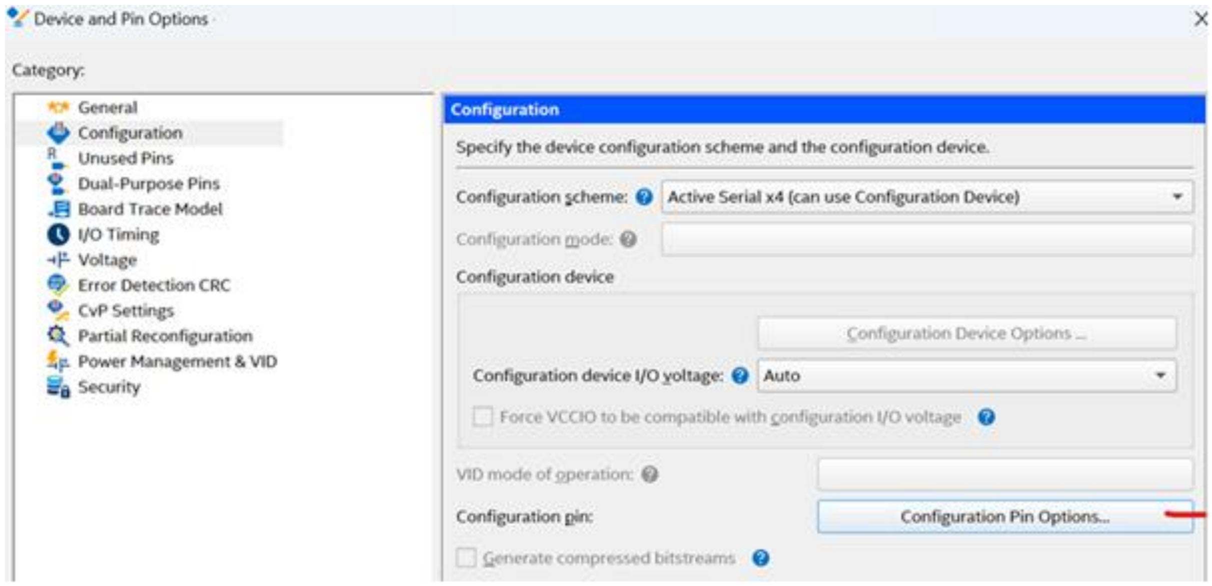
Target device
(●) Specific device selected in 'Available devices' list
( ) Other: n/a

Show in 'Available devices' list
Package: Any
Pin count: Any
Core speed grade: Any
[✓] Show advanced devices

Filter: Name

When the window opens:

- Select **Configuration** in the Category pane and Click the **Configuration Pin Options** bar as



Configuration

Specify the device configuration scheme and the configuration device.

Configuration scheme: Active Serial x4 (can use Configuration Device)

Configuration mode:

Configuration device

Configuration device I/O voltage: Auto

Force VCCIO to be compatible with configuration I/O voltage

VID mode of operation:

Configuration pin:

Generate compressed bitstreams

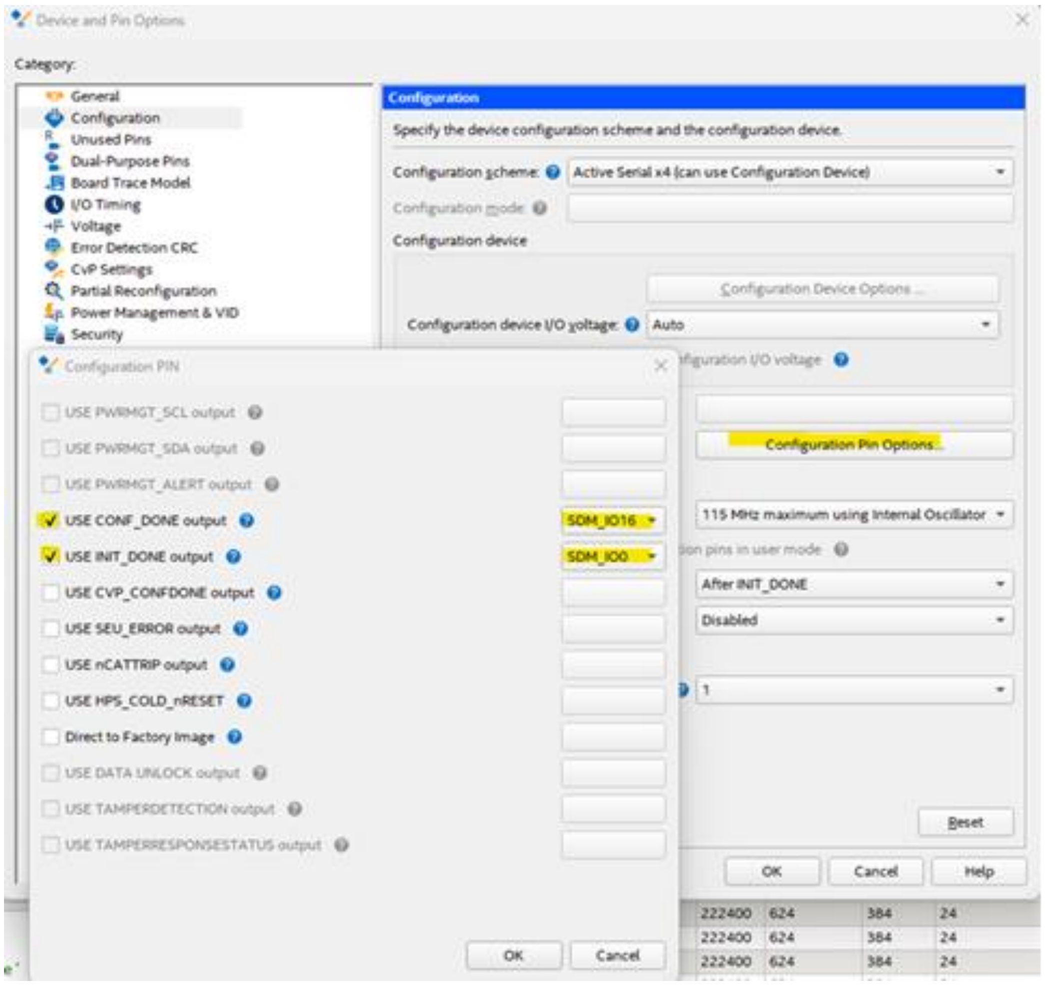
- Check the box to the left of **USE CONF_DONE** output and select ***SDM_IO16***

The Conf_done pin can be used to indicate that the configuration is complete and after the configuration phase, it can be used as general purpose I/O.

- Check the box to the left of **USE INIT_DONE** output and select ***SDM_IO0***

The init_done pin indicates that the initialization phase is completed after the boot up of the FPGA is complete. The Reset Release IP that will be described and created in this workshop will be connected to the Init_done pin.

- Click **OK** in multiple windows (three windows) until done with this task.



Device and Pin Options

Category:

- General
- Configuration
- Unused Pins
- Dual-Purpose Pins
- Board Trace Model
- I/O Timing
- Voltage
- Error Detection CRC
- CvP Settings
- Partial Reconfiguration
- Power Management & VID
- Security

Configuration

Specify the device configuration scheme and the configuration device.

Configuration scheme: Active Serial x4 (can use Configuration Device)

Configuration mode:

Configuration device

Configuration Device Options ...

Configuration device I/O voltage: Auto

Configuration Pin Options...

115 MHz maximum using Internal Oscillator

After INIT_DONE

Disabled

1

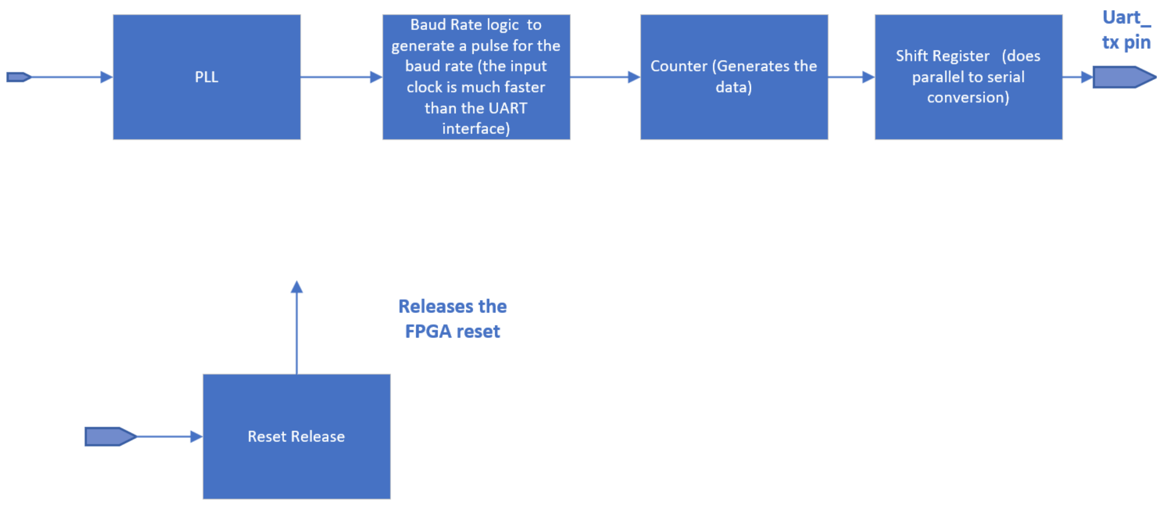
Configuration PIN

[ ] USE PWRMGT_SCL output	
[ ] USE PWRMGT_SDA output	
[ ] USE PWRMGT_ALERT output	
[✓] USE CONF_DONE output	SDM_IO16
[✓] USE INIT_DONE output	SDM_IO0
[ ] USE CVP_CONFDONE output	
[ ] USE SEU_ERROR output	
[ ] USE nCATTRIP output	
[ ] USE HPS_COLD_nRESET	
[ ] Direct to Factory Image	
[ ] USE DATA UNLOCK output	
[ ] USE TAMPERDETECTION output	
[ ] USE TAMPERRESPONSESTATUS output	

4.3 Design Flow

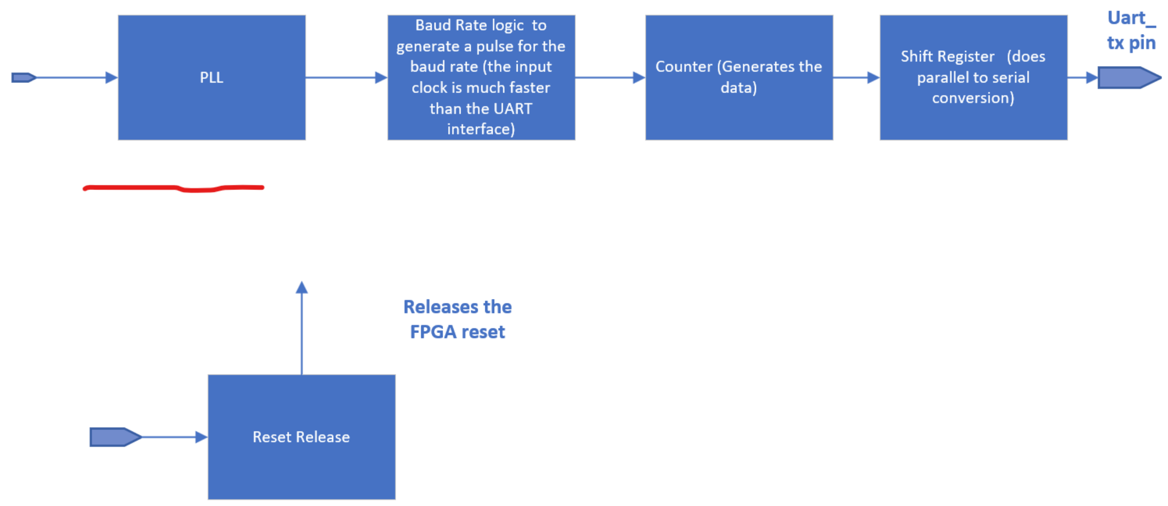
A small UART TX interface design will be implemented in the next steps. This is achieved (without parity bit) by using a PLL, Counter, Shift Register and the Reset Release IP, and a top-level file. The Signal Tap Logic Analyzer will be used to monitor and report various signals within the counter and shift register. The Signal Tap Logic Analyzer can monitor and report internal signals and external signals with no external test equipment.

At a high level, the FPGA design will look like the following:


PLL
Baud Rate logic to generate a pulse for the baud rate (the input clock is much faster than the UART interface)
Counter (Generates the data)
Shift Register (does parallel to serial conversion)
Uart_ tx pin
Releases the FPGA reset
Reset Release

4.3.1 Add a PLL to the Quartus Project

The first IP to be configured is the PLL (phase lock loop). The PLL is seen below, underlined in red.


PLL
Baud Rate logic to generate a pulse for the baud rate (the input clock is much faster than the UART interface)
Counter (Generates the data)
Shift Register (does parallel to serial conversion)
Uart_ tx pin
Releases the FPGA reset
Reset Release

The IP catalogue in Quartus contains many parameterizable functions including a PLL (phase lock loop).

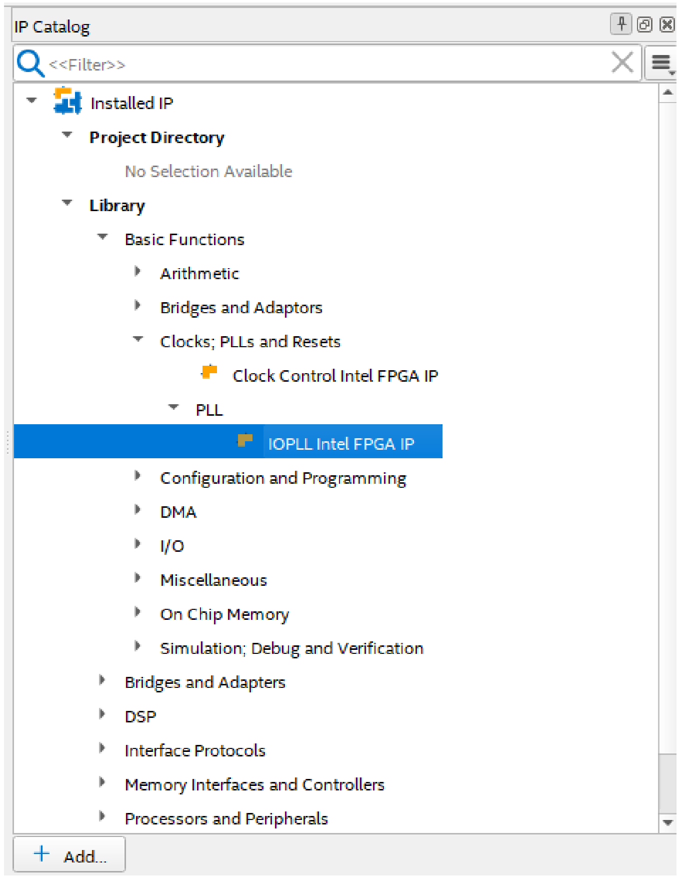
All FPGA designs need at least one clock associated with the data. Synchronous designs are more reliable, have better performance and are more verifiable than a combinational design. Combinational design generally is not a good design approach.

A PLL (phase lock loop) provides flexibility with the clock settings because with an input clock, there can be multiple output clocks with different rates and phases.

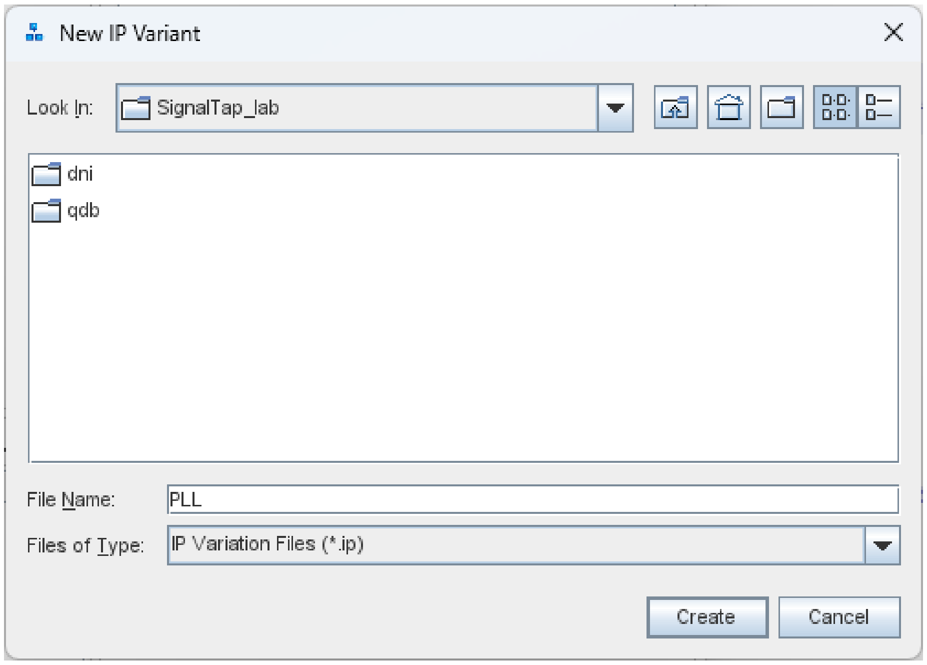
4.3.1.1 Open IP Catalog

From the IP Catalog panel on the right side, expand the menus for the **Basic Functions** → **Clocks; PLLs and Resets** → **PLL** and double click on **IOPLL Intel FPGA IP**.

If the IP Catalog is not visible, then right click on the toolbar and select IP Catalog.



In the pop-up window, give it the name ***PLL***, then click the **Create** button.



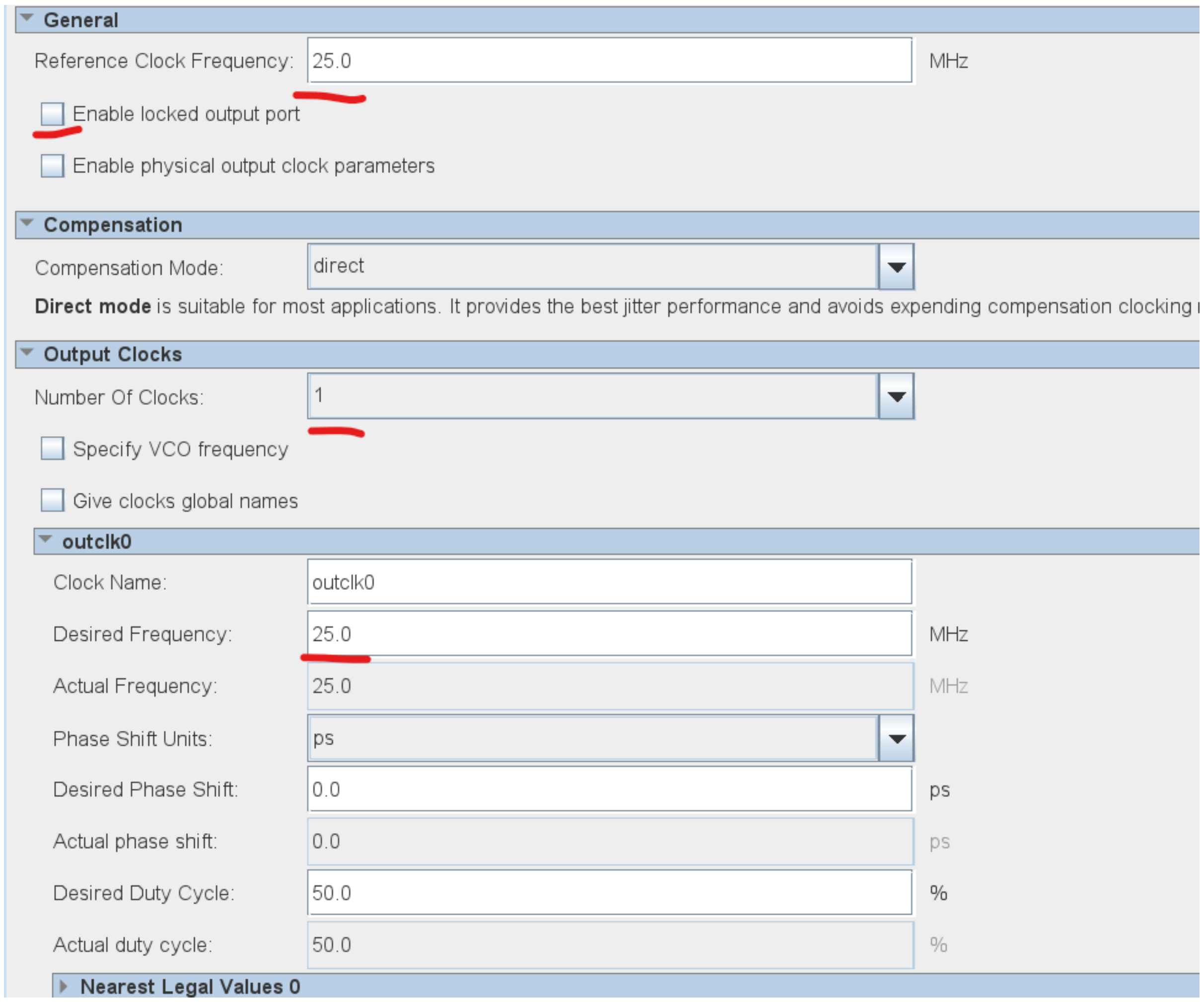
File Name: PLL

Files of Type: IP Variation Files (*.ip)

Under the **PLL** tab in the **General** section of the PLL Catalog, change the **Reference Clock Frequency** to **25 MHz.** This 25 MHz clock source is provided by an external clock source on the AXC3000 board.

Under the **General** section of the PLL tab, uncheck the **Enable Locked Output port** option. This design will not check for the locked output signal of the PLL

Under the **Output Clocks** section (you will need to scroll down the PLL tab) ensure **the Number Of Clocks** is set to 1 and set the **Desired Frequency** for outclk0 to **25.0 Mhz**. This 25 MHz will be used to connect to the baud rate IP.



General

Reference Clock Frequency:	25.0	MHz

☐ Enable locked output port

☐ Enable physical output clock parameters

Compensation

Compensation Mode:	direct	

Direct mode is suitable for most applications. It provides the best jitter performance and avoids expending compensation clocking

Output Clocks

Number Of Clocks:	1	

☐ Specify VCO frequency

☐ Give clocks global names

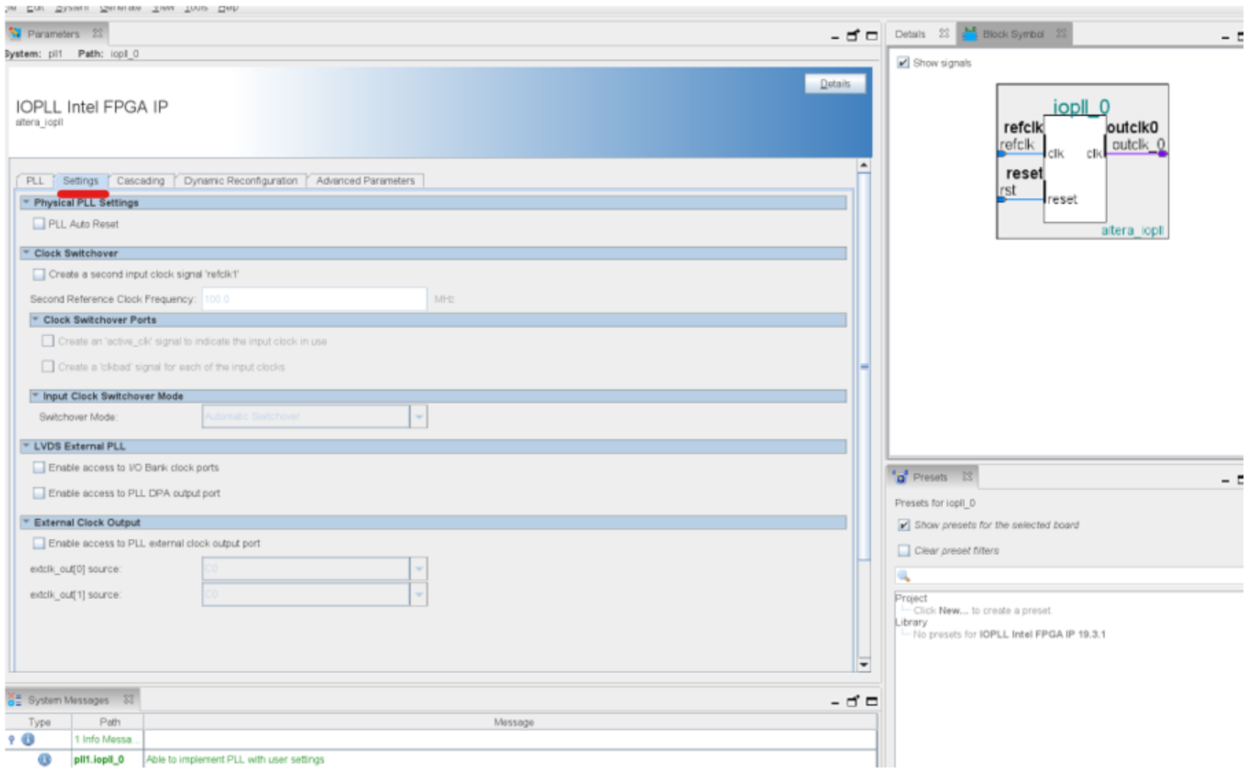
outclk0

Clock Name:	outclk0	
Desired Frequency:	25.0	MHz
Actual Frequency:	25.0	MHz
Phase Shift Units:	ps	
Desired Phase Shift:	0.0	ps
Actual phase shift:	0.0	ps
Desired Duty Cycle:	50.0	%
Actual duty cycle:	50.0	%

Nearest Legal Values 0

There are various options that can be selected as seen in the various tabs such as settings tab (clock switchover, LVDS External PLL), cascading tab (connecting multiple PLLs together so that different PLL combinations can be used), Dynamic reconfiguration, and other parameters. However, to make this workshop easier, the default settings are selected for these tabs.

For example, Select the Settings tab in the IOPLL and review the settings. No changes are needed.



IOPLL Intel FPGA IP
altera_iopll

Physical PLL Settings

☐ PLL Auto Reset

Clock Switchover

☐ Create a second input clock signal 'refclk1'

Second Reference Clock Frequency: 100.0 MHz

Clock Switchover Ports

☐ Create an 'active_clk' signal to indicate the input clock in use

☐ Create a 'clkbad' signal for each of the input clocks

Input Clock Switchover Mode

Switchover Mode: Automatic Switchover

LVDS External PLL

☐ Enable access to I/O Bank clock ports

☐ Enable access to PLL DPA output port

External Clock Output

☐ Enable access to PLL external clock output port

extclk_out[0] source: C0

extclk_out[1] source: C0

✓ Show signals

Presets for iopll_0

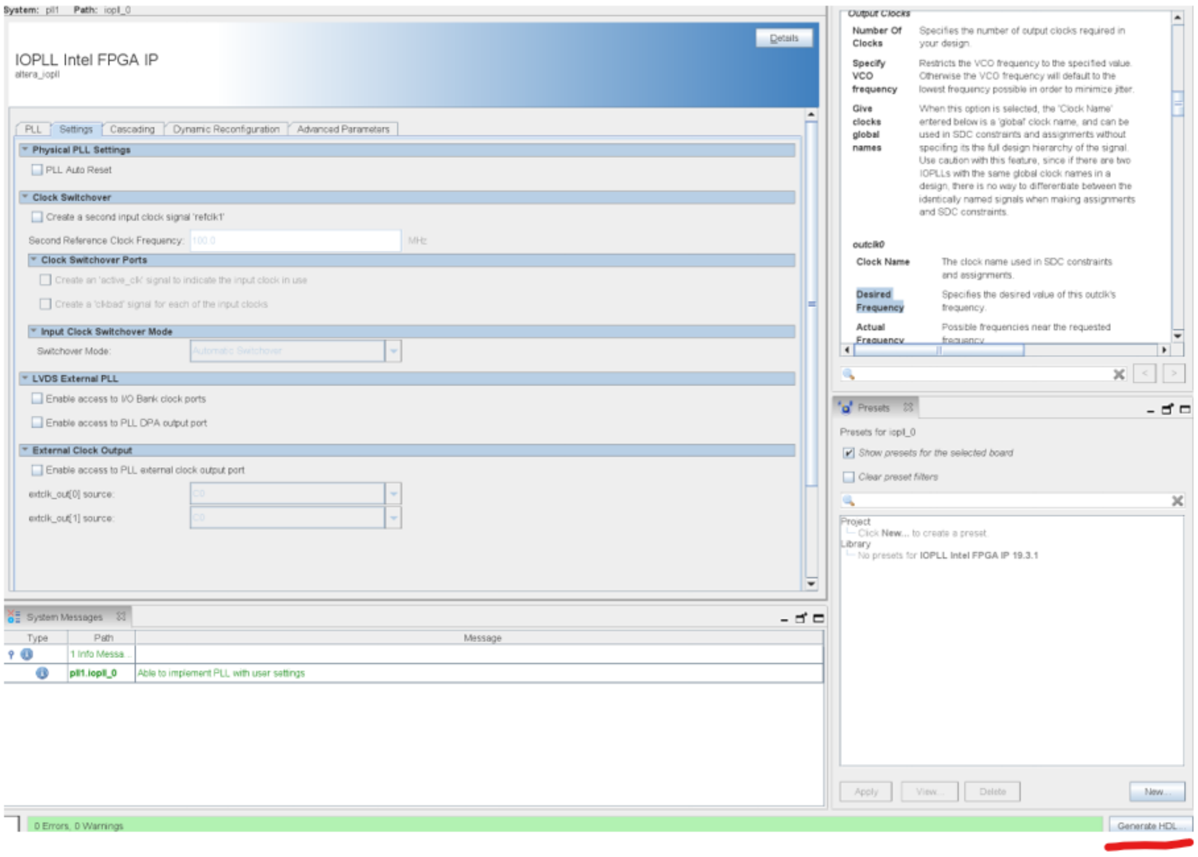
✓ Show presets for the selected board

☐ Clear preset filters

Project
Click **New...** to create a preset.
Library
No presets for **IOPLL Intel FPGA IP 19.3.1**

Type	Path	Message
	1 Info Messa...	
	pll1.iopll_0	Able to implement PLL with user settings

Select the **Generate HDL** button (bottom right-hand corner) as


System: pll1 Path: iopll_0

IOPLL Intel FPGA IP
altera_iopll

Physical PLL Settings

☐ PLL Auto Reset

Clock Switchover

☐ Create a second input clock signal 'refclk1'

Second Reference Clock Frequency: 100.0 MHz

Clock Switchover Ports

☐ Create an 'active_clk' signal to indicate the input clock in use

☐ Create a 'clkbad' signal for each of the input clocks

Input Clock Switchover Mode

Switchover Mode: Automatic Switchover

LVDS External PLL

☐ Enable access to I/O Bank clock ports

☐ Enable access to PLL DPA output port

External Clock Output

☐ Enable access to PLL external clock output port

extclk_out[0] source: C0

extclk_out[1] source: C0

System Messages

Type	Path	Message
	1 Info Messa...	
	pll1.iopll_0	Able to implement PLL with user settings

Output Clocks

Number Of Clocks	Specifies the number of output clocks required in your design.
Specify VCO frequency	Restricts the VCO frequency to the specified value. Otherwise the VCO frequency will default to the lowest frequency possible in order to minimize jitter.
Give clocks global names	When this option is selected, the 'Clock Name' entered below is a 'global' clock name, and can be used in SDC constraints and assignments without specifing its the full design hierarchy of the signal. Use caution with this feature, since if there are two IOPLLs with the same global clock names in a design, there is no way to differentiate between the identically named signals when making assignments and SDC constraints.

outclk0

Clock Name	The clock name used in SDC constraints and assignments.
Desired Frequency	Specifies the desired value of this outclk's frequency.
Actual Frequency	Possible frequencies near the requested frequency.

Presets

Presets for iopll_0

☑ *Show presets for the selected board*

☐ *Clear preset filters*

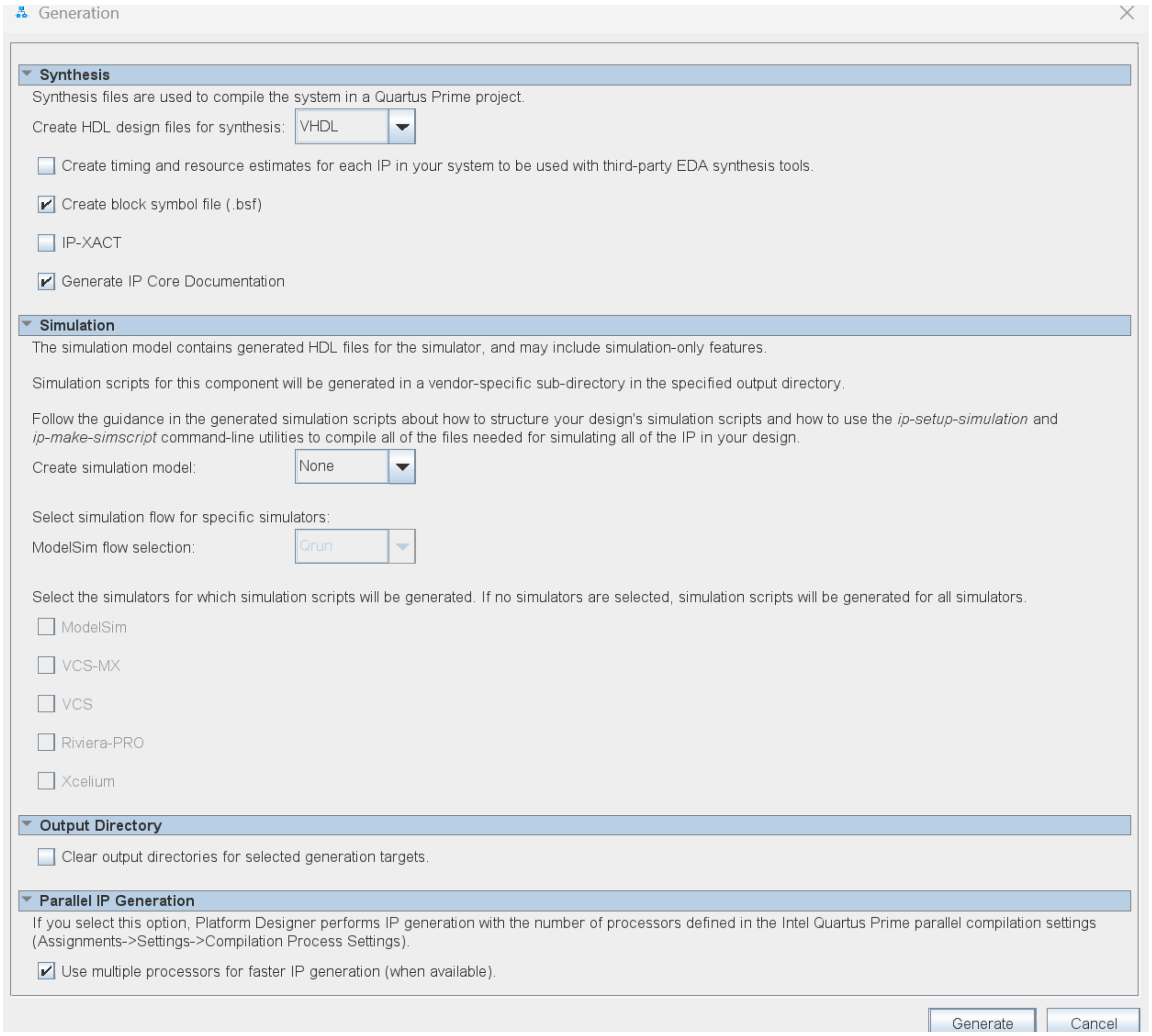
Project
└ Click **New...** to create a preset.
Library
└ No presets for **IOPLL Intel FPGA IP 19.3.1**

0 Errors, 0 Warnings

The Generate HDL converts the settings into an RTL file that can be instantiated and "connected" to the top-level RTL file.

Change the **Default settings to be VHDL**. Verilog could have been used as well but this workshop was created with VHDL files. The focus for this workshop is on synthesis (not on simulation).

The settings should look like:



Generation

Synthesis

Synthesis files are used to compile the system in a Quartus Prime project.

Create HDL design files for synthesis: VHDL

- [ ] Create timing and resource estimates for each IP in your system to be used with third-party EDA synthesis tools.
- [x] Create block symbol file (.bsf)
- [ ] IP-XACT
- [x] Generate IP Core Documentation

Simulation

The simulation model contains generated HDL files for the simulator, and may include simulation-only features.

Simulation scripts for this component will be generated in a vendor-specific sub-directory in the specified output directory.

Follow the guidance in the generated simulation scripts about how to structure your design's simulation scripts and how to use the *ip-setup-simulation* and *ip-make-simscript* command-line utilities to compile all of the files needed for simulating all of the IP in your design.

Create simulation model: None

Select simulation flow for specific simulators:

ModelSim flow selection: Qrun

Select the simulators for which simulation scripts will be generated. If no simulators are selected, simulation scripts will be generated for all simulators.

- [ ] ModelSim
- [ ] VCS-MX
- [ ] VCS
- [ ] Riviera-PRO
- [ ] Xcelium

Output Directory

- [ ] Clear output directories for selected generation targets.

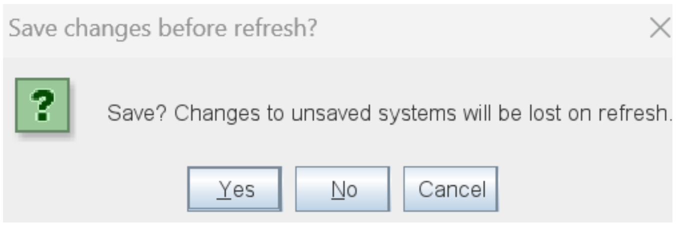
Parallel IP Generation

If you select this option, Platform Designer performs IP generation with the number of processors defined in the Intel Quartus Prime parallel compilation settings (Assignments->Settings->Compilation Process Settings).

- [x] Use multiple processors for faster IP generation (when available).

Select Generate.

A window will appear that states Save? Changes to unsaved system will be lost on refresh

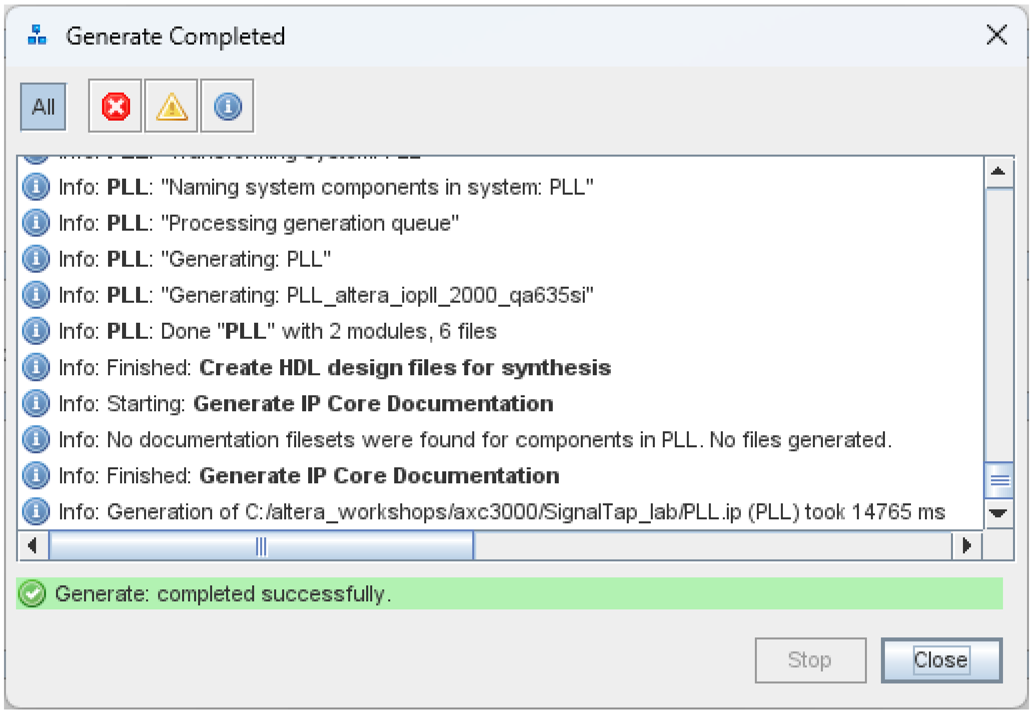


Save changes before refresh?

Save? Changes to unsaved systems will be lost on refresh.

Select **Yes**. The file is saved and generated.

A window will appear that shows when the RTL files (and other files) are generated. The green comment states when it is completed successfully.



```
Info: PLL: "Naming system components in system: PLL"
Info: PLL: "Processing generation queue"
Info: PLL: "Generating: PLL"
Info: PLL: "Generating: PLL_altera_iopll_2000_qa635si"
Info: PLL: Done "PLL" with 2 modules, 6 files
Info: Finished: Create HDL design files for synthesis
Info: Starting: Generate IP Core Documentation
Info: No documentation filesets were found for components in PLL. No files generated.
Info: Finished: Generate IP Core Documentation
Info: Generation of C:/altera_workshops/axc3000/SignalTap_lab/PLL.ip (PLL) took 14765 ms
Generate: completed successfully.
```

Select **Close**.

Close the IP Parameter Editor by selecting **File** → **Exit.**

There are different files that are generated by Quartus including:

.qip (a single file that contains paths for the IP core rather than adding individual files),

.rpt (Quartus report file),

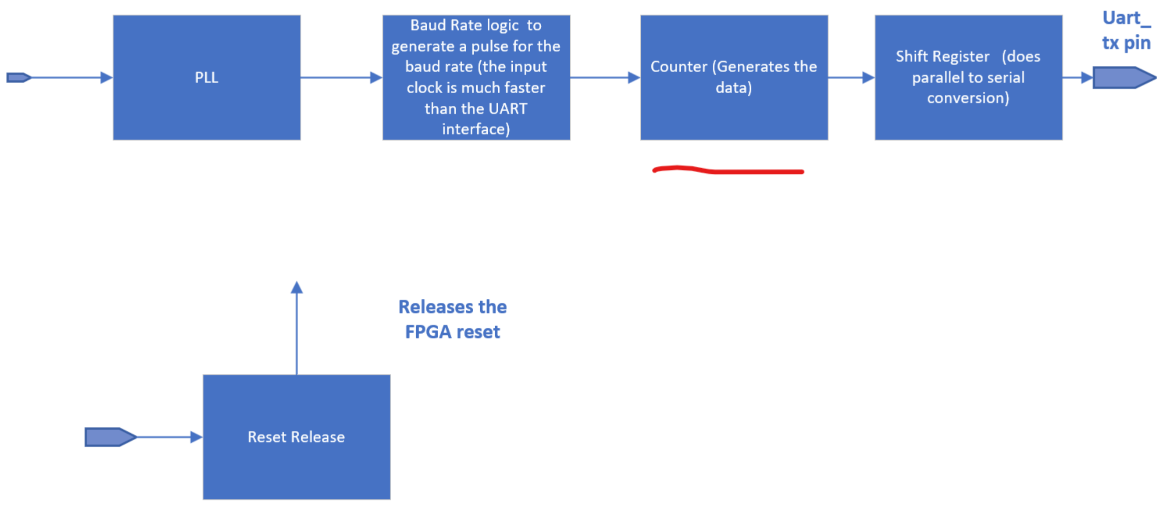
.ip file (intellectual parameterizable file)

.vhd (RTL file) and many other files.

4.3.2 Add a counter to the Quartus Project

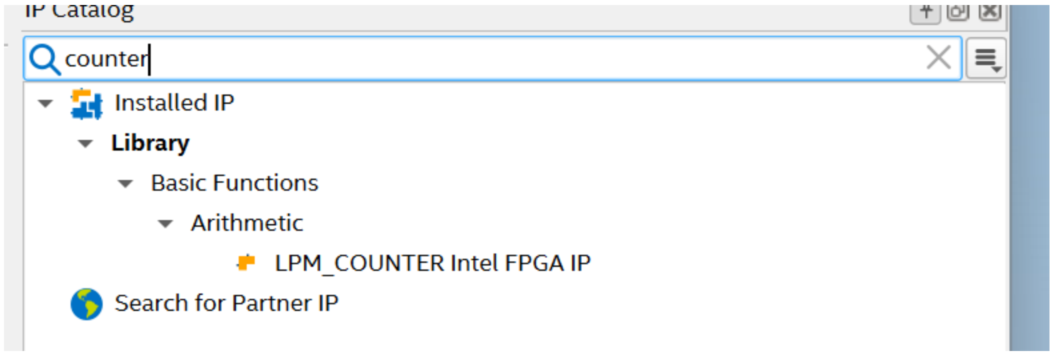
The next configurable IP that will be implemented is the counter IP. The counter will generate the data for the UART TX interface. (The baud rate section is in the top-level file that you have imported).

The counter in the top-level view can be seen as:

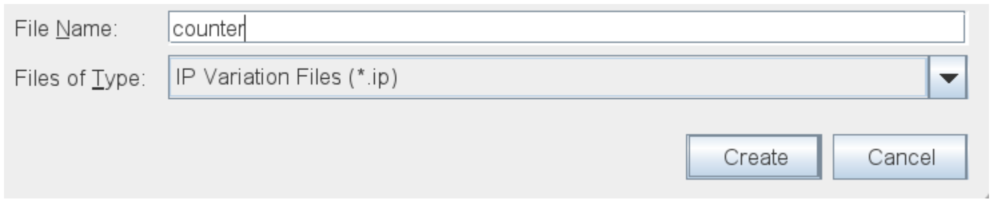

PLL
Baud Rate logic to generate a pulse for the baud rate (the input clock is much faster than the UART interface)
Counter (Generates the data)
Shift Register (does parallel to serial conversion)
Uart_ tx pin
Releases the FPGA reset
Reset Release

4.3.2.1 Open IP Catalog

In the search bar of the IP Catalog, type "**counter**", and **double click** on **LPM_COUNTER Intel FPGA IP** as below. Even if part of the word of count is typed, Quartus will show the closest IP.



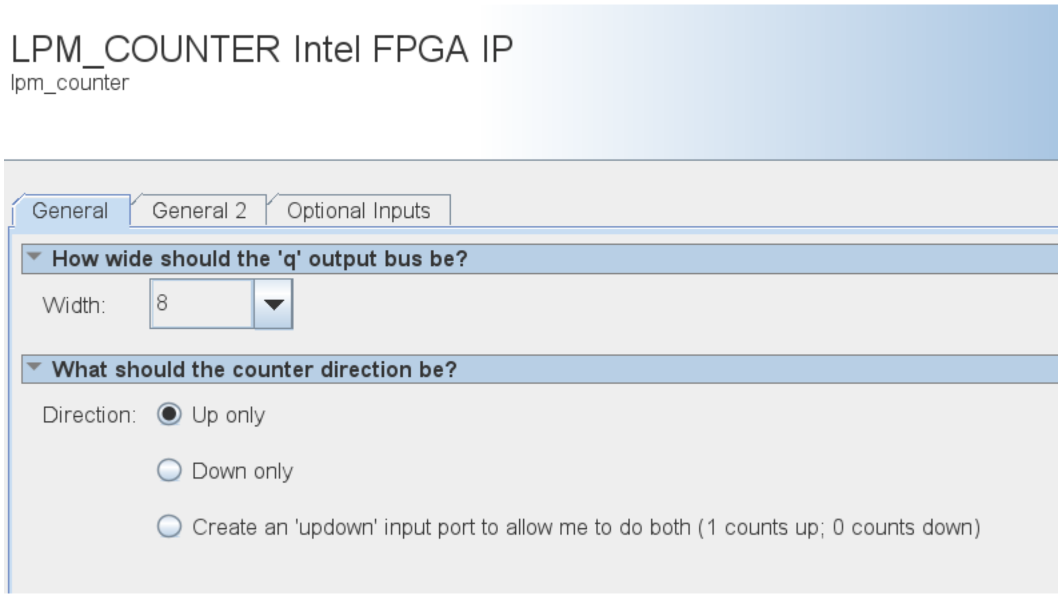
In the New IP Variant window enter **counter** for the IP variation file name as:


File Name: counter

Files of Type: IP Variation Files (*.ip)

Select **Create**

On the General Tab, do not make any changes to the default settings (bus width of 8, Up only).


LPM_COUNTER Intel FPGA IP
lpm_counter

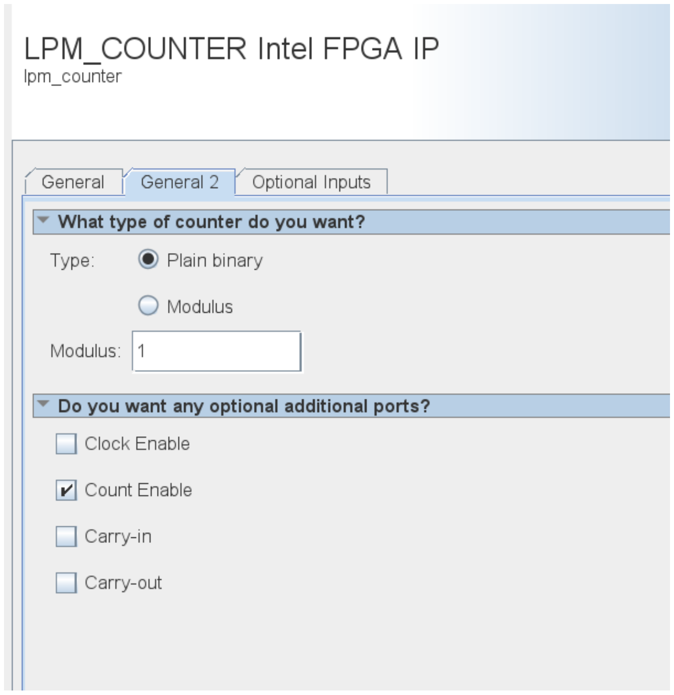
How wide should the 'q' output bus be?

Width: 8

What should the counter direction be?

Direction: [✓ Up only] Down only Create an 'updown' input port to allow me to do both (1 counts up; 0 counts down)

On the General Tab 2, select **Count Enable**


LPM_COUNTER Intel FPGA IP
lpm_counter

General | General 2 | Optional Inputs

What type of counter do you want?

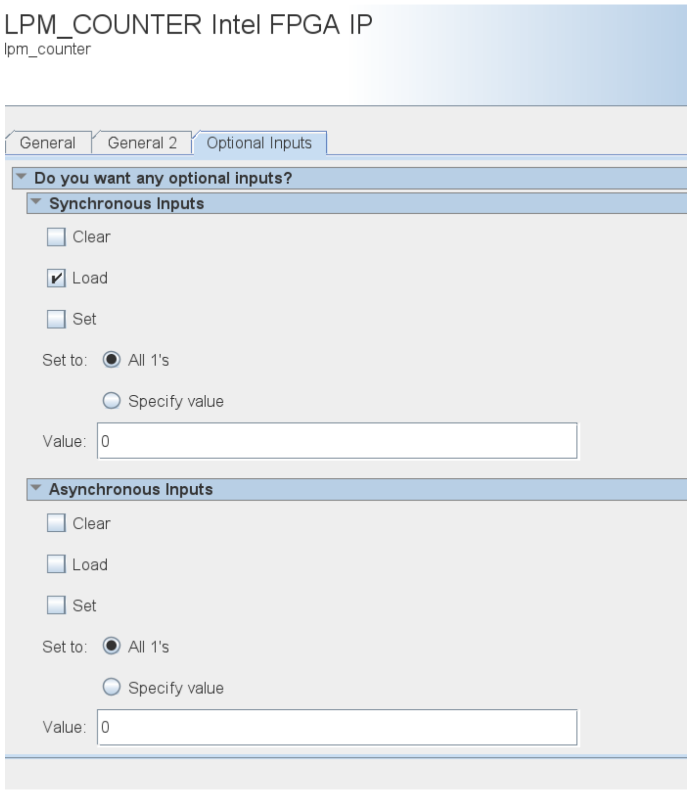
Type: [✓ Plain binary] Modulus

Modulus: 1

Do you want any optional additional ports?

☐ Clock Enable
☑ Count Enable
☐ Carry-in
☐ Carry-out

On the Optional Inputs tab, select **Load** under Synchronous Inputs


LPM_COUNTER Intel FPGA IP
lpm_counter

General | General 2 | Optional Inputs

Do you want any optional inputs?

Synchronous Inputs

☐ Clear
☑ Load
☐ Set

Set to: [✓ All 1's] Specify value

Value: 0

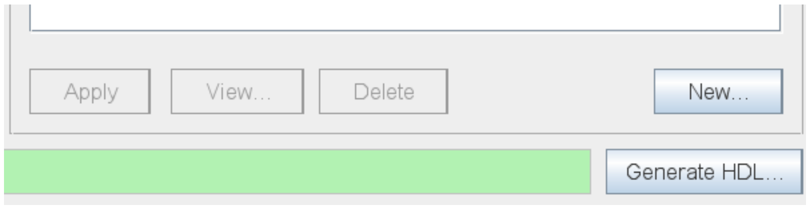
Asynchronous Inputs

☐ Clear
☐ Load
☐ Set

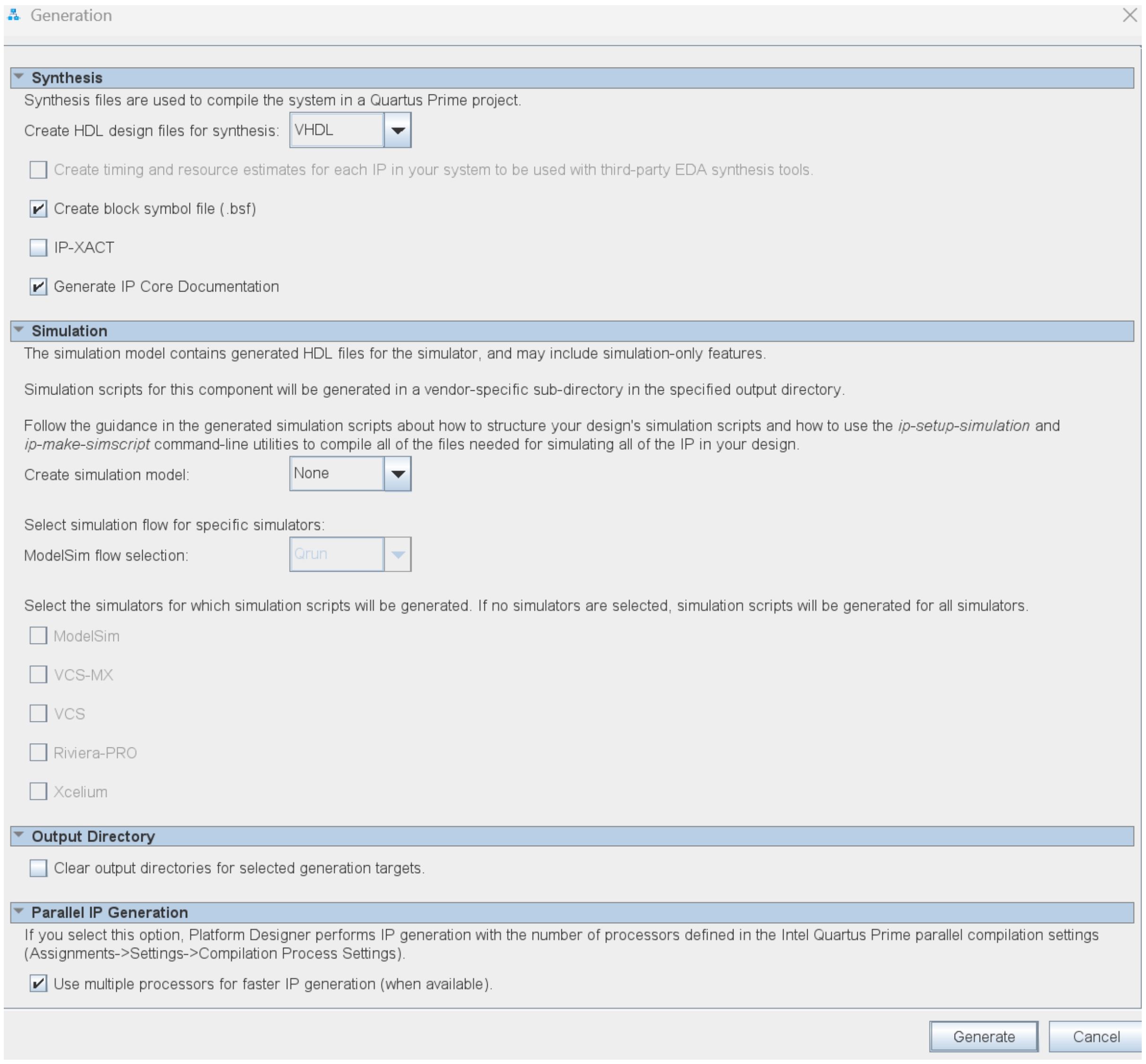
Set to: [✓ All 1's] Specify value

Value: 0

Select **Generate HDL** (bottom right-hand corner of the screen) so that it can generate an RTL file as



Change the **settings to be VHDL (if it is not set already)**, as VHDL was used for this workshop. Verilog could have been used as well.



Synthesis

Synthesis files are used to compile the system in a Quartus Prime project.

Create HDL design files for synthesis: VHDL

☐ Create timing and resource estimates for each IP in your system to be used with third-party EDA synthesis tools.

☑ Create block symbol file (.bsf)

☐ IP-XACT

☑ Generate IP Core Documentation

Simulation

The simulation model contains generated HDL files for the simulator, and may include simulation-only features.

Simulation scripts for this component will be generated in a vendor-specific sub-directory in the specified output directory.

Follow the guidance in the generated simulation scripts about how to structure your design's simulation scripts and how to use the *ip-setup-simulation* and *ip-make-simscript* command-line utilities to compile all of the files needed for simulating all of the IP in your design.

Create simulation model: None

Select simulation flow for specific simulators:

ModelSim flow selection: Qrun

Select the simulators for which simulation scripts will be generated. If no simulators are selected, simulation scripts will be generated for all simulators.

☐ ModelSim

☐ VCS-MX

☐ VCS

☐ Riviera-PRO

☐ Xcelium

Output Directory

☐ Clear output directories for selected generation targets.

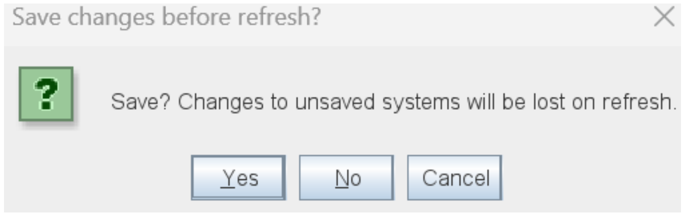
Parallel IP Generation

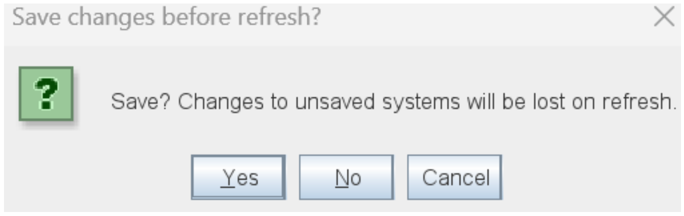
If you select this option, Platform Designer performs IP generation with the number of processors defined in the Intel Quartus Prime parallel compilation settings (Assignments->Settings->Compilation Process Settings).

☑ Use multiple processors for faster IP generation (when available).

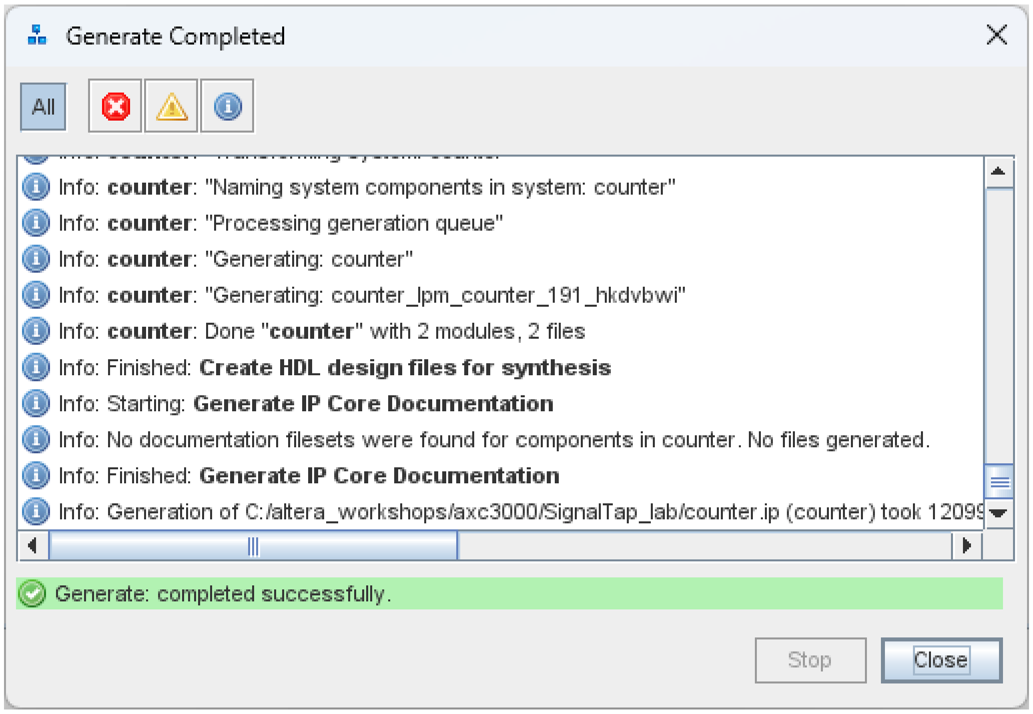
Select Generate.

Select **Yes** with the pop-up window that states **Save? Changes to the unsaved systems will be lost on refresh** to save the changes.


Save changes before refresh?
Save? Changes to unsaved systems will be lost on refresh.
Yes
No
Cancel


Save changes before refresh?
Save? Changes to unsaved systems will be lost on refresh.
Yes
No
Cancel

Select **Close** (bottom right of the window) once it is generated.



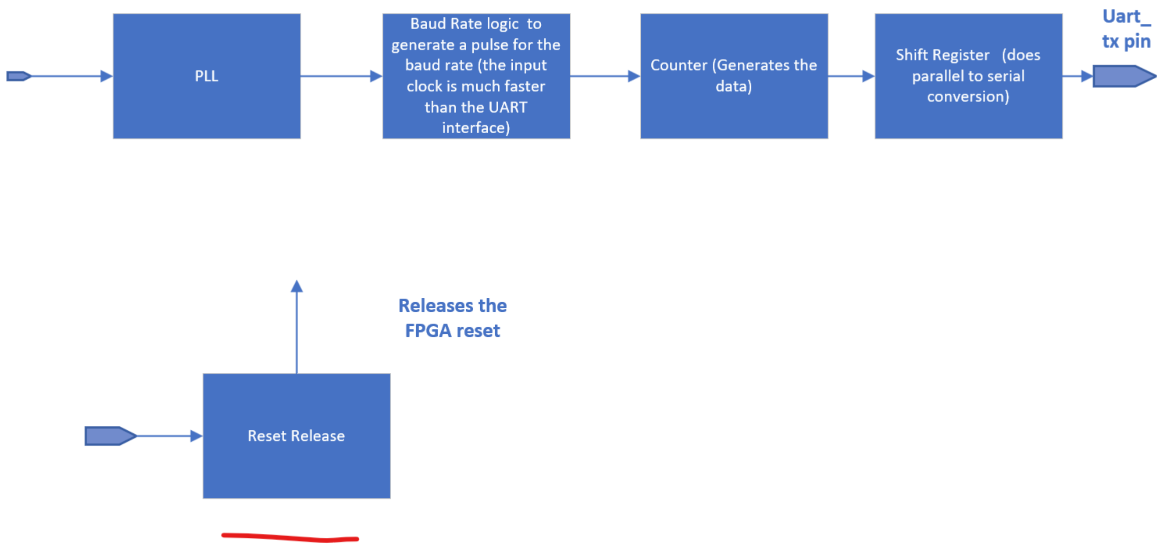
```
Info: counter: "Naming system components in system: counter"
Info: counter: "Processing generation queue"
Info: counter: "Generating: counter"
Info: counter: "Generating: counter_lpm_counter_191_hkdvbwi"
Info: counter: Done "counter" with 2 modules, 2 files
Info: Finished: Create HDL design files for synthesis
Info: Starting: Generate IP Core Documentation
Info: No documentation filesets were found for components in counter. No files generated.
Info: Finished: Generate IP Core Documentation
Info: Generation of C:/altera_workshops/axc3000/SignalTap_lab/counter.ip (counter) took 1209
Generate: completed successfully.
```

Close the IP Parameter Window for the counter IP by selecting **File** → **Exit**

4.3.3 Add a Reset Release IP

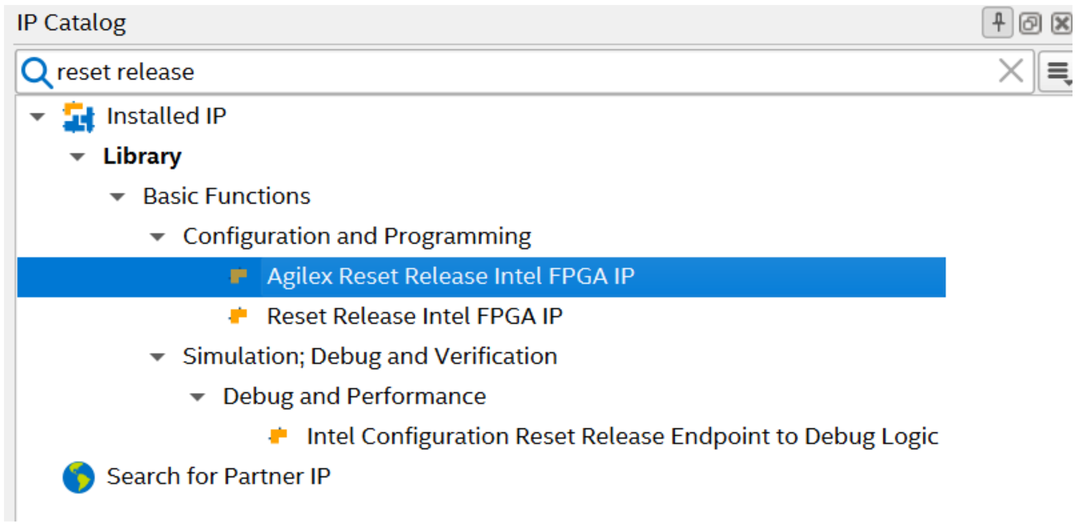
For Agilex 3 devices (and other families), it is recommended to use the Reset Release IP. The Reset Release is an IP that generates a signal to indicate the device configuration is complete. It holds the control circuit in reset until the configuration is complete and user mode is entered. If this IP is not implemented, Quartus will report a warning that the configuration might not be correct.

The Reset Release IP in the top-level view is

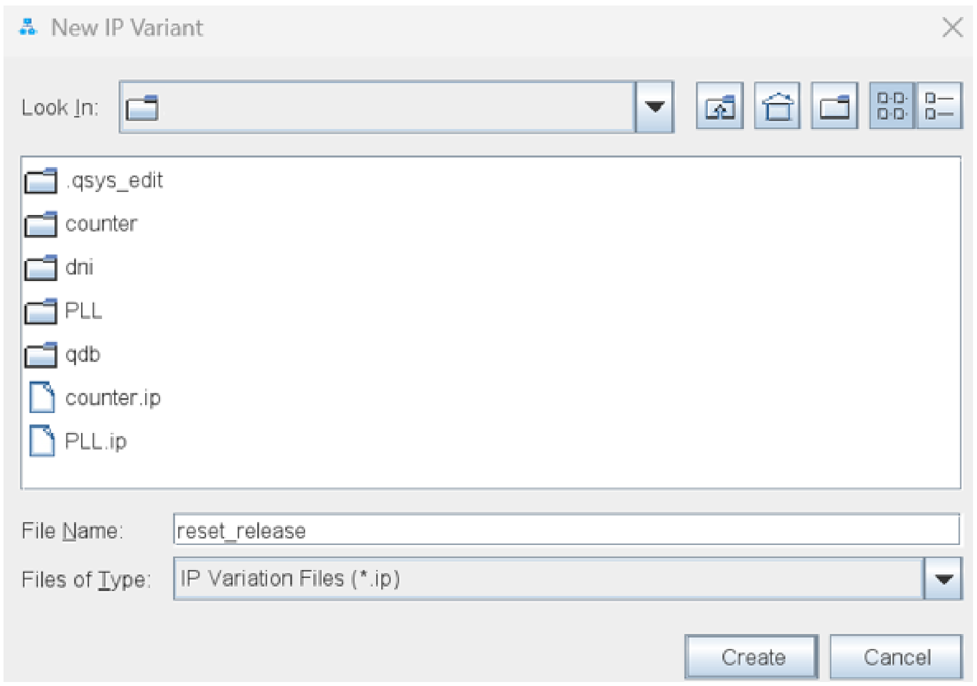

PLL
Baud Rate logic to generate a pulse for the baud rate (the input clock is much faster than the UART interface)
Counter (Generates the data)
Shift Register (does parallel to serial conversion)
Uart_ tx pin
Releases the FPGA reset
Reset Release

4.3.3.1 Open IP Catalog

In the search bar of the IP Catalog, type "**reset release**", and **double click** on **Agilex Reset Release Intel FPGA IP** as below. Even if part of the word "reset" is typed, Quartus will show the closest IP.


IP Catalog
reset release
Installed IP
Library
Basic Functions
Configuration and Programming
Agilex Reset Release Intel FPGA IP
Reset Release Intel FPGA IP
Simulation; Debug and Verification
Debug and Performance
Intel Configuration Reset Release Endpoint to Debug Logic
Search for Partner IP

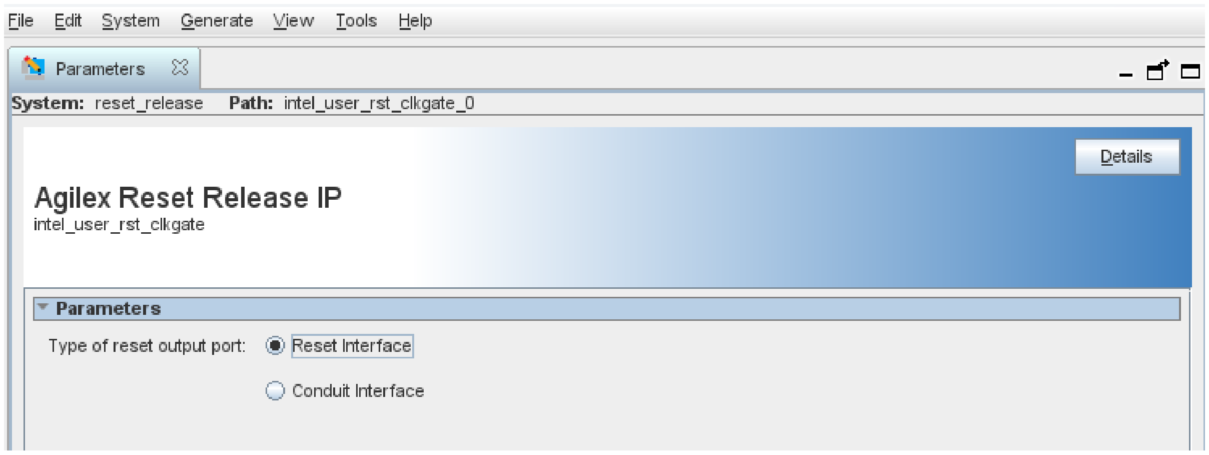
Enter **reset_release** for the File Name.



File Name: reset_release

Select Create (bottom right of the window)

Select the type of reset output port as **Reset interface**



System: reset_release Path: intel_user_rst_clkgate_0

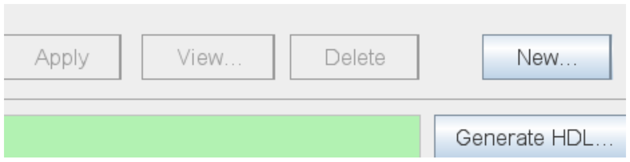
Agilex Reset Release IP
intel_user_rst_clkgate

Parameters

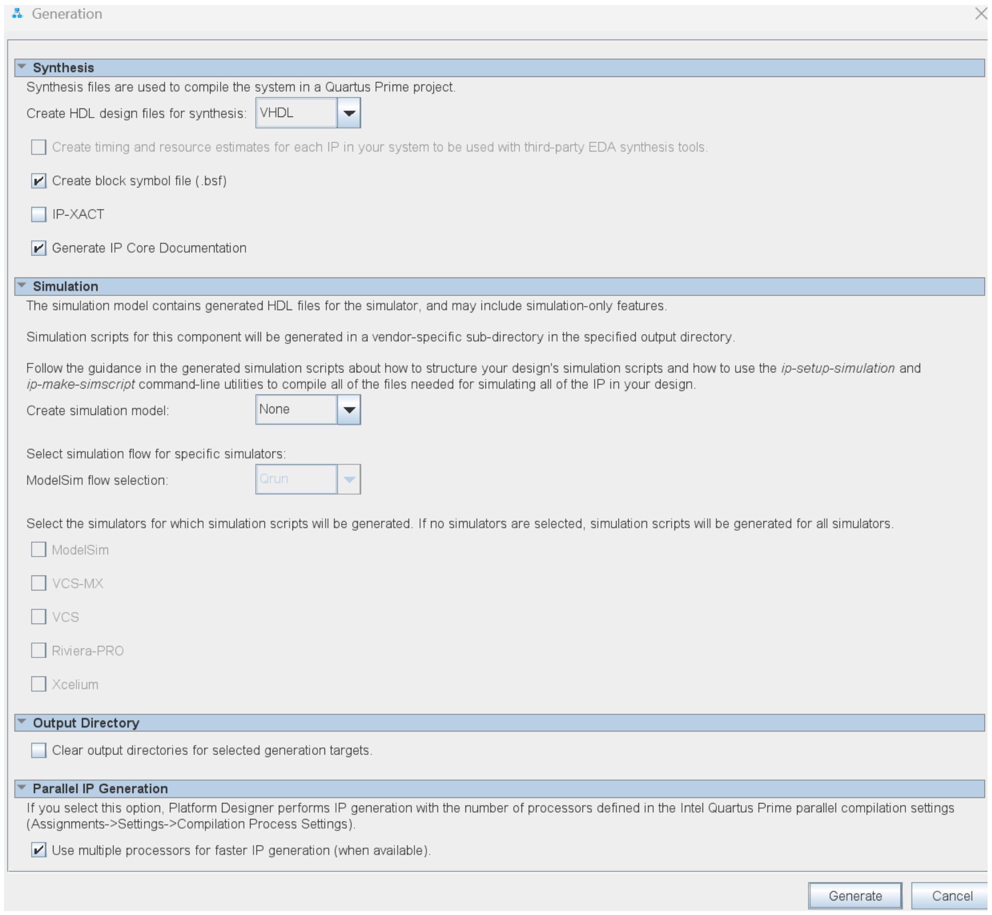
Type of reset output port: [✓ Reset Interface] Conduit Interface

Conduit interface is a legacy reset output, thus the Reset interface should be selected.

Select **Generate HDL** in bottom right of the window as



Select **Generate** (in the bottom right of the window) with no changes



Generation

Synthesis

Synthesis files are used to compile the system in a Quartus Prime project.

Create HDL design files for synthesis: VHDL

☐ Create timing and resource estimates for each IP in your system to be used with third-party EDA synthesis tools.

☑ Create block symbol file (.bsf)

☐ IP-XACT

☑ Generate IP Core Documentation

Simulation

The simulation model contains generated HDL files for the simulator, and may include simulation-only features.

Simulation scripts for this component will be generated in a vendor-specific sub-directory in the specified output directory.

Follow the guidance in the generated simulation scripts about how to structure your design's simulation scripts and how to use the *ip-setup-simulation* and *ip-make-simscript* command-line utilities to compile all of the files needed for simulating all of the IP in your design.

Create simulation model: None

Select simulation flow for specific simulators:

ModelSim flow selection: Qrun

Select the simulators for which simulation scripts will be generated. If no simulators are selected, simulation scripts will be generated for all simulators.

☐ ModelSim

☐ VCS-MX

☐ VCS

☐ Riviera-PRO

☐ Xcelium

Output Directory

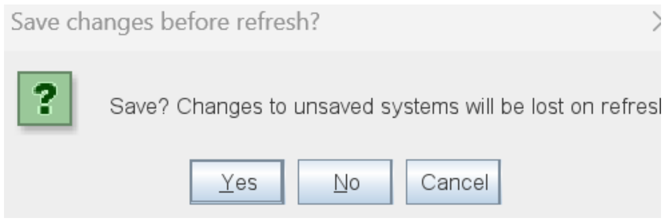
☐ Clear output directories for selected generation targets.

Parallel IP Generation

If you select this option, Platform Designer performs IP generation with the number of processors defined in the Intel Quartus Prime parallel compilation settings (Assignments->Settings->Compilation Process Settings).

☑ Use multiple processors for faster IP generation (when available).

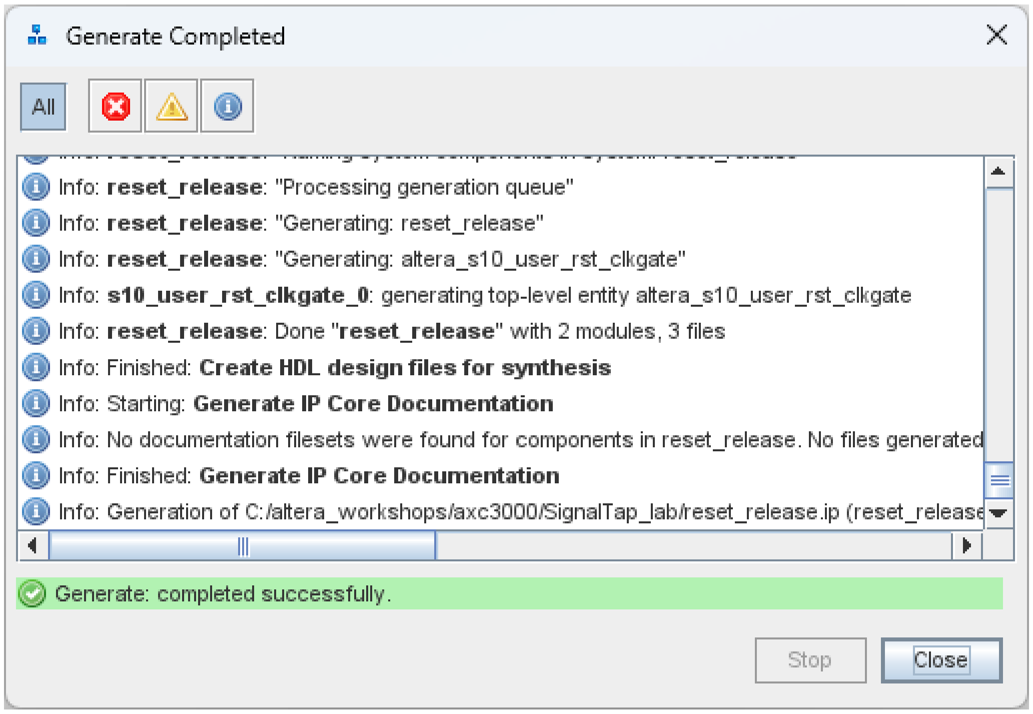
Select **Yes** to the window of Save? Changes to unsaved systems will be lost on refresh.



Save changes before refresh?

Save? Changes to unsaved systems will be lost on refresl

Select **Close** after the Generation is complete



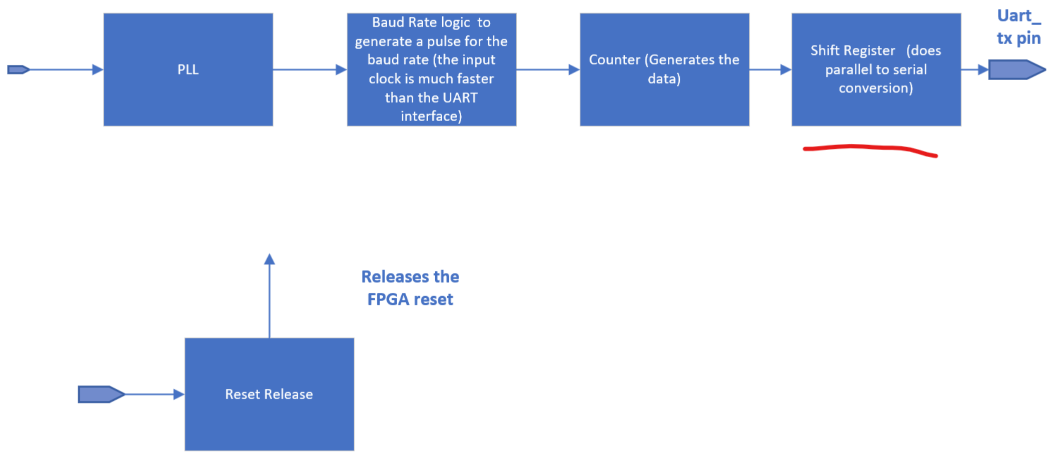
```
Info: reset_release: "Processing generation queue"
Info: reset_release: "Generating: reset_release"
Info: reset_release: "Generating: altera_s10_user_rst_clkgate"
Info: s10_user_rst_clkgate_0: generating top-level entity altera_s10_user_rst_clkgate
Info: reset_release: Done "reset_release" with 2 modules, 3 files
Info: Finished: Create HDL design files for synthesis
Info: Starting: Generate IP Core Documentation
Info: No documentation filesets were found for components in reset_release. No files generated
Info: Finished: Generate IP Core Documentation
Info: Generation of C:/altera_workshops/axc3000/SignalTap_lab/reset_release.ip (reset_release
Generate: completed successfully.
```

Close the IP Parameter Window by selecting **File** → **Exit**

4.3.4 Add a Shift Register IP

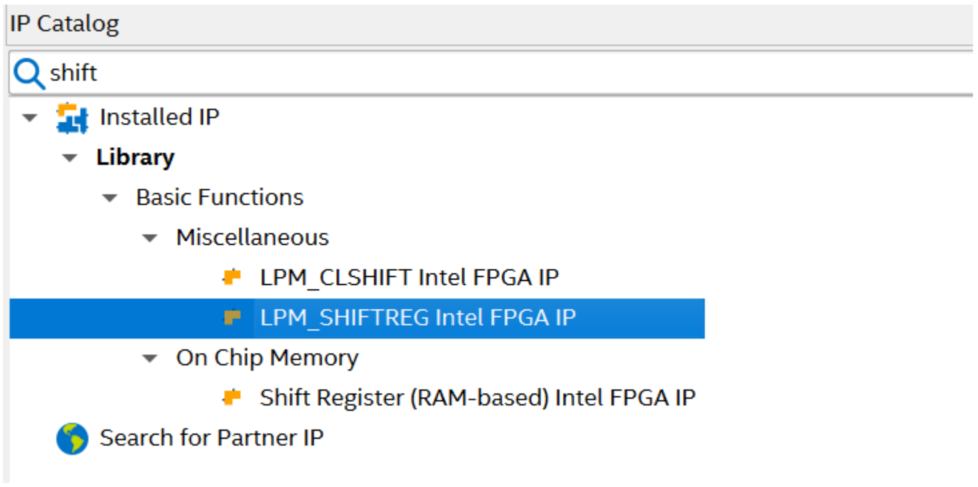
A Shift Register, which takes data in on clock cycles and then shifts the data out, is used for parallel to serial conversion.

The Shift Register IP in the top level file is seen as


PLL
Baud Rate logic to generate a pulse for the baud rate (the input clock is much faster than the UART interface)
Counter (Generates the data)
Shift Register (does parallel to serial conversion)
Uart_ tx pin
Releases the FPGA reset
Reset Release

4.3.4.1 Open IP Catalog

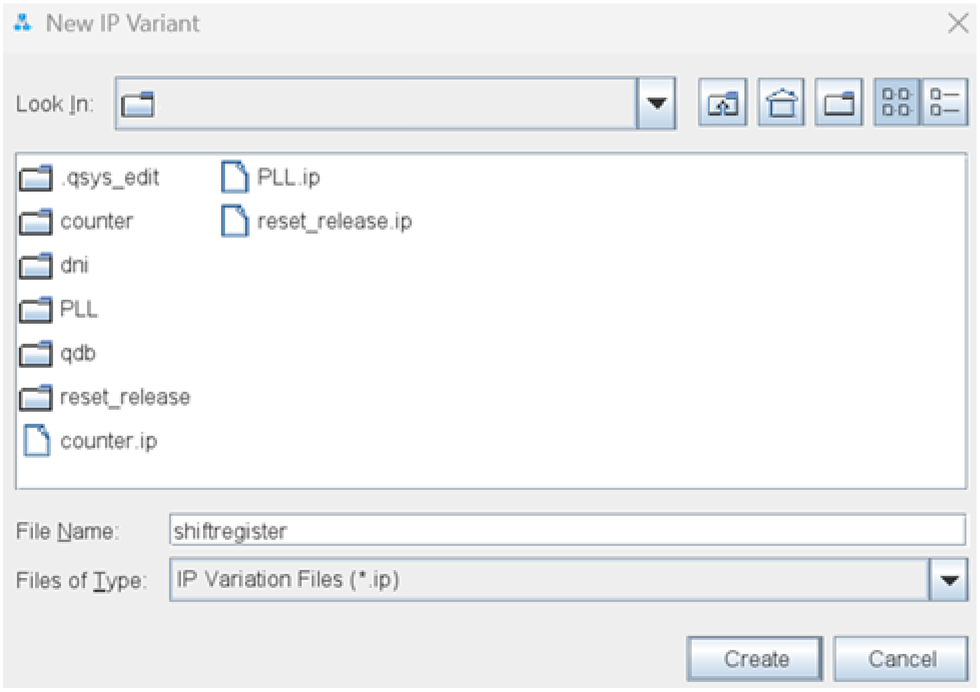
From the IP Catalog panel on the right side, expand the menus for the Basic Functions → and double-click on the **lpm_shiftreg Intel FPGA IP** as


IP Catalog

shift

- Installed IP
  - Library
    - Basic Functions
      - Miscellaneous
        - LPM_CLSHIFT Intel FPGA IP
        - LPM_SHIFTREG Intel FPGA IP
      - On Chip Memory
        - Shift Register (RAM-based) Intel FPGA IP
- Search for Partner IP

Enter **shiftregister** for the File Name.


New IP Variant

Look In:

.qsys_edit
counter
dni
PLL
qdb
reset_release
counter.ip
PLL.ip
reset_release.ip

File Name: shiftregister

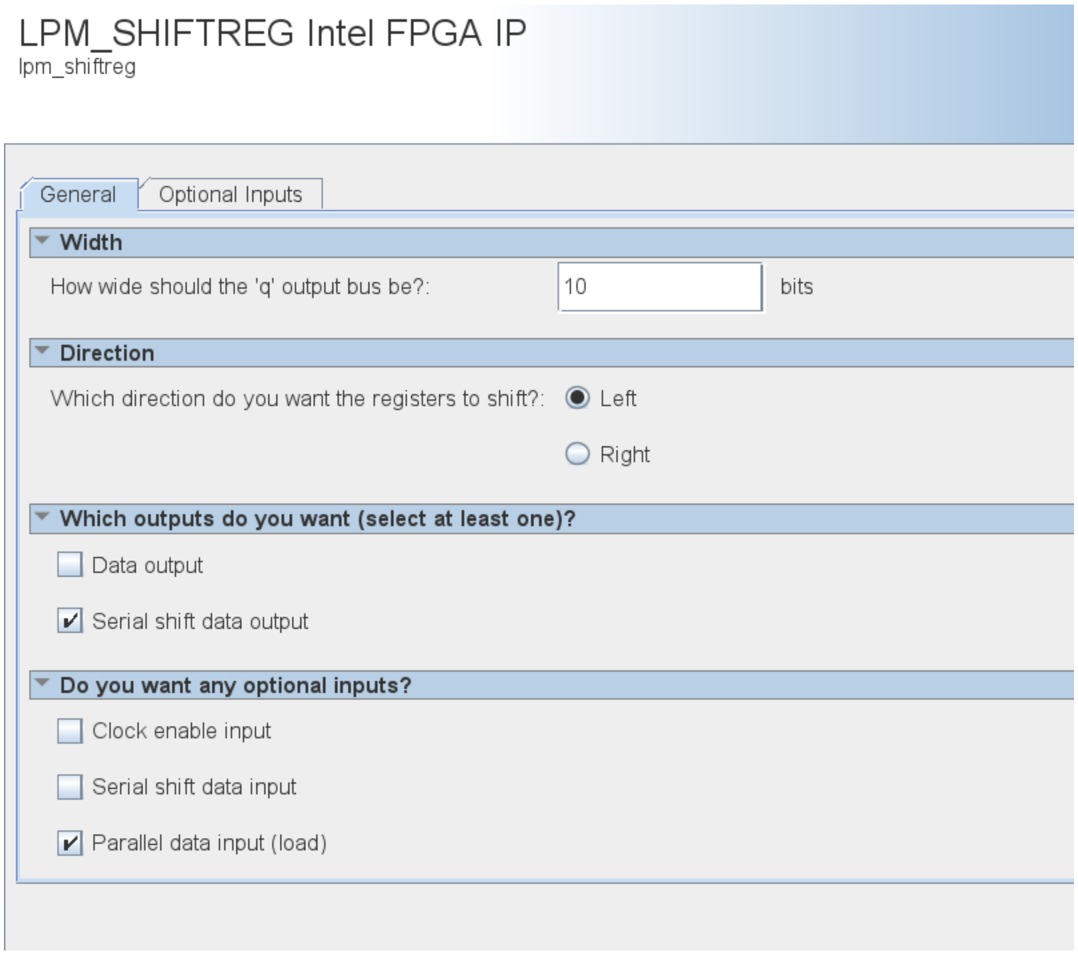
Files of Type: IP Variation Files (*.ip)

Select **Create**

Various changes will be made.

Set the output bus as 10 bits. For the output options, select Serial shift data output (unclick the data output) and for the input options, select Parallel data input (load) (unclick the serial shift data input) as

(This will enable parallel to serial conversion.)



LPM_SHIFTREG Intel FPGA IP
lpm_shiftreg

Width

How wide should the 'q' output bus be?: 10 bits

Direction

Which direction do you want the registers to shift?: [✓ Left] Right

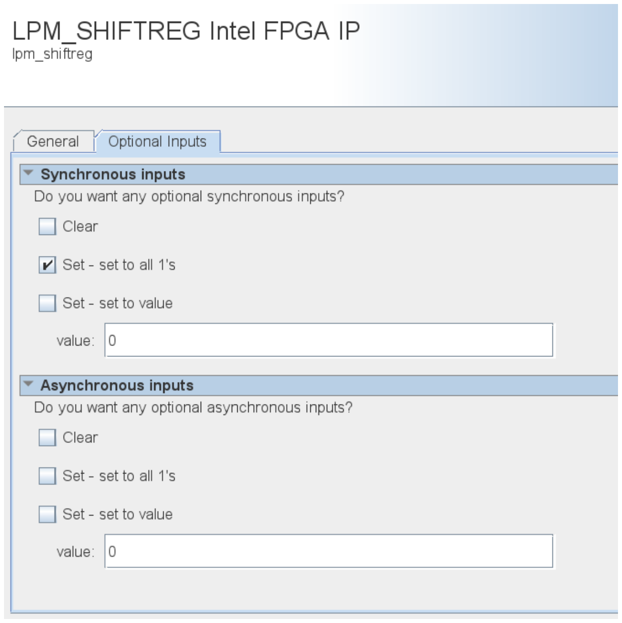
Which outputs do you want (select at least one)?

- [ ] Data output
- [✓] Serial shift data output

Do you want any optional inputs?

- [ ] Clock enable input
- [ ] Serial shift data input
- [✓] Parallel data input (load)

Select **Optional Inputs** tab and select the **Set to all 1's** for Synchronous Inputs as



LPM_SHIFTREG Intel FPGA IP
lpm_shiftreg

General | Optional Inputs

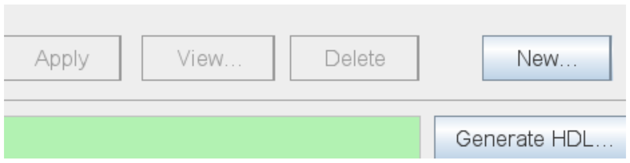
Synchronous inputs
Do you want any optional synchronous inputs?
- [ ] Clear
- [x] Set - set to all 1's
- [ ] Set - set to value

value: 0

Asynchronous inputs
Do you want any optional asynchronous inputs?
- [ ] Clear
- [ ] Set - set to all 1's
- [ ] Set - set to value

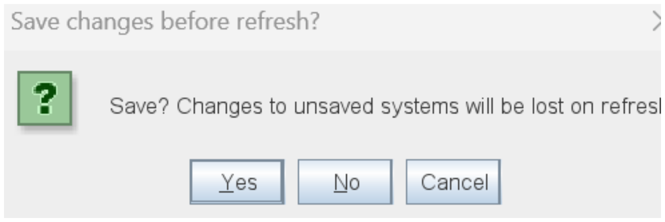
value: 0

Select **Generate HDL** in bottom right of the window as



Maintain the default settings and select **Generate.**

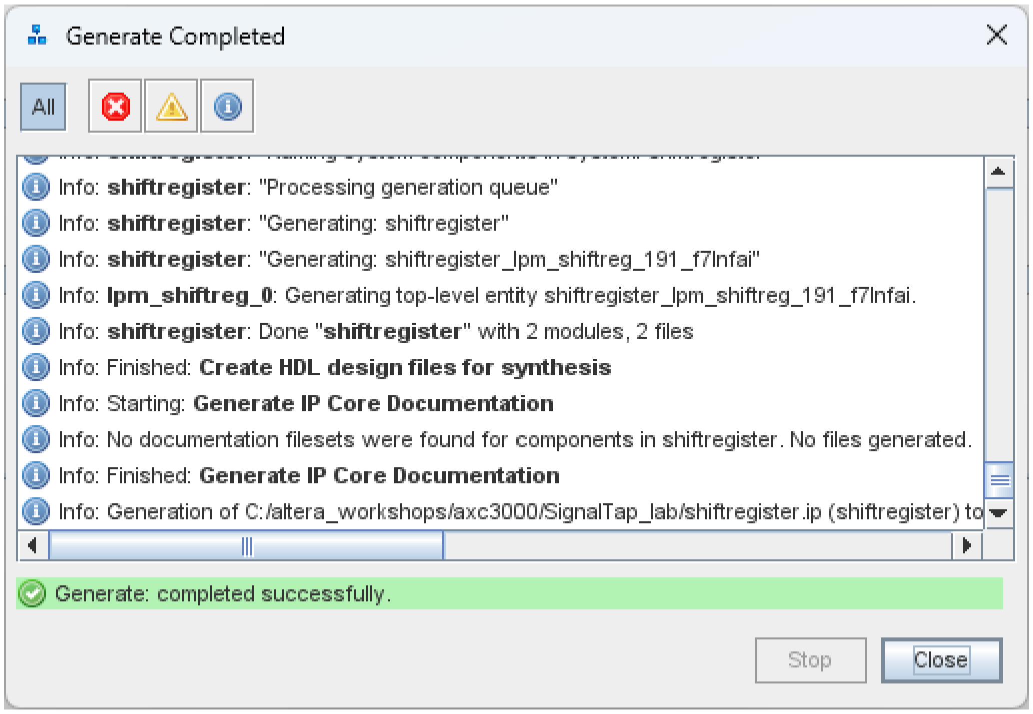
Select **Yes** to the window of Save? Changes to unsaved systems will be lost on refresh



Save changes before refresh?

Save? Changes to unsaved systems will be lost on refresl

Once the Generation is complete as below



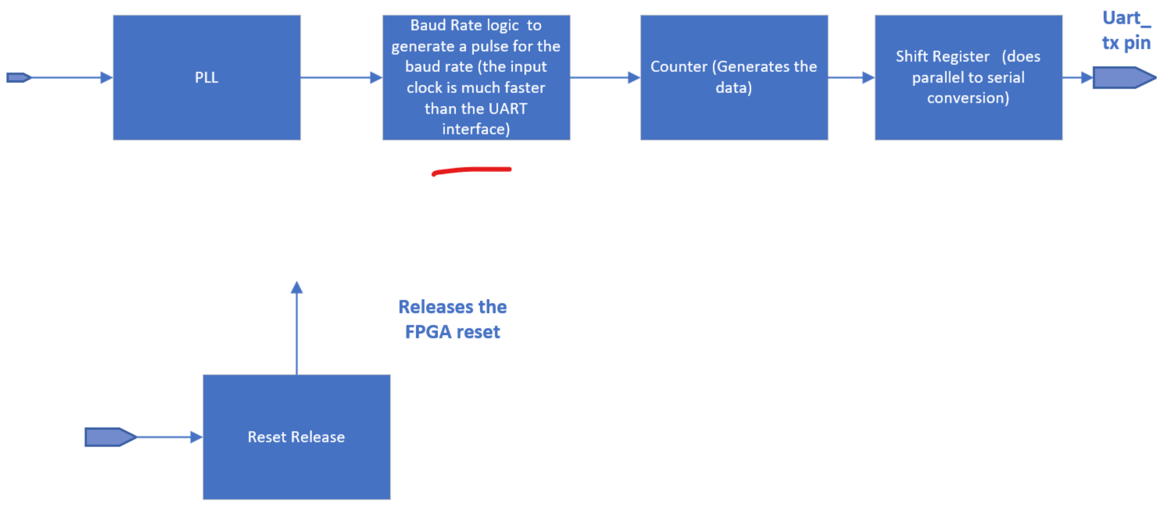
```
Info: shiftregister: "Processing generation queue"
Info: shiftregister: "Generating: shiftregister"
Info: shiftregister: "Generating: shiftregister_lpm_shiftreg_191_f7lnfai"
Info: lpm_shiftreg_0: Generating top-level entity shiftregister_lpm_shiftreg_191_f7lnfai.
Info: shiftregister: Done "shiftregister" with 2 modules, 2 files
Info: Finished: Create HDL design files for synthesis
Info: Starting: Generate IP Core Documentation
Info: No documentation filesets were found for components in shiftregister. No files generated.
Info: Finished: Generate IP Core Documentation
Info: Generation of C:/altera_workshops/axc3000/SignalTap_lab/shiftregister.ip (shiftregister) to
Generate: completed successfully.
```

Select **Close**

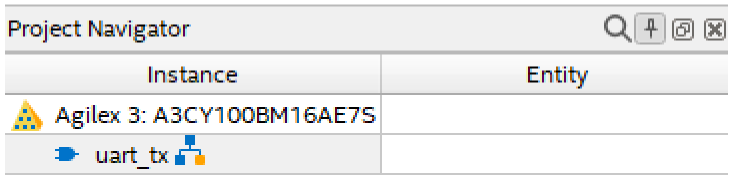
Select **File -> Exit** in the IP Parameter Window

4.3.5 Top Level File

A top-level file is needed for each project. In this case, the top-level file, which is completed for you, will “connect” together the various generated IP (PLL, counter, shift register, and reset release) as well as implement the baud rate logic.


PLL
Baud Rate logic to generate a pulse for the baud rate (the input clock is much faster than the UART interface)
Counter (Generates the data)
Shift Register (does parallel to serial conversion)
Uart_ tx pin
Releases the FPGA reset
Reset Release

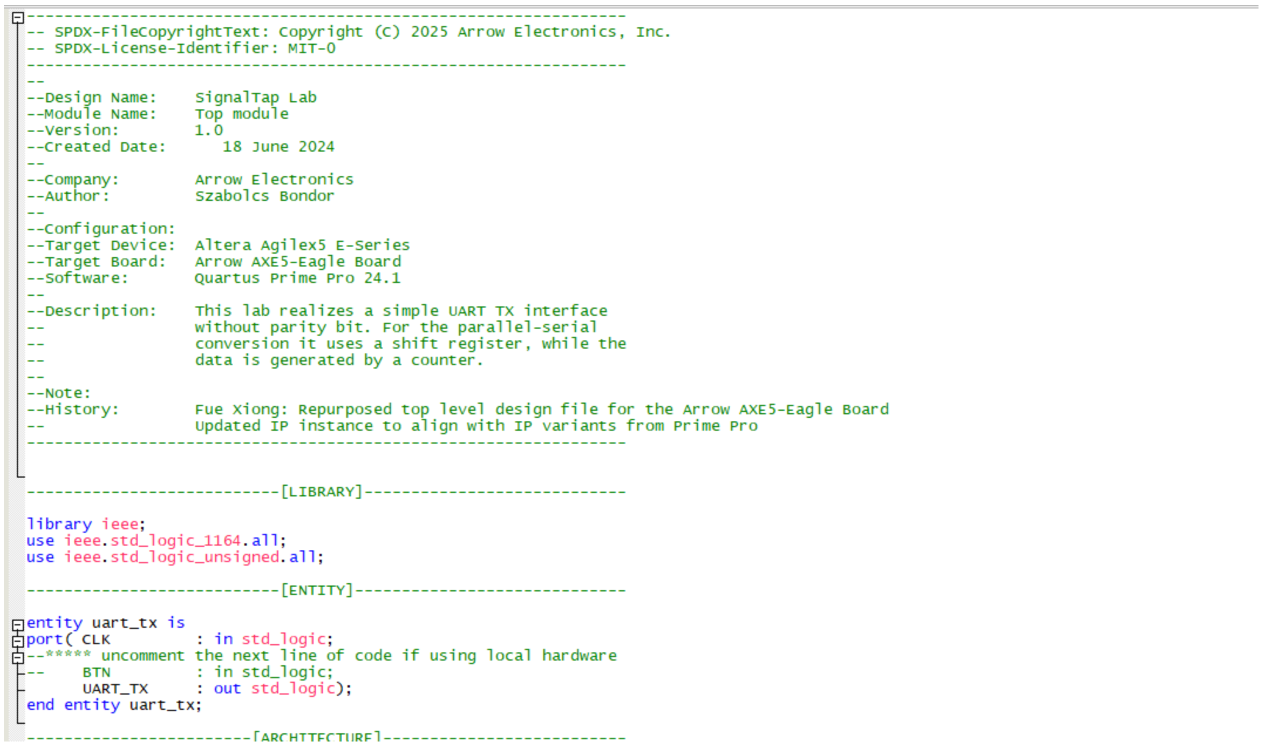
In Quartus, in the Project Navigator section (top left of the Quartus window), **double-click on the uart_tx** to open it



Project Navigator

Instance	Entity
Agilex 3: A3CY100BM16AE7S	
uart_tx	

The top of the file will look like:



```
-----------------------------------------------------------------
-- SPDX-FileCopyrightText: Copyright (C) 2025 Arrow Electronics, Inc.
-- SPDX-License-Identifier: MIT-0
-----------------------------------------------------------------
--
--Design Name:    SignalTap Lab
--Module Name:    Top module
--Version:        1.0
--Created Date:      18 June 2024
--
--Company:        Arrow Electronics
--Author:         Szabolcs Bondor
--
--Configuration:
--Target Device:  Altera Agilex5 E-Series
--Target Board:   Arrow AXE5-Eagle Board
--Software:       Quartus Prime Pro 24.1
--
--Description:    This lab realizes a simple UART TX interface
--                without parity bit. For the parallel-serial
--                conversion it uses a shift register, while the
--                data is generated by a counter.
--
--Note:
--History:        Fue Xiong: Repurposed top level design file for the Arrow AXE5-Eagle Board
--                Updated IP instance to align with IP variants from Prime Pro
-----------------------------------------------------------------


---------------------------[LIBRARY]----------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

---------------------------[ENTITY]-----------------------------

entity uart_tx is
port( CLK          : in std_logic;
--***** uncomment the next line of code if using local hardware
--      BTN        : in std_logic;
        UART_TX    : out std_logic);
end entity uart_tx;

------------------------[ARCHITECTURE]--------------------------
```

Scroll down in this file to review the connections.

This section calls the library files (which define the various functions that are used in the file). The entity section defines the inputs (CLK and a button) and output pin (UART_TX). The signal section defines the “wires” that connect items together. The sections called component <name> “tell” this top-level file that there will be sub-files i.e. the component counter.

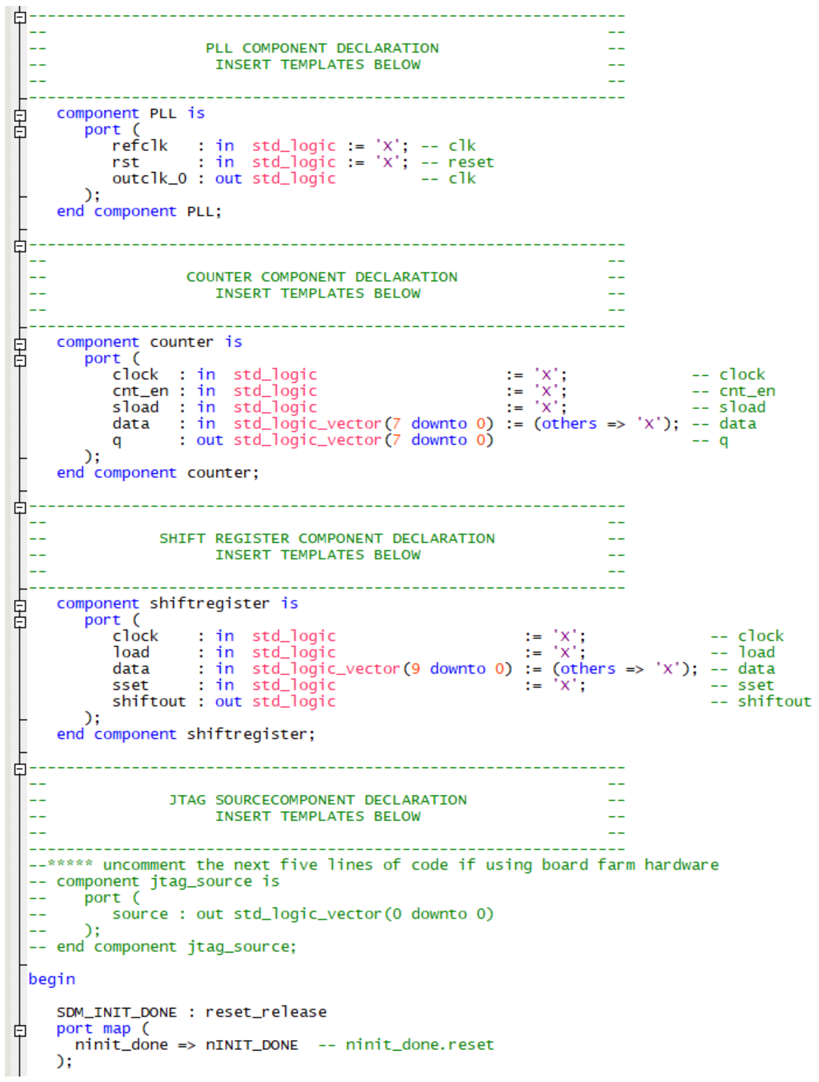
The port map section makes the connections from the sub-files to the top level file.

For example, for

```
clock : PLL
port map(refclk   => CLK,
         rst      => nINIT_DONE,
            outclk_0 => iCLK);
```

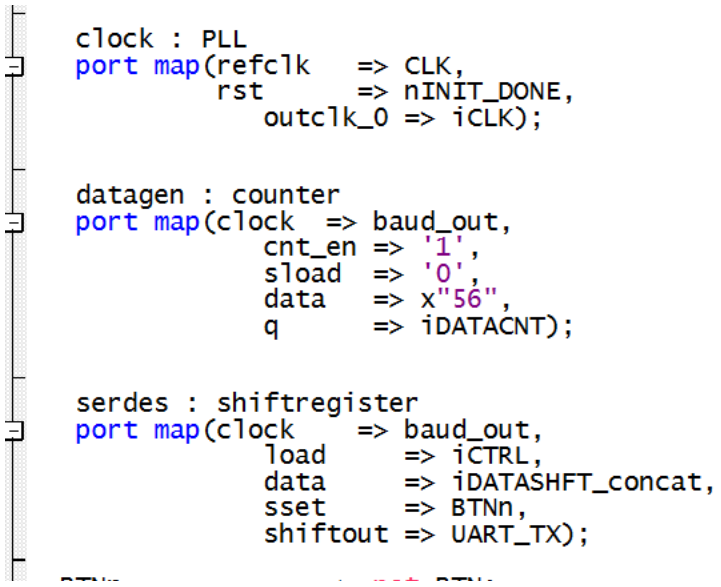
The word clock is an arbitrary word, while the PLL means that it looks for the sub-file PLL that you created earlier. The port map connects the signal refclk from the sub-file PLL to the signal CLK which is in this top-level file.

Note in particular, the components of the various IP i.e. PLL, counter, shift register, and reset are added for you in this file, along with the connection i.e. port mapping.



```
---------------------------------------------------------------------
--                                                                 --
--                   PLL COMPONENT DECLARATION                     --
--                    INSERT TEMPLATES BELOW                       --
--                                                                 --
---------------------------------------------------------------------
   component PLL is
      port (
         refclk   : in  std_logic := 'X'; -- clk
         rst      : in  std_logic := 'X'; -- reset
         outclk_0 : out std_logic         -- clk
      );
   end component PLL;

---------------------------------------------------------------------
--                                                                 --
--                 COUNTER COMPONENT DECLARATION                   --
--                    INSERT TEMPLATES BELOW                       --
--                                                                 --
---------------------------------------------------------------------
   component counter is
      port (
         clock  : in  std_logic                    := 'X';             -- clock
         cnt_en : in  std_logic                    := 'X';             -- cnt_en
         sload  : in  std_logic                    := 'X';             -- sload
         data   : in  std_logic_vector(7 downto 0) := (others => 'X'); -- data
         q      : out std_logic_vector(7 downto 0)                     -- q
      );
   end component counter;

---------------------------------------------------------------------
--                                                                 --
--             SHIFT REGISTER COMPONENT DECLARATION                --
--                    INSERT TEMPLATES BELOW                       --
--                                                                 --
---------------------------------------------------------------------
   component shiftregister is
      port (
         clock    : in  std_logic                    := 'X';             -- clock
         load     : in  std_logic                    := 'X';             -- load
         data     : in  std_logic_vector(9 downto 0) := (others => 'X'); -- data
         sset     : in  std_logic                    := 'X';             -- sset
         shiftout : out std_logic                                        -- shiftout
      );
   end component shiftregister;

---------------------------------------------------------------------
--                                                                 --
--              JTAG SOURCECOMPONENT DECLARATION                   --
--                    INSERT TEMPLATES BELOW                       --
--                                                                 --
---------------------------------------------------------------------
--***** uncomment the next five lines of code if using board farm hardware
-- component jtag_source is
--    port (
--       source : out std_logic_vector(0 downto 0)
--    );
-- end component jtag_source;

begin

   SDM_INIT_DONE : reset_release
   port map (
     ninit_done => nINIT_DONE   -- ninit_done.reset
   );
```

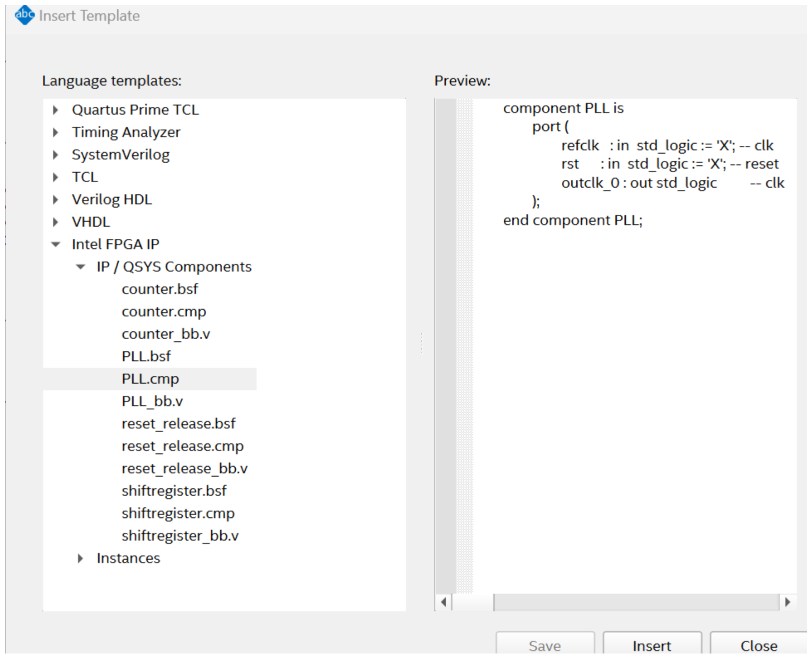
Note areas that must be uncommented, depending on whether validation is done using hardware locally or located in the board farm. The connections are shown below.



```
clock : PLL
port map(refclk    => CLK,
         rst       => nINIT_DONE,
            outclk_0 => iCLK);


datagen : counter
port map(clock   => baud_out,
            cnt_en => '1',
            sload  => '0',
            data   => x"56",
            q      => iDATACNT);


serdes : shiftregister
port map(clock    => baud_out,
            load     => iCTRL,
            data     => iDATASHFT_concat,
            sset     => BTNn,
            shiftout => UART_TX);
```

Note the PLL, counter, shift register and reset release components are included in this file, and these are connected to other logic.

No file changes are needed because the components are already added, but for reference, if you right-click in the file and select Insert Template -> Intel FPGA IP -> files with the .cmp, the component instantiation can be entered. **Do not make any changes.**



Insert Template

Language templates:

- Quartus Prime TCL
- Timing Analyzer
- SystemVerilog
- TCL
- Verilog HDL
- VHDL
- Intel FPGA IP
  - IP / QSYS Components
    - counter.bsf
    - counter.cmp
    - counter_bb.v
    - PLL.bsf
    - PLL.cmp
    - PLL_bb.v
    - reset_release.bsf
    - reset_release.cmp
    - reset_release_bb.v
    - shiftregister.bsf
    - shiftregister.cmp
    - shiftregister_bb.v
  - Instances

Preview:

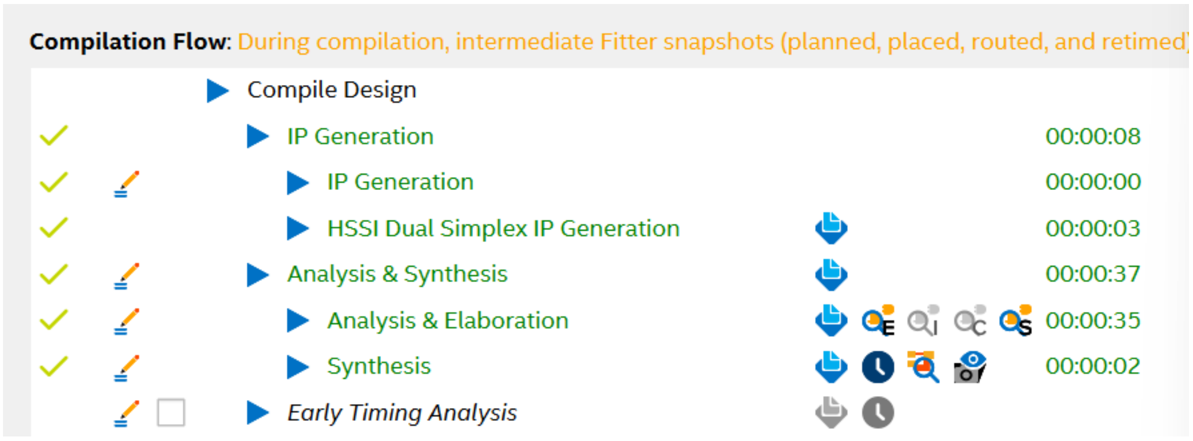
```
component PLL is
    port (
        refclk   : in  std_logic := 'X'; -- clk
        rst      : in  std_logic := 'X'; -- reset
        outclk_0 : out std_logic         -- clk
    );
end component PLL;
```

Optional logic has also been included to support testing in the remote board farm. Please note those lines of code available for the local or remote testing options. Please uncomment those lines of code as directed.

4.3.6 Analysis and Synthesis

Analysis and synthesis is where the design files are converted into a netlist that Quartus can use for later steps. Analysis and Synthesis will go further than Analysis and Elaboration in the flow. Quartus checks the legality of the connections. The netlist is used by Quartus so that the timing, IO and other constraints can be assigned.

Run Analysis and Synthesis by clicking on button on the toolbars, or **Processing** → **Start** → **Analysis and Synthesis**.



Compilation Flow: During compilation, intermediate Fitter snapshots (planned, placed, routed, and retimed)

		Compile Design	
✓		IP Generation	00:00:08
✓		IP Generation	00:00:00
✓		HSSI Dual Simplex IP Generation	00:00:03
✓		Analysis & Synthesis	00:00:37
✓		Analysis & Elaboration	00:00:35
✓		Synthesis	00:00:02
	☐	*Early Timing Analysis*	

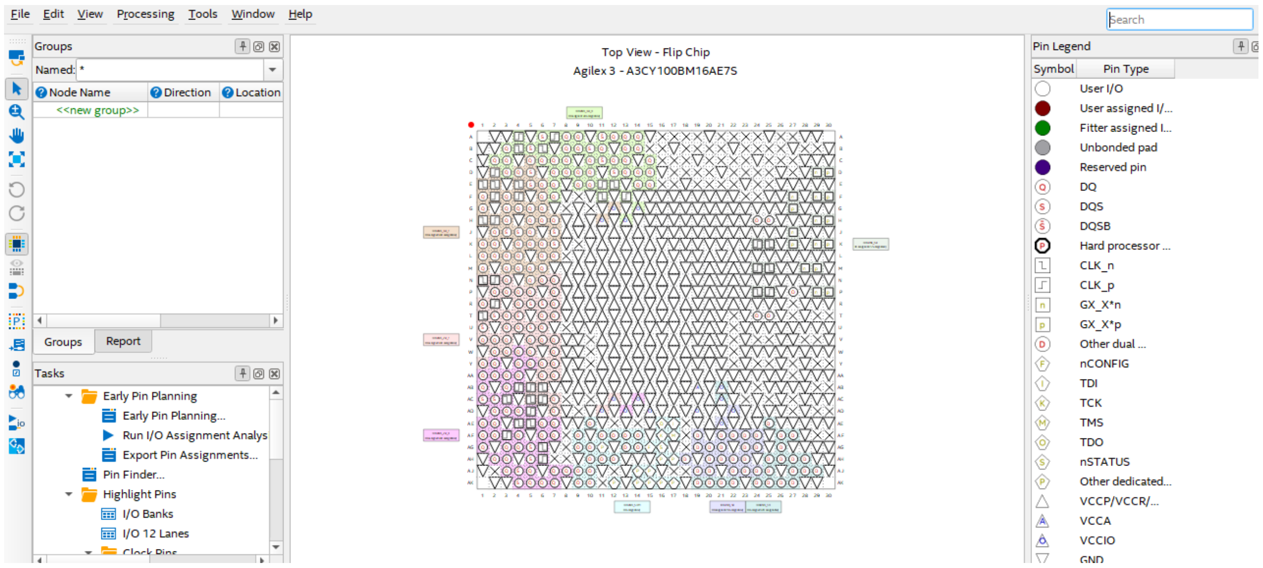
The items in green, with the yellow checkmarks show the steps that Quartus completed.

If there are errors (reported with be red text), they will need to be corrected before continuing. The time taken for this step will vary depending on the computer used. Once Synthesis process is completed, you can continue to the next step.

4.3.6.1 Pin Assignments

Pin Assignments are needed so the correct pins are assigned to the correct pins on the board. Without pin assignments, Quartus will use default assignments which will not correspond to the board pins.

Open **Pin Planner** by clicking on button on the toolbars, or **Assignments** → **Pin Planner**



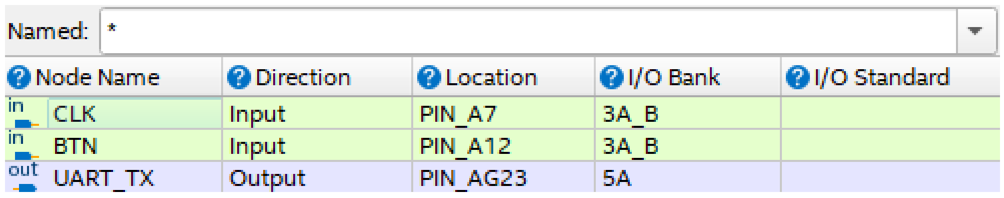
There are different ways to enter pins. A common way is to use a TCL script (which can be used for many pins). The way that we will do it in this workshop is to use the GUI as there are only 3 pins.

At the bottom of the window, there are various pins. The 2 inputs (BTN and CLK) and 1 output (UART_TX) need to have assigned pin locations and I/O standards.

These pins need location and I/O bank standards assigned so that Quartus will place the pins in the correct location (i.e. correspond to the board schematic and board layout) and have the correct voltage standards as defined by the board (to avoid conflict). Without assignments, Quartus will make assumptions on the locations, which will not match the board.

Note: If you are using the remote option, BTN might not be an available Node Name.

Select the Location field associated with each pin and type the following

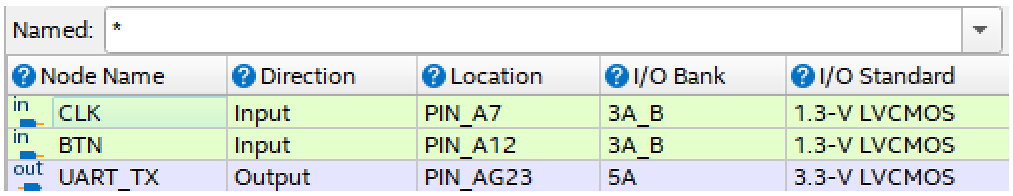


Named: *

Node Name	Direction	Location	I/O Bank	I/O Standard
in CLK	Input	PIN_A7	3A_B	
in BTN	Input	PIN_A12	3A_B	
out UART_TX	Output	PIN_AG23	5A	

The pull-down menu would also work. The corresponding I/O bank will be entered automatically.

Use the pull-down menu to select the I/O standards for each pin



Named: *

	Node Name	Direction	Location	I/O Bank	I/O Standard
in	CLK	Input	PIN_A7	3A_B	1.3-V LVCMOS
in	BTN	Input	PIN_A12	3A_B	1.3-V LVCMOS
out	UART_TX	Output	PIN_AG23	5A	3.3-V LVCMOS

4.3.6.2 Customize Columns

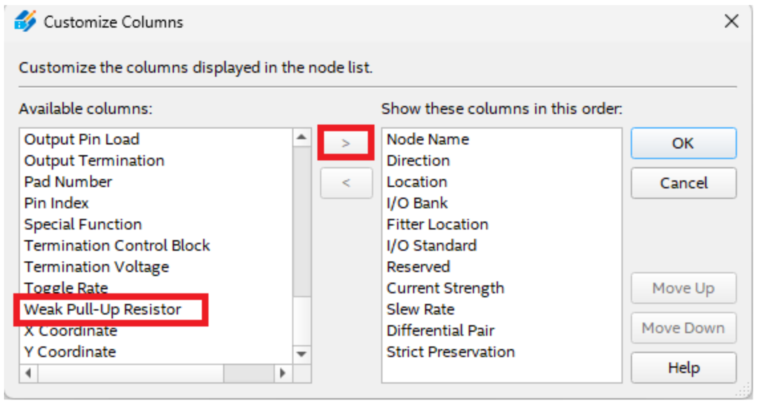
Right click in the spreadsheet part of Pin Planner. Select Customize Columns.by selecting File → Close. The settings are automatically saved.

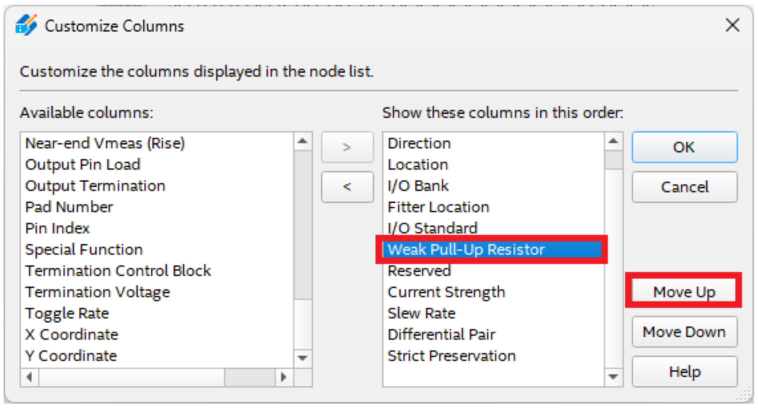
4.3.6.3 Add the Weak Pull-Up Resistor column.

Select Weak Pull-up Resistor in Available Columns and press the > button. Select Weak Pull-up Resistor in the right hand column and then use the Move-Up button to place at under the I/O Standard column selection.

4.3.6.4 Close the Customize Columns window.

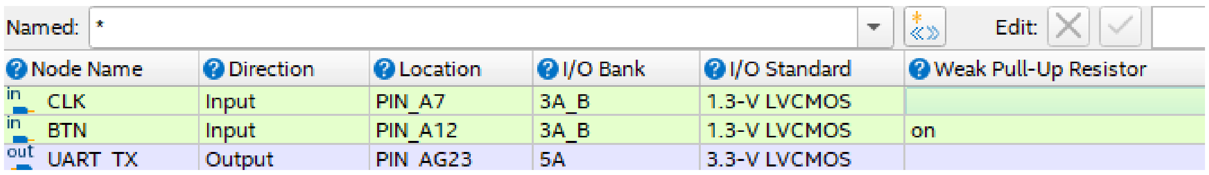
Select **OK**.





4.3.6.5 Enable Pullup

Select on in the Weak Pull-up Resistor column for the BUTTON Node Name



Named: *   Edit:

Node Name	Direction	Location	I/O Bank	I/O Standard	Weak Pull-Up Resistor
in CLK	Input	PIN_A7	3A_B	1.3-V LVCMOS	
in BTN	Input	PIN_A12	3A_B	1.3-V LVCMOS	on
out UART_TX	Output	PIN_AG23	5A	3.3-V LVCMOS	

Close the Pin Planner by selecting **File** → **Close**. The settings are automatically saved.

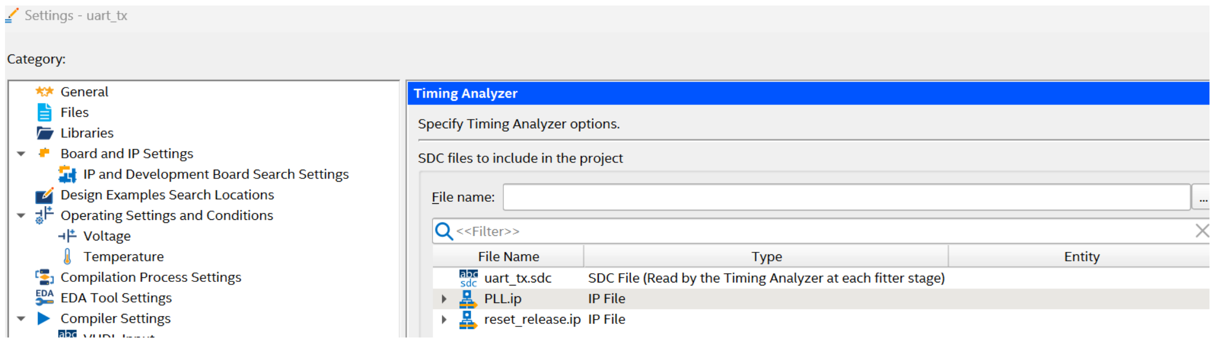
4.3.7 Timing Constraints

Timing Constraints are very important in a FPGA because they instruct Quartus on how fast/slow different parts of the design are to run. Quartus uses these constraints to meet/exceed the assignments. Providing accurate timing will enable the software to easily create a FPGA design. If the timing is over constrained the software will spend longer trying to meet timing.

Timing constraints for the design are defined in a .sdc file, where sdc stands for Synopsys Design Constraint.

Quartus automatically generates and uses a .sdc file for the Altera PLL, thus the .sdc file does not need to be generated. It generates the .sdc file when the settings in the PLL IP are set. The .sdc file for the Reset Release IP is also automatically set and added. The .sdc file was also imported into the design.

Select **Assignments** → **Settings -> Timing Analyzer** to see the timing constraint files.



Specify Timing Analyzer options.

SDC files to include in the project

File name:

File Name	Type	Entity
uart_tx.sdc	SDC File (Read by the Timing Analyzer at each fitter stage)	
PLL.ip	IP File	
reset_release.ip	IP File	

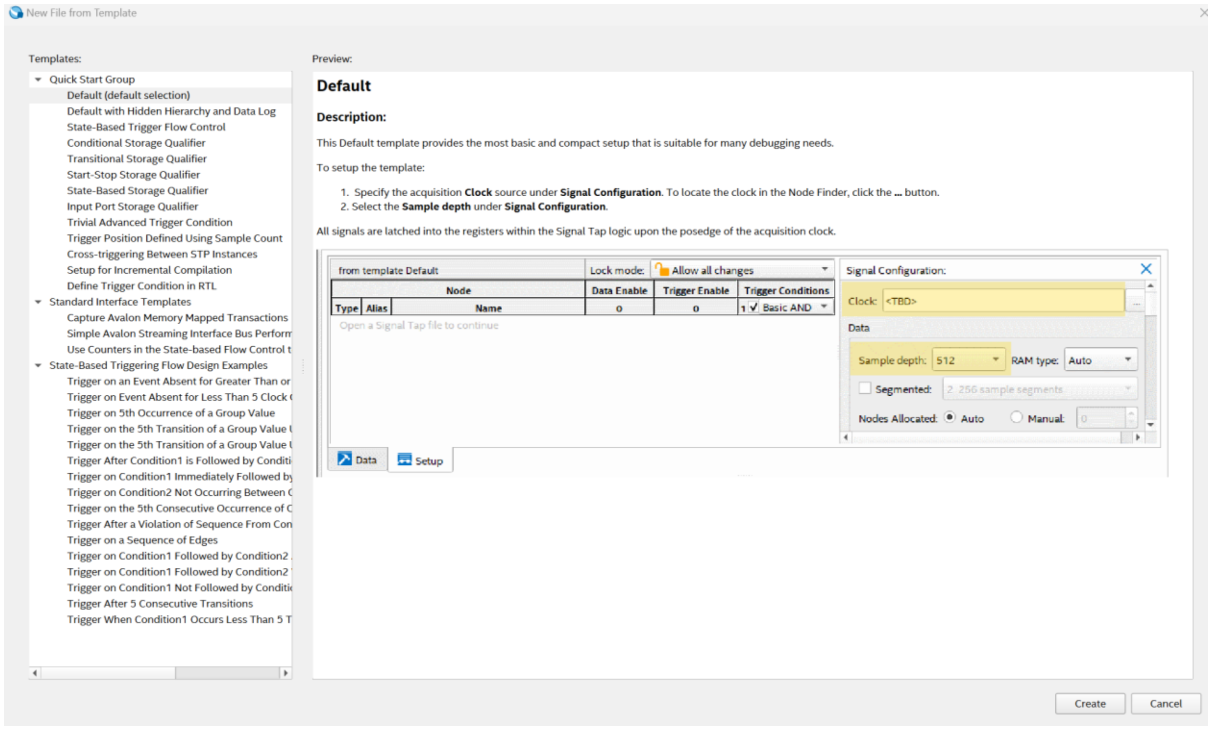
Select **Cancel** (bottom right of the window) as no changes were implemented.

4.3.8 Configuring the Signal Tap Logic Analyzer

The Signal Tap Logic Analyzer is an embedded logic analyzer. Before the Signal Tap Logic Analyzer can be used to capture and analyze data, it needs to be configured. The configuration will set up the clock, sample depth (how many samples and thus how many memory blocks will be required) and more.

Click Tools -> Signal Tap Logic Analyzer to open this tool. The default template will look like this.

There are various templates that can be used, but the default in this case will be used.



Templates:

- Quick Start Group
  - Default (default selection)
  - Default with Hidden Hierarchy and Data Log
  - State-Based Trigger Flow Control
  - Conditional Storage Qualifier
  - Transitional Storage Qualifier
  - Start-Stop Storage Qualifier
  - State-Based Storage Qualifier
  - Input Port Storage Qualifier
  - Trivial Advanced Trigger Condition
  - Trigger Position Defined Using Sample Count
  - Cross-triggering Between STP Instances
  - Setup for Incremental Compilation
  - Define Trigger Condition in RTL
- Standard Interface Templates
  - Capture Avalon Memory Mapped Transactions
  - Simple Avalon Streaming Interface Bus Perform
  - Use Counters in the State-based Flow Control t
- State-Based Triggering Flow Design Examples
  - Trigger on an Event Absent for Greater Than or
  - Trigger on Event Absent for Less Than 5 Clock
  - Trigger on 5th Occurrence of a Group Value
  - Trigger on the 5th Transition of a Group Value
  - Trigger on the 5th Transition of a Group Value
  - Trigger After Condition1 is Followed by Conditi
  - Trigger on Condition1 Immediately Followed by
  - Trigger on Condition2 Not Occurring Between C
  - Trigger on the 5th Consecutive Occurrence of C
  - Trigger After a Violation of Sequence From Con
  - Trigger on a Sequence of Edges
  - Trigger on Condition1 Followed by Condition2
  - Trigger on Condition1 Followed by Condition2
  - Trigger on Condition1 Not Followed by Conditi
  - Trigger After 5 Consecutive Transitions
  - Trigger When Condition1 Occurs Less Than 5 T

Preview:

Default

Description:

This Default template provides the most basic and compact setup that is suitable for many debugging needs.

To setup the template:

1. Specify the acquisition **Clock** source under **Signal Configuration**. To locate the clock in the Node Finder, click the **...** button.
2. Select the **Sample depth** under **Signal Configuration**.

All signals are latched into the registers within the Signal Tap logic upon the posedge of the acquisition clock.

from template Default — Lock mode: Allow all changes

Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
			0	0	1 ✓ Basic AND

Open a Signal Tap file to continue

Signal Configuration:

Clock: <TBD>

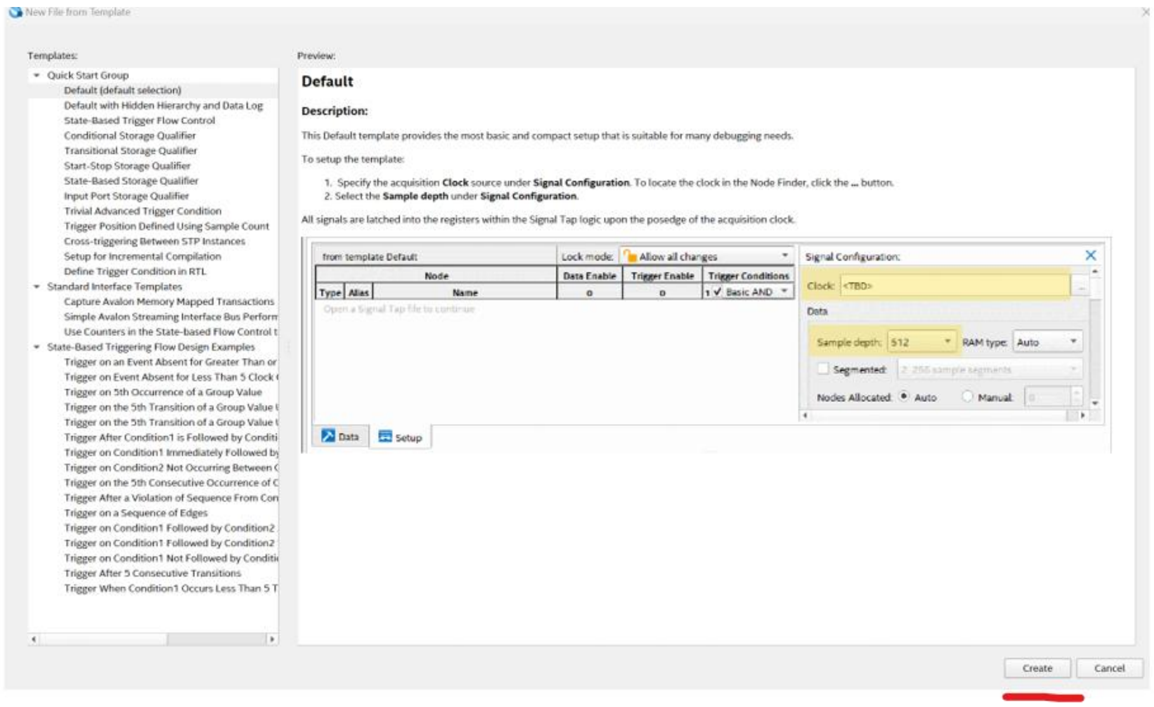
Data

Sample depth: 512 RAM type: Auto

Segmented: 2 256 sample segments

Nodes Allocated: ● Auto ○ Manual: 0

Select **Create** (bottom right of the window) as



Templates:

- Quick Start Group
  - Default (default selection)
  - Default with Hidden Hierarchy and Data Log
  - State-Based Trigger Flow Control
  - Conditional Storage Qualifier
  - Transitional Storage Qualifier
  - Start-Stop Storage Qualifier
  - State-Based Storage Qualifier
  - Input Port Storage Qualifier
  - Trivial Advanced Trigger Condition
  - Trigger Position Defined Using Sample Count
  - Cross-triggering Between STP Instances
  - Setup for Incremental Compilation
  - Define Trigger Condition in RTL
- Standard Interface Templates
  - Capture Avalon Memory Mapped Transactions
  - Simple Avalon Streaming Interface Bus Perform
  - Use Counters in the State-based Flow Control t
- State-Based Triggering Flow Design Examples
  - Trigger on an Event Absent for Greater Than or
  - Trigger on Event Absent for Less Than 5 Clock
  - Trigger on 5th Occurrence of a Group Value
  - Trigger on the 5th Transition of a Group Value
  - Trigger on the 5th Transition of a Group Value
  - Trigger After Condition1 is Followed by Conditi
  - Trigger on Condition1 Immediately Followed by
  - Trigger on Condition2 Not Occurring Between C
  - Trigger on the 5th Consecutive Occurrence of C
  - Trigger After a Violation of Sequence From Con
  - Trigger on a Sequence of Edges
  - Trigger on Condition1 Followed by Condition2
  - Trigger on Condition1 Followed by Condition2
  - Trigger on Condition1 Not Followed by Conditi
  - Trigger After 5 Consecutive Transitions
  - Trigger When Condition1 Occurs Less Than 5 T

Preview:

Default

Description:

This Default template provides the most basic and compact setup that is suitable for many debugging needs.

To setup the template:

1. Specify the acquisition **Clock** source under **Signal Configuration**. To locate the clock in the Node Finder, click the **...** button.
2. Select the **Sample depth** under **Signal Configuration**.

All signals are latched into the registers within the Signal Tap logic upon the posedge of the acquisition clock.

from template Default — Lock mode: Allow all changes

Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
			0	0	1 ✓ Basic AND

Open a Signal Tap file to continue

Signal Configuration:

Clock: <TBD>

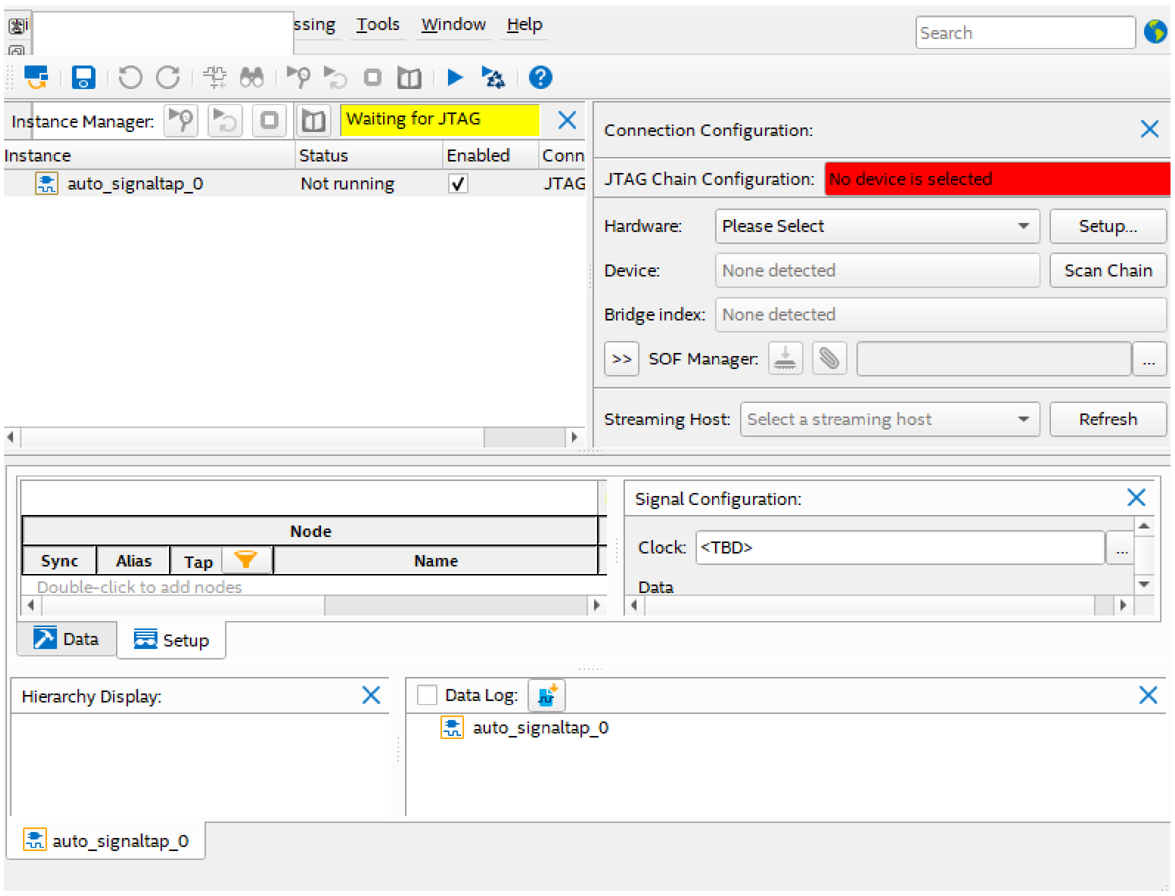
Data

Sample depth: 512 RAM type: Auto

Segmented: 2 256 sample segments

Nodes Allocated: ● Auto ○ Manual: 0

The default Signal Tap window will now appear as



Instance Manager: Waiting for JTAG

Instance	Status	Enabled	Conn
auto_signaltap_0	Not running	✓	JTAG

Connection Configuration:

JTAG Chain Configuration: No device is selected

Hardware: Please Select

Device: None detected

Bridge index: None detected

SOF Manager:

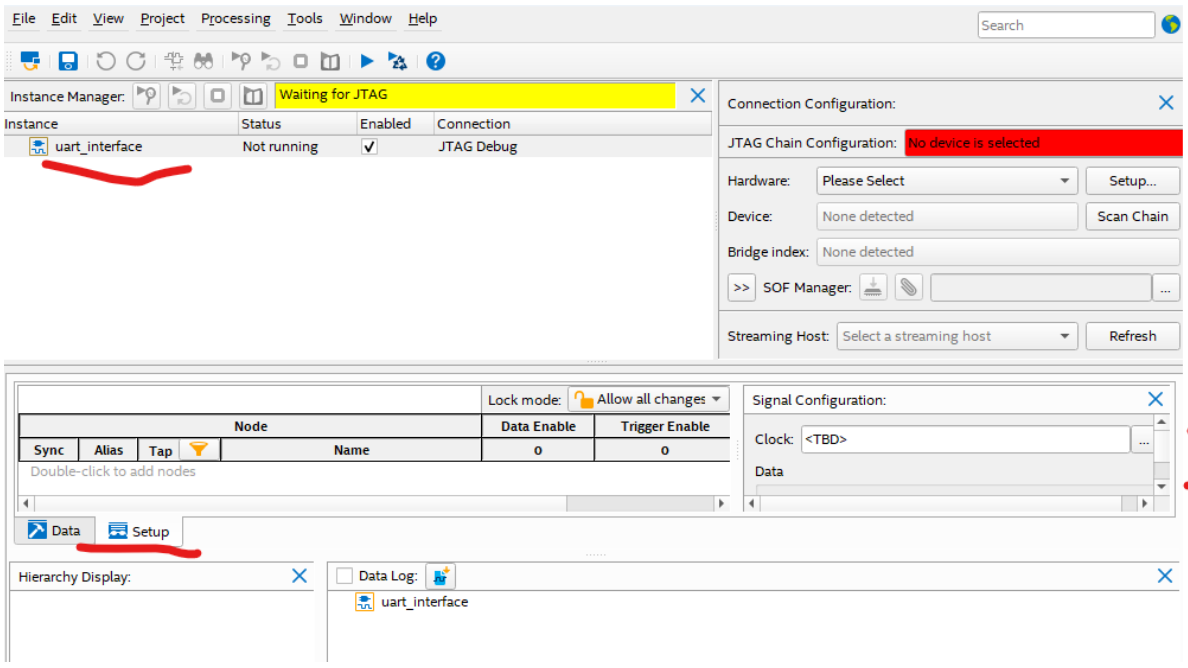
Streaming Host: Select a streaming host

Signal Configuration:

Clock: <TBD>

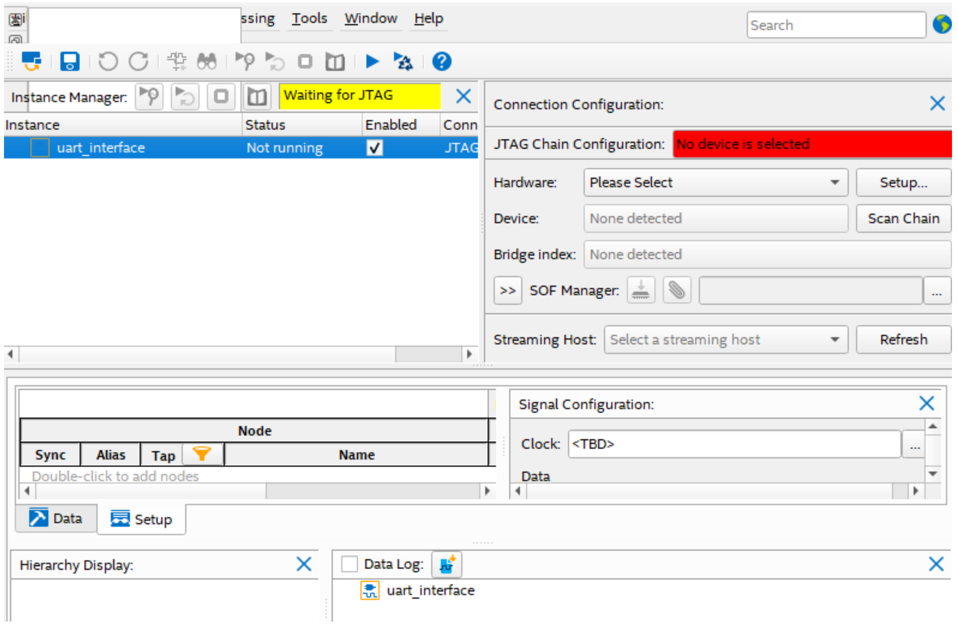
Double-click to add nodes

The next steps will make various changes to the initial Signal Tap window, such as shown with the red markings.


File Edit View Project Processing Tools Window Help
Search
Instance Manager:
Waiting for JTAG
Instance Status Enabled Connection
uart_interface Not running ✓ JTAG Debug
Connection Configuration:
JTAG Chain Configuration: No device is selected
Hardware: Please Select
Setup...
Device: None detected
Scan Chain
Bridge index: None detected
SOF Manager:
Streaming Host: Select a streaming host
Refresh
Lock mode: Allow all changes
Node
Data Enable
Trigger Enable
Sync Alias Tap Name 0 0
Double-click to add nodes
Signal Configuration:
Clock: <TBD>
Data
Data Setup
Hierarchy Display:
Data Log:
uart_interface

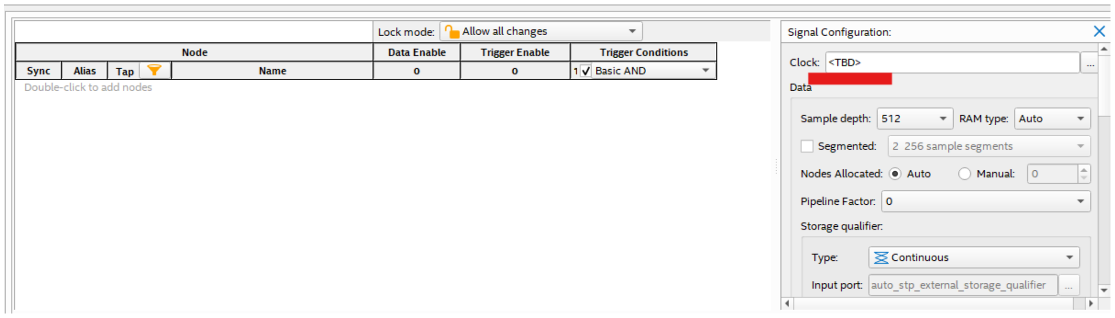
The first step will be to change the name.

Right click on the auto_signaltap_0 instance and select **Rename Instance** from the pop-up menu. **Rename it to uart_interface** as below. This will help keep track of different variations of the SignalTap Analyzers that can be instantiated.



The acquisition clock is needed before Signal Tap can be used.

In the Signal Configuration Window on the right-hand side of the Signal Tap Logic Analyzer window, **click on the** ... button to browse **the clock signal**.



Lock mode: Allow all changes

Sync | Alias | Tap | Name | Data Enable 0 | Trigger Enable 0 | Trigger Conditions 1 ✓ Basic AND

Double-click to add nodes

Signal Configuration:

Clock: <TBD>

Data

Sample depth: 512 RAM type: Auto

Segmented: 2 256 sample segments

Nodes Allocated: ● Auto ○ Manual: 0

Pipeline Factor: 0

Storage qualifier:

Type: Continuous

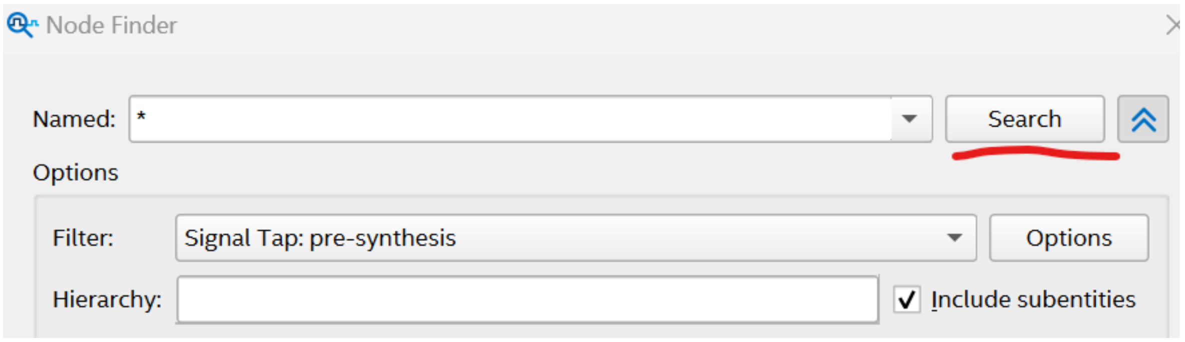
Input port: auto_stp_external_storage_qualifier

In the Node Finder window ensure the following are set:

Named: * (Asterisk is used as a wild card)

Filter: Signal Tap: pre-synthesis

Click on Search (underlined below)



Node Finder

Named: *

Search

Options

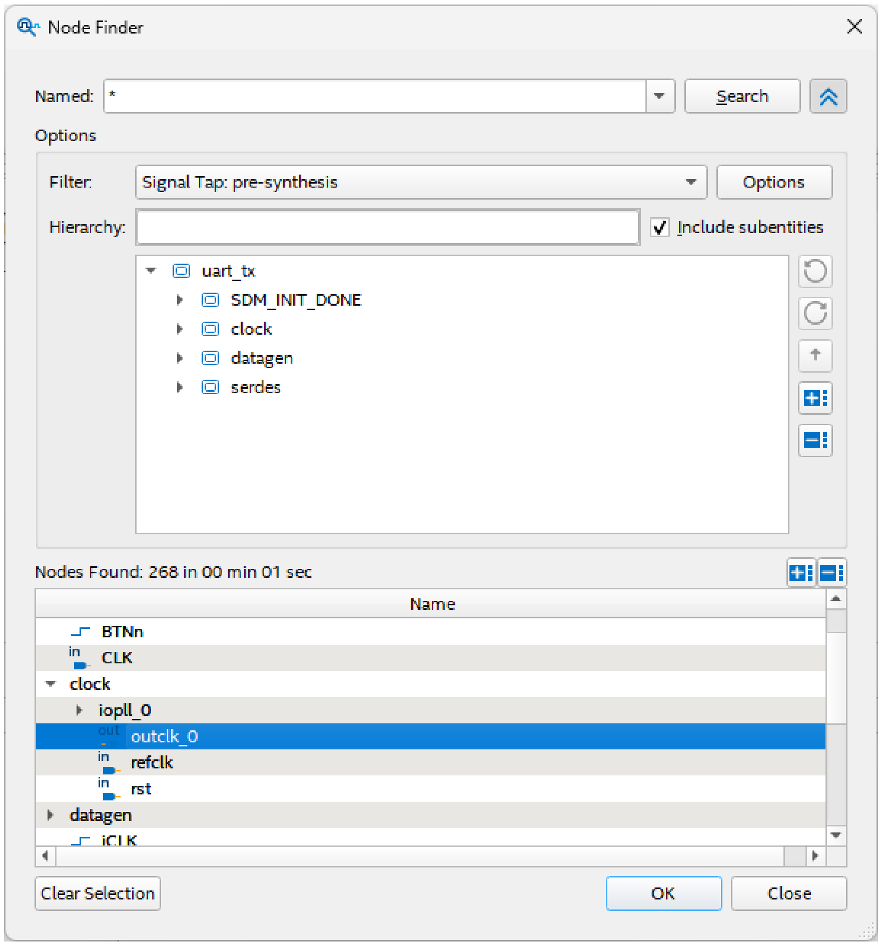
Filter: Signal Tap: pre-synthesis

Options

Hierarchy:

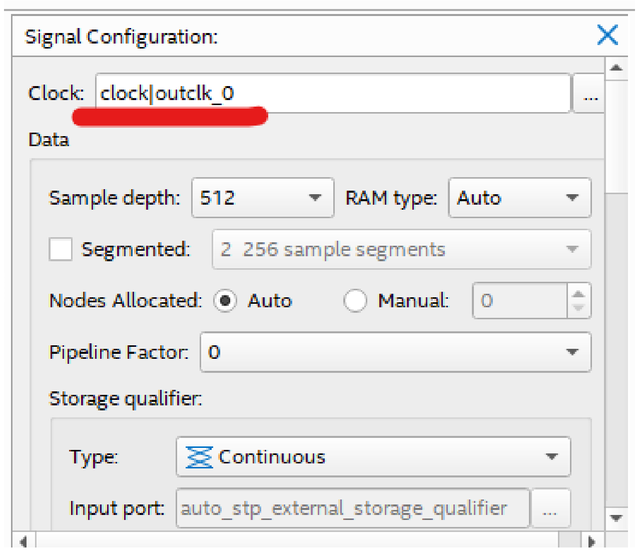
✓ Include subentities

Expand the clock section



And then **select outclk_0.** Click on the **OK** button or simply double click on the node.

The clock will now be set as



Signal Configuration:

Clock: clock|outclk_0

Data

Sample depth: 512 RAM type: Auto

Segmented: 2 256 sample segments

Nodes Allocated: (● Auto) ○ Manual: 0

Pipeline Factor: 0

Storage qualifier:

Type: Continuous

Input port: auto_stp_external_storage_qualifier

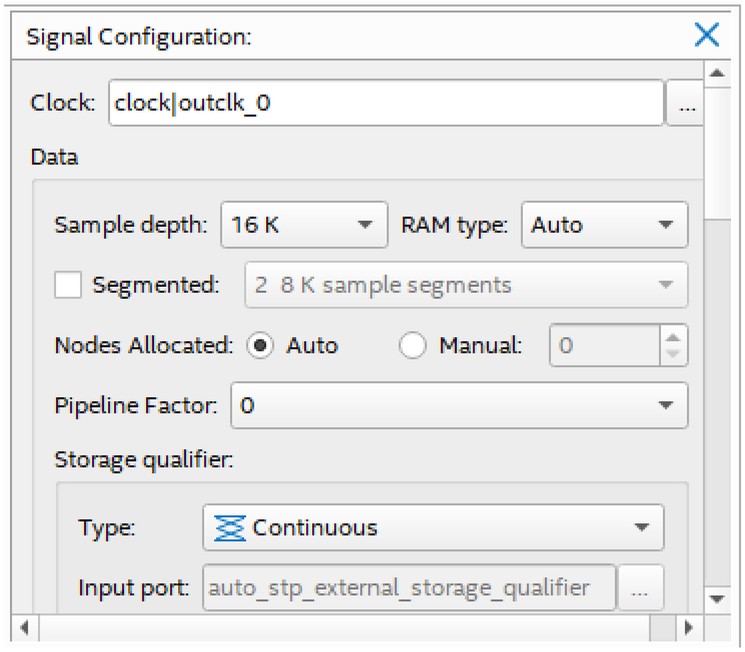
The sample depth (amount of memory) “determines the number of samples the logic analyzer captures and stores in the data buffer, for each signal”

There are different sample depths that are possible.

Set the Sample depth to 16K .

This will enable many more samples than the default value.

The rest of the settings are left as default, and it should look as follows.



Signal Configuration:

Clock: clock|outclk_0

Data

Sample depth: 16 K RAM type: Auto

Segmented: 2 8 K sample segments

Nodes Allocated: (● Auto) ○ Manual: 0

Pipeline Factor: 0

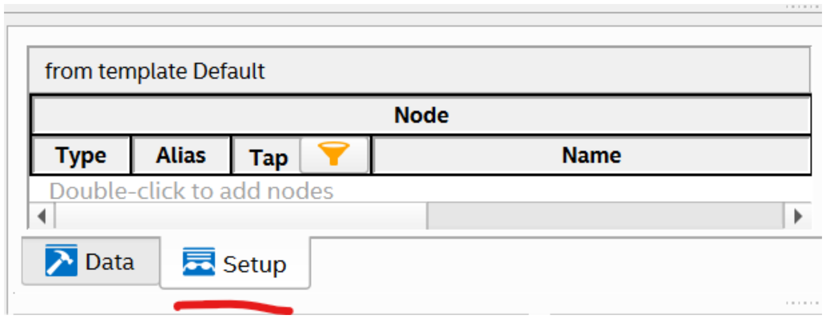
Storage qualifier:

Type: Continuous

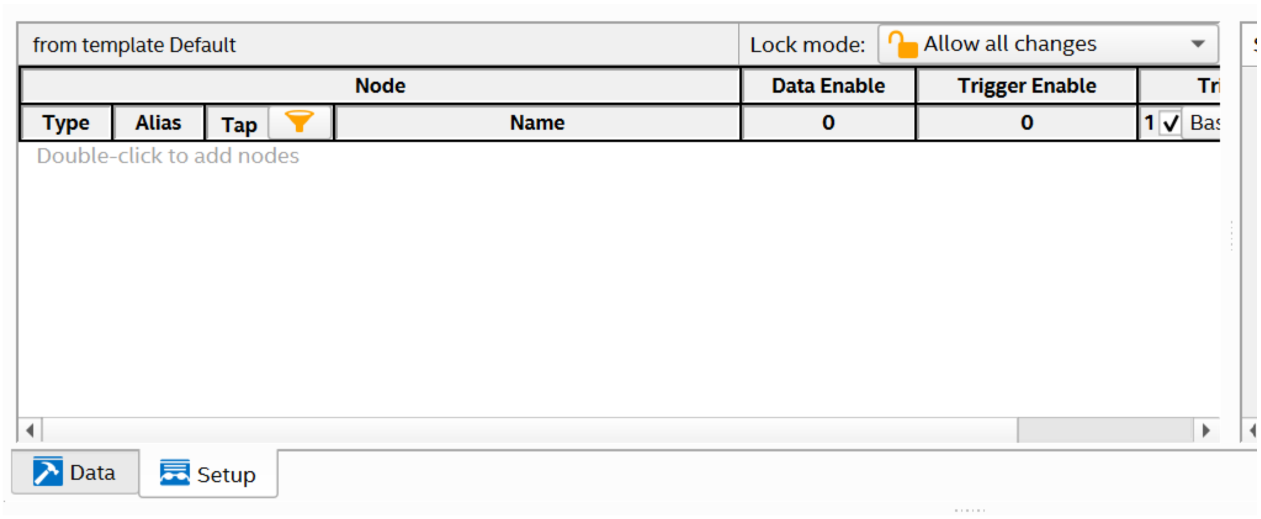
Input port: auto_stp_external_storage_qualifier

The next step is to add nodes (signals). These are the nodes that will be captured and displayed.

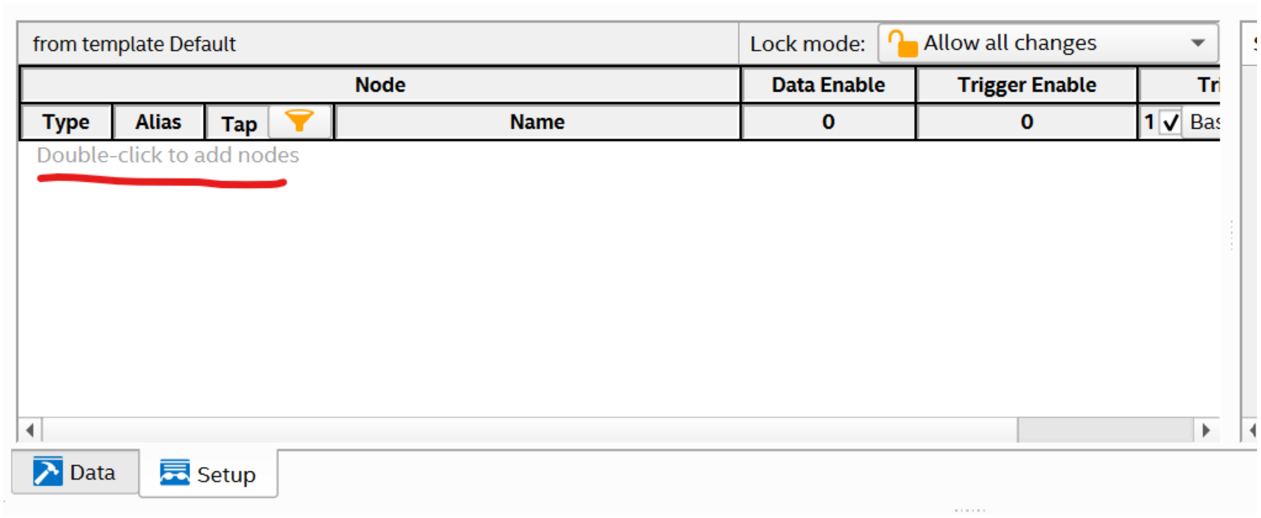
Click on the Setup tab as



And **“expand”** the window as



Double-click in the window to add nodes as


from template Default

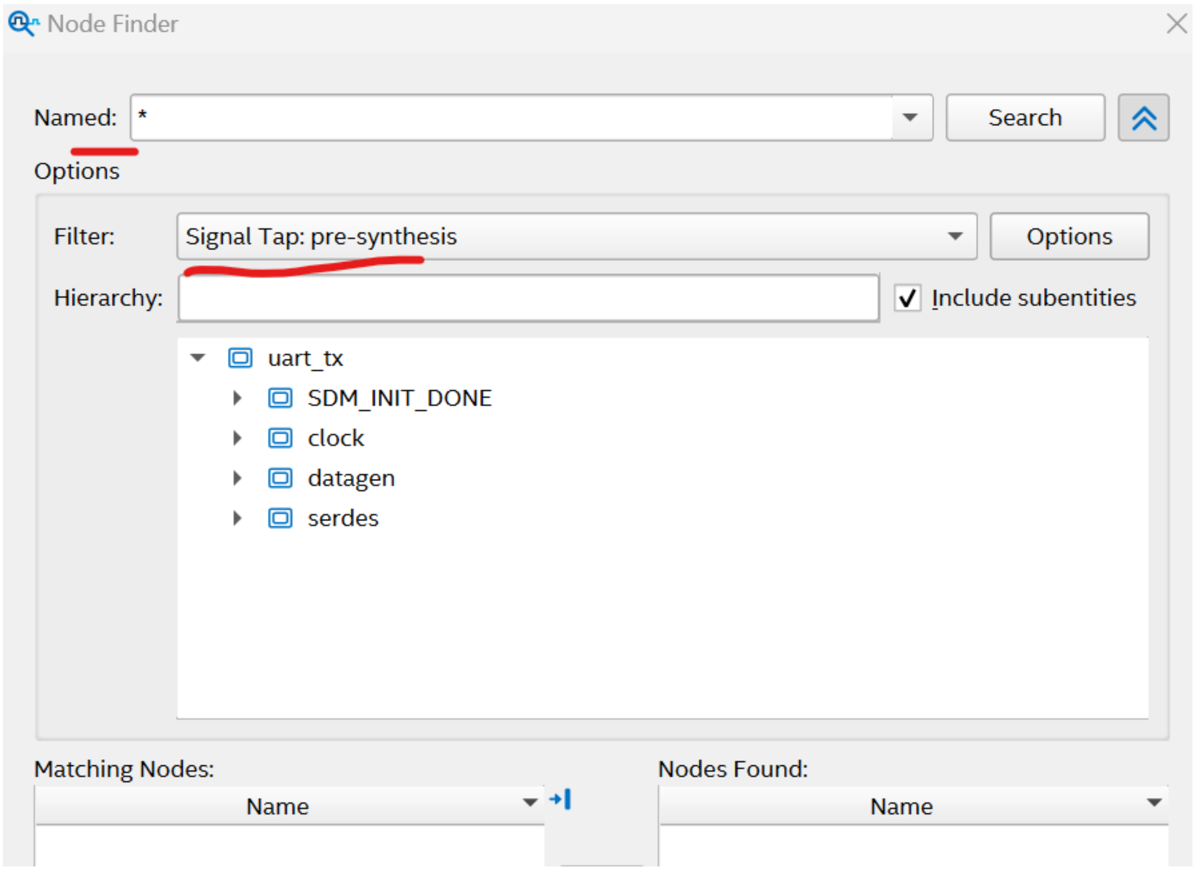
Lock mode: Allow all changes

Node				Data Enable	Trigger Enable	Tri
Type	Alias	Tap	Name	0	0	1 ✓ Bas

Double-click to add nodes

Type * (wild card) in the Named field.

The Filter: Signal Tap: pre-synthesis will be set by default. Pre-synthesis means that these are signals that are found before the full compile.


Node Finder

Named: *

Options

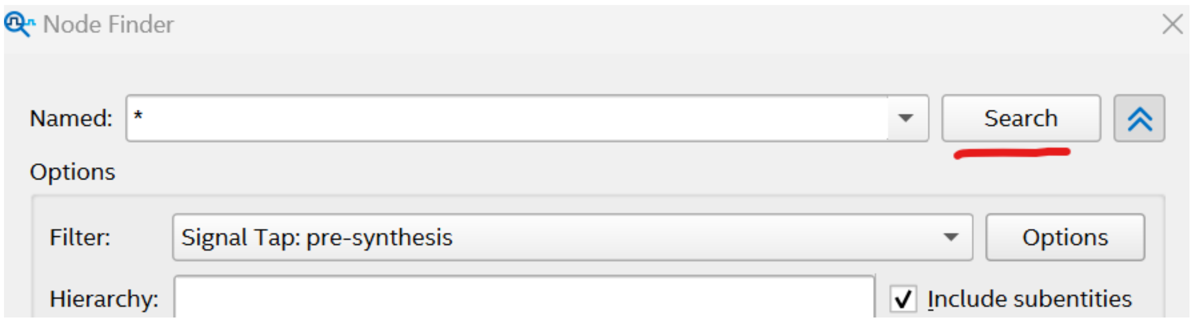
Filter: Signal Tap: pre-synthesis

Hierarchy:

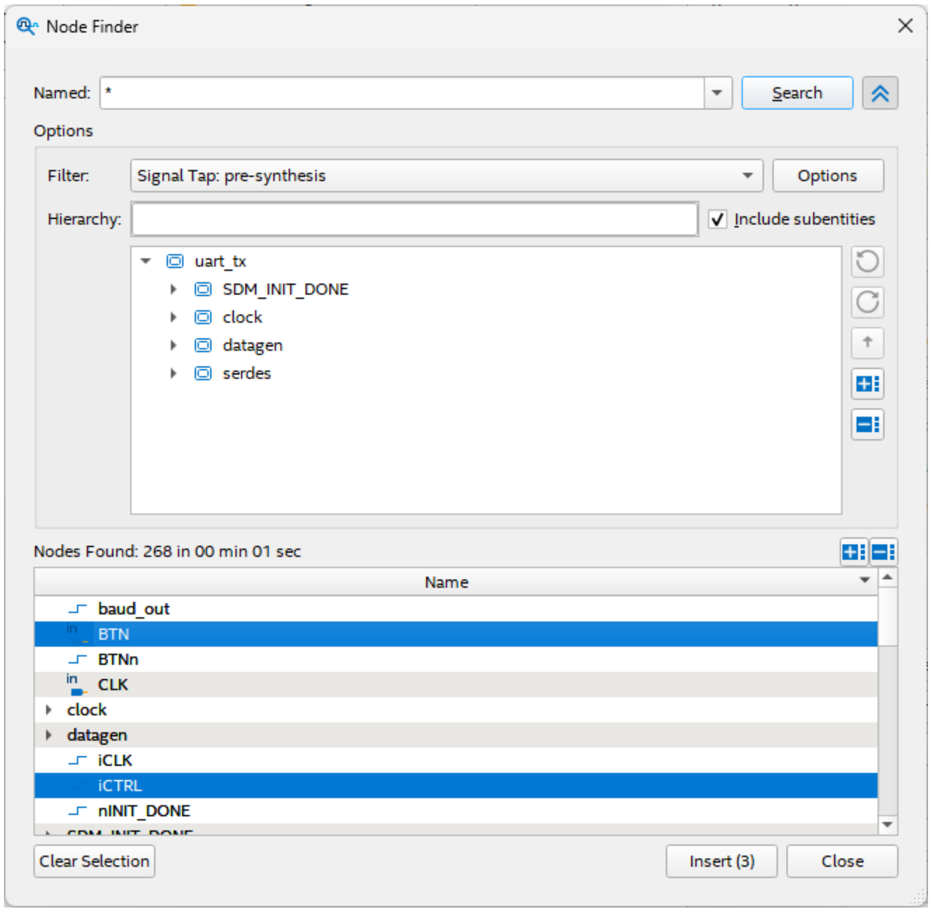
✓ Include subentities

- uart_tx
  - SDM_INIT_DONE
  - clock
  - datagen
  - serdes

Click on **Search**

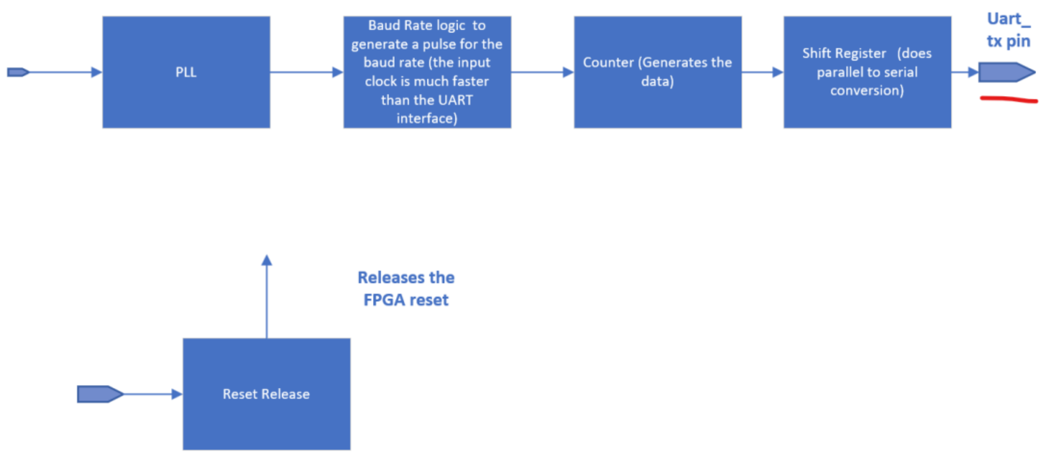


Select BTN, iCTRL, UART_TX and click the insert button.

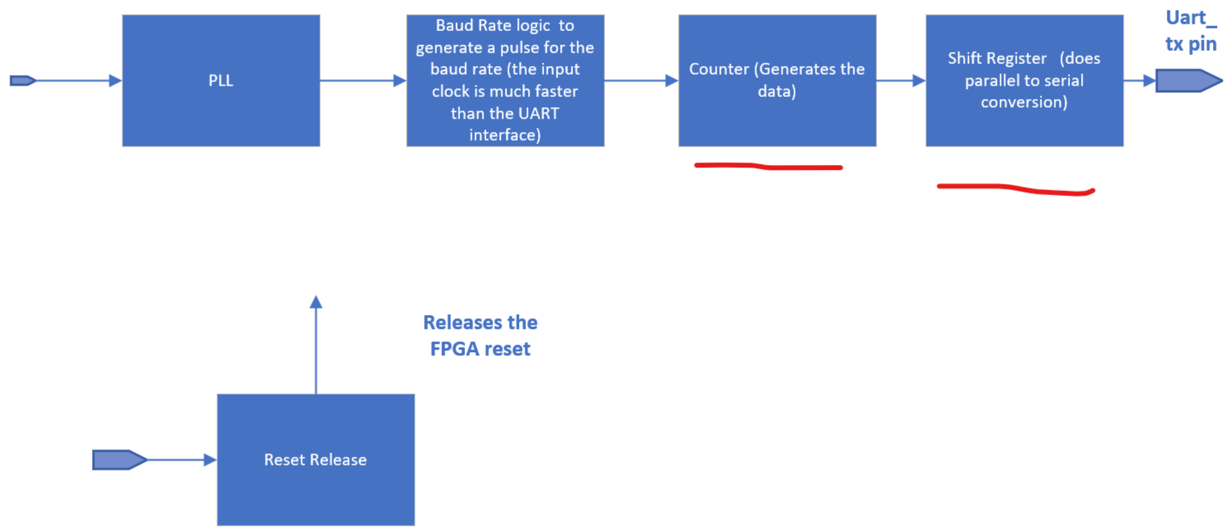


The BTN, which is the button on the board, will start the process of capturing analyzer data. The iCTRL is a control clock signal. The UART_TX is the output pin as seen below.

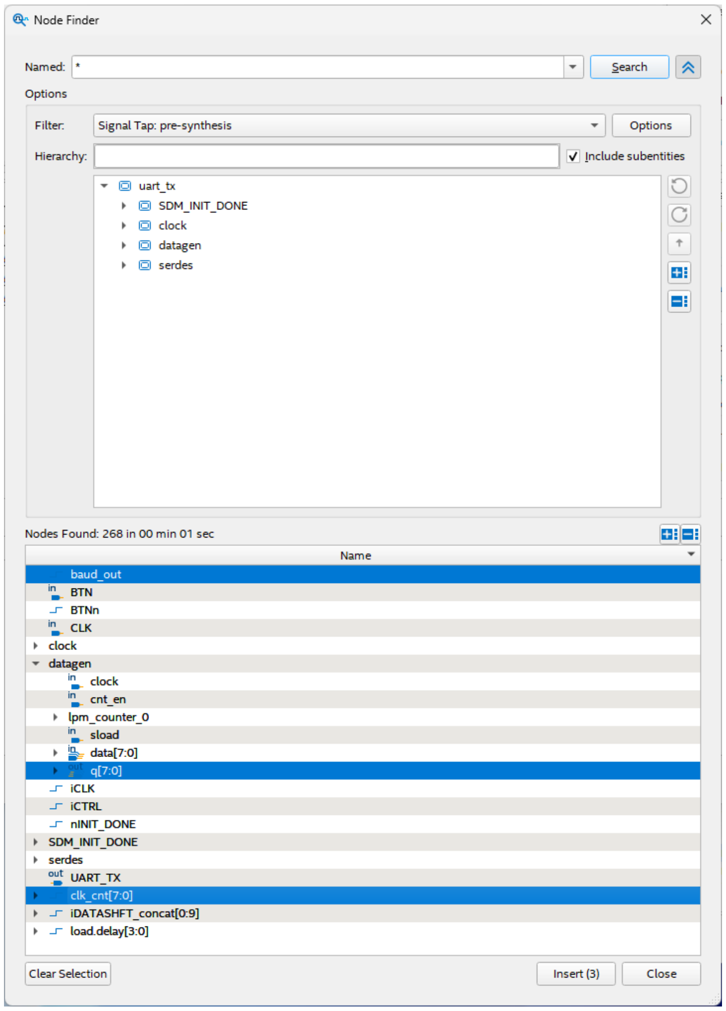
Click **Insert**. Click **OK** in the Clock Suggestion pop up window


PLL
Baud Rate logic to generate a pulse for the baud rate (the input clock is much faster than the UART interface)
Counter (Generates the data)
Shift Register (does parallel to serial conversion)
Uart_ tx pin
Releases the FPGA reset
Reset Release

Internal signals from the counter and shift registers will be added next.


PLL
Baud Rate logic to generate a pulse for the baud rate (the input clock is much faster than the UART interface)
Counter (Generates the data)
Shift Register (does parallel to serial conversion)
Uart_ tx pin
Releases the FPGA reset
Reset Release

Select the **datagen|q (expand the datagen)**, **clk_cnt ,** and **baud_out** nodes . To see the nodes, you will need to expand the options in the mode.



Node Finder

Named: *

Search

Options

Filter: Signal Tap: pre-synthesis

Options

Hierarchy:

✓ Include subentities

- uart_tx
  - SDM_INIT_DONE
  - clock
  - datagen
  - serdes

Nodes Found: 268 in 00 min 01 sec

Name
baud_out
BTN
BTNn
CLK
clock
datagen
clock
cnt_en
lpm_counter_0
sload
data[7:0]
q[7:0]
iCLK
iCTRL
nINIT_DONE
SDM_INIT_DONE
serdes
UART_TX
clk_cnt[7:0]
iDATASHFT_concat[0:9]
load.delay[3:0]

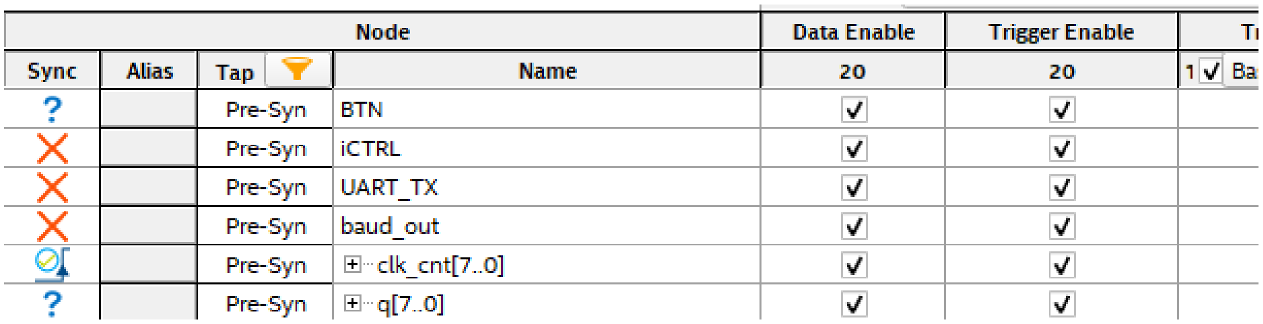
Clear Selection

Insert (3)

Close

Select **Insert** then **Close** to close the Node Finder.

The node finder will look like



Node				Data Enable	Trigger Enable	Tr
Sync	Alias	Tap	Name	20	20	1 ✓ Ba
?		Pre-Syn	BTN	✓	✓	
✗		Pre-Syn	iCTRL	✓	✓	
✗		Pre-Syn	UART_TX	✓	✓	
✗		Pre-Syn	baud_out	✓	✓	
		Pre-Syn	⊞ clk_cnt[7..0]	✓	✓	
?		Pre-Syn	⊞ q[7..0]	✓	✓	

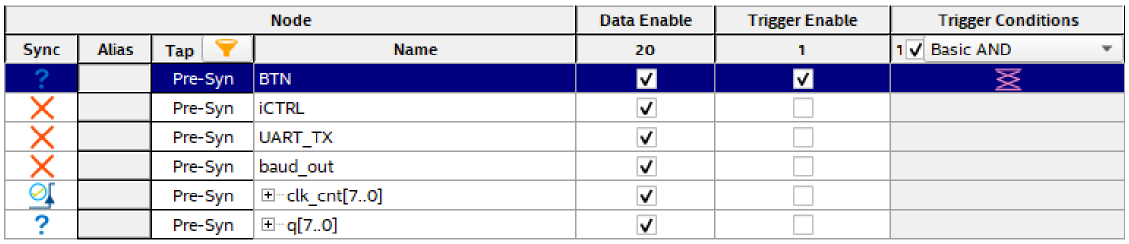
The next step is to set the trigger conditions.

Per the Quartus Prime Pro Edition User Guide: Debug Tools, the trigger is what starts or stops the data capture of the signals. More specifically, the "Signal Tap logic analyzer captures data continuously from the signals you specify while the logic analyzer is running. To capture and store only specific signal data, you can specify conditions that *trigger* the start or stop of data capture. A trigger activates—that is, the logic analyzer stops and displays the data—when the signals you specify reach the trigger conditions that you define.

The Signal Tap logic analyzer allows you to define trigger conditions that range from very simple, such as the rising edge of a single signal, to very complex, involving groups of signals, extra logic, and multiple conditions"

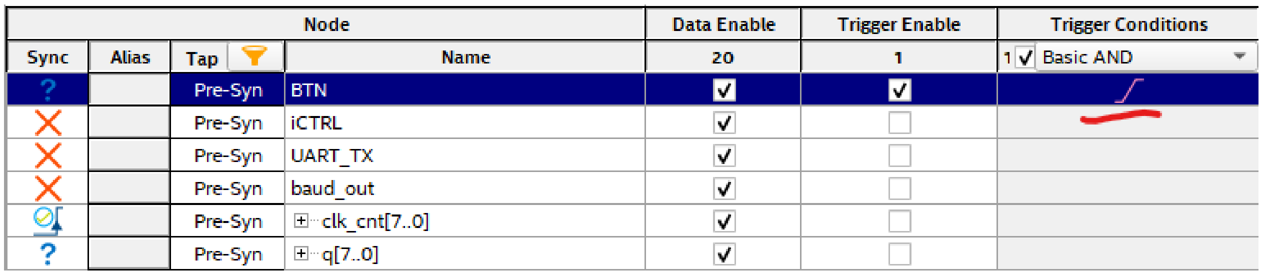
In this case, the trigger condition will be very simple as it will trigger on the rising edge of a button.

Leave the Trigger Enable for the BTN but **remove the check mark for the Trigger Enable for all the other nodes**. This will mean that only the button will act as the trigger.



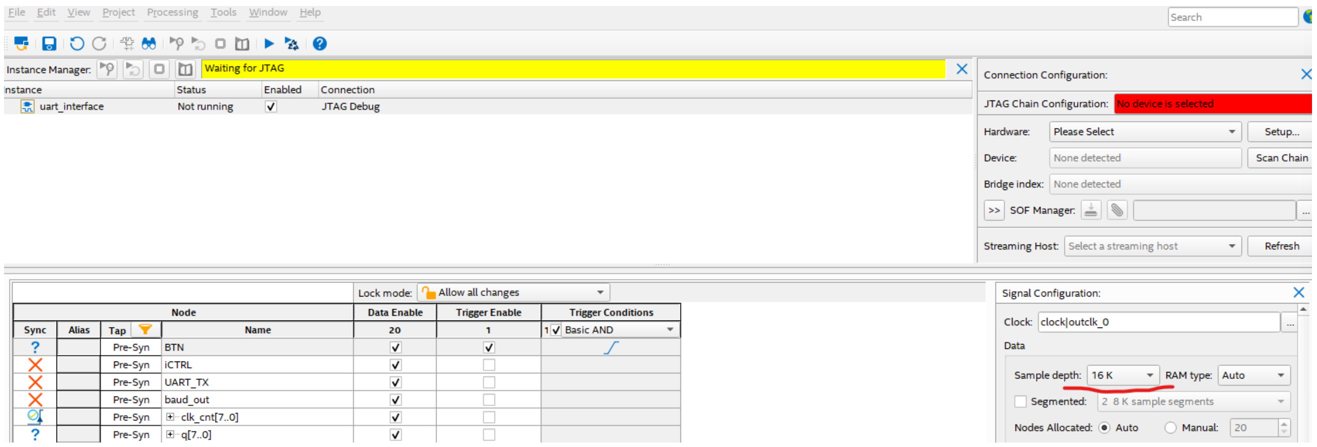
Node				Data Enable	Trigger Enable	Trigger Conditions
Sync	Alias	Tap	Name	20	1	1 ✓ Basic AND
?		Pre-Syn	BTN	✓	✓	
✗		Pre-Syn	iCTRL	✓	☐	
✗		Pre-Syn	UART_TX	✓	☐	
✗		Pre-Syn	baud_out	✓	☐	
		Pre-Syn	⊞ clk_cnt[7..0]	✓	☐	
?		Pre-Syn	⊞ q[7..0]	✓	☐	

Right click in the Trigger Conditions for BTN and select Rising Edge as



Node				Data Enable	Trigger Enable	Trigger Conditions
Sync	Alias	Tap	Name	20	1	1 ✓ Basic AND
?		Pre-Syn	BTN	✓	✓	
✗		Pre-Syn	iCTRL	✓	☐	
✗		Pre-Syn	UART_TX	✓	☐	
✗		Pre-Syn	baud_out	✓	☐	
		Pre-Syn	⊞ clk_cnt[7..0]	✓	☐	
?		Pre-Syn	⊞ q[7..0]	✓	☐	

The Signal Tap Logic Analyzer is now set up.



Instance Manager: Waiting for JTAG

Instance	Status	Enabled	Connection
uart_interface	Not running	✓	JTAG Debug

Connection Configuration:
JTAG Chain Configuration: No device is selected
Hardware: Please Select
Device: None detected
Bridge index: None detected
SOF Manager:
Streaming Host: Select a streaming host

Lock mode: Allow all changes

Sync	Alias	Tap	Name	Data Enable	Trigger Enable	Trigger Conditions
				20	1	1 ✓ Basic AND
?		Pre-Syn	BTN	✓	✓	
✗		Pre-Syn	iCTRL	✓		
✗		Pre-Syn	UART_TX	✓		
✗		Pre-Syn	baud_out	✓		
		Pre-Syn	clk_cnt[7..0]	✓		
?		Pre-Syn	q[7..0]	✓		

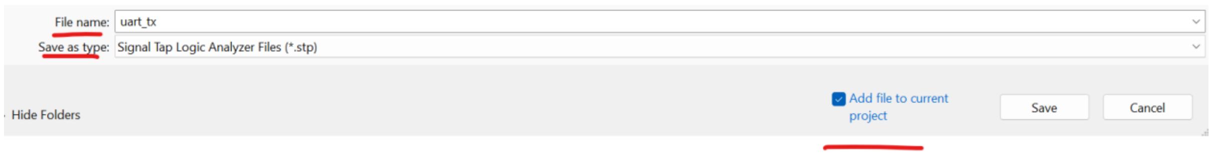
Signal Configuration:
Clock: clock|outclk_0
Data
Sample depth: 16 K RAM type: Auto
Segmented: 2 8 K sample segments
Nodes Allocated: (●) Auto ( ) Manual: 20

Save the Analyzer Settings in the Signal Tap Logic Analyzer window by clicking button or **File** → **Save** and enter the following information:

File name: **uart_tx**

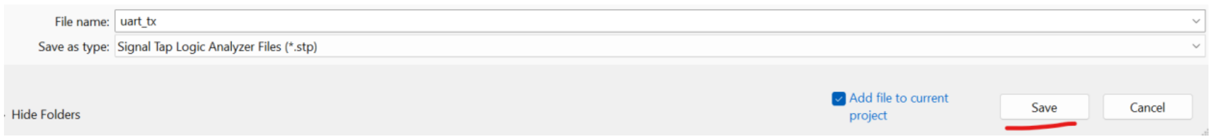
Save as type: Signal Tap Logic Analyzer Files (*.stp)

Make sure that "**Add file to current project**" option is checked (so that the file gets added to the project).



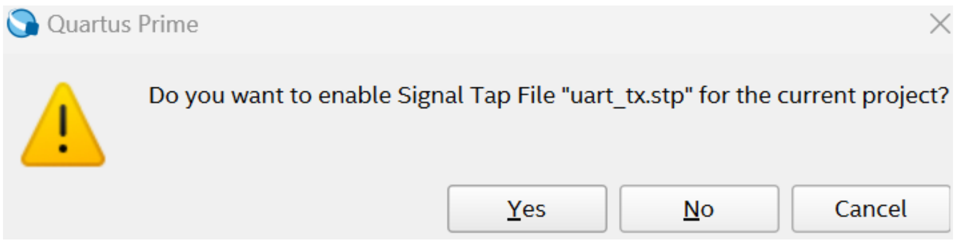
File name: uart_tx
Save as type: Signal Tap Logic Analyzer Files (*.stp)
☑ Add file to current project

Select **Save**.



File name: uart_tx
Save as type: Signal Tap Logic Analyzer Files (*.stp)
☑ Add file to current project

Select **Yes** to the pop-up window of



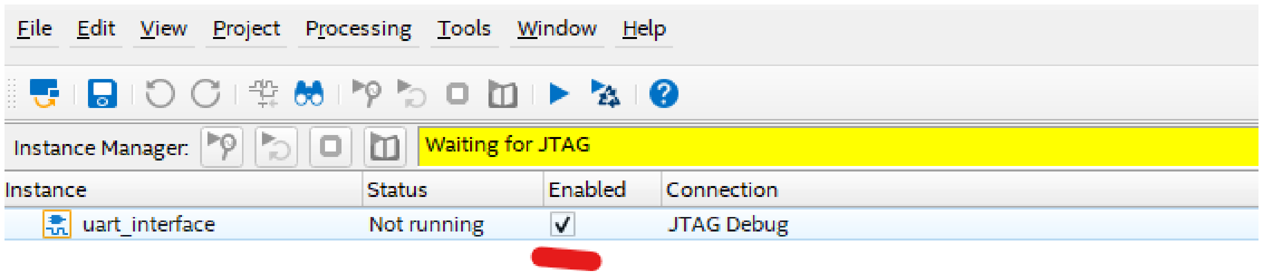
Quartus Prime

Do you want to enable Signal Tap File "uart_tx.stp" for the current project?

4.3.9 Compiling the Design

Compilation is where the software takes the database that it created during the earlier step (Analysis and Synthesis) along with the pinouts and the timing constraints and then the software places and routes (connect) the design into the device. During the Assembler stage of the compilation, a programming file is generated, which can be programmed into the FPGA.

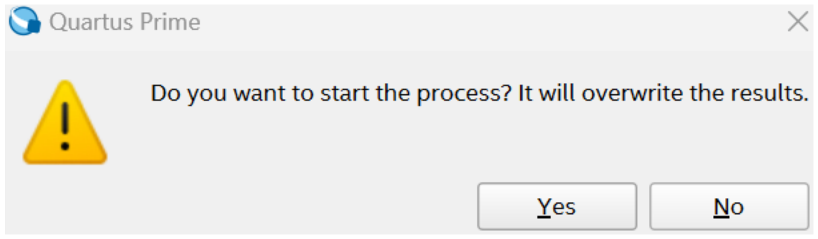
Before compiling, ensure that the **Signal Tap is enabled** as



Instance	Status	Enabled	Connection
uart_interface	Not running	✓	JTAG Debug

Start Compilation by clicking on ▶ button on the toolbars, or **Processing** → **Start Compilation**. Within the Signal Tap window, this compile process is also available to do this step. You can select either one.

A window will be seen that states



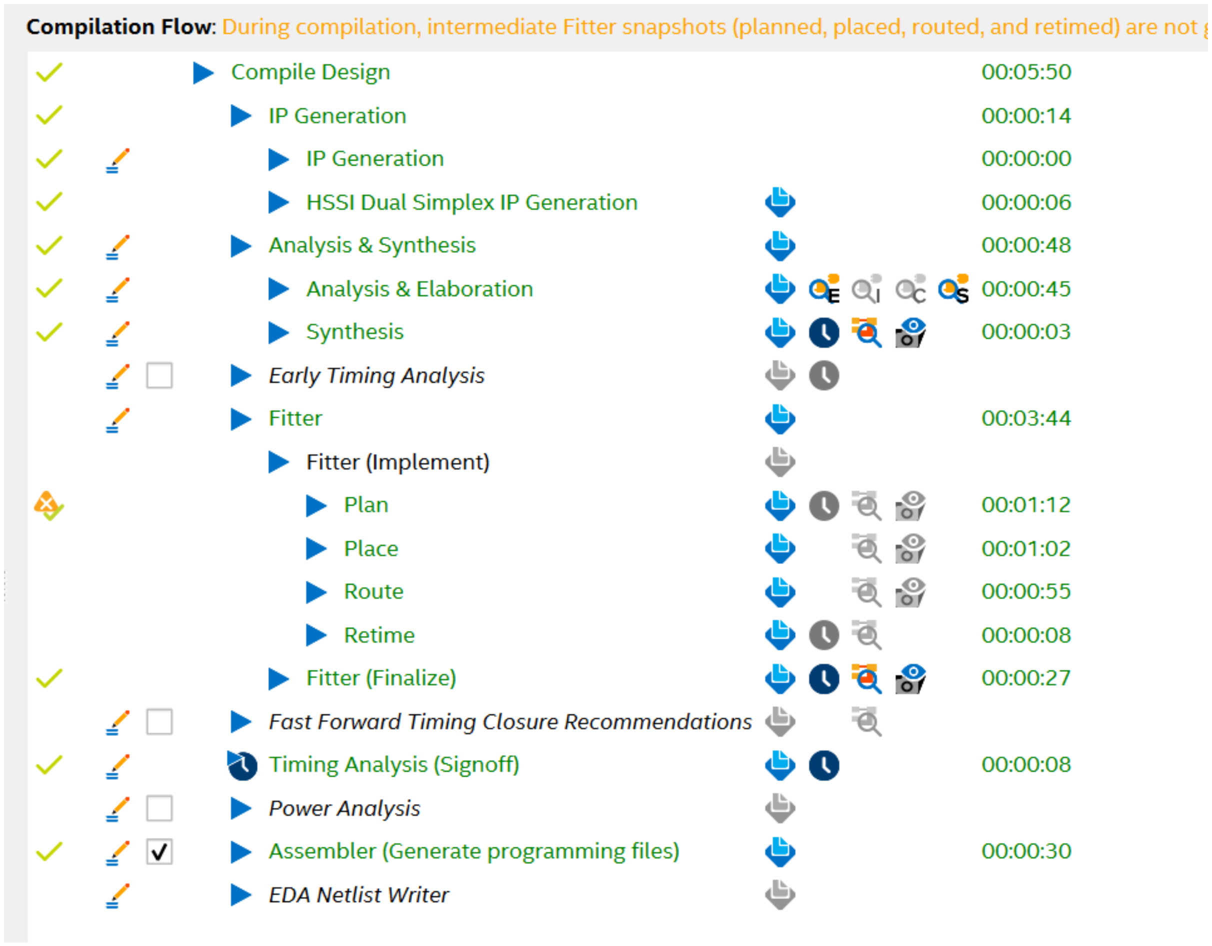
Do you want to start the process? It will overwrite the results.

Select **Yes**

There should be no errors during compilation. If there are errors, they should be fixed before re-compiling. The 100% in the lower right corner or a green checkmark next to the Compile Design in the Compilation task window indicates that the compilation was successful.

As Quartus compiles the design, there will be a progress bar that shows the steps that it does and the checkmarks as the steps are complete. Notice the checkmarks as steps are completed. The time used for each step depends on your computer.

There will be a Timing Analyzer Window that appears. **You can close the Timing Analyzer Window** with **File -> Close.** Our focus for this workshop is on the compilation flow rather than timing. The time for the compile can vary depending on the computer used



Compilation Flow: During compilation, intermediate Fitter snapshots (planned, placed, routed, and retimed) are not

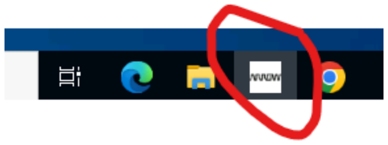
✓		Compile Design	00:05:50
✓		IP Generation	00:00:14
✓		IP Generation	00:00:00
✓		HSSI Dual Simplex IP Generation	00:00:06
✓		Analysis & Synthesis	00:00:48
✓		Analysis & Elaboration	00:00:45
✓		Synthesis	00:00:03
	☐	*Early Timing Analysis*	
		Fitter	00:03:44
		Fitter (Implement)	
		Plan	00:01:12
		Place	00:01:02
		Route	00:00:55
		Retime	00:00:08
✓		Fitter (Finalize)	00:00:27
	☐	*Fast Forward Timing Closure Recommendations*	
✓		Timing Analysis (Signoff)	00:00:08
	☐	*Power Analysis*	
✓	☑	Assembler (Generate programming files)	00:00:30
		EDA Netlist Writer	

5 Validate the design Locally

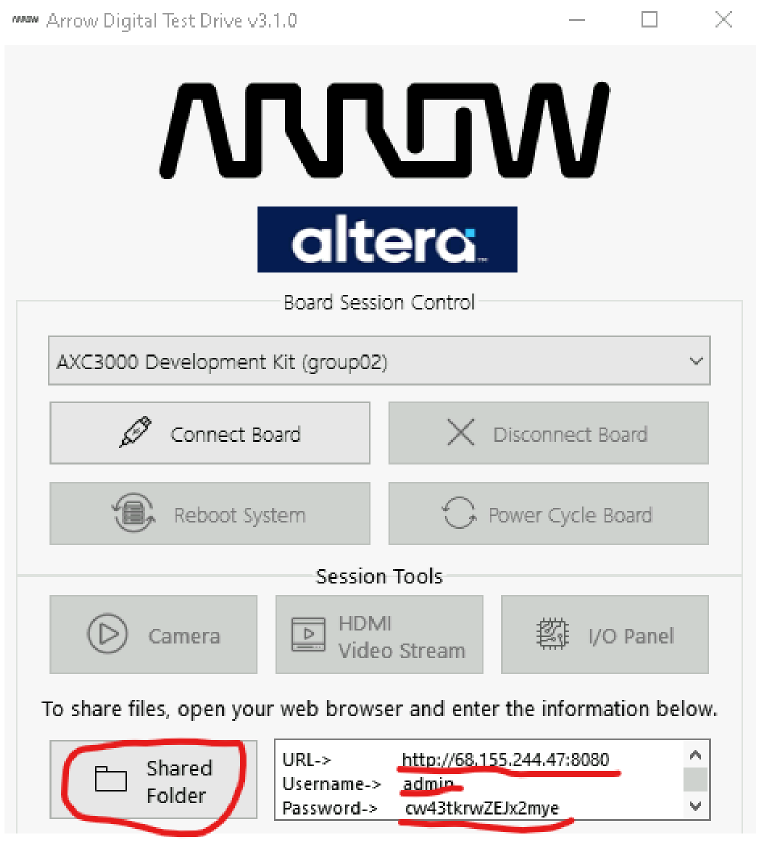
The next step is to see if the design works. If you have built the lab using CloudLabs then complete section 5.1. If you have built the lab on a local PC then skip to section 5.2.

5.1 File Download

5.1.1 Click on the ArrowDTD icon in CloudLabs to launch the tool.



5.1.2 Open the **Shared Folder** utility in the CloudLab VM. Open a browser in your local PC. Navigate to the **URL** indicated. Use the provided username and password.

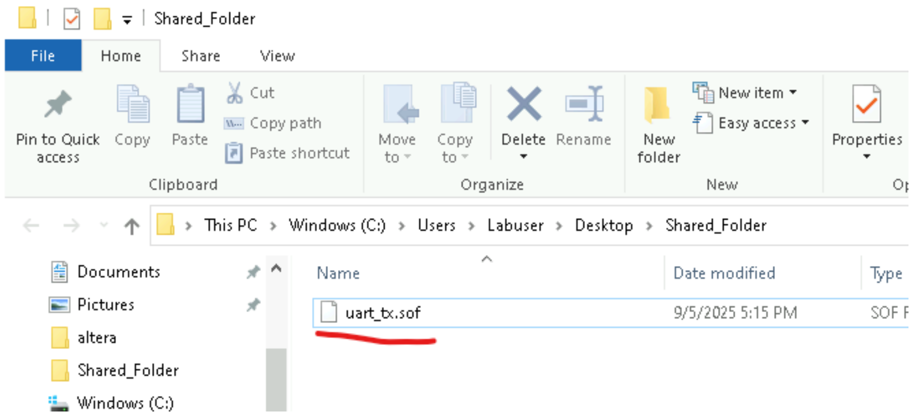


To share files, open your web browser and enter the information below.

URL-> http://68.155.244.47:8080
Username-> admin
Password-> cw43tkrwZEJx2mye

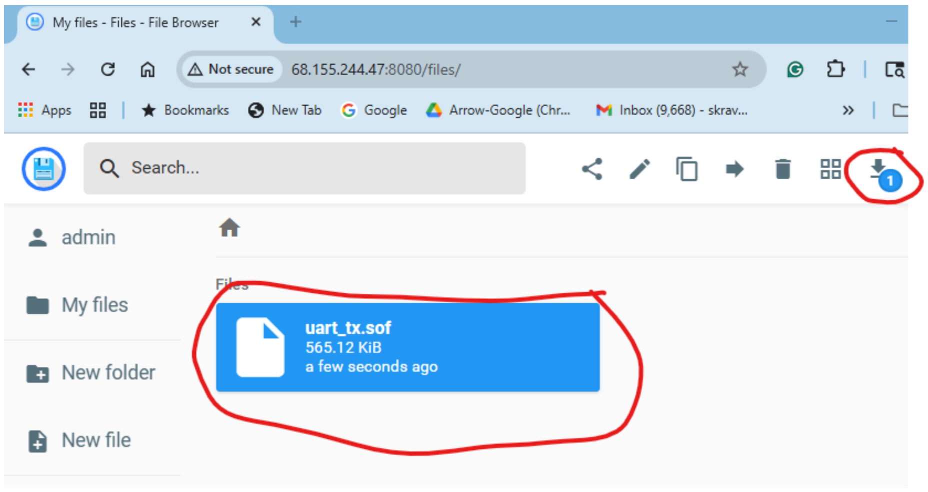
5.1.3 **Drag** and drop the following file into the **Shared** folder.

c:\altera_workshops\axc3000\Sign_Lab\output_files\uart_tx.sof



Name	Date modified	Type
uart_tx.sof	9/5/2025 5:15 PM	SOF F

5.1.4 This will be reflected in the **browser** on the local PC. **Download** the SOF file to a convenient location.



uart_tx.sof
565.12 KiB
a few seconds ago

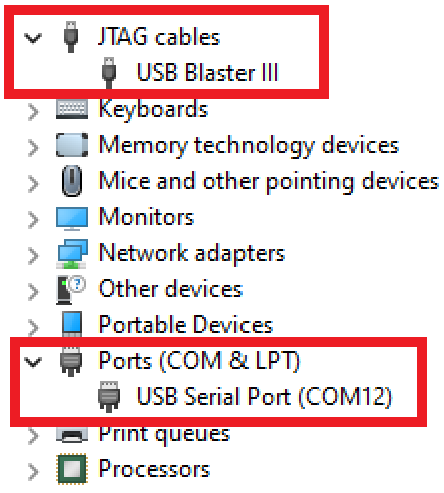
5.2 Program the FPGA

The rest of the instructions will be done on the local computer.

The next step is to program the design into the FPGA on the board. During the Assembler stage of the compilation, a programming file was generated.

5.2.1 Connect your AXC3000 Board to your PC using a USB C cable.

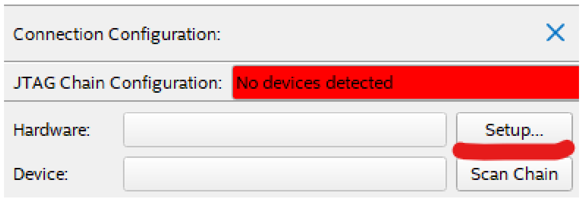
Since the USB Blaster III should be already installed, the Window's Device Manager should display the following entries, highlighted in red (COM port number may differ depending on your PC):



JTAG cables
USB Blaster III
Keyboards
Memory technology devices
Mice and other pointing devices
Monitors
Network adapters
Other devices
Portable Devices
Ports (COM & LPT)
USB Serial Port (COM12)
Print queues
Processors

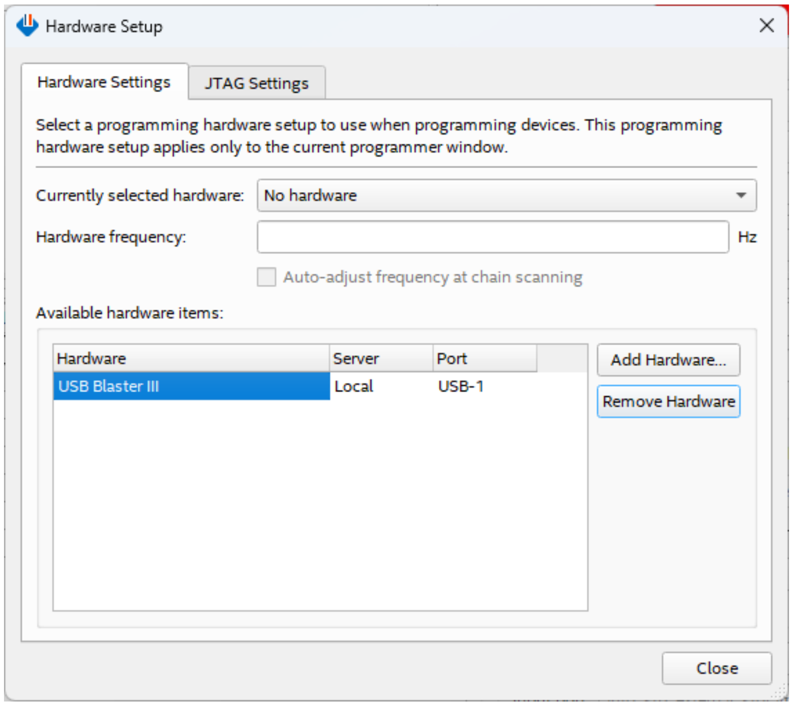
5.2.2 Open the Signal Tap window. From the Start menu open **Altera 25.1.1.125 Pro Edition --> SignalTap.**

5.2.3 Select the **Setup button**



Connection Configuration:
JTAG Chain Configuration: No devices detected
Hardware:
Device:

Click **Hardware Setup...** and double click **USB Blaster** III entry in the Hardware Setup window. **Currently selected hardware** should now show USB Blaster III [USB-1] (the USB port number may be different depending on your PC).


Hardware Setup

Hardware Settings | JTAG Settings

Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: No hardware

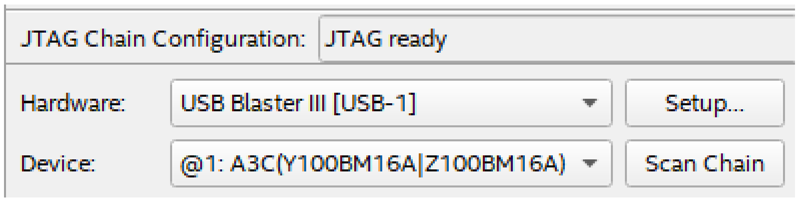
Hardware frequency: Hz

Auto-adjust frequency at chain scanning

Available hardware items:

Hardware	Server	Port
USB Blaster III	Local	USB-1

5.2.4 Click **Close**


JTAG Chain Configuration: JTAG ready

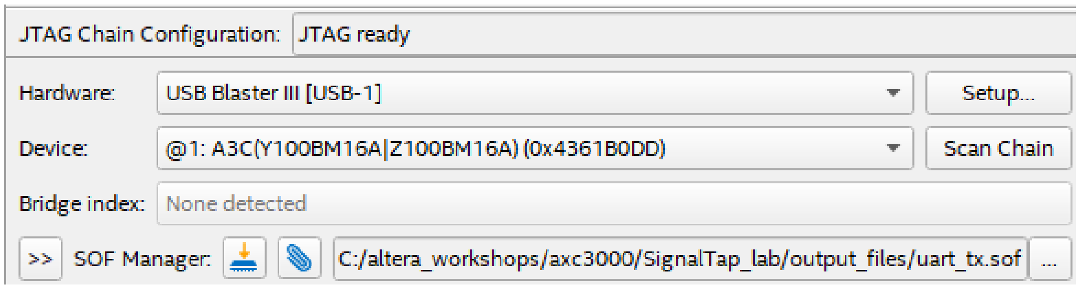
Hardware: USB Blaster III [USB-1]

Device: @1: A3C(Y100BM16A|Z100BM16A)

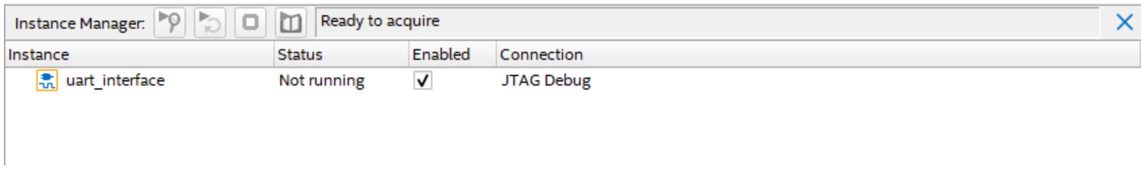
5.2.5 **Click on** ... button to choose the programming file

5.2.6 If the file was downloaded from Cloudlabs then navigate to the **folder where the sof file was downloaded** and select the **.sof** file. If the design was built locally then navigate to **project_directory>/output_files/** and select the uart_tx.sof file

5.2.7 Select **Open**


JTAG Chain Configuration: JTAG ready
Hardware: USB Blaster III [USB-1]
Device: @1: A3C(Y100BM16A|Z100BM16A) (0x4361B0DD)
Bridge index: None detected
SOF Manager: C:/altera_workshops/axc3000/SignalTap_lab/output_files/uart_tx.sof

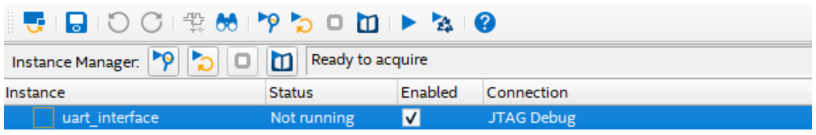
Click on the button to program the board. When the configuration is complete, the message box in the middle should write "Ready to acquire"


Instance Manager: Ready to acquire

Instance	Status	Enabled	Connection
uart_interface	Not running	✓	JTAG Debug

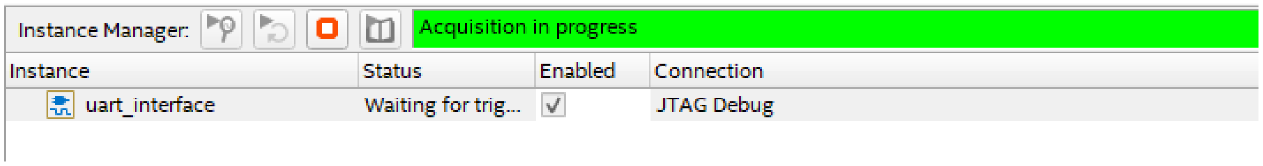
5.3 Run the design

Select the uart_interface and then **click on** the button to run analysis. The status of uart_interface instance should change to "Waiting for trigger".


Instance Manager: Ready to acquire

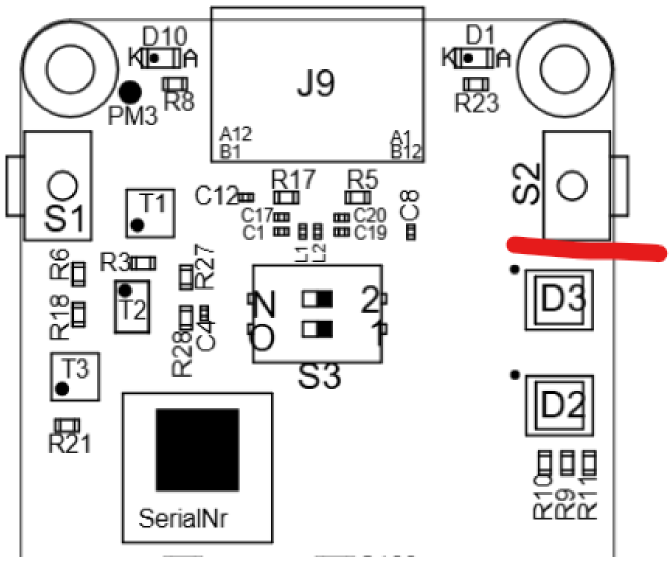
Instance	Status	Enabled	Connection
uart_interface	Not running	✓	JTAG Debug

This analyzer type is single acquisition, it runs only once after triggering.

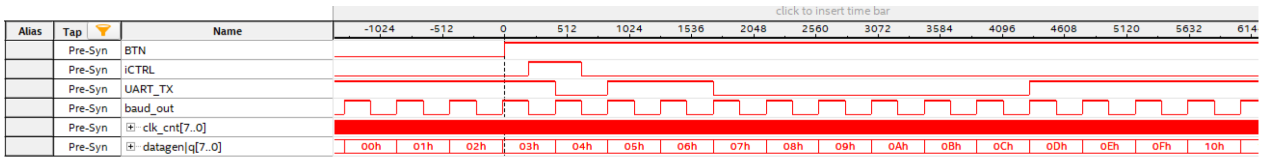

Instance Manager: Acquisition in progress

Instance	Status	Enabled	Connection
uart_interface	Waiting for trig...	✓	JTAG Debug

5.3.1 Press the **S2** button on the AXC3000 board to generate a trigger signal for the analyzer. The trigger is the rising edge of the BTN, so the analyses will start when you release the button.

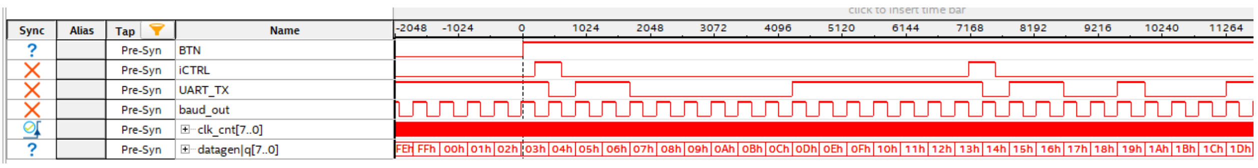


5.3.2 You can zoom into the waveform by pressing the left mouse button



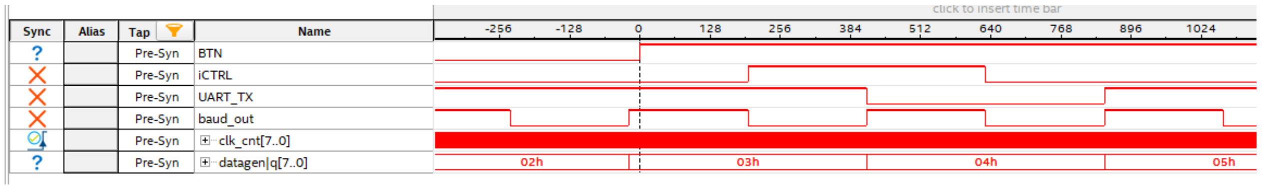
Alias	Tap	Name
	Pre-Syn	BTN
	Pre-Syn	iCTRL
	Pre-Syn	UART_TX
	Pre-Syn	baud_out
	Pre-Syn	⊞ clk_cnt[7..0]
	Pre-Syn	⊞ datagen|q[7..0]

5.3.3 You can zoom out of the waveform by selecting the right mouse button.

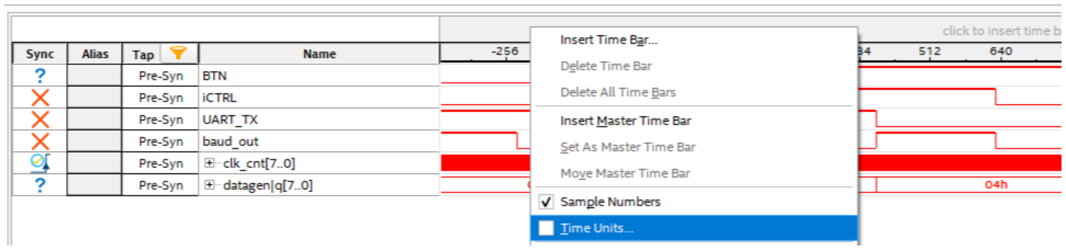

click to insert time bar
-2048
-1024
0
1024
2048
3072
4096
5120
6144
7168
8192
9216
10240
11264
FEh FFh 00h 01h 02h 03h 04h 05h 06h 07h 08h 09h 0Ah 0Bh 0Ch 0Dh 0Eh 0Fh 10h 11h 12h 13h 14h 15h 16h 17h 18h 19h 1Ah 1Bh 1Ch 1Dh

Sync	Alias	Tap	Name
?		Pre-Syn	BTN
✗		Pre-Syn	iCTRL
✗		Pre-Syn	UART_TX
✗		Pre-Syn	baud_out
		Pre-Syn	⊞ clk_cnt[7..0]
?		Pre-Syn	⊞ datagen|q[7..0]

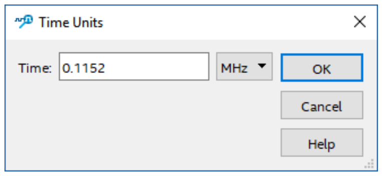
On the data screen you can see that the data is enabled on the rising edge of baud_out and it sends the data to the serial UART_TX output. The transmission signal contains a start bit, 8 bits of data, starting with LSB, and one stop bit.



5.3.4 On the timescale the default setup is the sample numbers, where each number represents a clock period. You can change it by right clicking on the timescale and select **Time Units...** from the pop-up menu.



5.3.5 In the Time Units window set the time to **0.1152 MHz** and press **OK**. After this process you can check the waveforms with horizontal µs scale.



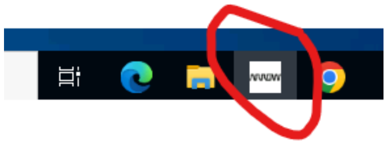
CONGRATULATIONS! YOU HAVE SUCCESSFULLY COMPLETED THE SIGNALTAP LAB!

6 Validate the Design Remotely

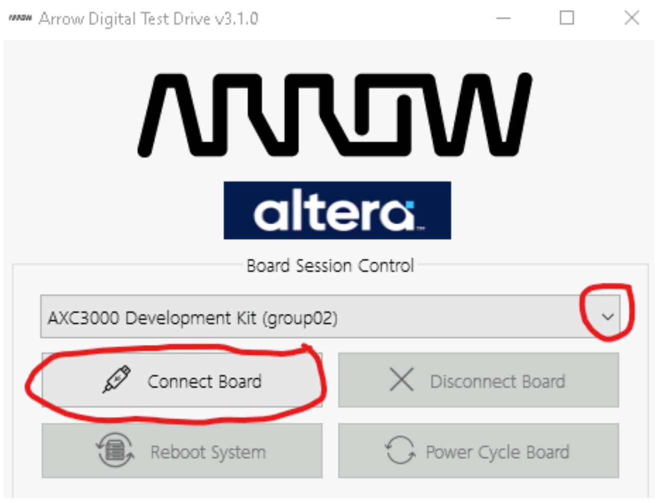
The following instructions require the user to be utilizing a CloudLabs Windows Virtual Machine.

6.1 Connect to the Board Farm

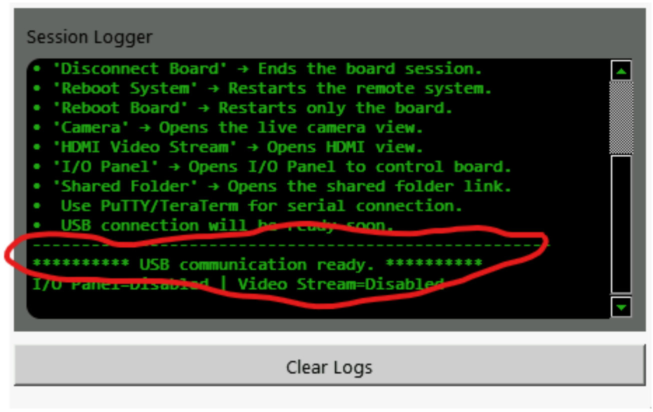
6.1.1 Connect to the remote AXC3000 Board using the ArrowDTD tool. Click on the ArrowDTD icon to launch the tool.



6.1.2 Select an available board from the dropdown menu and click connect board.

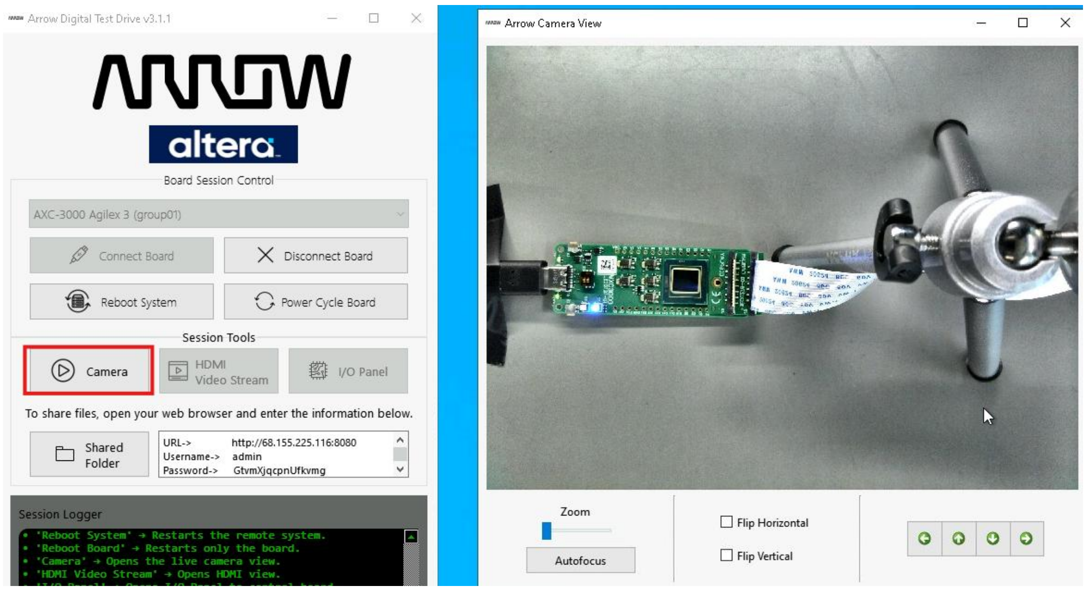


Wait until the board is ready.



```
• 'Disconnect Board' → Ends the board session.
• 'Reboot System' → Restarts the remote system.
• 'Reboot Board' → Restarts only the board.
• 'Camera' → Opens the live camera view.
• 'HDMI Video Stream' → Opens HDMI view.
• 'I/O Panel' → Opens I/O Panel to control board.
• 'Shared Folder' → Opens the shared folder link.
•  Use PuTTY/TeraTerm for serial connection.
•  USB connection will be ready soon.
---------------------------------------------------------
********** USB communication ready. **********
I/O Panel=Disabled | Video Stream=Disabled
```

6.1.3 Open the Camera View of the board.



AXC-3000 Agilex 3 (group01)

To share files, open your web browser and enter the information below.

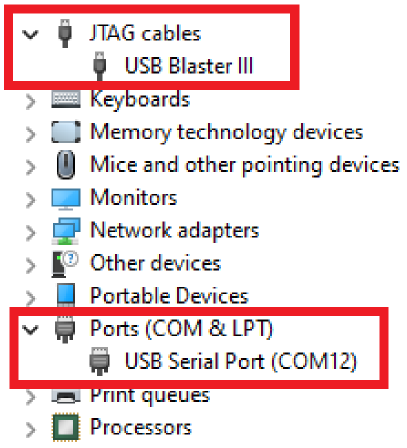
```
URL->        http://68.155.225.116:8080
Username->   admin
Password->   GtvmXjqcpnUfkvmg
```

```
• 'Reboot System' → Restarts the remote system.
• 'Reboot Board' → Restarts only the board.
• 'Camera' → Opens the live camera view.
• 'HDMI Video Stream' → Opens HDMI view.
```

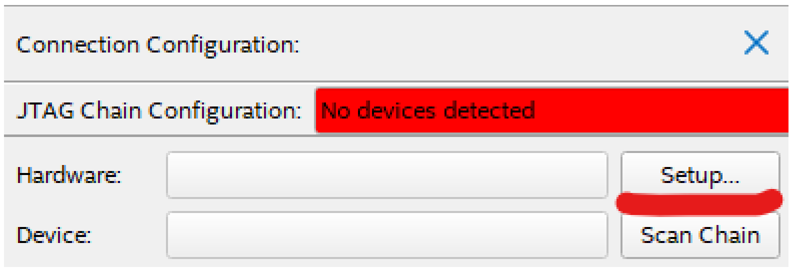
6.2 Program the FPGA

6.2.1 The next step is to program the design into the FPGA on the board. During the Assembler stage of the compilation, a programming file was generated.

6.2.2 **Connect your AXC3000 Board to your PC using a USB C cable**. (For remote workshops with a board connected, this will be done differently). Since the USB Blaster III should be already installed, the Window's Device Manager should display the following entries, highlighted in red (COM port number may differ depending on your PC):

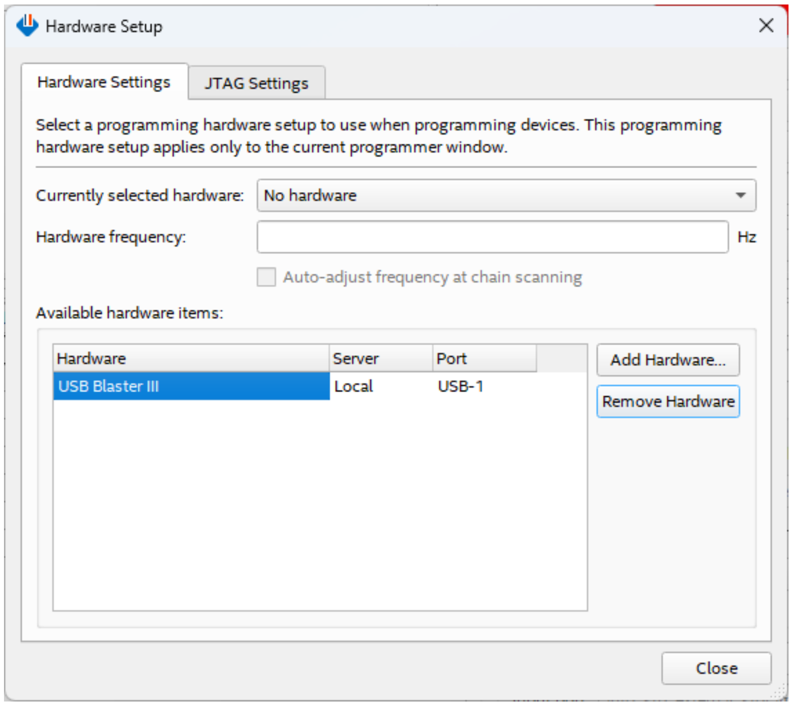


6.2.3 Open the Signal Tap window and select the **Setup button**


Connection Configuration:

JTAG Chain Configuration: No devices detected

6.2.4 Click **Hardware Setup...** and double click **USB Blaster** III entry in the Hardware Setup window. **Currently selected hardware** should now show USB Blaster III [USB-1] (the USB port number may be different depending on your PC).



Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: No hardware

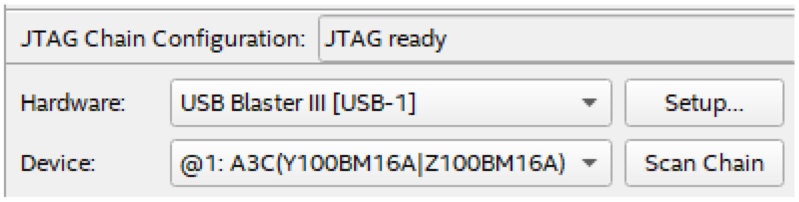
Hardware frequency: Hz

Auto-adjust frequency at chain scanning

Available hardware items:

Hardware	Server	Port
USB Blaster III	Local	USB-1

6.2.5 Click **Close**



JTAG Chain Configuration: JTAG ready

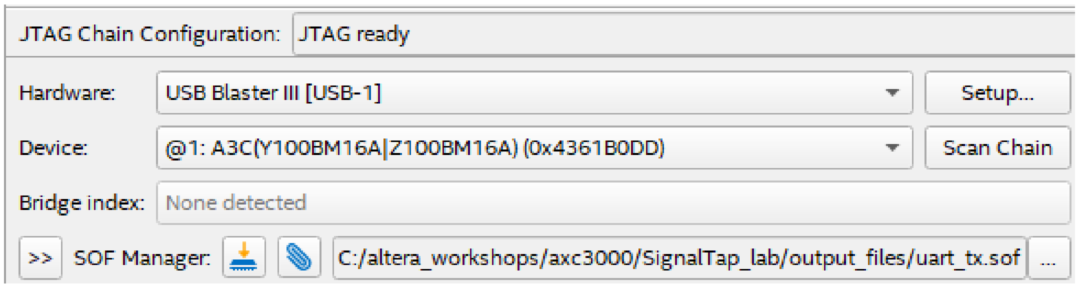
Hardware: USB Blaster III [USB-1]

Device: @1: A3C(Y100BM16A|Z100BM16A)

6.2.6 **Click on** ... button to choose the programming file

6.2.7 Navigate to **project_directory>/output_files/** and select the uart_tx.sof file

6.2.8 Select **Open**



JTAG Chain Configuration: JTAG ready

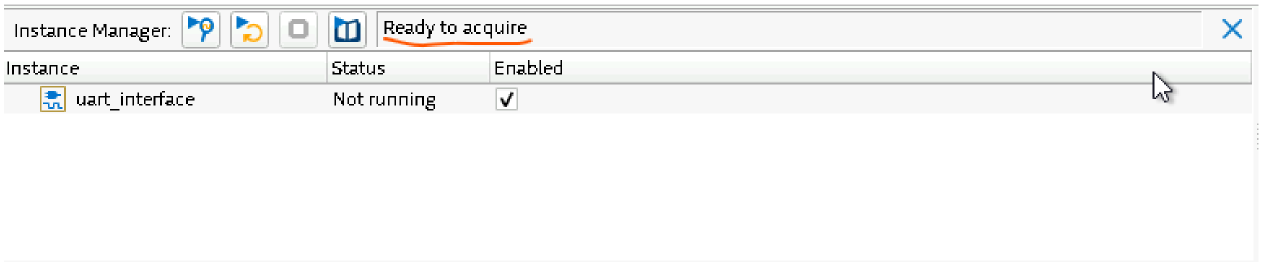
Hardware: USB Blaster III [USB-1]

Device: @1: A3C(Y100BM16A|Z100BM16A) (0x4361B0DD)

Bridge index: None detected

SOF Manager: C:/altera_workshops/axc3000/SignalTap_lab/output_files/uart_tx.sof

6.2.9 **Click on the** button to program the board. When the configuration is complete, the message box in the middle should write “Ready to acquire”



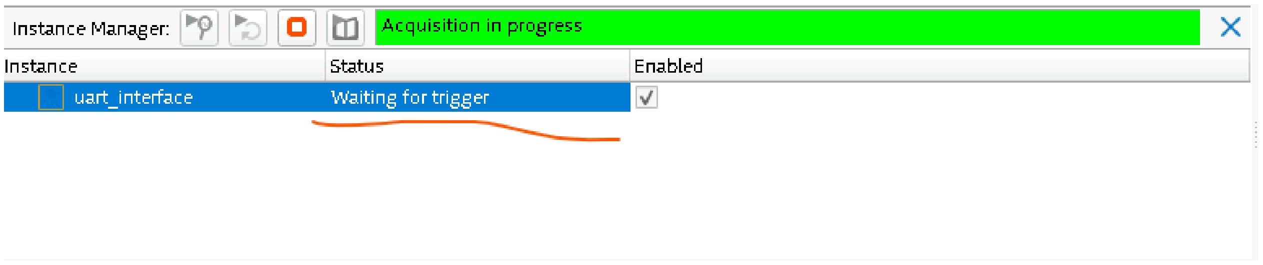
Instance Manager: Ready to acquire

Instance	Status	Enabled
uart_interface	Not running	✓

6.3 Run the design

6.3.1 Select the uart_interface and then **click on** the button to run analysis. The status of uart_interface instance should change to “Waiting for trigger”.

This analyzer type is a single acquisition, it runs only once after triggering.



Instance Manager: Acquisition in progress

Instance	Status	Enabled
uart_interface	Waiting for trigger	✓

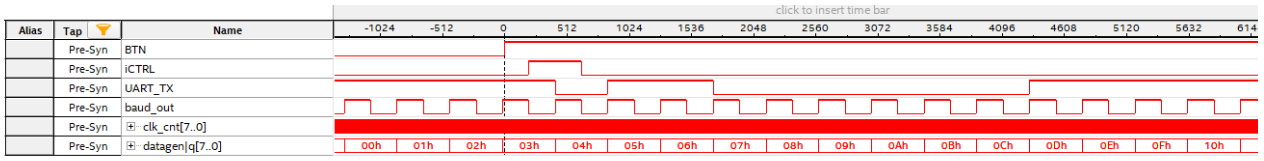
The AXC3000 board is remote. This lab has been modified to allow the BTN signal to be remotely asserted. This is done using the Source and Probes functionality. The JTAG interface enables specified signals to be asserted remotely. The design has been modified to remote the connection to the external **BTN**. This has been replaced by the output of the jtag_source output signal.

6.3.2 Open the **In-System Source and Probes Editor**. Tools --> In-System Sources and Probes Editor. Select the **USB Blaster III** in the Hardware dropdown menu.

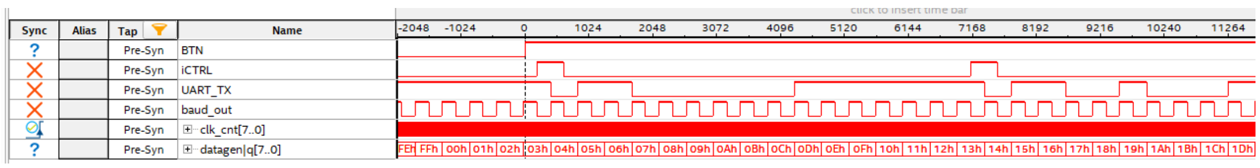
6.3.3 Press the **BTN** signal in the In-System Sources and Probes Editor to toggle the value and generate a trigger signal for the analyzer. The trigger is the rising edge of the BTN.

6.3.4 Wait up to 10 seconds for a trigger due to the distant remote location of the board.

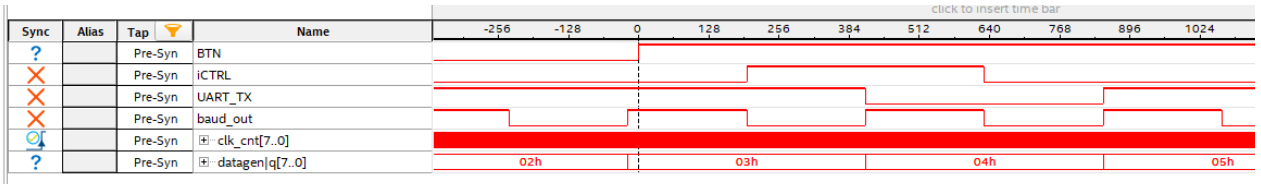
6.3.5 You can zoom into the waveform by pressing the left mouse button



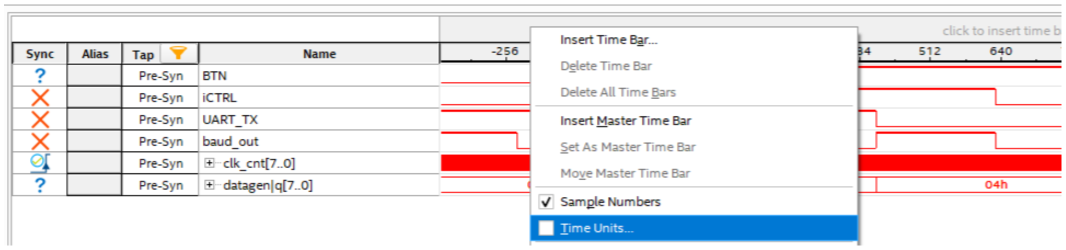
6.3.6 You can zoom out of the waveform by selecting the right mouse button.



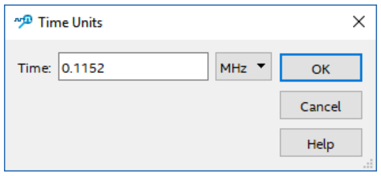
On the data screen you can see that the data is enabled on the rising edge of baud_out and it sends the data to the serial UART_TX output. The transmission signal contains a start bit, 8 bits of data, starting with LSB, and one stop bit.



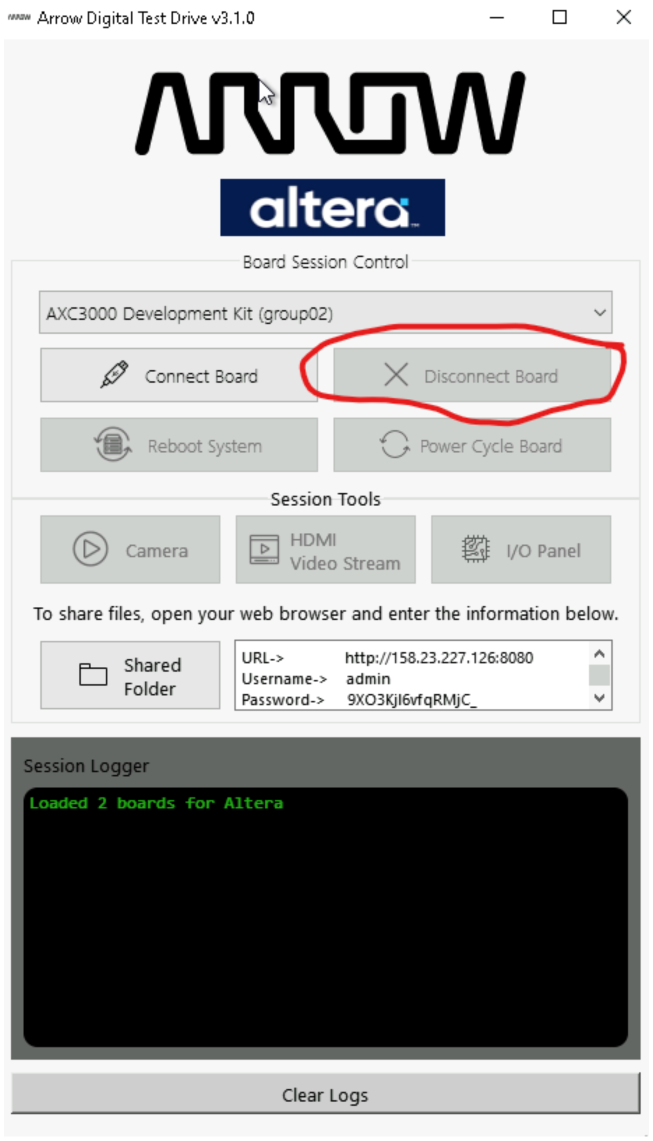
6.3.7 On the timescale the default setup is the sample numbers, where each number represents a clock period. You can change it by right clicking on the timescale and select **Time Units…** from the pop-up menu.



6.3.8 In the Time Units window set the time to **0.1152 MHz** and press **OK**. After this process you can check the waveforms with horizontal μs scale.



6.3.9 Please disconnect the board when debugging is complete. This will free up the board resource for others to use.



To share files, open your web browser and enter the information below.

URL-> http://158.23.227.126:8080
Username-> admin
Password-> 9XO3KjI6vfqRMjC_

```
Loaded 2 boards for Altera
```

CONGRATULATIONS! YOU HAVE SUCCESSFULLY COMPLETED THE SIGNALTAP LAB!

7 Legal Disclaimer

applications. Arrow expressly disclaims any responsibility for such usage which shall be made at Your sole risk.

WARRANTY

Arrow warrants that it has the right to provide the evaluation board to you. This warranty is provided by Arrow in lieu of all other warranties, written or oral, statutory, express or implied, including any warranty as to merchantability, non-infringement, fitness for any particular purpose, or uninterrupted or error-free operation, all of which are expressly disclaimed. The evaluation board is provided “as is” without any other rights or warranties, directly or indirectly.

You warrant to Arrow that the evaluation board is used only by electronics experts who understand the dangers of handling and using such items, you assume all responsibility and liability for any improper or unsafe handling or use of the evaluation board by you, your employees, affiliates, contractors, and designees.

LIMITATION OF LIABILITIES

In no event shall Arrow be liable to you, whether in contract, tort (including negligence), strict liability, or any other legal theory, for any direct, indirect, special, consequential, incidental, punitive, or exemplary damages with respect to any matters relating to this agreement. In no event shall arrow’s liability arising out of this agreement in the aggregate exceed the amount paid by you under this agreement for the purchase of the evaluation board.

IDENTIFICATION

You shall, at Your expense, defend Arrow and its Affiliates and Licensors against a claim or action brought by a third party for infringement or misappropriation of any patent, copyright, trade secret or other intellectual property right of a third party to the extent resulting from (1) Your combination of the Evaluation Board with any other component, system, software, or firmware, (2) Your modification of the Evaluation Board, or (3) Your use of the Evaluation Board in a manner not permitted under this Agreement. You shall indemnify Arrow and its Affiliates and Licensors against and pay any resulting costs and damages finally awarded against Arrow and its Affiliates and Licensors or agreed to in any settlement, provided that You have sole control of the defense and settlement of the claim or action, and Arrow cooperates in the defense and furnishes all related evidence under its control at Your expense. Arrow will be entitled to participate in the defense of such claim or action and to employ counsel at its own expense.

RECYCLING

The Evaluation Board is not to be disposed as an urban waste. At the end of its life cycle, differentiated waste collection must be followed, as stated in the directive 2002/96/EC. In all the countries belonging to the European Union (EU Dir. 2002/96/EC) and those following differentiated recycling, the Evaluation Board is subject to differentiated recycling at the end of its life cycle, therefore: It is forbidden to dispose the Evaluation Board as an undifferentiated waste or with other domestic wastes. Consult the local authorities for more information on the proper disposal channels. An incorrect Evaluation Board disposal may cause damage to the environment and is punishable by the law.