Simple HLS Design for Arria 10 SoC

This example design demonstrates a simple HLS sort component in a minimal system as described in the video series *HLS Walkthrough*:

- Part 1
- Part 2
- Part 3

This design uses a JTAG to Avalon MM Agent IP to expose an HLS component to the JTAG control interface. This lets the user control and observe the behaviour of the HLS sort component using the System Console application.

This design implementation is slightly different than the video in that it uses Platform Designer to generate the interconnect, and I avoid the use of the avmm_regmap.v file. This results in a simpler design, and RAM savings. You can view the register map in the `sort_bus_csr.h` file that HLS generates.

Board-specific Considerations

I have modified the design specifications slightly compared with the video series to work with the Arria 10 SoC Development Kit board. The board specific configurations are:

1. Choose `10AS066N3F40E2SG` device to match the devkit when compiling my HLS code
2. Choose pin `AP20 - CLKUSR` to drive the `i_clk` signal
3. Use `jtag.sdc` from the Arria 10 SoC Golden Hardware Reference Design (GHRD) source code.

Building the Design

Follow these steps to compile and test the design:

1. Compile HLS code to RTL using a Windows or Linux machine by following the build directions in the `hls` folder.

2. Copy the HLS-generated IP to the `ip` directory in the quartus project:

   Linux:

   ```
   cp -r hls/tutorial-fpga.prj/components/sort_bus ip
   ```

   Windows:

   ```
   xcopy hls\tutorial-fpga.prj\components\sort_bus ip\sort_bus /e /s /i
   ```

3. Open the `SimpleArria10Project.qpf` using Intel© Quartus Prime Pro.

4. Open Platform Designer, and select `system.qsys`. Once the design opens, observe the IPs, then click 'Generate HDL'. When the HDL generation completes, close Platform Designer.

5. Compile the full design by clicking the 'Start Compilation' button.

6. Once the compilation is complete and an SOF has been generated, you can run it on a board.

   NOTE The script for loading the SOF to the board and starting System Console is designed for Windows, but you should be able to use the same commands on a Linux system.

   1. Attach the A10 SoC devkit to your Windows PC using a micro USB cable

   2. navigate to the `system_console` directory, and run the script `test.bat`.

   3. In the System Console terminal that appears, run the following commands to exercise the HLS component:

      ```
      % source jtag_avmm.tcl
      % source load_vals.tcl
      % source read_inputs.tcl
      % source read_outputs.tcl
      ```

      You should observe that the output from the `read_outputs.tcl` script will be the same data as the output from `read_inputs.tcl`, but it will be sorted in increasing order.

Building the Design from Scratch

You can build this project yourself from scratch using Quartus IPs. These instructions are for targeting the Intel Arria 10 SoC development kit using Intel Quartus Prime Pro Edition v21.1, but they should generalize to other FPGAs and other versions of Quartus.

1. Choose a board to target and make note of the FPGA device on the board.

2. Compile HLS code to RTL using a Windows or Linux machine by following the build directions in the `hls` folder. Make sure you set the device property `FPGA_DEVICE` to match the device you chose in Step 1.

3. Open Quartus Pro and create a new empty project using the 'New Project' wizard.

   1. Set the `top-level entity` to be `sort`.

   2. Make sure you match the device you chose in step 1.

   3. Once the project is created create a folder called `ip`, and copy the IP generated by HLS to it.

      Linux:

      ```
      cp -r hls/tutorial-fpga.prj/components/sort_bus ip
      ```

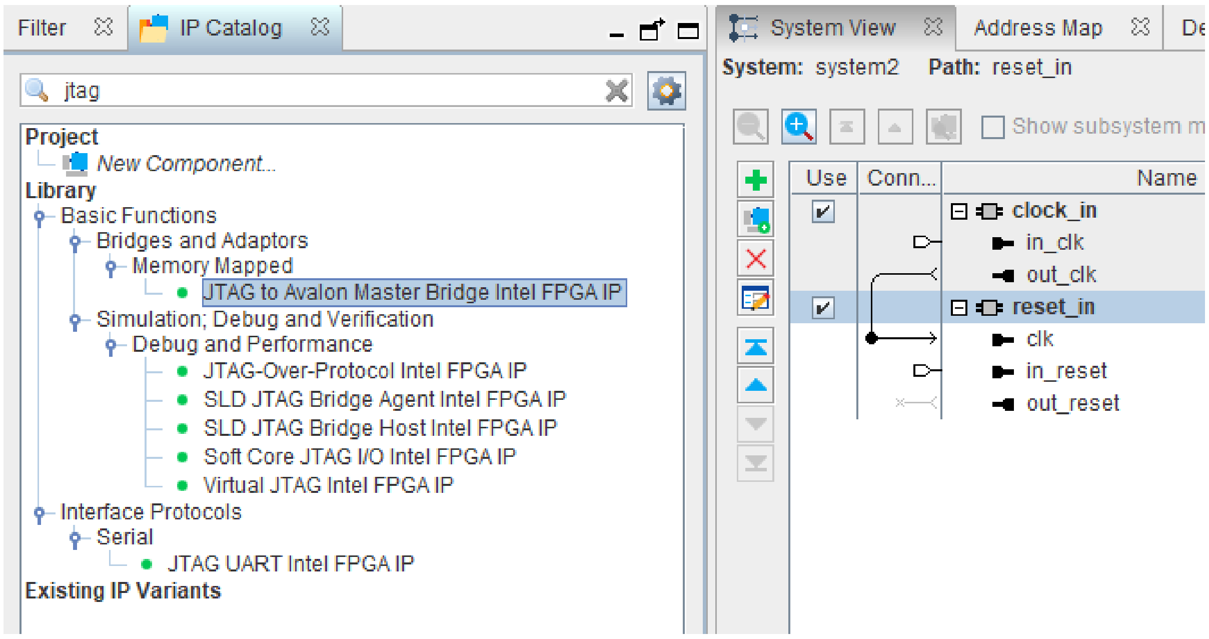
Windows:

```
xcopy hls\tutorial-fpga.prj\components\sort_bus ip\sort_bus /e /s /i
```

If you do not do this, you will need to either add the `hls` directory to your project's search path, or use a `.ipx` file so Platform Designer can see the IP. See Adding Third-Party IP Components in the Platform Designer User Guide for more details.

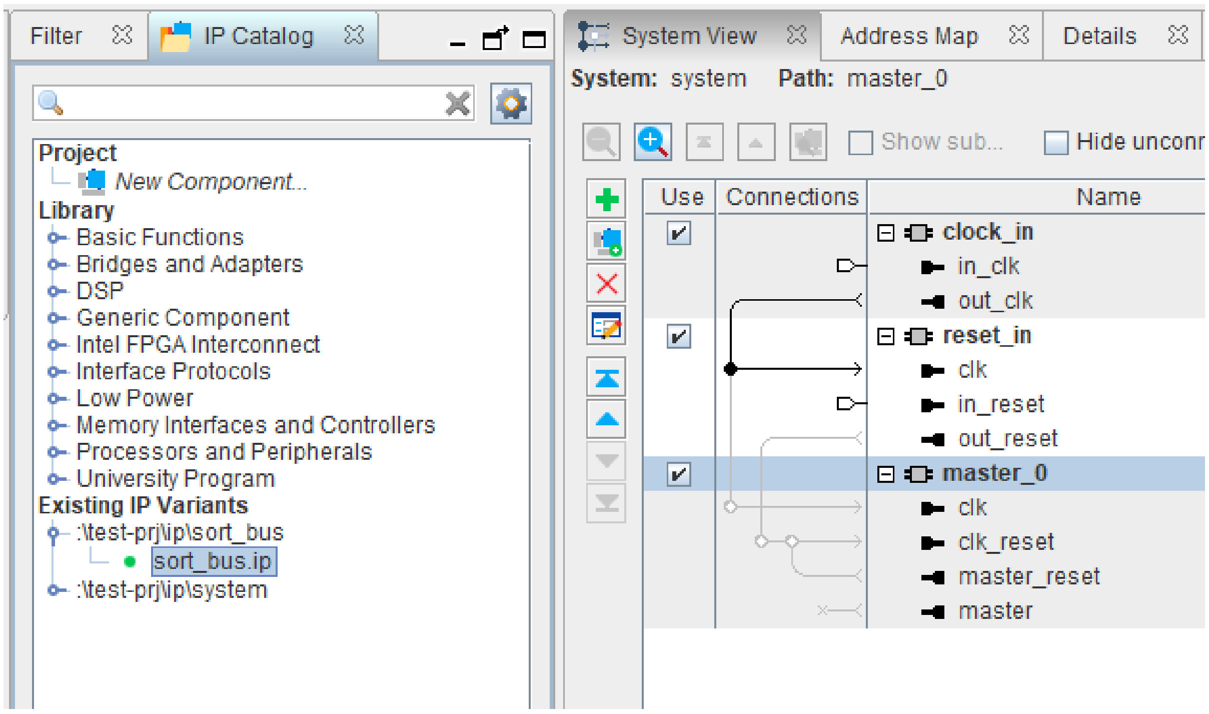
4. Open Platform Designer, and create a new system and give it a unique name, e.g. `system.qsys`. Once the design opens, you can add the necessary IPs to it, then generate HDL for your Quartus project.

    1. In the IP catalog pane, search for the `JTAG to Avalon Host Bridge IP`:



    Double-click the IP and then in the configuration window that appears, click 'Finish'.

    2. Add the IP generated by HLS. If you added it to the `ip` folder as described in Step 3 above, the IP should appear under `Existing IP Variants` in the IP Catalog (make sure you clear the search box):



System: system Path: master_0

Project
- New Component...

Library
- Basic Functions
- Bridges and Adapters
- DSP
- Generic Component
- Intel FPGA Interconnect
- Interface Protocols
- Low Power
- Memory Interfaces and Controllers
- Processors and Peripherals
- University Program

Existing IP Variants
- :\test-prj\ip\sort_bus
  - sort_bus.ip
- :\test-prj\ip\system

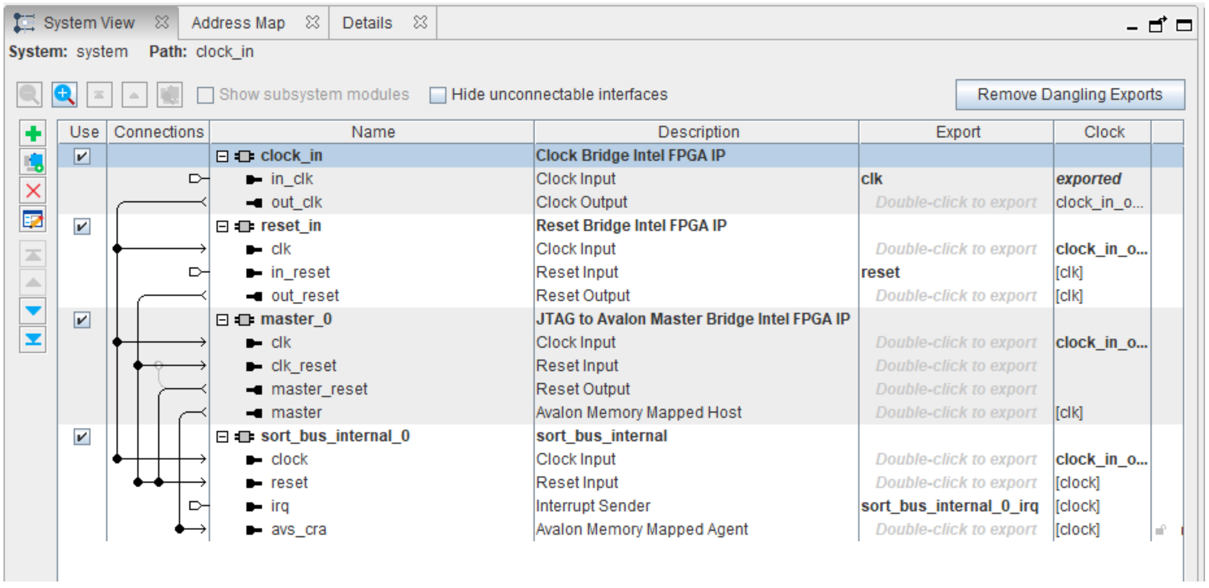
Use	Connections	Name
✓		clock_in
		in_clk
		out_clk
✓		reset_in
		clk
		in_reset
		out_reset
✓		master_0
		clk
		clk_reset
		master_reset
		master

As before, double-click the IP and click 'Finish' to add the IP to your system.

Note: This IP paramterization was generated by HLS, so if you remove the IP from your Platform Designer system, you will need to re-copy the IP from the `hls` folder, or regenerated it.

3. Connect the IPs:

   - all `clock` inputs should connect to the `out_clk` from the `clock_in` IP
   - all `reset` inputs should connect to the `out_reset` from the `reset_in` IP
   - connect the `avs_cra` interface on the `sort_bus_internal` IP to the `master` interface on the `JTAG to Avalon Master Bridge` IP.
   - export the `irq` signal on the `sort_bus_internal` IP.



System: system Path: clock_in

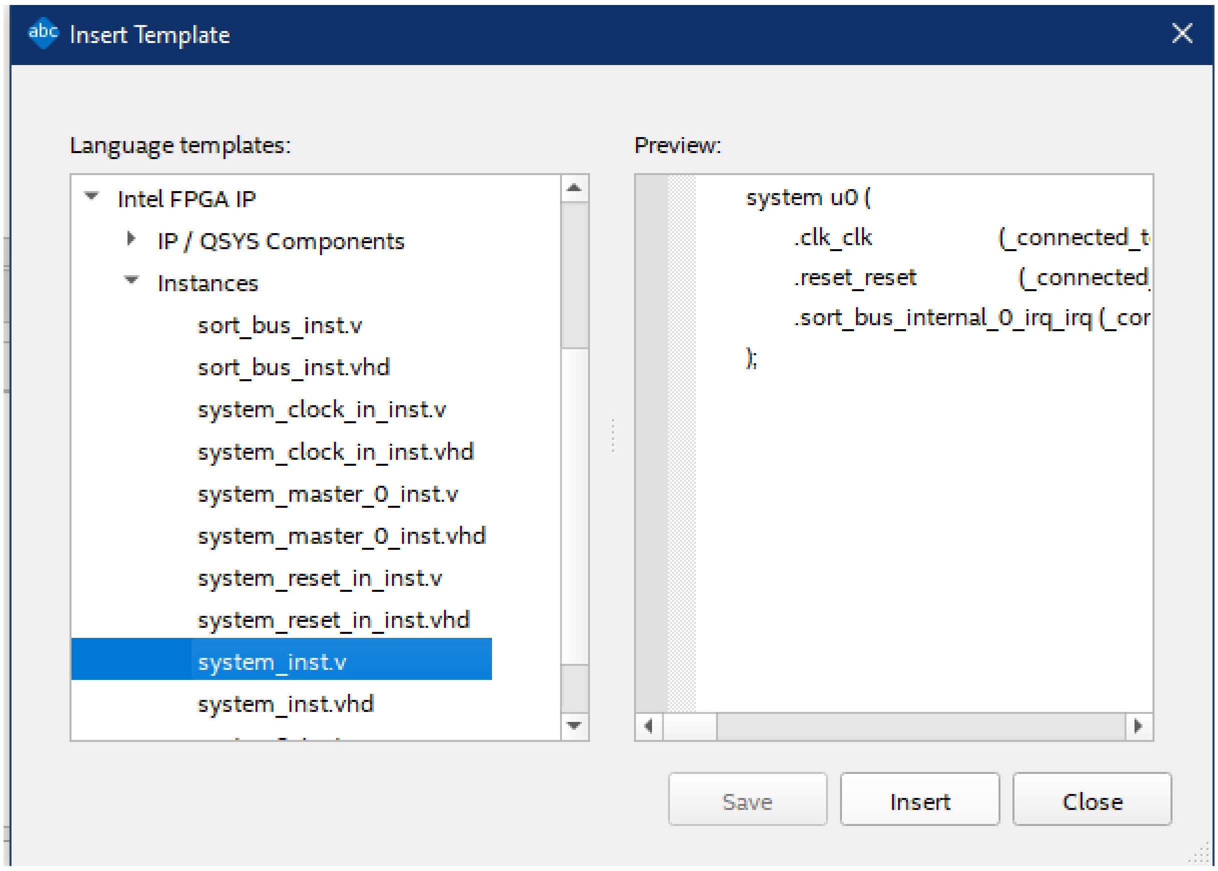
Use	Connections	Name	Description	Export	Clock
✓		clock_in	Clock Bridge Intel FPGA IP		
		in_clk	Clock Input	clk	*exported*
		out_clk	Clock Output	Double-click to export	clock_in_o...
✓		reset_in	Reset Bridge Intel FPGA IP		
		clk	Clock Input	Double-click to export	clock_in_o...
		in_reset	Reset Input	reset	[clk]
		out_reset	Reset Output	Double-click to export	[clk]
✓		master_0	JTAG to Avalon Master Bridge Intel FPGA IP		
		clk	Clock Input	Double-click to export	clock_in_o...
		clk_reset	Reset Input	Double-click to export	
		master_reset	Reset Output	Double-click to export	
		master	Avalon Memory Mapped Host	Double-click to export	[clk]
✓		sort_bus_internal_0	sort_bus_internal		
		clock	Clock Input	Double-click to export	clock_in_o...
		reset	Reset Input	Double-click to export	[clock]
		irq	Interrupt Sender	sort_bus_internal_0_irq	[clock]
		avs_cra	Avalon Memory Mapped Agent	Double-click to export	[clock]

4. Save the system, then generate HDL by clicking 'Generate HDL' in the bottom-left corner. When the HDL generation completes, close Platform Designer.

5. Back in Quartus, create a new file called `sort.v`. This module will wrap your Platform Designer system, and interfase it with the `reset` push-button and an LED on the board. The JTAG to Avalon Agent IP will handle the connection between your design and the JTAG pins on your board automatically.

```
module sort (
 input wire i_clk,
 input wire reset_button_n,
 output reg fpga_led
);

   // pipeline the `reset` signal so the fitter can move logic around on the
   // chip. Also synchronize `reset` signal to clock. Debouncing logic would
be
   // good, but it is omitted for simplicity.
   reg reset_button_d1;
   reg reset_button_d2;

   always @ (posedge i_clk)
   begin
      reset_button_d1 <= ~reset_button_n;
      reset_button_d2 <= reset_button_d1;
   end

   // register the signal used by the LED
   wire sort_done;
   always @(posedge i_clk)
   begin
      fpga_led <= sort_done;
   end

   assign reset = reset_button_d2;

   // instantiate Platform Designer System here

endmodule
```

You can let Quartus generate the Verilog code of the system you created by right-clicking and choosing 'Insert Template'. Then, in the 'Insert Template' window, under `Intel FPGA IP > Instances`, choose `system_inst.v`.

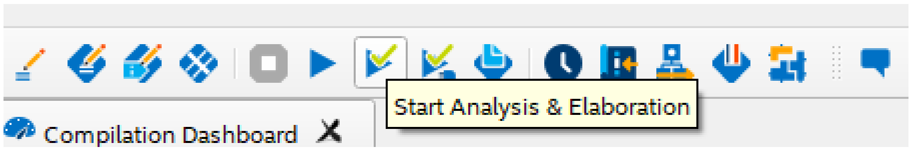


```
system u0 (
        .clk_clk              (_connected_t
        .reset_reset           (_connected
        .sort_bus_internal_0_irq_irq (_cor
);
```

You can populate the input signals like this:

```
system u0 (
   .clk_clk                     (i_clk),     //   input,  width = 1,
clk.clk
   .reset_reset                 (reset),     //   input,  width = 1,
reset.reset
   .sort_bus_internal_0_irq_irq (sort_done) //  output,  width = 1,
sort_bus_internal_0_irq.irq
);
```

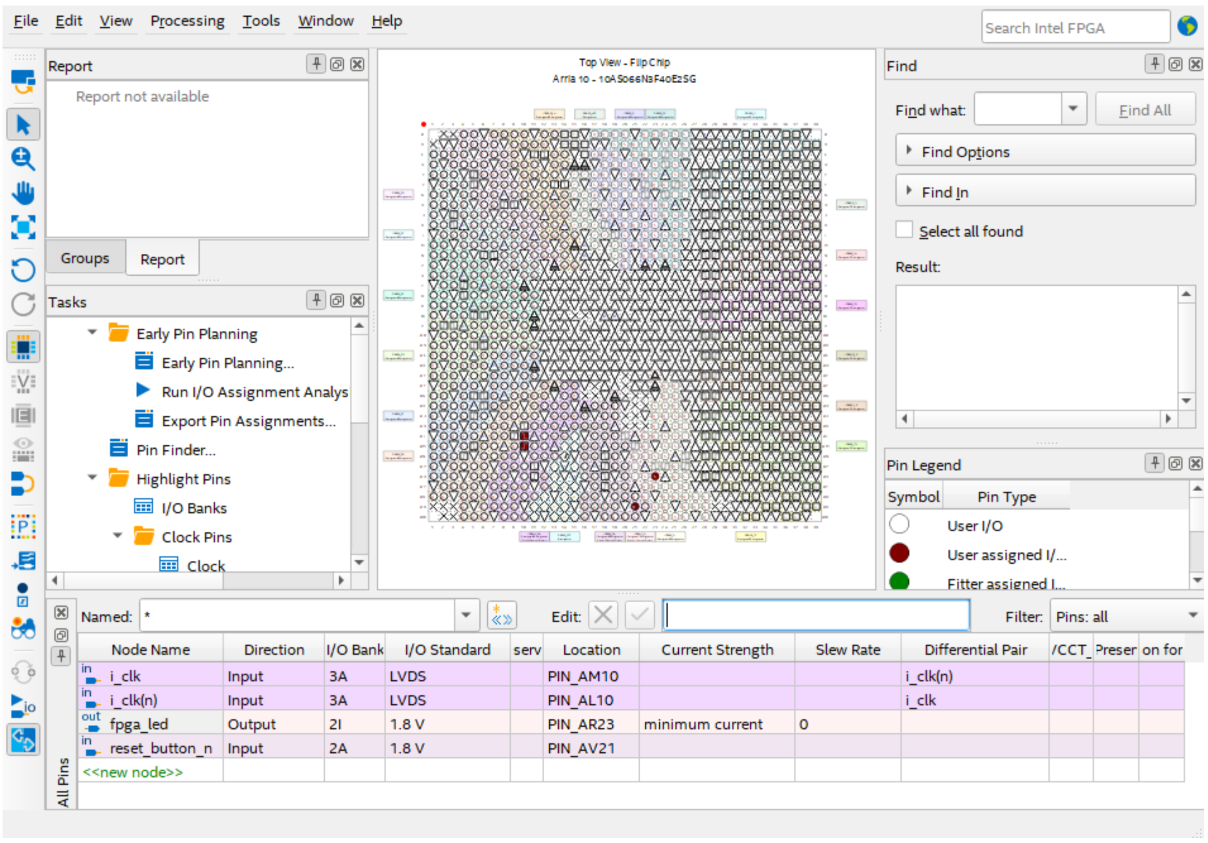
6. Run Analysis and Elaboration by clicking 'Start analysis and Elaboration' in the Quartus GUI.



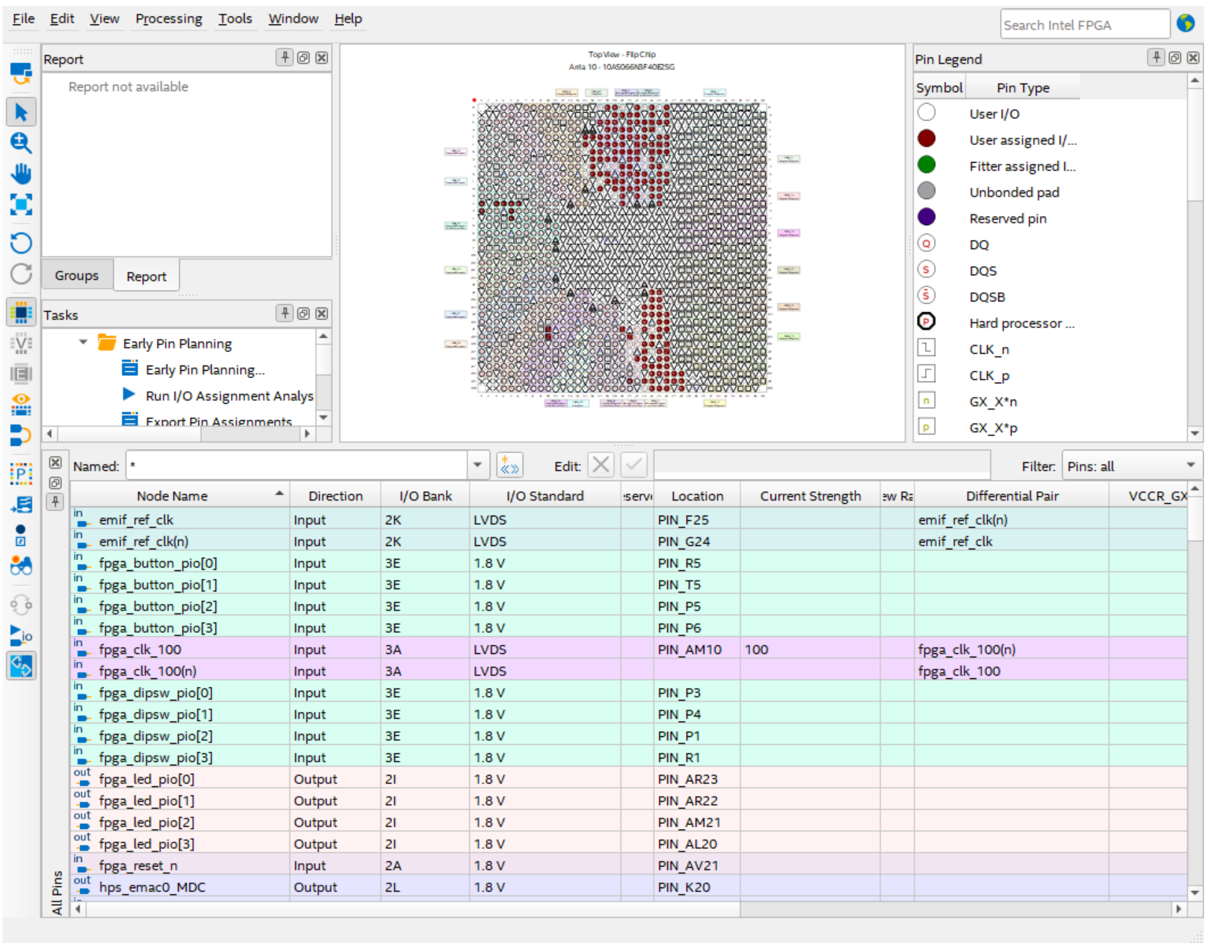
7. Now, we will select pins for the `i_clk` and `reset_button_n` inputs and `fpga_led` output. The JTAG to Avalon Agent IP will handle the connection between your design and the JTAG pins on your board automatically.

    1. Open the pin planner using `Assignments > Pin Planner` in the main Quartus GUI. Consult the data sheet for your board to choose an appropriate clock input. In this project, the `PIN_AM10` was chosen because it is used for supplying a 100MHz clock signal in the the Golden Hardware Reference Design (GHRD) source code.

2. Assign pins for the `fpga_led` and `reset_button_n` signals using the same methodology.



Node Name	Direction	I/O Bank	I/O Standard	serv	Location	Current Strength	Slew Rate	Differential Pair	/CCT_	Preser	on for
i_clk	Input	3A	LVDS		PIN_AM10			i_clk(n)			
i_clk(n)	Input	3A	LVDS		PIN_AL10			i_clk			
fpga_led	Output	2I	1.8 V		PIN_AR23	minimum current	0				
reset_button_n	Input	2A	1.8 V		PIN_AV21						
<<new node>>											



Node Name	Direction	I/O Bank	I/O Standard	servi	Location	Current Strength	ew Ra	Differential Pair	VCCR_GX
emif_ref_clk	Input	2K	LVDS		PIN_F25			emif_ref_clk(n)	
emif_ref_clk(n)	Input	2K	LVDS		PIN_G24			emif_ref_clk	
fpga_button_pio[0]	Input	3E	1.8 V		PIN_R5				
fpga_button_pio[1]	Input	3E	1.8 V		PIN_T5				
fpga_button_pio[2]	Input	3E	1.8 V		PIN_P5				
fpga_button_pio[3]	Input	3E	1.8 V		PIN_P6				
fpga_clk_100	Input	3A	LVDS		PIN_AM10	100		fpga_clk_100(n)	
fpga_clk_100(n)	Input	3A	LVDS					fpga_clk_100	
fpga_dipsw_pio[0]	Input	3E	1.8 V		PIN_P3				
fpga_dipsw_pio[1]	Input	3E	1.8 V		PIN_P4				
fpga_dipsw_pio[2]	Input	3E	1.8 V		PIN_P1				
fpga_dipsw_pio[3]	Input	3E	1.8 V		PIN_R1				
fpga_led_pio[0]	Output	2I	1.8 V		PIN_AR23				
fpga_led_pio[1]	Output	2I	1.8 V		PIN_AR22				
fpga_led_pio[2]	Output	2I	1.8 V		PIN_AM21				
fpga_led_pio[3]	Output	2I	1.8 V		PIN_AL20				
fpga_reset_n	Input	2A	1.8 V		PIN_AV21				
hps_emac0_MDC	Output	2L	1.8 V		PIN_K20				

8. Now you can add the timing constraints. This will vary from board to board.

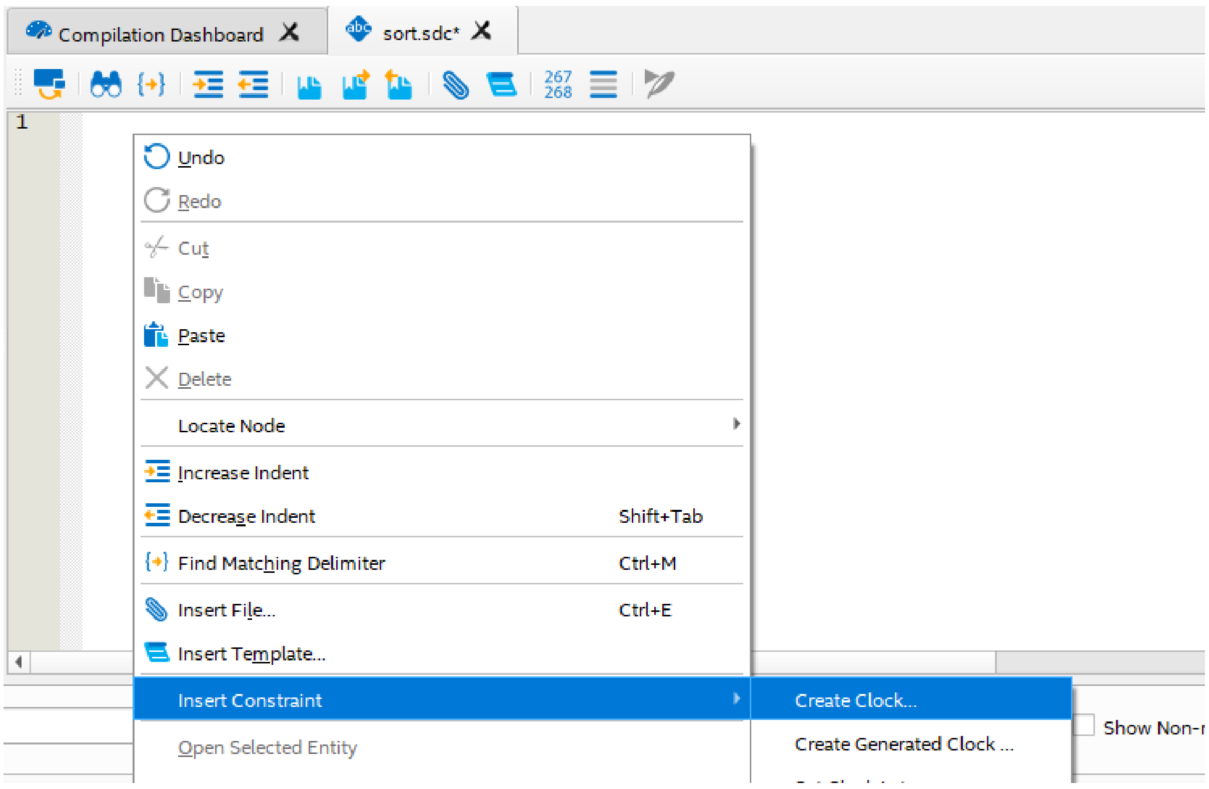
1. If you are using an official Intel FPGA devkit, you can find a timing constraints file for the JTAG interface (`jtag.sdc`) in the GHRD.
2. Create a new Synopsis Design Constraints (SDC) file and insert a new clock called `i_clk` to match the clock you defined in `sort.v`. Set the period to be 10ns:

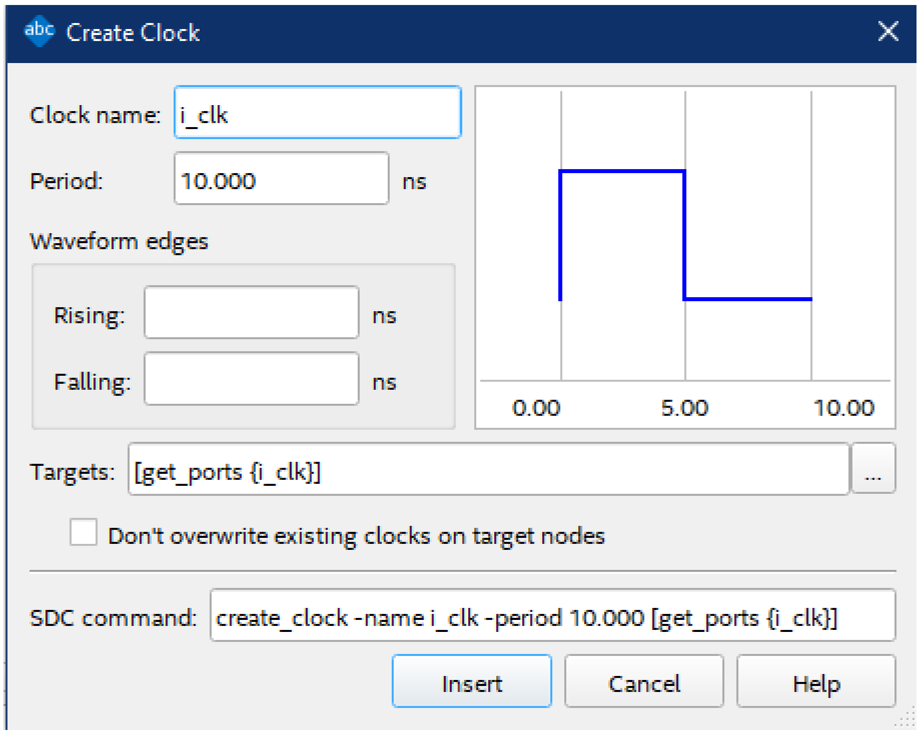
```
set_time_format -unit ns -decimal_places 3

create_clock -name i_clk -period 10 [get_ports {i_clk}]
```

3. Cut the clock paths for asynchronous I/O:

```
set_false_path -from [get_ports {reset_button_n}] -to *

set_false_path -from [get_ports {fpga_led}] -to *
set_false_path -from * -to [get_ports {fpga_led}]
```

If you run a full compilation, the Quartus GUI can help you populate the Targets of your SDC constraints.





Clock name: i_clk
Period: 10.000 ns
Waveform edges
Rising: ns
Falling: ns
Targets: [get_ports {i_clk}]
[ ] Don't overwrite existing clocks on target nodes
SDC command: create_clock -name i_clk -period 10.000 [get_ports {i_clk}]

9. Compile the full design by clicking the 'Start Compilation' button in the Quartus GUI.

   If you have any problems, compare your design with the files in this repository.

10. Test the design using the procedure in Step 6 above.