

Create a System Design with Platform Designer

Exercise Manual

Software Requirements:

The Intel® Quartus® Prime Standard Edition software version 17.1
Cyclone® V device support

Hardware Requirements:

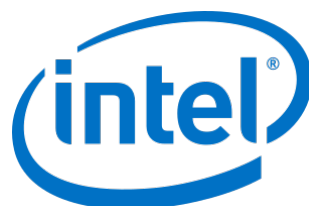
Terasic C5G Cyclone® V GX Starter Kit:

<https://www.altera.com/solutions/partners/partner-profile/terasic-inc-/board/cyclone-v-gx-starter-kit.html>

<http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=830>

Use the link below to download the design files for the exercises:

http://www.altera.com/customertraining/webex/Create_System_with_Platform_Designer.zip



DO NOT DISTRIBUTE

Exercise 1

Introduction to Platform Designer

Exercise 1

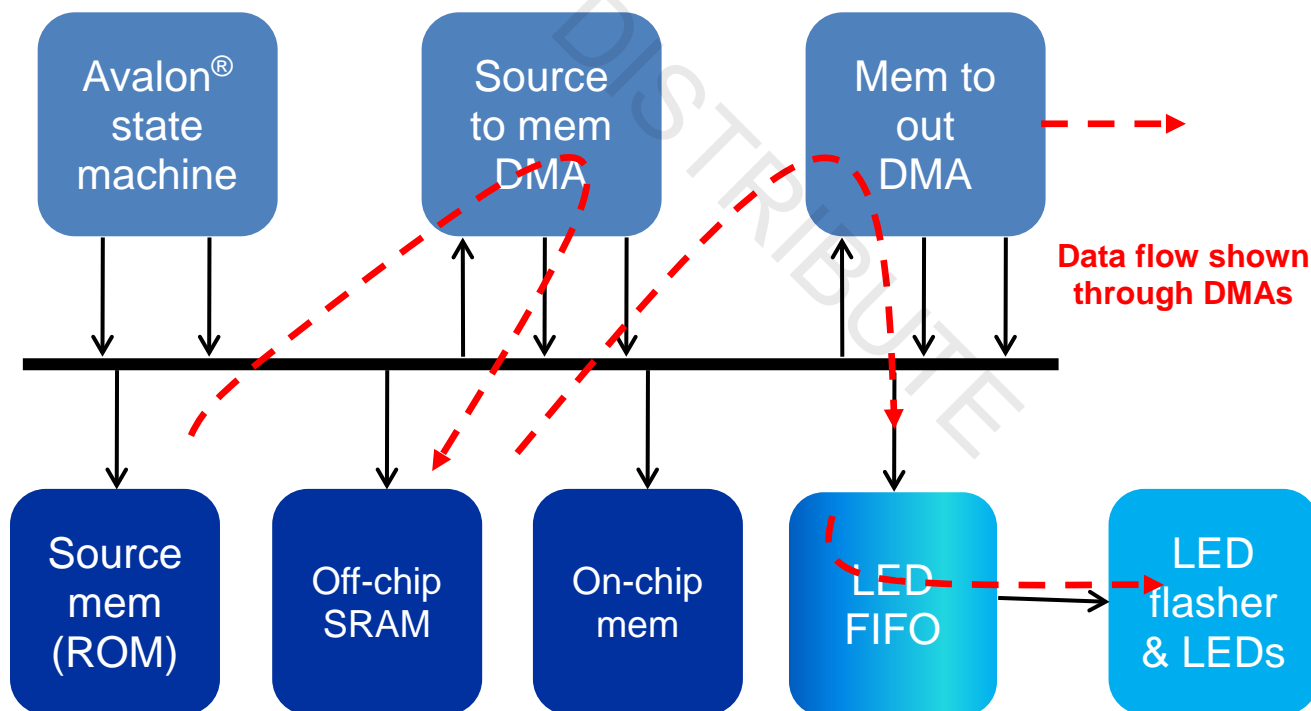
Objectives:

- Practice using the Platform Designer tool in the Intel® Quartus® Prime software's design flow
- Investigate the Platform Designer's user interface

Introduction:

In this exercise, you will start with a completed Platform Designer system (.qsys file) and use that file to gain an understanding of the tool's user interface. You will then take that file through the Intel Quartus Prime software design flow from synthesis all the way to programming the FPGA.

The design used in this exercise is intended to illustrate how systems can be built in Platform Designer without a processor. It uses a programmable state machine as the control block. The state machine is programmed with commands at instantiation-time in the system. This design also illustrates how direct memory access (DMA) blocks can be used to easily move data around the system, thereby reducing the complexity of your control block or state machine. In other words, you can have a system that performs complex data processing, but implements it with a simple state machine master plus DMAs.



The block diagram above represents the system used in this exercise. In this system, the state machine programs the DMAs to transfer data. After programming the DMAs, the state machine block must then monitor their status registers to determine when their transfers are complete.

The **Source to mem DMA** block is programmed to transfer a fixed amount of data from the **Source mem** block (an on-chip FPGA memory) to the external **SRAM**. For this exercise, the values in the **Source mem** block represent a pattern to be displayed on the LEDs and are pre-loaded during FPGA configuration. However, they could just as easily be loaded from a data generator like an ADC converter or a video camera.

The **Mem to out DMA** block is programmed to transfer a fixed amount of data from the external **SRAM** to the **LED FIFO** block. The transferred data is also “backed up” in a separate on-chip memory. Data is sent through the **LED FIFO** block to the green LEDs on the Cyclone V GX development kit. The LEDs are connected to the system through a custom block named **Avalon ST MM Flasher**. This component flashes the green LEDs on and off for a programmable number of clock cycles each time the block is written to. This models the behavior of a slower component that may exist in a system. Instead of slowing the entire system down, the **LED FIFO** block buffers the data between the system and the LEDs. The LED flasher block also drives the red LEDs, which are lit by turning on any of the slide switches on the board, and the four 7-segment displays. **HEX0** (the rightmost display on the board) always shows the hex value of the green LEDs, though the display is turned off when the value is 0 for easier viewing. **HEX3**, **HEX2**, and **HEX1** represent the hex value of the red LEDs. The 2 upper bits of the 10 LEDs on the board are displayed on **HEX3**.

All of the interfaces in the system are Avalon®-MM except for the **LED FIFO** to LED flasher/LED interface, which is Avalon-ST. As mentioned, the use of standard interfaces eases the connection between system components. The **LED FIFO** block also serves as an adapter, converting received Avalon-MM write transfers into Avalon-ST transfers.

As you proceed through the exercises, be sure to completely read the instructions for each step and sub-step in this lab manual. Each step first summarizes what you will be doing in that step before providing detailed instructions. Use the lines next to each step (____) to keep track of your progress or to check off completed steps in the exercises.

DO NOT DISTRIBUTE

Step 1: Create a project and open the .qsys file

- _____ 1. Before starting the Intel Quartus Prime software, unzip the lab project files from the **.zip** linked at the beginning of this lab manual.

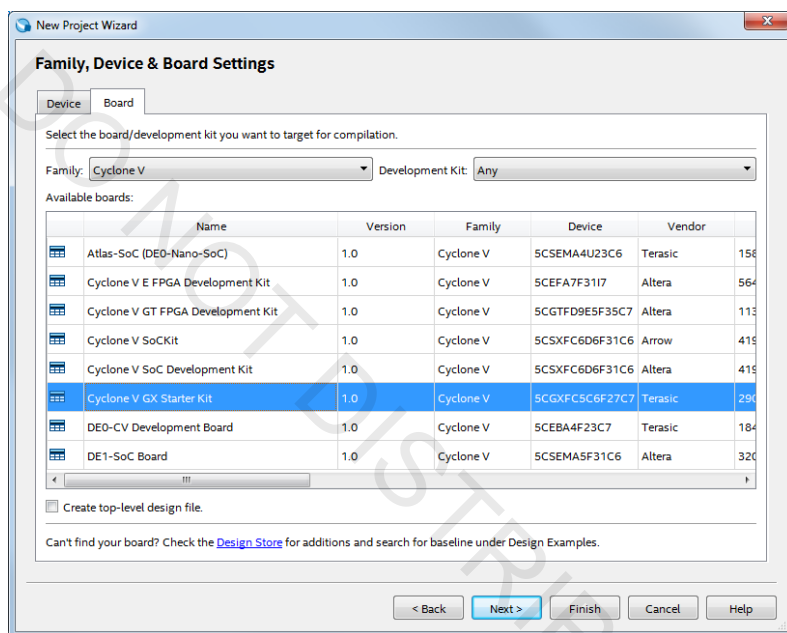
*In the **.zip** file is a folder named **IPD17_1**. Place the file wherever you'd like. The path to the subfolder **IPD17_1** will be referred to as the **<lab install directory>**.*

- _____ 2. Start the Intel Quartus Prime Standard Edition software, version 17.1, from the **Start** menu (**All Programs** → **Intel FPGA 17.1.0.585 Standard Edition** → **Quartus Prime Standard Edition 17.1.0.585** → **Quartus (Quartus Prime 17.1)**).

***Important:** you should confirm by looking at the status bar at the top or by selecting **About Quartus Prime** from the **Help** menu that you indeed have the Standard Edition and version 17.1.0 of the software opened. There are a number of versions of the software installed on the training computers, and this lab was designed for this version. However, newer versions of the software may still work. Your instructor should let you know if a newer version can be used.*

- _____ 3. Create a project.
- _____ 4. From the **File** menu, select **New Project Wizard**, or click **New Project Wizard** on the **Home** tab when you first start the software.
- _____ 5. When the Introduction page appears, click **Next** to continue.

- ____ 6. Choose the following options, clicking **Next** to go to each page:
1. **Directory, Name, Top-Level Entity:** Set the **working directory** to **<lab install directory>\Lab1** and name the project and top-level design entity **sm_transfer_system**.
 2. **Project Type:** Set to **Empty Project**.
 3. **Add Files:** Leave blank as you will add files later in the exercise.
 4. **Family, Device & Board Settings:** Switch to the **Board** tab and select the **Cyclone V GX Starter Kit** from Terasic. Make sure to turn **off** the option to **Create top-level design file**.



5. **EDA Tool Settings:** In the **Simulation** row, make sure the **Tool Name** to **<none>**.
6. Click **Finish**.
- ____ 7. From the **File** menu, select **Open** and open the file named **sm_transfer_system.qsys**.

*You also could have opened Platform Designer from the **Tools** menu (or toolbar) and then opened the **.qsys** file from the tool's **File** menu. Since the **.qsys** file is viewed by the tool as a valid design entry file (like an HDL file), you can open it directly from the Intel Quartus Prime software, as described.*

Use	Connections	Name	Description	Export	Clock	Base	End
<input checked="" type="checkbox"/>		clk	Clock Source	clk_clk_in	exported		
clk_in_reset		Clock Input	clk_clk_in_reset				
clk		Clock Output	<i>Double-click to export</i>	clk			
clk_reset		Reset Output	<i>Double-click to export</i>				
<input checked="" type="checkbox"/>		reset_debounce	Reset Button Debounce	<i>Double-click to export</i>	sys_clk		
clock		Clock Input	<i>Double-click to export</i>	[dock]			
button_debounce_in		Reset Input	<i>Double-click to export</i>	[dock]			
button_qual		Reset Output	<i>Double-click to export</i>	[dock]			
<input checked="" type="checkbox"/>		pll	Altera PLL	<i>Double-click to export</i>	clk		
refclk		Clock Input	<i>Double-click to export</i>				
reset		Reset Input	<i>Double-click to export</i>	sys_clk			
outclk0		Clock Output	<i>Double-click to export</i>	ssram			
<input checked="" type="checkbox"/>		button_switch	Button Switch PIO	<i>Double-click to export</i>	sys_clk		
clock		Clock Input	<i>Double-click to export</i>	[dock]			
reset		Reset Input	<i>Double-click to export</i>	[dock]			
buttonreg		Avalon Memory Mapped Slave	<i>Double-click to export</i>	0x0000_1040	0x0000_104f		
<input checked="" type="checkbox"/>		av_sm_master	Avalon State Machine Master	<i>Double-click to export</i>	sys_clk		
clock		Clock Input	<i>Double-click to export</i>	[dock]			
reset		Reset Input	<i>Double-click to export</i>	[dock]			
avalon_master		Avalon Memory Mapped Master	<i>Double-click to export</i>	[dock]			
<input checked="" type="checkbox"/>		led_out	Avalon ST MM Flasher	<i>Double-click to export</i>	sys_clk		
clock		Clock Input	<i>Double-click to export</i>	[dock]			
reset		Reset Input	<i>Double-click to export</i>	[dock]			
led		Avalon Memory Mapped Slave	<i>Double-click to export</i>	0x0000_1030	0x0000_103f		
swpb		Avalon Memory Mapped Master	<i>Double-click to export</i>	[dock]			
ledfifo		Avalon Streaming Sink	<i>Double-click to export</i>	[dock]			
greenled_out		Conduit	greenled_out	[dock]			
hex0_out		Conduit	hex0_out	[dock]			
redled_out		Conduit	redled_out	[dock]			
hex1_out		Conduit	hex1_out	[dock]			
hex2_out	Conduit	hex2_out	[dock]				
hex3_out	Conduit	hex3_out	[dock]				
<input checked="" type="checkbox"/>		led_fifo	Avalon FIFO Memory	<i>Double-click to export</i>	sys_clk		
clk_in		Clock Input	<i>Double-click to export</i>	[clk_in]			
reset_in		Reset Input	<i>Double-click to export</i>	[clk_in]			
in		Avalon Memory Mapped Slave	<i>Double-click to export</i>	mixed	mixed		
		out	Avalon Streaming Source	<i>Double-click to export</i>	[clk_in]		

You should see the file shown above.

Step 2: Investigate the system using the Platform Designer user interface

Now you will use the features of the Platform Designer UI to gather information about the system, starting with the **System Contents** tab.

- ____ 1. Locate the component named **clk** in the system, the **Clock Source** component.

⊟ clk	Clock Source	
⏏ clk_in	Clock Input	clk
⏏ clk_in_reset	Reset Input	reset
⏏ clk	Clock Output	<i>Double-click to export</i>
⏏ clk_reset	Reset Output	<i>Double-click to export</i>

The **clk** component has four interfaces to connect: one Clock Input, one Reset Input, one Clock Output, and one Reset Output. The two input interfaces have been exported from the system and named **clk** and **reset** as indicated by the **Export** column and the pin symbols to the left of the interface names.

- ____ 2. Highlight the **clk** (Clock Output) interface from the **clk** module.

⏏ clk	Clock Output	
⏏ clk_reset	Reset Output	
⊟ pll	Altera PLL	
refclk	Clock Input	

The Connections panel shows an arrow that points away from the **clk** interface and into the **refclk** clock input interface of the **pll** component. This means that the clock is being driven out of the clock source component and serves as the input clock to the **pll** component.

- ____ 3. Click in the **Clock** column for the **pll refclk** interface. Do not change the clock source!

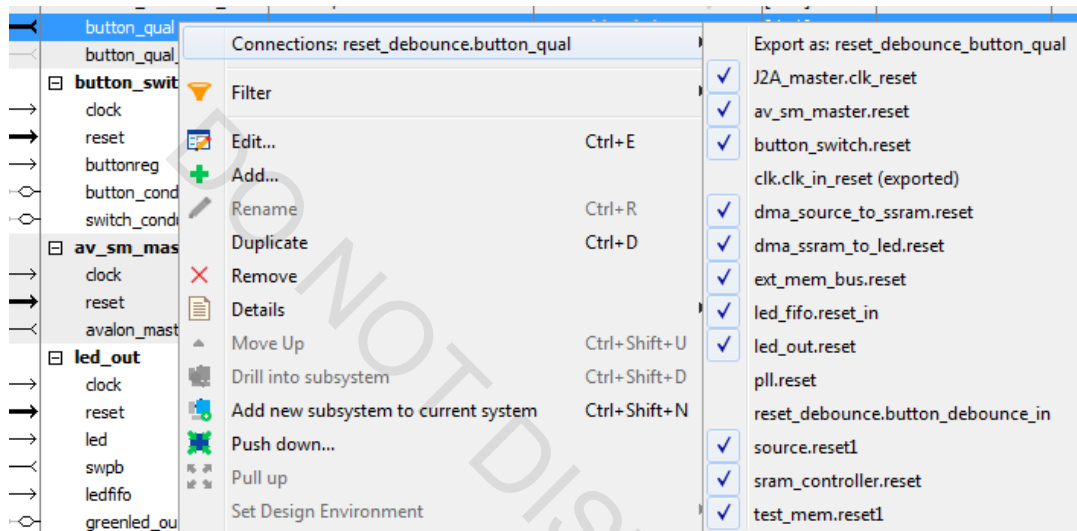
The pop-up menu here lists all the clock domains (clock output interfaces) in the system. Use it to quickly connect clock outputs to clock inputs.

- ____ 4. Highlight the **clk_reset** (Reset Output) interface of the **clk** component. Take a look at the **Connections** panel.

The **Connections** panel displays a matrix with toggles that allow you to make or break connections between component interfaces. You can see the black dots that indicate that the **Reset Output** interface of the **clk** module is connected to the reset input of the PLL and the reset input of a **Reset Button Debounce** component. To break a connection, you simply click on a black dot. To make a connection, you simply click on an empty, gray dot.

The **Reset Button Debounce** component is an example of a custom component, a component that is created using RTL coding and then added into the Platform Designer IP Catalog for use just like any other component. You'll see more custom components in this example system design and learn how to create them later in the training.

5. Examine all debounced reset connections. Right-click the **button_qual** output interface of the **reset_debounce** component and go to the **Connections:** **reset_debounce.button_qual** submenu. Observe all the connections. *Do not enable or disable any connections!*




This is another way you can make or break connections between component interfaces.

6. From the **View** menu, select **Connections**. If this view is blank, switch back to the **System Contents** tab and re-select the **button_qual** interface.

This is yet another way to manage connections between interfaces. Feel free to use any of these methods through these lab exercises, but do not change any of the connections here.

7. Switch back to the **System Contents** tab.

8. Examine the connection filters:

- Click the **Filters** button  in the toolbar at the bottom of the **System Contents** tab.
- In the **Filter:** drop-down menu of the **Filters** dialog box, choose **Clock and Reset Interfaces**.
- Close the **Filters** dialog box and review the **System Contents** tab.

The **System Contents** tab should now only show you the clock and reset interfaces in the system. The **Filters** window allows you to filter out certain connections or components. This greatly helps in larger systems where there may be a number of many different types of connections. You can filter your view to only the connections you want to verify.

9. Reopen the **Filters** dialog box and change the **Filter:** drop-down menu from **Clock and Reset Interfaces** back to **All Interfaces**. Do not close the dialog box yet.

10. Still in the **Filters** dialog box, click **New**. In the **Name:** field, type **State Machine**.

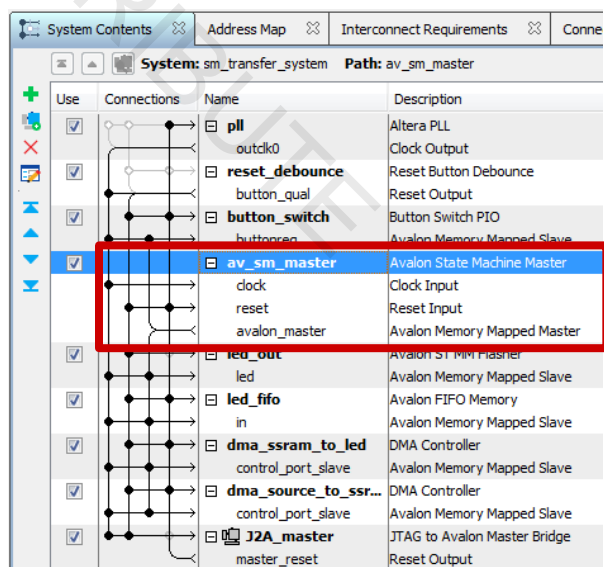
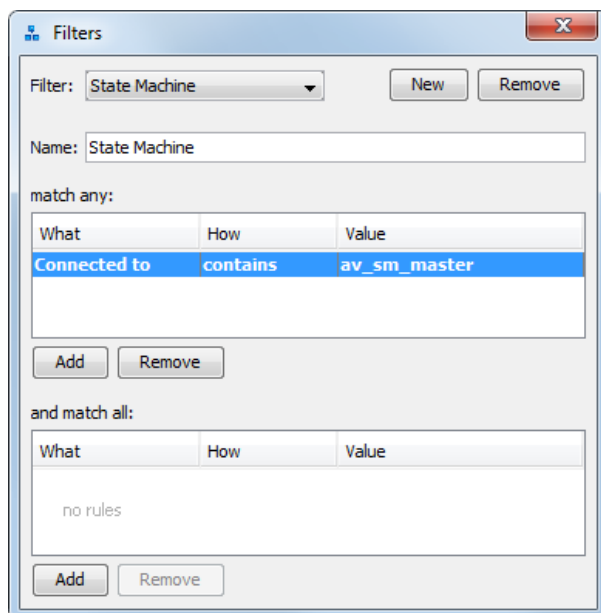
11. At the bottom of the **match any:** section, click **Add**.

12. Under the **What** column, click **Tag** and from the list, select **Connected to**.


13. Under the **How** column, choose **contains** from the pop-up list.

14. Click in the **Value** field and type **av_sm_master**.

15. Click anywhere else in the dialog to accept the filter settings.




Your **Filters** dialog box should look like the above screenshot. With the custom filter enabled, only components with interfaces that connect to the state machine's interfaces are displayed.

16. Click the remove filter button  or reopen the **Filters** dialog box again, if necessary. Change the **Filter:** drop-down back to **All Interfaces**. Close the **Filters** dialog box.

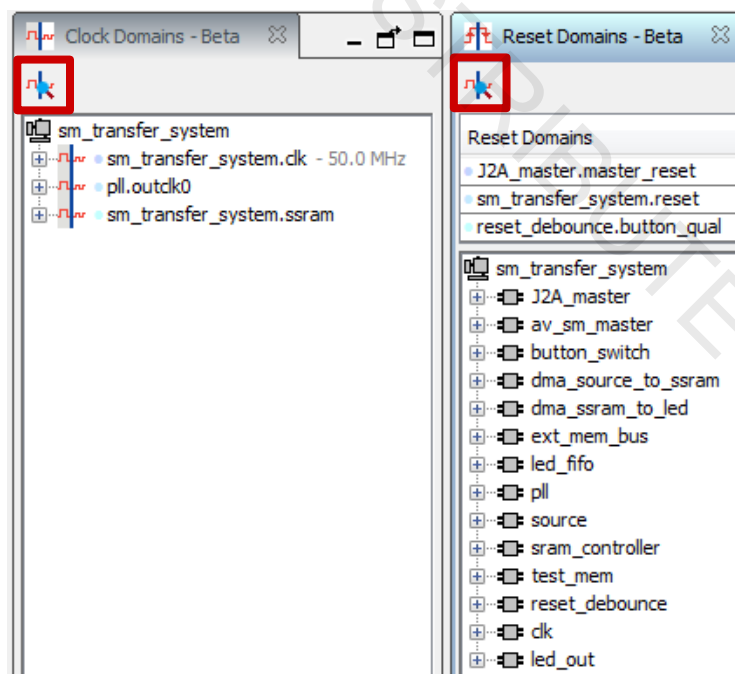
You can also quickly enable or disable filters by right-clicking anywhere in the **System Contents** tab and going to the **Filters** submenu.


17. Display only connected interfaces. Highlight the **button_qual** interface of the **reset_debounce** component again. Right-click, go to **Filter**, and select **Show Connected**.

You have now filtered the **System Contents** tab to only display interfaces that are connected to the **Reset Output** interface of the debounce component. This is another way to quickly use the filtering capabilities of the tool.

18. Set the view back to its original state. Right-click the **clk_reset** interface of the **clk** module and select **All Interfaces** from the **Filter** submenu, or click the remove filter button .

19. From the **View** menu, select **Clock Domains – Beta** and **Reset Domains - Beta**. Select different domains and interfaces in these tabs, and see what happens in the **System Contents** tab.



As the names imply, these tabs allow you to look at all the clock and reset domains in the system and all the interfaces that use each one. Clock names for clocks associated with each clock output interface appear in the **System Contents** tab in the **Clock** column. Clicking a domain highlights, in color, the clock or reset input and output interfaces and their associated connections used in that domain. Enabling the button at the top of either tab filters the interfaces in the **System Contents** tab to display only the interfaces connected to a particular domain in a similar manner to the filtering just performed. You can click the clock and reset interface filtering buttons () as a shortcut to opening these tabs and to remove the highlight coloring when you are finished.

- ____ 20. Observe the connections from an Avalon master interface in the Connections panel. Click on the **avalon_master** interface in the **av_sm_master** component.

You can see that the **avalon_master** interface can access the slave interfaces of the Button Switch PIO component (**button_switch**), the DMA control ports, the LED FIFO, and the LED flasher.

- ____ 21. Observe the connections from the Avalon master in list form. Right-click the **avalon_master** interface and go to the **Connections:** **av_sm_master.avalon_master** submenu. Do not enable or disable any connections in the submenu!

Again, you can use this method to make/break connections with Avalon slave interfaces versus using the **Connections** panel.

- ____ 22. Observe the connections to and from the DMAs. Click the **read_master** and **write_master** interfaces of the two DMA components to view their connections.

Note that, except for **dma_ssram_to_led.write_master**, each of the DMA master interfaces connects to a single slave.

- ____ 23. Locate the **led_out** component. Highlight the **ledfifo** interface.

Notice that unlike the interfaces you have examined so far, this is a streaming interface, so it does not have an address. Streaming interfaces are point-to-point connections.

- ____ 24. Look at the very bottom of the **Messages** tab for an information message about this interface's connection.

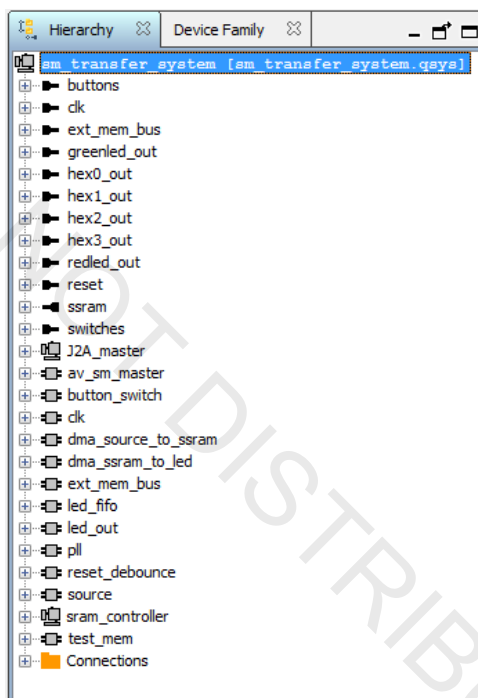
There is a timing difference (number of cycles for use of the **ready** signal) between the streaming source interface of **led_fifo** and the sink interface of **led_out**. The tool has automatically added an adapter into the interconnect to handle this. You will learn more about interface timing and the available adapters later in the class.

- ____ 25. Highlight any of the **Conduit** interfaces of the **led_out** component.


As mentioned in the presentation, a conduit interface is a group of signals representing a non-standard interface. For this component, the conduits include the exported signals that connect directly to all the LEDs on the development kit. You'll see the individual signals that make up these conduits in a moment.

*Now you will look at the **Hierarchy**, **Parameters**, and **Block Symbol** tabs of the tool, which let you quickly review and edit system, component, and connection details as needed.*

- ____ 26. If for some reason it is not visible, select **Hierarchy** from the **View** menu.



- ____ 27. Also from the **View** menu, select **Parameters** and **Block Symbol** if they are not already open.

*Note that if you ever want to return the view back to the default set of tabs, just select **Reset to System Layout** from the **View** menu. Also note that you can drag and drop any tab anywhere in the interface to rearrange it or disconnect it from the interface  into its own separate window. Feel free to organize the tabs however you want.*

28. In the **Hierarchy** tab, select the top-level system design, **sm_transfer_system**.

*When you do this, in the **Block Symbol** tab, you'll see a top-level block diagram of the system. You should also notice that the **Parameters** tab displays top-level project settings. Some of these settings can also be found in the **Device Family** and **Interconnect Requirements** tabs. On the block symbol diagram and listed in the **Hierarchy** tab are the names of all of the top-level exported interfaces.*

*Selecting something in the **Hierarchy** or other views automatically displays or highlights information for the selected object in every other view.*

29. In the **Block Symbol** tab, enable the **Show signals** option.

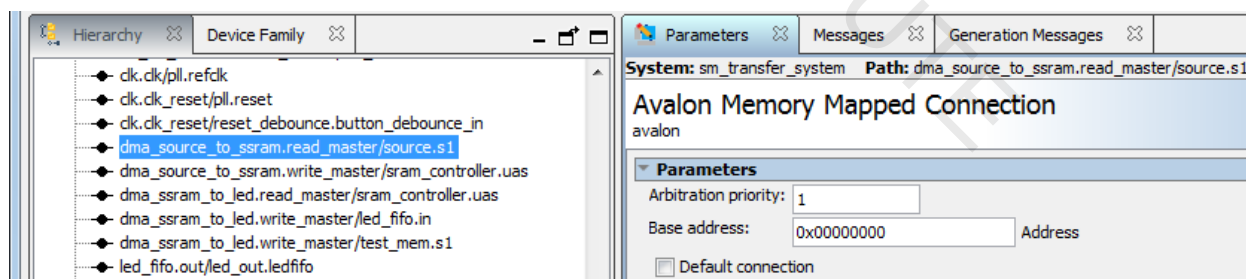
*All of the top-level interfaces are expanded out to their individual signals. Notice that the **ext_mem_bus** exported interface is a tristate master conduit (**tcm**) interface, allowing for access to multiple external devices on a tristate bus. We'll look at this in more detail later in the presentation.*

30. Turn off the **Show signals** option.

31. Back in the **Hierarchy** tab, expand the **greenled_out** exported interface, and highlight the **greenled_out_export[8]** port. View the **Parameters** for the signal.

*This lets you see details on the individual signals (ports) that make up this interface. There are no editable parameters for this particular interface, but if there were, you could edit them directly on the **Parameters** tab. This and the other top-level exported interfaces consist of the signals that can be mapped to FPGA I/O locations, connected to other Platform Designer systems when building a hierarchical design, or connected to other logic in the Intel Quartus Prime software project.*

32. Expand the **Connections** folder at the very bottom of the **Hierarchy** tab (you may need to scroll down to see it). Locate and select the **dma_source_to_ssram.read_master/source.s1** connection.



*Notice that this is an **Avalon Memory Mapped Connection**. In the **Parameters** tab, you can edit the slave **Base address**, which is the address the **dma_source_to_ssram read_master** interface must use to address the source memory slave interface. You can also set the arbitration priority for this master-slave connection in case other masters are connected to this slave.*

- ____ 33. Review a few other connections like the **led_fifo.out/led_out.ledfifo** and the **pll.outclk0/av_sm_master.clock**.

*These are examples of an **Avalon Streaming Connection** and a **Clock Connection**, respectively. These connections should make sense with the connections defined in the System Contents tab. However, these connections have no editable parameters, so the **Parameters** tab is blank.*

- ____ 34. Collapse the **Connections** folder in the **Hierarchy** tab. Find and highlight the **clk** component in the Hierarchy. Components have little chip symbols (⌚) in the Hierarchy.

*Now you should see the parameters for the **Clock Source** component. These are the same settings that appear in the component's Parameter Editor when the component is first added to the system. From here, you can quickly and easily change component settings if needed without having to remove and re-add the component to the system.*

- ____ 35. Highlight the **av_sm_master** component in the Hierarchy.

*This displays the component parameters for the highlighted system component, the state machine instance named **av_sm_master**. The **av_sm_master** module is another example of a custom component. This component uses programmed transactions found in the **Avalon Commands** parameter setting (scroll down in the UI to see it) to create a custom-built state machine written in HDL. (Read over the component description in the **Parameters** or **Details** tabs to learn more.) This state machine serves as the Avalon master to the system, controlling the DMAs and LEDs. Note again that you can quickly change component settings here if needed.*

Important note: If you are running this lab on your own computer in Linux (Intel FPGA training computers all run Windows), you need to modify a file for the state machine to allow it to operate correctly (you'll need to do this for Lab 3 as well). The state machine uses a Perl script to create HDL code based on parameters (discussed later), and Perl needs to be called differently in Linux versus Windows. In the Intel Quartus Prime software or in a text editor, open the files:

```
<lab install directory>/Lab1/avalon_state_machine_master/avalon_state_machine_master_hw.tcl
```

```
<lab install directory>/Lab2_3/ip/avalon_state_machine_master/avalon_state_machine_master_hw.tcl
```

Near the very end of the files (line 180), change....

```
# Call perl to execute perl script to generate HDL
global perl
exec $perl/perl.exe ...
```

to simply....

```
# Call perl to execute perl script to generate HDL
perl ... # If this doesn't work, use "exec perl ..."
```

- ____ 36. Expand the **av_sm_master** component and select the **avalon_master** interface.

*The **Parameters** tab now indicates the type of interface and the interface settings. Scroll down in the window to see timing waveform diagrams for this interface. This is helpful for understanding the interface's timing requirements and for debugging an interface. We'll be looking at Avalon timing and signaling in the next section of the class.*

- ____ 37. Back in the **System Contents** tab, look at the base address (**Base** column) for the interface **led_fifo.in**. Click this interface to help understand what this means.

*As discussed in the presentation, when multiple masters access the same slave, the base address for the slave can be unique for each master. The **mixed** text indicates that multiple masters access this slave. You can explore this on the **Address Map** tab.*

- ____ 38. Select the **Address Map** tab at the top of the Platform Designer window. If you don't see it, select **Address Map** from the **View** menu.

*The **Address Map** tab shows where slave interfaces are located in the memory maps of each master interface. Each column represents a master in the system (be sure to scroll to the right to see all the masters). Each row is a slave interface. Notice that the first three DMA master interfaces and the **led_out** master interface each connect to only one slave. Because of this, each respective slave serves as the entire address range of the connected master.*

*The fourth DMA master interface and the state machine master interface connect to multiple slaves. Each of their address ranges is the collection of addresses of the slaves they can access. The **led_fifo.in** slave interface (the one indicated with **mixed** on the **System Contents** tab) happens to have the same address (0x00000000) for both masters that connect to it in this system. This is OK because the address maps are separate for each master.*

The address range of the state machine master interface is 0x00000000 – 0x0000104f, representing the full address ranges of all the slaves connected to it. There are open gaps in the memory map, but this is OK. Accesses to unallocated address space are handled automatically by the interconnect. We'll look at this in the next section. You can double-click on a cell in the address map to change the address of a slave for a particular master. Care must be taken to not overlap addresses or an error message will appear.

*There is one more master in the system. The **J2A_master** component is a JTAG to Avalon bridge component used for run-time system debugging. This bridge's master interface must be connected to any slave interfaces that require access during debugging. Later in this exercise, as an optional method for running the design, you will have a chance to use a debugging tool called System Console. System Console uses the JTAG to Avalon master bridge to connect to and control and monitor the system.*

- ____ 39. From the **View** menu, select **Schematic**.

This view provides a familiar schematic block diagram of the components in the system and the connections between them. Some components have sub-components that can be displayed by clicking the + sign in the component. Different types of interface connections can be hidden using the filter toggle buttons at the bottom of the view. You can also make or break interface connections by right-clicking component blocks.

- ____ 40. Finally, open the **Device Family** and **Interconnect Requirements** tabs from the **View** menu if they are not already open. Leave the default settings.

On these tabs, you can select the target device family, the type of logic structure to be used for clock domain crossing, and a limit on how many pipeline stages you want to have in the interconnect to improve system performance.

Step 3: Generate the system

The next step is to generate the system, which you can do manually in Platform Designer (which we'll be doing), or have performed automatically during a full compilation in the Intel Quartus Prime software. When the system is generated, HDL files are created that represent the system and its interconnect logic. These are then compiled by the Intel Quartus Prime software just like any other design files. For generation, you can safely ignore the warning messages that appear for this system.

- ____ 1. From the **Generate** menu, select **Show Instantiation Template**.

If the system design is not going to be the top-level design entity in an Intel Quartus Prime software project, this template can be used to instantiate the system, just like any other HDL hierarchy design instantiation. The ports listed in the template match the exported interfaces in the system and the system block symbol, displayed earlier. For this exercise, the system will be the top-level design entity, actually the entire design, so this template is not required.

- ____ 2. **Close** the template.

- ____ 3. Again from the **Generate** menu, select **Generate HDL** or click **Generate HDL** in the lower right-hand corner of the tool.

- ____ 4. Enable generation of synthesis files.

- a. In the **Synthesis** section of the Generation dialog box, set **Create HDL design files for synthesis** to **Verilog**.
- b. Disable the **Create block symbol file (.bsf)** option.


Do not be concerned if you are more familiar with VHDL than Verilog. You will not be editing any code in this exercise.

*Since the design is entirely in HDL, you do not need to generate a symbol file for use in a schematic. Note the **Output Directory** path for output **Synthesis** files. The default is a new directory named after the system, located in the current project directory. This can be changed to any location.*


- _____ 5. Click **Generate** to generate the system, saving the **.qsys** file if prompted. Any warnings that appear can be safely ignored.
- _____ 6. Click **Close** when generation is complete.

Step 4: Compile system in the Intel Quartus Prime software and review results

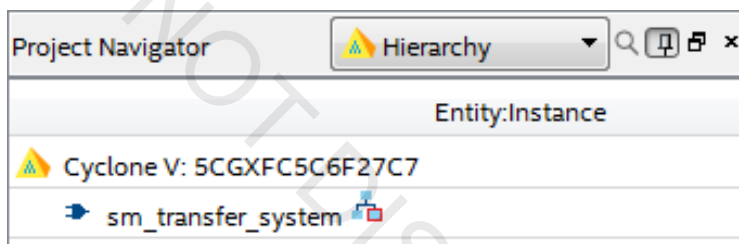
*Before compiling, you need to add all of the HDL files generated by Platform Designer to the project. There are two ways to do this: either add the **.qsys** file to the project, which will cause the system to be regenerated during compilation, or add the **.qip** file to the project. The **.qip** file is created when the system is generated manually in Platform Designer, so this will save some compile time. You can only add one file or the other to the project.*

- _____ 1. Add the **.qip** file to the project.
 - a. From the Intel Quartus Prime software **Project** menu, select **Add/Remove Files in Project**.
 - b. Click the browse  button.
 - c. Change the file type list filter to **IP Variation Files (*.qip *.qsys *.sip *.ip)**.
 - d. Browse to the **sm_transfer_system\synthesis** subdirectory and **Open** the file named **sm_transfer_system.qip**.
 - e. If the file does not immediately appear in the files list, click **Add** to actually add the file to the project (easy to miss!). Don't close the **Settings** dialog box yet.

*The **source** ROM component in the system (the **Source mem** block in the diagram at the beginning of this lab) gets pre-loaded during FPGA device configuration. For this to happen, you should also add the **.hex** memory initialization file that contains the ROM data to your project.*

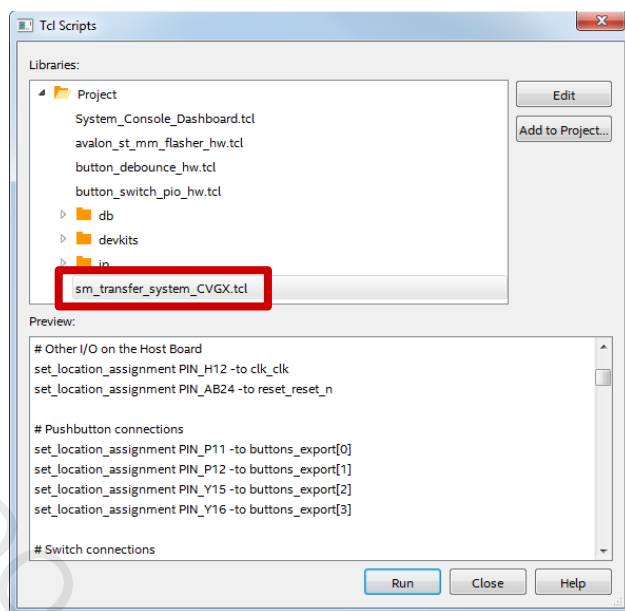
- ____ 2. Add the **.hex** file for the **source** memory to the project.
- Click the browse  button again and navigate up to the project directory (<**lab install directory**>\Lab1), if necessary.
 - Change the file type list filter to **Memory Files (*.mif *.hex)** and **Open** the file named **source.hex**.
 - Again, click **Add**, if necessary, to actually add the file to the project.
 - Click **OK** to close the **Settings** dialog box.



***Note:** The **source** ram in your system specifically points to the **source.hex** file, so if you were to test out a different memory initialization file, you would not only need to include the correct file in the project but also initialize the on-chip memory accordingly in the system. This can be done by either double-clicking the memory component in Platform Designer to open the on-chip memory's parameter editor or by simply selecting the memory in the **Hierarchy** tab and editing the settings on the **Parameters** tab.*



Shown above is the **Project Navigator** with **sm_transfer_system** defined as the top-level entity, matching the name of the system.

Now we need to constrain the design for synthesis and fitting. This includes I/O location assignments, optimization settings, and SDC timing constraints. To accelerate this process, a Tcl script file and a **.sdc** file have been provided for you.



- ____ 3. Execute the constraint Tcl script.
- From the **Tools** menu, select **Tcl Scripts**.
 - In the **Project** folder at the bottom of the list, choose the Tcl file named **sm_transfer_system_CVGX.tcl**, similar to the above screenshot.
 - Click **Run**. Click **OK** when the script completes, then click **Close**.
- ____ 4. Add the **.sdc** file to the project.
- Open the **Settings** dialog box from the **Assignments** menu again.
 - Select the **TimeQuest Timing Analyzer** category. You should see two **.sdc** files already added to the project that are pointed to by the **.qip** file (expand the **.qip** file to see them).
 - Click the browse  button and select **sm_transfer_system.sdc** and click **Open**.
 - Click **Add** to actually add the file to the project if necessary (*again, easy to miss!*).
 - Click **OK** to close the **Settings** dialog box.
- ____ 5. Compile the design. In the toolbar, click the  button or choose **Start Compilation** from the **Processing** menu. Any warnings that appear can be safely ignored.

The compilation should take about 5-10 minutes.

- _____ 6. Use the **Project Navigator** to explore the design hierarchy. You should be able to find the components that make up the design (e.g. PLL, DMAs, source memory, LED output) as well as the components generated by Platform Designer that make up the interconnect (e.g. routers, muxes, demuxes, translators).

Step 5: Program the FPGA and run the design on the board

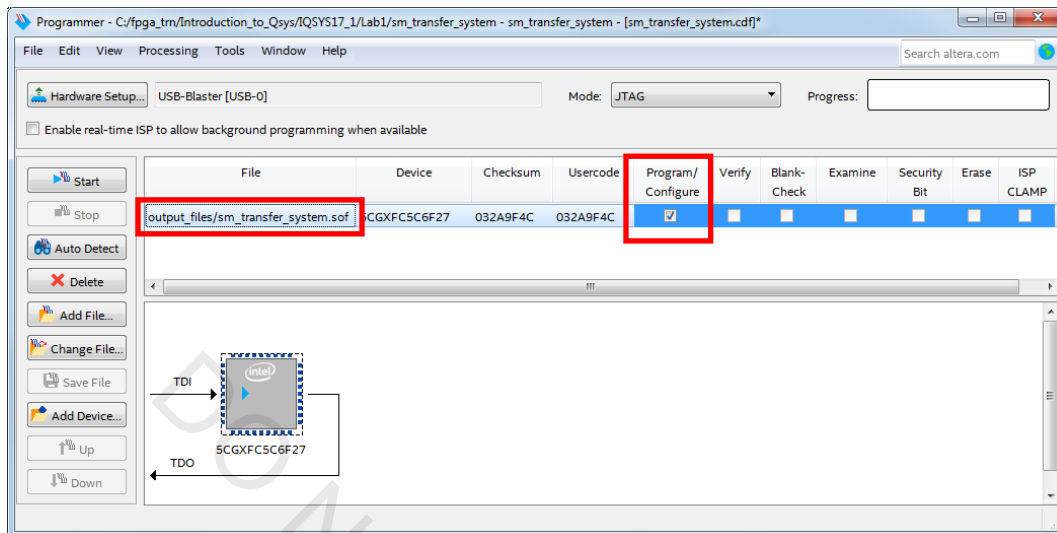
With the design fully compiled, you can now program the FPGA on the development kit.

- _____ 1. Take out and power up your development kit. Connect the USB cable to the board as well. Make sure you connect the USB-B connector to the connector named **USB BLASTER** on the board (there are 2 USB-B connectors; the correct one is on the left, nearer to the power jack). You do not need to use the ethernet cable.
- _____ 2. Open the Intel Quartus Prime software **Programmer** from the **Tools** menu.
- _____ 3. Select the Intel FPGA Download Cable (may be named **USB-Blaster**) as the configuration cable if it's not already selected.
- a. Click **Hardware Setup**.
 - b. In the Hardware Setup dialog box, use the **Currently selected hardware:** drop-down to select the **USB-Blaster [USB-0]**.
 - c. Click **Close**.

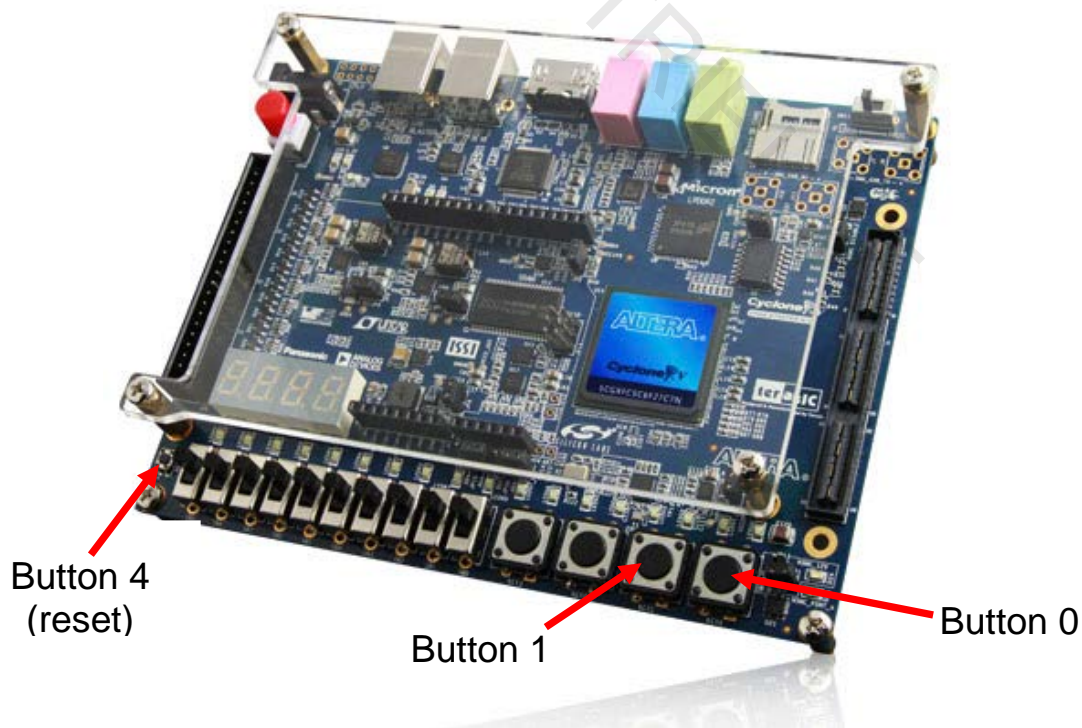
*If the cable does not appear in the **Currently selected hardware:** drop-down, ask your instructor for assistance, but sometimes simply closing the Programmer and reopening it will work.*

- _____ 4. Back in the main Programmer window, click **Auto Detect** to detect the devices on the JTAG chain. If prompted, select the **5CGXFC5C6** Cyclone V device, and click **OK** to select the device.

5. If a file is not already listed in the Programmer, double-click in the **File** column to select the **sm_transfer_system.sof** programming file, which is located in the **Lab1\output_files** subdirectory.

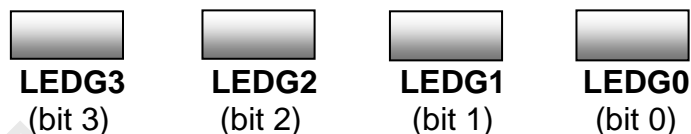


6. Double-click the **sm_transfer_system.sof** file to select it and assign it to the device.
7. Enable the **Program/Configure** option to indicate that the device should be programmed.
8. Click **Start** to configure the FPGA and run the design. Watch the green LEDs and the right-most hex display (HEX0) as you do this.



*The first LED sequence, described below, occurs immediately after device programming is complete. If you miss it, you can reset the design by pushing the CPU_RESET button (**KEY4**) in the lower-left corner of the board.)*

- ____ 9. Using the board push buttons, walk through the sequence of the system design.
- The state machine immediately writes to the (active high) bank of green LEDs with data pattern, **F-F-3-C**. This signals that the design is functioning by flashing the lowest four LEDs twice, the lowest two LEDs once, and then the next higher two LEDs once.



*Each write is actually 8 bits for the 8 green LEDs, but the upper bits are always set to **F**. If you're interested in the setup for the state machine, open the text file named **SM_sequence.txt** in the **Lab1** directory. You'll get to look at the state machine code in more detail later in Lab 3.*

- Press pushbutton #0 (**KEY0**) on the board, the button farthest to the right when the buttons are oriented at the bottom of the board. The state machine flashes LED patterns **1-2-4-8-A-5** sequentially to acknowledge the button press. The HEX0 display matches the green LED pattern.
- The state machine programs the first DMA to transfer data from the source memory to the SRAM. The **A-5** from the LED pattern is to acknowledge the completion of the first DMA setup and transfer.
- Press pushbutton #0 (**KEY0**) on the board one more time. The state machine flashes LEDs 0-1-2-3 (hex **1-2-4-8**) sequentially to acknowledge the button press.
- The state machine then immediately programs the second DMA to transfer data from the SRAM to the LEDs and the onboard memory. This causes the LEDs to flash with the pattern stored in the source memory, **source.hex**. The LEDs should count from 15 down to 1. This pattern can be changed by loading in other **.hex** files.
- When the second DMA transaction is done, press pushbutton #1 (**KEY1**) on the board. The state machine acknowledges the button press by flashing LEDs 4-3-2-1 (hex **8-4-2-1**) this time. It then loops back to step (a) and again flashes the LEDs with the pattern **F-F-3-C**.

*Press **KEY0** to repeat the sequence as per Step (b). You can also flip the slide switches to turn on or off the red LEDs and hex displays.*

Exercise summary

- Created an Intel Quartus Prime project consisting of a Platform Designer system
- Examined hardware system thoroughly in Platform Designer, practicing tool usage in the process
- Generated the system, compiled the project, and ran the design on a development board

END OF EXERCISE 1

DO NOT DISTRIBUTE