System Console

Exercise Manual

Software Requirements:

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

Hardware Requirements:

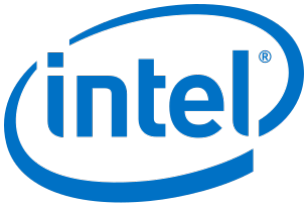
Terasic C5G Cyclone® V GX Starter Kit:

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 exercise:

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



Exercise

Testing a System with System Console

Objectives:

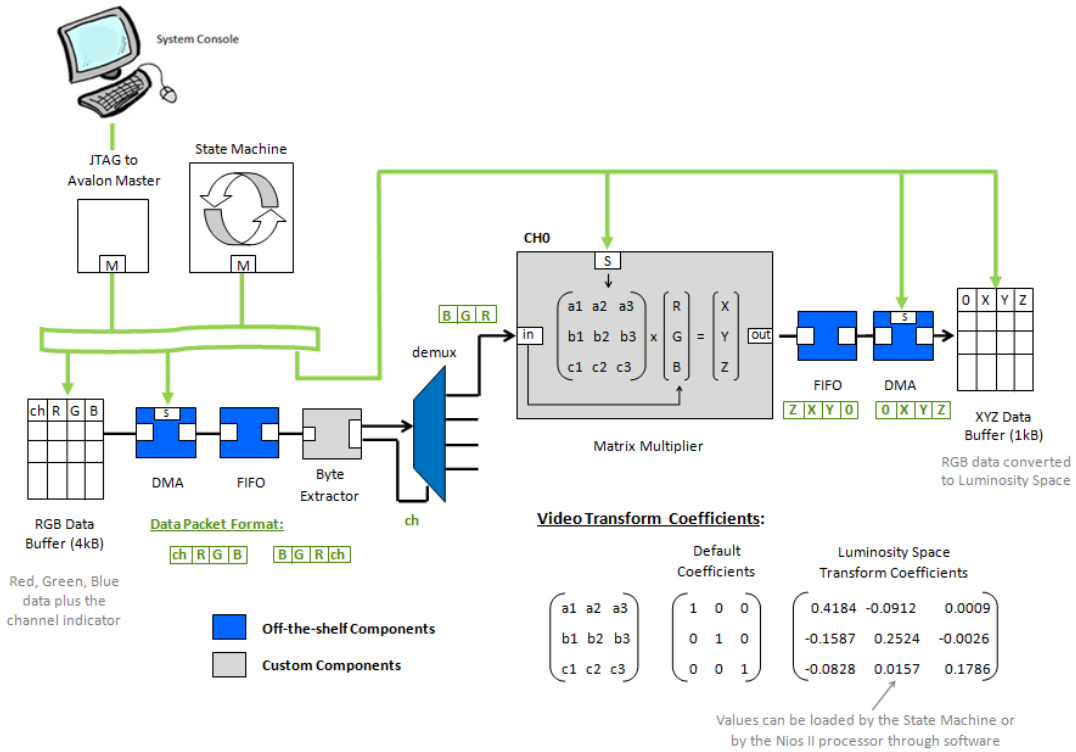
- *Add a JTAG-to-Avalon Master Bridge for communication with the System Console*
- *Use the System Console to verify system clock and reset*
- *Perform simple master read and write operations*
- *Use Tcl scripting and a toolkit to perform complicated tasks from System Console*

Introduction:

In this exercise, you will use the System Console to control a system. First you will use Platform Designer to add the JTAG-to-Avalon® master bridge component to the system. The JTAG-to-Avalon master bridge can communicate with System Console and act as a master in the system. You will then use the System Console to check the system reset and clock signals. After that, you will perform simple master reads and writes. Finally, you will use Tcl scripting and a toolkit GUI to control the system from the System Console.

These lab instructions are written for use in the Windows* operating system, but all tool steps should work identically if you are working in Linux.

Platform Designer system



Step 1: Add the JTAG to Avalon Master Bridge

____ 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 **System_Console**. Place the unzipped folder (**System_Console**) wherever you'd like. The path to the **System_Console** folder contents 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 17.1.0.590 Standard Edition → Quartus Prime Standard Edition 17.1.0.590 → Quartus (Quartus Prime 17.1)**).

____ 3. From the **File** menu, select **Open Project**.

____ 4. Open the following project file:

***<lab install directory>*/PD_Sys_Console_Lab.qpf**

____ 5. Launch Platform Designer by selecting it from the **Tools** menu or by clicking the Platform Designer icon  in the toolbar.

____ 6. Open the system named **system.qsys**.

____ 7. From the **Basic Functions → Bridges and Adaptors → Memory Mapped** folder in the IP Catalog, double-click **JTAG to Avalon Master Bridge** to instantiate the component.

____ 8. Click **Finish** to accept the default parameters and add the component to the system.

You can safely ignore the errors that appear. They will be fixed in the next steps.

____ 9. Right-click and **Rename** the component to **J2A_master**.

____ 10. Connect the **clk** interface to **sys_clk** (**pll.outclk0**).

____ 11. Make connections from the **master** interface of the **J2A_master** to the following slave interfaces using any technique you'd like (clicking dots in the **Connections** column, right-clicking the interface and selecting from the **Connections** sub-menu, the **Connections** tab, etc.).

RGB_DATA.s1
RGB_DMA.control_port_slave
CH0_TRANSFORM.s0
CH0_DMA.control_port_slave
CH0_BUFFER.s1
led_out.debug

*By connecting the above interfaces to the **J2A_master**, you're giving System Console the ability to control data flow through the system as well as to read from and write to the RGB and XYZ data buffers. These connections also allow System Console to edit the matrix multiplier coefficients and control the LEDs.*

____ 12. Go to the Platform Designer **System** menu and choose **Create Global Reset Network** to connect both the **clk_reset** input and **master_reset** output interfaces to the global reset.

By connecting the two reset interfaces, the System Console will have the ability to sample and issue reset requests.

____ 13. From the **Generate** menu, select **Generate HDL**.

____ 14. Set **Create HDL design files for synthesis** to Verilog.

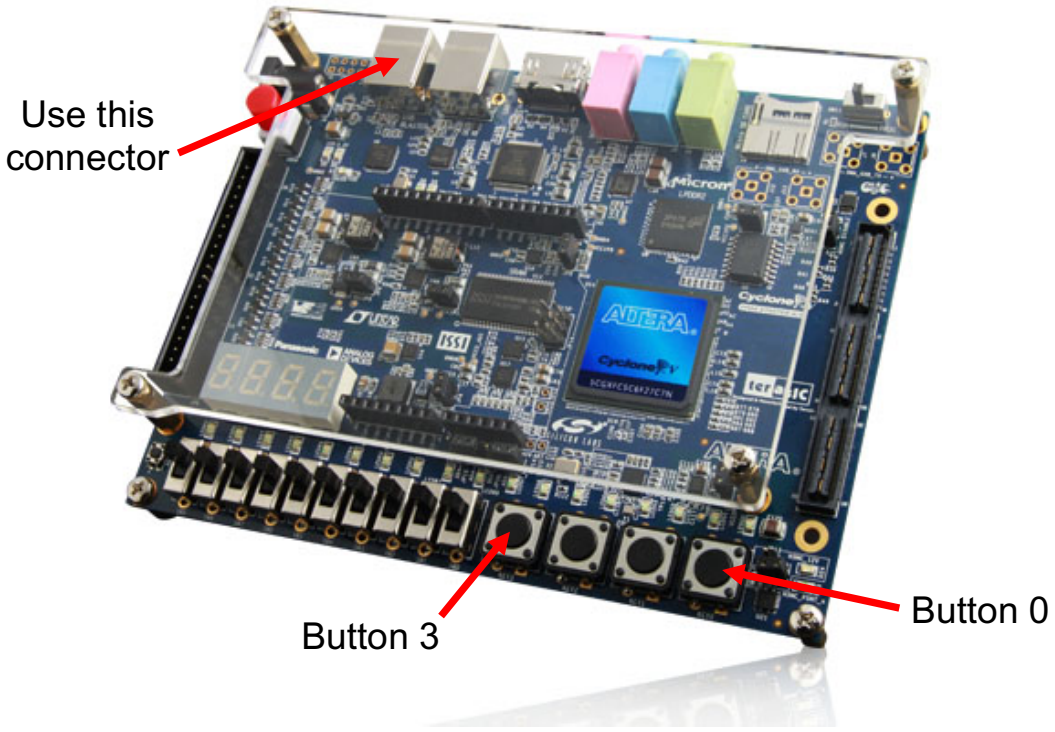
____ 15. Set **Create simulation model** to **None**.

____ 16. Click **Generate**, saving changes if necessary. All warnings can be safely ignored. Click **Close** when complete.

____ 17. Once the system generates, return to the Intel Quartus Prime software and compile the project (**Processing** menu → **Start Compilation**).

Depending on the speed of your computer, the compilation could take up to about 5-10 minutes.

____ 18. 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).



Step 2: Use System Console to perform board bring-up

With the design fully compiled, you can now program the FPGA on the development kit. To prepare for using a graphical toolkit later, you first need to "register" the toolkit to use it with System Console.

____ 1. In Windows Explorer, go up to the ***<lab install directory>***, and copy the file there named **PD_Sys_Console_Lab_toolkit.toolkit**.

____ 2. Go to your user directory (**C:\Users*<username>***) and create a new folder there named **system_console** if you don't have one already.

____ 3. In this new folder, create another new folder named **toolkits**.

____ 4. Paste the **PD_Sys_Console_Lab_toolkit.toolkit** file into this new **toolkits** directory.

____ 5. Open the **.toolkit** file in a text editor.

____ 6. Replace the text on line 3 (`<path to:>`) with the path to the ***<lab install directory>***.

*This file should be pointing to **PD_System_Console_toolkit.tcl**, the file you'll be using later to run a toolkit. So line 3 should have a path that looks like*

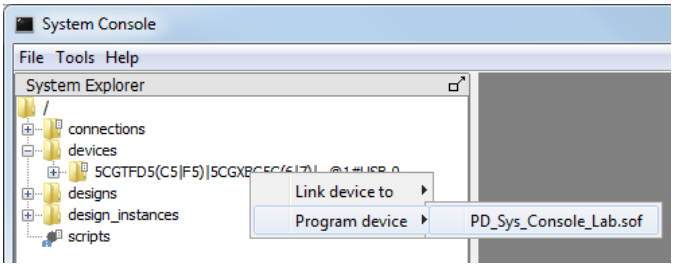
```
C:\System_Console\PD_System_Console_toolkit.tcl
```

____ 7. Back in Platform Designer, from the **Tools** menu, select **System Console**.

____ 8. Verify the System Console connection with the board from the **System Explorer** pane by expanding the **devices** section. If you don't see the **devices** folder, select **Refresh Connections** from the **Tools** menu.

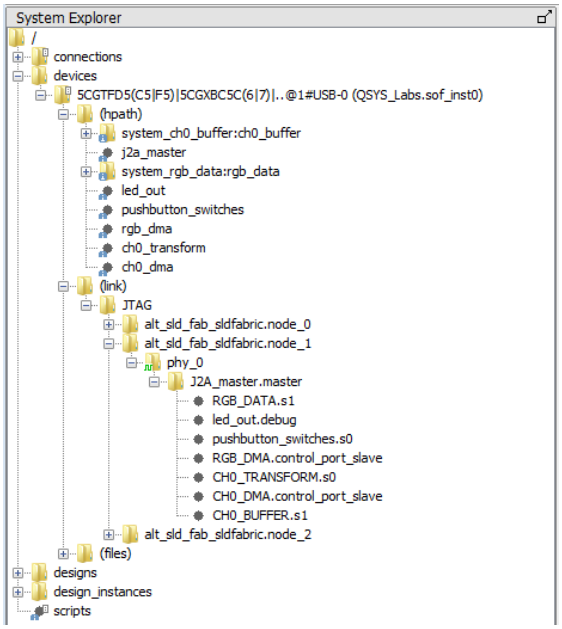
You should see the Cyclone V component on the board.

____ 9. Program the component by right-clicking the 5CGXBC5C component and selecting **Program device**. Select the **PD_Sys_Console_Lab.sof** image. Programming is complete when the status bar in the lower righthand corner of the System Console stops scrolling.



____ 10. Expand the component in the **devices** section to see the components in the system that are connected to the Avalon bus in the **(hpath)** section or to the JTAG bus in the **(link)** section.

*Notice the connections in the **J2A_master** component you made in Platform Designer. These connections and names are discovered automatically when System Console links the project as can be seen in the Messages window ("**Auto linking…**").*



____ 11. In the **Tcl Console**, create a Tcl variable to store the name of the **jtag_debug** service path.

You can see the command to perform this in the training or in the following screen shot. Spaces before and after brackets in Tcl are required.

```
% set jtag_debug_path [lindex [get_service_paths jtag_debug] 0]
/devices/5CGTFD5(C5|F5)|5CGXBC5C(6|7)|..@1#USB-0/(link)/JTAG/alt_sld_fab_sldfabric.node_1/phy_0
```

____ 12. Verify that the clock is toggling by issuing the following commands:

a. Use the `jtag_debug_sense_clock` command which will return 1 if the clock has ever toggled (see below for full command).

b. Sample the clock signal by using the `jtag_debug_sample_clock` command multiple times (see below for full command).

Verify that after a few different samplings of the clock you see both 1's and 0's. The sense clock command should return 1 as long as the clock has ever toggled.


```
% jtag_debug_sample_clock $jtag_debug_path
1

% jtag_debug_sample_clock $jtag_debug_path
0

% jtag_debug_sample_clock $jtag_debug_path
0

% jtag_debug_sense_clock $jtag_debug_path
1
```

Note: Pressing up arrow will bring up the previous System Console command.

____ 13. Use the `jtag_debug_sample_reset` command to sample the value of the reset signal and verify that the reset is released.

The reset signal is active low so the result of this command should be 1, denoting the reset is released.

```
% jtag_debug_sample_reset $jtag_debug_path
1
```

Step 3: Perform master reads and writes

____ 1. Use the **Tcl Console** to get access to the **master** service:

a. Create a Tcl variable to store the name of the master service path.

b. Create another variable and set it to the claimed path from the master service path.

```
% get_service_paths master
/devices/5CGTFD5(C5|F5)|5CGXBC5C(6|7)|..@1#USB-0/(link)/JTAG/alt_sld_fab_sldfabric.node_1/phy_0/J2A_master.master
% set master_path [lindex [get_service_paths master] 0]
/devices/5CGTFD5(C5|F5)|5CGXBC5C(6|7)|..@1#USB-0/(link)/JTAG/alt_sld_fab_sldfabric.node_1/phy_0/J2A_master.master
% set m_path [claim_service master $master_path ""]
/channels/local/(lib)/master_1
```

____ 2. Turn on all eight green LEDs using one of the `master_write` commands. The **led_out** component is located at address **0x1030**, as defined in Platform Designer, and the green LEDs use the lowest 8 bits at that location (the red LEDs can normally be accessed with higher bits at that location, but they are not connected in this design). The LEDs are active high.

```
% master_write_32 $m_path 0x1030 0xff
```


*In this scenario, it's best to use master_write_32 because the slave port on the **led_out** component is 32 bits. The **status_leds** (**greenled** conduit) on the **led_out** component is eight bits wide, so you could've used `master_write_8`, `master_write_16`, or `master_write_memory` with the same result. However accessing the LEDs using one of these commands may be less efficient.*

____ 3. Experiment by writing other numbers to the green status LEDs and observe the results on the development board.

____ 4. Read from the **RGB_DATA** memory. The **RGB_DATA** on-chip memory is located at addresses **0x0000** to **0x0fff**, as defined in the Platform Designer system. Practice using the different read commands and see the differences in the results.

```
% master_read_32 $m_path 0x4 4
0x00030405 0x00060708 0x00090a0b 0x000c0d0e
% master_read_16 $m_path 0x4 4
0x0405 0x0003 0x0708 0x0006
% master_read_memory  $m_path 0x4 4
0x05 0x04 0x03 0x00
```

____ 5. Close the master service.

```
% close_service master $m_path
```

Step 4: Complete the toolkit script and run the transform system from a System Console toolkit

____ 1. Back in the Intel Quartus Prime software, open **PD_System_Console_toolkit.tcl**, found in the project directory. You may need to change the file type to **Script Files** to see the file.

Examine the script. The procedures in the script mimic what gets performed by the state machine to control the system. Much of the script is the toolkit commands that setup the GUI.

____ 2. Make your first edit to the Tcl script. Go to line **199**.

*Look for the comment "**Step 4, #2**".*

____ 3. Create a **transform** button for the GUI. The **transform** button belongs to a grouping of other buttons called **buttons**, and it should call the procedure **transform** when clicked. Three lines of code are required.

*The code for the "Load Identity Coefficients" and "Load Luminosity Space Transform Coefficients" buttons has already been inserted for you in the few lines before this point. Follow the same format to create the Transform button. (If you get stuck, take a look at the solutions file provided in the **Solutions** folder for some hints.)*

____ 4. Go up to line **153**, and enter the commands to load the matrix coefficients into the transform. Use the **master_write_32** command and **$claim_path** to access the master service. The coefficient values themselves can be referenced in the code by **$a1**, **$a2**, etc. The base address of the **transform** component's slave interface is **0x2000**, which is where **a1** resides, so this will be the address you will write to first. Each of the other coefficients' base addresses increases by **4** from there. Nine lines of code are required for the 9 coefficients.

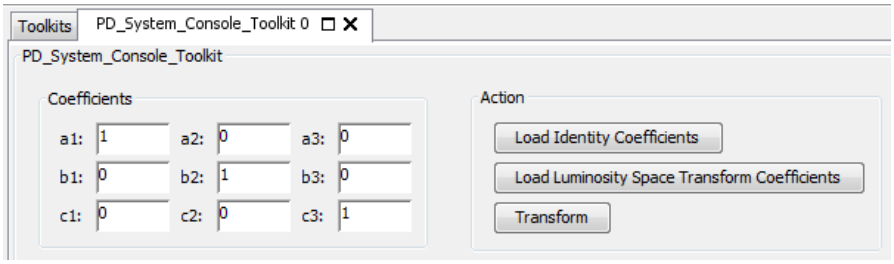
*Look for the comment "**Step 4, #4**". Again, you may refer to the solution file for hints.*

____ 5. Save the file.

____ 6. From the **Tools** menu in System Console, select **PD_System_Console_Toolkit**.

*If you don't see the toolkit listed in the **Tools** menu, double-check the **.toolkit** file you copied, pasted, and edited earlier. Close and reopen System Console if you needed to fix anything.*

*If you encounter errors or the toolkit does not look like the screenshot below, debug your Tcl file, close the toolkit's tab in the System Console, and reopen the toolkit from the **Tools** menu.*



____ 7. Back in the Intel Quartus Prime software, from the **Tools** menu, select **In-System Memory Content Editor**.

*You'll use this tool to monitor the contents of two on-chip memories in the design. The **RGBB** memory stores RGB source data for processing through the datapath, while **CH0B** stores the output transformed data.*

____ 8. Select **USB-Blaster [USB-0]** from the **Hardware** list in the **JTAG Chain Configuration** section of the tool.

*Once you've selected the hardware, make sure you can see the device in the **Device** list and the two memory instances in the **Instance Manager**. If you don't see them, click **Scan Chain**.*

____ 9. Select both the **RGBB** and the **CH0B** instances and click  to have **both** instances continuously read memory contents. Select an instance to see its contents.

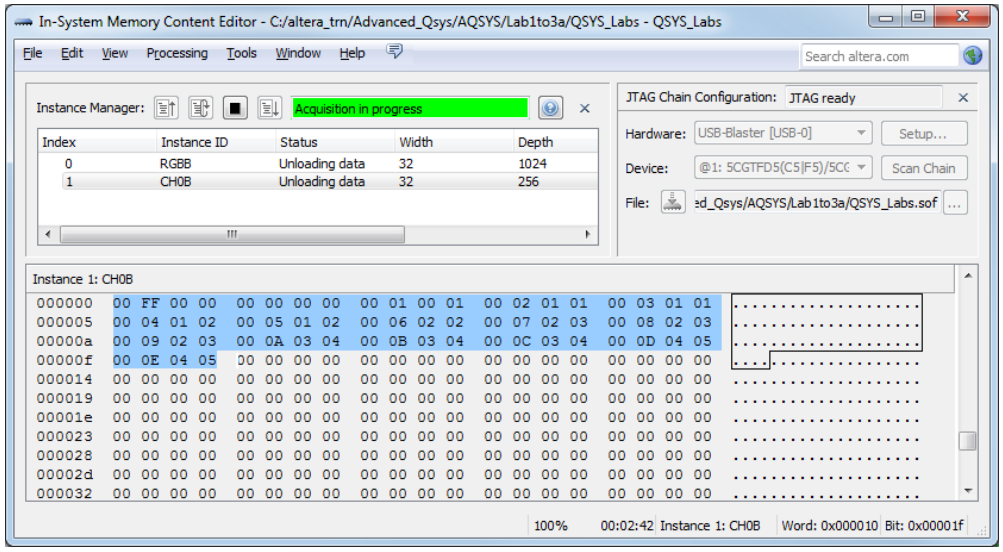
*The **RGBB** memory should contain incrementing data, while **CH0B** should be all 0's.*

____ 10. Go to System Console, and in the toolkit, enter in your own coefficients or choose one of the two preset sets of coefficients.

____ 11. Run the transform by clicking **Transform**.

____ 12. Re-check the contents of system memory. Go back to the In-System Memory Content Editor and reexamine the Channel 0 buffer.

If you used the Luminosity Coefficients you should see the following results:



You have successfully controlled your system from the System Console GUI!

____ 13. Experiment with the System Consule graphical UI that you have created. Try loading in other coefficients, performing the transform, and checking the results in the In-System Memory Content Editor.

*Note that the **Transform** button always overwrites the channel 0 buffer data because it always writes to the same location in the buffer.*

END OF EXERCISE

