

Interacting with FPGA Designs using System Console

PDF created: December 13, 2017
Validated using tools release: 17.0.0

Contents

Overview	1
Prerequisites	1
Preparing the DE10-Nano Board	2
Getting Started with System Console	4

Overview

This tutorial demonstrates how to use the System Console debugging tool to program a compiled FPGA design into an FPGA device, then access the hardware modules instantiated in that FPGA design. This System Console tutorial is based on the FPGA design created in the “My First Qsys System” tutorial. In that tutorial you create this Qsys system:

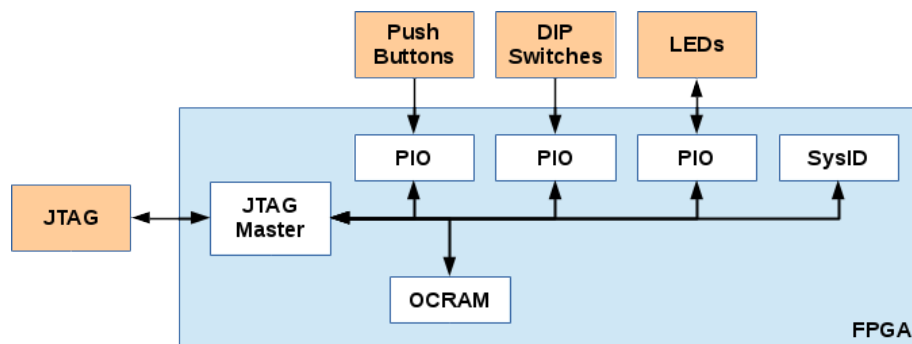


Figure 1: Qsys System Block Diagram

In this tutorial we will demonstrate how you can interact with the Qsys system through a JTAG cable connected to the FPGA that can send read and write transactions through the JTAG master in the Qsys system to interact with the slave peripherals it is connected to.

Prerequisites

The following are required:

- Windows* or Linux* development host PC
- Installed Intel® Quartus® Prime Software Suite. Either the Lite or Standard Edition, but not the Pro Edition.
- Completed Intel Quartus software project from “My First Qsys System”
 - Either follow the tutorial steps presented [here](#).
 - Or, you can download an archive of the required contents from that tutorial to your local file system [here](#).
- Terasic DE10-Nano board
- Power supply for Terasic DE10-Nano
- Mini USB cable to connect Terasic DE10-Nano USB Blaster II port with PC

Intel, and Quartus are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
Other names and brands may be claimed as the property of others.

Preparing the Terasic DE10-Nano Board

This section describes how to prepare the Terasic DE10-Nano board for use in this tutorial.

- Step 1.** Remove the microSD* card from your Terasic DE10-Nano board and store it in a safe place. During this tutorial you will not need the microSD card at all.
- Step 2.** Plug the mini USB cable into the USB-Blaster II connector of the Terasic DE10-Nano board. Then plug the other end of the USB cable to your host PC.

Image used with permission from Terasic Technologies Inc.

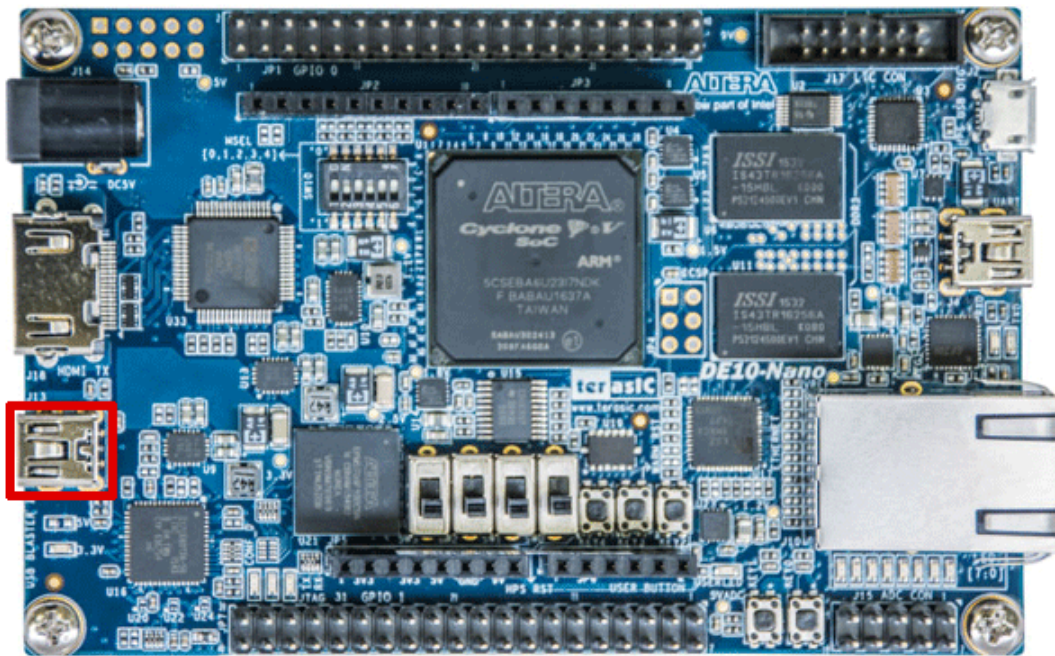


Figure 2: USB-Blaster II Connector on Terasic DE10-Nano Board

Step 3. Power the Terasic DE10-Nano board by inserting the power supply cord into the power connector on the Terasic DE10-Nano board.

Image used with permission from Terasic Technologies Inc.

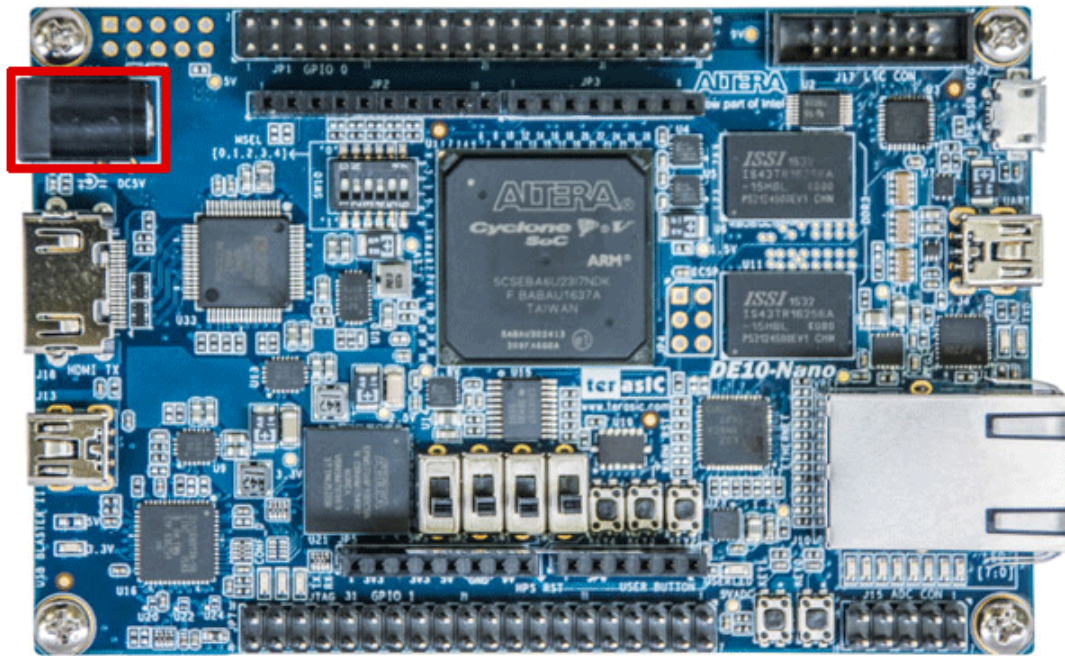


Figure 3: Power Connector on Terasic DE10-Nano Board

Getting Started with System Console

This section describes how to use System Console to download the FPGA design into the FPGA device, and then interact with the JTAG master inside the Qsys system to read and write the slave peripherals that it is attached to. It also demonstrates JTAG link debug and Qsys system health observability provided by System Console.

Step 1. With the **blink** project from the previous tutorial open in the Intel Quartus software tool, start by launching System Console from the Tools->System Debugging Tools->System Console menu.

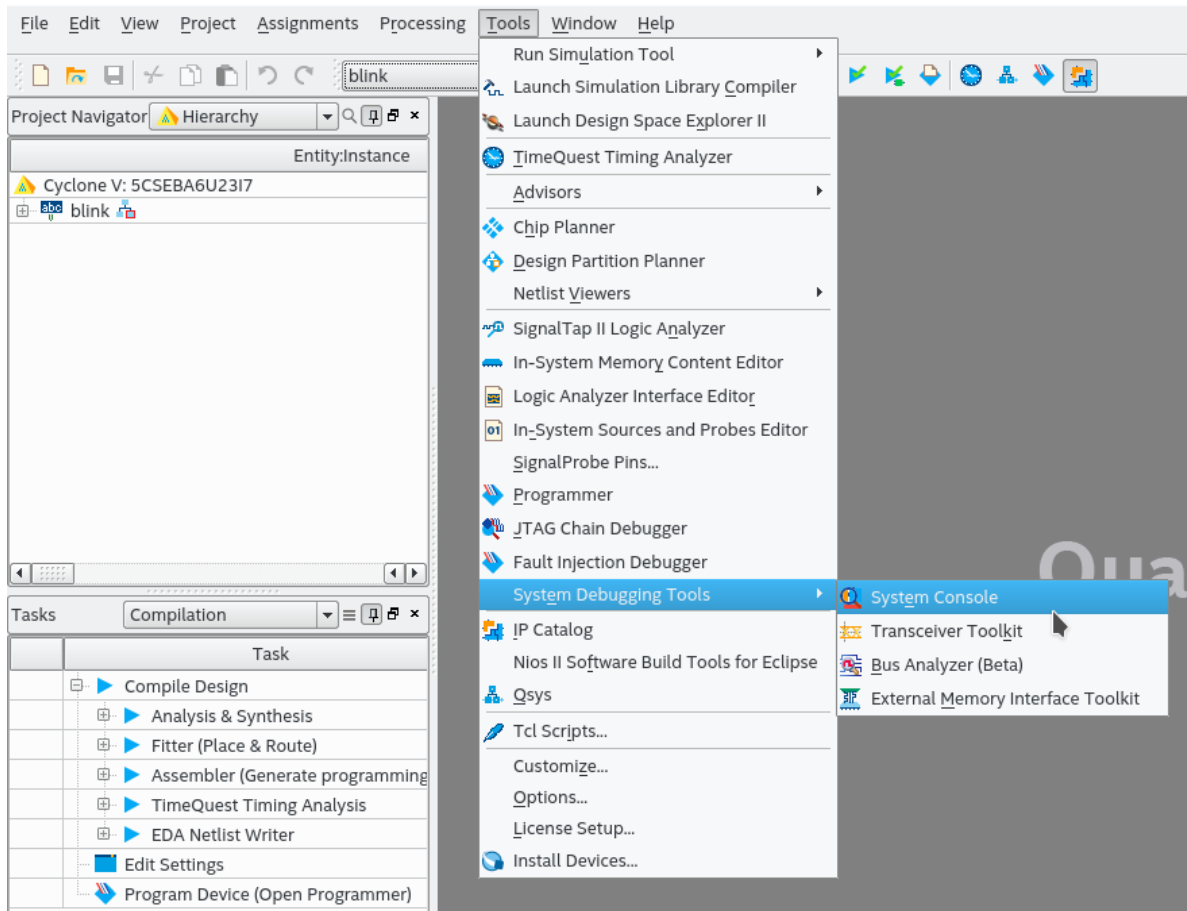
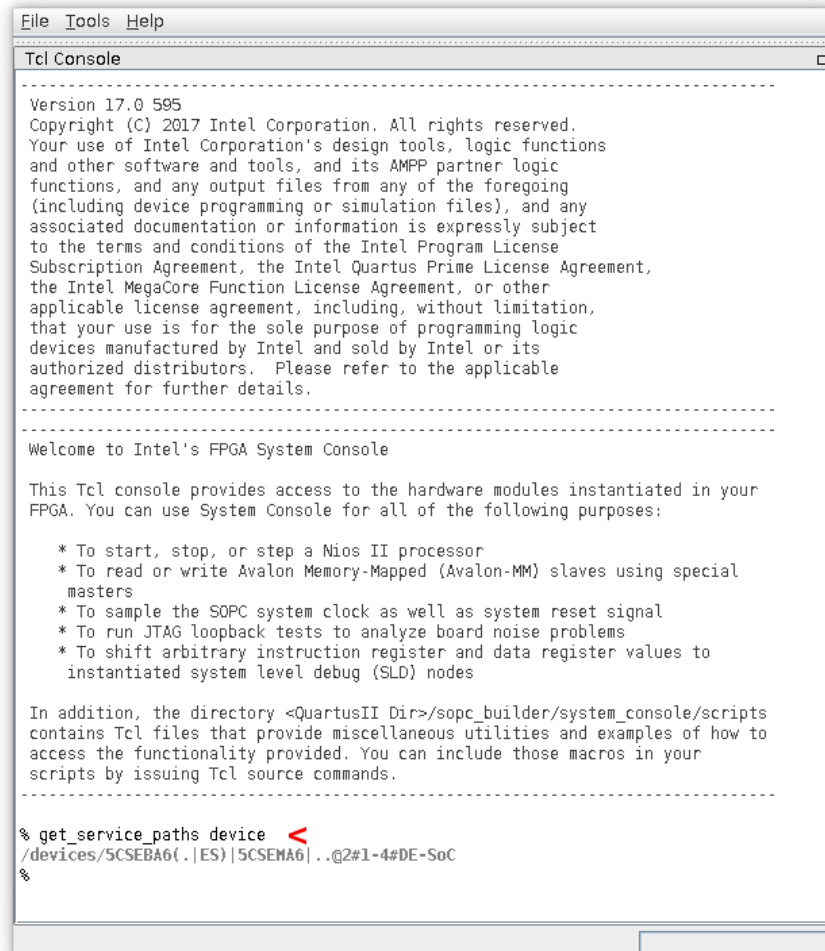


Figure 4: System Console Menu

Step 2. The System Console tool launches and looks for available devices on available JTAG cables. To list the available devices, type the command **get_service_paths device** at the **%** prompt in the **TCL Console** pane in the bottom right corner of the System Console window.

Note: you can double click on the title bar of the **TCL Console** pane to expand it to the full window size. Double click it again to reduce the pane to the original size.



```
File Tools Help
Tcl Console
-----
Version 17.0 595
Copyright (C) 2017 Intel Corporation. All rights reserved.
Your use of Intel Corporation's design tools, logic functions
and other software and tools, and its AMPP partner logic
functions, and any output files from any of the foregoing
(including device programming or simulation files), and any
associated documentation or information is expressly subject
to the terms and conditions of the Intel Program License
Subscription Agreement, the Intel Quartus Prime License Agreement,
the Intel MegaCore Function License Agreement, or other
applicable license agreement, including, without limitation,
that your use is for the sole purpose of programming logic
devices manufactured by Intel and sold by Intel or its
authorized distributors. Please refer to the applicable
agreement for further details.
-----
Welcome to Intel's FPGA System Console

This Tcl console provides access to the hardware modules instantiated in your
FPGA. You can use System Console for all of the following purposes:

* To start, stop, or step a Nios II processor
* To read or write Avalon Memory-Mapped (Avalon-MM) slaves using special
  masters
* To sample the SOPC system clock as well as system reset signal
* To run JTAG loopback tests to analyze board noise problems
* To shift arbitrary instruction register and data register values to
  instantiated system level debug (SLD) nodes

In addition, the directory <QuartusII Dir>/sopc_builder/system_console/scripts
contains Tcl files that provide miscellaneous utilities and examples of how to
access the functionality provided. You can include those macros in your
scripts by issuing Tcl source commands.
-----
% get_service_paths device <
/devices/5CSEBA6(.|ES)|5CSENA6|..@2#1-4#DE-SoC
%
```

Figure 5: System Console's TCL Console

You can also use tab completion to complete commands as you type them and even show you what possible commands can complete from the prefix you type. For instance if we type **get** on the command line and then press **TAB**, we see a pop-up list of all commands that begin with **get**, and we can use the up and down arrows on the keyboard to scroll the list up and down to select one of the commands. Press **Enter** to select a command from the pop-up menu.

```
In addition, the directory <QuartusII Dir>/sopc
contains Tcl files that provide miscellaneous t
access the functionality provided. You can incl
scripts by issuing Tcl source commands.
-----
% get_service_paths device
/devices/5CSEBA6(.|ES)|5CSEMA6|..@2#1-4#DE-SoC
% get|
```

(a) Pre-tab Command Line

```
In addition, the directory <QuartusII Dir>/sopc
contains Tcl files that provide miscellaneous t
aget_claimed_serviceslity provided. You can incl
sget_quartus_ini      Tcl source commands.
-----
--get_service_paths
--get_service_types
% get_services_to_add device
/dget_version         |5CSEMA6|..@2#1-4#DE-SoC
% get|s
```

(b) Post-tab Popup Completion Menu

Figure 6: Tab Completion Example

The TCL console history buffer can also be scrolled backward and forward using the up and down arrows on the keyboard, so you can easily recall previously executed command lines.

All of the commands generally provide help if you pass them into the **help** command, like this:

```
% help get_service_paths
Returns a list of paths to nodes that implement the requested service type.
Service paths can be filtered based on a number of different optional
parameters.
    Arguments:
        ?-type <component-type>?
        ?-hpath <hpath-prefix>?
        ?-device <device-path>?
        <service-type>

Examples:
To list all paths to nodes that implement a supported service type:
    get_service_paths <service-type>
To list all paths to nodes that implement the requested service type under a
specific hpath:
    get_service_paths -hpath <hpath-prefix> <service-type>
To list all paths to nodes attached to a specific component type that implement
the requested service:
    get_service_paths -type <component-type> <service-type>
To list all paths to nodes on a specific device that implement the requested
service type:
    get_service_paths -device <device-path> <service-type>
```

Step 3. Program the FPGA device with the compiled design using the following commands:

```
# get the service path to the device, assume there is only one device attached
% set d_path [get_service_paths device]
/devices/5CSEBA6(.|ES)|5CSEMA6|..@2#1-4#DE-SoC

# program the device with the SOF file
% device_download_sof $d_path output_files/blink.sof
```

You should see the amber **conf_done** LED illuminate once the commands above complete.

Step 4. Exercise the JTAG DEBUG service path provided by the JTAG master component.

Step 4a. Locate the JTAG DEBUG service path of the JTAG master.

```
% set jbg_path [get_service_paths jtag_debug]
/devices/5CSEBA6(.|ES)|5CSEMA6|..@2#1-4#DE-SoC/(link)/JTAG/
alt_sld_fab_sldfabric.node_0/phy_0
```

Step 4b. Exercise the JTAG interface with the loop back mechanism. This checks the physical interface on the board and into the JTAG TAP controller pins of the FPGA device.

```
% jtag_debug_loop $jbg_path [list 1 2 4 8 15 16]
0x01 0x02 0x04 0x08 0x0f 0x10
```

Step 4c. Sense the clock connected to the JTAG master component in the Qsys system. This senses the clock provided to this JTAG master component inside the Qsys system.

```
% jtag_debug_sense_clock $jbg_path
1
```

Step 4d. Sample the clock connected to the JTAG master component. Repeatedly sample the clock until you catch it in both the high and low state. Again this is clock provided inside the Qsys system.

```
% jtag_debug_sample_clock $jbg_path
0
% jtag_debug_sample_clock $jbg_path
0
% jtag_debug_sample_clock $jbg_path
1
% jtag_debug_sample_clock $jbg_path
1
% jtag_debug_sample_clock $jbg_path
0
```

Step 4e. Sample the reset connected to the JTAG master component inside the Qsys system. The KEY0 push button on the DE10-Nano is connected to the Qsys system reset input. Sample the reset, press and hold the KEY0 push button and resample the reset, then release the KEY0 push button and resample the reset.

```
# first sample with KEY0 not pressed
% jtag_debug_sample_reset $jbg_path
1
# second sample with KEY0 pressed
% jtag_debug_sample_reset $jbg_path
0
# third sample with KEY0 not pressed
% jtag_debug_sample_reset $jbg_path
1
```

Step 5. Next we will exercise the master service path provided by the JTAG master. This will require us to recall the address map produced in the previous “MyFirstQsys” tutorial. Remember that we used the **sopc-create-header-files** in that tutorial to create the **master_0.h** header file, and then we dumped all of the base address macros from that file. To refresh your memory, here is what those addresses are:

```
#define OGRAM_64K_BASE 0x0
#define LED_PIO_BASE 0x10000
#define BUTTON_PIO_BASE 0x10010
#define SWITCH_PIO_BASE 0x10020
#define SYSTEM_ID_BASE 0x10030
```

We will set a number of TCL variables to allow us to more easily recall these base addresses in the rest of this tutorial. Execute these commands to setup these variables:

```
% set ocram 0x0
0x0
% set led 0x10000
0x10000
% set button 0x10010
0x10010
% set switch 0x10020
0x10020
% set sysid 0x10030
0x10030
```

Step 5a. Open the master service path.

```
# locate the service path for the JTAG master component
% set m_path [get_service_paths master]
/devices/5CSEBA6(.|ES)|5CSEMA6|..@2#1-4#DE-SoC/(link)/JTAG/
alt_sld_fab_sldfabric.node_0/phy_0/master_0.master

# claim that service path so we can interact with it
% set c_path [claim_service master $m_path ""]
/channels/local/(lib)/master_1
```

Step 5b. Interact with the onchip RAM component.

```
# read the onchip RAM
% master_read_32 $c_path $ocram 1
0x00000000

# write a pattern to the onchip RAM
% master_write_32 $c_path $ocram 0x1234de10

# verify the pattern remains in the onchip RAM
% master_read_32 $c_path $ocram 1
0x1234de10
```

Step 5c. Interact with the LED PIO component.

```
# turn on half the LEDs
% master_write_32 $c_path $led 0x55

# turn off those LEDs and turn on the other half of the LEDs
% master_write_32 $c_path $led 0xaa

# write a loop to toggle LED0 and LED1 every half second for 10 times
% set COUNT 0
% while {$COUNT < 10} {
master_write_32 $c_path $led 0x01
after 500
master_write_32 $c_path $led 0x02
after 500
set COUNT [expr $COUNT + 1]
}

# turn on all of the LEDs
% master_write_32 $c_path $led 0xff
```


- Step 5d.** Exercise the JTAG DEBUG service reset functionality. Now that we have some LEDs illuminated, let's look at one more JTAG DEBUG service provided by the JTAG master component. We have the ability to assert the reset provided into the Qsys system by the JTAG master component. To do that we use the following command:

```
% jtag_debug_reset_system $jb_path
```

After executing that command, the LEDs should all turn off, returned to their reset state. You can trigger the same reset effect by pressing the KEY0 push button like we did before. Turn on some LEDs again and try pressing KEY0 to prove that as well.

- Step 5e.** Interact with the button PIO component. The KEY1 push button is connected to this PIO peripheral.

```
# read the button with KEY1 not pressed
% master_read_32 $c_path $button 1
0x00000001

# read the button with KEY1 pressed
% master_read_32 $c_path $button 1
0x00000000

# read the button with KEY1 not pressed
% master_read_32 $c_path $button 1
0x00000001
```

- Step 5f.** Interact with the switch PIO component. The four slide switches are connected to this PIO peripheral.

```
# all switches in the up position
% master_read_32 $c_path $switch 1
0x0000000f

# far right switch moved down
% master_read_32 $c_path $switch 1
0x0000000e

# next switch to the left moved down
% master_read_32 $c_path $switch 1
0x0000000c

# next switch to the left moved down
% master_read_32 $c_path $switch 1
0x00000008

# last switch moved down
% master_read_32 $c_path $switch 1
0x00000000
```

- Step 5g.** Interact with the system ID component. This component contains two 32-bit words. The first word is the ID value that we set to 0xde10de10 in the previous hardware portion of this tutorial, and the second word that represents the unix second time value when the Qsys system was generated.

```
% master_read_32 $c_path $sysid 2
0xde10de10 0x592b40d8
```

Step 5h. Exercise the default slave peripheral. Here we will demonstrate what happens when we read and write to unmapped address spans.

```
# our peripherals are mapped into the address span from 0x0000_0000 thru
# 0x0001_0038 so we choose an arbitrary high address well above our peripherals
# and we read the 16 bytes of the default slave peripheral which should be
# initialized at device configuration to 0x0000_0000
% master_read_32 $c_path 0x01000000 4
0x00000000 0x00000000 0x00000000 0x00000000

# now lets set those 4 words with a specific incrementing pattern
% master_write_32 $c_path 0x01000000 [list 0x0badf00d 0x1badf00d 0x2badf00d 0x3badf00d]

# now validate that pattern repeats every 4 words
% master_read_32 $c_path 0x01000000 8
0x0badf00d 0x1badf00d 0x2badf00d 0x3badf00d 0x0badf00d 0x1badf00d 0x2badf00d 0x3badf00d

# now lets write to a slave that has no write interface, like the system ID
# peripheral. That write will not be decoded by any slave in the system and
# will be directed into the default slave
% master_write_32 $c_path $sysid 0xfacecafe

# now verify that the first word of the default slave was actually written
% master_read_32 $c_path 0x01000000 4
0xfacecafe 0x1badf00d 0x2badf00d 0x3badf00d
```

Step 6. Close the master service with this command:

```
% close_service master $c_path
```

That's it, you've interacted with the Qsys system using system console. Continue on to the "My First HPS System" tutorial where we demonstrate how to instantiate and integrate the Hard Processor System (HPS) into your Qsys system.