Interacting with FPGA Designs using Linux*

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

Contents

Overview

This tutorial demonstrates how to use various Linux capabilities to interact with the hardware modules instantiated in an FPGA design from the HPS processor. This Linux tutorial is based on the FPGA design created in a prior tutorial "My First HPS System". In that tutorial you create this Qsys system:

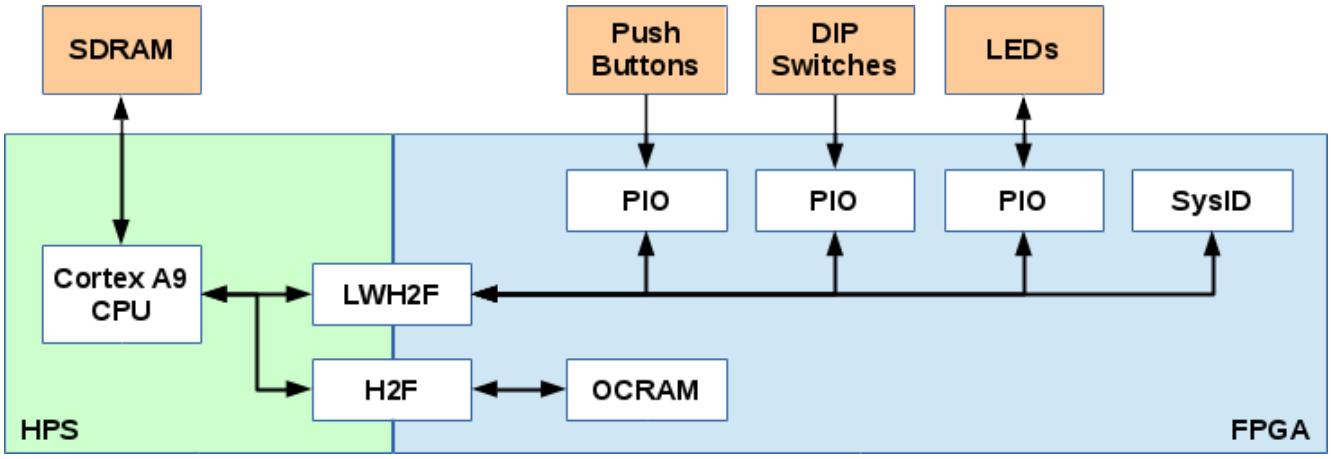

Figure 1: HPS System Block Diagram

In this tutorial we will demonstrate how you can interact with the Qsys system in the FPGA through the HPS-to-FPGA bridges provided on the HPS core, reading and writing to the slave peripherals they are connected to. We demonstrate the *devmem* and *devmem2* programs that can peek and poke at the FPGA peripherals. We also demonstrate how to build a Linux application to interact with those same soft IP peripherals. Finally we demonstrate how to use a number of *sysfs* capabilities provided by existing device drivers for the FPGA based peripherals which we will enable with a *device tree overlay*.

Prerequisites

The following are required:

- Linux development host PC with internet connection and serial terminal emulator
- Completed Intel® Quartus® software project from "My First HPS System"
  - Either follow the tutorial steps presented here.
  - Or, obtain the prebuilt output required from that tutorial here
- Terasic DE-10 Nano SD Card Image Archive: de10-nano-image-Angstrom-v2016.12.socfpga-sdimg.2017.03.31.tgz
  - You can download the archive from this location https://downloadcenter.intel.com/download/26687/Downloads-for-the-Terasic-DE10-Nano-Kit-Featuring-an-Intel-Cyclone-V-FPGA-SoC
- Terasic DE10-Nano board
- MicroSD* card for Terasic DE10-Nano
- MicroSD card adapter for host PC if necessary
- Power supply for Terasic DE10-Nano
- Mini USB cable to connect Terasic DE10-Nano UART console port with PC

Notes:

- The SD card image archive and the **blink** hardware project from the "My First HPS System" tutorial are assumed to be stored in the **$DE10_NANO** folder on the host PC. Make sure you define this environment variable to point at the location where these files are stored. For instance, if you store these files in your **HOME** directory, in a directory called **de10-nano**, you can define the environment variable like this:

```
$ export DE10_NANO=~/de10-nano
```

- The following files from the **blink** hardware project are used by this guide:
  - `blink/output_files/blink.rbf` - for configuring the FPGA from U-Boot
  - `blink/qsys_headers/hps_0_arm_a9_0.h` - for details about IP addresses inside the FPGA fabric
- If you need the prebuilt files from the **blink** hardware project, you can download them to your local file system here. Save the ZIP archive file to this location, **$DE10_NANO/**, the location stored in the environment variable suggested in the note above. After you have stored the archive you can extract the contents like this:

```
$ cd $DE10_NANO
$ unzip blink_for_uboot.zip
```

- Throughout this guide, filenames and parameters that can change based on the latest release filenames or host computer configuration are marked in red. You may need to change those values based on your specific environment and the version that you are using.

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 insert it into your host PC using an appropriate adapter if necessary.

Step 2. Extract the SD card image from the archive by running the following command:

```
$ cd $DE10_NANO
$ tar xf de10-nano-image-Angstrom-v2016.12.socfpga-sdimg.2017.03.31.tgz
```

Step 3. Write the SD card image to the microSD card using the **dd** command:

```
$ sudo dd if=de10-nano-image-Angstrom-v2016.12.socfpga-sdimg of=/dev/sdx bs=16M
```

After the SD image is copied to your microSD card some systems may automatically detect the new partitions that have been copied onto the microSD card and you will not have to manually probe them. If you do need to manually probe the partitions to get them to appear in your environment properly, you can do it like this:

```
$ sudo partprobe /dev/sdx
```

Step 4. Download the Linux examples archive here. Save the ZIP archive file to this location **$DE10_NANO/linux_examples.zip**.

Step 5. Download the source code for the *devmem* utility:

```
# you can do this two different ways, use git to clone the repo and extract the
# desired file, or download the repo archive and extract the desired file

# using git you can do this:
$ git clone https://gfiber.googlesource.com/vendor/opensource/toolbox
$ cp toolbox/devmem.c .
$ rm -rf toolbox

# or you can download the repo archive like this:
$ wget https://gfiber.googlesource.com/vendor/opensource/toolbox/+archive/master.tar.gz
$ tar xf master.tar.gz devmem.c
$ rm master.tar.gz
```

Step 6. Download the source code for the *devmem2* utility:

```
$ wget http://free-electrons.com/pub/mirror/devmem2.c
```

Step 7. Mount the FAT partition of the microSD card on the host PC, and copy the required files to it. Depending on your host PC configuration, the mounting may happen automatically. The FAT partition is partition 1 on the microSD card. The instructions below assume manual mounting is needed:

```
# create a directory in $DE10_NANO to use as a mount point
$ mkdir sdcard

# mount partition 1 of the microSD card, this is the FAT partition
$ sudo mount /dev/sdx1 sdcard

# copy the required files onto the FAT partition
$ sudo cp $DE10_NANO/blink/output_files/blink.rbf sdcard/
$ sudo cp $DE10_NANO/blink/qsys_headers/hps_0_arm_a9_0.h sdcard/
$ sudo cp $DE10_NANO/devmem.c sdcard/
$ sudo cp $DE10_NANO/devmem2.c sdcard/
$ sudo cp $DE10_NANO/linux_examples.zip sdcard/

# unmount the FAT partition
$ sudo umount sdcard
$ sudo sync
```

Step 8. Remove the microSD card from your host PC and insert it into the Terasic DE10-Nano board.

Step 9. Configure the Terasic DE10-Nano dip switch block **SW10** (highlighted in red below) to these positions from left to right as viewed with the board oriented as shown in the image below: `UP-DOWN-UP-DOWN-UP-UP`

Image used with permission from Terasic Technologies Inc.

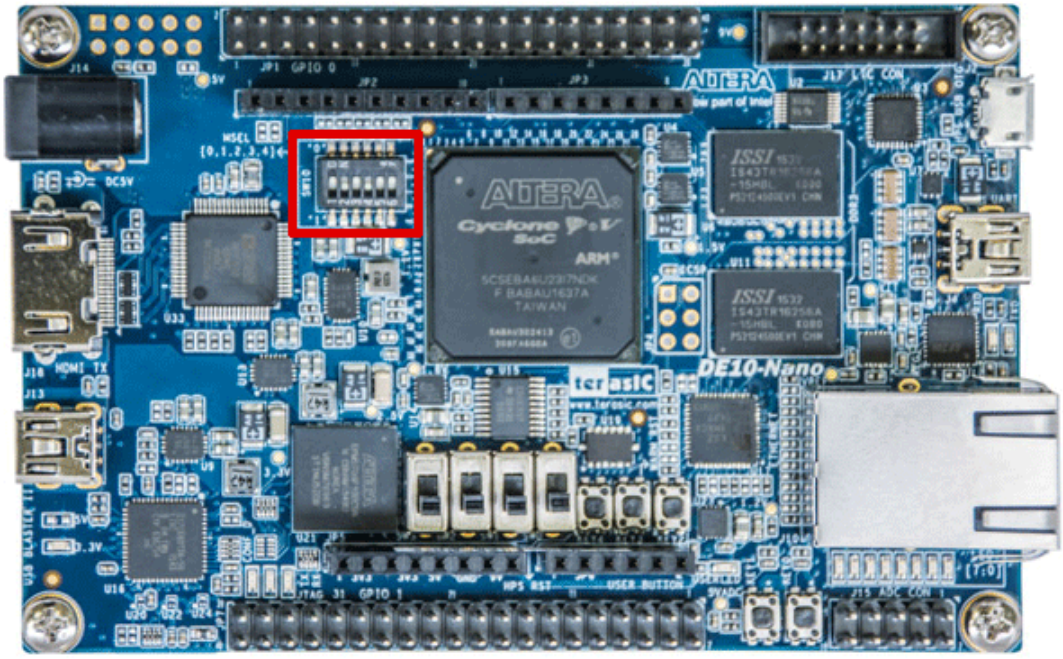

Figure 2: SW10 Configuration

Step 10. Plug a mini USB cable into the Terasic DE10-Nano UART console connector (highlighted in red below), then plug the other end of the USB cable into your host PC.

Image used with permission from Terasic Technologies Inc.

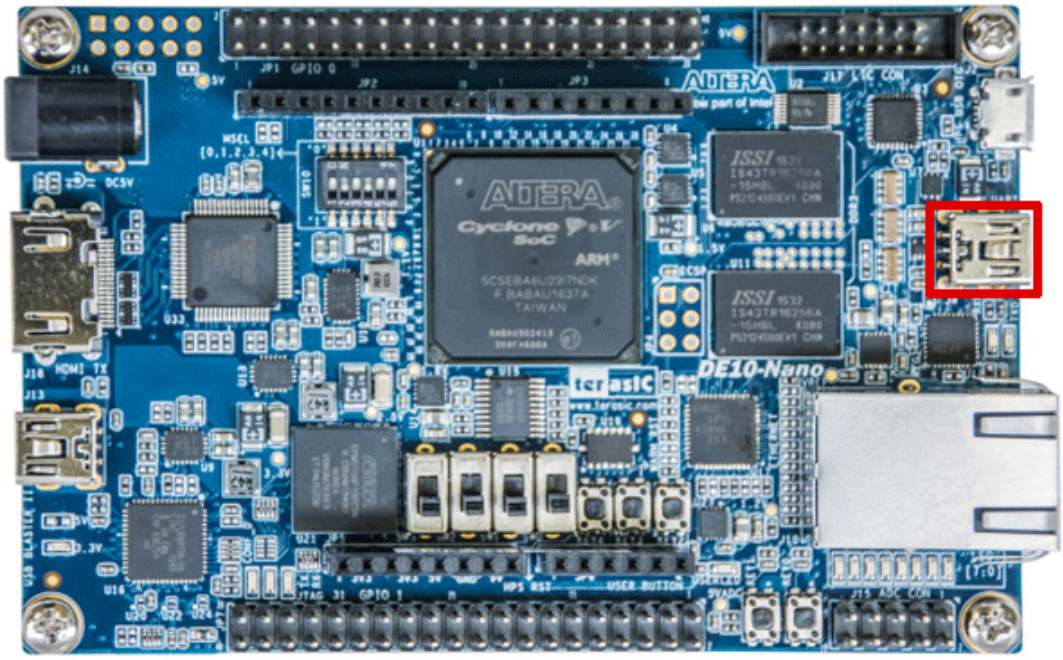

Figure 3: UART Console Connector on Terasic DE10-Nano Board

Step 11. On the host PC, start a serial terminal (for example minicom, or putty) and connect to the serial port corresponding to the Terasic DE10-Nano UART console using the serial communication settings: `115,200-8-N-1`.

Step 12. Power the Terasic DE10-Nano board by inserting the power supply cord into the power connector (highlighted in red below).

Image used with permission from Terasic Technologies Inc.

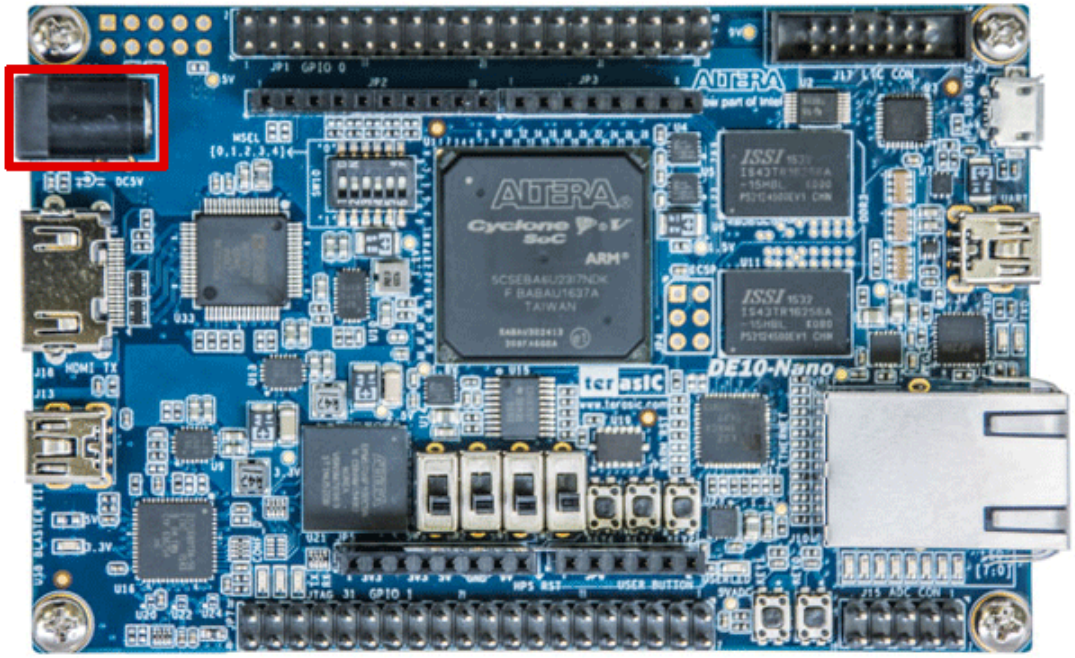

Figure 4: Power Connector on Terasic DE10-Nano Board

Step 13. In the serial terminal, you should observe the boot messages as U-Boot starts and configures the FPGA and boots the Linux kernel. After a few seconds you will see the Angstrom logo scroll into view and a login prompt. You may also see some lingering boot messages related to the CRDA update that the cfg80211 driver is attempting to perform, this will eventually timeout and stop.

```
[   31.907948] cfg80211: Calling CRDA to update world regulatory domain
[   35.067904] cfg80211: Exceeded CRDA call max attempts. Not calling CRDA
```

If your login prompt was overwritten by the lingering boot message output, press return to get a clean login prompt again. Login with the username **root** and empty password (simply press the **enter** key for the password).

```
.---O---.
|       |                  .-.           o o
|   |   |-----.-----.-----.| |   .----..-----.-----.
|       |     | __  |  ---'| '--.|  .-'|     |     |
|   |   |  |  |     |---  ||  --'|  |  |  '  | | | |
'---'---'--'--'--.  |-----''----''--'  '-----'-'-'-'
                -'  |
                '---'

The Angstrom Distribution de10-nano ttyS0

Angstrom v2016.12 - Kernel 4.1.33-ltsi-altera
```

```
de10-nano login: root
Password: <the password is NULL so just press ENTER>
root@de10-nano:~#
```

Step 14. We need to disable some of the default functionality that the standard Terasic DE10-Nano SD card image assumes. Primarily we need to disable all of the functionality that depends on the default FPGA image so that we can load our **blink.rbf** instead. There are five services that get started in the systemd init environment which require the default FPGA image to be available, we will disable those services with the following commands:

```
# configure the linux environment so we can run our custom blink.rbf instead of
# the default RBF image.  The following five services must be disabled:
root@de10-nano:~# systemctl disable de10-nano-fpga-leds.service
root@de10-nano:~# systemctl disable de10-nano-synergy-init.service
root@de10-nano:~# systemctl disable de10-nano-fftsw-init.service
root@de10-nano:~# systemctl disable de10-nano-fpga-init.service
root@de10-nano:~# systemctl disable de10-nano-xfce-init.service
```

If you ever want to enable the Linux environment to run with the default RBF image again, you can simply re-enable those five services like this:

```
# configure the linux environment so we can run the default RBF image.
# The following five services must be enabled:
root@de10-nano:~# systemctl enable de10-nano-fpga-leds.service
root@de10-nano:~# systemctl enable de10-nano-synergy-init.service
root@de10-nano:~# systemctl enable de10-nano-fftsw-init.service
root@de10-nano:~# systemctl enable de10-nano-fpga-init.service
root@de10-nano:~# systemctl enable de10-nano-xfce-init.service
```

Step 15. *Read the next step before completing this step.* After the five services described above are disabled we can reboot the Terasic DE10-Nano board.

```
# reboot the DE10-Nano
root@de10-nano:~# reboot
```

Step 16. In the serial terminal, interrupt the U-Boot autoboot countdown by pressing any key. This will provide access to the U-Boot console. If you are too slow and the Linux kernel begins to boot, you should allow Linux to load, login with the username **root** and empty password, simply press the **enter** key for the password. Then run the Linux command **reboot** which will shutdown the Linux session and reboot through U-Boot. Try to be quicker on the next pass through U-Boot and stop it by pressing any key during the autoboot countdown. Repeat as necessary.

```
U-Boot SPL 2017.03-rc2 (Mar 30 2017 - 19:07:16)
/data/de10-nano/release-build-2017.03.31/build/tmp-angstrom-glibc/work/de10_nano
-angstrom-linux-gnueabi/u-boot-socfpga/v2017.03gitAUTOINCd03450606b-r0/git/dri
vers/ddr/altera/sequencer.c: Preparing to start memory calibration
/data/de10-nano/release-build-2017.03.31/build/tmp-angstrom-glibc/work/de10_nano
-angstrom-linux-gnueabi/u-boot-socfpga/v2017.03gitAUTOINCd03450606b-r0/git/dri
vers/ddr/altera/sequencer.c: CALIBRATION PASSED
/data/de10-nano/release-build-2017.03.31/build/tmp-angstrom-glibc/work/de10_nano
-angstrom-linux-gnueabi/u-boot-socfpga/v2017.03gitAUTOINCd03450606b-r0/git/dri
vers/ddr/altera/sequencer.c: Calibration complete
Trying to boot from MMC1


U-Boot 2017.03-rc2 (Mar 30 2017 - 19:07:16 -0700)
```

```
CPU:   Altera SoCFPGA Platform
FPGA:  Altera Cyclone V, SE/A6 or SX/C6 or ST/D6, version 0x0
BOOT:  SD/MMC Internal Transceiver (3.0V)
       Watchdog enabled
I2C:   ready
DRAM:  1 GiB
MMC:   dwmmc0@ff704000: 0
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Model: Terasic DE10-Nano
Net:
Error: ethernet@ff702000 address not set.
No ethernet found.
Hit any key to stop autoboot:  0
=>
```

Step 17. On the U-Boot console, run the following commands to configure the FPGA with the **blink.rbf** file from the SD card, then load the Linux kernel and device tree, configure the boot arguments to pass into the kernel and boot the kernel:

```
=> setenv fpga_file blink.rbf
=> run fpga_cfg
=> fatload mmc 0:1 ${kernel_addr_r} zImage
=> fatload mmc 0:1 ${fdt_addr} socfpga_cyclone5_de10_nano.dtb
=> setenv bootargs 'console=ttyS0,115200 root=/dev/mmcblk0p2 rootwait'
=> bootz ${kernel_addr_r} - ${fdt_addr}
```

You should see the Linux boot messages fill the serial terminal once the commands above complete.

The Terasic DE10-Nano should now be running Linux with our custom **blink.rbf** image loaded in the FPGA. Proceed to the next section to see how to interact with our FPGA design from the Linux environment.

Prepare the Linux Example Material

In this section we will simply prepare the environment by organizing the materials that we copied onto the FAT partition on the host PC.

Step 1. Boot into the U-Boot console, and configure FPGA with the **blink.rbf** as described in the "Preparing the Terasic DE10-Nano Board" section above.

Step 2. Move the Linux example contents from the FAT partition into the HOME directory of the root user that we logged in as. This requires a few commands to extract the ZIP archive and move the other files into the resulting directory structure, like this:

```
root@de10-nano:~# mv /media/FAT/linux_examples.zip .
root@de10-nano:~# unzip linux_examples.zip
root@de10-nano:~# mv /media/FAT/hps_0_arm_a9_0.h linux_examples/c_examples/
root@de10-nano:~# mv /media/FAT/devmem.c linux_examples/c_examples/
root@de10-nano:~# mv /media/FAT/devmem2.c linux_examples/c_examples/
root@de10-nano:~# find linux_examples -type 'f' | sort
linux_examples/c_examples/build_app.sh
linux_examples/c_examples/build_devmem.sh
```

```
linux_examples/c_examples/build_devmem2.sh
linux_examples/c_examples/devmem.c
linux_examples/c_examples/devmem2.c
linux_examples/c_examples/hps_0_arm_a9_0.h
linux_examples/c_examples/linux_blink_app.c
linux_examples/devicetree_overlay/soc_system.dtbo
linux_examples/devicetree_overlay/soc_system.dtso
linux_examples/sysfs_examples/read_button_pio_sysfs.sh
linux_examples/sysfs_examples/read_switch_pio_sysfs.sh
linux_examples/sysfs_examples/read_system_id_sysfs.sh
linux_examples/sysfs_examples/toggle_leds_sysfs.sh
```

When complete, you should see the file contents as listed in the last command above. These files can also be downloaded from the GitHub* repository if you would like to observe them on your local development host PC, some of them you already have on your host PC.

GitHub file locations:

- linux_examples/c_examples/build_app.sh - Download here or view here.
- linux_examples/c_examples/build_devmem.sh - Download here or view here.
- linux_examples/c_examples/build_devmem2.sh - Download here or view here.
- linux_examples/c_examples/linux_blink_app.c - Download here or view here.
- linux_examples/devicetree_overlay/soc_system.dtso - Download here or view here.
- linux_examples/sysfs_examples/read_button_pio_sysfs.sh - Download here or view here.
- linux_examples/sysfs_examples/read_switch_pio_sysfs.sh - Download here or view here.
- linux_examples/sysfs_examples/read_system_id_sysfs.sh - Download here or view here.
- linux_examples/sysfs_examples/toggle_leds_sysfs.sh - Download here or view here.

Build the *devmem* Utilities to Interact with the FPGA Design

We want to interact with the IP modules inside the FPGA fabric, and in the Linux environment there are a couple of utility programs that allow us to perform simple peek and poke operations into the memory map called *devmem* and *devmem2*. These programs are not built into the default image for the Terasic DE10-Nano SD card so we had you download the source files from the internet and place them on the target so we can build them and use them.

Step 1. Start by changing into the **c_examples** directory of the **linux_examples** directory and execute the build scripts for *devmem* and *devmem2*. You can download the build script files from the GitHub repo using the links above.

```
root@de10-nano:~# cd linux_examples/c_examples/
root@de10-nano:~# ./build_devmem.sh
root@de10-nano:~# ./build_devmem2.sh
```

If you are curious about the build scripts that you just executed or the C source files that they just compiled, feel free to study their contents. You can use any number of utilities on the Terasic DE10-Nano target to view these contents, like **vi**, **less**, or **cat** to name a few.

Step 2. Next, we will interact with the IP modules inside the FPGA fabric using the *devmem* utilities we just built. This will require us to recall the address map produced in the previous tutorial. Remember that we used the **sopc-create-header-files** in that tutorial to create the **hps_0_arm_a9_0.h** header file, and then we dumped all of the base address macros from that file. We had you copy that header file onto the Terasic DE10-Nano target in the **c_examples** directory we are now in. To refresh your memory, dump the relevant FPGA base addresses out of that header file like this:

```
root@de10-nano:~# grep "_BASE" hps_0_arm_a9_0.h | grep -v "HPS_"
#define OCRAM_64K_BASE 0xc0000000
#define LED_PIO_BASE 0xff210000
#define BUTTON_PIO_BASE 0xff210010
#define SWITCH_PIO_BASE 0xff210020
#define SYSTEM_ID_BASE 0xff210030
```

We will set a number of environment variables to allow us to recall these base addresses easier for the rest of this tutorial. Execute these commands to setup these variables:

```
root@de10-nano:~# export OCRAM_64K_BASE=0xc0000000
root@de10-nano:~# export LED_PIO_BASE=0xff210000
root@de10-nano:~# export BUTTON_PIO_BASE=0xff210010
root@de10-nano:~# export SWITCH_PIO_BASE=0xff210020
root@de10-nano:~# export SYSTEM_ID_BASE=0xff210030
```

Step 3. Exercise the IP inside the FPGA fabric with *devmem* and *devmem2*

These are the usage requirements for the *devmem* and *devmem2* utilities:

```
# dump the devmem usage
root@de10-nano:~# ./devmem
usage: devmem <addr> [default:32|16|8] [data]

# dump the devmem2 usage
root@de10-nano:~# ./devmem2

Usage:  ./devmem2 { address } [ type [ data ] ]
        address : memory address to act upon
        type    : access operation type : [b]yte, [h]alfword, [w]ord
        data    : data to be written
```

Both utilities allow you to read or write a memory location but they use slightly different command arguments and they output their results slightly differently as we demonstrate below.

Step 3a. Interact with the onchip RAM component. We will demonstrate *devmem* and *devmem2*.

```
# use the devmem utility first
# read the onchip RAM
root@de10-nano:~# ./devmem $OCRAM_64K_BASE 32
0x00000000

# write a pattern to the onchip RAM
root@de10-nano:~# ./devmem $OCRAM_64K_BASE 32 0x1234de10
0x00000000
0x1234de10

# verify the pattern remains in the onchip RAM
root@de10-nano:~# ./devmem $OCRAM_64K_BASE 32
0x1234de10

# now do the same things with the devmem2 utility
# read the onchip RAM
root@de10-nano:~# ./devmem2 $OCRAM_64K_BASE w
/dev/mem opened.
Memory mapped at address 0xb6fea000.
Value at address 0xC0000000 (0xb6fea000): 0x1234DE10
```

```
# write a pattern to the onchip RAM
root@de10-nano:~# ./devmem2 $OCRAM_64K_BASE w 0x5678de10
/dev/mem opened.
Memory mapped at address 0xb6fe6000.
Value at address 0xC0000000 (0xb6fe6000): 0x1234DE10
Written 0x5678DE10; readback 0x5678DE10

# verify the pattern remains in the onchip RAM
root@de10-nano:~# ./devmem2 $OCRAM_64K_BASE w
/dev/mem opened.
Memory mapped at address 0xb6f7d000.
Value at address 0xC0000000 (0xb6f7d000): 0x5678DE10
```

Step 3b. Interact with the LED PIO component. We will demonstrate *devmem* only.

```
# turn on half the LEDs
root@de10-nano:~# ./devmem $LED_PIO_BASE 32 0x55

# turn off those LEDs and turn on the other half of the LEDs
root@de10-nano:~# ./devmem $LED_PIO_BASE 32 0xaa

# write a loop to toggle LED0 and LED1 every half second for 10 times
root@de10-nano:~# COUNT=0
root@de10-nano:~# while [ $COUNT -lt 10 ]
> do
> ./devmem $LED_PIO_BASE 32 0x01 > /dev/null
> usleep 500000
> ./devmem $LED_PIO_BASE 32 0x02 > /dev/null
> usleep 500000
> ((COUNT++))
> done

# turn on all of the LEDs
root@de10-nano:~# ./devmem $LED_PIO_BASE 32 0xff
```

Step 3c. Exercise the HPS-to-FPGA reset functionality. Now that we have some LEDs illuminated, let's look at how the HPS-to-FPGA reset can be triggered. We have the ability to assert the reset provided into the FPGA by the HPS core, to do that we use the following commands to set and clear the **s2f** bit of the **miscmodrst** register located in the HPS Reset Manager, that is bit 6 of the HPS register at address 0xFFD05020:

```
# assert the H2F reset
root@de10-nano:~# ./devmem 0xFFD05020 32 0x40

# deassert the H2F reset
root@de10-nano:~# ./devmem 0xFFD05020 32 0x00
```

After executing the first command, the LEDs should all turn off, returned to their reset state. Do not forget to release the reset by clearing that bit with the second command. You can trigger the same reset effect by pressing the KEY0 push button. Turn on some LEDs again and try pressing KEY0 to prove that as well.

Please note that resetting the FPGA logic design while software is running on the HPS core can be dangerous for the software environment. If software drivers are interacting with FPGA logic at the time you invoke a reset like this, the hardware transactions can hang which causes the software environment to hang. So resetting part of the hardware system like this should be done with caution.

Step 3d. Interact with the button PIO component. The KEY1 push button is connected to this PIO peripheral.

```
# read the button with KEY1 not pressed
root@de10-nano:~# ./devmem $BUTTON_PIO_BASE 32
0x00000001

# read the button with KEY1 pressed
root@de10-nano:~# ./devmem $BUTTON_PIO_BASE 32
0x00000000

# read the button with KEY1 not pressed
root@de10-nano:~# ./devmem $BUTTON_PIO_BASE 32
0x00000001
```

Step 3e. Interact with the switch PIO component. The four slide switches are connected to this PIO peripheral.

```
# all switches in the up position
root@de10-nano:~# ./devmem $SWITCH_PIO_BASE 32
0x0000000f

# far right switch moved down
root@de10-nano:~# ./devmem $SWITCH_PIO_BASE 32
0x0000000e

# next switch to the left moved down
root@de10-nano:~# ./devmem $SWITCH_PIO_BASE 32
0x0000000c

# next switch to the left moved down
root@de10-nano:~# ./devmem $SWITCH_PIO_BASE 32
0x00000008

# last switch moved down
root@de10-nano:~# ./devmem $SWITCH_PIO_BASE 32
0x00000000
```

Step 3f. Interact with the system ID component. This component contains two 32-bit words, one ID value that we set to the value 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.

```
root@de10-nano:~# ./devmem $SYSTEM_ID_BASE 32
0xde10de10
root@de10-nano:~# ./devmem $(($SYSTEM_ID_BASE + 4)) 32
0x593b4f62
```

Step 3g. 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 HPS address span through the LWHPS bridge
# from 0xFF21_0000 thru 0xFF21_0038 and the HPS bridge from 0xC000_0000 thru
# 0xC000_FFFF, 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

# first create a function macro that we can use to dump the default slave easily
root@de10-nano:~# function dump_default_slave {
```

```
> for INDEX in $(seq 0 4 $((${2} * 4 - 4)))
> do
> printf "0x%08X: " $((${1:?} + INDEX))
> ./devmem $((${1:?} + INDEX)) 32
> done
> }

root@de10-nano:~# dump_default_slave 0xFF211000 4
0xFF211000: 0x00000000
0xFF211004: 0x00000000
0xFF211008: 0x00000000
0xFF21100C: 0x00000000

# now lets set those 4 words with a specific incrementing pattern
root@de10-nano:~# ./devmem 0xFF211000 32 0x0badf00d
root@de10-nano:~# ./devmem 0xFF211004 32 0x1badf00d
root@de10-nano:~# ./devmem 0xFF211008 32 0x2badf00d
root@de10-nano:~# ./devmem 0xFF21100C 32 0x3badf00d

# now validate that pattern repeats every 4 words
root@de10-nano:~# dump_default_slave 0xFF211000 8
0xFF211000: 0x0badf00d
0xFF211004: 0x1badf00d
0xFF211008: 0x2badf00d
0xFF21100C: 0x3badf00d
0xFF211010: 0x0badf00d
0xFF211014: 0x1badf00d
0xFF211018: 0x2badf00d
0xFF21101C: 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
root@de10-nano:~# ./devmem $SYSTEM_ID_BASE 32 0xfacecafe

# now verify that the first word of the default slave was actually written
root@de10-nano:~# dump_default_slave 0xFF211000 4
0xFF211000: 0xfacecafe
0xFF211004: 0x1badf00d
0xFF211008: 0x2badf00d
0xFF21100C: 0x3badf00d

# we have been using an address in the LWHPS bridge address span, but we can see
# the default slave through the HPS bridge as well
root@de10-nano:~# dump_default_slave 0xC0010000 4
0xC0010000: 0xfacecafe
0xC0010004: 0x1badf00d
0xC0010008: 0x2badf00d
0xC001000C: 0x3badf00d
```

That's it, you've interacted with the Qsys system using the *devmem* and *devmem2* utilities in Linux. Continue on to the next section where we demonstrate how to write a Linux program to interact with the Qsys system.

Build a Linux Application to Interact with the FPGA Design

In this section we will demonstrate how to build a Linux application that can interact with the FPGA design. We supplied you with the **linux_blink_application.c** C source file in the Linux examples archive.

```
root@de10-nano:~# cd ~/linux_examples/c_examples/
root@de10-nano:~# ls linux_blink_app.c
linux_blink_app.c
```

The contents of that application are shown below. Please study it and observe how we use the **/dev/mem** driver to **mmap()** the address space in the FPGA that we need to interact with and we include the **hps_0_arm_a9_0.h** header in order to obtain the FPGA peripheral base addresses and other parameters that we require in the application. You can download the C source file from the GitHub repo using the links above.

```
/*
 * Copyright (c) 2017 Intel Corporation
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to
 * deal in the Software without restriction, including without limitation the
 * rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
 * sell copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
 * IN THE SOFTWARE.
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <error.h>
#include <errno.h>
#include <string.h>
#include <sys/mman.h>
#include <termios.h>
#include <sys/utsname.h>

#include "hps_0_arm_a9_0.h"

/* Expected System Environment */
#define SYSFS_FPGA0_STATE_PATH "/sys/class/fpga_manager/fpga0/state"
#define SYSFS_FPGA0_STATE "operating"

#define SYSFS_LWH2F_BRIDGE_NAME_PATH "/sys/class/fpga_bridge/br0/name"
#define SYSFS_LWH2F_BRIDGE_NAME "lwhps2fpga"

#define SYSFS_LWH2F_BRIDGE_STATE_PATH "/sys/class/fpga_bridge/br0/state"
#define SYSFS_LWH2F_BRIDGE_STATE "enabled"

#define SYSFS_H2F_BRIDGE_NAME_PATH "/sys/class/fpga_bridge/br1/name"
#define SYSFS_H2F_BRIDGE_NAME "hps2fpga"
```

```
49
50  #define SYSFS_H2F_BRIDGE_STATE_PATH "/sys/class/fpga_bridge/br1/state"
51  #define SYSFS_H2F_BRIDGE_STATE "enabled"
52
53  #define LED_DELAY_US 250000
54
55  void validate_system_status(void);
56  void demo_sysid(void);
57  void demo_ocram(void);
58  void demo_leds(void);
59  void demo_switches(void);
60  void demo_buttons(void);
61
62  int main(int argc, char * const argv[]) {
63
64          int i;
65          int result;
66          struct termios termios_s;
67          struct termios original_termios;
68          struct utsname uname_s;
69
70          /*
71           * configure stdio as non-canonical with no echo so we can poll for key
72           * input as they occur rather than line by line and prevent echo back
73           * on the input terminal
74           */
75
76          result = tcgetattr(STDIN_FILENO, &termios_s);
77          if(result != 0)
78                  error(1, errno, "tcgetattr");
79
80          original_termios = termios_s;
81          termios_s.c_lflag &= ~ICANON;
82          termios_s.c_lflag &= ~ECHO;
83          result = tcsetattr(STDIN_FILENO, TCSANOW, &termios_s);
84          if(result != 0)
85                  error(1, errno, "tcsetattr");
86
87          /* configure stdin as non-blocking */
88          fcntl(STDIN_FILENO, F_SETFL, fcntl(STDIN_FILENO, F_GETFL) | O_NONBLOCK);
89
90          /* Display welcome message */
91          printf("\n");
92          printf("Blink application started.\n");
93          printf("\n");
94
95          /* Display command line parameters */
96          printf("Number of command line parameters: %d\n", argc);
97          for(i = 0 ; i < argc ; i++)
98                  printf("Command line parameter %d = %s\n", (i + 1), argv[i]);
99
100         printf("\n");
101
102         /* Display Linux version information */
103         result = uname(&uname_s);
104         if(result != 0)
105                 error(1, errno, "uname");
```

```
106
107         printf("'%s'\n'%s'\n'%s'\n'%s'\n", uname_s.sysname, uname_s.release
108                                         ,uname_s.version, uname_s.machine);
109         printf("\n");
110
111         /* Exercise FPGA IP */
112         validate_system_status();
113         demo_sysid();
114         demo_ocram();
115         demo_leds();
116         demo_switches();
117         demo_buttons();
118
119         /* Goodbye message */
120         printf("Blink Application completed.\n");
121         printf("\n");
122
123         /* restore the original termios */
124         result = tcsetattr(STDIN_FILENO, TCSANOW, &original_termios);
125         if(result != 0)
126                 error(1, errno, "tcsetattr");
127
128         return(0);
129 }
130
131 void validate_system_status(void) {
132
133         int result;
134         int sysfs_fd;
135         char sysfs_str[256];
136
137         /* validate the FPGA state */
138         sysfs_fd = open(SYSFS_FPGA0_STATE_PATH, O_RDONLY);
139         if(sysfs_fd < 0)
140                 error(1, errno, "open sysfs FPGA state");
141         result = read(sysfs_fd, sysfs_str, strlen(SYSFS_FPGA0_STATE));
142         if(result < 0)
143                 error(1, errno, "read sysfs FPGA state");
144         close(sysfs_fd);
145         if(strncmp(SYSFS_FPGA0_STATE, sysfs_str, strlen(SYSFS_FPGA0_STATE)))
146                 error(1, 0, "FPGA not in operate state");
147
148         /* validate the LWH2F bridge name */
149         sysfs_fd = open(SYSFS_LWH2F_BRIDGE_NAME_PATH, O_RDONLY);
150         if(sysfs_fd < 0)
151                 error(1, errno, "open sysfs LWH2F bridge name");
152         result = read(sysfs_fd, sysfs_str, strlen(SYSFS_LWH2F_BRIDGE_NAME));
153         if(result < 0)
154                 error(1, errno, "read sysfs LWH2F bridge name");
155         close(sysfs_fd);
156         if(strncmp(SYSFS_LWH2F_BRIDGE_NAME, sysfs_str,
157                                 strlen(SYSFS_LWH2F_BRIDGE_NAME)))
158                 error(1, 0, "bad LWH2F bridge name");
159
160         /* validate the LWH2F bridge state */
161         sysfs_fd = open(SYSFS_LWH2F_BRIDGE_STATE_PATH, O_RDONLY);
162         if(sysfs_fd < 0)
```

```
163                 error(1, errno, "open sysfs LWH2F bridge state");
164         result = read(sysfs_fd, sysfs_str, strlen(SYSFS_LWH2F_BRIDGE_STATE));
165         if(result < 0)
166                 error(1, errno, "read sysfs LWH2F bridge state");
167         close(sysfs_fd);
168         if(strncmp(SYSFS_LWH2F_BRIDGE_STATE, sysfs_str,
169                                 strlen(SYSFS_LWH2F_BRIDGE_STATE)))
170                 error(1, 0, "LWH2F bridge not enabled");
171
172         /* validate the H2F bridge name */
173         sysfs_fd = open(SYSFS_H2F_BRIDGE_NAME_PATH, O_RDONLY);
174         if(sysfs_fd < 0)
175                 error(1, errno, "open sysfs H2F bridge name");
176         result = read(sysfs_fd, sysfs_str, strlen(SYSFS_H2F_BRIDGE_NAME));
177         if(result < 0)
178                 error(1, errno, "read sysfs H2F bridge name");
179         close(sysfs_fd);
180         if(strncmp(SYSFS_H2F_BRIDGE_NAME, sysfs_str,
181                                 strlen(SYSFS_H2F_BRIDGE_NAME)))
182                 error(1, 0, "bad H2F bridge name");
183
184         /* validate the H2F bridge state */
185         sysfs_fd = open(SYSFS_H2F_BRIDGE_STATE_PATH, O_RDONLY);
186         if(sysfs_fd < 0)
187                 error(1, errno, "open sysfs H2F bridge state");
188         result = read(sysfs_fd, sysfs_str, strlen(SYSFS_H2F_BRIDGE_STATE));
189         if(result < 0)
190                 error(1, errno, "read sysfs H2F bridge state");
191         close(sysfs_fd);
192         if(strncmp(SYSFS_H2F_BRIDGE_STATE, sysfs_str,
193                                 strlen(SYSFS_H2F_BRIDGE_STATE)))
194                 error(1, 0, "H2F bridge not enabled");
195
196         printf("FPGA appears to be configured and bridges are not in reset\n");
197         printf("\n");
198 }
199
200 void demo_sysid(void) {
201
202         int result;
203         int dev_mem_fd;
204         void *mmap_addr;
205         size_t mmap_length;
206         int mmap_prot;
207         int mmap_flags;
208         int mmap_fd;
209         off_t mmap_offset;
210         void *mmap_ptr;
211         volatile u_int *system_id_ptr;
212         u_int32_t system_ID_value;
213         u_int32_t system_TS_value;
214
215         /* map the peripheral span through /dev/mem */
216         dev_mem_fd = open("/dev/mem", O_RDWR | O_SYNC);
217         if(dev_mem_fd < 0)
218                 error(1, errno, "open /dev/mem");
```

```
219
220         mmap_addr = NULL;
221         mmap_length = SYSTEM_ID_SPAN;
222         mmap_prot = PROT_READ;
223         mmap_flags = MAP_SHARED;
224         mmap_fd = dev_mem_fd;
225         mmap_offset = SYSTEM_ID_BASE & ~(sysconf(_SC_PAGE_SIZE) - 1);
226         mmap_ptr = mmap(mmap_addr, mmap_length, mmap_prot, mmap_flags,
227                                                         mmap_fd, mmap_offset);
228         if(mmap_ptr == MAP_FAILED)
229                 error(1, errno, "mmap /dev/mem");
230
231         system_id_ptr = (u_int*)((u_int)mmap_ptr +
232                         (SYSTEM_ID_BASE & (sysconf(_SC_PAGE_SIZE) - 1)));
233
234         /* read the system ID values */
235         system_ID_value = system_id_ptr[0];
236         system_TS_value = system_id_ptr[1];
237
238         printf("System ID ID: 0x%08X\n", system_ID_value);
239         printf("System ID TS: 0x%08X\n", system_TS_value);
240
241         /* unmap /dev/mem mappings */
242         result = munmap(mmap_ptr, mmap_length);
243         if(result < 0)
244                 error(1, errno, "munmap /dev/mem");
245
246         close(dev_mem_fd);
247         printf("\n");
248 }
249
250 void demo_ocram(void) {
251
252         int result;
253         int dev_mem_fd;
254         void *mmap_addr;
255         size_t mmap_length;
256         int mmap_prot;
257         int mmap_flags;
258         int mmap_fd;
259         off_t mmap_offset;
260         void *mmap_ptr;
261         volatile u_int *ocram_64k_ptr;
262         void *original_ocram_64k_content_ptr;
263         int i;
264
265         /* map the peripheral span through /dev/mem */
266         dev_mem_fd = open("/dev/mem", O_RDWR | O_SYNC);
267         if(dev_mem_fd < 0)
268                 error(1, errno, "open /dev/mem");
269
270         mmap_addr = NULL;
271         mmap_length = OCRAM_64K_SPAN;
272         mmap_prot = PROT_READ | PROT_WRITE;
273         mmap_flags = MAP_SHARED;
274         mmap_fd = dev_mem_fd;
275         mmap_offset = OCRAM_64K_BASE & ~(sysconf(_SC_PAGE_SIZE) - 1);
```

```
276         mmap_ptr = mmap(mmap_addr, mmap_length, mmap_prot, mmap_flags,
277                                                         mmap_fd, mmap_offset);
278         if(mmap_ptr == MAP_FAILED)
279                 error(1, errno, "mmap /dev/mem");
280
281         ocram_64k_ptr = (u_int*)((u_int)mmap_ptr +
282                         (OCRAM_64K_BASE & (sysconf(_SC_PAGE_SIZE) - 1)));
283
284         /* save current ocram 64k contents */
285         original_ocram_64k_content_ptr = malloc(OCRAM_64K_SPAN);
286         if(original_ocram_64k_content_ptr == NULL)
287                 error(1, errno, "malloc");
288
289         memcpy(original_ocram_64k_content_ptr, (const void *)ocram_64k_ptr,
290                                                             OCRAM_64K_SPAN);
291         result = memcmp(original_ocram_64k_content_ptr,
292                                 (const void *)ocram_64k_ptr, OCRAM_64K_SPAN);
293         if(result != 0)
294                 error(1, errno, "memcmp original copy");
295
296         printf("Saved initial ocram 64k values\n");
297
298         /* test ocram 64k */
299         printf("Writing sequential word values to ocram 64k\n");
300         for(i = 0 ; i < (OCRAM_64K_SPAN / 4) ; i++) {
301                 ocram_64k_ptr[i] = i;
302         }
303
304         printf("Verifying sequential word values in ocram 64k\n");
305         for(i = 0 ; i < (OCRAM_64K_SPAN / 4) ; i++) {
306                 if(ocram_64k_ptr[i] != (u_int)(i)) {
307                         printf("mismatch at word %d\n", i);
308                         printf("expected 0x%08X\n", i);
309                         printf("got 0x%08X\n", ocram_64k_ptr[i]);
310                 }
311         }
312
313         printf("Writing complimented sequential word values to ocram 64k\n");
314         for(i = 0 ; i < (OCRAM_64K_SPAN / 4) ; i++) {
315                 ocram_64k_ptr[i] = ~i;
316         }
317
318         printf("Verifying complimented sequential word values in ocram 64k\n");
319         for(i = 0 ; i < (OCRAM_64K_SPAN / 4) ; i++) {
320                 if(ocram_64k_ptr[i] != ~(u_int)(i)) {
321                         printf("mismatch at word %d\n", i);
322                         printf("expected 0x%08X\n", ~i);
323                         printf("got 0x%08X\n", ocram_64k_ptr[i]);
324                 }
325         }
326
327         /* restore ocram 64k contents */
328         memcpy((void *)ocram_64k_ptr, original_ocram_64k_content_ptr,
329                                                             OCRAM_64K_SPAN);
330         result = memcmp((const void *)ocram_64k_ptr,
331                                 original_ocram_64k_content_ptr, OCRAM_64K_SPAN);
```

```
332         if(result != 0)
333                 error(1, errno, "memcmp restore copy");
334
335         printf("Restored initial ocram 64k values\n");
336
337         /* free the malloc buffer */
338         free(original_ocram_64k_content_ptr);
339
340         /* unmap /dev/mem mappings */
341         result = munmap(mmap_ptr, mmap_length);
342         if(result < 0)
343                 error(1, errno, "munmap /dev/mem");
344
345         close(dev_mem_fd);
346         printf("\n");
347 }
348
349 void demo_leds(void) {
350
351         int result;
352         int dev_mem_fd;
353         void *mmap_addr;
354         size_t mmap_length;
355         int mmap_prot;
356         int mmap_flags;
357         int mmap_fd;
358         off_t mmap_offset;
359         void *mmap_ptr;
360         volatile u_int *led_pio_ptr;
361         u_int32_t led_pio_value;
362         u_int32_t led = 0;
363
364         /* map the peripheral span through /dev/mem */
365         dev_mem_fd = open("/dev/mem", O_RDWR | O_SYNC);
366         if(dev_mem_fd < 0)
367                 error(1, errno, "open /dev/mem");
368
369         mmap_addr = NULL;
370         mmap_length = LED_PIO_SPAN;
371         mmap_prot = PROT_READ | PROT_WRITE;
372         mmap_flags = MAP_SHARED;
373         mmap_fd = dev_mem_fd;
374         mmap_offset = LED_PIO_BASE & ~(sysconf(_SC_PAGE_SIZE) - 1);
375         mmap_ptr = mmap(mmap_addr, mmap_length, mmap_prot, mmap_flags,
376                                                         mmap_fd, mmap_offset);
377         if(mmap_ptr == MAP_FAILED)
378                 error(1, errno, "mmap /dev/mem");
379
380         led_pio_ptr = (u_int*)((u_int)mmap_ptr +
381                         (LED_PIO_BASE & (sysconf(_SC_PAGE_SIZE) - 1)));
382
383         /* read the LED PIO value */
384         led_pio_value = led_pio_ptr[0];
385
386         /* Blink the LEDs until any key is pressed */
387         printf("Blinking LEDs\n");
```

```
388         printf("Press any key to stop LED lightshow\n");
389
390         while(getc(stdin) == -1) {
391
392                 /* Turn on LED */
393                 led_pio_ptr[0] = 1L << led;
394
395                 /* Delay */
396                 usleep(LED_DELAY_US);
397
398                 /* Advance to next LED */
399                 led = (led + 1) % LED_PIO_DATA_WIDTH;
400         }
401
402         /* restore the LED PIO value */
403         led_pio_ptr[0] = led_pio_value;
404
405         /* unmap /dev/mem mappings */
406         result = munmap(mmap_ptr, mmap_length);
407         if(result < 0)
408                 error(1, errno, "munmap /dev/mem");
409
410         close(dev_mem_fd);
411         printf("\n");
412 }
413
414 void demo_switches(void) {
415
416         int result;
417         int dev_mem_fd;
418         void *mmap_addr;
419         size_t mmap_length;
420         int mmap_prot;
421         int mmap_flags;
422         int mmap_fd;
423         off_t mmap_offset;
424         void *mmap_ptr;
425         volatile u_int *switch_pio_ptr;
426         u_int32_t prev_switches;
427         u_int32_t switches;
428
429         /* map the peripheral span through /dev/mem */
430         dev_mem_fd = open("/dev/mem", O_RDWR | O_SYNC);
431         if(dev_mem_fd < 0)
432                 error(1, errno, "open /dev/mem");
433
434         mmap_addr = NULL;
435         mmap_length = SWITCH_PIO_SPAN;
436         mmap_prot = PROT_READ;
437         mmap_flags = MAP_SHARED;
438         mmap_fd = dev_mem_fd;
439         mmap_offset = SWITCH_PIO_BASE & ~(sysconf(_SC_PAGE_SIZE) - 1);
440         mmap_ptr = mmap(mmap_addr, mmap_length, mmap_prot, mmap_flags,
441                                                         mmap_fd, mmap_offset);
442         if(mmap_ptr == MAP_FAILED)
443                 error(1, errno, "mmap /dev/mem");
```

```
444
445         switch_pio_ptr = (u_int*)((u_int)mmap_ptr +
446                         (SWITCH_PIO_BASE & (sysconf(_SC_PAGE_SIZE) - 1)));
447
448         /* Display state of switches */
449         printf("Displaying the state of switches\n");
450         printf("Slide switch SW0, Sw1, SW2 or SW3 to observe updates\n");
451         printf("Press any key to stop displaying the switches\n");
452         prev_switches = ~switch_pio_ptr[0];
453
454         while(getc(stdin) == -1) {
455                 switches = switch_pio_ptr[0];
456                 if(switches != prev_switches) {
457                         printf("Switch value: 0x%01x\n", (int)switches);
458                         prev_switches = switches;
459                 }
460         }
461
462         /* unmap /dev/mem mappings */
463         result = munmap(mmap_ptr, mmap_length);
464         if(result < 0)
465                 error(1, errno, "munmap /dev/mem");
466
467         close(dev_mem_fd);
468         printf("\n");
469 }
470
471 void demo_buttons(void) {
472
473         int result;
474         int dev_mem_fd;
475         void *mmap_addr;
476         size_t mmap_length;
477         int mmap_prot;
478         int mmap_flags;
479         int mmap_fd;
480         off_t mmap_offset;
481         void *mmap_ptr;
482         volatile u_int *button_pio_ptr;
483         u_int32_t prev_buttons;
484         u_int32_t buttons;
485
486         /* map the peripheral span through /dev/mem */
487         dev_mem_fd = open("/dev/mem", O_RDWR | O_SYNC);
488         if(dev_mem_fd < 0)
489                 error(1, errno, "open /dev/mem");
490
491         mmap_addr = NULL;
492         mmap_length = BUTTON_PIO_SPAN;
493         mmap_prot = PROT_READ;
494         mmap_flags = MAP_SHARED;
495         mmap_fd = dev_mem_fd;
496         mmap_offset = BUTTON_PIO_BASE & ~(sysconf(_SC_PAGE_SIZE) - 1);
497         mmap_ptr = mmap(mmap_addr, mmap_length, mmap_prot, mmap_flags,
498                                                         mmap_fd, mmap_offset);
499         if(mmap_ptr == MAP_FAILED)
500                 error(1, errno, "mmap /dev/mem");
```

```
        button_pio_ptr = (u_int*)((u_int)mmap_ptr +
                        (BUTTON_PIO_BASE & (sysconf(_SC_PAGE_SIZE) - 1)));

        /* Display state of buttons */
        printf("Displaying the state of buttons\n");
        printf("Press push button KEY1 to observe updates\n");
        printf("Press any key to stop displaying the buttons\n");
        prev_buttons = ~button_pio_ptr[0];

        while(getc(stdin) == -1) {
                buttons = button_pio_ptr[0];
                if(buttons != prev_buttons) {
                        printf("Button value: 0x%01x\n", (int)buttons);
                        prev_buttons = buttons;
                }
        }

        /* unmap /dev/mem mappings */
        result = munmap(mmap_ptr, mmap_length);
        if(result < 0)
                error(1, errno, "munmap /dev/mem");

        close(dev_mem_fd);
        printf("\n");
}
```

Now, it's important to realize that this method of creating a userspace application to interact with the FPGA peripherals using **/dev/mem** to map their address spans allows us to perform some very simple interactions like peek and poke into the FPGA peripherals. But this method is completely inadequate to support many types of interactions that more complex peripherals may require, like interrupt handling, DMA data movement and other higher end functions that the kernel environment provides to kernel modules which operate in kernel space. This tutorial will not demonstrate any kernel space module development concepts.

Step 1. Run the build script to build the *linux_blink_app* program. You can download the build script file from the GitHub repo using the links above.

```
root@de10-nano:~# cd ~/linux_examples/c_examples/
root@de10-nano:~# ./build_app.sh linux_blink_app.c
```

Step 2. Run the *linux_blink_app* program:

```
root@de10-nano:~# ./linux_blink_app
```

Step 3. Interact with the application as instructed on the Linux console:

```
Blink application started.

Number of command line parameters: 1
Command line parameter 1 = ./linux_blink_app

'Linux'
'4.1.33-ltsi-altera'
'#1 SMP Thu Mar 30 10:37:56 PDT 2017'
'armv7l'
```

```
FPGA appears to be configured and bridges are not in reset

System ID ID: 0xDE10DE10
System ID TS: 0x593B4F62

Saved initial ocram 64k values
Writing sequential word values to ocram 64k
Verifying sequential word values in ocram 64k
Writing complimented sequential word values to ocram 64k
Verifying complimented sequential word values in ocram 64k
Restored initial ocram 64k values

Blinking LEDs
Press any key to stop LED lightshow

Displaying the state of switches
Slide switch SW0, Sw1, SW2 or SW3 to observe updates
Press any key to stop displaying the switches
Switch value: 0xf
Switch value: 0xe
Switch value: 0xc
Switch value: 0x8
Switch value: 0x0

Displaying the state of buttons
Press push button KEY1 to observe updates
Press any key to stop displaying the buttons
Button value: 0x1
Button value: 0x0
Button value: 0x1
Button value: 0x0
Button value: 0x1

Blink Application completed.
```

That's it, you've interacted with the Qsys system using a Linux application. Continue on to the next section where we demonstrate how to load a device tree overlay and interact with the kernel device drivers provided for some of the FPGA peripherals.

Load a Device Tree Overlay to Interact with the FPGA Design

In this section we demonstrate how to construct and load a device tree overlay on the running Linux environment to enable the default Linux drivers that control various peripherals in our FPGA design. The PIO peripherals and the System ID peripheral that are in our FPGA design have standard Linux drivers that are compiled into the kernel already, once we load a device tree overlay to inform the kernel of the presence of these peripherals in the FPGA, we can use the standard GPIO and System ID drivers to interact with those peripherals. We supplied you with the **soc_system.dtbo** precompiled device tree binary overlay file in the Linux examples archive.

```
root@de10-nano:~# cd ~/linux_examples/devicetree_overlay/
root@de10-nano:~# ls
soc_system.dtbo  soc_system.dtso
```

We also supplied the **soc_system.dtso** device tree source overlay file that the binary overlay file was compiled from, it's contents are shown below. Please study it and observe how we define the memory mapped structure and clocking structure of the FPGA peripherals. Fundamentally, this device tree overlay informs the kernel what drivers it should bind

with our peripherals in the FPGA, then the driver can discover where its base address is and other relevant information about the peripheral. Because we defined our PIO peripherals with a **gpio-controller** binding, that informs the kernel to add these into the standard kernel GPIO framework. And because we created a node for **gpio-leds** and added our LED PIOs to that node, those will show up in the standard kernel LED framework. You can download the device tree source overlay file from the GitHub repo using the links above.

```
//
// Copyright (c) 2017 Intel Corporation
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to
// deal in the Software without restriction, including without limitation the
// rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
// sell copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
// FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
// IN THE SOFTWARE.
//

/dts-v1/ /plugin/;

/ {
        fragment@0 {
                target-path = "/soc/base-fpga-region";

                #address-cells = <1>;
                #size-cells = <1>;

                __overlay__ {
                        external-fpga-config;

                        fpga-bridges = <&fpga_bridge0 &fpga_bridge1 &fpga_bridge2>;

                        //
                        // define the memory mapped details of our overlay
                        //

                        #address-cells = <2>;
                        #size-cells = <1>;

                        // first ranges line represents the H2F bridge
                        // second ranges line represents the LWH2F bridge
                        ranges = <0x00000000 0x00000000 0xc0000000 0x00010000>,
                                 <0x00000001 0x00000000 0xff200000 0x00010038>;

                        // there is no driver for the onchip memory core, but
                        // since this is the only peripheral on the H2F bridge
                        // we'll define it as an example of how the reg binding
```

```
52                      // relates to the ranges binding above
53                      ocram_64k: memory@0x000000000 {
54                              compatible = "not,compatible";
55                              reg = <0x00000000 0x00000000 0x00010000>;
56                              clocks = <&clk_0>;
57                      }; // ocram_64k
58
59                      // the rest of the peripherals below are on the LWH2F
60
61                      led_pio: gpio@0x100010000 {
62                              compatible = "altr,pio-1.0";
63                              reg = <0x00000001 0x00010000 0x00000010>;
64                              clocks = <&clk_0>;
65                              altr,ngpio = <8>;
66                              #gpio-cells = <2>;
67                              gpio-controller;
68                      }; // led_pio
69
70                      button_pio: gpio@0x100010010 {
71                              compatible = "altr,pio-1.0";
72                              reg = <0x00000001 0x00010010 0x00000010>;
73                              clocks = <&clk_0>;
74                              altr,ngpio = <1>;
75                              #gpio-cells = <2>;
76                              gpio-controller;
77                      }; // button_pio
78
79                      switch_pio: gpio@0x100010020 {
80                              compatible = "altr,pio-1.0";
81                              reg = <0x00000001 0x00010020 0x00000010>;
82                              clocks = <&clk_0>;
83                              altr,ngpio = <4>;
84                              #gpio-cells = <2>;
85                              gpio-controller;
86                      }; // switch_pio
87
88                      system_id: sysid@0x100010030 {
89                              compatible = "altr,sysid-1.0";
90                              reg = <0x00000001 0x00010030 0x00000008>;
91                              clocks = <&clk_0>;
92                      }; // system_id
93
94                      //
95                      // define the non-memory mapped details of our overlay
96                      //
97
98                      clk_0: clk_0 {
99                              compatible = "fixed-clock";
100                             #clock-cells;
101                             clock-frequency;
102                             clock-output-names = "clk_0-clk";
103                     }; // clk_0
104
105                     // define the led_pio pins as gpio-leds, this will allow
106                     // the gpio-leds driver to control these ports
107                     leds: leds {
```

```
                                compatible = "gpio-leds";

                                led_fpga0: fpga0 {
                                        label = "fpga_led0";
                                        gpios = <&led_pio 0 0>;
                                }; // led_fpga0

                                led_fpga1: fpga1 {
                                        label = "fpga_led1";
                                        gpios = <&led_pio 1 0>;
                                }; // led_fpga1

                                led_fpga2: fpga2 {
                                        label = "fpga_led2";
                                        gpios = <&led_pio 2 0>;
                                }; // led_fpga2

                                led_fpga3: fpga3 {
                                        label = "fpga_led3";
                                        gpios = <&led_pio 3 0>;
                                }; // led_fpga3

                                led_fpga4: fpga4 {
                                        label = "fpga_led4";
                                        gpios = <&led_pio 4 0>;
                                }; // led_fpga4

                                led_fpga5: fpga5 {
                                        label = "fpga_led5";
                                        gpios = <&led_pio 5 0>;
                                }; // led_fpga5

                                led_fpga6: fpga6 {
                                        label = "fpga_led6";
                                        gpios = <&led_pio 6 0>;
                                }; // led_fpga6

                                led_fpga7: fpga7 {
                                        label = "fpga_led7";
                                        gpios = <&led_pio 7 0>;
                                }; // led_fpga7
                        }; // leds
                }; // __overlay__
        }; // fragment@0
}; // /
```

The device tree source overlay file above was compiled into the device tree binary overlay file by using the **dtc** compiler provided in the Intel SoC FPGA EDS tools installation. Since the standard Terasic DE10-Nano SD card image does not contain the device tree compiler, we have supplied you with the precompiled **soc_system.dtbo** file. This is the command line that was used to compile the device tree overlay using **dtc** from the SoC EDS installation:

```
$ ${SOCEDS_DEST_ROOT:?}/host_tools/gnu/dtc/dtc \
        -@ \
        -I dts \
        -O dtb \
        -o soc_system.dtbo \
```

```
        soc_system.dtso
```

Step 1. Before we apply the device tree overlay, let's observe the current state of the system so we can see the effects of the overlay after it is applied.

```
# observe that we do not see anything in sysfs related to the system ID
root@de10-nano:~# find /sys -name "*sysid*"

# observe that we only see three gpiochip groups defined, these are HPS gpios
root@de10-nano:~# ls /sys/class/gpio/
export       gpiochip427  gpiochip454  gpiochip483  unexport

# observe that we only see one led defined, this is an HPS led
root@de10-nano:~# ls /sys/class/leds/
hps_led0
```

Step 2. Apply the device tree overlay.

Step 2a. Copy the device tree binary overlay file into the **/lib/firmware** directory. This is where the kernel will look for the overlay as we invoke future commands.

```
root@de10-nano:~# cd ~/linux_examples/devicetree_overlay/
root@de10-nano:~# cp soc_system.dtbo /lib/firmware
```

Step 2b. Create a directory to install our overlay in the kernel sysfs tree.

```
root@de10-nano:~# mkdir /sys/kernel/config/device-tree/overlays/soc_system.dtbo
```

Step 2c. Observe that the kernel creates a number of virtual files in this new directory and they indicate that no overlay has been applied yet.

```
root@de10-nano:~# ls /sys/kernel/config/device-tree/overlays/soc_system.dtbo
dtbo    path    status
root@de10-nano:~# cat /sys/kernel/config/device-tree/overlays/soc_system.dtbo/path

root@de10-nano:~# cat /sys/kernel/config/device-tree/overlays/soc_system.dtbo/status
unapplied
```

Step 2d. Apply the overlay by writing the name of our device tree binary overlay file into the **path** file of our overlay directory. This causes the kernel to search the **/lib/firmware** directory for our overlay file.

```
root@de10-nano:~# echo soc_system.dtbo > /sys/kernel/config/device-tree/overlays/soc_system.dtbo/path
```

Step 2e. And now we should observe that the special files indicate that our overlay has been applied.

```
root@de10-nano:~# cat /sys/kernel/config/device-tree/overlays/soc_system.dtbo/path
soc_system.dtbo
root@de10-nano:~# cat /sys/kernel/config/device-tree/overlays/soc_system.dtbo/status
applied
```

Step 3. Now observe the state of the system and see the effects of the overlay.

```
# observe that we now see a number of system ID related entries in sysfs
root@de10-nano:~# find /sys -name "*sysid*"
/sys/bus/platform/devices/ff210030.sysid
/sys/bus/platform/drivers/altera_sysid
/sys/bus/platform/drivers/altera_sysid/ff210030.sysid
/sys/devices/platform/soc/soc:base-fpga-region/ff210030.sysid
/sys/devices/platform/soc/soc:base-fpga-region/ff210030.sysid/sysid
```

```
/sys/firmware/devicetree/base/soc/base-fpga-region/sysid@0x100010030
/sys/module/altera_sysid
/sys/module/altera_sysid/drivers/platform:altera_sysid

# observe that the system ID driver has bound to our system ID peripheral
root@de10-nano:~# ls /sys/bus/platform/drivers/altera_sysid
bind            ff210030.sysid  module          uevent          unbind

# observe that a number of additional gpiochip definitions have appeared
root@de10-nano:~# ls /sys/class/gpio/
export       gpiochip418  gpiochip427  gpiochip483
gpiochip414  gpiochip419  gpiochip454  unexport

# observe that a number of additional led definitions have appeared
root@de10-nano:~# ls /sys/class/leds/
fpga_led0  fpga_led2  fpga_led4  fpga_led6  hps_led0
fpga_led1  fpga_led3  fpga_led5  fpga_led7
```

Step 4. We have provided a number of shell scripts that demonstrate how to interact with the sysfs entries. You can download the shell script files from the GitHub repo using the links above.

```
root@de10-nano:~# cd ~/linux_examples/sysfs_examples/
root@de10-nano:~# ls
read_button_pio_sysfs.sh  read_system_id_sysfs.sh
read_switch_pio_sysfs.sh  toggle_leds_sysfs.sh
```

Step 5. You can study the contents of each script to understand how they interact with the sysfs entries to discover the peripheral that they need to interact with, and then interact with it. You can run these shell scripts as follows.

```
# this script will toggle the LEDs all on and then all off in sequential order
root@de10-nano:~# ./toggle_leds_sysfs.sh

# reading the switches with all switches in the up position
root@de10-nano:~# ./read_switch_pio_sysfs.sh
SWITCH_PIO[0]: '1'
SWITCH_PIO[1]: '1'
SWITCH_PIO[2]: '1'
SWITCH_PIO[3]: '1'

# reading the switches with the far right switch in the down position
root@de10-nano:~# ./read_switch_pio_sysfs.sh
SWITCH_PIO[0]: '0'
SWITCH_PIO[1]: '1'
SWITCH_PIO[2]: '1'
SWITCH_PIO[3]: '1'

# reading the push button while not pressed
root@de10-nano:~# ./read_button_pio_sysfs.sh
BUTTON_PIO: '1'

# reading the push button while pressed
root@de10-nano:~# ./read_button_pio_sysfs.sh
BUTTON_PIO: '0'

# reading the system ID peripheral
root@de10-nano:~# ./read_system_id_sysfs.sh
```

```
System ID ID: '3725647376'
System ID TS: '1497059170 (2017-6-10 1:46:10 UTC)'
```

Step 6. Now let's remove our overlay from the running system and witness the effects. To remove our overlay we simply remove the overlay directory that we created in the sysfs:

```
root@de10-nano:~# rmdir /sys/kernel/config/device-tree/overlays/soc_system.dtbo
```

Step 7. And now we can observe the state of our system after removing the overlay:

```
# observe that the sysfs driver is no longer bound to the peripheral
root@de10-nano:~# ls /sys/bus/platform/drivers/altera_sysid
bind    module  uevent  unbind

# observe that we are back down to the original three gpiochip definitions
root@de10-nano:~# ls /sys/class/gpio/
export       gpiochip427  gpiochip454  gpiochip483  unexport

# observe that we are back down to the single led definition
root@de10-nano:~# ls /sys/class/leds/
hps_led0
```

That's it, you've interacted with the Qsys system using a device tree overlay and the standard Linux drivers.