# Interacting with FPGA Designs using U-Boot

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

## Contents



## Overview

This tutorial demonstrates how to use the U-Boot boot loader to program a compiled FPGA design into an SoC FPGA device, then interact with the hardware modules instantiated in that FPGA design from the HPS processor using U-Boot. This U-Boot tutorial is based on the FPGA design created in a prior tutorial "My First HPS System". In that tutorial you create this Qsys system:

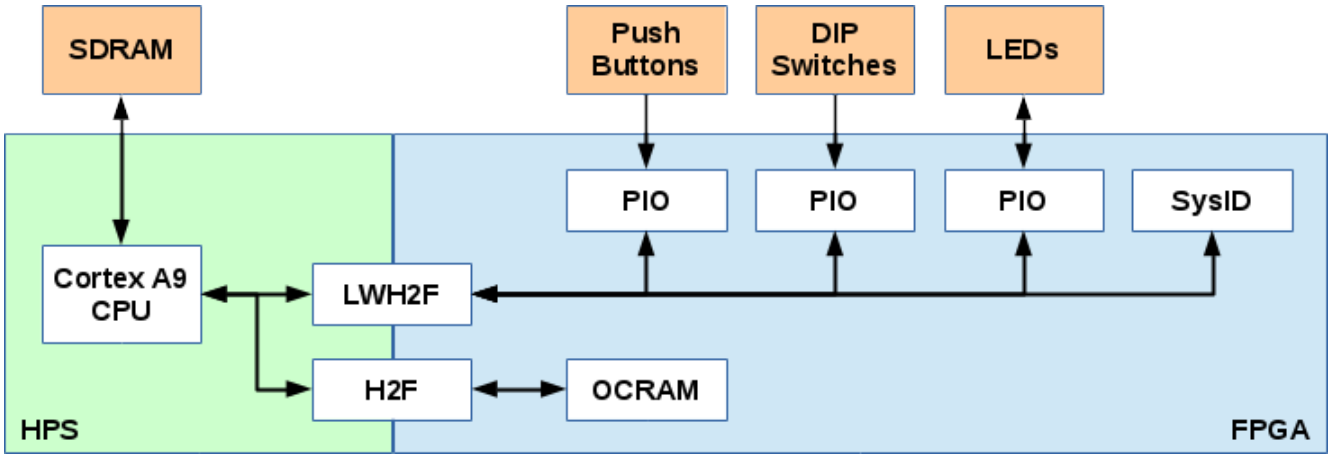


Figure 1: HPS System Block Diagram

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. Both U-Boot console commands and a U-Boot standalone application are used to interact with the soft IP peripherals residing in the FPGA fabric.

The following reference materials provide more information about the U-Boot features that are demonstrated in this tutorial:

- U-Boot Command Line Interface: https://www.denx.de/wiki/view/DULG/UBootCommandLineInterface
- U-Boot Command Line Parsing: https://www.denx.de/wiki/view/DULG/CommandLineParsing
- U-Boot Scripting Capabilities: https://www.denx.de/wiki/DULG/UBootScripts
- U-Boot Standalone Applications: https://www.denx.de/wiki/view/DULG/UBootStandalone

# 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 DE10-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 Sources Archive: de10-nano-image-Angstrom-v2016.12.socfpga-sdimg.2017.03.31.sources.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, source 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.** Mount the FAT partition of the microSD card on the host PC, and copy the **blink.rbf** FPGA configuration file 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 file onto the FAT partition
$ sudo cp $DE10_NANO/blink/output_files/blink.rbf sdcard/

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

**Step 5.** Remove the microSD card from your host PC and insert it into the Terasic DE10-Nano board.

**Step 6.** 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.

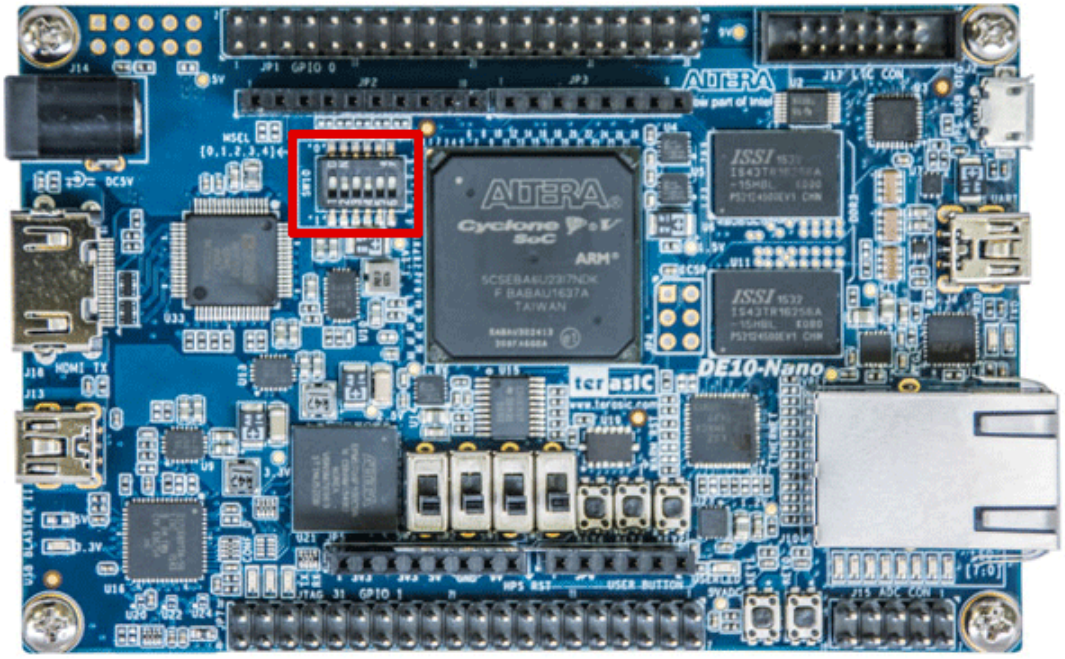


Figure 2: SW10 Configuration

**Step 7.** 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.

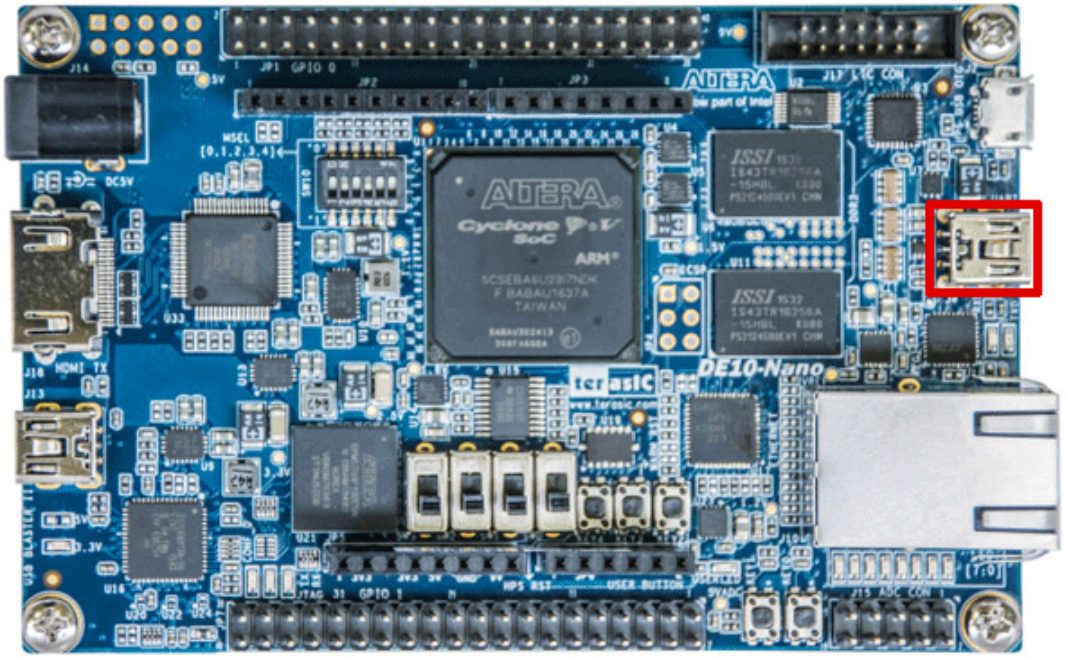


Figure 3: UART Console Connector on Terasic DE10-Nano Board

**Step 8.** 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 9.** *Read the next step before completing this step.* 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.

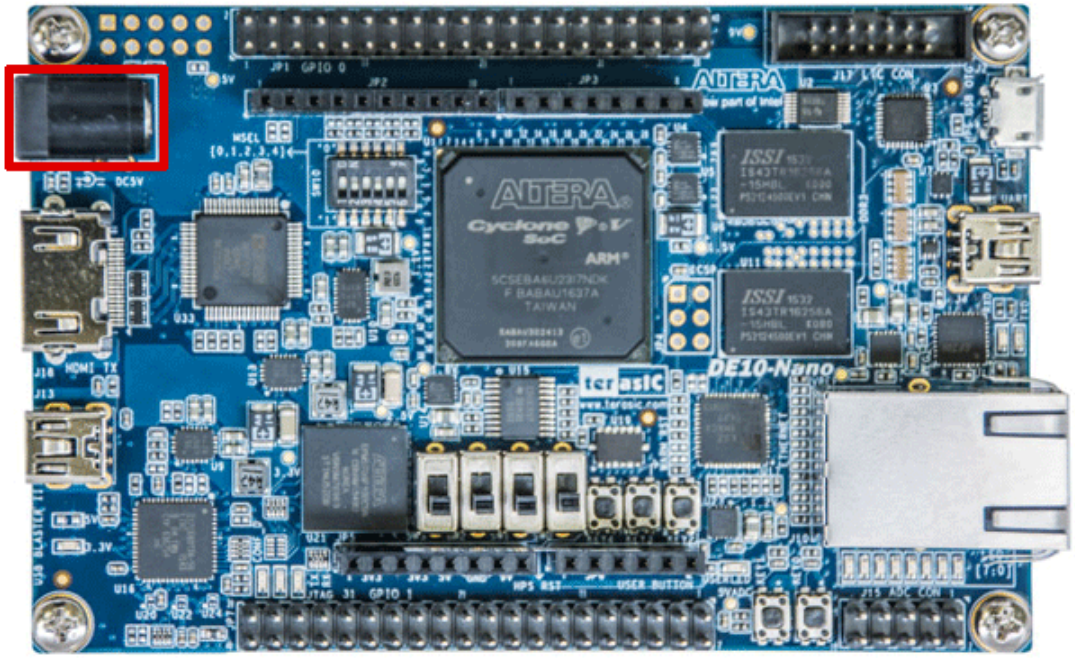


Figure 4: Power Connector on Terasic DE10-Nano Board

**Step 10.** 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 11.** On the U-Boot console, run the following commands to configure the FPGA with the **blink.rbf** file from the SD card

```
=> load mmc 0:1 ${kernel_addr_r} blink.rbf
=> fpga load 0 ${kernel_addr_r} ${filesize}
=> bridge enable
```

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

## Using U-Boot Commands to Interact with the FPGA Design

This section describes various U-Boot commands and techniques for interacting with the FPGA design.

**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.** Next, we will interact with the IP modules inside the FPGA fabric. 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. To refresh your memory, here is what those addresses are:

```
#define OCRAM_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 U-Boot variables to allow us to recall these base addresses easier in the rest of this tutorial. Execute these commands on the U-Boot console to setup these variables:

```
=> OCRAM_64K_BASE=0xc0000000
=> LED_PIO_BASE=0xff210000
=> BUTTON_PIO_BASE=0xff210010
=> SWITCH_PIO_BASE=0xff210020
=> SYSTEM_ID_BASE=0xff210030
```

**Step 3.** Exercise the IP inside the FPGA fabric by running various commands

**Step 3a.** Interact with the onchip RAM component.

```
# read the onchip RAM
=> md.l $OCRAM_64K_BASE 1
c0000000: 00000000                                  ....

# write a pattern to the onchip RAM
=> mw.l $OCRAM_64K_BASE 0x1234de10

# verify the pattern remains in the onchip RAM
=> md.l $OCRAM_64K_BASE 1
c0000000: 1234de10                                  ..4.
```

**Step 3b.** Interact with the LED PIO component.

```
# turn on half the LEDs
=> mw.l $LED_PIO_BASE 0x55

# turn off those LEDs and turn on the other half of the LEDs
=> mw.l $LED_PIO_BASE 0xaa

# write a loop to toggle LED0 and LED1 every half second for 10 times
=> setenv COUNT 0
=> while itest ${COUNT} < 0x0a
> do
> mw.l $LED_PIO_BASE 0x01
> sleep 0.5
> mw.l $LED_PIO_BASE 0x02
> sleep 0.5
> setexpr COUNT ${COUNT} + 1
> done

# turn on all of the LEDs
=> mw.l $LED_PIO_BASE 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
=> mw.l 0xFFD05020 0x40

# deassert the H2F reset
=> mw.l 0xFFD05020 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
=> md.l $BUTTON_PIO_BASE 1
ff210010: 00000001                                ....

# read the button with KEY1 pressed
=> md.l $BUTTON_PIO_BASE 1
ff210010: 00000000                                ....

# read the button with KEY1 not pressed
=> md.l $BUTTON_PIO_BASE 1
ff210010: 00000001                                ....
```

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

```
# all switches in the up position
=> md.l $SWITCH_PIO_BASE 1
ff210020: 0000000f                                ....

# far right switch moved down
=> md.l $SWITCH_PIO_BASE 1
ff210020: 0000000e                                ....

# next switch to the left moved down
=> md.l $SWITCH_PIO_BASE 1
ff210020: 0000000c                                ....

# next switch to the left moved down
=> md.l $SWITCH_PIO_BASE 1
ff210020: 00000008                                ....

# last switch moved down
=> md.l $SWITCH_PIO_BASE 1
ff210020: 00000000                                ....
```

**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.

```
=> md.l $SYSTEM_ID_BASE 2
ff210030: de10de10 59187c42                       ....B|.Y
```

**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
=> md.l 0xff211000 4
ff211000: 00000000 00000000 00000000 00000000    ................

# now lets set those 4 words with a specific incrementing pattern
=> mw.l 0xff211000 0x0badf00d
=> mw.l 0xff211004 0x1badf00d
```

```
=> mw.l 0xff211008 0x2badf00d
=> mw.l 0xff21100c 0x3badf00d

# now validate that pattern repeats every 4 words
=> md.l 0xff211000 8
ff211000: 0badf00d 1badf00d 2badf00d 3badf00d    ..............;
ff211010: 0badf00d 1badf00d 2badf00d 3badf00d    ..............;

# 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
=> mw.l $SYSTEM_ID_BASE 0xfacecafe

# now verify that the first word of the default slave was actually written
=> md.l 0xff211000 4
ff211000: facecafe 1badf00d 2badf00d 3badf00d    ..............;

# 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
=> md.l 0xc0010000 4
c0010000: facecafe 1badf00d 2badf00d 3badf00d    ..............;
```

That's it, you've interacted with the Qsys system using U-Boot commands. Continue on to the next section where we demonstrate how to write a standalone U-Boot program to interact with the Qsys system.

## Using U-Boot Applications to Interact with the FPGA Design

This section presents the following:

- Getting the required cross compiler toolchain and the U-Boot sources
- Writing a U-Boot application that exercises the IP inside the FPGA fabric
- Compiling the U-Boot application
- Running the U-Boot application

**Step 1.** Save the sources archive de10-nano-image-Angstrom-v2016.12.socfpga-sdimg.2017.03.31.sources.tgz to the **$DE10_NANO** directory on the host PC.

**Step 2.** Run the following commands to get the toolchain, and prepare the U-Boot sources:

```
# change into the tutorial directory
$ cd $DE10_NANO

# get the linaro ARM GCC toolchain
$ wget https://releases.linaro.org/components/toolchain/binaries/6.2-2016.11/\
> arm-linux-gnueabihf/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf.tar.xz

# extract the toolchain
$ tar xf gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf.tar.xz

# extract the sources archive
$ tar xf de10-nano-image-Angstrom-v2016.12.socfpga-sdimg.2017.03.31.sources.tgz

# retrieve the U-Boot archives from the extracted sources archive
$ cp -r sources/arm-angstrom-linux-gnueabi/u-boot-socfpga* uboot-socfpga
```

```
# extract the U-Boot archives
$ cd uboot-socfpga
$ tar xf u-boot-socfpga-*-r0.tar.gz
$ tar xf u-boot-socfpga-*-r0-hardware.tar.gz

# patch the U-Boot sources with the DE10-Nano changes
$ cd git
$ cat ../*.patch | patch -p1
```

The U-Boot standalone application examples are stored in **$DE10_NANO/uboot-socfpga/git/examples/standalone/**.

**Step 3.** Copy the **blink/qsys_headers/hps_0_arm_a9_0.h** header created in the previous tutorial into the U-Boot standalone examples directory:

```
# copy hps_0_arm_a9_0.h into u-boot standalone examples directory
$ cp $DE10_NANO/blink/qsys_headers/hps_0_arm_a9_0.h \
> $DE10_NANO/uboot-socfpga/git/examples/standalone/
```

**Step 4.** Download the blink_app.c file here. Save the C source file to this location **$DE10_NANO/uboot-socfpga/git/examples/standalone/blink_app.c**. If you wish to view the blink_app.c file in the GitHub* repo you can look here. These are the contents of the standalone application:

```
/*
 * 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 <common.h>
#include <exports.h>

#include "hps_0_arm_a9_0.h"

/* Macro for reading from memory */
#define readl(addr)         *((volatile unsigned int *)(addr))

/* Macro for writing to memory */
#define writel(value,addr)        *((volatile unsigned int *)(addr)) = (value)

```

```c
#define LED_DELAY_US 250000

void demo_sysid(void);
void demo_ocram(void);
void demo_leds(void);
void demo_switches(void);
void demo_buttons(void);

int blink_app(int argc, char * const argv[]) {

        int i;

        /* Start application */
        app_startup(argv);

        /* Display welcome message */
        printf("\n");
        printf("Blink application started.\n");
        printf("\n");

        /* Display command line parameters */
        printf("Number of command line parameters: %d\n", argc);
        for(i=0; i<argc; i++)
                printf("Command line parameter %d = %s\n", (i + 1), argv[i]);

        printf("\n");

        /* Display U-Booot ABI version */
        printf("U-Boot ABI version %d\n", (int)get_version());
        printf("\n");

        /* Exercise FPGA IP */
        demo_sysid();
        demo_ocram();
        demo_leds();
        demo_switches();
        demo_buttons();

        /* Goodbye message */
        printf("Blink Application completed.\n");
        printf("\n");

        return(0);
}

void demo_sysid(void) {

        /* Display SysID parameters */
        printf("SysID.id = 0x%08x\n", (int)readl(SYSTEM_ID_BASE));
        printf("SysID.timestamp = 0x%08x\n", (int)readl(SYSTEM_ID_BASE + 0x4));
        printf("\n");
}

void demo_ocram(void) {

        unsigned int * ptr = (unsigned int *)OCRAM_64K_BASE;
```

```
90          unsigned int i;
91
92          printf("Writing pattern into FPGA OCRAM.\n");
93          for(i = 0; i< OCRAM_64K_SIZE_VALUE / sizeof(unsigned int); i++)
94                  ptr[i] = i;
95
96          printf("Checking pattern from FPGA OCRAM: ");
97          for(i = 0; i< OCRAM_64K_SIZE_VALUE / sizeof(unsigned int); i++) {
98                  if(ptr[i] != i) {
99                          printf("FAILED.\n");
100                         return;
101                 }
102         }
103
104         printf("passed.\n");
105         printf("\n");
106 }
107
108 void demo_leds(void) {
109
110         unsigned int led = 0;
111
112         /* Blink the LEDs until any key is pressed */
113         printf("Blinking LEDs\n");
114         printf("Press any key to stop LED lightshow\n");
115         while(!tstc()) {
116
117                 /* Turn on LED */
118                 writel(1L << led, LED_PIO_BASE);
119
120                 /* Delay */
121                 udelay(LED_DELAY_US);
122
123                 /* Advance to next LED */
124                 led = (led + 1) % LED_PIO_DATA_WIDTH;
125         }
126
127         /* Discard the key that was pressed */
128         (void) getc();
129         printf("\n");
130 }
131
132 void demo_switches(void) {
133
134         unsigned int switches, prev_switches;
135
136         /* Display state of switches */
137         printf("Displaying the state of switches\n");
138         printf("Slide switch SW0, Sw1, SW2 or SW3 to observe updates\n");
139         printf("Press any key to stop displaying the switches\n");
140         prev_switches = ~readl(SWITCH_PIO_BASE);
141
142         while(!tstc()) {
143                 switches = readl(SWITCH_PIO_BASE);
144                 if(switches != prev_switches) {
145                         printf("Switch value: 0x%01x\n", (int)switches);
```

```
                        prev_switches = switches;
                }
        }

        /* Discard the key that was pressed */
        (void) getc();
        printf("\n");
}

void demo_buttons(void) {

        unsigned int buttons, prev_buttons;

        /* 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 = ~readl(BUTTON_PIO_BASE);

        while(!tstc()) {
                buttons = readl(BUTTON_PIO_BASE);
                if(buttons != prev_buttons) {
                        printf("Button value: 0x%01x\n", (int)buttons);
                        prev_buttons = buttons;
                }
        }

        /* Discard the key that was pressed */
        (void) getc();
        printf("\n");
}
```

**Step 5.** Edit the file **$DE10_NANO/uboot-socfpga/git/examples/standalone/Makefile** to add the **blink_app** to the list of applications to be compiled, by adding the line highlighted below:

```
extra-y         := hello_world
extra-y         += blink_app
extra-y         += de10_nano_hdmi_config
```

You can do this by opening the Makefile in a text editor, or you can apply the update using **sed** like this:

```
$ sed -i.bak -e "/.*hello_world/a extra-y\ \ \ \ \ \ \ \ += blink_app" \
> $DE10_NANO/uboot-socfpga/git/examples/standalone/Makefile
```

**Step 6.** Compile the U-Boot applications by running the commands below:

```
# change into the U-Boot top level source directory
$ cd $DE10_NANO/uboot-socfpga/git

# configure the CROSS_COMPILE environment variable to use the toolchain that
# we previously downloaded
$ export CROSS_COMPILE=$DE10_NANO/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/
$ export CROSS_COMPILE=${CROSS_COMPILE}bin/arm-linux-gnueabihf-

# build the U-Boot standalone examples
```

```
$ make socfpga_de10_nano_defconfig
$ make examples
```

This will build all the examples, including the **blink_app** we added, which will be accessible at **$DE10_NANO/uboot-socfpga/git/examples/standalone/blink_app.bin**.

**Step 7.** Use the microSD card created in the "Preparing the Terasic DE10-Nano Board" section above and copy the **blink_app.bin** file to it the way we copied the **blink.rbf** file. The instructions below assume manual mounting is needed:

```
$ cd $DE10_NANO
$ mkdir sdcard
$ sudo mount /dev/sdx1 sdcard
$ sudo cp $DE10_NANO/uboot-socfpga/git/examples/standalone/blink_app.bin sdcard/
$ sudo umount /dev/sdx1
$ sudo sync
```

**Step 8.** Boot into the U-Boot console, and configure the FPGA with the **blink.rbf** file as described in the "Preparing the Terasic DE10-Nano Board" section above.

**Step 9.** Load and execute the **blink_app** from the U-Boot console:

```
=> load mmc 0:1 0x0c100000 blink_app.bin
=> go 0x0C100001 hello world
```

**Step 10.** Interact with the application as instructed on the U-Boot console:

```
## Starting application at 0x0C100001 ...

Blink application started.

Number of command line parameters: 3
Command line parameter 1 = 0x0C100001
Command line parameter 2 = hello
Command line parameter 3 = world

U-Boot ABI version 9

SysID.id = 0xde10de10
SysID.timestamp = 0x5931e56f

Writing pattern into FPGA OCRAM.
Checking pattern from FPGA OCRAM: passed.

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.

## Application terminated, rc = 0x0
=>
```

That's it, you've interacted with the Qsys system using a custom standalone U-Boot application. Continue on to the "Interacting with FPGA Designs using Linux" tutorial if you would like to see how to do similar things from a Linux environment running on the HPS processor.