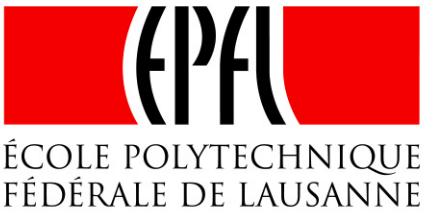
SoC-FPGA Design Guide

LAP – IC – EPFL

Version 1.08

Sahand Kashani-Akhavan

René Beuchat



1 TABLE OF CONTENTS

2 List of Figures 5
3 Table of Tables 7
4 Prerequisites 8
4.1 Software 8
4.2 Software Versions 8
5 Introduction 9
6 Terasic DE1-SoC Board 10
6.1 Specifications 10
6.1.1 FPGA Device 10
6.1.2 Configuration and Debug 10
6.1.3 Memory Device 10
6.1.4 Communication 10
6.1.5 Connectors 11
6.1.6 Display 11
6.1.7 Audio 11
6.1.8 Video Input 11
6.1.9 ADC 11
6.1.10 Switches, Buttons and Indicators 11
6.1.11 Sensors 11
6.1.12 Power 11
6.1.13 Block Diagram 12
6.2 Layout 12
7 Cyclone V Overview 14
7.1 Introduction to the Cyclone V Hard Processor System 14
7.2 Features of the HPS 16
7.3 System Integration Overview 17
7.3.1 MPU Subsystem 17
7.3.2 SDRAM Controller Subsystem 17
7.3.3 Support Peripherals 17
7.3.3.1 System Manager 17
7.3.3.2 FPGA Manager 17

7.3.4 Interface Peripherals 18
7.3.4.1 GPIO Interfaces 18
7.3.5 On-Chip Memory 18
7.3.5.1 On-Chip RAM 18
7.3.5.2 Boot ROM 18
7.4 HPS-FPGA Interfaces 18
7.5 HPS Address Map 18
7.5.1 HPS Address Spaces 18
7.5.2 HPS Peripheral Region Address Map 20
7.6 HPS Booting and FPGA Configuration 22
7.6.1 HPS Boot and FPGA Configuration Ordering 22
7.6.2 Zooming In On the HPS Boot Process 24
7.6.2.1 Preloader 25
8 Using the Cyclone V – General Information 26
8.1 Introduction 26
8.2 FPGA-only 26
8.3 HPS & FPGA 26
8.3.1 Bare-metal Application 26
8.3.2 Application Over an Operating System (Linux) 27
8.4 Goals 27
8.5 Project Structure 27
9 Using the Cyclone V – Hardware 29
9.1 General Quartus Prime Setup 29
9.2 System Design with Qsys – Nios II 29
9.3 System Design with Qsys – HPS 32
9.3.1 Instantiating the HPS Component 32
9.3.1.1 FPGA Interfaces Tab 32
9.3.1.2 Peripheral Pins Tab 33
9.3.1.2.1 Theory 33
9.3.1.2.2 Configuration 34
9.3.1.3 HPS Clocks Tab 35
9.3.1.4 SDRAM Tab 35
9.3.2 Interfacing with FPGA Peripherals 37
9.4 Generating the Qsys System 38

9.5 Instantiating the Qsys System 39
9.6 HPS DDR3 Pin Assignments 42
9.7 Programming the FPGA 43
9.8 Creating Target sdcard Artifacts 44
10 Using the Cyclone V – FPGA – Nios II – Bare-metal 45
10.1 Project Setup 45
10.2 Nios II Programming Theory – Accessing Peripherals 45
10.3 Nios II Programming Practice 46
11 Using the Cylone V – HPS – ARM – General 49
11.1 Partitioning the sdcard 49
11.2 Generating a Header File for HPS Peripherals 49
11.3 HPS Programming Theory 50
12 Using the Cyclone V – HPS – ARM – Bare-metal 52
12.1 Preloader 52
12.1.1 Preloader Generation 52
12.1.2 Creating Target sdcard Artifacts 53
12.2 ARM DS-5 53
12.2.1 Setting Up a New C Project 53
12.2.2 Writing a DS-5 Debug Script 55
12.2.3 Setting Up the Debug Configuration 56
12.2.4 Bare-metal Programming 57
12.2.4.1 Accessing FPGA Peripherals 58
12.2.4.2 Accessing HPS Peripherals 59
12.2.4.2.1 Using Altera's HWLIB - Prerequisites 59
12.2.4.2.2 Global Timer & Clock Manager 60
12.2.4.2.3 GPIO 61
12.2.4.3 Launching the Bare-metal Code in the Debugger 62
12.2.4.4 DS-5 Bare-metal Debugger Tour 63
12.2.4.4.1 "Registers" View 63
12.2.4.4.2 App Console 65
13 Using the Cyclone V – HPS – ARM – Linux 66
13.1.1 Setting Up a New C Project 66
13.1.2 Creating a Remote Debug Connection to the Linux Distribution 67
13.1.2.1 Find the Linux Distribution's IP Address 67

13.1.2.2 Create an SSH Remote Connection 69
13.1.2.3 Setting Up the Debug Configuration 70
13.1.3 Linux Programming 71
13.1.3.1 Using Altera's HWLIB - Prerequisites 73
13.1.3.2 Accessing Hardware Peripherals from User Space 73
13.1.3.2.1 Opening the Physical Memory File Descriptor 73
13.1.3.2.2 Accessing HPS Peripherals 73
13.1.3.2.3 Accessing FPGA Peripherals 75
13.1.3.2.4 Cleaning Up Before Application Exit 76
13.1.3.3 Launching the Linux code in the Debugger 76
13.1.3.4 App Console 77
13.1.3.5 DS-5 Linux Debugger Restrictions 78
14 TODO 79
15 Appendix 80
15.1 DE1-SoC Top-level VHDL Entity 80
15.2 DE1-SoC Pin Assignment TCL script 83
16 References 104

2 LIST OF FIGURES

Figure 6-1. Terasic DE1-SoC Board [1] 10
Figure 6-2. Block Diagram of the DE1-SoC Board [1] 12
Figure 6-3. Back [1] 12
Figure 6-4. Front [1] 13
Figure 7-1. Altera SoC FPGA Device Block Diagram [2, pp. 1-1] 14
Figure 7-2. HPS Block Diagram [2, pp. 1-3] 16
Figure 7-3. HPS Address Space Relations [2, pp. 1-14] 19
Figure 7-4. Simplified HPS Boot Flow [2, pp. A-3] 22
Figure 7-5. Independent FPGA Configuration and HPS Booting [2, pp. A-2] 23
Figure 7-6. FPGA Configuration before HPS Booting (HPS boots from FPGA) [2, pp. A-2] 23
Figure 7-7. HPS Boots and Performs FPGA Configuration [2, pp. A-3] 24
Figure 7-8. HPS Boot Flows [2, pp. A-3] 24
Figure 8-1. Project Folder Structure 28
Figure 9-1. Exporting the pll_0.outclk2 Signal 30
Figure 9-2. Basic Nios II System with SDRAM and JTAG UART 31
Figure 9-3. Adding LEDs and Switches to the System 32
Figure 9-4. HPS Component Parameters 32
Figure 9-5. HPS_KEY & HPS_LED on DE1-SoC Schematic 33
Figure 9-6. HPS_KEY & HPS_LED on Qsys Peripheral Pins Tab 33
Figure 9-7. Using Pin G21 for SPI 34
Figure 9-8. Ethernet MAC configuration 34
Figure 9-9. SD/MMC configuration 34
Figure 9-10. UART configuration 35
Figure 9-11. Exported peripheral pins 35
Figure 9-12. Adding the "Standalone" HPS to the System 37
Figure 9-13. Adding Buttons and 7-segment Displays to the Lightweight HPS-to-FPGA Bridge 38
Figure 9-14. Generate Qsys System 39
Figure 9-15. Qsys Component Instantiation 40
Figure 9-16. Final Top-level Entity 42
Figure 9-17. Correct HPS DDR3 Pin Assignment TCL Script Selection 42
Figure 9-18. Quartus Prime Programmer 43

Figure 9-19. FPGA Selection 43
Figure 9-20. JTAG Scan Chain 43
Figure 9-21. Programming the FPGA 44
Figure 10-1. Incorrect Nios II Peripheral Access in C 45
Figure 10-2. Correct Nios II Peripheral Access in C 46
Figure 10-3. nios.c 47
Figure 10-4. Nios II Target Connection Dialog 47
Figure 11-1. Partitioning the sdcard 49
Figure 11-2. hps_soc_system.h 50
Figure 12-1. New BSP Dialog 52
Figure 12-2. Preloader Settings Dialog 53
Figure 12-3. New C Project Dialog 54
Figure 12-4. debug_setup.ds 56
Figure 12-5. Debug Configuraton “Connection” Tab 56
Figure 12-6. Debug Configuration "Files" Tab 57
Figure 12-7. Debug Configuration "Debugger" Tab 57
Figure 12-8. hps_baremetal.c main() function 58
Figure 12-9. Accessing FPGA Buttons from the HPS 58
Figure 12-10. Setting the 7-Segment Displays from the HPS 59
Figure 12-11. Programming the HPS Global Timer 60
Figure 12-12. Programming the HPS GPIO Peripheral 62
Figure 12-13. Switching to the DS-5 Debug Perspective 62
Figure 12-14. Debug Control View 62
Figure 12-15. DS-5 Debugger Controls 63
Figure 12-16. DS-5 Debugger Registers View 64
Figure 12-17. DS-5 App Console View 65
Figure 13-1. New C Project Dialog 66
Figure 13-2. hps_linux.c with an empty main() function. 67
Figure 13-3. ARM DS-5 Serial Terminal 67
Figure 13-4. ARM DS-5 Serial Terminal Settings (Windows settings on the left, Linux settings on the right) 68
Figure 13-5. ARM DS-5 Serial Terminal Linux Prompt 68
Figure 13-6. Obtaining the DE1-SoC's IP Address through ARM DS-5’s Serial Terminal 69
Figure 13-7. New SSH Only Connection 69

Figure 13-8. New SSH Connection In "Remote Systems" View 69
Figure 13-9. Debug Configuraton “Connection” Tab 70
Figure 13-10. Debug Configuration "Files" Tab 71
Figure 13-11. Debug Configuration "Debugger" Tab 71
Figure 13-12. hps_linux.c main() Function 72
Figure 13-13. Prototype of the mmap() Function 73
Figure 13-14. open_physical_memory_device() Function 73
Figure 13-15. mmap_hps_peripherals() Function 74
Figure 13-16. setup_hps_gpio() Function 74
Figure 13-17. handle_hps_led() Function 74
Figure 13-18. mmap_fpga_peripherals() Function. 75
Figure 13-19. is_fpga_button_pressed() Function 76
Figure 13-20. munmap_hps_peripherals() Function 76
Figure 13-21. close_physical_memory_device() Function 76
Figure 13-22. Switching to the DS-5 Debug Perspective 76
Figure 13-23. Debug Control View 77
Figure 13-24. DS-5 Debugger Controls 77
Figure 13-25. DS-5 App Console View 77
Figure 15-1. DE1-SoC Top-level VHDL Entity 82
Figure 15-2. DE1-SoC Pin Assignment TCL Script 103

3 TABLE OF TABLES

Table 7-1. Possible HPS and FPGA Power Configurations 15
Table 7-2. HPS Address Spaces [2, pp. 1-13] 19
Table 7-3. Common Address Space Regions [2, pp. 1-15] 19
Table 7-4. HPS Peripheral Region Address Map [2, pp. 1-16] 21
Table 11-1. Predefined Data Sizes in socal.h 50

4 PREREQUISITES

4.1 SOFTWARE

This guide assumes users know how to use the following programs

- Quartus Prime
- Nios II Software Build Tools
- Qsys
- ModelSim-Altera

All command-line examples provided in this guide must be executed in a ***SOC EDS EMBEDDED COMMAND SHELL***. The executable for the *SoC EDS Embedded Command Shell* can be found at "`<altera_install_directory>/<version>/embedded/embedded_command_shell.sh`"

4.2 SOFTWARE VERSIONS

- All ***HARDWARE*** and ***SOFTWARE*** examples in this guide were made with *Quartus Prime*, *SoC EDS* and *Nios II SBT* version ***15.1***.
- Some ***FIGURES*** in this guide were made with *Quartus Prime*, *SoC EDS* and *Nios II SBT* version ***14.0***.
- The operating system used is ***KUBUNTU 15.10***.

5 INTRODUCTION

The development of embedded systems based on chips containing one or more microprocessors and hardcore peripherals, as well as an FPGA part is becoming more and more important. This technology gives the designer a lot of freedom and powerful abilities. Classical design flows with microcontrollers are emphasized with the full power of FPGAs.

Mixed designs are becoming a reality. One can now design specific accelerators to greatly improve algorithms, or create specific programmable interfaces with the external world.

Two main HDL (**H**ardware **D**esign **L**anguage) languages are available for the design of the FPGA part: **VHDL** and Verilog. There also exist other tools that perform automatic translations from C to HDL. New emerging technologies like OpenCL allow compatibility between high-level software design, and low-level hardware implementations such as:

- Compilation for single or multicore processors
- Compilation for GPUs (Graphical Processing Unit)
- Translation and compilation for FPGAs. The latest models use a PCIe interface or some other way of parameters passing between the main processor and the FPGA

We will introduce and use the Terasic DE1-SoC board, as well as the *ARM DS-5* IDE.

6 TERASIC DE1-SOC BOARD

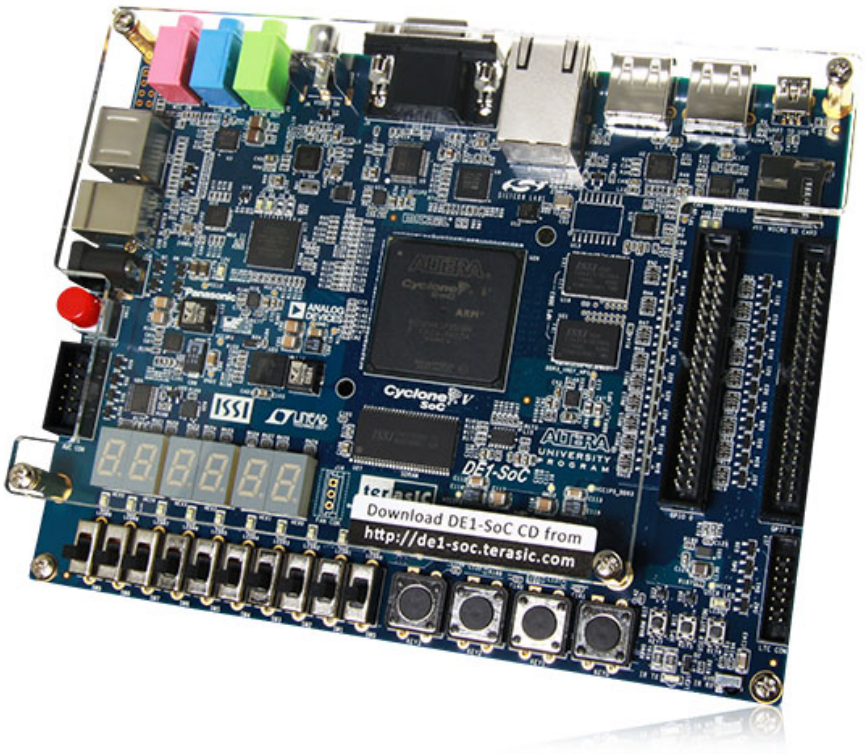


Figure 6-1. Terasic DE1-SoC Board [1]

The DE1-SoC board has many features that allow users to implement a wide range of designed circuits. We will discuss some noteworthy features in this guide.

6.1 SPECIFICATIONS

6.1.1 FPGA Device

- Cyclone V SoC ***5CSEMA5F31C6*** Device
- Dual-core ***ARM CORTEX-A9*** (HPS)
- ***85K*** Programmable Logic Elements
- 4'450 Kbits embedded memory
- 6 Fractional PLLs
- 2 Hard Memory Controllers (only seems to be used for the HPS DDR3 SDRAM, not the FPGA SDRAM)

6.1.2 Configuration and Debug

- Quad Serial Configuration device – ***EPCQ256*** on FPGA
- On-Board ***USB BLASTER II*** (Normal type B USB connector)

6.1.3 Memory Device

- ***64 MB*** (32Mx16) SDRAM on FPGA
- ***1 GB*** (2x256Mx16) DDR3 SDRAM on HPS
- ***MICRO SD*** Card Socket on HPS

6.1.4 Communication

- Two Port USB 2.0 Host (ULPI interface with USB type A connector)
- USB to UART (micro USB type B connector)
- 10/100/1000 Ethernet
- PS/2 mouse/keyboard
- IR Emitter/Receiver

6.1.5 Connectors

- Two 40-pin Expansion Headers
- One 10-pin ADC Input Header
- One LTC connector (One Serial Peripheral Interface (SPI) Master, one I2C and one GPIO interface)

6.1.6 Display

- 24-bit VGA DAC

6.1.7 Audio

- 24-bit CODEC, line-in, line-out, and microphone-in jacks

6.1.8 Video Input

- TV Decoder (NTSC/PAL/SECAM) and TV-in connector

6.1.9 ADC

- Fast throughput rate: 1 MSPS
- Channel number: 8
- Resolution: 12 bits
- Analog input range : 0 ~ 2.5 V or 0 ~ 5V as selected via the RANGE bit in the control register

6.1.10 Switches, Buttons and Indicators

- 4 User Keys (FPGA x4)
- 10 User switches (FPGA x10)
- 11 User LEDs (FPGA x10; HPS x 1)
- 2 HPS Reset Buttons (HPS_RST_n and HPS_WARM_RST_n)
- Six 7-segment displays

6.1.11 Sensors

- G-Sensor on HPS

6.1.12 Power

- 12V DC input

6.1.13 Block Diagram

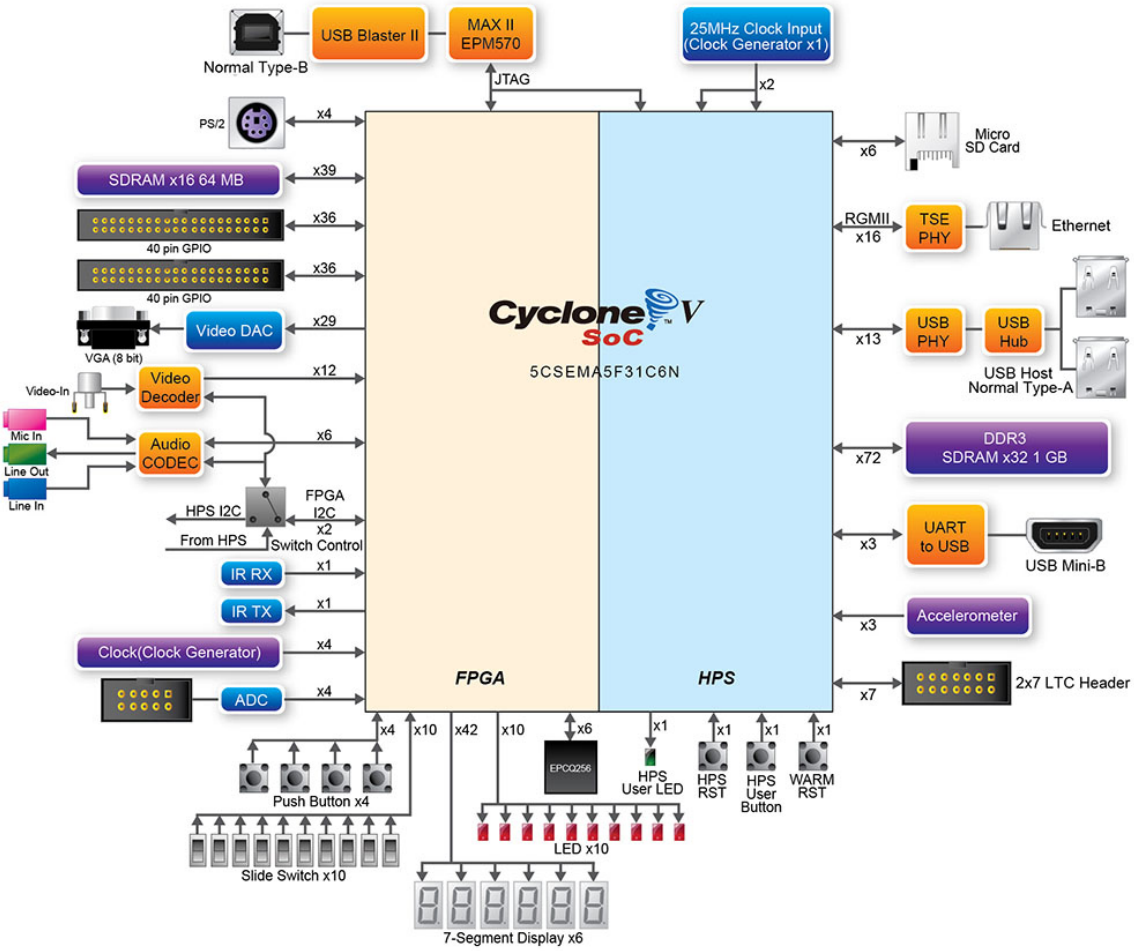


Figure 6-2. Block Diagram of the DE1-SoC Board [1]

6.2 LAYOUT

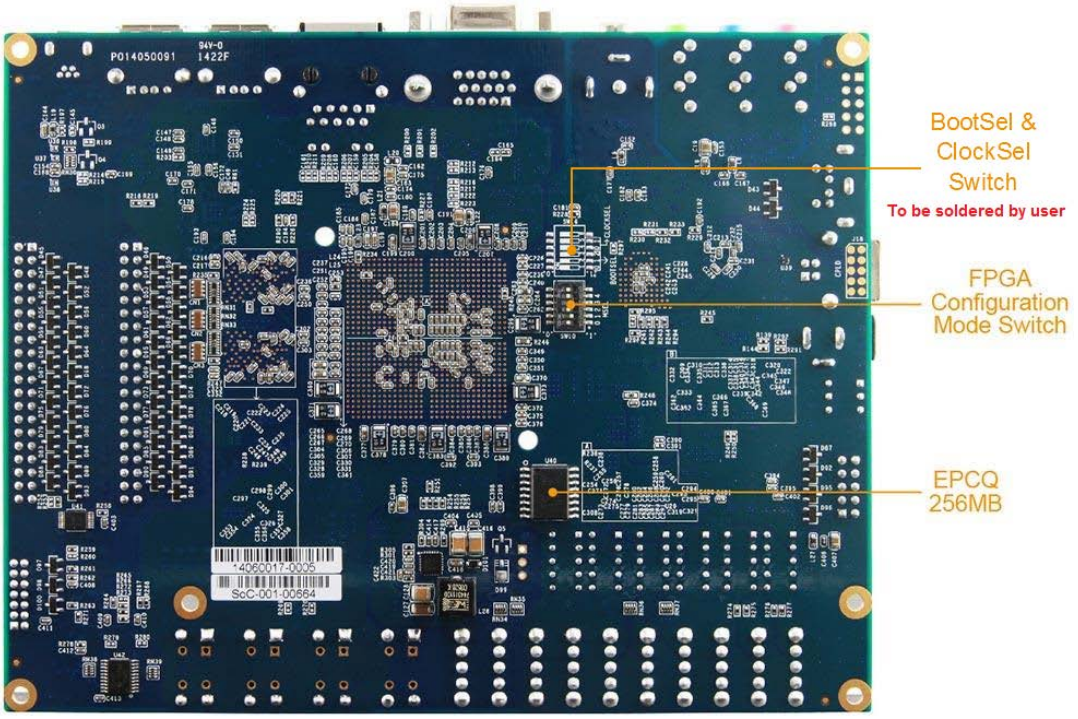


Figure 6-3. Back [1]

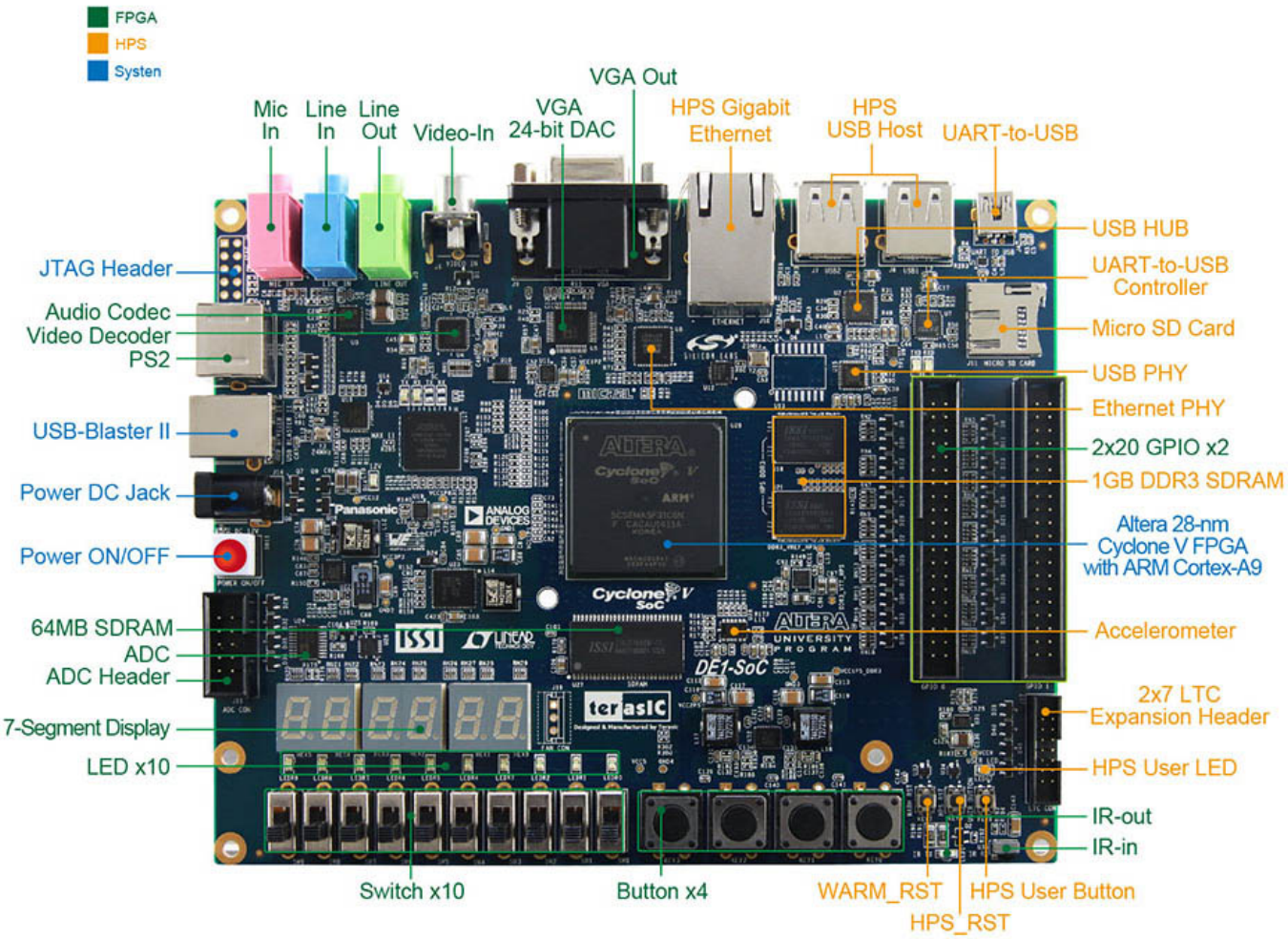


Figure 6-4. Front [1]

- Green for peripherals directly connected to the FPGA
- Orange for peripherals directly connected to the HPS
- Blue for board control

7 CYCLONE V OVERVIEW

This section describes some features of the Cyclone V family of devices. We do not list all features, but only the ones most important to us. All information below, along with the most complete documentation regarding this family can be found in the Cyclone V Device Handbook [2].

7.1 INTRODUCTION TO THE CYCLONE V HARD PROCESSOR SYSTEM

The Cyclone V device is a single-die system on a chip (SoC) that consists of two distinct parts – a hard processor system (HPS) portion and an FPGA portion.

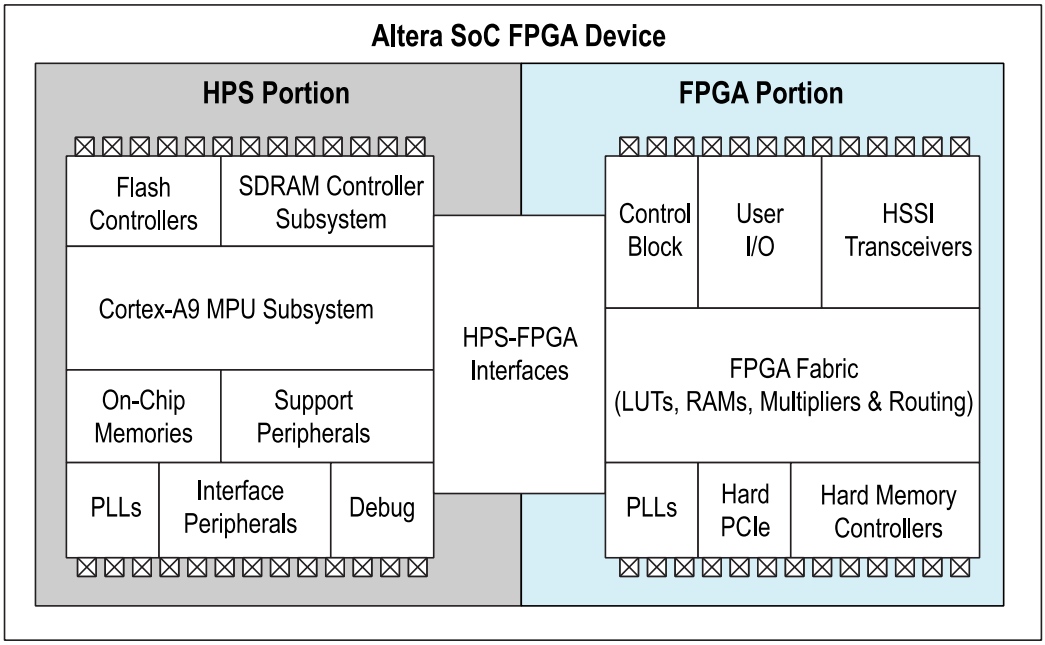


Figure 7-1. Altera SoC FPGA Device Block Diagram [2, pp. 1-1]

The HPS contains a microprocessor unit (MPU) subsystem with single or dual ARM Cortex-A9 MPCore processors, flash memory controllers, SDRAM L3 Interconnect, on-chip memories, support peripherals, interface peripherals, debug capabilities, and phase-locked loops (PLLs). The dual-processor HPS supports symmetric (SMP) and asymmetric (AMP) multiprocessing.

*The DE1-SoC has a **DUAL**-processor HPS.*

The FPGA portion of the device contains the FPGA fabric, a control block (CB), phase-locked loops (PLLs), and depending on the device variant, high-speed serial interface (HSSI) transceivers, hard PCI Express (PCIe) controllers, and hard memory controllers.

The DE1-SoC does not contain any HSSI transceivers, or hard PCIe controllers.

The HPS and FPGA portions of the device are distinctly different. The HPS can boot from

- the FPGA fabric,
- external flash, or
- JTAG

In contrast, the FPGA must be configured either through

- the HPS, or
- an externally supported device such as the *Quartus Prime* programmer.

The MPU subsystem can boot from

- flash devices connected to the HPS pins, or
- from memory available on the FPGA portion of the device (when the FPGA portion is previously configured by an external source).

The HPS and FPGA portions of the device each have their own pins. Pins are not freely shared between the HPS and the FPGA fabric. The ***FPGA I/O PINS*** are configured by an ***FPGA CONFIGURATION IMAGE*** through the HPS or any external source supported by the device. The ***HPS I/O PINS*** are configured by ***SOFTWARE*** executing in the HPS. Software executing on the HPS accesses control registers in the Cyclone V system manager to assign HPS I/O pins to the available HPS modules.

*The **SOFTWARE** that configures the **HPS I/O PINS** is called the **PRELOADER**.*

The HPS and FPGA portions of the device have separate external power supplies and independently power on. You can power on the HPS without powering on the FPGA portion of the device. However, to power on the FPGA portion, the HPS must already be on or powered on at the same time as the FPGA portion. Table 7-1 summarizes the possible configurations.

HPS Power	FPGA Power
On	On
On	Off
Off	Off

Table 7-1. Possible HPS and FPGA Power Configurations

7.2 FEATURES OF THE HPS

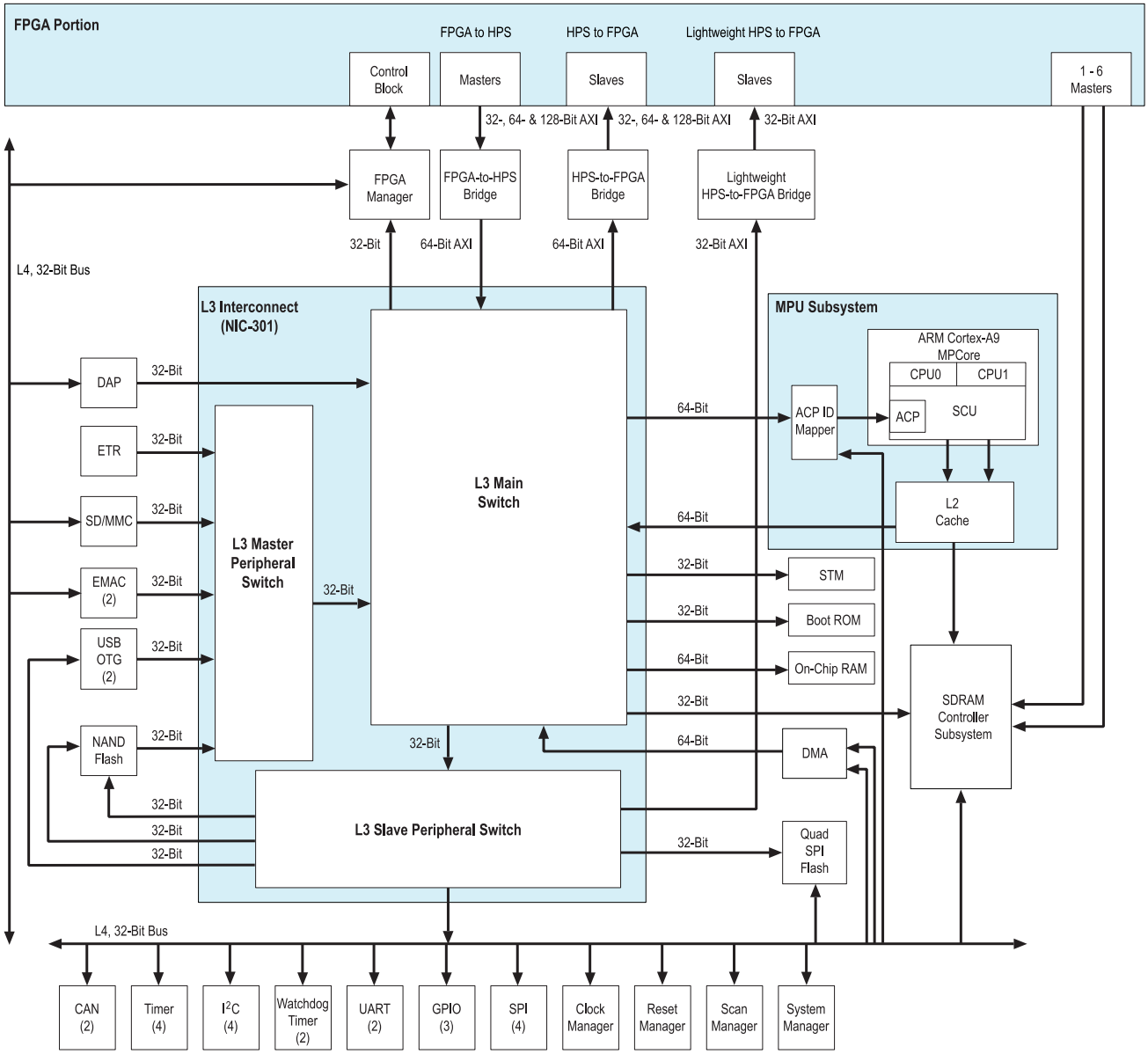


Figure 7-2. HPS Block Diagram [2, pp. 1-3]

The following list contains the main modules of the HPS:

- Masters
  - MPU subsystem featuring dual ARM Cortex-A9 MPCore processors
  - General-purpose Direct Memory Access (DMA) controller
  - Two Ethernet media access controllers (EMACs)
  - Two USB 2.0 On-The-Go (OTG) controllers
  - NAND flash controller
  - Secure Digital (SD) / MultiMediaCard (MMC) controller
  - Two serial peripheral interface (SPI) master controllers
  - ARM CoreSight debug components
- Slaves
  - Quad SPI flash controller
  - Two SPI slave controllers
  - Four inter-integrated circuit (I^2C) controllers
  - 64 KB on-chip RAM

- 64 KB on-chip boot ROM
- Two UARTs
- Four timers
- Two watchdog timers
- Three general-purpose I/O (GPIO) interfaces
- Two controller area network (CAN) controllers
- System manager
- Clock manager
- Reset manager
- Scan manager
- FPGA manager

7.3 SYSTEM INTEGRATION OVERVIEW

In this part, we briefly go through *some* features provided by the most important HPS components.

7.3.1 MPU Subsystem

Here are a few important features of the MPU subsystem:

- Interrupt controller
- One general-purpose timer and one watchdog timer per processor
- One Memory management unit (MMU) per processor

The HPS masters the L3 interconnect and the SDRAM controller subsystem.

7.3.2 SDRAM Controller Subsystem

The SDRAM controller subsystem is ***MASTERED*** by ***HPS MASTERS*** and ***FPGA FABRIC MASTERS***. It supports DDR2, DDR3, and LPDDR2 devices. It is composed of 2 parts:

- SDRAM controller
- DDR PHY (interfaces the single port memory controller to the HPS I/O)

*The DE1-SoC contains DDR**3** SDRAM*

7.3.3 Support Peripherals

7.3.3.1 System Manager

This is one of the most *essential* HPS components. It offers a few important features:

- ***PIN MULTIPLEXING*** (term used for the ***SOFTWARE*** configuration of the ***HPS I/O PINS*** by the ***PRELOADER***)
- Freeze controller that places I/O elements into a safe state for configuration
- Low-level control of peripheral features not accessible through the control and status registers (CSRs)

*The low-level control of some peripheral features that are not accessible through the CSRs is **NOT** externally documented. You will see this type of code when you generate your custom preloader, but must **NOT** use the constructs in your own code.*

7.3.3.2 FPGA Manager

The FPGA manager offers the following features:

- Manages the configuration of the FPGA portion of the device
- Monitors configuration-related signals in the FPGA
- Provides 32 general-purpose inputs and 32 general-purpose outputs to the FPGA fabric

7.3.4 Interface Peripherals

7.3.4.1 GPIO Interfaces

The HPS provides three GPIO interfaces and offer the following features:

- Supports digital de-bounce
- Configurable interrupt mode
- Supports up to 71 I/O pins and 14 input-only pins, based on device variant
- Supports up to 67 I/O pins and 14 input-only pins

The DE1-SoC has 67 I/O pins and 14 input-only pins

7.3.5 On-Chip Memory

*The following on-chip memories are **DIFFERENT** from any on-chip memories located in the FPGA fabric.*

7.3.5.1 On-Chip RAM

The on-chip RAM offers the following features:

- 64 KB size
- High performance for all burst lengths

7.3.5.2 Boot ROM

The boot ROM offers the following features:

- 64 KB size
- Contains the code required to support HPS boot from cold or warm reset
- Used ***EXCLUSIVELY*** for booting the HPS

*The code in the boot ROM **CANNOT** be changed.*

7.4 HPS-FPGA Interfaces

The HPS-FPGA interfaces provide a variety of communication channels between the HPS and the FPGA fabric. The HPS-FPGA interfaces include:

- **FPGA-to-HPS bridge** – a high performance bus with a configurable data width of 32, 64, or 128 bits. It allows the FPGA fabric to master transactions to slaves in the HPS. This interface allows the FPGA fabric to have full visibility into the HPS address space.
- **HPS-to-FPGA bridge** – a high performance bus with a configurable data width of 32, 64, or 128 bits. It allows the HPS to master transactions to slaves in the FPGA fabric. I will sometimes call this the "*heavyweight*" HPS-to-FPGA bridge to distinguish its "*lightweight*" counterpart (see below).
- **Lightweight HPS-to-FPGA bridge** – a bus with a 32-bit fixed data width. It allows the HPS to master transactions to slaves in the FPGA fabric.
- **FPGA manager interface** – signals that communicate with FPGA fabric for boot and configuration.
- **Interrupts** – allow soft IP to supply interrupts directly to the MPU interrupt controller.
- **HPS debug interface** – an interface that allows the HPS debug control domain to extend into the FPGA.

7.5 HPS Address Map

7.5.1 HPS Address Spaces

The HPS address map specifies the address of slaves, such as memory and peripherals, as viewed by the HPS masters. The HPS has 3 address spaces:

Name	Description	Size
MPU	MPU subsystem	4 GB
L3	L3 interconnect	4 GB
SDRAM	SDRAM controller subsystem	4 GB

Table 7-2. HPS Address Spaces [2, pp. 1-13]

The following figure shows the relationships between the different HPS address spaces. The figure is ***NOT*** to scale.

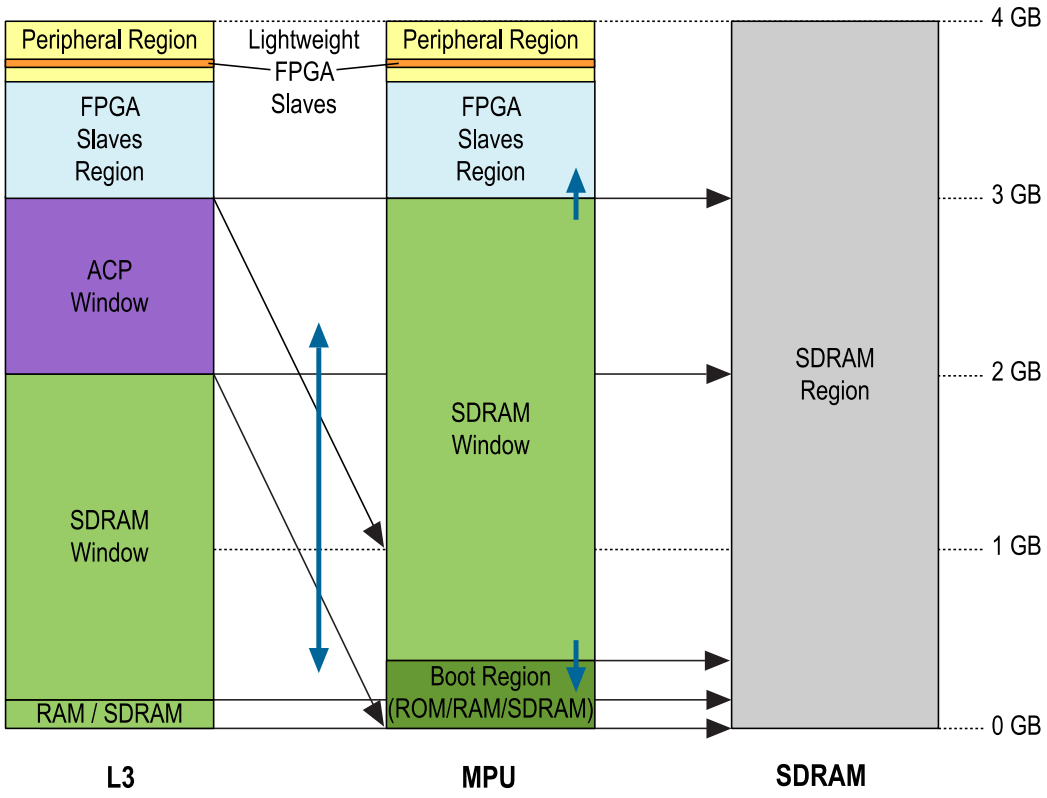


Figure 7-3. HPS Address Space Relations [2, pp. 1-14]

The window regions provide access to other address spaces. The thin black arrows indicate which address space is accessed by a window region (arrows point to accessed address space).

The SDRAM window in the MPU can grow and shrink at the top and bottom (short blue vertical arrows) at the expense of the FPGA slaves and boot regions. The ACP window can be mapped to any 1 GB region in the MPU address space (blue vertical bidirectional arrow), on gigabyte-aligned boundaries.

The following table shows the base address and size of each region that is common to the L3 and MPU address spaces.

Region Name	Description	Base Address	Size
FPGA slaves	FPGA slaves connected to the HPS-to-FPGA bridge	0xC0000000	960 MB
HPS peripherals	Slaves directly connected to the HPS (corresponds to all orange colored elements on Figure 6-4 and Figure 6-3)	0xFC000000	64 MB
Lightweight FPGA slaves	FPGA slaves connected to the lightweight HPS-to-FPGA bridge	0xFF200000	2 MB

Table 7-3. Common Address Space Regions [2, pp. 1-15]

7.5.2 HPS Peripheral Region Address Map

The following table lists the slave identifier, slave title, base address, and size of each slave in the HPS peripheral region. The *Slave Identifier* column lists the names used in the HPS register map file provided by Altera (more on this later).

Slave Identifier	Slave Title	Base Address	Size
STM	STM	0xFC000000	48 MB
DAP	DAP	0xFF000000	2 MB
LWFPGASLAVES	FPGA slaves accessed with lightweight HPS-to-FPGA bridge	0xFF200000	2 MB
LWHPS2FPGAREGS	Lightweight HPS-to-FPGA bridge GPV	0xFF400000	1 MB
HPS2FPGAREGS	HPS-to-FPGA bridge GPV	0xFF500000	1 MB
FPGA2HPSREGS	FPGA-to-HPS bridge GPV	0xFF600000	1 MB
EMAC0	EMAC0	0xFF700000	8 KB
EMAC1	EMAC1	0xFF702000	8 KB
SDMMC	SD/MMC	0xFF704000	4 KB
QSPIREGS	Quad SPI flash controller registers	0xFF705000	4 KB
FPGAMGRREGS	FPGA manager registers	0xFF706000	4 KB
ACPIDMAP	ACP ID mapper registers	0xFF707000	4 KB
GPIO0	GPIO0	0xFF708000	4 KB
GPIO1	GPIO1	0xFF709000	4 KB
GPIO2	GPIO2	0xFF70A000	4 KB
L3REGS	L3 interconnect GPV	0xFF800000	1 MB
NANDDATA	NAND controller data	0xFF900000	1 MB
QSPIDATA	Quad SPI flash data	0xFFA00000	1 MB
USB0	USB0 OTG controller registers	0xFFB00000	256 KB
USB1	USB1 OTG controller registers	0xFFB40000	256 KB
NANDREGS	NAND controller registers	0xFFB80000	64 KB
FPGAMGRDATA	FPGA manager configuration data	0xFFB90000	4 KB
CAN0	CAN0 controller registers	0xFFC00000	4 KB
CAN1	CAN1 controller registers	0xFFC01000	4 KB
UART0	UART0	0xFFC02000	4 KB
UART1	UART1	0xFFC03000	4 KB
I2C0	I2C0	0xFFC04000	4 KB
I2C1	I2C1	0xFFC05000	4 KB
I2C2	I2C2	0xFFC06000	4 KB
I2C3	I2C3	0xFFC07000	4 KB
SPTIMER0	SP Timer0	0xFFC08000	4 KB
SPTIMER1	SP Timer1	0xFFC09000	4 KB
SDRREGS	SDRAM controller subsystem registers	0xFFC20000	128 KB
OSC1TIMER0	OSC1 Timer0	0xFFD00000	4 KB
OSC1TIMER1	OSC1 Timer1	0xFFD01000	4 KB
L4WD0	Watchdog0	0xFFD02000	4 KB
L4WD1	Watchdog1	0xFFD03000	4 KB
CLKMGR	Clock manager	0xFFD04000	4 KB
RSTMGR	Reset manager	0xFFD05000	4 KB
SYSMGR	System manager	0xFFD08000	16 KB
DMANONSECURE	DMA nonsecure registers	0xFFE00000	4 KB
DMASECURE	DMA secure registers	0xFFE01000	4 KB
SPIS0	SPI slave0	0xFFE02000	4 KB
SPIS1	SPI slave1	0xFFE03000	4 KB
SPIM0	SPI master0	0xFFF00000	4 KB
SPIM1	SPI master1	0xFFF01000	4 KB

SCANMGR	Scan manager registers	0xFFF02000	4 KB
ROM	Boot ROM	0xFFFD0000	64 KB
MPUSCU	MPU SCU registers	0xFFFEC000	8 KB
MPUL2	MPU L2 cache controller registers	0xFFFEF000	4 KB
OCRAM	On-chip RAM	0xFFFF0000	64 KB

Table 7-4. HPS Peripheral Region Address Map [2, pp. 1-16]

The programming model for accessing the HPS peripherals in Table 7-4 is the same as for peripherals created on the FPGA fabric. That is, every peripheral has a base address at which a certain number of registers can be found. You can then read and write to a certain set of these registers in order to modify the peripheral's behavior.

When using a HPS peripheral in
Table 7-4, you do not need to hard-code any base address or peripheral register map in your programs, as Altera provides a header file for each one.

Three directories contain all ***HPS***-related ***HEADER FILES***:

1. "<altera_install_directory>/<version>/embedded/ip/altera/hps/altera_hps/hwlib/include"
   Contains ***HIGH-LEVEL*** header files that typically contain a few ***FUNCTIONS*** which facilitate control over the HPS components. These functions are all part of Altera's ***HWLIB***, which was created to make programming the HPS easier. This directory contains code that is common to the Cyclone V, Arria V, and Arria 10 devices.

2. "<altera_install_directory>/<version>/embedded/ip/altera/hps/altera_hps/hwlib/include/soc_cv_av"
   Same as above, but more specifically for the Cyclone V and Arria V FPGA families.

3. "<altera_install_directory>/<version>/embedded/ip/altera/hps/altera_hps/hwlib/include/soc_cv_av/socal"
   Contains ***LOW-LEVEL*** header files that provide a peripheral's ***BIT-LEVEL REGISTER DETAILS***. For example, any bits in a peripheral's register that correspond to undefined behavior will be specified in these header files.

To illustrate the differences among the high and low-level header files, we can compare the ones related to the FPGA manager peripheral:

1. "…/hwlib/include/soc_cv_av/alt_fpga_manager.h"

```
ALT_STATUS_CODE alt_fpga_reset_assert(void);
ALT_STATUS_CODE alt_fpga_configure(const void* cfg_buf, size_t cfg_buf_len);
```

2. "…/hwlib/include/soc_cv_av/socal/alt_fpgamgr.h"

```
/* The width in bits of the ALT_FPGAMGR_CTL_EN register field. */
#define ALT_FPGAMGR_CTL_EN_WIDTH      1
/* The mask used to set the ALT_FPGAMGR_CTL_EN register field value. */
#define ALT_FPGAMGR_CTL_EN_SET_MSK    0x00000001
/* The mask used to clear the ALT_FPGAMGR_CTL_EN register field value. */
#define ALT_FPGAMGR_CTL_EN_CLR_MSK    0xfffffffe
```

An *important* header file is "…/hwlib/include/soc_cv_av/socal/hps.h". It contains the HPS component's full ***REGISTER MAP***, as provided in
Table 7-4.

Note however, that there exists ***NO HEADER FILE*** for the "*heavyweight*" HPS-to-FPGA bridge, as it is not located in the "HPS peripherals" region in Figure 7-3. Indeed, the "*heavyweight*" HPS-to-FPGA bridge is not

considered a HPS peripheral, whereas the *"lightweight"* HPS-to-FPGA bridge is. Therefore, in order to use the *"heavyweight"* HPS-to-FPGA bridge, you will have to define a macro in your code, as follows:

```
#define ALT_HWFPGASLVS_OFST    0xc0000000
```

*Note that HWLIB can only be directly used in a **BARE-METAL APPLICATION**, as it directly references physical addresses. The library can unfortunately **NOT** be used directly in a **LINUX DEVICE DRIVER**, because it uses standard header files that are not available in the kernel. Needless to say that a userspace linux program cannot use the library either, as the linux kernel would terminate a user process that tries to access any of these physical addresses directly.*

7.6 HPS Booting and FPGA Configuration

Before being able to use the Cyclone V SoC, one needs to understand how the HPS boots and how the FPGA is configured. We'll first take a look at the ordering between the HPS and FPGA.

7.6.1 HPS Boot and FPGA Configuration Ordering

The ***HPS BOOT*** starts when the processor is released from reset (for example, on power up) and executes code in the internal *boot ROM* at the reset exception address. The boot process ends when the code in the boot ROM jumps to the next stage of the boot software. This next stage of the boot software is referred to as the *preloader*. Figure 7-4 illustrates this *initial* incomplete HPS boot flow.

Figure 7-4. Simplified HPS Boot Flow [2, pp. A-3]

The processor can boot from the following sources:

- NAND flash memory through the NAND flash controller
- SD/MMC flash memory through the SD/MMC flash controller
- SPI and QSPI flash memory through the QSPI flash controller using *Slave Select 0*
- FPGA fabric on-chip memory

The choice of the boot source is done by modifying the *BOOTSEL* and *CLKSEL* values ***BEFORE THE DEVICE IS POWERED UP***. Therefore, the Cyclone V device normally uses a ***PHYSICAL DIP SWITCH*** to configure the *BOOTSEL* and *CLKSEL*.

*The DE1-SoC can **ONLY BOOT** from **SD/MMC** flash memory, as its BOOTSEL and CLKSEL values are hard-wired on the board. Although its HPS contains all necessary controllers, the board doesn't have a physical DIP switch to modify the BOOTSEL and CLKSEL values. The actual location of the DIP switch is present underneath the board, as can be seen in Figure 6-3, but a switch isn't soldered.*

CONFIGURATION OF THE FPGA portion of the device starts when the FPGA portion is released from reset state (for example, on power up). The control block (CB) in the FPGA portion of the device is responsible for obtaining an FPGA configuration image and configuring the FPGA. The FPGA configuration ends when the configuration image has been fully loaded and the FPGA enters user mode. The FPGA configuration image is provided by users and is typically stored in non-volatile flash-based memory. The FPGA CB can obtain a configuration image from the HPS through the FPGA manager, or from another external source, such as the *Quartus Prime Programmer*.

The following three figures illustrate the possible HPS boot and FPGA configuration schemes. Note that Cyclone V devices can also be fully configured through a JTAG connection.

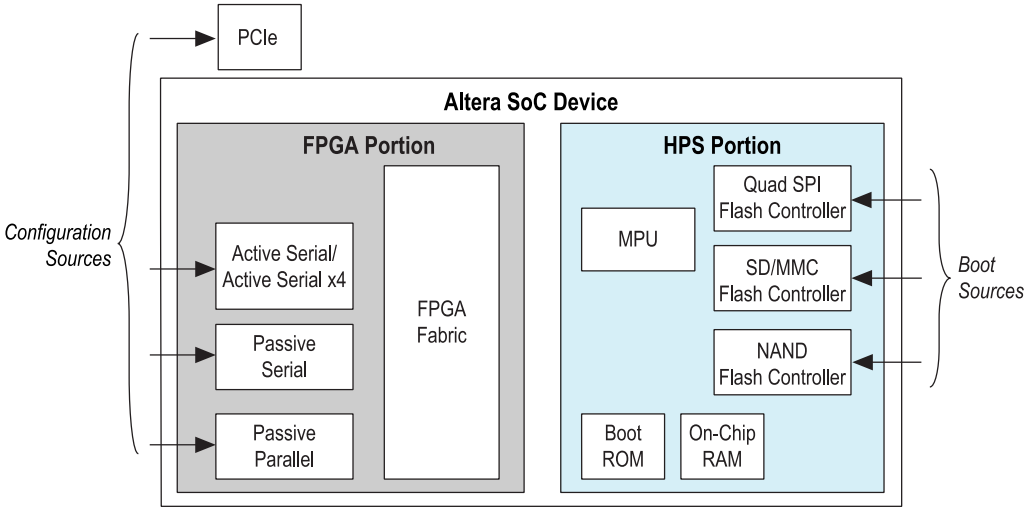


Figure 7-5. Independent FPGA Configuration and HPS Booting [2, pp. A-2]

Figure 7-5 shows the scheme where the FPGA configuration and the HPS boot occur independently. The FPGA configuration obtains its image from a non-HPS source (*Quartus Prime Programmer*), while the HPS boot obtains its configuration image from a non-FPGA fabric source.

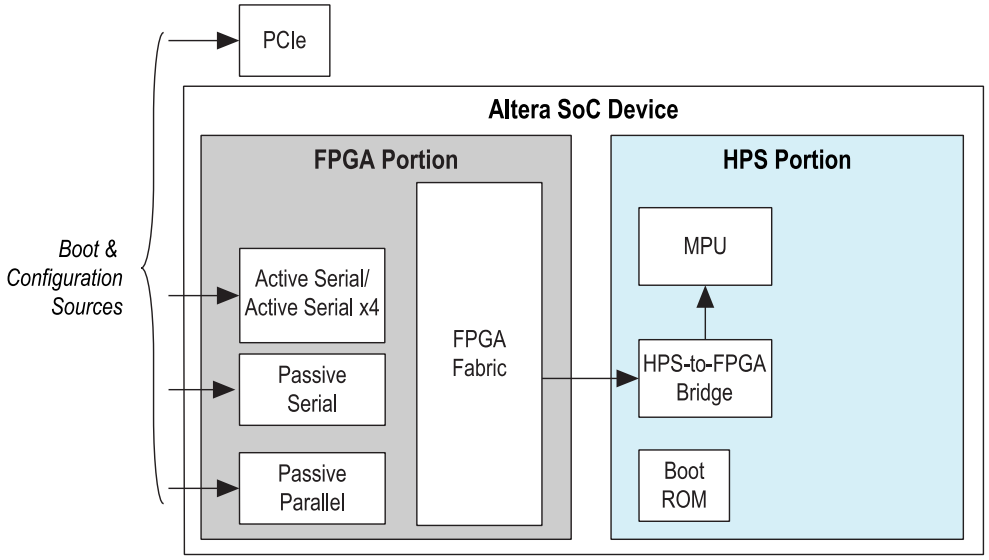


Figure 7-6. FPGA Configuration before HPS Booting (HPS boots from FPGA) [2, pp. A-2]

Figure 7-6 shows the scheme where the FPGA is first configured through the *Quartus Prime Programmer*, then the HPS boots from the FPGA fabric. The HPS boot waits for the FPGA fabric to be powered on and in user mode before executing. The HPS boot ROM code executes the preloader from the FPGA fabric over the HPS-to-FPGA bridge. The preloader can be obtained from the FPGA on-chip memory, or by accessing an external interface (such as a larger external SDRAM).

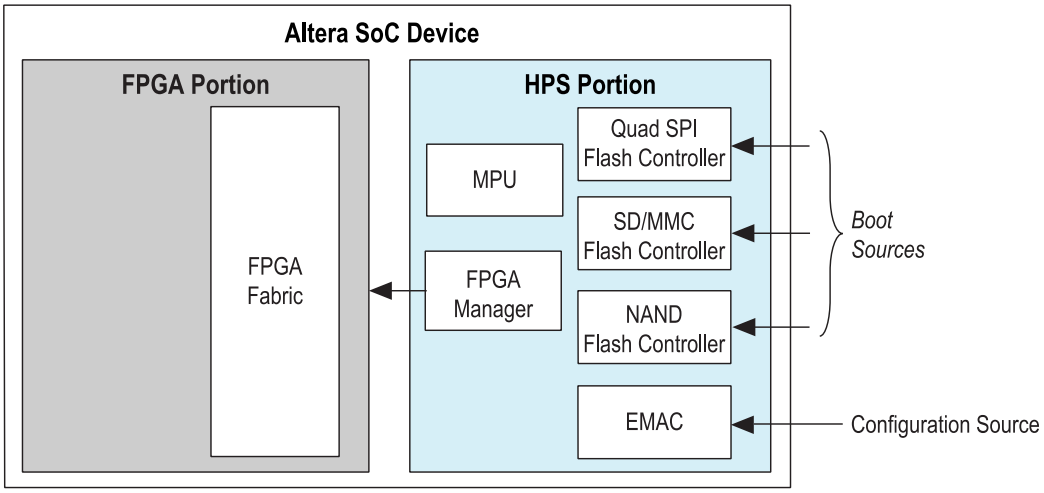


Figure 7-7. HPS Boots and Performs FPGA Configuration [2, pp. A-3]

Figure 7-7 shows the scheme under which the HPS first boots from one of its non-FPGA fabric boot sources, then software running on the HPS configures the FPGA fabric through the FPGA manager. The software on the HPS obtains the FPGA configuration image from any of its flash memory devices or communication interfaces, such as the SD/MMC memory, or the Ethernet port. The software is provided by users and the boot ROM is not involved in configuring the FPGA fabric.

7.6.2 Zooming In On the HPS Boot Process

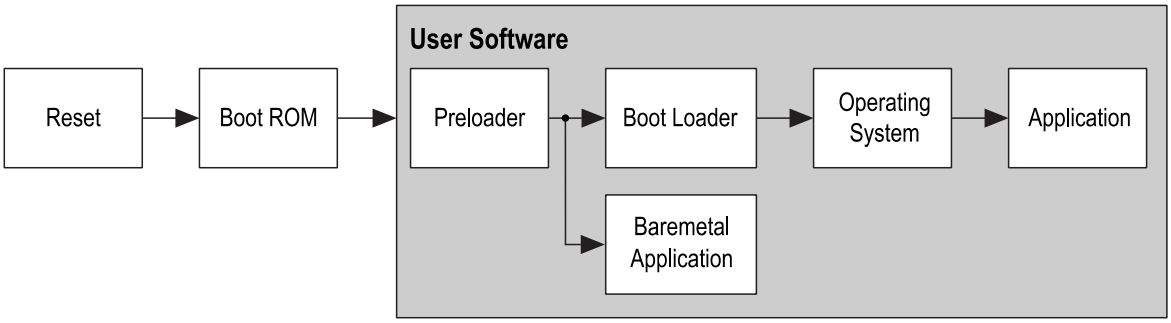


Figure 7-8. HPS Boot Flows [2, pp. A-3]

Booting software on the HPS is a multi-stage process. Each stage is responsible for loading the next stage. The first software stage is the *boot ROM*. The boot ROM code locates and executes the second software stage, called the *preloader*. The preloader locates, and ***IF PRESENT***, executes the next software stage. The preloader and subsequent software stages are collectively referred to as *user software*.

The *reset*, *boot ROM*, and *preloader* stages are always present in the HPS boot flow. What comes after the preloader then depends on the type of application you want to run. The HPS can execute 2 types of applications:

- Bare-metal applications (no operating system)
- Applications on top of an operating system (Linux)

Figure 7-8 shows the HPS' available boot flows. The *Reset* and *Boot ROM* stages are the only *fixed* parts of the boot process. Everything in the *user software* stages can be *customized*.

*Although the DE1-SoC has a **DUAL**-processor HPS (CPU0 and CPU1), the boot flow only executes on CPU0 and CPU1 is under reset. If you want to use both processors of the DE1-SoC, then **USER SOFTWARE** executing on CPU0 is responsible for releasing CPU1 from reset.*

7.6.2.1 Preloader

The preloader is one of the most important boot stages. It is actually what one would call the boot "*source*", as all stages before it are unmodifiable. The preloader can be stored on external flash-based memory, or in the FPGA fabric.

The preloader typically performs the following actions:

- Initialize the SDRAM interface
- Configure the HPS I/O through the scan manager
- Configure pin multiplexing through the system manager
- Configure HPS clocks through the clock manager
- Initialize the flash controller (NAND, SD/MMC, QSPI) that contains the next stage boot software
- Load the next boot software into the SDRAM and pass control to it

The preloader does ***NOT*** release CPU1 from reset. The subsequent stages of the boot process are responsible for it if they want to use the extra processor.

8 USING THE CYCLONE V – GENERAL INFORMATION

8.1 INTRODUCTION

The HPS component is a ***SOFT*** component, but it does ***NOT*** mean that the HPS is a softcore processor. In fact, the HPS exclusively contains ***HARD LOGIC***. The reason it is considered a softcore component originates from the fact that it enables other soft components to interface with the HPS hard logic. As such, the HPS component has a *small footprint* in the FPGA fabric, as its only purpose is to connect the soft and hard logic together.

Therefore, it is possible to use the Cyclone V SoC in 3 different configurations:

- FPGA-only
- HPS-only
- HPS & FPGA

We will look at the *FPGA-only* and *HPS & FPGA* configurations below. We will not cover the *HPS-only* configuration as it is identical to the *HPS & FPGA* one where you simply don't load any design on the FPGA fabric. The configurations using the HPS are more difficult to set up than the *FPGA-only* one.

8.2 FPGA-ONLY

Exclusively using the FPGA part of the Cyclone V is easy, as the design process is identical to any other Altera FPGA. You can build a complete design in *Quartus Prime* & *Qsys*, simulate it in *ModelSim-Altera*, then program the FPGA through the *Quartus Prime Programmer*. If you instantiated a Nios II processor in *Qsys,* you can use the *Nios II SBT* IDE to develop software for the processor.

The DE1-SoC has a lot of pins, which makes it tedious to start an FPGA design. It is recommended to use the ***ENTITY*** in Figure 15-1 for your ***TOP-LEVEL VHDL FILE***, as it contains all the board's FPGA and HPS pins.

After having defined a top-level module, it is necessary to map your design's pins to the ones available on the DE1-SoC. The ***TCL SCRIPT*** in Figure 15-2 can be executed in *Quartus Prime* to specify the board's device ID and all its ***PIN ASSIGNMENTS.*** In order to execute the TCL script, place it in your quartus working directory, then run it through the "`Tools > Tcl Scripts…`" menu item in *Quartus Prime*.

8.3 HPS & FPGA

8.3.1 Bare-metal Application

On one hand, bare-metal software enjoys the advantage of having no OS overhead. This has many consequences, the most visible of which are that code executes at native speed as no context switching is ever performed, and additionally, that code can directly address the HPS peripherals using their ***PHYSICAL*** memory-mapped addresses, as no virtual memory system is being used. This is very useful when trying to use the HPS as a high-speed microcontroller. Such a programming environment is very similar to the one used by other microcontrollers, like the TI MSP430.

On the other hand, bare-metal code has one great disadvantage, as the programmer must continue to configure the Cyclone V to use all its resources. For example, we saw in 7.6.2.1 that the preloader does not release CPU1 from reset, and that it is up to the *user software* to perform this, which is the bare-metal application itself in this case. Furthermore, supposing CPU1 is available for use, it is still difficult to run multi-threaded code, as an OS generally handles program scheduling and CPU affinity for the programmer. The programmer must now manually assign code fragments to each CPU.

8.3.2 Application Over an Operating System (Linux)

Running code over a Linux operating system has several advantages. First of all, the kernel releases CPU1 from reset upon boot, so all processors are available. Furthermore, the kernel initializes and makes most, if not all HPS peripherals available for use by the programmer. This is possible since the Linux kernel has access to a huge amount of device drivers. Multi-threaded code is also much easier to write, as the programmer has access to the familiar **Pthreads system calls**. Finally, the Linux kernel is not restricted to running compiled C programs. Indeed, you can always run code written in another programming language providing you first install the runtime environment required (that must be available for ARM processors).

However, running an ***"EMBEDDED"*** application on top of an operating system also has disadvantages. Due to the virtual memory system put in place by the OS, a program cannot directly access the HPS peripherals through their physical memory-mapped addresses. Instead, one first needs to map the physical addresses of interest into the running program's virtual address space. Only then will it be possible to access a peripheral's registers. Ideally, the programmer should write a device driver for each specific component that is designed to have a clean interface between user code, and device accesses.

At the end of the day, bare-metal applications and applications running code on top of Linux can do the same things. Generally speaking, programming on top of Linux is superior and much easier compared to bare-metal code, as its advantages greatly outweigh its drawbacks.

8.4 GOALS

Let's start by defining what we want to achieve in this tutorial. We want to create a system in which both the HPS and FPGA can do some computation simultaneously. More specifically, we want the following capabilities:

1. A Nios II processor on the ***FPGA*** must be able to use the 10 LEDs and 10 switches connected to the ***FPGA PORTION*** of the device. The Nios II processor will create a strobing light effect on the 10 LEDs, with the 10 switches acting as enable signals for the corresponding LEDs.
2. The Nios II processor will use its SDRAM instead of any form of on-chip memory.
3. The ***HPS*** must be able to use the LED and button that are directly connected to the ***HPS PORTION*** of the device. Pressing the button should toggle the LED.
4. The ***HPS*** must be able to use 2 buttons and the six 7-segment displays connected to the ***FPGA PORTION*** of the device. The HPS will increment and decrement a counter that will be shown on the 7-segment displays. Pressing the first button should invert the counting direction, and pushing the second button should reset the counter to 0.
5. The ***HPS*** must be able to use the ethernet port on the board.
6. The ***HPS*** must be able to use the microSD card port on the board to which we will write anything we want.

8.5 PROJECT STRUCTURE

The development process creates a lot more files compared to an FPGA-only design. We will use the folder structure shown in Figure 8-1 to organize our project. In this demo, we will use "`DE1_SoC_demo`" as the project name.

- The "`hw`" directory contains all hardware-related files.
- The "`sw`" directory contains all software-related files.
- The "`sdcard`" directory contains all final targets needed to create a valid sdcard from which the DE1-SoC can boot.

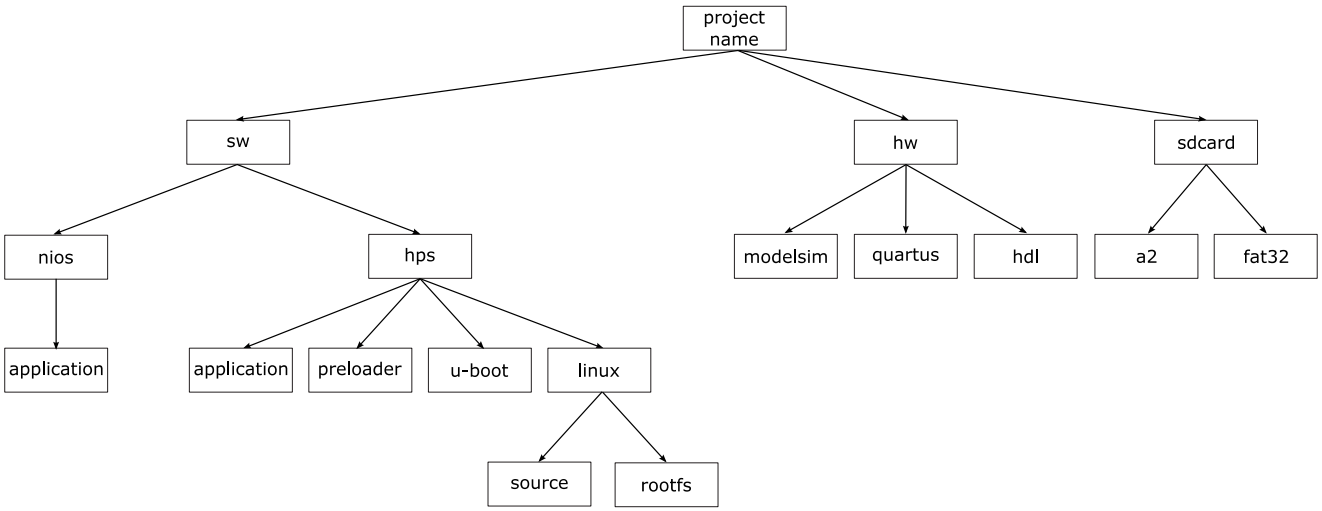


Figure 8-1. Project Folder Structure

Many steps have to be performed in order to configure the Cyclone V before you can use the HPS.

- The ***HARDWARE*** design is ***IDENTICAL*** whether you want to write bare-metal applications, or Linux HPS applications.
- The ***SOFTWARE*** design is ***DIFFERENT*** for bare-metal and Linux HPS applications.

The complete design for this tutorial can be found in `DE1_SoC_demo.zip` [3].

9 USING THE CYCLONE V – HARDWARE

The details below give step-by-step instructions to create a full system from scratch.

9.1 GENERAL QUARTUS PRIME SETUP

1. Create a new *Quartus Prime* project. You only need to specify the project name and destination, as all other settings will be set at a later stage by a TCL script. For this demo, we will call our project "DE1_SoC_demo" and will store it in "DE1_SoC_demo/hw/quartus".
2. Download DE1_SoC_top_level.vhd [4] and save it in "DE1_SoC_demo/hw/hdl". We will use this file as the project's top-level VHDL file, as it contains a complete list of pin names available on the DE1-SoC for use in your designs. Add the file to the *Quartus Prime* project by using "Project > Add/Remove Files in Project…" and set it as your design's top-level entity.
3. Download pin_assignment_DE1_SoC.tcl [5] and save it in "DE1_SoC_demo/hw/quartus". This script assigns pin locations and I/O standards to all pins names in "DE1_SoC_top_level.vhd". Execute the TCL script by using "Tools > Tcl Scripts…" in *Quartus Prime*.

At this stage, all general *Quartus Prime* settings have sbeen performed, and we can start creating our design. We want to use the HPS, as well as a Nios II processor in our design, so we will use the *Qsys* tool to create the system.

4. Launch the *Qsys* tool and create a new system. Save it under the name "soc_system.qsys".

9.2 SYSTEM DESIGN WITH QSYS – NIOS II

In this section, we assemble all system components needed to allow the Nios II processor to create a strobing light effect on the 10 LEDs with the 10 switches acting as enable signals for the corresponding LEDs.

We want to use a Nios II processor with an SDRAM. To use an SDRAM, we need 2 things:

- An SDRAM controller.
- A PLL to generate a clock for the softcore SDRAM controller and a phase-shifted clock for the off-chip SDRAM component. The reference clocks and timings needed for the SDRAM can be found on its datasheet: IS42R16320D.pdf [6].

5. Add an "Altera PLL" to the system.
    - Reference Clock Frequency: 50 MHz
    - Operation Mode: normal
    - Uncheck "Enable locked output port"

    We need to generate 3 clocks:
    a. 50 MHz clock for the Nios II processor and all its peripherals.
    b. 100 MHz clock for the SDRAM controller.
    c. 100 MHz, -3758 ps phase-shifted clock for the off-chip SDRAM component.

    In *Qsys*' "System Contents" tab:

    - Export "pll_0.outclk2" under the name "pll_0_sdram", as shown in Figure 9-1. Exporting the pll_0.outclk2 Signal. This clock will be used for the off-chip SDRAM component.

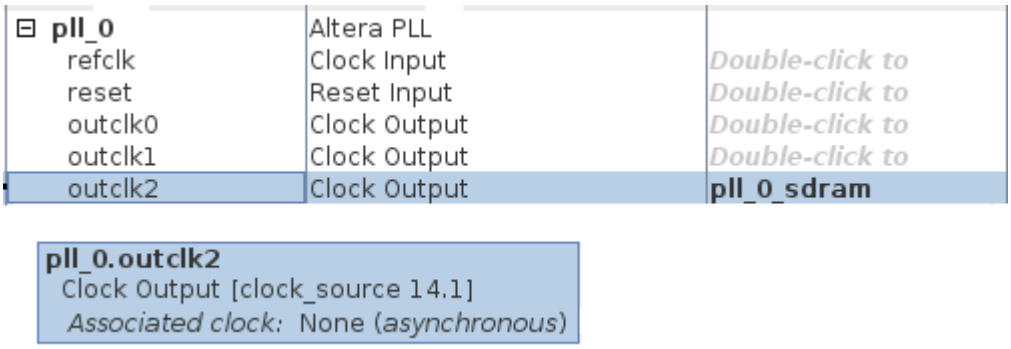


Figure 9-1. Exporting the pll_0.outclk2 Signal

6. Add an softcore SDRAM controller to the system. Use the following settings (taken from the SDRAM's datasheet):
    a. Memory Profile
        - Data Width
            - Bits: 16
        - Architecture
            - Chip select: 1
            - Banks: 4
        - Address Width
            - Row: 13
            - Column: 10
    b. Timing
        - CAS latency cycles: 3
        - Initialization refresh cycles: 2
        - Issue one refresh command every: 7.8125 us
        - Delay after powerup, before initialization: 100.0 us
        - Duration of refresh command (t_rfc): 70.0 ns
        - Duration of precharge command (t_rp): 15.0 ns
        - ACTIVE to READ or WRITE delay (t_rcd): 15.0 ns
        - Access time (t_ac): 5.4 ns
        - Write recovery time (t_wr, no auto precharge): 14.0 ns

    In *Qsys*' "`System Contents`" tab:
    - Rename "`new_sdram_controller_0`" to "`sdram_controller_0`".
    - Export "`sdram_controller_0.wire`" under the name "`sdram_controller_0_wire`".
7. Add a Nios II processor to the system. You can choose any variant. In this demo, we use the "`Nios II (classic)`" processor, with configuration "`Nios II/f`".
8. Add a System ID Peripheral to the system. In *Qsys*' "`System Contents`" tab:
    - Rename the component to "`sysid`"
9. Add a JTAG UART to the system. This serial console will be used to be able to see the output generated by the `printf()` function when programming the Nios II processor.
10. Connect the system as shown in Figure 9-2 below:

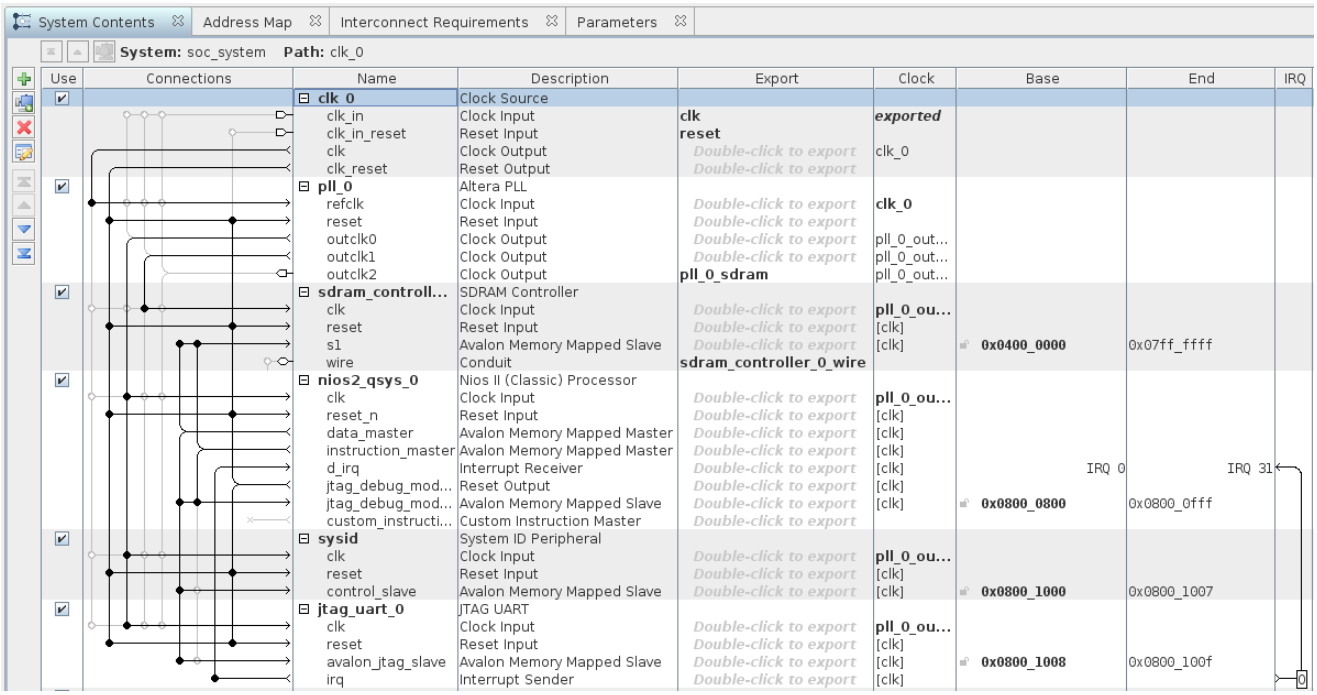


Figure 9-2. Basic Nios II System with SDRAM and JTAG UART

11. Edit the Nios II processor and set "`sdram_controller_0.s1`" as its Reset and Exception vectors.
12. Add a PIO component to the system for the LEDs. The DE1-SoC has 10 LEDs, so we will use a 10-bit PIO component.
    a. Width: 10 bits
    b. Direction: Output
    c. Output Port Reset Value: 0x00

    In *Qsys*' "`System Contents`" tab:
    - Rename the component to "`leds_0`"
    - Export "`leds_0.external_connection`"
13. Add a PIO component to the system for the switches. The DE1-SoC has 10 Switches, so we will again use a 10-bit PIO component.
    a. Width: 10 bits
    b. Direction: Input

    In *Qsys*' "`System Contents`" tab:
    c. Rename the component to "`switches_0`"
    d. Export "`switches_0.external_connection`"
14. Connect the system as shown in Figure 9-3 below (we don't show the full system to make figures hold on one page):

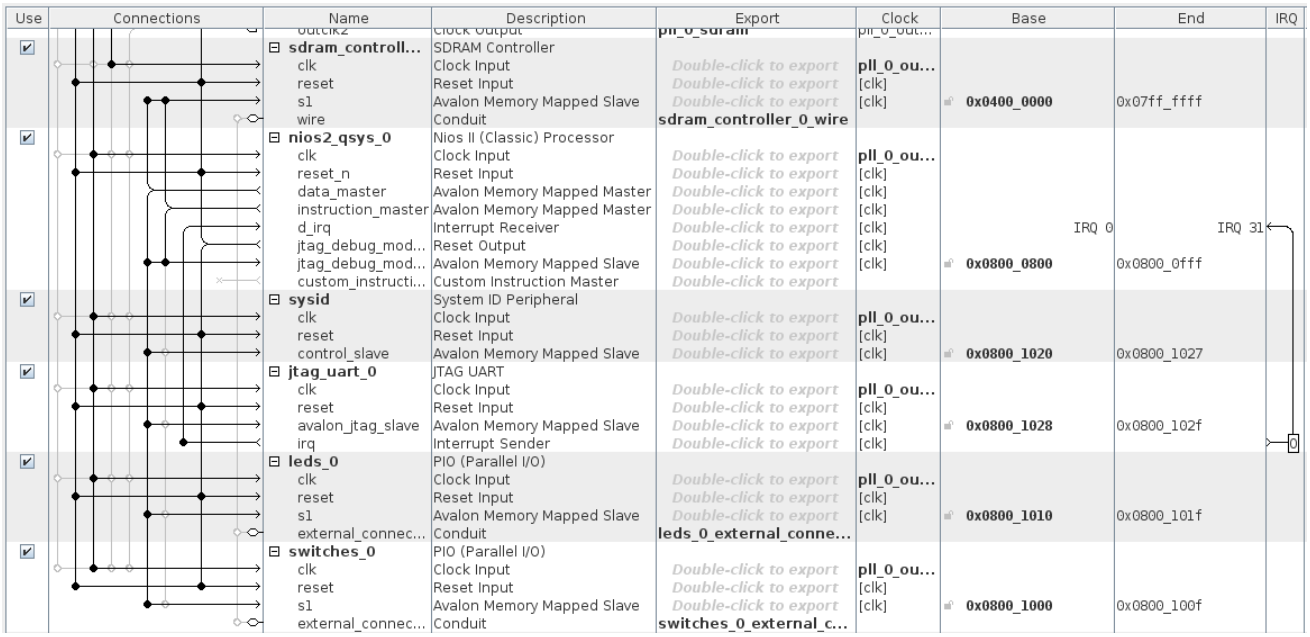


Figure 9-3. Adding LEDs and Switches to the System

At this stage, we have created a system that satisfies goals 1 and 2 defined in 8.4.

9.3 System Design with Qsys – HPS

In this section, we assemble all system components needed to allow the HPS to access a button and LED connected directly to itself, as well as a button and the 7-segment displays connected to the FPGA portion of the device.

Note: When using *Qsys* to manipulate any signal or menu item related to the HPS, the GUI will seem as though it is not responding, but this is not the case. The GUI is just checking all parameters in the background, which makes the interface hang momentarily. It is working correctly behind the scenes.

9.3.1 Instantiating the HPS Component

15. To use the HPS, add an "`Arria V/Cyclone V Hard Processor System`" to the system.
16. Open the HPS' parameters and have a look around. There are 4 tabs that control various aspects of the HPS' behaviour, as shown on Figure 9-4.

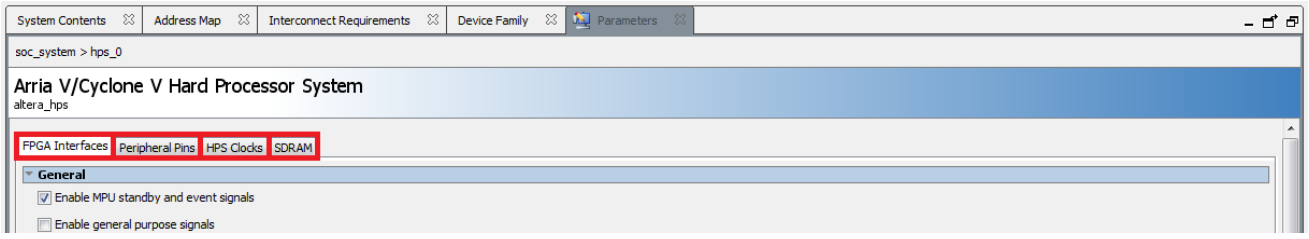


Figure 9-4. HPS Component Parameters

9.3.1.1 FPGA Interfaces Tab

This tab configures everything related to the interfaces between the HPS and the FPGA. You can configure which bridges to use, interrupts, …

17. We want to use the HPS to access FPGA peripherals, so we need to enable one of the following buses:
    a. HPS-to-FPGA AXI bridge
    b. Lightweight HPS-to-FPGA AXI bridge

    Since we are not going to be using any high performance FPGA peripherals in this demo, we'll choose to enable the Lightweight HPS-to-FPGA AXI bridge.
    - Set the FPGA-to-HPS interface width to "`Unused`".

- Set the HPS-to-FPGA interface width to "Unused".

By default, *Qsys* checks "Enable MPU standby and event signals", but we are not going to use this feature, so

- Uncheck "Enable MPU standby and event signals".

Qsys also adds an FPGA-to-HPS SDRAM port by default, which we are not going to use either, so

- Remove the port listed under "FPGA-to-HPS SDRAM Interface".

9.3.1.2 Peripheral Pins Tab

This tab configures the physical pins that are available on the device. Most device pins have various sources, and are *multiplexed*. The pins can be configured to be sourced by the FPGA, or by various HPS peripherals.

9.3.1.2.1 Theory

We want to use the HPS to access the button and LED that are directly connected to it. These HPS peripherals correspond to pins "HPS_KEY_N" and "HPS_LED" on the device's top-level entity. We need to know how these 2 pins are connected to the HPS to access them. To find out this information, we have to look at the board's schematics. You can find the schematics in DE1-SoC.pdf [7].

The right side of Figure 9-5 shows the area of interest on the DE1-SoC's schematics. We see that "HPS_KEY_N" and "HPS_LED" are respectively connected to pins G21 and A24.

Figure 9-5. HPS_KEY & HPS_LED on DE1-SoC Schematic

Figure 9-5 allows us to explain what *Qsys*' *Peripheral Pins* tab does. The *Qsys* GUI doesn't make any reference to pins G21 and A24, as they depend on the device being used, and cannot be generalized to other Cyclone V devices. However, the GUI does have references to what is displayed on the left side of Figure 9-5. We will examine the details of pin G21, to which "HPS_KEY_N" is connected. The schematic shows that pin G21 is connected to 4 sources:

c. TRACE_D5
d. SPIS1_MOSI
e. CAN1_TX
f. HPS_GPIO54

This can be seen in *Qsys*, as shown in Figure 9-6.

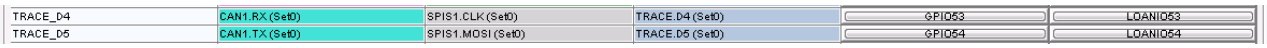


Figure 9-6. HPS_KEY & HPS_LED on Qsys Peripheral Pins Tab

Depending on how you configure the Peripheral Pins tab, you can configure pin G21 to use any of the sources above. For example, if you want to use this pin as an SPI slave control signal, you would use the configuration shown in Figure 9-7.

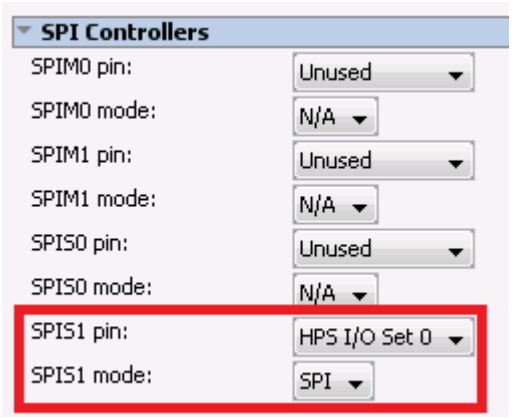


Figure 9-7. Using Pin G21 for SPI

However, if you don't want to use any of the peripherals available at the top of the *Peripheral Pins* tab, then you can always use one of the 2 buttons on the right side of Figure 9-6:

- ***GPIO****XY*: Configures the pin to be connected to the ***HPS' GPIO*** peripheral.
- ***LOANIO****XY*: Configures the pin to be connected to the ***FPGA*** fabric. This pin can be exported from *Qsys* to be used by the FPGA.

9.3.1.2.2 Configuration

18. We want the HPS to directly control the "HPS_KEY_N" and "HPS_LED" pins. To do this, we will connect pins G21 and A24 to the HPS' GPIO peripheral.
    a. Click on the "GPIO53" button. This corresponds to pin A24, which is connected to "HPS_LED".
    b. Click on the "GPIO54" button. This corresponds to pin G21, which is connected to "HPS_KEY_N".
19. We want to connect to our DE1-SoC with an SSH connection later in the tutorial, so we need to enable the Ethernet MAC interface.
    a. Configure "EMAC1 pin" to "HPS I/O Set 0" and the "EMAC 1 mode" to "RGMII", as shown in Figure 9-8.
    b. Click on the "GPIO35" button. This corresponds to pin C19, which is connected to "HPS_ENET_INT_N".

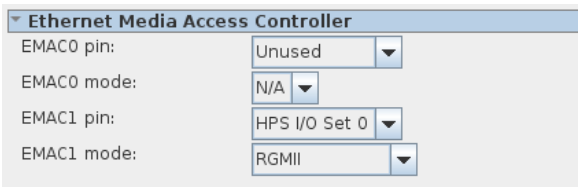


Figure 9-8. Ethernet MAC configuration

20. Our system will boot from the microSD card slot, so we need to enable the SD/MMC controller.
    a. Configure "SDIO pin" to "HPS I/O Set 0" and "SDIO mode" to "4-bit Data", as shown in Figure 9-9.

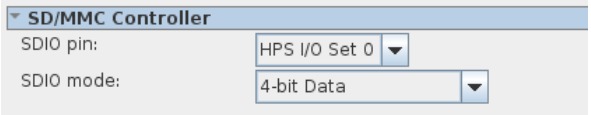


Figure 9-9. SD/MMC configuration

21. When initially configuring our system, we will need to connect a keyboard to our system. We will do this through a serial UART connection, so we need to enable the UART controller.
    a. Configure "`UART0 pin`" to "`HPS I/O Set 0`" and "`UART0 mode`" to "`No Flow Control`", as shown in Figure 9-10.

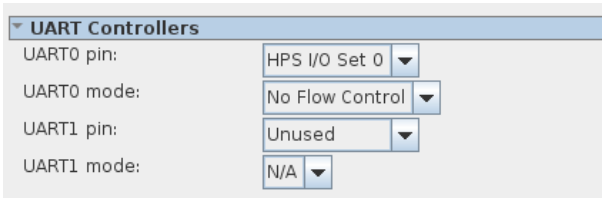


Figure 9-10. UART configuration

At this stage, you should have the same configuration shown in Figure 9-11.

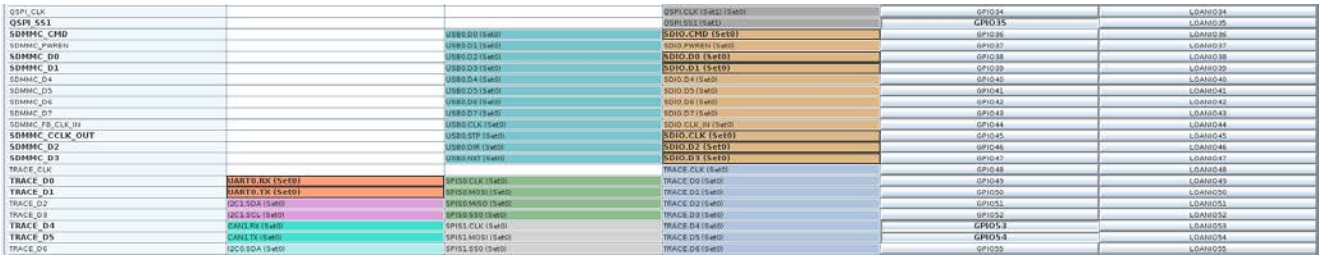


Figure 9-11. Exported peripheral pins

22. In *Qsys*' "`System Contents`" tab:
    - Export "`hps_0.hps_io`" under the name "`hps_0_io`". This is a conduit that contains all the pins configured in the *Peripheral Pins* tab. We will connect these to our top-level entity later.

9.3.1.3 HPS Clocks Tab

This tab configures the clocking system of the HPS. We will generally use the default settings here, so no need to change anything.

9.3.1.4 SDRAM Tab

This tab configures the memory subsystem of the HPS.

23. We need to configure all clocks and timings related to the memory used on our system. The DE1-SoC uses DDR3 memory, so we need to consult its datasheet to find all the settings. The datasheet is available at `43TR16256A-85120AL(ISSI).pdf` [8] . Based on the memory's datasheet, we can fill in the following memory settings (you will soon see that it is quite tedious to enter these values):
    - SDRAM Protocol: DDR3
    - PHY Settings:
        - Clocks:
            - Memory clock frequency: 400.0 MHz
            - PLL reference clock frequency: 25.0 MHz
        - Advanced PHY Settings:
            - Supply Voltage: 1.5V DDR3
    - Memory Parameters:
        - Memory vendor: Other
        - Memory device speed grade: 800.0 MHz
        - Total interface width: 32
        - Number of chip select/depth expansion: 1
        - Number of clocks: 1
        - Row address width: 15
        - Column address width: 10

        - Bank-address width: 3
        - Enable DM pins
        - DQS# Enable
        - Memory Initialization Options:
            - Mirror Addressing: 1 per chip select: 0
            - Mode Register 0:
                - Burst Length: Burst chop 4 or 8 (on the fly)
                - Read Burst Type: Sequential
                - DLL precharge power down: DLL off
                - Memory CAS latency setting: 11
            - Mode Register 1:
                - Output drive strength setting: RZQ/7
                - ODT Rtt nominal value: RZQ/4
            - Mode Register 2:
                - Auto selfrefresh method: Manual
                - Selfrefresh temperature: Normal
                - Memory write CAS latency setting: 8
                - Dynamic ODT (Rtt_WR) value: RZQ/4
- Memory Timing:
    - tIS (base): 180 ps
    - tIH (base): 140 ps
    - tDS (base): 30 ps
    - tDH (base): 65 ps
    - tDQSQ: 125 ps
    - tQH: 0.38 cycles
    - tDQSCK: 255 ps
    - tDQSS: 0.25 cycles
    - tQSH: 0.4 cycles
    - tDSH: 0.2 cycles
    - tDSS: 0.2 cycles
    - tINIT: 500 us
    - tMRD: 4 cycles
    - tRAS: 35.0 ns
    - tRCD: 13.75 ns
    - tRP: 13.75 ns
    - tREFI: 7.8 us
    - tRFC: 260.0 ns
    - tWR: 15.0 ns
    - tWTR: 4 cycles
    - tFAW: 30.0 ns
    - tRRD: 7.5 ns
    - tRTP: 7.5 ns
- Board Settings:
    - Setup and Hold Derating:
        - Use Altera's default settings
    - Channel Signal Integrity:
        - Use Altera's default settings
    - Board Skews:
        - Maximum CK delay to DIMM/device: 0.03 ns
        - Maximum DQS delay to DIMM/device: 0.02 ns

- Minimum delay difference between CK and DQS: 0.06 ns
- Maximum delay difference between CK and DQS: 0.12 ns
- Maximum skew within DQS group: 0.01 ns
- Maximum skew between DQS groups: 0.06 ns
- Average delay difference between DQ and DQS: 0.05 ns
- Maximum skew within address and command bus: 0.02 ns
- Average delay difference between address and command and CK: 0.01 ns

24. In *Qsys*' "`System Contents`" tab:
    - Export "`hps_0.memory`" under the name "`hps_0_ddr`".
25. Connect the system as shown in Figure 9-12 below:

Figure 9-12. Adding the "Standalone" HPS to the System

At this stage, we have a functional HPS unit that can be programmed and that satisfies goals 1, 2, 3, 5, and 6 defined in 8.4. In our current system however, the HPS can only be used "standalone" and cannot access any FPGA peripherals.

9.3.2 Interfacing with FPGA Peripherals

The next step is to connect the HPS to FPGA peripherals through one of its interface bridges. The setup we have uses the Lightweight HPS-to-FPGA bridge to communicate with the FPGA.

26. Add a PIO component to the system for the buttons. The DE1-SoC has 4 buttons, so we will use a 4-bit PIO component.
    - Width: 4 bits
    - Direction: Input

    In *Qsys*' "`System Contents`" tab:
    - Rename the component to "`buttons_0`"
    - Export "`buttons_0.external_connection`"
27. Add a PIO component for one of the 7-segment displays. We will use a 7-bit PIO component.
    - Width: 7 bits
    - Direction: Output
    - Output Port Reset Value: 0x7f

    In *Qsys*' "`System Contents`" tab:
    - Rename the component to "`hex_0`"
    - Export "`hex_0.external_connection`"

28. Repeat step 27 five more times to obtain a total of six 7-segment displays "hex_0", "hex_1", "hex_2", "hex_3", "hex_4", and "hex_5".
29. Connect the system as shown in Figure 9-13 below. Notice that we use "hps_0.h2f_reset" as the reset signal for the components connected to the HPS. This is a design choice so we can separately reset FPGA-only peripherals, and FPGA peripherals connected to the HPS.

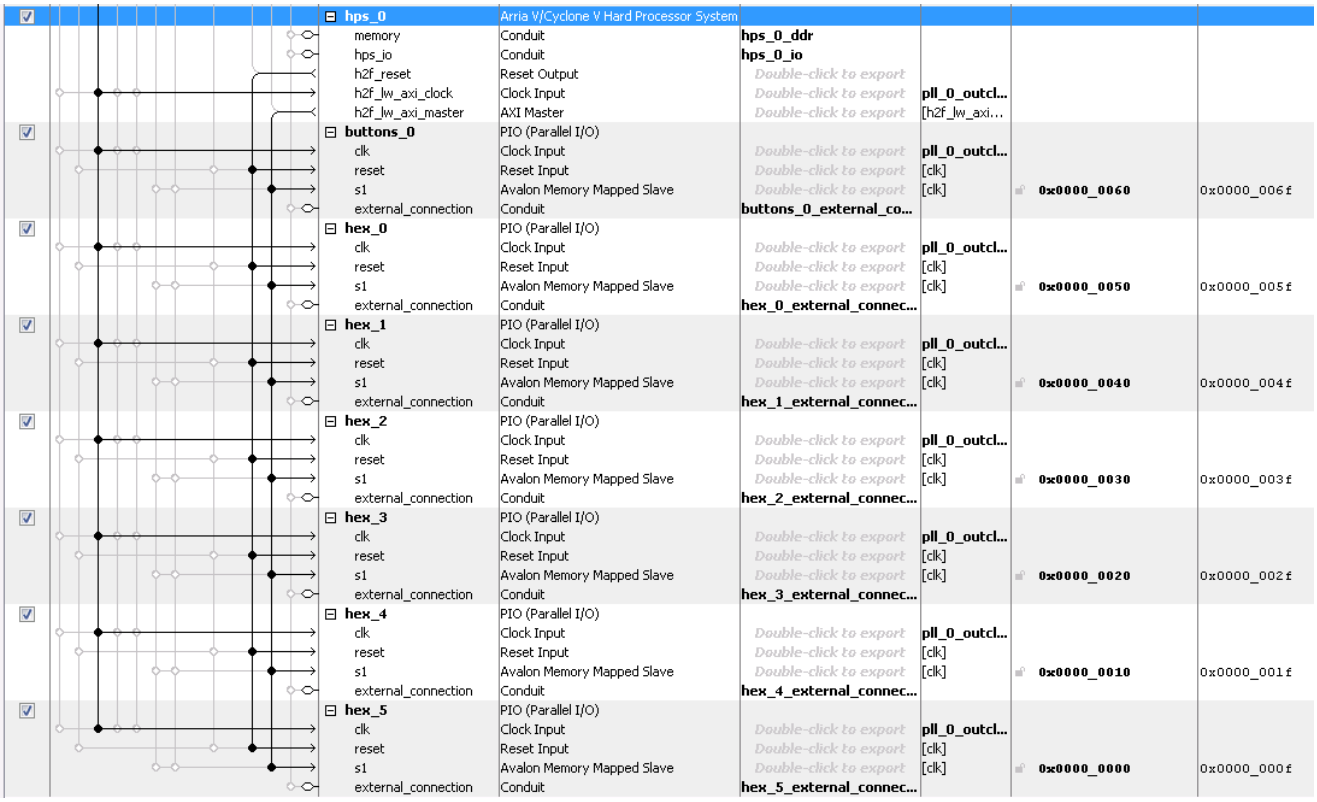


Figure 9-13. Adding Buttons and 7-segment Displays to the Lightweight HPS-to-FPGA Bridge

30. In the main *Qsys* window, select "System > Assign Base Addresses" to get rid of any error messages regarding memory space overlaps among the different components in the system.

At this stage, we finally have a system that satisfies all goals defined in 8.4. Our design work with *Qsys* is now done.

9.4 GENERATING THE QSYS SYSTEM

31. Click on the "Generate HDL" button.
32. Select "VHDL" for "Create HDL design files for synthesis".
33. Click on the "Generate" button to generate the system.
34. Save the design and exit *Qsys*. When asked if you want to generate the design, select "No", as we have already done it in the previous step.

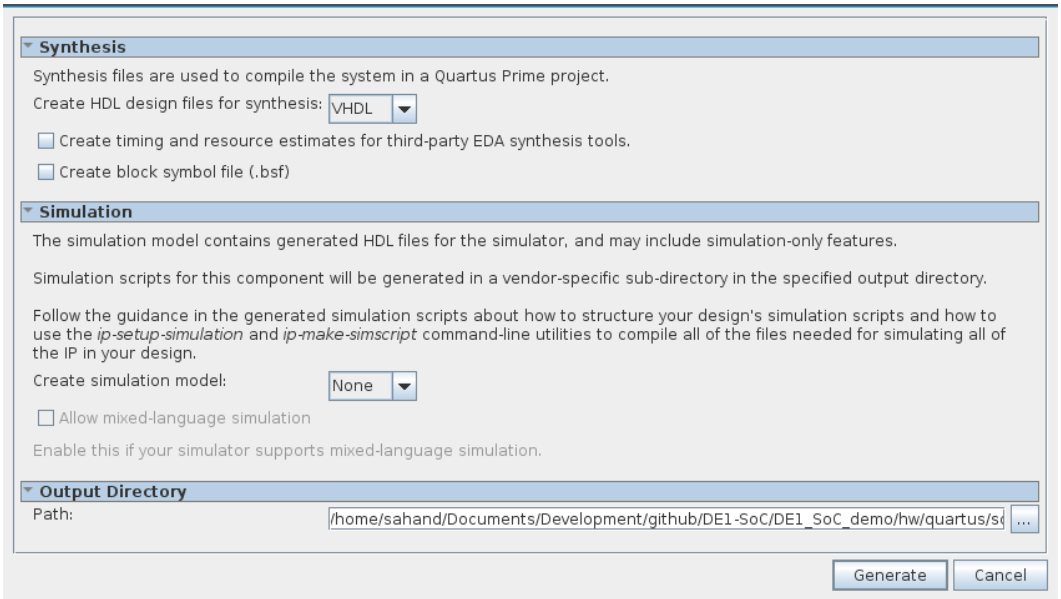


Figure 9-14. Generate Qsys System

9.5 INSTANTIATING THE QSYS SYSTEM

You now have a complete *Qsys* system. The system will be available as an instantiable component in your design files. However, in order for *Quartus Prime* to see the *Qsys* system, you will have to add the system's files to your *Quartus Prime* project.

35. Add "`DE1_SoC_demo/hw/quartus/soc_system/synthesis/soc_system.qip`" to the *Quartus Prime* project by using "`Project > Add/Remove Files in Project…`".
36. To use the *Qsys* system in your design, you have to declare its component, and then instantiate it. Qsys already provides you with a component declaration. You can find it among the numerous files that were generated. The one we are looking for is "`DE1_SoC_demo/hw/quartus/soc_system/soc_system.cmp`".
37. Copy the component declaration code in "`DE1_SoC_demo/hw/hdl/DE1_SoC_top_level.vhd`". Be sure to instantiate the component and assign all the correct pins of the DE1-SoC board. For our demo project, we would use the instantiation shown in Figure 9-15.

```
soc_system_inst : component soc_system
    port map(
        clk_clk                                => CLOCK_50,
        reset_reset_n                          => '1',
        buttons_0_external_connection_export   => KEY_N,
        hex_0_external_connection_export       => HEX0_N,
        hex_1_external_connection_export       => HEX1_N,
        hex_2_external_connection_export       => HEX2_N,
        hex_3_external_connection_export       => HEX3_N,
        hex_4_external_connection_export       => HEX4_N,
        hex_5_external_connection_export       => HEX5_N,
        leds_0_external_connection_export      => LEDR,
        switches_0_external_connection_export  => SW,
        pll_0_sdram_clk                        => DRAM_CLK,
        sdram_controller_0_wire_addr           => DRAM_ADDR,
        sdram_controller_0_wire_ba             => DRAM_BA,
        sdram_controller_0_wire_cas_n          => DRAM_CAS_N,
        sdram_controller_0_wire_cke            => DRAM_CKE,
        sdram_controller_0_wire_cs_n           => DRAM_CS_N,
        sdram_controller_0_wire_dq             => DRAM_DQ,
        sdram_controller_0_wire_dqm(1)         => DRAM_UDQM,
```

```
        sdram_controller_0_wire_dqm(0)        => DRAM_LDQM,
        sdram_controller_0_wire_ras_n         => DRAM_RAS_N,
        sdram_controller_0_wire_we_n          => DRAM_WE_N,
        hps_0_ddr_mem_a                       => HPS_DDR3_ADDR,
        hps_0_ddr_mem_ba                      => HPS_DDR3_BA,
        hps_0_ddr_mem_ck                      => HPS_DDR3_CK_P,
        hps_0_ddr_mem_ck_n                    => HPS_DDR3_CK_N,
        hps_0_ddr_mem_cke                     => HPS_DDR3_CKE,
        hps_0_ddr_mem_cs_n                    => HPS_DDR3_CS_N,
        hps_0_ddr_mem_ras_n                   => HPS_DDR3_RAS_N,
        hps_0_ddr_mem_cas_n                   => HPS_DDR3_CAS_N,
        hps_0_ddr_mem_we_n                    => HPS_DDR3_WE_N,
        hps_0_ddr_mem_reset_n                 => HPS_DDR3_RESET_N,
        hps_0_ddr_mem_dq                      => HPS_DDR3_DQ,
        hps_0_ddr_mem_dqs                     => HPS_DDR3_DQS_P,
        hps_0_ddr_mem_dqs_n                   => HPS_DDR3_DQS_N,
        hps_0_ddr_mem_odt                     => HPS_DDR3_ODT,
        hps_0_ddr_mem_dm                      => HPS_DDR3_DM,
        hps_0_ddr_oct_rzqin                   => HPS_DDR3_RZQ,
        hps_0_io_hps_io_emac1_inst_TX_CLK     => HPS_ENET_GTX_CLK,
        hps_0_io_hps_io_emac1_inst_TX_CTL     => HPS_ENET_TX_EN,
        hps_0_io_hps_io_emac1_inst_TXD0       => HPS_ENET_TX_DATA(0),
        hps_0_io_hps_io_emac1_inst_TXD1       => HPS_ENET_TX_DATA(1),
        hps_0_io_hps_io_emac1_inst_TXD2       => HPS_ENET_TX_DATA(2),
        hps_0_io_hps_io_emac1_inst_TXD3       => HPS_ENET_TX_DATA(3),
        hps_0_io_hps_io_emac1_inst_RX_CLK     => HPS_ENET_RX_CLK,
        hps_0_io_hps_io_emac1_inst_RX_CTL     => HPS_ENET_RX_DV,
        hps_0_io_hps_io_emac1_inst_RXD0       => HPS_ENET_RX_DATA(0),
        hps_0_io_hps_io_emac1_inst_RXD1       => HPS_ENET_RX_DATA(1),
        hps_0_io_hps_io_emac1_inst_RXD2       => HPS_ENET_RX_DATA(2),
        hps_0_io_hps_io_emac1_inst_RXD3       => HPS_ENET_RX_DATA(3),
        hps_0_io_hps_io_emac1_inst_MDIO       => HPS_ENET_MDIO,
        hps_0_io_hps_io_emac1_inst_MDC        => HPS_ENET_MDC,
        hps_0_io_hps_io_sdio_inst_CLK         => HPS_SD_CLK,
        hps_0_io_hps_io_sdio_inst_CMD         => HPS_SD_CMD,
        hps_0_io_hps_io_sdio_inst_D0          => HPS_SD_DATA(0),
        hps_0_io_hps_io_sdio_inst_D1          => HPS_SD_DATA(1),
        hps_0_io_hps_io_sdio_inst_D2          => HPS_SD_DATA(2),
        hps_0_io_hps_io_sdio_inst_D3          => HPS_SD_DATA(3),
        hps_0_io_hps_io_uart0_inst_RX         => HPS_UART_RX,
        hps_0_io_hps_io_uart0_inst_TX         => HPS_UART_TX,
        hps_0_io_hps_io_gpio_inst_GPIO35      => HPS_ENET_INT_N,
        hps_0_io_hps_io_gpio_inst_GPIO53      => HPS_LED,
        hps_0_io_hps_io_gpio_inst_GPIO54      => HPS_KEY_N
    );
```

Figure 9-15. Qsys Component Instantiation

38. After finishing the design, ***REMOVE*** all unused pins from the top-level VHDL file. Your top-level entity should look like the one shown in Figure 9-16.

```
entity DE1_SoC_top_level is
    port(
        -- CLOCK
        CLOCK_50         : in    std_logic;

        -- SDRAM
        DRAM_ADDR        : out   std_logic_vector(12 downto 0);
        DRAM_BA          : out   std_logic_vector(1 downto 0);
        DRAM_CAS_N       : out   std_logic;
```

```
        DRAM_CKE         : out   std_logic;
        DRAM_CLK         : out   std_logic;
        DRAM_CS_N        : out   std_logic;
        DRAM_DQ          : inout std_logic_vector(15 downto 0);
        DRAM_LDQM        : out   std_logic;
        DRAM_RAS_N       : out   std_logic;
        DRAM_UDQM        : out   std_logic;
        DRAM_WE_N        : out   std_logic;

        -- SEG7
        HEX0_N           : out   std_logic_vector(6 downto 0);
        HEX1_N           : out   std_logic_vector(6 downto 0);
        HEX2_N           : out   std_logic_vector(6 downto 0);
        HEX3_N           : out   std_logic_vector(6 downto 0);
        HEX4_N           : out   std_logic_vector(6 downto 0);
        HEX5_N           : out   std_logic_vector(6 downto 0);

        -- KEY_N
        KEY_N            : in    std_logic_vector(3 downto 0);

        -- LED
        LEDR             : out   std_logic_vector(9 downto 0);

        -- SW
        SW               : in    std_logic_vector(9 downto 0);

        -- HPS
        HPS_DDR3_ADDR    : out   std_logic_vector(14 downto 0);
        HPS_DDR3_BA      : out   std_logic_vector(2 downto 0);
        HPS_DDR3_CAS_N   : out   std_logic;
        HPS_DDR3_CK_N    : out   std_logic;
        HPS_DDR3_CK_P    : out   std_logic;
        HPS_DDR3_CKE     : out   std_logic;
        HPS_DDR3_CS_N    : out   std_logic;
        HPS_DDR3_DM      : out   std_logic_vector(3 downto 0);
        HPS_DDR3_DQ      : inout std_logic_vector(31 downto 0);
        HPS_DDR3_DQS_N   : inout std_logic_vector(3 downto 0);
        HPS_DDR3_DQS_P   : inout std_logic_vector(3 downto 0);
        HPS_DDR3_ODT     : out   std_logic;
        HPS_DDR3_RAS_N   : out   std_logic;
        HPS_DDR3_RESET_N : out   std_logic;
        HPS_DDR3_RZQ     : in    std_logic;
        HPS_DDR3_WE_N    : out   std_logic;
        HPS_ENET_GTX_CLK : out   std_logic;
        HPS_ENET_INT_N   : inout std_logic;
        HPS_ENET_MDC     : out   std_logic;
        HPS_ENET_MDIO    : inout std_logic;
        HPS_ENET_RX_CLK  : in    std_logic;
        HPS_ENET_RX_DATA : in    std_logic_vector(3 downto 0);
        HPS_ENET_RX_DV   : in    std_logic;
        HPS_ENET_TX_DATA : out   std_logic_vector(3 downto 0);
        HPS_ENET_TX_EN   : out   std_logic;
        HPS_KEY_N        : inout std_logic;
        HPS_LED          : inout std_logic;
        HPS_SD_CLK       : out   std_logic;
        HPS_SD_CMD       : inout std_logic;
        HPS_SD_DATA      : inout std_logic_vector(3 downto 0);
        HPS_UART_RX      : in    std_logic;
        HPS_UART_TX      : out   std_logic
```

```
    );
end entity DE1_SoC_top_level;
```

Figure 9-16. Final Top-level Entity

9.6 HPS DDR3 Pin Assignments

In a normal FPGA design flow, you would be able to compile your design at this stage. However, this isn't possible at the moment in our design. The reason is that the HPS' DDR3 pins assignments have not been performed yet.

How is this possible? We said earlier that our TCL script assigns pin locations and I/O standards to all pins names in "`DE1_SoC_top_level.vhd`". The truth is that it assigns values for all pin names, except those related to the HPS' DDR3 memory. The reason is that the DDR3 pin assignments depend on how you parameterize the HPS memory timings in *Qsys*. Our TCL script could not have known what timings you were going to use, so it doesn't set those pin locations and I/O standards.

However, *Qsys* knows what the parameters are (since you provided it with all the necessary information), and it has generated a custom TCL script for the HPS DDR3 pin assignments.

39. Start the "`Analysis and Synthesis`" flow to perform a preliminary analysis of the system.
40. Go to "`Tools > Tcl Scripts…`" in *Quartus Prime*.

IF AT THIS POINT YOU DO NOT SEE THE SAME THING AS ON Figure 9-17***, THEN CLOSE AND RELAUNCH QUARTUS PRIME AGAIN. SOME VERSIONS OF QUARTUS PRIME SUFFER FROM A BUG, WHERE THE PROGRAM DOESN'T CORRECTLY DETECT TCL FILES GENERATED BY QSYS. YOU SHOULD SEE THE SAME THING AS ON*** Figure 9-17***.***

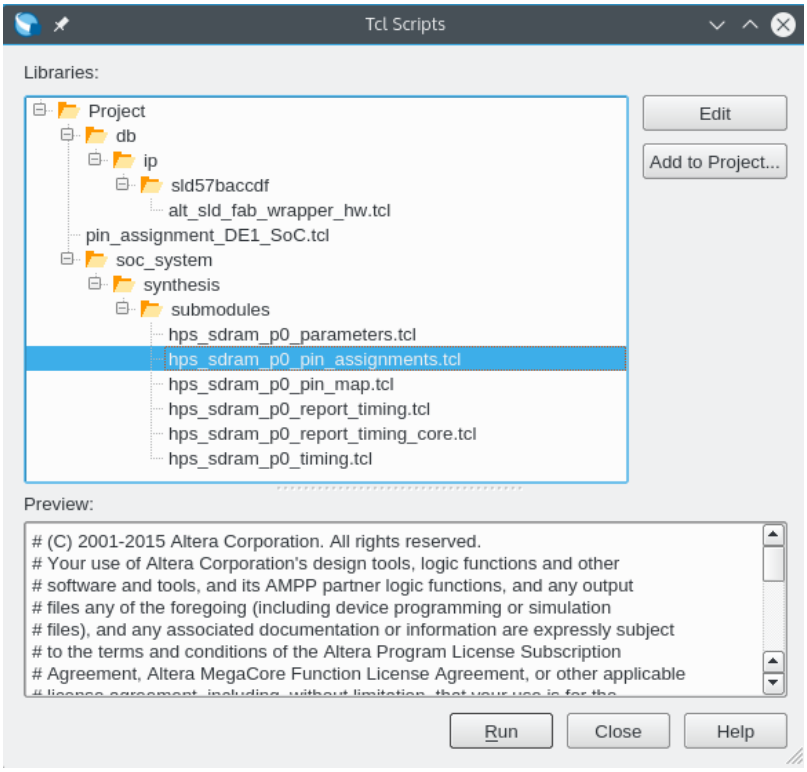


Figure 9-17. Correct HPS DDR3 Pin Assignment TCL Script Selection

41. Execute "`hps_sdram_p0_pin_assignments.tcl`".
42. You can now start the full compilation of your design with the "`Start Compilation`" flow.

At this point, we have finished the hardware design process and can proceed to programming the FPGA.

9.7 PROGRAMMING THE FPGA

43. Open the *Quartus Prime Programmer*.

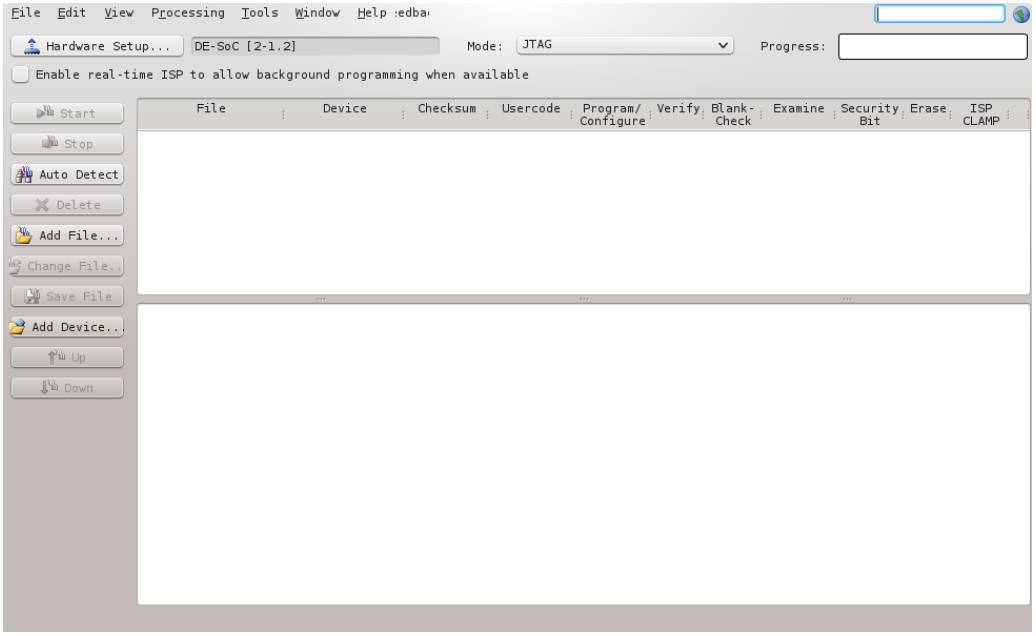


Figure 9-18. Quartus Prime Programmer

44. Choose the "`Auto Detect`" button on the left of Figure 9-18, then choose "`5CSEMA5`", as shown in Figure 9-19.

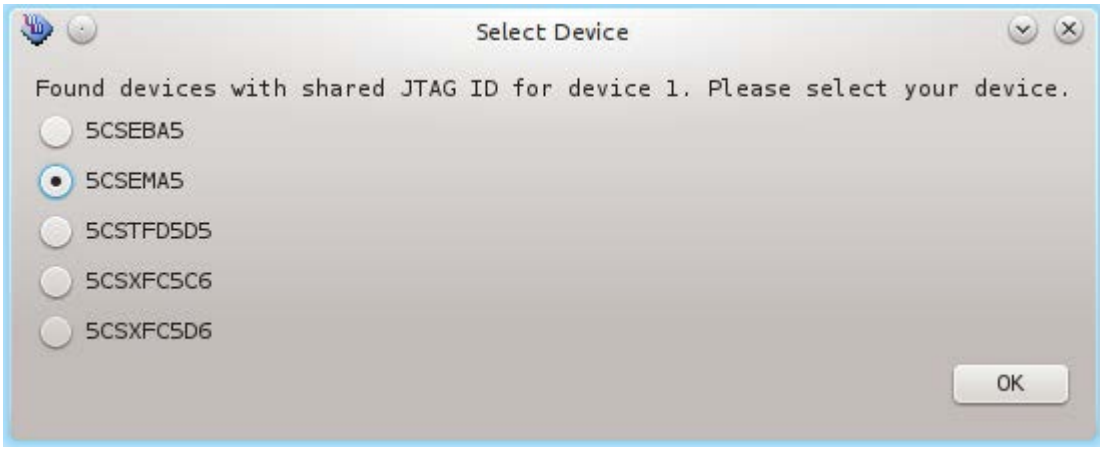


Figure 9-19. FPGA Selection

You should now see 2 devices on the JTAG scan chain, as shown in Figure 9-20.

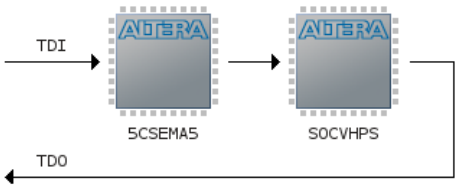


Figure 9-20. JTAG Scan Chain

45. Right-click on the "`5CSEMA5`" device shown in Figure 9-20 and choose "`Edit > Change File`". Then, select "`DE1_SoC_demo/hw/quartus/output_files/DE1_SoC_demo.sof`" through the file browser.
46. Enable the "`Program/Configure`" checkbox for device "`5CSEMA5F31`", then press the "`Start`" button, as shown in Figure 9-21.

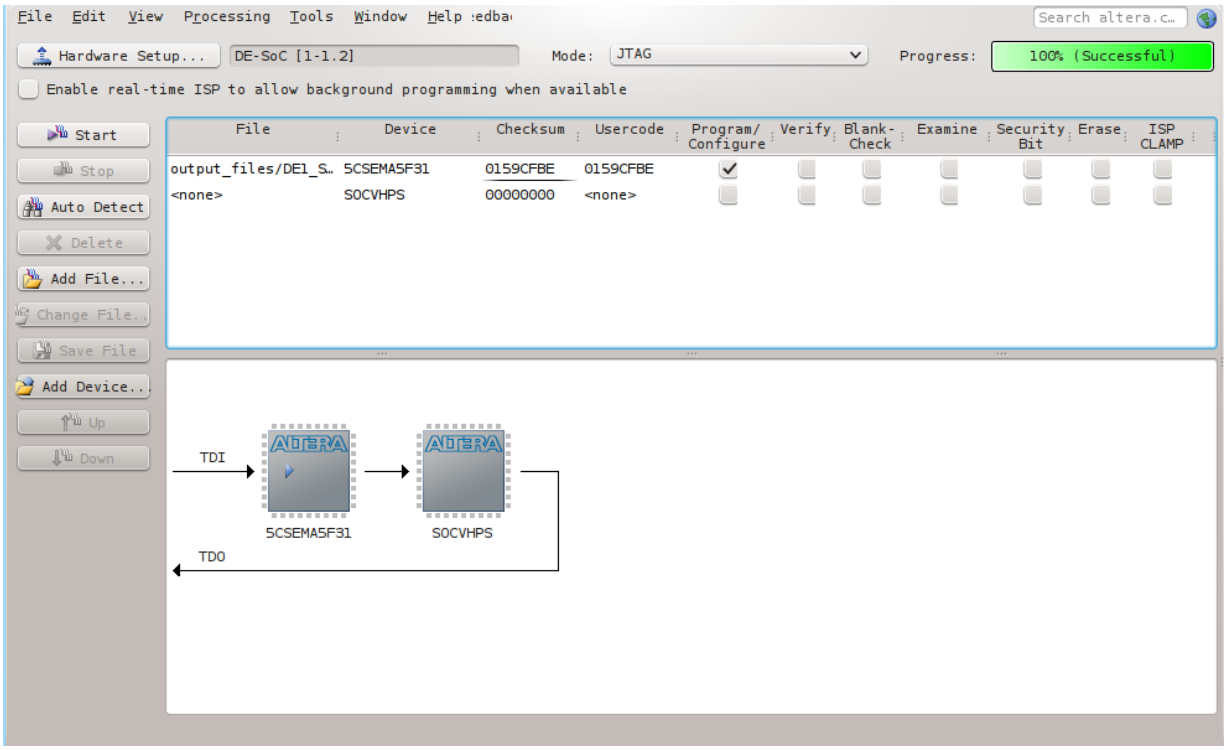


Figure 9-21. Programming the FPGA

We are now done with the *Quartus Prime* program, and will no longer need it for the rest of this tutorial.

9.8 Creating Target SDCard Artifacts

Later in this tutorial, we will sometimes want to avoid having to manually program the FPGA through the *Quartus Prime* programmer, and would instead like the HPS to take care of this programmatically.

Quartus Prime generates an *SRAM Object File* (`.sof`) as its default FPGA target image. However, the HPS can only program the FPGA by using a *Raw Binary File* (`.rbf`). Therefore, we must convert our `.sof` file to a `.rbf` to later satisfy this requirement.

47. Execute the following command to convert the `.sof` file to a `.rbf` file.

```
$ quartus_cpf -c \
  DE1_SoC_demo/hw/quartus/output_files/DE1_SoC_demo.sof \
  DE1_SoC_demo/sdcard/fat32/socfpga.rbf
```

10 USING THE CYCLONE V – FPGA – NIOS II – BARE-METAL

10.1 PROJECT SETUP

1. Launch the *Nios II SBT* IDE by executing the following command.
   `$ eclipse-nios2`

2. Choose "`File > New > Nios II Application and BSP from Template`".
   a. All the information needed to program a Nios II processor is contained within the "`.sopcinfo`" file created by *Qsys*. For the "`SOPC Information File name`" use "`DE1_SoC_demo/hw/quartus/soc_system.sopcinfo`".
   b. Use "`DE1_SoC_demo_nios`" as the project name.
   c. Disable the "`Use default location`" checkbox
   d. Use "`DE1_SoC_demo/sw/nios/application/DE1_SoC_demo_nios`" as the project location.
   e. Choose the "`Blank Project`" template.
   f. Click on the "`Finish`" button to create the project.
3. Right-click on the "`DE1_SoC_demo_nios`" project folder and select "`New > Source file`". Use the default C source template, and set "`nios.c`" as the file name.
4. Right-click on the "`DE1_SoC_demo_nios_bsp`" project, and select "`Build Project`". Once the build is completed, a number of files will be generated, the most useful of which is the "`system.h`" file. This file contains all the details related to the Nios II processor's various peripherals, as defined in *Qsys* in 9.2.

10.2 NIOS II PROGRAMMING THEORY – ACCESSING PERIPHERALS

The Nios II processor can be programmed in C similarly to any other microcontroller. However, care must be taken when accessing any of the processor's peripherals. Depending on which version of the Nios II you instantiated in *Qsys*, you may not be able to correctly read data at a peripheral's address space using pointers. The issue arises when your Nios II processor has a *data cache*.

Suppose we use the code in Figure 10-1 to read data from the switches of our *Qsys* design.

```
#include <stdbool.h>
#include <inttypes.h>
#include "system.h"

int main() {
    uint32_t *p_switches = SWITCHES_0_BASE;
    while (true) {
        alt_u32 switches_value = *p_switches;
        printf("switches_value = %" PRIx32 "\n", switches_value);
    }
    return 0;
}
```

Figure 10-1. Incorrect Nios II Peripheral Access in C

When this code is run, the initial value of the "`switches_value`" variable, as obtained from the first iteration of the `while` loop, will be the correct representation of the switches' state. However, at each iteration of the `while` loop, the "`switches_value`" variable will ***NEVER*** change again, even if the switches are flipped between each iteration. The issue is that each successive access is being served by the data cache, which doesn't see that the switches have been modified.

The solution to this issue is to use special instructions that bypass the data cache when reading or writing to peripherals. These instructions are part of the IO family of load and store instructions and bypass all caches.

The available instructions are listed below, and an example of how to correctly access Nios II peripherals is shown in Figure 10-2.

- Reading
    - IORD_8DIRECT(BASE, OFFSET)
    - IORD_16DIRECT(BASE, OFFSET)
    - IORD_32DIRECT(BASE, OFFSET)
- Writing
    - IOWR_8DIRECT(BASE, OFFSET, DATA)
    - IOWR_16DIRECT(BASE, OFFSET, DATA)
    - IOWR_32DIRECT(BASE, OFFSET, DATA)

```
#include <stdbool.h>
#include <inttypes.h>
#include "system.h"
#include “io.h”

int main() {
    while (true) {
        uint32_t switches_value = IORD_32DIRECT(SWITCHES_0_BASE, 0);
        printf("switches_value = %" PRIx32 "\n", switches_value);    }
    return 0;
}
```

Figure 10-2. Correct Nios II Peripheral Access in C

10.3 NIOS II PROGRAMMING PRACTICE

5. Write the code provided in Figure 10-3 in “nios.c”. The code instructs the Nios II processor to create a strobing light effect on its 10 peripheral LEDs. The processor’s 10 peripheral switches are used as enable signals for each corresponding LED. This corresponds to specification 1 in 8.4.

```
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include "io.h"
#include "altera_avalon_pio_regs.h"
#include "system.h"

#define LEDS_MAX_ITERATION (1000)
#define SLEEP_DELAY_US (100 * 1000)

void rotate_leds() {
    int loop_count = 0;
    int leds_mask = 0x01;

    // 0/1 = left/right direction
    int led_direction = 0;

    while (loop_count < LEDS_MAX_ITERATION) {
        uint32_t switches_value = IORD_ALTERA_AVALON_PIO_DATA(SWITCHES_0_BASE);
        uint32_t leds_value = ~leds_mask;

        // only turn on leds which have their corresponding switch enabled
        IOWR_ALTERA_AVALON_PIO_DATA(LEDS_0_BASE, leds_value & switches_value);

        usleep(SLEEP_DELAY_US);

        if (led_direction == 0) {
```

```
            leds_mask <<= 1;
            if (leds_mask == (0x01 << (LEDS_0_DATA_WIDTH - 1))) {
                led_direction = 1;
            }
        } else {
            leds_mask >>= 1;
            if (leds_mask == 0x01) {
                led_direction = 0;
                loop_count++;
            }
        }
    }
}

int main() {
    rotate_leds();
    return 0;
}
```

Figure 10-3. nios.c

6. Right-click on "DE1_SoC_demo_nios" project, and select "Build Project".
7. The code is now ready to be run on the FPGA. Right-click on "DE1_SoC_demo_nios" project, and select "Run As > Nios II Hardware". You should be able to see a strobing light effect on the 10 FPGA LEDs. You can use the 10 FPGA switches as enable signals for the corresponding LED.
8. In some cases, it is possible that the program will not immediately run on the Nios II processor, and you will be prompted with a "Target Connection" dialog, as shown in Figure 10-4. If your Nios II CPU doesn't appear in the list of available processors, then
    a. Click on the "Refresh Connections" button on the right of Figure 10-4.
    b. Click on the "Run" button to finish.

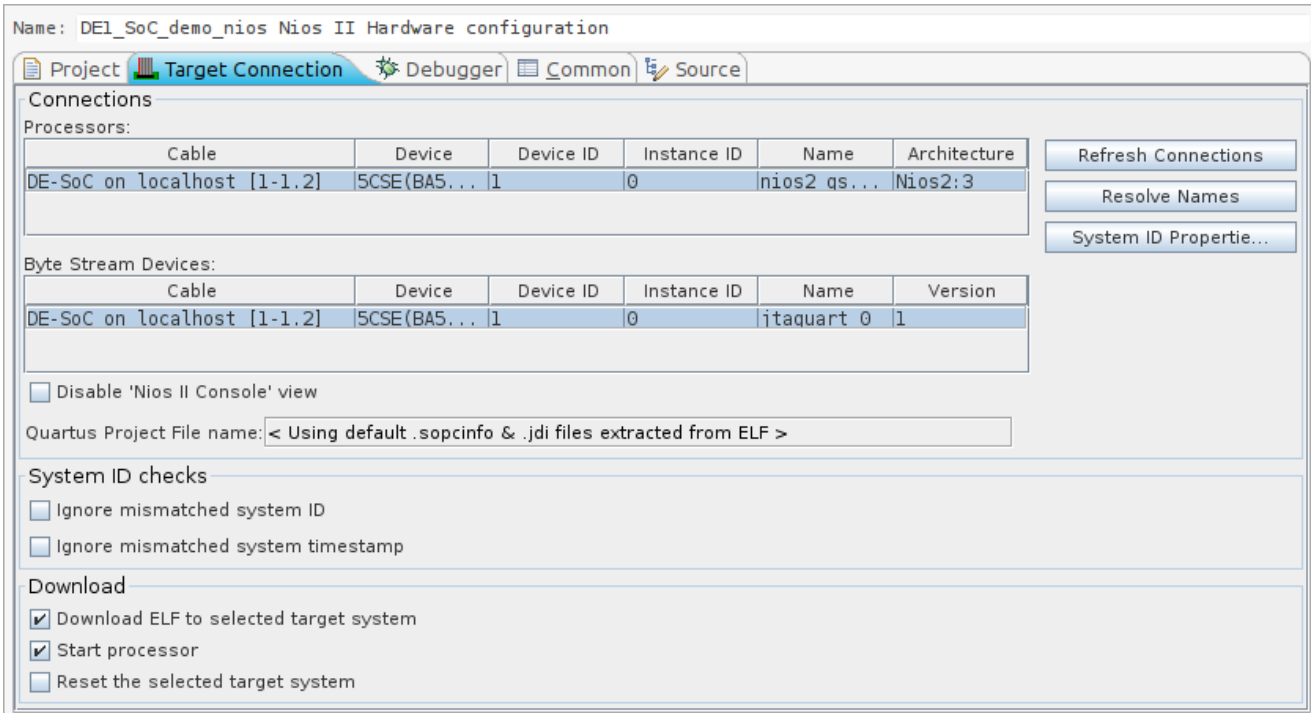


Figure 10-4. Nios II Target Connection Dialog

We now have a programmed Nios II processor on the FPGA. Of course, the design we had specified didn't require the power of a Nios II processor, and could have easily been done in pure VHDL. Nevertheless, the

idea was to show that one can have a secondary programmable processor functioning on the FPGA parallelly to the HPS. We are now done with the *Nios II SBT* IDE, and will no longer need it for the rest of this tutorial.

11 USING THE CYLONE V – HPS – ARM – GENERAL

11.1 PARTITIONING THE SDCARD

The DE1-SoC needs to boot off of a microSD card, so we need to partition it appropriately before we can write to it.

1. Plug your sdcard into your computer.
2. Find out the device's identifier. When writing this tutorial, the sdcard was recognized as entry "`/dev/sdb`" on my computer.

*Please be careful and choose the correct /dev/sdX entry for your sdcard. Failure to do so will ensure that the following commands will **WIPE THE WRONG PARTITION OFF OF YOUR MACHINE**, which will be a most unfortunate outcome!*

3. Wipe the partition table of the sdcard by executing the following command.
   ```
   $ sudo dd if="/dev/zero" of="/dev/sdb" bs=512 count=1
   ```

4. Manually partition the device by using the "`fdisk`" command. "`fdisk`" is an interactive program, so you have to interactively provide the configuration of your device. You can do this by using the following sequence of commands whenever "`fdisk`" prompts you for what to do.

```
$ sudo fdisk /dev/sdx
# use the following commands
    # n p 3 <default> 4095  t   a2 (2048 is default first sector)
    # n p 1 <default> +32M  t 1  b (4096 is default first sector)
    # n p 2 <default> +512M t 2 83 (69632 is default first sector)
    # w
```

Figure 11-1. Partitioning the sdcard

5. Create the required filesystems on the device. We need a FAT32 partition for various boot-time files (FPGA raw binary file, Linux kernel zImage file, u-boot configuration script …), and an EXT3 partition for the Linux root filesystem.
   ```
   $ sudo mkfs.vfat "/dev/sdb1"
   $ sudo mkfs.ext3 -F "/dev/sdb2"
   ```

11.2 GENERATING A HEADER FILE FOR HPS PERIPHERALS

We need the HPS to be able to programmatically access peripherals that are part of the FPGA fabric. In order to do this, we must generate a header file.

1. Execute the following command.
   ```
   $ sopc-create-header-files \
     DE1_SoC_demo/hw/quartus/soc_system.sopcinfo \
     --single hps_soc_system.h
     --module hps_0
   ```

2. Move the generate header file to the "`DE1_SoC_demo/sw/hps/application`" directory.
   ```
   $ mv hps_soc_system.h DE1_SoC_demo/sw/hps/application/
   ```

Figure 11-2 shows a short extract of the generated "`hps_soc_system.h`" header file. At the top of the file, it says that macros for devices connected to master port "`h2f_lw_axi_master`" of module "`hps_0`" have been generated.

```
/*
 * This file was automatically generated by the swinfo2header utility.
 *
```

```
 * Created from SOPC Builder system 'soc_system' in
 * file hw/quartus/soc_system.sopcinfo'.
 */

/*
 * This file contains macros for module 'hps_0' and devices
 * connected to the following master:
 *   h2f_lw_axi_master
 *
 * Do not include this header file and another header file created for a
 * different module or master group at the same time.
 * Doing so may result in duplicate macro names.
 * Instead, use the system header file which has macros with unique names.
 */

/*
 * Macros for device 'hex_5', class 'altera_avalon_pio'
 * The macros are prefixed with 'HEX_5_'.
 * The prefix is the slave descriptor.
 */
#define HEX_5_COMPONENT_TYPE altera_avalon_pio
#define HEX_5_COMPONENT_NAME hex_5
#define HEX_5_BASE 0x0
#define HEX_5_SPAN 16
#define HEX_5_END 0xf
```

Figure 11-2. hps_soc_system.h

11.3 HPS PROGRAMMING THEORY

The HPS works just like any other "microcontroller".

- If you want to access a peripheral, you have to read/write at its address.
- If a peripheral is connected to a bus, its address is obtained by adding its offset in the bus to the bus' address.

Altera provides useful utility functions in "`<altera_install_directory>/<version>/embedded/ip/altera/hps/altera_hps/hwlib/include/soc_cv_av/socal/socal.h`", a few of which are listed below. Most functions exist for multiple sizes. These sizes are summarized in Table 11-1. Note that "socal" means "SoC Abstraction Layer".

- `alt_write_byte(dest_addr, byte_data)`
- `alt_read_byte(src_addr)`
- `alt_setbits_byte(dest_addr, byte_data)`
- `alt_clrbits_byte(dest_addr, byte_data)`
- `alt_xorbits_byte(dest_addr, byte_data)`
- `alt_replbits_byte(dest_addr, msk, byte_data)`

Name	Size (bits)
byte	8
hword	16
word	32
dword	64

Table 11-1. Predefined Data Sizes in socal.h

Up until this point, the hardware and software design process has been ***IDENTICAL*** for both ***BARE-METAL*** and ***LINUX HPS*** applications. This is where the design process ***DIVERGES*** between bare-metal and Linux HPS

applications. If you want to write a bare-metal application for the HPS, then read section 12. If instead you want to write a linux application for the HPS, then read section 13.

Note: In addition to the example used in this tutorial, you can find many more in "<altera_install_directory>/<version>/embedded/examples/software/"

12 USING THE CYCLONE V – HPS – ARM – BARE-METAL

12.1 PRELOADER

In Figure 7-8, we saw that a bare-metal application can only be launched after the preloader has setup the HPS. So, the first thing that needs to be done for bare-metal applications is to generate and compile a preloader for the HPS.

12.1.1 Preloader Generation

1. Execute the following command to launch the preloader generator.
   `$ bsp-editor`

2. Choose "`File > New BSP…`".
   a. The preloader will need to know which of the HPS' peripherals were enabled so it can appropriately initialize them in the boot process. Under "`Preloader settings directory`", select the "`DE1_SoC_demo/hw/quartus/hps_isw_handoff/soc_system_hps_0`" directory. This directory contains settings relative to the HPS' ***HARD*** peripherals, as configured in the "`Arria V/Cyclone V Hard Processor System`" component in *Qsys*.
   b. Disable the "`Use default locations`" checkbox and under the "`BSP target directory`", select the "`DE1_SoC_demo/sw/hps/preloader`" directory. You should have something similar to Figure 12-1.

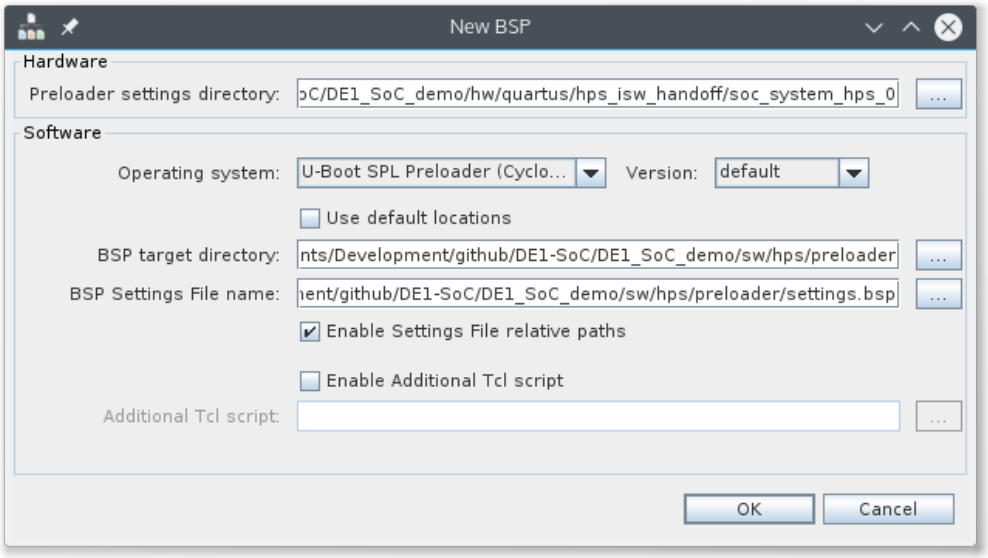


Figure 12-1. New BSP Dialog

   c. Press the "`OK`" button. You should then arrive on a page with many settings, as shown on Figure 12-2. Take some time to read through them to see what the preloader has the ability to do.

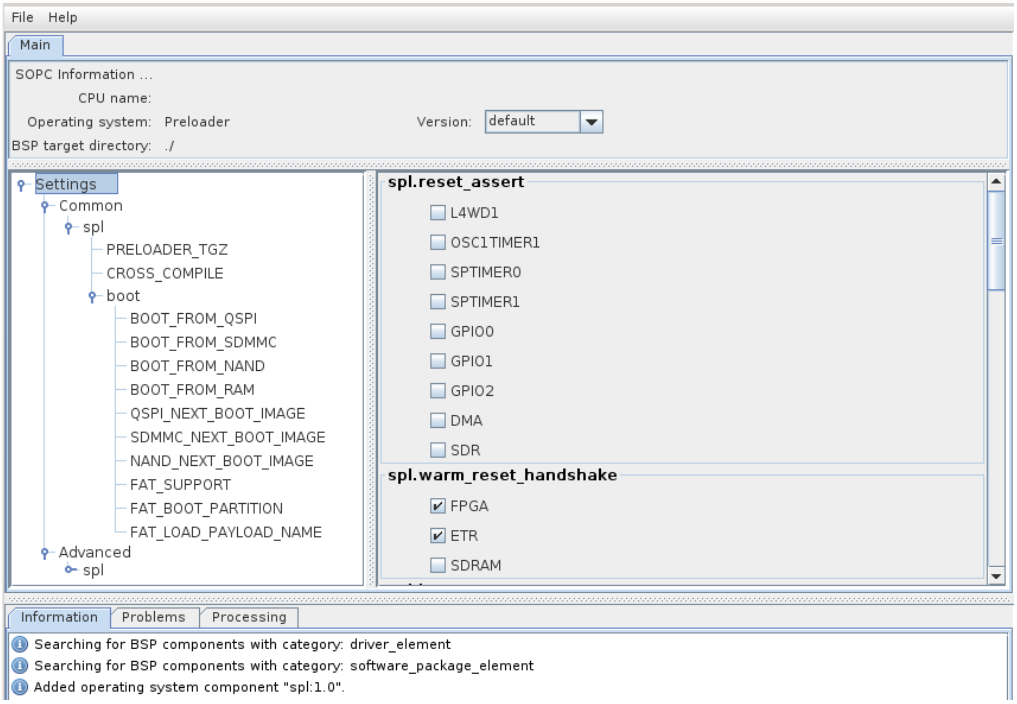


Figure 12-2. Preloader Settings Dialog

3. On the main settings page of Figure 12-2, we will only need to modify 3 parameters for our design.
    a. Under "`spl.boot`", disable the "`WATCHDOG_ENABLE`" checkbox. This is necessary to prevent the system from being automatically reset after a certain time has elapsed. Note that we only disable this option since we intend on writing a bare-metal program and want to simplify the code. Any operating system would periodically write to the watchdog timer to avoid it from resetting the system, and this is a good thing.
    b. Press the "`Generate`" button to finish. You can then exit the `bsp-editor`.
4. Execute the following command to build the preloader.

```
$ make -C DE1_SoC_demo/sw/hps/preloader
```

IF YOU EVER DECIDE TO MOVE THE "`DE1_SoC_demo`" PROJECT DIRECTORY DEFINED IN FIGURE 8-1, YOU WILL HAVE TO REGENERATE THE PRELOADER. UNFORTUNATELY, THE SCRIPT PROVIDED BY ALTERA WHICH GENERATES THE PRELOADER HARD-CODES MULTIPLE ABSOLUTE PATHS DIRECTLY IN THE RESULTING FILES, RENDERING THEM USELESS ONCE MOVED.

12.1.2 Creating Target sdcard Artifacts

5. Copy the preloader binary to the sdcard target directory. Execute the following command.

```
$ cp \
  DE1_SoC_demo/sw/hps/preloader/preloader-mkpimage.bin \
  DE1_SoC_demo/sdcard/a2/preloader-mkpimage.bin
```

12.2 ARM DS-5

6. Launch the *ARM DS-5* IDE by executing the following command.

```
$ eclipse
```

BE SURE YOU ARE PART OF THE "`dialout`" GROUP BEFORE YOU LAUNCH ARM DS-5, OTHERWISE YOU WON'T BE ABLE TO ACCESS THE SERIAL CONSOLE ON YOUR MACHINE IN ORDER TO CONNECT TO THE DE1-SOC.

12.2.1 Setting Up a New C Project

7. Create a new C project by going to "`File > New > Project > C/C++ > C Project`".
    a. Use "`DE1_SoC_demo_hps_baremetal`" as the project name.

b. Disable the "Use default location" checkbox.
c. Set "DE1_SoC_demo/sw/hps/application/DE1_SoC_demo_hps_baremetal" as the target location for the project.
d. We want to create a single output executable for our project, so choose "Executable > Empty Project" as the project type.
e. Choose "Altera Baremetal GCC" as the Toolchain.
f. You should have something similar to Figure 12-3. Then, press the "Finish" button to create the project.

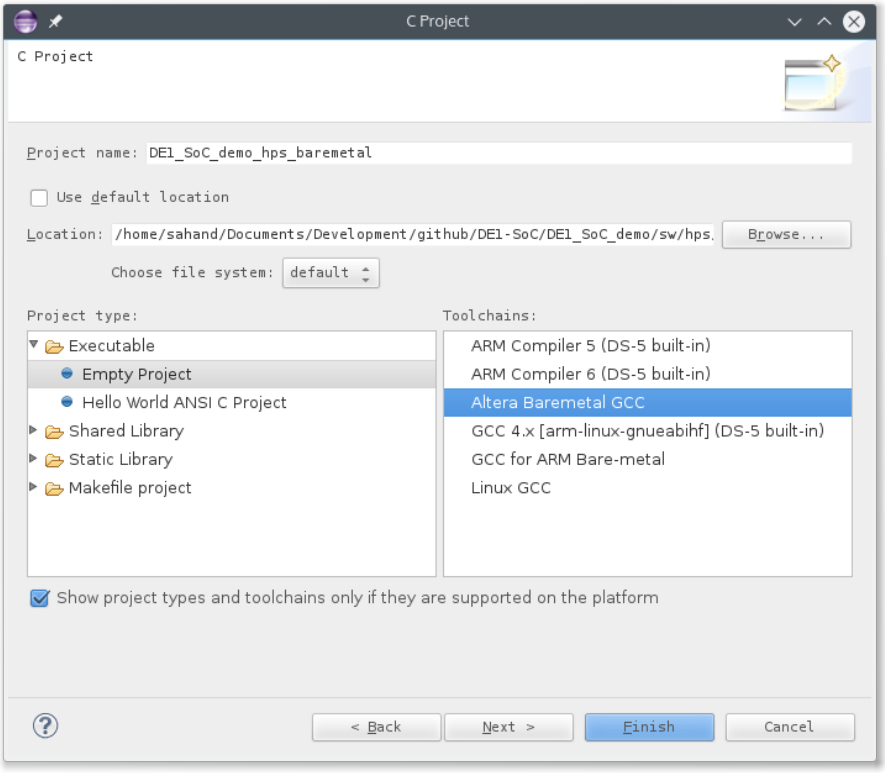


Figure 12-3. New C Project Dialog

8. When programming the HPS, we will need access to a few standard header and linker files provided by Altera. We need to add these files to the *ARM DS-5* project.
   a. Right-click on the "DE1_SoC_demo_hps_baremetal" project, and go to "Properties".
   b. We are going to use Altera's HWLIB to develop our bare-metal application, so we need to define a macro that is needed by the library to know which board is being targetted.
   Under "C/C++ Build > Settings > GCC C Compiler > Symbols", add "soc_cv_av" to the "Defined symbols (-D)" list.
   c. Under "C/C++ Build > Settings > GCC C Compiler > Includes", add "<altera_install_directory>/<version>/embedded/ip/altera/hps/altera_hps/hwlib/include" to the "Include paths (-I)" list.
   d. Under "C/C++ Build > Settings > GCC C Compiler > Includes", add "<altera_install_directory>/<version>/embedded/ip/altera/hps/altera_hps/hwlib/include/soc_cv_av" to the "Include paths (-I)" list.
   e. Since we are not going to be running any operating system, we will need to use a linker script in order to correctly layout our bare-metal program in memory. Altera provides linker scripts for the HPS' on-chip memory, as well as for it's DDR3 memory. We want our code to be loaded in the HPS' DDR3 memory and will not use any on-chip memory in our design, so we will use the DDR3 linker script.
   Under "C/C++ Build > Settings > GCC C Linker > Image", set the linker script to "<altera_install_directory>/<version>/embedded/host_tools/mentor/gnu/arm/bar

emetal/arm-altera-eabi/lib/cycloneV-dk-ram-***HOSTED***.ld". The "hosted" script allows the bare-metal application to use some of the host's functionality. In this case, we use the "hosted" script to be able to see the output of the printf() function on the host's console.

f. Click on the "Apply" button, then on the "Ok" button to close the project properties dialog.

12.2.2 Writing a DS-5 Debug Script

In Figure 7-8, we saw that a bare-metal application cannot run immediately upon boot, and that the HPS must first go through the preloader. The preloader executes, and, before terminating, it jumps to the next stage of the user software. In the case of a bare-metal application, the preloader jumps to the start of the bare-metal code.

Jumping directly to the bare-metal code is useful for production environments, but it would be great if we could use a debugger when testing our bare-metal code. To do this, we will use a ***DS-5 DEBUG SCRIPT*** to instruct the DS-5 debugger exactly how to load our application in the HPS' memory. This debugger script will load and execute the preloader, then jump to our bare-metal code.

9. Create a new file for our *DS-5* debug script and save it under "DE1_SoC_demo/sw/hps/application/DE1_SoC_demo_hps_baremetal/debug_setup.ds".
10. Populate the file with the code shown in Figure 12-4. This script tells the debugger to load the prealoder, then to load our bare-metal application. This is performed by placing a breakpoint at the very last function executed by the preloader prior to handing control of the cpu to the next boot stage. This function is "spl_boot_device()", which is responsible for choosing the next boot medium on the DE1-SoC and jumping to it's address. For bare-metal applications, we don't want the boot process to continue on towards another device. Instead, we want to load our bare-metal code and jump to it's address. This is exactly what the debug script in Figure 12-4 does.

```
# Reset and stop the system.
stop
wait 5s
reset system
wait 5s

# Delete all breakpoints.
delete breakpoints

# Disable semihosting
set semihosting enabled false

# Load the preloader.
loadfile "$sdir/../../preloader/uboot-socfpga/spl/u-boot-spl" 0x0

# Enable semihosting to allow printing even if you don't have a uart module
# available.
set semihosting enabled true

# Set a breakpoint at the "spl_boot_device()" function. This function is the
# last step of the preloader. It looks for a boot device (qspi flash, sdcard,
# fpga), and jumps to that address. For our bare-metal programs, we don't want
# to use any boot device, but want to run our own program, so we want the
# processor to stop here. Then, we will modify its execution to make it run our
# program.
tbreak spl_boot_device

# Set the PC register to the entry point address previously recorded by the
# "load" or "loadfile" command and start running the target.
run
```

```
# Instruct the debugger to wait until either the application completes or a
# breakpoint is hit. In our case, it will hit the breakpoint.
wait

# Load our bare-metal program.
loadfile "$sdir/Debug/DE1_SoC_demo_hps_baremetal.axf"

# Set a breakpoint at our program's "main()" function.
tbreak main

# Start running the target.
run

# wait at main().
Wait
```

Figure 12-4. debug_setup.ds

For a comprehensive list of commands supported by the *DS-5* debugger, please refer to [9].

12.2.3 Setting Up the Debug Configuration

11. Right-click on the "DE1_SoC_demo_hps_baremetal" project, and go to "Debug As > Debug Configurations…".
12. Choose to create a new debugger configuration by right-clicking on "DS-5 Debugger" on the left and selecting "New". Use "DE1_SoC_demo_hps_baremetal" as the name of the new debug configuration.
13. Under the "Connection" tab:
    a. Use "Altera > Cyclone V SoC (Dual Core) > Bare Metal Debug > Debug Cortex-A9_0" as the target platform.
    b. Set the "Target Connection" to "USB-Blaster".
    c. Use the "Browse" button to select the DE1-SoC that is connected to your machine.
    d. You should have something similar to Figure 12-5.

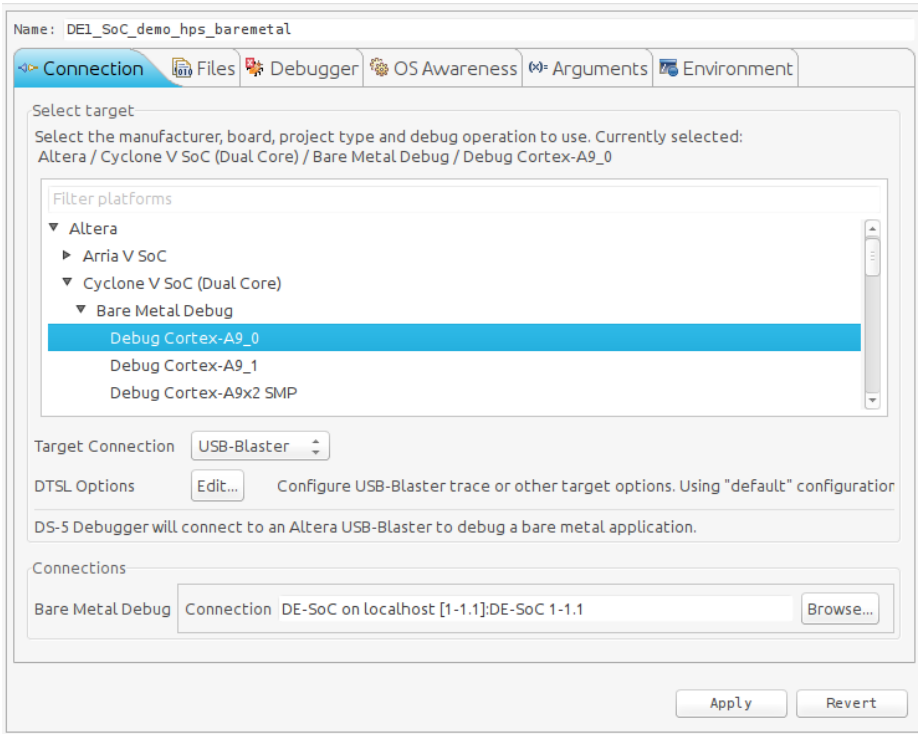


Figure 12-5. Debug Configuraton "Connection" Tab

14. Under the "Files" tab:

a. Leave the "Application on host to download" empty. We do this since we are using a debug script to instruct the debugger how to load our application.
b. In 9.3.2, we configured our HPS to use some FPGA peripherals. We can instruct the debugger about this so it can show more detailed information when debugging. To do this, set the combobox to "Add peripheral description files from directory" and set it to the "DE1_SoC_demo/hw/quartus/soc_system/synthesis" directory, as shown in Figure 12-6. This directory contains a file called "soc_system_hps_0_hps.svd" which has information on all of the HPS' peripherals which are in the FPGA fabric.

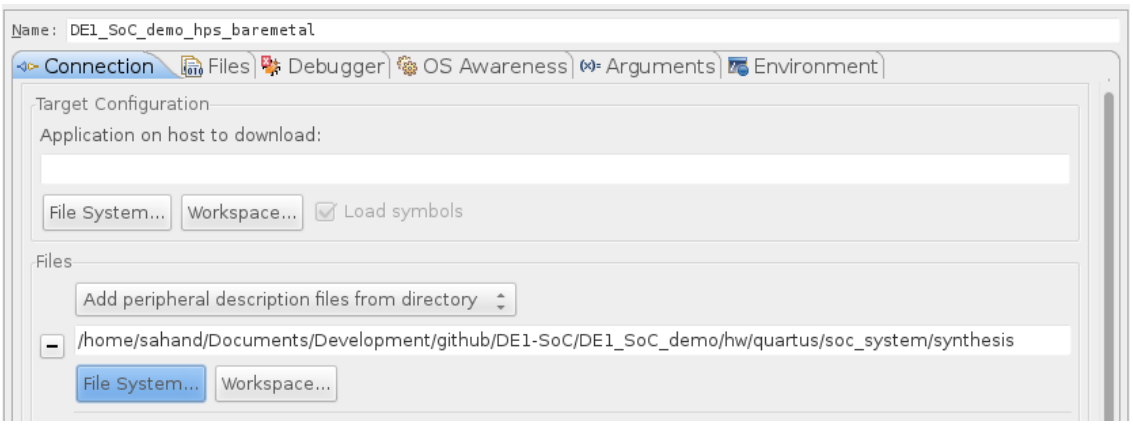


Figure 12-6. Debug Configuration "Files" Tab

15. Under the "Debugger" tab:
    a. Since we are going to use a debug script to launch the application, we don't need to specify any function to be loaded by the debugger. So, choose "Connect only" under "Run control".
    b. Enable the "Run ***DEBUG*** initialization debugger script (.ds / .py)" checkbox. Set the debug script to the one we defined for the project in 12.2.2. You should have something similar to Figure 12-7.
16. Click on the "Apply" button, then on the "Close" button to save the debug configuration.

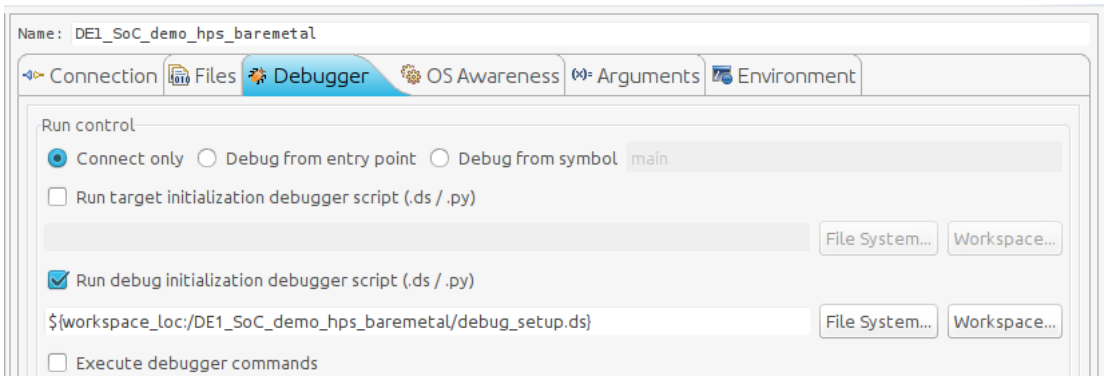


Figure 12-7. Debug Configuration "Debugger" Tab

12.2.4 Bare-metal Programming

We can now start writing bare-metal code for the HPS.

17. Right-click on the "DE1_SoC_demo_hps_baremetal" project, and go to "New > Source File". Use "hps_baremetal.c" as the file name, and click on the "Finish" button to create the new source file.
18. Right-click on the "DE1_SoC_demo_hps_baremetal" project, and go to "New > Header File". Use "hps_baremetal.h" as the file name, and click on the "Finish" button to create the new header file.

The code for this part of the application is quite large to be inserted in this document. Therefore, we will just go over a few practical aspects of the code which are worth paying attention to. The full source can be found in `DE1_SoC_demo.zip` [3].

We are not going to implement any interrupts for the various buttons on the board at this time. Therefore, in order to satisfy the HPS-related goals specified in 8.4, we will need to use an infinite loop and do some polling.

This can be seen in our application's "`main()`" function, which is shown in Figure 12-8.

```
int main() {
    printf("DE1-SoC bare-metal demo\n");

    setup_peripherals();

    uint32_t hex_counter = 0;
    while (true) {
        handle_hex_displays(&hex_counter);
        handle_hps_led();
        delay_us(ALT_MICROSECS_IN_A_SEC / 10);
    }

    return 0;
}
```

Figure 12-8. hps_baremetal.c main() function

12.2.4.1 Accessing FPGA Peripherals

Accessing the FPGA peripherals connected to the HPS' *lightweight* HPS-to-FPGA bridge is quite simple, as no libraries are needed. One can simply use the low-level functions listed in 11.3 to address the peripherals at an offset from the *lightweight* HPS-to-FPGA bridge's base address.

Figure 12-9 shows an example where the HPS accesses the buttons on the FPGA.

```
// fpga buttons can be found at an offset from the base of the lightweight HPS-to-FPGA bridge
void *fpga_buttons = ALT_LWFPGASLVS_ADDR + BUTTONS_0_BASE;

bool is_fpga_button_pressed(uint32_t button_number) {
    // buttons are active-low
    return ((~alt_read_word(fpga_buttons)) & (1 << button_number));
}
```

Figure 12-9. Accessing FPGA Buttons from the HPS

A more sophisticated example can be found in Figure 12-10, where the HPS sets the value to be displayed on the FPGA's 7-segment displays.

```
// The 7-segment display is active low
#define HEX_DISPLAY_CLEAR (0x7F)
#define HEX_DISPLAY_ZERO  (0x40)
#define HEX_DISPLAY_ONE   (0x79)
#define HEX_DISPLAY_TWO   (0x24)
#define HEX_DISPLAY_THREE (0x30)
#define HEX_DISPLAY_FOUR  (0x19)
#define HEX_DISPLAY_FIVE  (0x12)
#define HEX_DISPLAY_SIX   (0x02)
#define HEX_DISPLAY_SEVEN (0x78)
#define HEX_DISPLAY_EIGHT (0x00)
#define HEX_DISPLAY_NINE  (0x18)
#define HEX_DISPLAY_A     (0x08)
#define HEX_DISPLAY_B     (0x03)
#define HEX_DISPLAY_C     (0x46)
#define HEX_DISPLAY_D     (0x21)
```

```
#define HEX_DISPLAY_E     (0x06)
#define HEX_DISPLAY_F     (0x0E)

// The HPS will only use HEX_DISPLAY_COUNT of the 6 7-segment displays
#define HEX_DISPLAY_COUNT (6)
#define HEX_COUNTER_MASK  ((1 << (4 * HEX_DISPLAY_COUNT)) - 1)

void *fpga_hex_displays[HEX_DISPLAY_COUNT] = {ALT_LWFPGASLVS_ADDR + HEX_0_BASE,
                                              ALT_LWFPGASLVS_ADDR + HEX_1_BASE,
                                              ALT_LWFPGASLVS_ADDR + HEX_2_BASE,
                                              ALT_LWFPGASLVS_ADDR + HEX_3_BASE,
                                              ALT_LWFPGASLVS_ADDR + HEX_4_BASE,
                                              ALT_LWFPGASLVS_ADDR + HEX_5_BASE};

uint32_t hex_display_table[16] = {HEX_DISPLAY_ZERO , HEX_DISPLAY_ONE,
                                  HEX_DISPLAY_TWO  , HEX_DISPLAY_THREE,
                                  HEX_DISPLAY_FOUR , HEX_DISPLAY_FIVE,
                                  HEX_DISPLAY_SIX  , HEX_DISPLAY_SEVEN,
                                  HEX_DISPLAY_EIGHT, HEX_DISPLAY_NINE,
                                  HEX_DISPLAY_A    , HEX_DISPLAY_B,
                                  HEX_DISPLAY_C    , HEX_DISPLAY_D,
                                  HEX_DISPLAY_E    , HEX_DISPLAY_F};

void set_hex_displays(uint32_t value) {
    char current_char[2] = " \0";
    char hex_counter_hex_string[HEX_DISPLAY_COUNT + 1];

    // get hex string representation of input value on HEX_DISPLAY_COUNT 7-segment displays
    snprintf(hex_counter_hex_string, HEX_DISPLAY_COUNT + 1, "%0*x", HEX_DISPLAY_COUNT, (unsigned int) value);

    uint32_t hex_display_index = 0;
    for (hex_display_index = 0; hex_display_index < HEX_DISPLAY_COUNT; hex_display_index++) {
        current_char[0] = hex_counter_hex_string[HEX_DISPLAY_COUNT - hex_display_index - 1];

        // get decimal representation for this 7-segment display
        uint32_t number = (uint32_t) strtol(current_char, NULL, 16);

        // use lookup table to find active-low value to represent number on the 7-segment display
        uint32_t hex_value_to_write = hex_display_table[number];

        alt_write_word(fpga_hex_displays[hex_display_index], hex_value_to_write);
    }
}
```

Figure 12-10. Setting the 7-Segment Displays from the HPS

12.2.4.2 Accessing HPS Peripherals

It is possible to do everything with the low-level functions listed in 11.3. However, a better way would be to use Altera's ***HWLIB***, as discussed In 7.5.2. You can easily use *HWLIB* to access all the HPS' ***HARD*** peripherals.

Note that some things may not be available in *HWLIB*, and you will then have to resort to using the low-level functions. One example of this scenario which we have already seen is when accessing any FPGA peripherals through the *lightweight* or *heavyweight* HPS-to-FPGA bus (as there is no standard header file for any FPGA peripherals).

Since we already demonstrated how to use low-level functions to access peripherals in 12.2.4.1, we will instead use Altera's *HWLIB* to access the HPS' hard peripherals.

12.2.4.2.1 Using Altera's HWLIB - Prerequisites

In order to be able to use *HWLIB* to configure a peripheral, 2 steps need to be performed:

- You need to ***INCLUDE*** the HPS peripheral's *HWLIB* ***HEADER FILE*** to your code.
- You must ***COPY*** the HPS peripheral's *HWLIB* ***SOURCE FILE*** in your *DS-5* project directory. The *HWLIB* source files can be found in directory "`<altera_install_directory>/<version>/embedded/ip/altera/hps/altera_hps/hwlib/src`", and must be copied to "`DE1_SoC_demo/sw/hps/application/DE1_SoC_demo_hps_baremetal`".

12.2.4.2.2 Global Timer & Clock Manager

If you look closely at the code in Figure 12-8, you'll see that we used a "`delay_us()`" function to slow the counter down. It turns out that among all the code available for the HPS, Altera does not provide any "`sleep()`" function (unlike for the Nios II processor). Therefore, we will have to write the "`delay_us()`" function ourselves.

The easiest way to create a delay in the HPS is to use one of it's timers. There are numerous timers on Cyclone V SoCs:

- One such timer is the ***GLOBAL TIMER***. This timer is actually shared by both HPS cores, as well as by the FPGA.
- In addition to the unique global timer, each HPS core also has 7 other timers which it can use exclusively, if needed.

For simplicity, we will use the global timer to implement the "`delay_us()`" function.

As described in 12.2.4.2.1, we need to add the required *HWLIB* sources to our project, and their headers to our code. To program the global timer, we will need information regarding the clock frequency, as well as any timer-specific functions. We can access this information by using the following source and header files:

- `alt_clock_manager.c`
- `alt_clock_manager.h`
- `alt_globaltmr.c`
- `alt_globaltmr.h`

Figure 12-11 shows how we implement the "`delay_us()`" function using the global timer.

```
#include "alt_clock_manager.h"
#include "alt_globaltmr.h"

void setup_hps_timer() {
    assert(ALT_E_SUCCESS == alt_globaltmr_init());
}

/* The HPS doesn't have a sleep() function like the Nios II, so we can make one
 * by using the global timer. */
void delay_us(uint32_t us) {
    uint64_t start_time = alt_globaltmr_get64();
    uint32_t timer_prescaler = alt_globaltmr_prescaler_get() + 1;
    uint64_t end_time;
    alt_freq_t timer_clock;

    assert(ALT_E_SUCCESS == alt_clk_freq_get(ALT_CLK_MPU_PERIPH, &timer_clock));
    end_time = start_time + us * ((timer_clock / timer_prescaler) / ALT_MICROSECS_IN_A_SEC);

    // polling wait
    while(alt_globaltmr_get64() < end_time);
}
```

Figure 12-11. Programming the HPS Global Timer

12.2.4.2.3 GPIO

Figure 12-12 shows how we implement the "handle_hps_led()" function. This function uses the HPS_KEY_N button to toggle HPS_LED.

Once again, we need to add the *HWLIB* source file for the GPIO peripheral to our *DS-5* project directory. The files we will use are listed below:

- alt_generalpurpose_io.c
- alt_generalpurpose_io.h

As stated in 12.2.4.2 previously, *HWLIB* is quite a broad library, but it sometimes lacks certain "obvious" things. In such cases, you have to fall back on using lower-level functions to implement whatever you are missing.

In our case, we see that *HWLIB* has functions that allow us to write to the GPIO peripheral's "data" register, but it doesn't have any function to read the it back. We get around this issue by directly reading the register with "alt_read_word(ALT_GPIO1_SWPORTA_DR_ADDR)".

Note that we also need to include the "socal/alt_gpio.h" header file to have access to the lower-level ALT_GPIO1_SWPORTA_DR_ADDR macro.

```
#include "alt_generalpurpose_io.h"
#include "socal/alt_gpio.h"

// |=============|==========|==============|==========|
// | Signal Name | HPS GPIO | Register/bit | Function |
// |=============|==========|==============|==========|
// |   HPS_LED   |  GPIO53  |   GPIO1[24]  |    I/O   |
// |  HPS_KEY_N  |  GPIO54  |   GPIO1[25]  |    I/O   |
// |=============|==========|==============|==========|
#define HPS_LED_IDX        (ALT_GPIO_1BIT_53)                      // GPIO53
#define HPS_LED_PORT       (alt_gpio_bit_to_pid(HPS_LED_IDX))      // ALT_GPIO_PORTB
#define HPS_LED_PORT_BIT   (alt_gpio_bit_to_port_pin(HPS_LED_IDX)) // 24 (from GPIO1[24])
#define HPS_LED_MASK       (1 << HPS_LED_PORT_BIT)
#define HPS_KEY_N_IDX      (ALT_GPIO_1BIT_54)                        // GPIO54
#define HPS_KEY_N_PORT     (alt_gpio_bit_to_pid(HPS_KEY_N_IDX))      // ALT_GPIO_PORTB
#define HPS_KEY_N_PORT_BIT (alt_gpio_bit_to_port_pin(HPS_KEY_N_IDX)) // 25 (from GPIO1[25])
#define HPS_KEY_N_MASK     (1 << HPS_KEY_N_PORT_BIT)

void setup_hps_gpio() {
    uint32_t hps_gpio_config_len = 2;
    ALT_GPIO_CONFIG_RECORD_t hps_gpio_config[] = {
        {HPS_LED_IDX  , ALT_GPIO_PIN_OUTPUT, 0, 0, ALT_GPIO_PIN_DEBOUNCE, ALT_GPIO_PIN_DATAZERO},
        {HPS_KEY_N_IDX, ALT_GPIO_PIN_INPUT , 0, 0, ALT_GPIO_PIN_DEBOUNCE, ALT_GPIO_PIN_DATAZERO}
    };

    assert(ALT_E_SUCCESS == alt_gpio_init());
    assert(ALT_E_SUCCESS == alt_gpio_group_config(hps_gpio_config, hps_gpio_config_len));
}

void handle_hps_led() {
    uint32_t hps_gpio_input = alt_gpio_port_data_read(HPS_KEY_N_PORT, HPS_KEY_N_MASK);

    // HPS_KEY_N is active-low
    bool toggle_hps_led = (~hps_gpio_input & HPS_KEY_N_MASK);

    if (toggle_hps_led) {
        uint32_t hps_led_value = alt_read_word(ALT_GPIO1_SWPORTA_DR_ADDR);
        hps_led_value >>= HPS_LED_PORT_BIT;
```

```
        hps_led_value = !hps_led_value;
        hps_led_value <<= HPS_LED_PORT_BIT;
        assert(ALT_E_SUCCESS == alt_gpio_port_data_write(HPS_LED_PORT, HPS_LED_MASK, hps_led_value));
    }
}
```

Figure 12-12. Programming the HPS GPIO Peripheral

12.2.4.3 Launching the Bare-metal Code in the Debugger

19. Once you have finished writing all the application's code, right-click on the "`DE1_SoC_demo_hps_baremetal`" project, and select "`Build Project`".
20. Switch to the *DS-5 Debug* perspective, as shown in Figure 12-13.

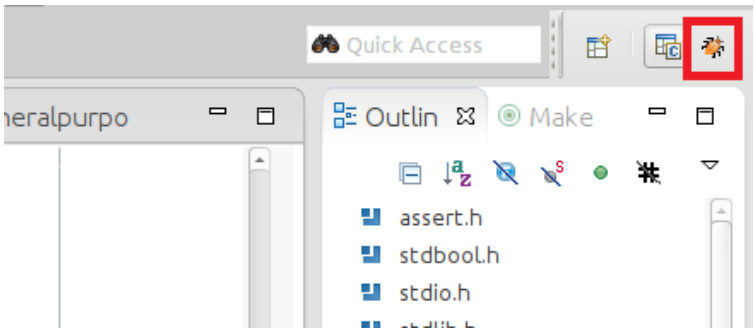


Figure 12-13. Switching to the DS-5 Debug Perspective

21. In the "`Debug Control`" view, click on the "`DE1_SoC_demo_hps_baremetal`" entry, then click on the "`Connect to Target`" button, as shown on Figure 12-14. Our debug script will load and execute the preloader, then it will load and wait at our application's "`main()`" function.

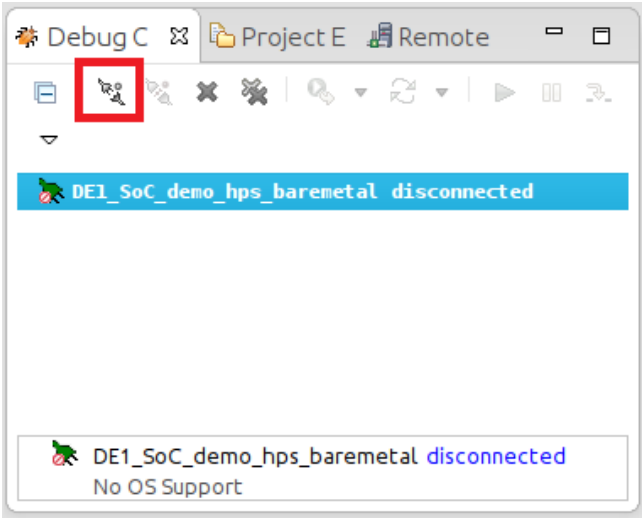


Figure 12-14. Debug Control View

22. You can the use the buttons in the "`Debug Control`" view to control the application's execution.

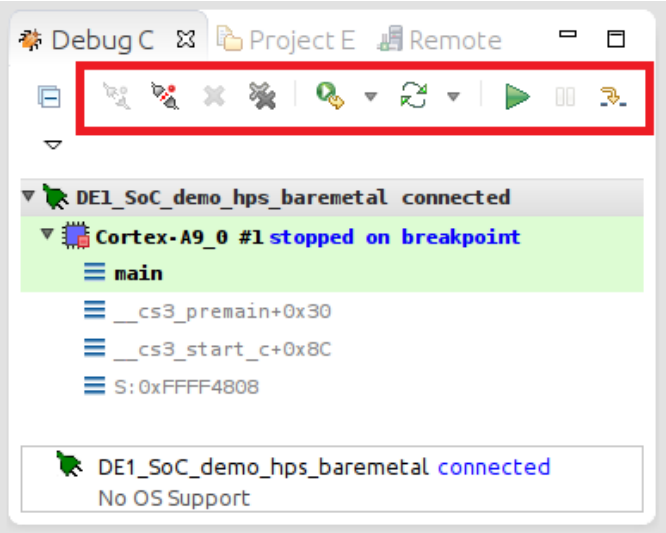


Figure 12-15. DS-5 Debugger Controls

12.2.4.4 DS-5 Bare-metal Debugger Tour

12.2.4.4.1 "Registers" View

DS-5's greatest feature is its "`Registers`" view.

Recall that we provided the debugger with a ***PERIPHERAL DESCRIPTION FILE*** in 12.2.3. This file allows the debugger's "`Registers`" view to display information about all the HPS' internal and FPGA peripherals, as shown in Figure 12-16.

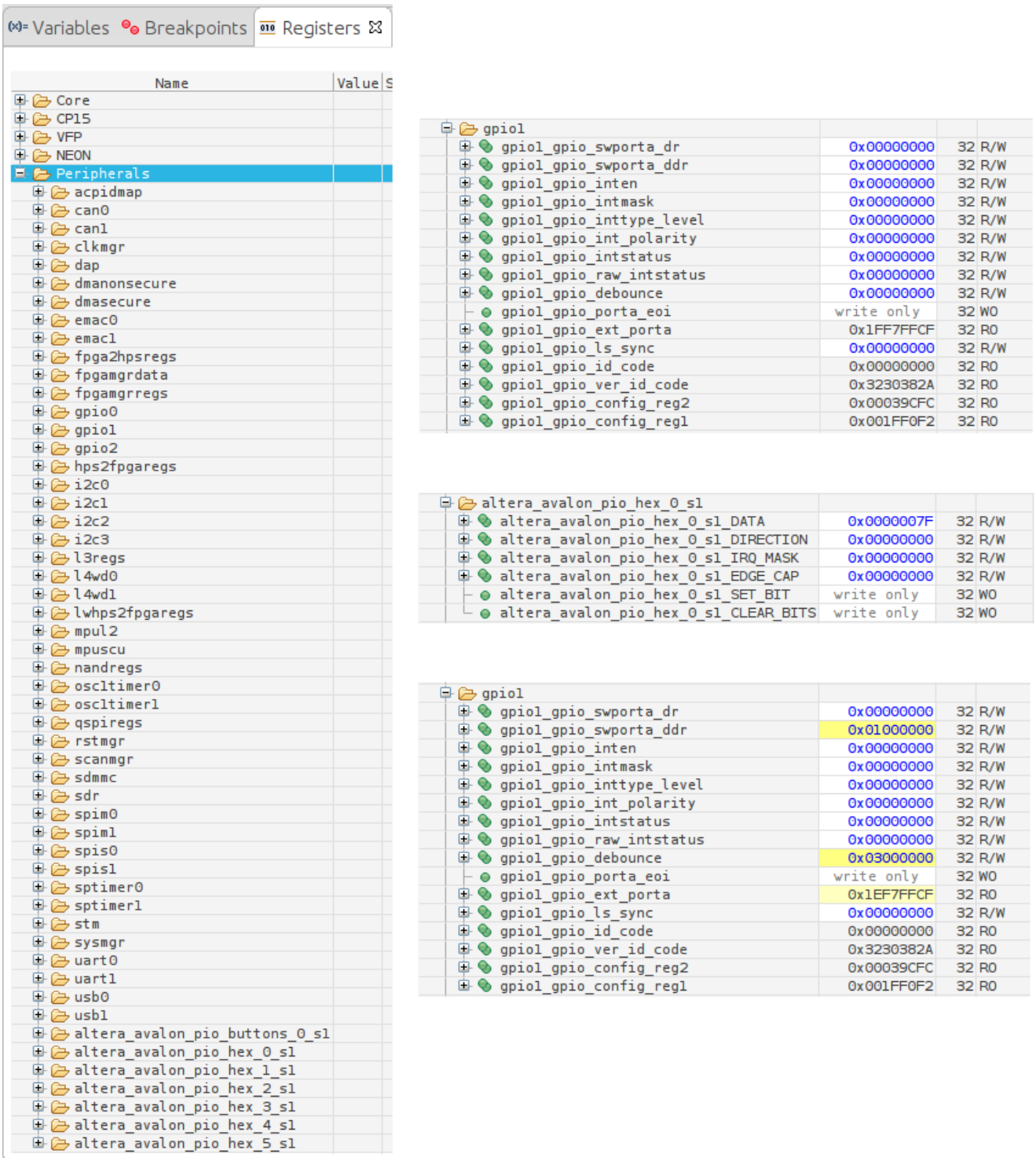


Figure 12-16. DS-5 Debugger Registers View

You can ***MODIFY*** any value in this view, and they will automatically be applied to the corresponding peripheral. For example, you can manually switch on one of the 7-segment displays, or manually trigger a button press of HPS_KEY_N (assuming you write the correct bit in the correct place).

The view also highlights the values that changed when stepping through the code while debugging, which helps you track down invalid peripheral writes, side-effects, …

However, there is one downside with the "Registers" view. With so many details in this view, one would normally start browsing through each peripheral's registers (much easier than reading the Cyclone V manual, isn't it?).

The problem occurs when you expand a peripheral that *has not been enabled* in the preloader, or that has *side-effects* when some of its registers are accessed.

Indeed, *DS-5* will try to access an invalid address, and it will crash the debugging session, therefore leaving the software on the board in an unrecoverable state. You will have to ***SWITCH OFF THE BOARD*** and reprogram it to relaunch the application. Don't forget to ***REPROGRAM THE FPGA FABRIC*** with your design as well.

12.2.4.4.2 App Console

Data sent to standard output is shown in the "`App Console`" view. Figure 12-17 shows the result of a "`printf()`" call in our demo code shown in Figure 12-8.

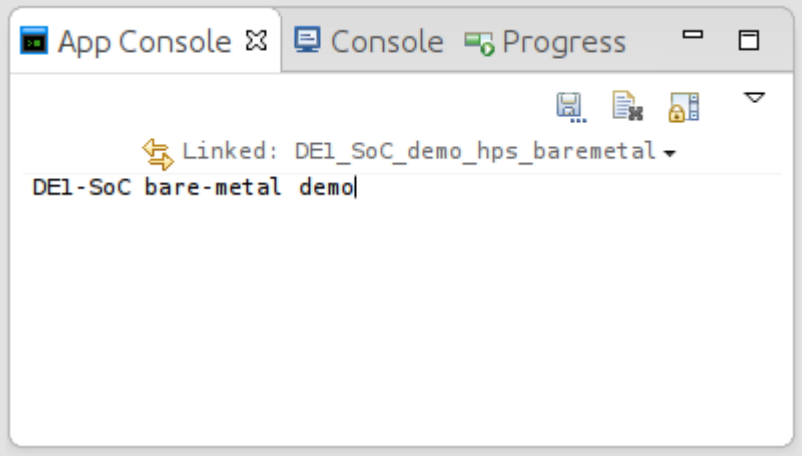


Figure 12-17. DS-5 App Console View

13 USING THE CYCLONE V – HPS – ARM – LINUX

1. Launch the *ARM DS-5* IDE. Type "`eclipse`" in a *SoC Embedded Command Shell* and press enter.

BE SURE YOU LAUNCH ARM DS-5 WITH ADMIN RIGHTS, OTHERWISE YOU WON'T BE ABLE TO ACCESS CERTAIN PERIPHERALS ON YOUR MACHINE IN ORDER TO CONNECT TO THE DE1-SOC.

13.1.1 Setting Up a New C Project

2. Create a new C project by going to "`File > New > C Project`".
    a. Use "`DE1_SoC_demo_hps_linux`" as the project name.
    b. Disable the "`Use default location`" checkbox.
    c. Set "`DE1_SoC_demo/sw/hps/application/DE1_SoC_demo_hps_linux`" as the target location for the project.
    d. We want to create a single output executable for our project, so choose "`Executable > Empty Project`" as the project type.
    e. Choose "`GCC 4.x [arm-linux-gnueabihf] (DS-5 built-in)`" as the Toolchain.
    f. You should have something similar to Figure 13-1. Then, press the "`Finish`" button to create the project.

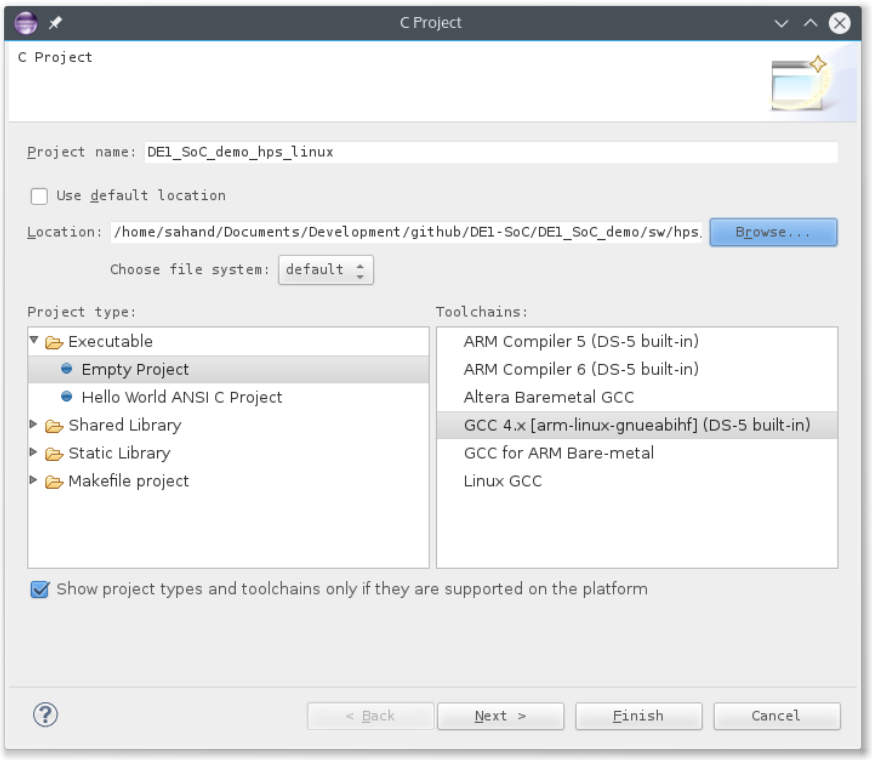


Figure 13-1. New C Project Dialog

3. When programming the HPS, we will need access to a few standard header and linker files provided by Altera. We need to add these files to the *ARM DS-5* project.
    a. Right-click on the "`DE1_SoC_demo_hps_linux`" project, and go to "`Properties`".
    b. Under "`C/C++ Build > Settings > GCC C Compiler > Includes`", add "`<altera_install_directory>/<version>/embedded/ip/altera/hps/altera_hps/hwlib/include`" to the "`Include paths (-I)`" list.
    c. Click on the "`Apply`" button, then on the "`Ok`" button to close the project properties dialog.
4. In order to unlock a few settings later in this tutorial, we will create a C file that simply contains an empty "`main()`" function for the moment.

a. Right-click on the "DE1_SoC_demo_linux" project, and go to "New > Source File". Use "hps_linux.c" as the file name, and click on the "Finish" button to create the new source file.
b. Right-click on the "DE1_SoC_demo_linux" project, and go to "New > Header File". Use "hps_linux.h" as the file name, and click on the "Finish" button to create the new header file.
c. Fill "hps_linux.c" with the code shown in Figure 13-2.

```
int main(void) {
    return 0;
}
```

Figure 13-2. hps_linux.c with an empty main() function.

d. Right-click on the "DE1_SoC_demo_linux" project and select "Build Project".

13.1.2 Creating a Remote Debug Connection to the Linux Distribution

13.1.2.1 Find the Linux Distribution's IP Address

Later in this tutorial, we will need to know the IP address assigned to the DE1-SoC so *ARM DS-5* can automatically use an SSH connection to transfer Linux binaries and launch gdb debug sessions for us. In this step, we will use a serial terminal to manually connect to the Linux distribution running on the board and find out its IP address.

5. We will use the built-in serial terminal available in *ARM DS-5*. Go to "Window > Show View > Other… > Terminal > Terminal" to open *ARM DS-5*'s the built-in serial terminal. You should see the terminal shown in Figure 13-3.

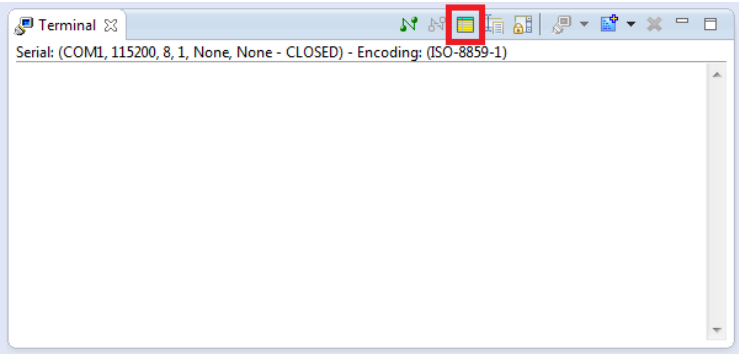


Figure 13-3. ARM DS-5 Serial Terminal

6. Modify the serial terminal's settings to match those shown in Figure 13-4, then press "OK" to start the connection.

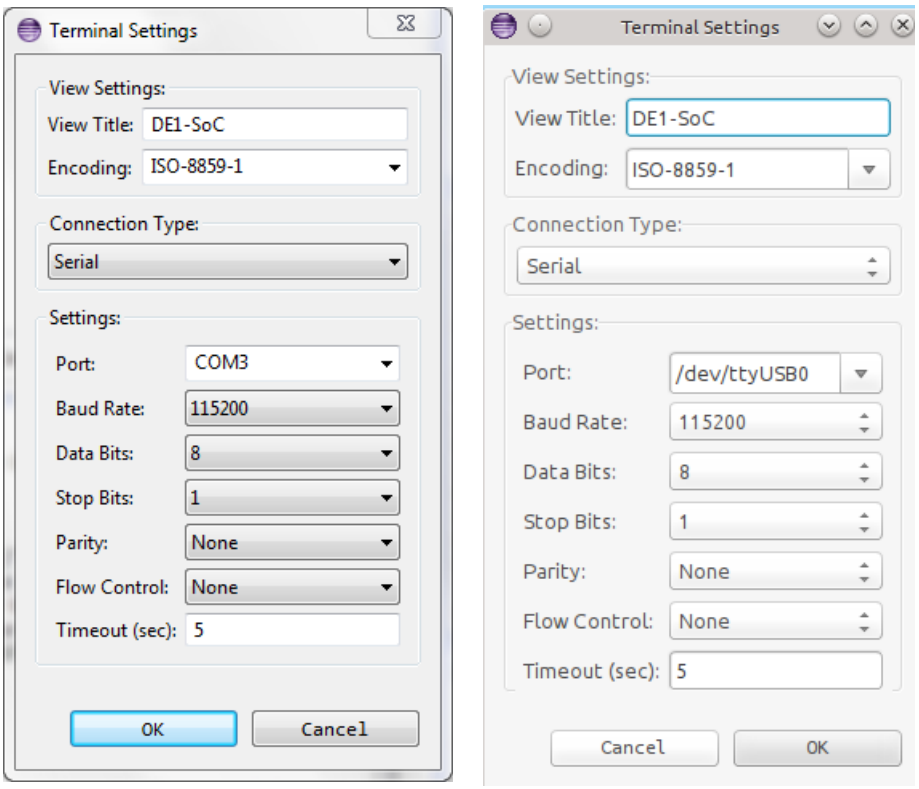


Figure 13-4. ARM DS-5 Serial Terminal Settings (Windows settings on the left, Linux settings on the right)

7. You should see the Linux login prompt. Log is as user "`root`", and use "`terasic`" as password if asked. You should see something similar as Figure 13-5.

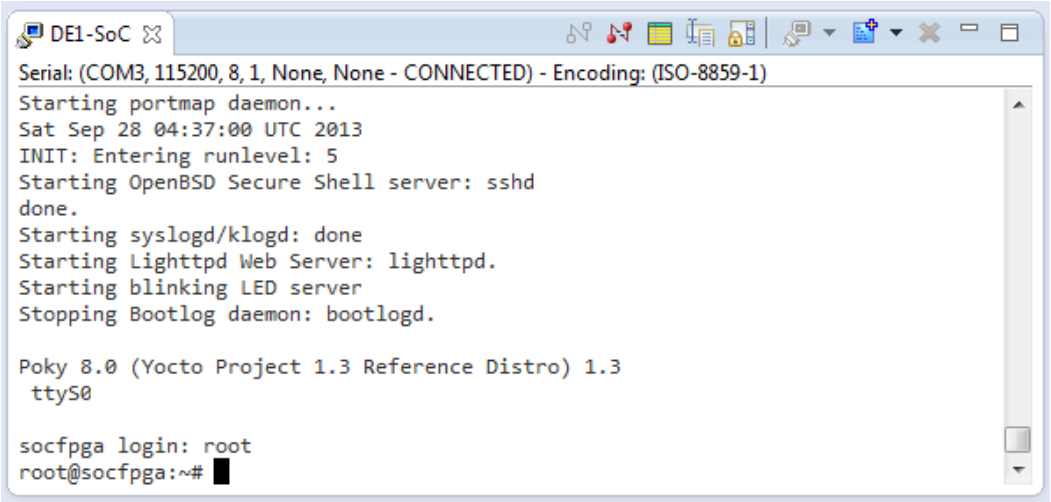


Figure 13-5. ARM DS-5 Serial Terminal Linux Prompt

8. Type "`ifconfig eth0 | grep inet`" to obtain the IP address attributed to the device. You should get something similar to Figure 13-6. If you don't see an IP address listed, then run "`udhcpc`" to try to automatically obtain one.

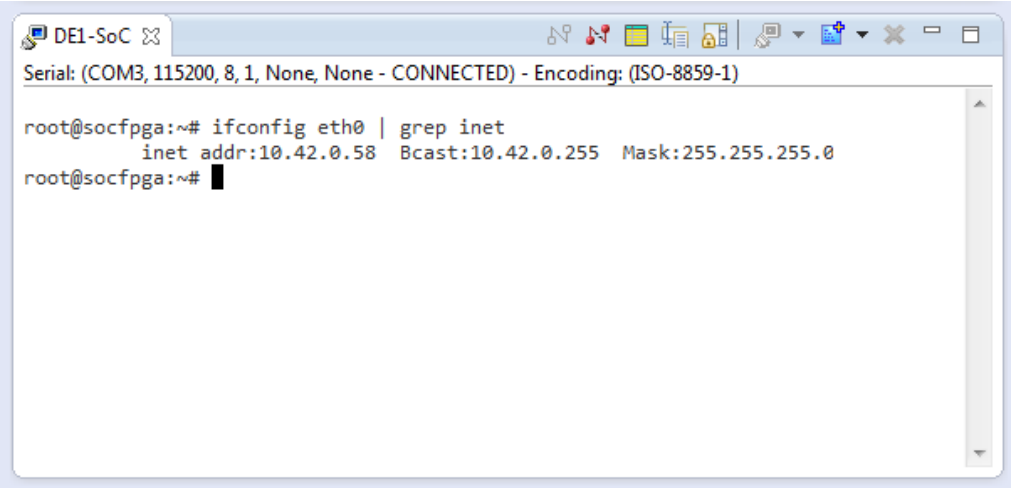


Figure 13-6. Obtaining the DE1-SoC's IP Address through ARM DS-5's Serial Terminal

13.1.2.2 Create an SSH Remote Connection

9. Go to "`File > New > Other… > Remote System Explorer > Connection`".
10. Choose to create an "`SSH Only`" connection.
11. Enter the IP address you found in 13.1.2.1 as the "`Host name`".
12. Enter "`DE1-SoC`" as the "`Connection name`". You should have something similar to Figure 13-7.
13. Click on "`Finish`" to create the connection.

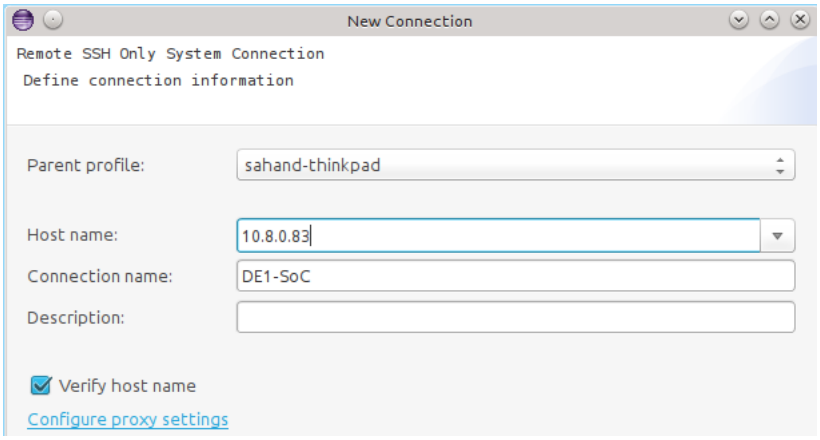


Figure 13-7. New SSH Only Connection

14. You should be able to see the remote system in *ARM DS-5*'s "`Remote Systems`" view, as shown in Figure 13-8.

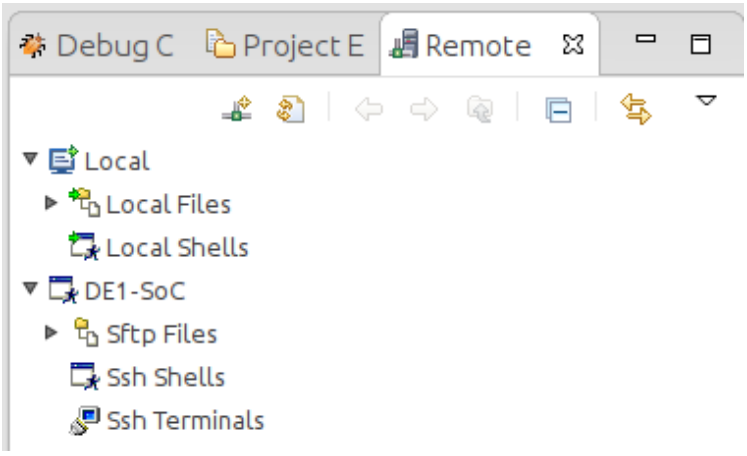


Figure 13-8. New SSH Connection In "Remote Systems" View

13.1.2.3 Setting Up the Debug Configuration

15. Right-click on the "DE1_SoC_demo_linux" project, and go to "Debug As > Debug Configurations…".
16. Choose to create a new debugger configuration by right-clicking on "DS-5 Debugger" on the left and selecting "New". Use "DE1_SoC_demo_hps_linux" as the name of the new debug configuration.
17. Under the "Connection" tab:
    a. Use "Altera > Cyclone V SoC (Dual Core) > Linux Application Debug > Download and debug application" as the target platform.
    b. Set the "RSE connection" to "DE1-SoC". This is the remote system connection we created earlier. You should have something similar to Figure 13-9.

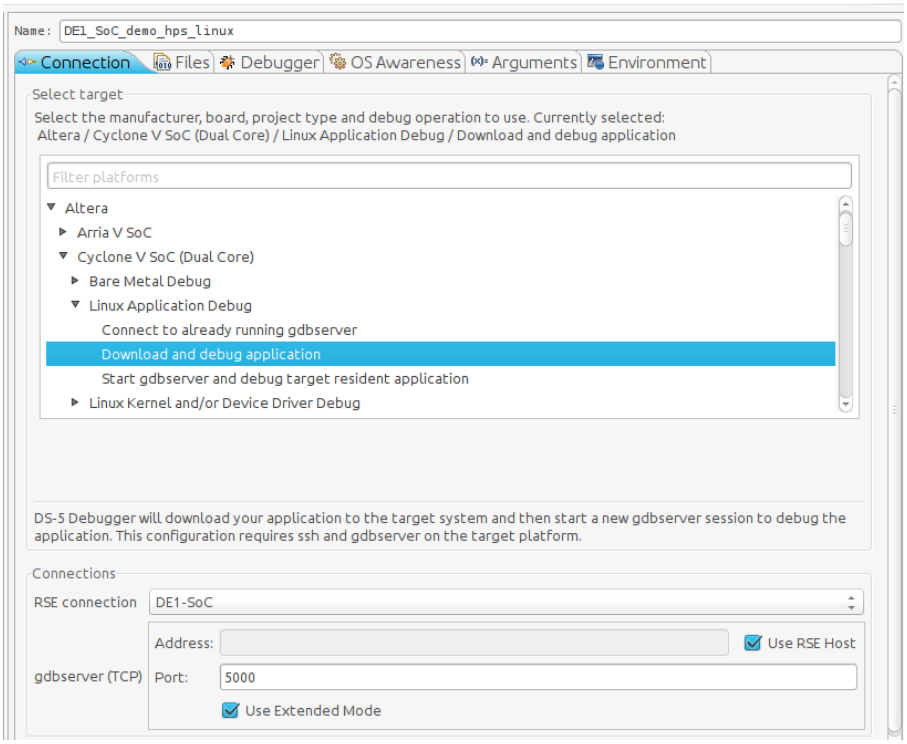


Figure 13-9. Debug Configuraton "Connection" Tab

18. Under the "Files" tab:
    a. Set "Application on host to download" to the built binary of our project. Use the "Workspace" button to choose the binary. You should have something similar to "${workspace_loc:/DE1_SoC_demo_hps_linux/Debug/DE1_SoC_demo_hps_linux}".
    b. Set the "Target download directory" to "/root".
    c. Set the "Target working directory" to "/root". You should have something similar to Figure 13-10.

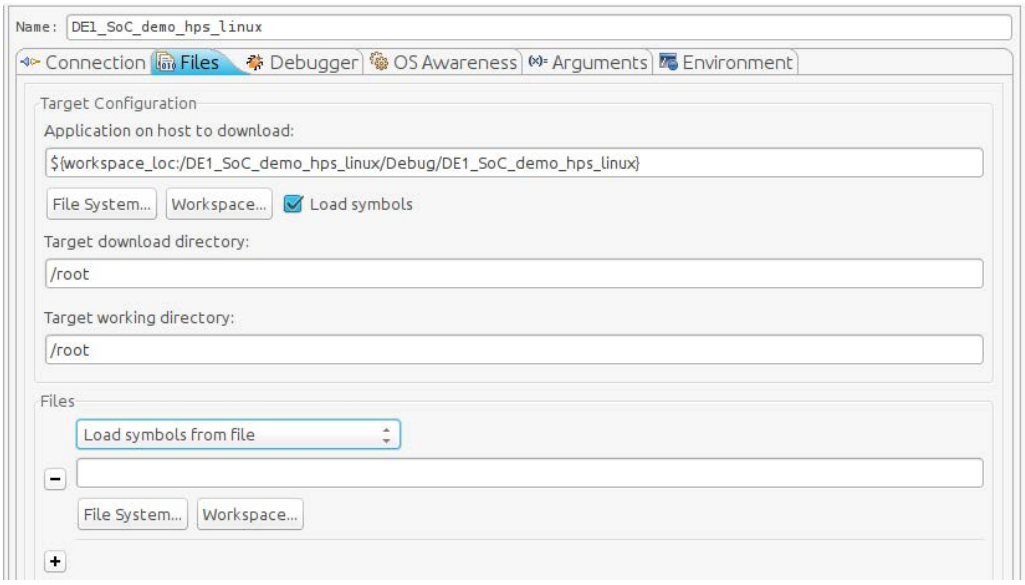


Figure 13-10. Debug Configuration "Files" Tab

19. Under the "Debugger" tab, make sure that "Debug from symbol" is selected and that "main" is the name of the symbol, as shown in Figure 13-11.
20. Click on the "Apply" button, then on the "Close" button to save the debug configuration.

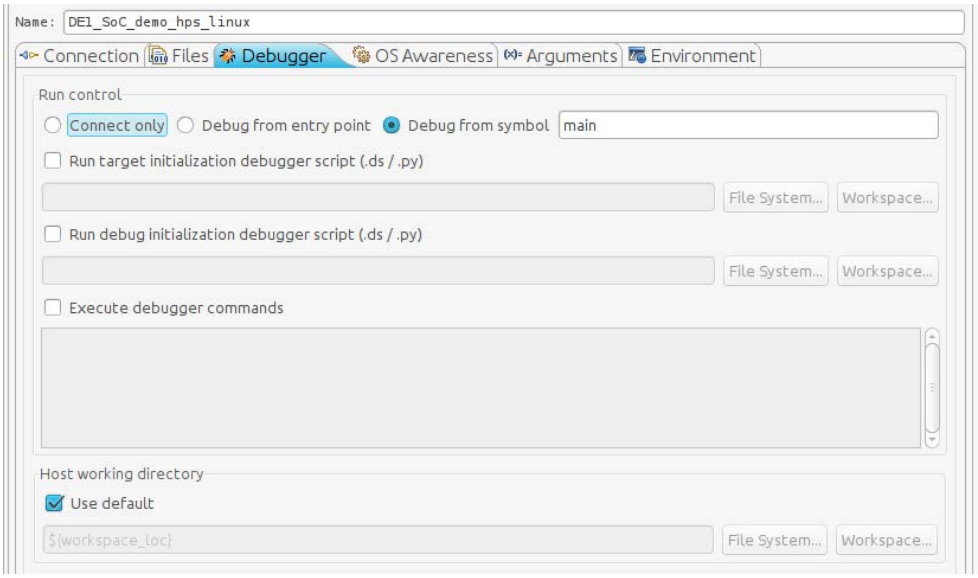


Figure 13-11. Debug Configuration "Debugger" Tab

13.1.3 Linux Programming

The interrupt-driven nature of operating systems requires that error-prone processes be unable to harm the correct operation of the computer. Modern processors provide a hardware solution to this issue by means of a ***DUAL-MODE*** operating state. CPUs define two *modes* which operating systems can then use to implement protection mechanisms among processes they are handling.

The Linux operating system calls these modes ***USER MODE*** and ***KERNEL MODE***. Processors remain in user mode when executing *harmless* code in user applications, whereas they transition to kernel mode when executing potentially *dangerous* code in the system kernel. Examples of dangerous code are handling an interrupt from a peripheral, copying data from a peripheral's registers to main memory, …

User code cannot be executed in kernel mode. When a user process needs to perform an action that is only allowed in kernel mode, it performs a system call and asks the operating system to take care of the task in its place. What this boils down to is that ***USER CODE CANNOT ACCESS THE HARDWARE DIRECTLY***, as there is too

much of a risk for the code to have an error and cause the system to crash. User code must always ask the operating system to perform dangerous operations in its place.

The main advantage of Cyclone V SoCs is the ability to have the HPS and FPGA communicate with each other easily. This is simple to accomplish in a standard bare-metal application as there are absolutely no protection mechanisms implemented. However, this is not possible while the HPS is running Linux, as user code doesn't have the right to access hardware directly.

There are 2 solutions to this problem:

- If developers are knowledgeable enough, they can write a device driver for the target peripheral they want to access in their user code, and package this in a loadable Linux kernel module. This is the correct way to access hardware in Linux, but it requires that the developer know how to write a device driver. The system's *root* user can load the kernel module, then any *standard* user code can interact with the peripheral.
- A simpler technique often used in embedded Linux environments is to leverage the virtual memory system in order to access any ***MEMORY-MAPPED*** peripherals (peripherals and operations that are only accessible through priviledged machine instructions cannot be accessed with this method). Unfortunately, this method requires code to be run with *root* privileges. However, it does not require any kernel code to be written.

Writing a Linux device driver is outside the scope of this tutorial, so we will use the memory mapping technique here.

The code for this part of the application is quite large to be inserted in this document. Therefore, we will just go over a few practical aspects of the code which are worth paying attention to. The full source can be found in `DE1_SoC_demo.zip` [3].

Recall that we cannot handle interrupts in Linux user mode. Therefore, in order to satisfy the HPS-related goals specified in 8.4, we will need to use an infinite loop and do some polling. This can be seen in our application's "`main()`" function, which is shown in Figure 13-12.

```
int main() {
    printf("DE1-SoC linux demo\n");

    open_physical_memory_device();
    mmap_peripherals();

    setup_hps_gpio();
    setup_hex_displays();

    uint32_t hex_counter = 0;
    while (true) {
        handle_hex_displays(&hex_counter);
        handle_hps_led();
        usleep(ALT_MICROSECS_IN_A_SEC / 10);
    }

    munmap_peripherals();
    close_physical_memory_device();

    return 0;
}
```

Figure 13-12. hps_linux.c main() Function

13.1.3.1 Using Altera's HWLIB - Prerequisites

We will use a ***SUBSET*** of Altera's *HWLIB* in this tutorial. In order to be able to use *HWLIB* to configure a peripheral, 2 steps need to be performed:

- You need to add the HPS peripheral's *HWLIB* ***HEADER FILE*** to your code.
- You must ***COPY*** the HPS peripheral's *HWLIB* ***SOURCE FILE*** in your *DS-5* project directory. The *HWLIB* source files can be found in directory "`<altera_install_directory>/<version>/embedded/ip/altera/hps/altera_hps/hwlib/src`", and must be copied to "`DE1_SoC_demo/sw/hps/application/DE1_SoC_demo_hps_linux`".

In the example used in this Linux programming tutorial, we use some *HWLIB* functions related to the HPS' GPIO peripheral, so you must copy "`alt_generalpurpose_io.c`" to your *DS-5* project directory.

13.1.3.2 Accessing Hardware Peripherals from User Space

13.1.3.2.1 Opening the Physical Memory File Descriptor

In Figure 7-3 we saw that the FPGA slaves and HPS peripherals are visible to the MPU unit and are therefore subject to memory-mapped IO. We need to be able to access these peripherals' addresses in order to interact with them.

Unfortunately, a process can only interact with the *virtual address space* it is assigned by the Linux kernel. Any attempt to access memory outside this region will cause the process to be terminated. Nevertheless, it is possible for a process to gain access to another virtual memory region by using the "`mmap()`" function. The `mmap()` function maps another memory region into the running process' virtual address space. Therefore, all we need to do is to `mmap()` the FPGA slaves and HPS peripherals' memory regions into our address space.

The `mmap()` function's prototype is shown in Figure 13-13. Note that it memory maps a ***FILE*** into the running process' address space, so we need to find a file that "represents" our peripherals.

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset)
```

Figure 13-13. Prototype of the mmap() Function

By design, Linux represents everything as a file, including all devices. In particular, the special "`/dev/mem`" file represents the content of the system's physical memory. This is the file we will `mmap()` in order to access the memory regions we are interested in.

Since we are memory-mapping a file, the first step is to open this file. Figure 13-14 shows how to open the `/dev/mem` file. Remember that `/dev/mem` grants access to physical memory, so a user requires elevates rights in order to open it. Therefore, don't forget to launch this code as the *root* user in order to have enough privileges.

```
// physical memory file descriptor
int fd_dev_mem = 0;

void open_physical_memory_device() {
    fd_dev_mem = open("/dev/mem", O_RDWR | O_SYNC);
    if(fd_dev_mem  == -1) {
        printf("ERROR: could not open \"/dev/mem\"...\n");
        printf("    errno = %s\n", strerror(errno));
        exit(EXIT_FAILURE);
    }
}
```

Figure 13-14. open_physical_memory_device() Function

13.1.3.2.2 Accessing HPS Peripherals

Now that we have opened the physical memory file, we can memory-map a subset of it into our process' virtual address space. Figure 13-15 shows how this is done for memory-mapping the HPS' GPIO peripheral.

Note that you must know the offset of your peripheral within the physical memory file, as well as the amount of memory you want to be memory-mapped from that offset. In our case, we will start memory-mapping from the GPIO1 peripheral's offset, and we choose to map the size of the full peripheral.

```
void   *hps_gpio     = NULL;
size_t hps_gpio_span = ALT_GPIO1_UB_ADDR - ALT_GPIO1_LB_ADDR + 1;
size_t hps_gpio_ofst = ALT_GPIO1_OFST;

void mmap_hps_peripherals() {
    hps_gpio = mmap(NULL, hps_gpio_span, PROT_READ | PROT_WRITE, MAP_SHARED, fd_dev_mem, hps_gpio_ofst);
    if (hps_gpio == MAP_FAILED) {
        printf("Error: hps_gpio mmap() failed.\n");
        printf("    errno = %s\n", strerror(errno));
        close(fd_dev_mem);
        exit(EXIT_FAILURE);
    }
}
```

Figure 13-15. mmap_hps_peripherals() Function

Finally, after having memory-mapped the HPS' GPIO peripheral, we can access any of its internal registers with the low-level functions we saw in 11.3. Figure 13-16 shows how we configure the HPS' GPIO peripheral, and Figure 13-17 shows how we can toggle HPS_LED on the DE1-SoC by using the HPS_KEY_N button.

```
void setup_hps_gpio() {
    // Initialize the HPS PIO controller:
    //     Set the direction of the HPS_LED GPIO bit to "output"
    //     Set the direction of the HPS_KEY_N GPIO bit to "input"
    void *hps_gpio_direction = ALT_GPIO_SWPORTA_DDR_ADDR(hps_gpio);
    alt_setbits_word(hps_gpio_direction, ALT_GPIO_PIN_OUTPUT << HPS_LED_PORT_BIT);
    alt_setbits_word(hps_gpio_direction, ALT_GPIO_PIN_INPUT << HPS_KEY_N_PORT_BIT);
```

Figure 13-16. setup_hps_gpio() Function

```
void handle_hps_led() {
    void *hps_gpio_data = ALT_GPIO_SWPORTA_DR_ADDR(hps_gpio);
    void *hps_gpio_port = ALT_GPIO_EXT_PORTA_ADDR(hps_gpio);

    uint32_t hps_gpio_input = alt_read_word(hps_gpio_port) & HPS_KEY_N_MASK;

    // HPS_KEY_N is active-low
    bool toggle_hps_led = (~hps_gpio_input & HPS_KEY_N_MASK);

    if (toggle_hps_led) {
        uint32_t hps_led_value = alt_read_word(hps_gpio_data);
        hps_led_value >>= HPS_LED_PORT_BIT;
        hps_led_value = !hps_led_value;
        hps_led_value <<= HPS_LED_PORT_BIT;
        alt_replbits_word(hps_gpio_data, HPS_LED_MASK, hps_led_value);
    }
}
```

Figure 13-17. handle_hps_led() Function

The key to doing memory-mapped IO in Linux is to use *HWLIB*'s ***OFFSET***-based macros with the virtual address returned by `mmap()` as the base address. Note that *HWLIB* also has macros with ***ABSOLUTE*** addresses for every device, but those can only be used in bare-metal or Linux device driver code as they directly access certain physical addresses.

In Figure 13-17 and Figure 13-17, we used three such offset-based macros to access the HPS GPIO peripheral's "`Port A Data Register`", "`Port A Data Direction Register`", and "`External Port A Register`". These macros were the following:

- `ALT_GPIO_SWPORTA_DR_ADDR(base)`
- `ALT_GPIO_SWPORTA_DDR_ADDR(base)`
- `ALT_GPIO_EXT_PORTA_ADDR(base)`

13.1.3.2.3 Accessing FPGA Peripherals

Memory-mapping FPGA peripherals is identical to the process used for HPS peripherals. However, there is one subtlety that must be taken care of. When using `mmap()` you must specify an offset within the file that is to be mapped, as well as the amount of memory to be mapped. The `mmap()` manual page states that the offset provided ***MUST BE A MULTIPLE OF THE SYSTEM'S PAGE SIZE***, which is `0x1000` bytes in our case.

If you look closely at the addresses in
Table 7-4, you will realize that this requirement always holds for the HPS' peripherals. However, this is not always true for the FPGA peripherals. For example, the design we used in this tutorial puts the FPGA buttons at address 0xFF200060 (offset 0x60 from the base address of the Lightweight HPS-to-FPGA bridge), which is not a multiple of the system's page size.

This implies that it isn't possible to memory-map the FPGA buttons alone, but we must instead use some offset which Is a multiple of the system's page size. To get around this issue, we will memory-map FPGA peripherals from the HPS peripheral to which they are connected, as we are sure that the particular HPS peripheral's base addres is a multiple of the page size.

Figure 13-18 shows how we memory-map the FPGA peripherals in our design from the Lightweight HPS-to-FPGA bridge, and Figure 13-19 shows how we can check if one of the FPGA buttons are being pressed.

```
void   *h2f_lw_axi_master     = NULL;
size_t h2f_lw_axi_master_span = ALT_LWFPGASLVS_UB_ADDR - ALT_LWFPGASLVS_LB_ADDR + 1;
size_t h2f_lw_axi_master_ofst = ALT_LWFPGASLVS_OFST;

void *fpga_buttons                          = NULL;
void *fpga_hex_displays[HEX_DISPLAY_COUNT] = {NULL, NULL, NULL, NULL, NULL, NULL};

void mmap_fpga_peripherals() {
    h2f_lw_axi_master = mmap(NULL, h2f_lw_axi_master_span, PROT_READ | PROT_WRITE, MAP_SHARED, fd_dev_mem,
                             h2f_lw_axi_master_ofst);

    if (h2f_lw_axi_master == MAP_FAILED) {
        printf("Error: h2f_lw_axi_master mmap() failed.\n");
        printf("    errno = %s\n", strerror(errno));
        close(fd_dev_mem);
        exit(EXIT_FAILURE);
    }

    fpga_buttons         = h2f_lw_axi_master + BUTTONS_0_BASE;
    fpga_hex_displays[0] = h2f_lw_axi_master + HEX_0_BASE;
    fpga_hex_displays[1] = h2f_lw_axi_master + HEX_1_BASE;
    fpga_hex_displays[2] = h2f_lw_axi_master + HEX_2_BASE;
    fpga_hex_displays[3] = h2f_lw_axi_master + HEX_3_BASE;
    fpga_hex_displays[4] = h2f_lw_axi_master + HEX_4_BASE;
    fpga_hex_displays[5] = h2f_lw_axi_master + HEX_5_BASE;
```

Figure 13-18. mmap_fpga_peripherals() Function.

```
bool is_fpga_button_pressed(uint32_t button_number) {
    // buttons are active-low
    return ((~alt_read_word(fpga_buttons)) & (1 << button_number));
```

```
}
```

Figure 13-19. is_fpga_button_pressed() Function

13.1.3.2.4 Cleaning Up Before Application Exit

Although the operating system should take care of this for you, it is always a good practice to remove any unneeded memory mappings and to close the physical memory file descriptor before your application terminates.

Figure 13-20 shows how to unmap the GPIO peripheral's memory-mapping, and Figure 13-21 shows how to close the physical memory file descriptor.

```
void munmap_hps_peripherals() {
    if (munmap(hps_gpio, hps_gpio_span) != 0) {
        printf("Error: hps_gpio munmap() failed\n");
        printf("    errno = %s\n", strerror(errno));
        close(fd_dev_mem);
        exit(EXIT_FAILURE);
    }

    hps_gpio = NULL;
}
```

Figure 13-20. munmap_hps_peripherals() Function

```
void close_physical_memory_device() {
    close(fd_dev_mem);
}
```

Figure 13-21. close_physical_memory_device() Function

13.1.3.3 Launching the Linux code in the Debugger

21. Once you have finished writing all the application's code, right-click on the "`DE1_SoC_demo_hps_linux`" project, and select "`Build Project`".
22. Switch to the *DS-5 Debug* perspective, as shown in Figure 12-13.

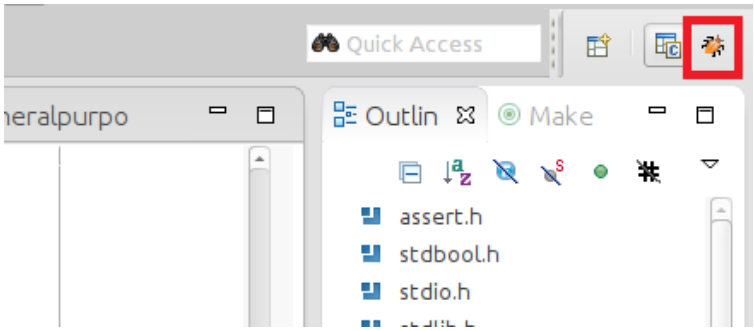


Figure 13-22. Switching to the DS-5 Debug Perspective

23. In the "`Debug Control`" view, click on the "`DE1_SoC_demo_hps_linux`" entry, then click on the "`Connect to Target`" button, as shown on Figure 13-23. The debugger will start an SSH conection to the Linux distribution running on the DE1-SoC and will automatically transfer our binary file and wait at our application's "`main()`" function.

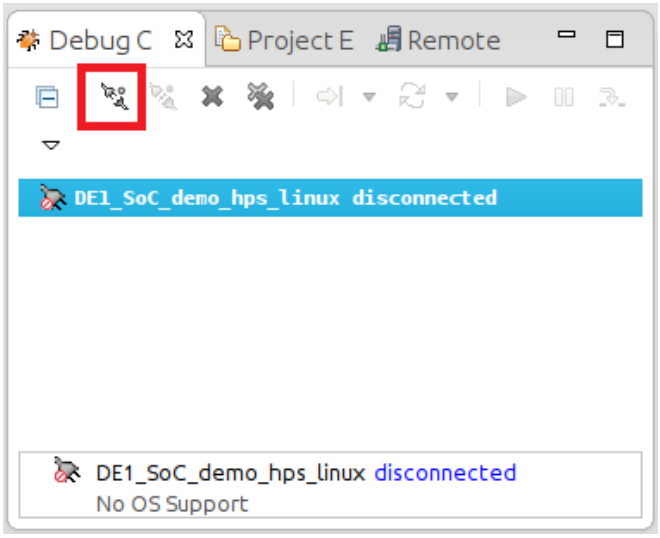


Figure 13-23. Debug Control View

24. You can the use the buttons in the "Debug Control" view to control the application's execution.

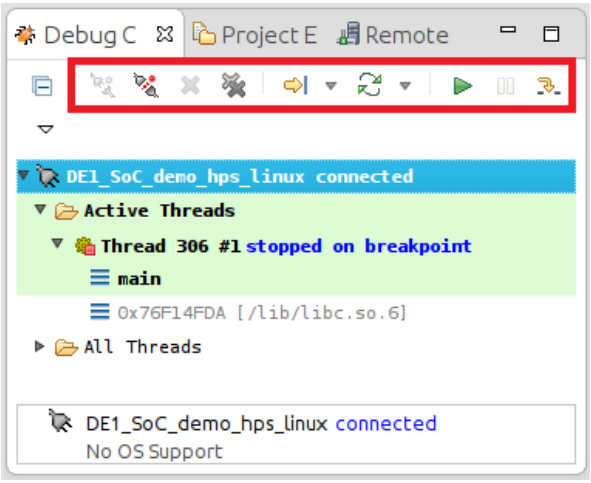


Figure 13-24. DS-5 Debugger Controls

13.1.3.4 App Console

Data sent to standard output is shown in the "App Console" view. Figure 13-25 shows the result of a "printf()" call in our demo code shown in Figure 13-12.

Figure 13-25. DS-5 App Console View

13.1.3.5 DS-5 Linux Debugger Restrictions

In 12.2.4.4.1, we saw that the *DS-5* ***BARE-METAL*** debugger had a **"**`Registers`**"** view which could show the registers of all HPS and FPGA peripherals. This was a very handy tool, as it made it easy to verify if registers were accessed and updated correctly.

Unfortunately, when it comes to debugging ***LINUX*** binaries, the *DS-5* debugger is subject to the same constraints our Linux applications are. Namely, it cannot directly access physical hardware addresses directly. As such, there is no **"**`Registers`**"** view when debugging Linux applications, and you must resort to manually memory-mapping and verifying peripheral accesses yourself.

14TODO

- Explain MSEL when reprogramming the FPGA from the HPS.
- Talk about what the JTAG to Avalon masters are.
- Find out how to automatically program the FPGA when writing a bare-metal HPS application.

15 APPENDIX

15.1 DE1-SOC TOP-LEVEL VHDL ENTITY

The DE1-SoC has a lot of pins, which makes it tedious to start an FPGA design. It is recommended to use the following ***ENTITY*** for your ***TOP-LEVEL VHDL FILE***, as it contains all the board's FPGA and HPS pins. The file can be found at `DE1_SoC_top_level.vhd` [4].

```
-- ##################################################################################
-- DE1_SoC_top_level.vhd
--
-- BOARD         : DE1-SoC from Terasic
-- Author        : Sahand Kashani-Akhavan from Terasic documentation
-- Revision      : 1.3
-- Creation date : 04/02/2015
--
-- Syntax Rule : GROUP_NAME_N[bit]
--
-- GROUP  : specify a particular interface (ex: SDR_)
-- NAME   : signal name (ex: CONFIG, D, ...)
-- bit    : signal index
-- _N     : to specify an active-low signal
-- ##################################################################################

library ieee;
use ieee.std_logic_1164.all;

entity DE1_SoC_top_level is
    port(
        -- ADC
        ADC_CS_n         : out   std_logic;
        ADC_DIN          : out   std_logic;
        ADC_DOUT         : in    std_logic;
        ADC_SCLK         : out   std_logic;

        -- Audio
        AUD_ADCDAT       : in    std_logic;
        AUD_ADCLRCK      : inout std_logic;
        AUD_BCLK         : inout std_logic;
        AUD_DACDAT       : out   std_logic;
        AUD_DACLRCK      : inout std_logic;
        AUD_XCK          : out   std_logic;

        -- CLOCK
        CLOCK_50         : in    std_logic;
        CLOCK2_50        : in    std_logic;
        CLOCK3_50        : in    std_logic;
        CLOCK4_50        : in    std_logic;

        -- SDRAM
        DRAM_ADDR        : out   std_logic_vector(12 downto 0);
        DRAM_BA          : out   std_logic_vector(1 downto 0);
        DRAM_CAS_N       : out   std_logic;
        DRAM_CKE         : out   std_logic;
        DRAM_CLK         : out   std_logic;
        DRAM_CS_N        : out   std_logic;
        DRAM_DQ          : inout std_logic_vector(15 downto 0);
        DRAM_LDQM        : out   std_logic;
```

```
        DRAM_RAS_N       : out   std_logic;
        DRAM_UDQM        : out   std_logic;
        DRAM_WE_N        : out   std_logic;

        -- I2C for Audio and Video-In
        FPGA_I2C_SCLK    : out   std_logic;
        FPGA_I2C_SDAT    : inout std_logic;

        -- SEG7
        HEX0_N           : out   std_logic_vector(6 downto 0);
        HEX1_N           : out   std_logic_vector(6 downto 0);
        HEX2_N           : out   std_logic_vector(6 downto 0);
        HEX3_N           : out   std_logic_vector(6 downto 0);
        HEX4_N           : out   std_logic_vector(6 downto 0);
        HEX5_N           : out   std_logic_vector(6 downto 0);

        -- IR
        IRDA_RXD         : in    std_logic;
        IRDA_TXD         : out   std_logic;

        -- KEY_N
        KEY_N            : in    std_logic_vector(3 downto 0);

        -- LED
        LEDR             : out   std_logic_vector(9 downto 0);

        -- PS2
        PS2_CLK          : inout std_logic;
        PS2_CLK2         : inout std_logic;
        PS2_DAT          : inout std_logic;
        PS2_DAT2         : inout std_logic;

        -- SW
        SW               : in    std_logic_vector(9 downto 0);

        -- Video-In
        TD_CLK27         : inout std_logic;
        TD_DATA          : out   std_logic_vector(7 downto 0);
        TD_HS            : out   std_logic;
        TD_RESET_N       : out   std_logic;
        TD_VS            : out   std_logic;

        -- VGA
        VGA_B            : out   std_logic_vector(7 downto 0);
        VGA_BLANK_N      : out   std_logic;
        VGA_CLK          : out   std_logic;
        VGA_G            : out   std_logic_vector(7 downto 0);
        VGA_HS           : out   std_logic;
        VGA_R            : out   std_logic_vector(7 downto 0);
        VGA_SYNC_N       : out   std_logic;
        VGA_VS           : out   std_logic;

        -- GPIO_0
        GPIO_0           : inout std_logic_vector(35 downto 0);

        -- GPIO_1
        GPIO_1           : inout std_logic_vector(35 downto 0);

        -- HPS
```

```
        HPS_CONV_USB_N   : inout std_logic;
        HPS_DDR3_ADDR    : out   std_logic_vector(14 downto 0);
        HPS_DDR3_BA      : out   std_logic_vector(2 downto 0);
        HPS_DDR3_CAS_N   : out   std_logic;
        HPS_DDR3_CK_N    : out   std_logic;
        HPS_DDR3_CK_P    : out   std_logic;
        HPS_DDR3_CKE     : out   std_logic;
        HPS_DDR3_CS_N    : out   std_logic;
        HPS_DDR3_DM      : out   std_logic_vector(3 downto 0);
        HPS_DDR3_DQ      : inout std_logic_vector(31 downto 0);
        HPS_DDR3_DQS_N   : inout std_logic_vector(3 downto 0);
        HPS_DDR3_DQS_P   : inout std_logic_vector(3 downto 0);
        HPS_DDR3_ODT     : out   std_logic;
        HPS_DDR3_RAS_N   : out   std_logic;
        HPS_DDR3_RESET_N : out   std_logic;
        HPS_DDR3_RZQ     : in    std_logic;
        HPS_DDR3_WE_N    : out   std_logic;
        HPS_ENET_GTX_CLK : out   std_logic;
        HPS_ENET_INT_N   : inout std_logic;
        HPS_ENET_MDC     : out   std_logic;
        HPS_ENET_MDIO    : inout std_logic;
        HPS_ENET_RX_CLK  : in    std_logic;
        HPS_ENET_RX_DATA : in    std_logic_vector(3 downto 0);
        HPS_ENET_RX_DV   : in    std_logic;
        HPS_ENET_TX_DATA : out   std_logic_vector(3 downto 0);
        HPS_ENET_TX_EN   : out   std_logic;
        HPS_FLASH_DATA   : inout std_logic_vector(3 downto 0);
        HPS_FLASH_DCLK   : out   std_logic;
        HPS_FLASH_NCSO   : out   std_logic;
        HPS_GPIO         : inout std_logic_vector(1 downto 0);
        HPS_GSENSOR_INT  : inout std_logic;
        HPS_I2C_CONTROL  : inout std_logic;
        HPS_I2C1_SCLK    : inout std_logic;
        HPS_I2C1_SDAT    : inout std_logic;
        HPS_I2C2_SCLK    : inout std_logic;
        HPS_I2C2_SDAT    : inout std_logic;
        HPS_KEY_N        : inout std_logic;
        HPS_LED          : inout std_logic;
        HPS_SD_CLK       : out   std_logic;
        HPS_SD_CMD       : inout std_logic;
        HPS_SD_DATA      : inout std_logic_vector(3 downto 0);
        HPS_SPIM_CLK     : out   std_logic;
        HPS_SPIM_MISO    : in    std_logic;
        HPS_SPIM_MOSI    : out   std_logic;
        HPS_SPIM_SS      : inout std_logic;
        HPS_UART_RX      : in    std_logic;
        HPS_UART_TX      : out   std_logic;
        HPS_USB_CLKOUT   : in    std_logic;
        HPS_USB_DATA     : inout std_logic_vector(7 downto 0);
        HPS_USB_DIR      : in    std_logic;
        HPS_USB_NXT      : in    std_logic;
        HPS_USB_STP      : out   std_logic
    );
end entity DE1_SoC_top_level;

architecture rtl of DE1_SoC_top_level is
begin
end;
```

Figure 15-1. DE1-SoC Top-level VHDL Entity

15.2 DE1-SoC Pin Assignment TCL script

After having defined a top-level module, it is necessary to map your design's pins to the ones available on the DE1-SoC. The following ***TCL SCRIPT*** can be executed in *Quartus Prime* to specify the board's device ID and all its ***PIN ASSIGNMENTS.*** The file can be found at `pin_assignment_DE1_SoC.tcl` [5].

```
#============================================================================
# pin_assignment_DE1_SoC.tcl
#
# BOARD         : DE1-SoC from Terasic
# Author        : Sahand Kashani-Akhavan from Terasic documentation
# Revision      : 1.3
# Creation date : 04/02/2015
#
# Syntax Rule : GROUP_NAME_N[bit]
#
# GROUP  : specify a particular interface (ex: SDR_)
# NAME   : signal name (ex: CONFIG, D, ...)
# bit    : signal index
# _N     : to specify an active-low signal
#
# You can run this script from Quartus by observing the following steps:
# 1. Place this TCL script in your project directory
# 2. Open your project in Quartus
# 3. Go to the View > Utility Windows -> Tcl Console
# 4. In the Tcl Console type:
#        source pin_assignment_DE1_SoC.tcl
#
# 5. The script will assign pins and return an "assignment made" message.
#============================================================================

set_global_assignment -name FAMILY "Cyclone V"
set_global_assignment -name DEVICE 5CSEMA5F31C6
set_global_assignment -name DEVICE_FILTER_PACKAGE FBGA
set_global_assignment -name DEVICE_FILTER_PIN_COUNT 896
set_global_assignment -name DEVICE_FILTER_SPEED_GRADE 6

#============================================================
# ADC
#============================================================
set_location_assignment PIN_AJ4 -to ADC_CS_N
set_location_assignment PIN_AK4 -to ADC_DIN
set_location_assignment PIN_AK3 -to ADC_DOUT
set_location_assignment PIN_AK2 -to ADC_SCLK

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to ADC_CS_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to ADC_DIN
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to ADC_DOUT
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to ADC_SCLK

#============================================================
# Audio
#============================================================
set_location_assignment PIN_K7 -to AUD_ADCDAT
set_location_assignment PIN_K8 -to AUD_ADCLRCK
set_location_assignment PIN_H7 -to AUD_BCLK
set_location_assignment PIN_J7 -to AUD_DACDAT
set_location_assignment PIN_H8 -to AUD_DACLRCK
set_location_assignment PIN_G7 -to AUD_XCK
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to AUD_ADCDAT
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to AUD_ADCLRCK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to AUD_BCLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to AUD_DACDAT
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to AUD_DACLRCK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to AUD_XCK

#============================================================
# CLOCK
#============================================================
set_location_assignment PIN_AF14 -to CLOCK_50
set_location_assignment PIN_AA16 -to CLOCK2_50
set_location_assignment PIN_Y26  -to CLOCK3_50
set_location_assignment PIN_K14  -to CLOCK4_50

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK_50
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK2_50
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK3_50
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK4_50

#============================================================
# SDRAM
#============================================================
set_location_assignment PIN_AK14 -to DRAM_ADDR[0]
set_location_assignment PIN_AK14 -to DRAM_ADDR_0
set_location_assignment PIN_AH14 -to DRAM_ADDR[1]
set_location_assignment PIN_AH14 -to DRAM_ADDR_1
set_location_assignment PIN_AG15 -to DRAM_ADDR[2]
set_location_assignment PIN_AG15 -to DRAM_ADDR_2
set_location_assignment PIN_AE14 -to DRAM_ADDR[3]
set_location_assignment PIN_AE14 -to DRAM_ADDR_3
set_location_assignment PIN_AB15 -to DRAM_ADDR[4]
set_location_assignment PIN_AB15 -to DRAM_ADDR_4
set_location_assignment PIN_AC14 -to DRAM_ADDR[5]
set_location_assignment PIN_AC14 -to DRAM_ADDR_5
set_location_assignment PIN_AD14 -to DRAM_ADDR[6]
set_location_assignment PIN_AD14 -to DRAM_ADDR_6
set_location_assignment PIN_AF15 -to DRAM_ADDR[7]
set_location_assignment PIN_AF15 -to DRAM_ADDR_7
set_location_assignment PIN_AH15 -to DRAM_ADDR[8]
set_location_assignment PIN_AH15 -to DRAM_ADDR_8
set_location_assignment PIN_AG13 -to DRAM_ADDR[9]
set_location_assignment PIN_AG13 -to DRAM_ADDR_9
set_location_assignment PIN_AG12 -to DRAM_ADDR[10]
set_location_assignment PIN_AG12 -to DRAM_ADDR_10
set_location_assignment PIN_AH13 -to DRAM_ADDR[11]
set_location_assignment PIN_AH13 -to DRAM_ADDR_11
set_location_assignment PIN_AJ14 -to DRAM_ADDR[12]
set_location_assignment PIN_AJ14 -to DRAM_ADDR_12
set_location_assignment PIN_AF13 -to DRAM_BA[0]
set_location_assignment PIN_AF13 -to DRAM_BA_0
set_location_assignment PIN_AJ12 -to DRAM_BA[1]
set_location_assignment PIN_AJ12 -to DRAM_BA_1
set_location_assignment PIN_AF11 -to DRAM_CAS_N
set_location_assignment PIN_AK13 -to DRAM_CKE
set_location_assignment PIN_AG11 -to DRAM_CS_N
set_location_assignment PIN_AH12 -to DRAM_CLK
set_location_assignment PIN_AK6  -to DRAM_DQ[0]
```

```
set_location_assignment PIN_AK6  -to DRAM_DQ_0
set_location_assignment PIN_AJ7  -to DRAM_DQ[1]
set_location_assignment PIN_AJ7  -to DRAM_DQ_1
set_location_assignment PIN_AK7  -to DRAM_DQ[2]
set_location_assignment PIN_AK7  -to DRAM_DQ_2
set_location_assignment PIN_AK8  -to DRAM_DQ[3]
set_location_assignment PIN_AK8  -to DRAM_DQ_3
set_location_assignment PIN_AK9  -to DRAM_DQ[4]
set_location_assignment PIN_AK9  -to DRAM_DQ_4
set_location_assignment PIN_AG10 -to DRAM_DQ[5]
set_location_assignment PIN_AG10 -to DRAM_DQ_5
set_location_assignment PIN_AK11 -to DRAM_DQ[6]
set_location_assignment PIN_AK11 -to DRAM_DQ_6
set_location_assignment PIN_AJ11 -to DRAM_DQ[7]
set_location_assignment PIN_AJ11 -to DRAM_DQ_7
set_location_assignment PIN_AH10 -to DRAM_DQ[8]
set_location_assignment PIN_AH10 -to DRAM_DQ_8
set_location_assignment PIN_AJ10 -to DRAM_DQ[9]
set_location_assignment PIN_AJ10 -to DRAM_DQ_9
set_location_assignment PIN_AJ9  -to DRAM_DQ[10]
set_location_assignment PIN_AJ9  -to DRAM_DQ_10
set_location_assignment PIN_AH9  -to DRAM_DQ[11]
set_location_assignment PIN_AH9  -to DRAM_DQ_11
set_location_assignment PIN_AH8  -to DRAM_DQ[12]
set_location_assignment PIN_AH8  -to DRAM_DQ_12
set_location_assignment PIN_AH7  -to DRAM_DQ[13]
set_location_assignment PIN_AH7  -to DRAM_DQ_13
set_location_assignment PIN_AJ6  -to DRAM_DQ[14]
set_location_assignment PIN_AJ6  -to DRAM_DQ_14
set_location_assignment PIN_AJ5  -to DRAM_DQ[15]
set_location_assignment PIN_AJ5  -to DRAM_DQ_15
set_location_assignment PIN_AB13 -to DRAM_LDQM
set_location_assignment PIN_AE13 -to DRAM_RAS_N
set_location_assignment PIN_AK12 -to DRAM_UDQM
set_location_assignment PIN_AA13 -to DRAM_WE_N

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_7
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_8
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_9
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[10]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_10
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[11]
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_11
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR[12]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_ADDR_12
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_BA[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_BA_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_BA[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_BA_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_CAS_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_CKE
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_CS_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_CLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_7
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_8
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_9
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[10]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_10
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[11]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_11
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[12]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_12
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[13]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_13
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[14]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_14
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ[15]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_DQ_15
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_LDQM
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_RAS_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_UDQM
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to DRAM_WE_N

#============================================================
# I2C for Audio and Video-In
#============================================================
set_location_assignment PIN_J12 -to FPGA_I2C_SCLK
set_location_assignment PIN_K12 -to FPGA_I2C_SDAT

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to FPGA_I2C_SCLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to FPGA_I2C_SDAT

#============================================================
# SEG7
```

```
#============================================================
set_location_assignment PIN_AE26 -to HEX0_N[0]
set_location_assignment PIN_AE26 -to HEX0_N_0
set_location_assignment PIN_AE27 -to HEX0_N[1]
set_location_assignment PIN_AE27 -to HEX0_N_1
set_location_assignment PIN_AE28 -to HEX0_N[2]
set_location_assignment PIN_AE28 -to HEX0_N_2
set_location_assignment PIN_AG27 -to HEX0_N[3]
set_location_assignment PIN_AG27 -to HEX0_N_3
set_location_assignment PIN_AF28 -to HEX0_N[4]
set_location_assignment PIN_AF28 -to HEX0_N_4
set_location_assignment PIN_AG28 -to HEX0_N[5]
set_location_assignment PIN_AG28 -to HEX0_N_5
set_location_assignment PIN_AH28 -to HEX0_N[6]
set_location_assignment PIN_AH28 -to HEX0_N_6
set_location_assignment PIN_AJ29 -to HEX1_N[0]
set_location_assignment PIN_AJ29 -to HEX1_N_0
set_location_assignment PIN_AH29 -to HEX1_N[1]
set_location_assignment PIN_AH29 -to HEX1_N_1
set_location_assignment PIN_AH30 -to HEX1_N[2]
set_location_assignment PIN_AH30 -to HEX1_N_2
set_location_assignment PIN_AG30 -to HEX1_N[3]
set_location_assignment PIN_AG30 -to HEX1_N_3
set_location_assignment PIN_AF29 -to HEX1_N[4]
set_location_assignment PIN_AF29 -to HEX1_N_4
set_location_assignment PIN_AF30 -to HEX1_N[5]
set_location_assignment PIN_AF30 -to HEX1_N_5
set_location_assignment PIN_AD27 -to HEX1_N[6]
set_location_assignment PIN_AD27 -to HEX1_N_6
set_location_assignment PIN_AB23 -to HEX2_N[0]
set_location_assignment PIN_AB23 -to HEX2_N_0
set_location_assignment PIN_AE29 -to HEX2_N[1]
set_location_assignment PIN_AE29 -to HEX2_N_1
set_location_assignment PIN_AD29 -to HEX2_N[2]
set_location_assignment PIN_AD29 -to HEX2_N_2
set_location_assignment PIN_AC28 -to HEX2_N[3]
set_location_assignment PIN_AC28 -to HEX2_N_3
set_location_assignment PIN_AD30 -to HEX2_N[4]
set_location_assignment PIN_AD30 -to HEX2_N_4
set_location_assignment PIN_AC29 -to HEX2_N[5]
set_location_assignment PIN_AC29 -to HEX2_N_5
set_location_assignment PIN_AC30 -to HEX2_N[6]
set_location_assignment PIN_AC30 -to HEX2_N_6
set_location_assignment PIN_AD26 -to HEX3_N[0]
set_location_assignment PIN_AD26 -to HEX3_N_0
set_location_assignment PIN_AC27 -to HEX3_N[1]
set_location_assignment PIN_AC27 -to HEX3_N_1
set_location_assignment PIN_AD25 -to HEX3_N[2]
set_location_assignment PIN_AD25 -to HEX3_N_2
set_location_assignment PIN_AC25 -to HEX3_N[3]
set_location_assignment PIN_AC25 -to HEX3_N_3
set_location_assignment PIN_AB28 -to HEX3_N[4]
set_location_assignment PIN_AB28 -to HEX3_N_4
set_location_assignment PIN_AB25 -to HEX3_N[5]
set_location_assignment PIN_AB25 -to HEX3_N_5
set_location_assignment PIN_AB22 -to HEX3_N[6]
set_location_assignment PIN_AB22 -to HEX3_N_6
set_location_assignment PIN_AA24 -to HEX4_N[0]
set_location_assignment PIN_AA24 -to HEX4_N_0
```

```
set_location_assignment PIN_Y23  -to HEX4_N[1]
set_location_assignment PIN_Y23  -to HEX4_N_1
set_location_assignment PIN_Y24  -to HEX4_N[2]
set_location_assignment PIN_Y24  -to HEX4_N_2
set_location_assignment PIN_W22  -to HEX4_N[3]
set_location_assignment PIN_W22  -to HEX4_N_3
set_location_assignment PIN_W24  -to HEX4_N[4]
set_location_assignment PIN_W24  -to HEX4_N_4
set_location_assignment PIN_V23  -to HEX4_N[5]
set_location_assignment PIN_V23  -to HEX4_N_5
set_location_assignment PIN_W25  -to HEX4_N[6]
set_location_assignment PIN_W25  -to HEX4_N_6
set_location_assignment PIN_V25  -to HEX5_N[0]
set_location_assignment PIN_V25  -to HEX5_N_0
set_location_assignment PIN_AA28 -to HEX5_N[1]
set_location_assignment PIN_AA28 -to HEX5_N_1
set_location_assignment PIN_Y27  -to HEX5_N[2]
set_location_assignment PIN_Y27  -to HEX5_N_2
set_location_assignment PIN_AB27 -to HEX5_N[3]
set_location_assignment PIN_AB27 -to HEX5_N_3
set_location_assignment PIN_AB26 -to HEX5_N[4]
set_location_assignment PIN_AB26 -to HEX5_N_4
set_location_assignment PIN_AA26 -to HEX5_N[5]
set_location_assignment PIN_AA26 -to HEX5_N_5
set_location_assignment PIN_AA25 -to HEX5_N[6]
set_location_assignment PIN_AA25 -to HEX5_N_6

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0_N_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1_N_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N_1
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2_N_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3_N_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4_N_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5_N_6

#============================================================
# IR
#============================================================
set_location_assignment PIN_AA30 -to IRDA_RXD
set_location_assignment PIN_AB30 -to IRDA_TXD
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to IRDA_RXD
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to IRDA_TXD

#============================================================
# KEY_N
#============================================================
set_location_assignment PIN_AA14 -to KEY_N[0]
set_location_assignment PIN_AA14 -to KEY_N_0
set_location_assignment PIN_AA15 -to KEY_N[1]
set_location_assignment PIN_AA15 -to KEY_N_1
set_location_assignment PIN_W15  -to KEY_N[2]
set_location_assignment PIN_W15  -to KEY_N_2
set_location_assignment PIN_Y16  -to KEY_N[3]
set_location_assignment PIN_Y16  -to KEY_N_3

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY_N[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY_N_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY_N[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY_N_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY_N[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY_N_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY_N[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY_N_3

#============================================================
# LED
#============================================================
set_location_assignment PIN_V16 -to LEDR[0]
set_location_assignment PIN_V16 -to LEDR_0
set_location_assignment PIN_W16 -to LEDR[1]
set_location_assignment PIN_W16 -to LEDR_1
set_location_assignment PIN_V17 -to LEDR[2]
set_location_assignment PIN_V17 -to LEDR_2
set_location_assignment PIN_V18 -to LEDR[3]
set_location_assignment PIN_V18 -to LEDR_3
set_location_assignment PIN_W17 -to LEDR[4]
set_location_assignment PIN_W17 -to LEDR_4
set_location_assignment PIN_W19 -to LEDR[5]
set_location_assignment PIN_W19 -to LEDR_5
set_location_assignment PIN_Y19 -to LEDR[6]
set_location_assignment PIN_Y19 -to LEDR_6
set_location_assignment PIN_W20 -to LEDR[7]
set_location_assignment PIN_W20 -to LEDR_7
set_location_assignment PIN_W21 -to LEDR[8]
set_location_assignment PIN_W21 -to LEDR_8
set_location_assignment PIN_Y21 -to LEDR[9]
set_location_assignment PIN_Y21 -to LEDR_9

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[5]
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR_7
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR_8
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR_9

#============================================================
# PS2
#============================================================
set_location_assignment PIN_AD7 -to PS2_CLK
set_location_assignment PIN_AD9 -to PS2_CLK2
set_location_assignment PIN_AE7 -to PS2_DAT
set_location_assignment PIN_AE9 -to PS2_DAT2

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to PS2_CLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to PS2_CLK2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to PS2_DAT
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to PS2_DAT2

#============================================================
# SW
#============================================================
set_location_assignment PIN_AB12 -to SW[0]
set_location_assignment PIN_AB12 -to SW_0
set_location_assignment PIN_AC12 -to SW[1]
set_location_assignment PIN_AC12 -to SW_1
set_location_assignment PIN_AF9  -to SW[2]
set_location_assignment PIN_AF9  -to SW_2
set_location_assignment PIN_AF10 -to SW[3]
set_location_assignment PIN_AF10 -to SW_3
set_location_assignment PIN_AD11 -to SW[4]
set_location_assignment PIN_AD11 -to SW_4
set_location_assignment PIN_AD12 -to SW[5]
set_location_assignment PIN_AD12 -to SW_5
set_location_assignment PIN_AE11 -to SW[6]
set_location_assignment PIN_AE11 -to SW_6
set_location_assignment PIN_AC9  -to SW[7]
set_location_assignment PIN_AC9  -to SW_7
set_location_assignment PIN_AD10 -to SW[8]
set_location_assignment PIN_AD10 -to SW_8
set_location_assignment PIN_AE12 -to SW[9]
set_location_assignment PIN_AE12 -to SW_9

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW_5
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW_7
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW_8
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW_9

#============================================================
# Video-In
#============================================================
set_location_assignment PIN_H15 -to TD_CLK27
set_location_assignment PIN_D2  -to TD_DATA[0]
set_location_assignment PIN_D2  -to TD_DATA_0
set_location_assignment PIN_B1  -to TD_DATA[1]
set_location_assignment PIN_B1  -to TD_DATA_1
set_location_assignment PIN_E2  -to TD_DATA[2]
set_location_assignment PIN_E2  -to TD_DATA_2
set_location_assignment PIN_B2  -to TD_DATA[3]
set_location_assignment PIN_B2  -to TD_DATA_3
set_location_assignment PIN_D1  -to TD_DATA[4]
set_location_assignment PIN_D1  -to TD_DATA_4
set_location_assignment PIN_E1  -to TD_DATA[5]
set_location_assignment PIN_E1  -to TD_DATA_5
set_location_assignment PIN_C2  -to TD_DATA[6]
set_location_assignment PIN_C2  -to TD_DATA_6
set_location_assignment PIN_B3  -to TD_DATA[7]
set_location_assignment PIN_B3  -to TD_DATA_7
set_location_assignment PIN_A5  -to TD_HS
set_location_assignment PIN_F6  -to TD_RESET_N
set_location_assignment PIN_A3  -to TD_VS

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_CLK27
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_DATA_7
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_HS
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_RESET_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to TD_VS

#============================================================
# VGA
#============================================================
set_location_assignment PIN_B13 -to VGA_B[0]
set_location_assignment PIN_B13 -to VGA_B_0
```

```
set_location_assignment PIN_G13 -to VGA_B[1]
set_location_assignment PIN_G13 -to VGA_B_1
set_location_assignment PIN_H13 -to VGA_B[2]
set_location_assignment PIN_H13 -to VGA_B_2
set_location_assignment PIN_F14 -to VGA_B[3]
set_location_assignment PIN_F14 -to VGA_B_3
set_location_assignment PIN_H14 -to VGA_B[4]
set_location_assignment PIN_H14 -to VGA_B_4
set_location_assignment PIN_F15 -to VGA_B[5]
set_location_assignment PIN_F15 -to VGA_B_5
set_location_assignment PIN_G15 -to VGA_B[6]
set_location_assignment PIN_G15 -to VGA_B_6
set_location_assignment PIN_J14 -to VGA_B[7]
set_location_assignment PIN_J14 -to VGA_B_7
set_location_assignment PIN_F10 -to VGA_BLANK_N
set_location_assignment PIN_A11 -to VGA_CLK
set_location_assignment PIN_J9  -to VGA_G[0]
set_location_assignment PIN_J9  -to VGA_G_0
set_location_assignment PIN_J10 -to VGA_G[1]
set_location_assignment PIN_J10 -to VGA_G_1
set_location_assignment PIN_H12 -to VGA_G[2]
set_location_assignment PIN_H12 -to VGA_G_2
set_location_assignment PIN_G10 -to VGA_G[3]
set_location_assignment PIN_G10 -to VGA_G_3
set_location_assignment PIN_G11 -to VGA_G[4]
set_location_assignment PIN_G11 -to VGA_G_4
set_location_assignment PIN_G12 -to VGA_G[5]
set_location_assignment PIN_G12 -to VGA_G_5
set_location_assignment PIN_F11 -to VGA_G[6]
set_location_assignment PIN_F11 -to VGA_G_6
set_location_assignment PIN_E11 -to VGA_G[7]
set_location_assignment PIN_E11 -to VGA_G_7
set_location_assignment PIN_B11 -to VGA_HS
set_location_assignment PIN_A13 -to VGA_R[0]
set_location_assignment PIN_A13 -to VGA_R_0
set_location_assignment PIN_C13 -to VGA_R[1]
set_location_assignment PIN_C13 -to VGA_R_1
set_location_assignment PIN_E13 -to VGA_R[2]
set_location_assignment PIN_E13 -to VGA_R_2
set_location_assignment PIN_B12 -to VGA_R[3]
set_location_assignment PIN_B12 -to VGA_R_3
set_location_assignment PIN_C12 -to VGA_R[4]
set_location_assignment PIN_C12 -to VGA_R_4
set_location_assignment PIN_D12 -to VGA_R[5]
set_location_assignment PIN_D12 -to VGA_R_5
set_location_assignment PIN_E12 -to VGA_R[6]
set_location_assignment PIN_E12 -to VGA_R_6
set_location_assignment PIN_F13 -to VGA_R[7]
set_location_assignment PIN_F13 -to VGA_R_7
set_location_assignment PIN_C10 -to VGA_SYNC_N
set_location_assignment PIN_D11 -to VGA_VS

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[3]
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_B_7
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_BLANK_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_CLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_G_7
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_HS
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_R_7
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_SYNC_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to VGA_VS

#============================================================
# HPS
#============================================================
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[0]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_0
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[1]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_1
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[2]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_2
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[3]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_3
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[4]
```

```
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_4
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[5]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_5
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[6]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_6
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[7]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_7
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[8]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_8
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[9]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_9
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[10]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_10
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[11]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_11
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[12]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_12
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[13]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_13
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR[14]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ADDR_14
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_BA[0]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_BA_0
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_BA[1]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_BA_1
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_BA[2]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_BA_2
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DM[0]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DM_0
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DM[1]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DM_1
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DM[2]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DM_2
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DM[3]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DM_3
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[0]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_0
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[1]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_1
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[2]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_2
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[3]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_3
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[4]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_4
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[5]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_5
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[6]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_6
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[7]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_7
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[8]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_8
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[9]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_9
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[10]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_10
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[11]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_11
```

```
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[12]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_12
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[13]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_13
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[14]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_14
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[15]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_15
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[16]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_16
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[17]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_17
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[18]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_18
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[19]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_19
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[20]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_20
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[21]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_21
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[22]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_22
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[23]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_23
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[24]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_24
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[25]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_25
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[26]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_26
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[27]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_27
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[28]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_28
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[29]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_29
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[30]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_30
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ[31]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_DQ_31
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_CAS_N
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_CKE
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_CS_N
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_ODT
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_RAS_N
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_WE_N
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_RESET_N
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to HPS_DDR3_RZQ
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_P[0]
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_P_0
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_N[0]
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_N_0
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_P[1]
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_P_1
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_N[1]
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_N_1
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_P[2]
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_P_2
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_N[2]
```

```
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_N_2
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_P[3]
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_P_3
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_N[3]
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_DQS_N_3
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_CK_P
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to HPS_DDR3_CK_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_GTX_CLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_INT_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_MDC
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_MDIO
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_RX_CLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_RX_DATA[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_RX_DATA_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_RX_DATA[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_RX_DATA_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_RX_DATA[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_RX_DATA_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_RX_DATA[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_RX_DATA_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_RX_DV
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_TX_DATA[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_TX_DATA_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_TX_DATA[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_TX_DATA_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_TX_DATA[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_TX_DATA_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_TX_DATA[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_TX_DATA_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_ENET_TX_EN
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_FLASH_DATA[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_FLASH_DATA_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_FLASH_DATA[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_FLASH_DATA_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_FLASH_DATA[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_FLASH_DATA_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_FLASH_DATA[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_FLASH_DATA_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_FLASH_DCLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_FLASH_NCSO
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_GPIO[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_GPIO_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_GPIO[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_GPIO_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_GSENSOR_INT
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_I2C1_SCLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_I2C1_SDAT
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_I2C2_SCLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_I2C2_SDAT
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_I2C_CONTROL
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_KEY_N
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_LED
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SD_CLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SD_CMD
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SD_DATA[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SD_DATA_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SD_DATA[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SD_DATA_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SD_DATA[2]
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SD_DATA_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SD_DATA[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SD_DATA_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SPIM_CLK
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SPIM_MISO
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SPIM_MOSI
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_SPIM_SS
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_UART_RX
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_UART_TX
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_CLKOUT
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DATA_7
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_DIR
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_NXT
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_USB_STP
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HPS_CONV_USB_N

#============================================================
# GPIO_0, GPIO_0 connect to GPIO Default
#============================================================
set_location_assignment PIN_AC18 -to GPIO_0[0]
set_location_assignment PIN_AC18 -to GPIO_0_0
set_location_assignment PIN_Y17  -to GPIO_0[1]
set_location_assignment PIN_Y17  -to GPIO_0_1
set_location_assignment PIN_AD17 -to GPIO_0[2]
set_location_assignment PIN_AD17 -to GPIO_0_2
set_location_assignment PIN_Y18  -to GPIO_0[3]
set_location_assignment PIN_Y18  -to GPIO_0_3
set_location_assignment PIN_AK16 -to GPIO_0[4]
set_location_assignment PIN_AK16 -to GPIO_0_4
set_location_assignment PIN_AK18 -to GPIO_0[5]
set_location_assignment PIN_AK18 -to GPIO_0_5
set_location_assignment PIN_AK19 -to GPIO_0[6]
set_location_assignment PIN_AK19 -to GPIO_0_6
set_location_assignment PIN_AJ19 -to GPIO_0[7]
set_location_assignment PIN_AJ19 -to GPIO_0_7
set_location_assignment PIN_AJ17 -to GPIO_0[8]
set_location_assignment PIN_AJ17 -to GPIO_0_8
set_location_assignment PIN_AJ16 -to GPIO_0[9]
set_location_assignment PIN_AJ16 -to GPIO_0_9
set_location_assignment PIN_AH18 -to GPIO_0[10]
set_location_assignment PIN_AH18 -to GPIO_0_10
set_location_assignment PIN_AH17 -to GPIO_0[11]
set_location_assignment PIN_AH17 -to GPIO_0_11
set_location_assignment PIN_AG16 -to GPIO_0[12]
```

```
set_location_assignment PIN_AG16 -to GPIO_0_12
set_location_assignment PIN_AE16 -to GPIO_0[13]
set_location_assignment PIN_AE16 -to GPIO_0_13
set_location_assignment PIN_AF16 -to GPIO_0[14]
set_location_assignment PIN_AF16 -to GPIO_0_14
set_location_assignment PIN_AG17 -to GPIO_0[15]
set_location_assignment PIN_AG17 -to GPIO_0_15
set_location_assignment PIN_AA18 -to GPIO_0[16]
set_location_assignment PIN_AA18 -to GPIO_0_16
set_location_assignment PIN_AA19 -to GPIO_0[17]
set_location_assignment PIN_AA19 -to GPIO_0_17
set_location_assignment PIN_AE17 -to GPIO_0[18]
set_location_assignment PIN_AE17 -to GPIO_0_18
set_location_assignment PIN_AC20 -to GPIO_0[19]
set_location_assignment PIN_AC20 -to GPIO_0_19
set_location_assignment PIN_AH19 -to GPIO_0[20]
set_location_assignment PIN_AH19 -to GPIO_0_20
set_location_assignment PIN_AJ20 -to GPIO_0[21]
set_location_assignment PIN_AJ20 -to GPIO_0_21
set_location_assignment PIN_AH20 -to GPIO_0[22]
set_location_assignment PIN_AH20 -to GPIO_0_22
set_location_assignment PIN_AK21 -to GPIO_0[23]
set_location_assignment PIN_AK21 -to GPIO_0_23
set_location_assignment PIN_AD19 -to GPIO_0[24]
set_location_assignment PIN_AD19 -to GPIO_0_24
set_location_assignment PIN_AD20 -to GPIO_0[25]
set_location_assignment PIN_AD20 -to GPIO_0_25
set_location_assignment PIN_AE18 -to GPIO_0[26]
set_location_assignment PIN_AE18 -to GPIO_0_26
set_location_assignment PIN_AE19 -to GPIO_0[27]
set_location_assignment PIN_AE19 -to GPIO_0_27
set_location_assignment PIN_AF20 -to GPIO_0[28]
set_location_assignment PIN_AF20 -to GPIO_0_28
set_location_assignment PIN_AF21 -to GPIO_0[29]
set_location_assignment PIN_AF21 -to GPIO_0_29
set_location_assignment PIN_AF19 -to GPIO_0[30]
set_location_assignment PIN_AF19 -to GPIO_0_30
set_location_assignment PIN_AG21 -to GPIO_0[31]
set_location_assignment PIN_AG21 -to GPIO_0_31
set_location_assignment PIN_AF18 -to GPIO_0[32]
set_location_assignment PIN_AF18 -to GPIO_0_32
set_location_assignment PIN_AG20 -to GPIO_0[33]
set_location_assignment PIN_AG20 -to GPIO_0_33
set_location_assignment PIN_AG18 -to GPIO_0[34]
set_location_assignment PIN_AG18 -to GPIO_0_34
set_location_assignment PIN_AJ21 -to GPIO_0[35]
set_location_assignment PIN_AJ21 -to GPIO_0_35

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[5]
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_7
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_8
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_9
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[10]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_10
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[11]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_11
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[12]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_12
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[13]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_13
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[14]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_14
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[15]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_15
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[16]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_16
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[17]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_17
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[18]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_18
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[19]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_19
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[20]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_20
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[21]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_21
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[22]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_22
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[23]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_23
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[24]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_24
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[25]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_25
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[26]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_26
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[27]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_27
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[28]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_28
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[29]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_29
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[30]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_30
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[31]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_31
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[32]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_32
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[33]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_33
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[34]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_34
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0[35]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_0_35

#============================================================
# GPIO_1, GPIO_1 connect to GPIO Default
#============================================================
set_location_assignment PIN_AB17 -to GPIO_1[0]
set_location_assignment PIN_AB17 -to GPIO_1_0
set_location_assignment PIN_AA21 -to GPIO_1[1]
set_location_assignment PIN_AA21 -to GPIO_1_1
set_location_assignment PIN_AB21 -to GPIO_1[2]
set_location_assignment PIN_AB21 -to GPIO_1_2
set_location_assignment PIN_AC23 -to GPIO_1[3]
set_location_assignment PIN_AC23 -to GPIO_1_3
set_location_assignment PIN_AD24 -to GPIO_1[4]
set_location_assignment PIN_AD24 -to GPIO_1_4
set_location_assignment PIN_AE23 -to GPIO_1[5]
set_location_assignment PIN_AE23 -to GPIO_1_5
set_location_assignment PIN_AE24 -to GPIO_1[6]
set_location_assignment PIN_AE24 -to GPIO_1_6
set_location_assignment PIN_AF25 -to GPIO_1[7]
set_location_assignment PIN_AF25 -to GPIO_1_7
set_location_assignment PIN_AF26 -to GPIO_1[8]
set_location_assignment PIN_AF26 -to GPIO_1_8
set_location_assignment PIN_AG25 -to GPIO_1[9]
set_location_assignment PIN_AG25 -to GPIO_1_9
set_location_assignment PIN_AG26 -to GPIO_1[10]
set_location_assignment PIN_AG26 -to GPIO_1_10
set_location_assignment PIN_AH24 -to GPIO_1[11]
set_location_assignment PIN_AH24 -to GPIO_1_11
set_location_assignment PIN_AH27 -to GPIO_1[12]
set_location_assignment PIN_AH27 -to GPIO_1_12
set_location_assignment PIN_AJ27 -to GPIO_1[13]
set_location_assignment PIN_AJ27 -to GPIO_1_13
set_location_assignment PIN_AK29 -to GPIO_1[14]
set_location_assignment PIN_AK29 -to GPIO_1_14
set_location_assignment PIN_AK28 -to GPIO_1[15]
set_location_assignment PIN_AK28 -to GPIO_1_15
set_location_assignment PIN_AK27 -to GPIO_1[16]
set_location_assignment PIN_AK27 -to GPIO_1_16
set_location_assignment PIN_AJ26 -to GPIO_1[17]
set_location_assignment PIN_AJ26 -to GPIO_1_17
set_location_assignment PIN_AK26 -to GPIO_1[18]
set_location_assignment PIN_AK26 -to GPIO_1_18
set_location_assignment PIN_AH25 -to GPIO_1[19]
set_location_assignment PIN_AH25 -to GPIO_1_19
set_location_assignment PIN_AJ25 -to GPIO_1[20]
set_location_assignment PIN_AJ25 -to GPIO_1_20
set_location_assignment PIN_AJ24 -to GPIO_1[21]
set_location_assignment PIN_AJ24 -to GPIO_1_21
set_location_assignment PIN_AK24 -to GPIO_1[22]
set_location_assignment PIN_AK24 -to GPIO_1_22
set_location_assignment PIN_AG23 -to GPIO_1[23]
set_location_assignment PIN_AG23 -to GPIO_1_23
set_location_assignment PIN_AK23 -to GPIO_1[24]
set_location_assignment PIN_AK23 -to GPIO_1_24
set_location_assignment PIN_AH23 -to GPIO_1[25]
set_location_assignment PIN_AH23 -to GPIO_1_25
set_location_assignment PIN_AK22 -to GPIO_1[26]
```

```
set_location_assignment PIN_AK22 -to GPIO_1_26
set_location_assignment PIN_AJ22 -to GPIO_1[27]
set_location_assignment PIN_AJ22 -to GPIO_1_27
set_location_assignment PIN_AH22 -to GPIO_1[28]
set_location_assignment PIN_AH22 -to GPIO_1_28
set_location_assignment PIN_AG22 -to GPIO_1[29]
set_location_assignment PIN_AG22 -to GPIO_1_29
set_location_assignment PIN_AF24 -to GPIO_1[30]
set_location_assignment PIN_AF24 -to GPIO_1_30
set_location_assignment PIN_AF23 -to GPIO_1[31]
set_location_assignment PIN_AF23 -to GPIO_1_31
set_location_assignment PIN_AE22 -to GPIO_1[32]
set_location_assignment PIN_AE22 -to GPIO_1_32
set_location_assignment PIN_AD21 -to GPIO_1[33]
set_location_assignment PIN_AD21 -to GPIO_1_33
set_location_assignment PIN_AA20 -to GPIO_1[34]
set_location_assignment PIN_AA20 -to GPIO_1_34
set_location_assignment PIN_AC22 -to GPIO_1[35]
set_location_assignment PIN_AC22 -to GPIO_1_35

set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_0
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_1
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_2
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_3
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_5
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_6
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_7
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_8
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_9
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[10]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_10
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[11]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_11
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[12]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_12
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[13]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_13
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[14]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_14
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[15]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_15
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[16]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_16
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[17]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_17
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[18]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_18
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[19]
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_19
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[20]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_20
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[21]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_21
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[22]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_22
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[23]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_23
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[24]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_24
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[25]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_25
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[26]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_26
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[27]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_27
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[28]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_28
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[29]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_29
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[30]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_30
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[31]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_31
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[32]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_32
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[33]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_33
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[34]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_34
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1[35]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to GPIO_1_35
```

Figure 15-2. DE1-SoC Pin Assignment TCL Script

16 REFERENCES

[1] Terasic Technologies, "Terasic - DE Main Boards - Cyclone - DE1-SoC Board," [Online]. Available: http://de1-soc.terasic.com.

[2] Altera Corporation, "Cyclone V Device Handbook, Volume 3: Hard Processor System Technical Reference Manual," 31 July 2014. [Online]. Available: http://www.altera.com/literature/hb/cyclone-v/cv_5v4.pdf.

[3] S. Kashani-Akhavan. [Online]. Available: https://github.com/sahandKashani/DE1-SoC/blob/master/DE1_SoC_demo.zip.

[4] S. Kashani-Akhavan. [Online]. Available: https://github.com/sahandKashani/Altera-FPGA-top-level-files/blob/master/DE1-SoC/DE1_SoC_top_level.vhd.

[5] S. Kashani-Akhavan. [Online]. Available: https://github.com/sahandKashani/Altera-FPGA-top-level-files/blob/master/DE1-SoC/pin_assignment_DE1_SoC.tcl.

[6] ISSI. [Online]. Available: https://github.com/sahandKashani/DE1-SoC/blob/master/Documentation/SDRAM%20Datasheet.pdf.

[7] Terasic Technologies, [Online]. Available: https://github.com/sahandKashani/DE1-SoC/blob/master/Documentation/DE1-SoC%20Schematic.pdf.

[8] ISSI. [Online]. Available: https://github.com/sahandKashani/DE1-SoC/blob/master/Documentation/DDR3%20SDRAM%20Datasheet.pdf.

[9] ARM, "DS-5 Debugger Commands," [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0452c/CIHJIBIH.html.

[10] Altera Corporation, "Documentation: Cyclone V Devices," [Online]. Available: http://www.altera.com/literature/lit-cyclone-v.jsp?ln=devices_fpga&l3=Low-Cost%20FPGAs-Cyclone%20V%20%28E,%20GX,%20GT,%20SE,%20SX,%20ST%29&l4=Documentation.

[11] Altera Corporation, "Address Map for HPS," [Online]. Available: http://www.altera.com/literature/hb/cyclone-v/hps.html.

[12] Altera Corporation, "A Look Inside: SoC FPGAs Embedded Development Tools (Part 5 of 5)," 25 November 2013. [Online]. Available: http://www.youtube.com/watch?v=NxZznvf5EKc.

[13] Altera Corporation, "A Look Inside: SoC FPGAs Introduction (Part 1 of 5)," 25 November 2013. [Online]. Available: http://www.youtube.com/watch?v=RVM-ESUMOMU.

[14] Altera Corporation, "A Look Inside: SoC FPGAs Reliability and Flexibility (Part 3 of 5)," 25 November 2013. [Online]. Available: http://www.youtube.com/watch?v=cWIaqt2RU84.

[15] Altera Corporation, "A Look Inside: SoC FPGAs System Cost and Power (Part 4 of 5)," 25 November 2013. [Online]. Available: http://www.youtube.com/watch?v=gUE669XKhUY.

[16] Altera Corporation, "A Look Inside: SoC FPGAs System Performance (Part 2 of 5)," 25 November 2013. [Online]. Available: http://www.youtube.com/watch?v=Ssxf8ggmQk4.

[17] Altera Corporation, "Cyclone V Device Datasheet," July 2014. [Online]. Available: http://www.altera.com/literature/hb/cyclone-v/cv_51002.pdf.

[18] Altera Corporation, "Cyclone V Device Handbook, Volume 1: Device Interfaces and Integration," 22 July 2014. [Online]. Available: http://www.altera.com/literature/hb/cyclone-v/cv_5v2.pdf.

[19] ARM, "DS-5 Altera Edition: Bare-metal Debug and Trace," 21 October 2013. [Online]. Available: http://www.youtube.com/watch?v=u_xKybPhcHI.

[20] ARM, "FPGA-adaptive debug on the Altera SoC using ARM DS-5," 16 December 2013. [Online]. Available: http://www.youtube.com/watch?v=2NBcUv2TxbI.

[21] EE Journal, "OpenCL on FPGAs Accelerating Performance and Design Productivity -- Altera," 28 November 2013. [Online]. Available: http://www.youtube.com/watch?v=M6vpq6s1h_A.

[22] Altera Corporation, "Bare-Metal Debugging using ARM DS-5 Altera Edition," 3 December 2013. [Online]. Available: http://www.youtube.com/watch?v=CJ0EHJ9oQ7Y.

[23] Altera Corporation, "Cyclone V Device Overview," 7 July 2014. [Online]. Available: http://www.altera.com/literature/hb/cyclone-v/cv_51001.pdf.

[24] Altera Corporation, "Linux Kernel Debug using ARM DS-5 Altera Edition," 3 December 2013. [Online]. Available: http://www.youtube.com/watch?v=QcA39O6ofGw.

[25] Altera Corporation, "Architecting FPGAs beyond 1M LEs," Altera Corporation, 3 September 2014. [Online]. Available: http://www.fpl2014.org/fileadmin/w00bpo/www/hutton.pdf.