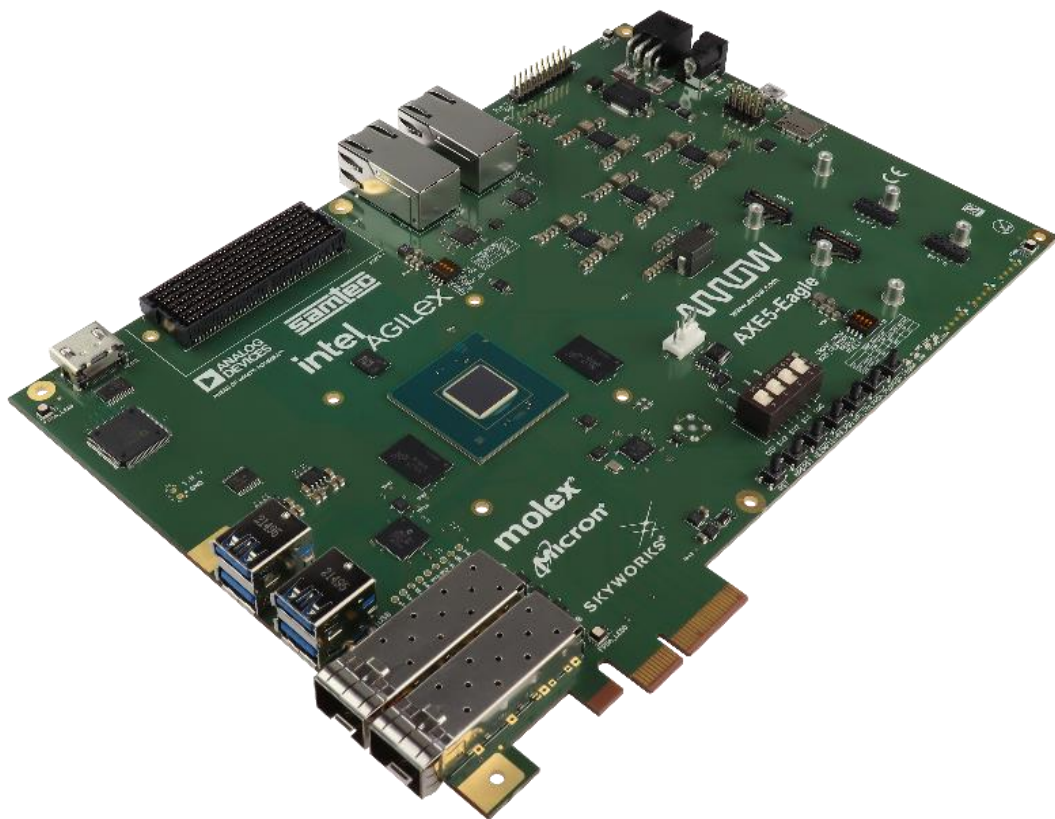




AXE5-Eagle

Create a Linux Boot Image



Software and hardware requirements to complete all exercises.

Requirements: Install all software as required in the installation Guide.



Document Control

Document Version:	Version 1.3
Document Date:	06/22/2025
Document Author(s):	Steven Kravatsky, Erika Peter, Naji Naufel, Fue Xiong, Helmut Ploetz, Henry Alexander, Mark Travaglini, ESC Team
Document Classification:	Released
Document Distribution:	This document is still under development. All specifications, procedures, and processes described in this document are subject to change without prior notice
Prior Version History:	Version 1.2

Please read the legal disclaimer at the end of this document.

Contents

1	Introduction	6
1.1	Readers Guide	6
1.2	Definitions	7
2	Getting Started	9
3	Agilex™ 5 SoC FPGAs	10
3.1	Agilex 5 Family of Devices	10
3.1.1	E Series, Group B.....	10
3.1.2	E Series, Group A.....	11
3.1.3	D Series	11
3.2	Agilex™ 5 SoC FPGA Architecture	12
3.2.1	The Secure Device Manager (SDM)	12
3.2.2	MPU Cluster	13
3.2.3	HPS/FPGA Bridges	13
3.3	AXE5-Eagle Golden System Reference Design (GSRD)	14
3.4	AXE5-Eagle Golden Hardware Reference Design (GHRD).....	14
3.5	Agilex 5 SoC FPGA Boot Overview.....	17
3.5.1	FPGA Configuration First Mode	17
3.5.2	HPS Boot First Mode.....	18
3.5.3	FPGA Configuration First Mode – Detail	18
3.6	System Layout for HPS Boot First Mode.....	22
3.6.1	Dual Flash System.....	22
3.7	The HPS Boot Flow	22
3.8	The HPS Boot Sequence	25
3.8.1	Exception Levels.....	25
3.8.2	Arm Trustzone	26
3.8.3	Arm Trusted Firmware (ATF), Boot Loader Stage 3-1 (BL31)	29
4	HPS Customization	31
4.1	HPS FPGA Interfaces.....	33
4.1.1	FPGA to SDRAM Bridge (F2SDRAM).....	34
4.1.2	Lightweight HPS to FPGA Bridge (LWH2F)	35
4.1.3	HPS to FPGA Bridge (H2F).....	36
4.1.4	FPGA to HPS Bridge (F2H).....	36
4.2	HPS Clocks, Resets, Power	38
4.2.1	Input Clocks	38
4.2.2	PLL Clocks.....	38
4.2.3	Power & Resets	39

4.3	Pin Mux and Peripherals	39
4.3.1	Advanced IP Placement	40
4.3.2	Advanced FPGA Placement	42
5	Creating a Bootable Image	43
5.1	Code repositories	45
5.2	Launch the Oracle VirtualBox Linux Virtual Machine	46
5.2.1	Launch the VirtualBox based Ubuntu 22.04 LTS Virtual Machine.	46
5.2.2	Copy and paste guide	46
5.2.3	Host password	47
5.2.4	VirtualBox Menu Bar	47
5.3	Setup the build environment	47
5.4	Arm Trusted Firmware (BL31)	47
5.5	U-boot	48
5.5.1	defconfig	48
5.5.2	Devicetree	48
5.5.3	Compile U-boot	49
5.6	Linux	51
5.6.1	defconfig	51
5.6.2	Devicetree	51
5.6.3	Compile Linux	51
5.7	Create a Linux Root File System (rootfs) with Yocto	53
5.8	Create the FPGA Configuration Bitstream Image	54
5.9	Create the SD Card Image	56
5.9.1	Write the SD Card image	57
5.10	Configure the board	59
5.10.1	Configure the MSEL DIP Switches	59
5.10.2	Assemble the Hardware	59
5.11	Connect to the target terminal	60
5.12	Boot the Linux Image	60
5.12.1	Enable the Arrow Blaster in the VirtualBox VM	60
5.12.2	Download the FPGA configuration file	61
5.12.3	Download the SOF file	62
5.13	View the Linux Boot Log	62
5.14	Turn RGB LEDs On and Off	64
5.14.1	Access the LEDS from Linux using devmem2	66
5.14.2	Specify each LED in Linux as a device	66
5.14.3	Defining the RGB LEDs as devices on the AXE5-Eagle board	67
5.14.4	Access the LEDS from Linux as devices	69



6	Additional Resources	70
7	Legal Disclaimer.....	71

1 Introduction

This lab provides comprehensive information showing the steps that an Agilex™ 5 SoC FPGA takes from power on to booting the Linux operating system.

You will review

- The architecture of the Agilex 5 SoC FPGA
- The AXE5-Eagle Golden System Reference Design (GSRD)
- The AXE5-Eagle Golden Hardware Reference Design (GHRD)
- The Agilex 5 SoC FPGA Configuration modes
- The HPS Boot flow
- The Linux Boot log from a successful boot sequence.

You will learn

- How to customize the Hard Processing System (HPS)
- About the different boot stages that the Secure Device Manager (SDM) and the HPS transition through to reach the Linux prompt.
- About the software required for the HPS Boot flow
- How to source and compile each of these software components.
- How to create a bootable Linux image.
- How to create a custom, flashable, FPGA image required by the SDM.
- How to run the bootable Linux image on an AXE5-Eagle board.

1.1 Readers Guide

If you are new to Agilex™ 5 SoC FPGAs and building Embedded Linux images, it is recommended that all sections of this document are read. Portions of section 3 and 4 can be disregarded depending on the readers' familiarity with the topic.

Abbreviations are referenced throughout the lab document. Use the list below for expansions/definitions of these abbreviations.

1.2 Definitions

ARM	Originally an abbreviation of Acorn RISC Machines
ARMv7-A	ARM version 7 for 32-bit Cortex A architecture devices
ARMv8-A	ARM version 8 for 64-bit Cortex A architecture devices
ATF	Arm Trusted Firmware
BL31	ATF Boot Loader section 3-1
CMF	Configuration Management Firmware
CPU	Central Processing Unit
DSU	ARM DynamicIQ Shared Unit
DTB	Device Tree Blob, binary version of Device Tree source
ECC	Error Checking and Correction
EL1, EL2, EL3, EL4	ARM Exception Levels 1 to 4
EMAC	Ethernet Media Access Controller
EMIF	External Memory Interface
eMMC	Embedded Multi Media Card
EOSC	External Oscillator
F2H	FPGA to HPS
F2SDRAM	FPGA to SDRAM
FAT	File Allocation Table
FIT	Flattened Image Tree
FIQ	Fast Interrupt Request
FPGA	Field Programmable Gate Array
FSBL	First Stage Boot Loader
GB	Giga Byte
GHRD	Golden Hardware Reference Design
GIC	Generic Interrupt Controller
GPIO	General Purpose Input Output
GSRD	Golden System Reference Design
H2F	HPS to FPGA
HPS	Hard Processing System
IO	Input Output
IP	Intellectual Property
ITS	Image Tree Source
ITB	Image Tree Blob, binary version of ITS
JIC	JTAG Indirect Configuration
JTAG	Joint Test Action Group

KB	Kilo Byte
L1, L2, L3	Level 1,2 or 3 cache memory
LPDDR4	Low Power DDR4
LSM	SDM Local Sector Manager
LWH2F	Lightweight HPS to FPGA
LZMA	Lempel-Ziv-Markov chain Algorithm
make	Utility to control the generation of executable code
MB	Mega Byte
mkimage	Utility used to create images for use with U-boot
MPU	Micro Processor Unit
MUX	Multiplexer
NS	Non-secure
OS	Operating System
OTG	On The Go
PDD	Platform Design Document
PSCI	Power State Coordination Interface
PLL	Phase Lock Loop
POR	Power On Reset
QSPI	Quad Serial Peripheral Interface
RAM	Random Access Memory
ROM	Read Only Memory
SD	Secure Digital
SDM	Secure Device Manager
SDRAM	Synchronous Dynamic Random Access Memory
SOC	System on Chip
SEU	Single Event Upset
SPI	Serial Peripheral Interface
SPIM	SPI Master
SPL	Secondary Program Loader
SSBL	Second Stage Boot Loader
TBBR	Trusted Board Boot Requirements
TEE	Trusted Execution Environment
UART	Universal Asynchronous Receiver Transmitter
UIMG	U-boot Image

2 Getting Started

The first objective is to ensure that you have all the necessary items so that the lab can be completed successfully. Below is a list of items required to complete this lab:

- Personal computer or laptop running 64-bit Windows 10 or later with at least an Intel i3 core (or equivalent), 8GB RAM.
- The build environment will be provided on a VirtualBox Virtual Machine, running Ubuntu Linux 22.04 LTS. Refer to the installation guide for details.
- A desire to learn!

3 Agilex™ 5 SoC FPGAs

3.1 Agilex 5 Family of Devices

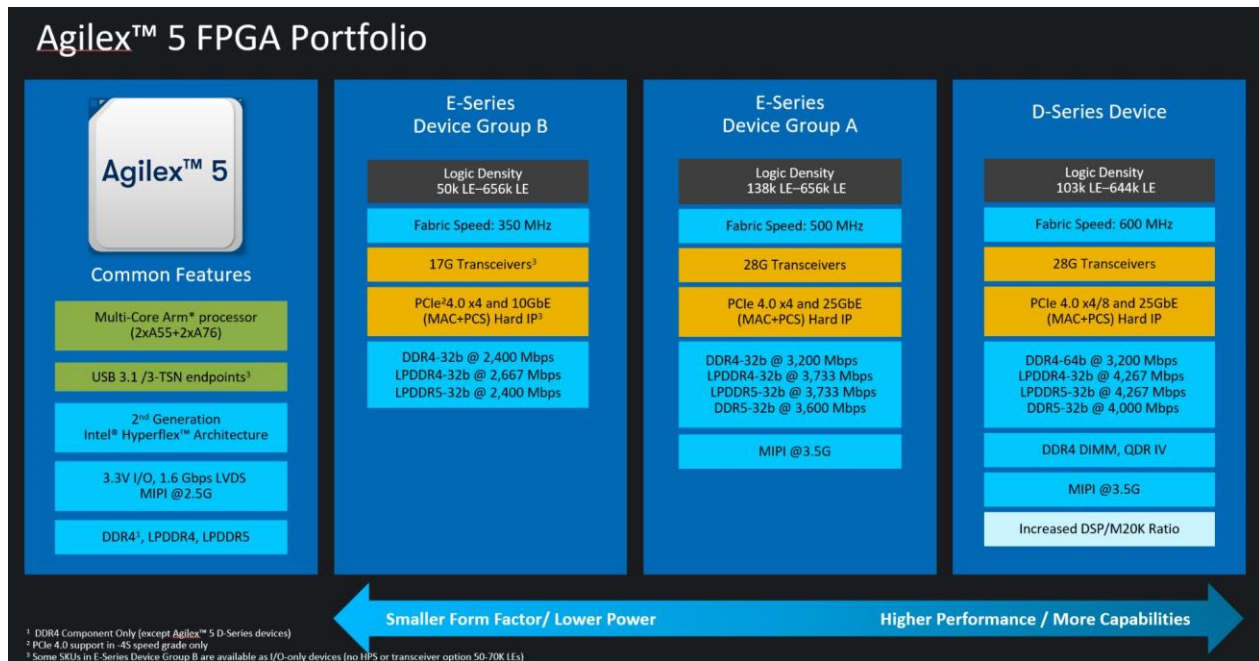


Figure 3-1 : Agilex 5 Family of Devices

Three major variants are offered in the Agilex 5 Product family.

- D series
- E Series, Group A
- E Series, Group B

3.1.1 E Series, Group B

Power Optimized FPGA with IO Only- these are the smallest devices in the family that are meant for the most power efficient applications. These will have IO only- no ARM processor or transceivers.

Power Optimized FPGA with HPS - these devices will have a quad core ARM option and 17G transceivers. These devices are for mid-range applications that will need an ARM processor and other peripherals.

3.1.2 E Series, Group A

Performance Optimized FPGA with HPS - These are the highest-performance devices in the E Series family. These devices have a higher fabric speed and a 28G transceiver option, along with the quad core ARM processor.

3.1.3 D Series

Performance Optimized FPGA with HPS - These are the highest-performance devices in this family. These devices have the highest fabric speed, highest external memory speeds and a 28G transceiver option, along with the quad core ARM processor.

The AXE5-Eagle board uses an **E-series, group B device** that includes a quad core ARM and transceivers.

3.2 Agilex™ 5 SoC FPGA Architecture

The Agilex™ 5 system-on-a-chip (SoC) is composed of two distinct portions: a dual-core Arm Cortex-A76 and dual-core Arm Cortex-A55 Hard Processor System (HPS), and a FPGA. The HPS architecture integrates a wide set of peripherals that reduces board size and increases performance within a system. A short description of some key sections of the HPS is provided below.

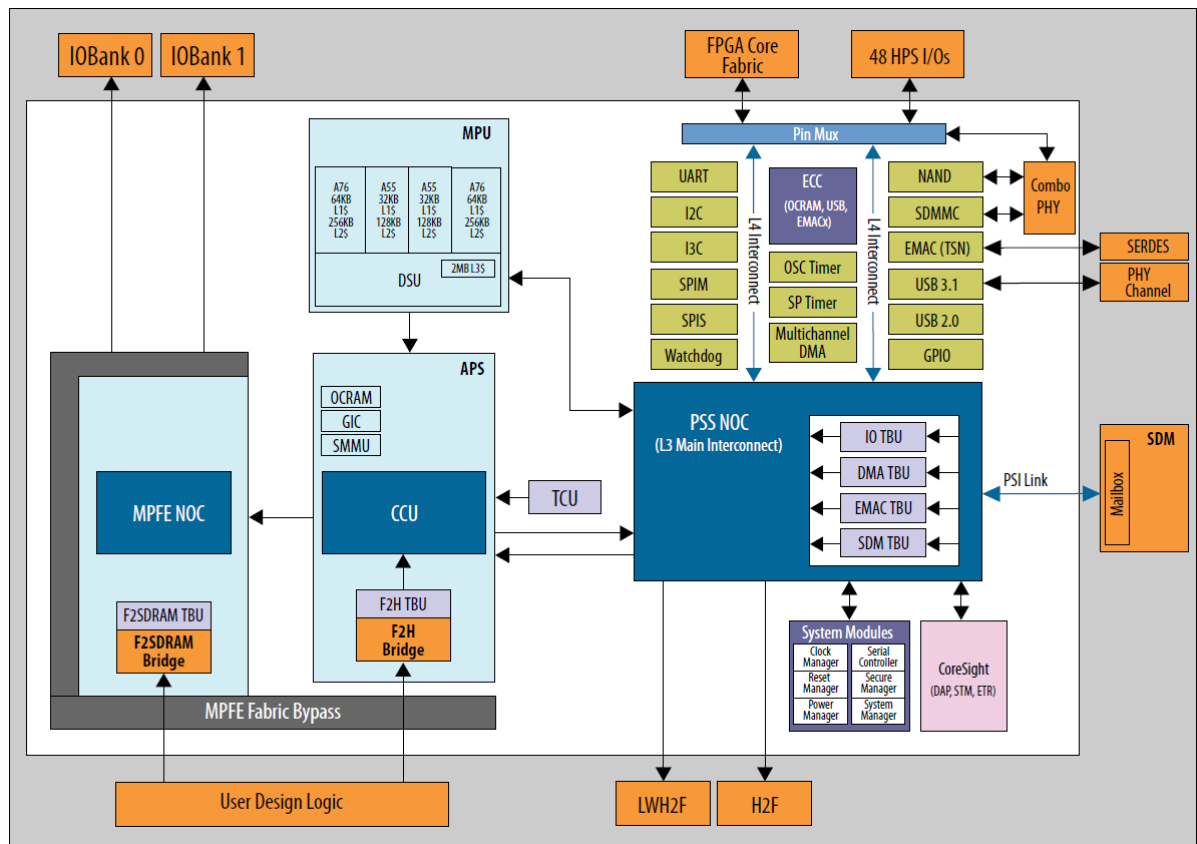


Figure 3-2 : Agilex 5 Architecture

3.2.1 The Secure Device Manager (SDM)

All Agilex™ 5 FPGAs and SoCs contain an SDM. The SDM is a triple-redundant processor that serves as the point of entry into the device for all JTAG and configuration commands.

The SDM bootstraps the HPS in Agilex™ 5 SoCs. This bootstrapping ensures that the HPS boots using the same security features available to the FPGA.

3.2.2 MPU Cluster

The MPU Cluster includes the following features:

- Two Arm Cortex-A55 core with 32 KB L1 instruction cache and data cache per core and a unified 128 KB L2 cache per core
- Two Arm Cortex-A76 core with 64 KB L1 instruction cache and data cache per core and a unified 256 KB L2 cache per core
- DSU with 2 MB L3 cache
- Hardware cache coherency maintained using the L3 memory system
- ECC support for L1, L2 and L3 memories
- Static power-gated domains for Cortex-A55 and Cortex-A76 cores

3.2.3 HPS/FPGA Bridges

Bridges are used to move data between the FPGA fabric and HPS logic.

The Lightweight-to-FPGA (LWH2F) bridge extends the HPS peripherals to the FPGA like the H2F. However, the LWH2F is meant for a narrower (32-bit data bus) use case involving simple peripherals on the FPGA, where latency is prioritized over bandwidth. The LWH2F bridge is meant for strongly ordered single transactions.

This allows usage of the LWH2F as the configuration bus for FPGA IPs. The FPGA IP can then make use of H2F or F2H/F2SDRAM as the main data mover bus.

The HPS-to-FPGA (H2F) bridge extends the HPS peripherals to the FPGA. Additional IPs implemented on FPGA can be used as part of the HPS subsystem. The H2F bridge can also be connected to another 256GB of FPGA SDRAM, extending the amount of physical memory available to HPS.

The FPGA-to-HPS (F2H) bridge provides a way for initiators (IPs, accelerators) in the fabric to access HPS peripherals, which makes the peripherals extensions of the FPGA system.

The FPGA-to-SDRAM (F2SDRAM) bridge provides the asynchronous clock domain crossing logic for the F2SDRAM port from the fabric. The primary traffic is transactions to the DRAM subsystems from all the fabric agents.

3.3 AXE5-Eagle Golden System Reference Design (GSRD)

The AXE5-Eagle Golden System Reference Design is a thoroughly tested known good design showcasing a system using both HPS and FPGA resources, intended to be used as a baseline project.

The GSRD is comprised of the following components:

- Golden Hardware Reference Design (GHRD)
- Reference HPS software including:
 - Arm Trusted Firmware
 - U-Boot
 - Linux Kernel
 - Linux drivers
 - Sample applications

The current GSRD uses FPGA-First configuration mode.

3.4 AXE5-Eagle Golden Hardware Reference Design (GHRD)

The AXE5-Eagle GHRD, part of the AXE5-Eagle Golden System Reference Design (GSRD), is an Intel® Quartus® Prime project that contains a full HPS design for the Arrow AXE5-Eagle board. The GHRD has connections to a boot source, SDRAM memory and other peripherals on the development board.

You must always use a hardware design with the Intel® Agilex™ SoC if you choose to take advantage of the HPS features. The purpose of the hardware design is to configure the SoC, including the FPGA portion, the HPS pin multiplexers and I/Os, and the SDRAM. All software projects depend on a hardware design.

The GHRD is regression tested with every major release of the Quartus Prime Design Suite (QPDS) and includes the latest bug fixes for known hardware issues. As such, the GHRD serves as a well-known configuration of a SoC FPGA hardware system.

GUIDELINE: Use the latest GHRD as a baseline for new SoC FPGA hardware projects. You may then modify the design to suit your end application needs.

A block diagram of the GHRD is shown in figure 3-3 below. The design is constructed using the Altera Platform Designer tool. The GHRD is constructed as a hierarchical design. The GHRD represents the top level, with a number of subsystems.

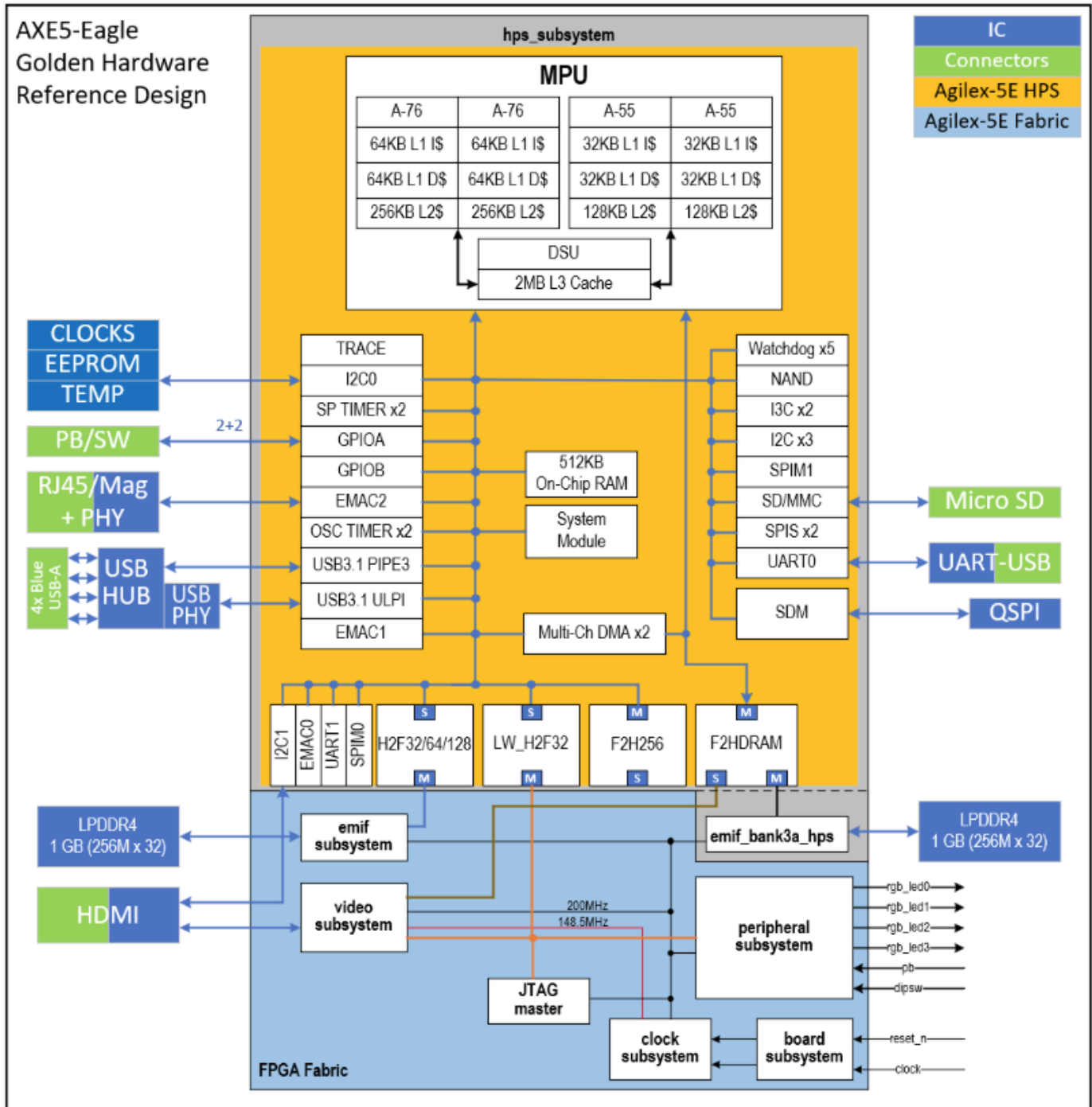


Figure 3-3 : GHRD Block Diagram

The following subsystems are included in the GHRD:

- Board
 - Board level reset and clock inputs
- Clock
 - Includes a Phase Lock Loop that generates clocks for all clock domains in the design. Each clock has an associated, synchronous reset.
- HPS
 - This subsystem includes the customized HPS section, the HPS EMIF controller and a JTAG interface for use with System Console.
- EMIF
 - This subsystem adds an additional EMIF controller. This can be mastered from the FPGA fabric or from the HPS.
- Peripheral
 - This is a collection of FPGA fabric GPIO peripherals used for controlling LEDs and reading pushbutton and DIP switch inputs. It also includes two GPIOs used for releasing the Ethernet PHY and USB PHY from reset.
- Ethernet
 - An additional HPS EMAC is routed through the FPGA and connected to the second set of ethernet hardware on the board.
- Video
 - The AXE5-Eagle includes an ADV7511 HDMI PHY output device. This can be used by a Nios V soft processor or the HPS ARM cores running Linux. It can be used to provide a graphical desktop for Linux in conjunction with a USB keyboard and mouse. The subsystem includes a DMA used for reading video from a framebuffer and writing it to an HDMI peripheral. The HDMI peripheral manipulates the raw video data into a format compatible with the ADV7511 PHY.

The GHRD top level in Platform Designer can be viewed in Figure 3-4.

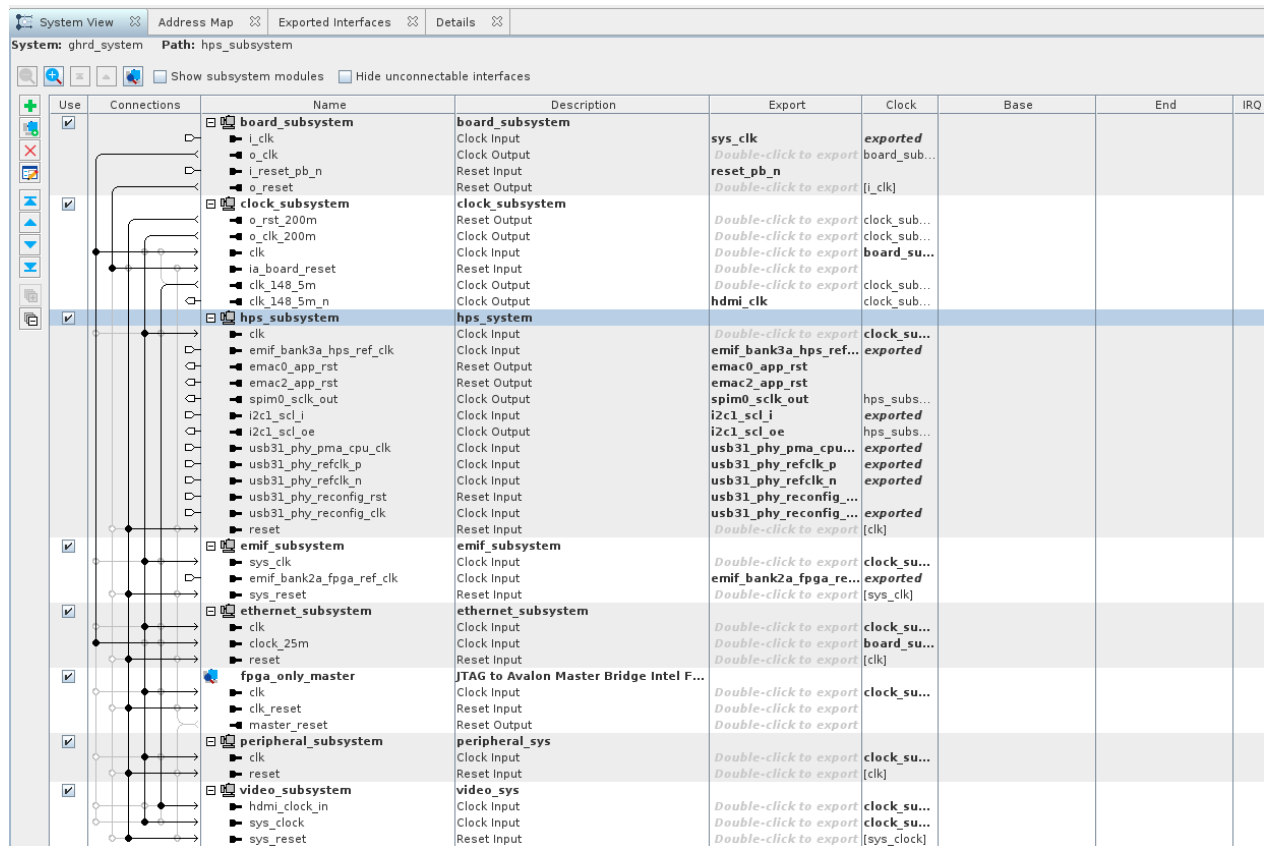


Figure 3-4 : GHRD Top level view in Platform Designer

3.5 Agilex 5 SoC FPGA Boot Overview

The Agilex 5 SoC FPGA combines an FPGA with a hard processor system (HPS) that is capable of booting operating systems such as Linux and Zephyr. When booting the device from a power-on reset (POR), you can choose between two different methods of booting.

3.5.1 FPGA Configuration First Mode

When you select the FPGA First option, the SDM fully configures the FPGA, then configures the HPS IO and HPS EMIF controller, loads the HPS first-stage bootloader (FSBL) and takes the HPS out of reset. *Note:* The FPGA and all of the IOs are fully configured before the HPS is released from reset. Thus, when the HPS boots, the FPGA is in user mode and is ready to interact with the HPS.

3.5.2 HPS Boot First Mode

When you select the HPS First option, the SDM first configures the HPS IO and HPS EMIF controller, loads the HPS FSBL and takes the HPS out of reset. Then the HPS configures the FPGA IO and FPGA fabric at a later time. *Note:* This mode is also referred to as Early IO Release Mode or Early IO Configuration. After power-on, the device configures a minimal amount of IO required by the HPS before releasing the HPS from reset. This mode allows the HPS to boot quickly without having to wait for the full configuration to complete. Subsequently, the HPS may trigger an FPGA configuration request during the SSBL or OS stage.

The GSRD currently uses the **HPS Boot First Mode**.

3.5.3 FPGA Configuration First Mode – Detail

Figure 3-4 shows the boot stages of the Agilex 5 device from POR all the way to a user application running on an operating system.

Table 1 provides more detail on each boot stage.

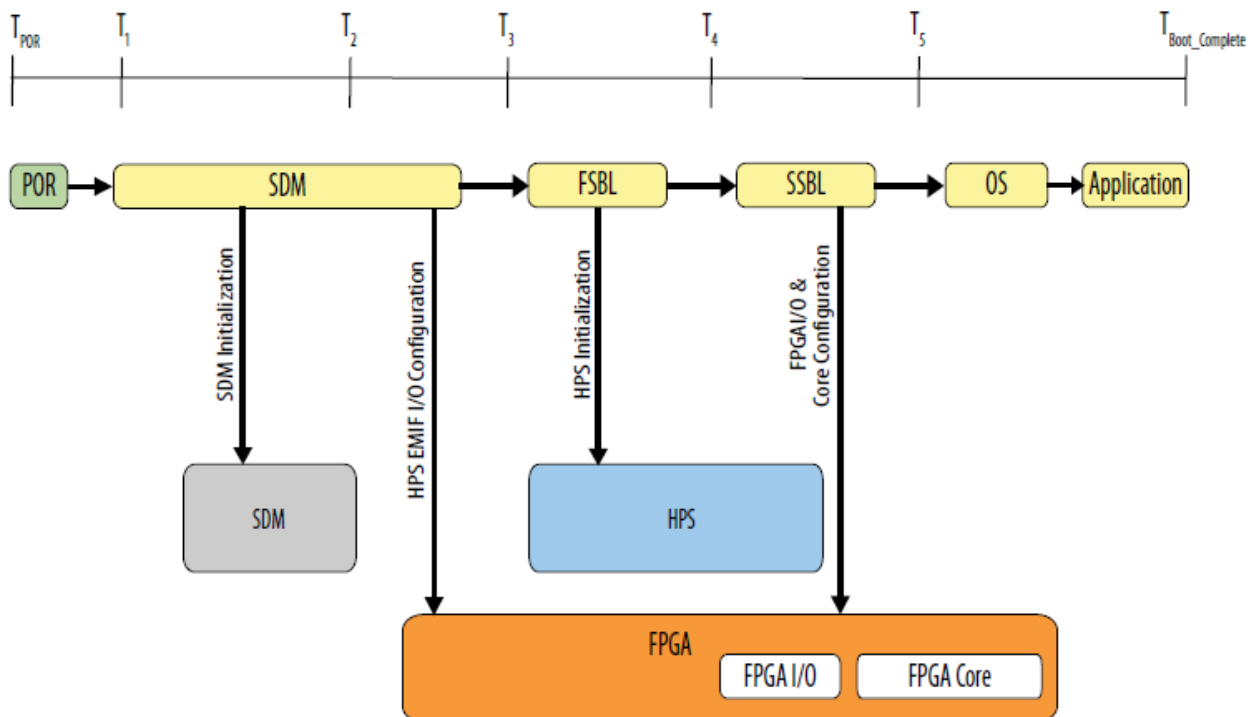


Figure 3-5 : HPS First Boot Flow

Time	Boot Stage	Device State
T _{POR}	POR	Power-on reset
T ₁ to T ₂	SDM- Boot ROM	<ol style="list-style-type: none"> 1. SDM samples the MSEL pins to determine the configuration and boot source. It also establishes the device security level based on eFuse values. 2. SDM firmware initializes the device. 3. SDM authenticates and decrypts the bitstream (this process occurs as necessary throughout the configuration).
T ₂ to T ₃	SDM- Configuration Firmware	<ol style="list-style-type: none"> 1. SDM configures the HPS EMIF I/O and the rest of the user-configured SDM I/O. 2. SDM loads the FSBL from the bitstream into HPS on-chip RAM. 3. HPS boot core start executing FSBL code. 4. SDM enables HPS SDRAM I/O and optionally enables HPS debug. 5. HPS is released from reset.
T ₃ to T ₄	First Stage Bootloader (FSBL)	<ol style="list-style-type: none"> 1. The FSBL initializes the HPS, including the SDRAM. 2. FSBL obtains the SSBL from HPS flash or by requesting flash access from the SDM. 3. FSBL loads the SSBL into SDRAM. 4. HPS peripheral I/O pin multiplexer and buffers are configured. Clocks, resets and bridges are also configured. 5. HPS I/O peripherals are available. 6. HPS bootstrap completes.
T ₄ to T ₅	Second Stage Bootloader (SSBL)	<p>After bootstrap completes, any of the following steps may occur:</p> <ol style="list-style-type: none"> 1. The FPGA core configuration loads into SDRAM from one of the following sources: <ul style="list-style-type: none"> • SDM flash • HPS alternate flash • EMAC interface 2. HPS requests that the SDM configures the FPGA core. <i>Note:</i> This step is applicable for U-Boot ATF Linux boot only. For ATF Linux Boot and ATF Zephyr Boot, the FPGA configuration happens in the next stage. 3. FPGA enters user mode. 4. OS is loaded into SDRAM.
T ₅ to T _{Boot_Complete}	Operating System (OS)	<ol style="list-style-type: none"> 1. OS boot occurs and the OS schedules applications for runtime launch. 2. (Optional step) The OS initiates FPGA configuration through a secure monitor call (SMC) to the resident SMC handler (typically SSBL), which then initiates the request to the SDM.

Table 1 : FPGA Configuration First Stages

The sections following this table describe each stage in more detail.

3.5.3.1 Power-On Reset (POR)

Ensure you power each of the power rails according to the power sequencing consideration until they reach the required voltage levels. In addition, the power-up sequence must meet either the standard or the fast power-on reset (POR) delay time.

3.5.3.2 Secure Device Manager

The Secure Device Manager (SDM) is a triple-redundant processor-based module that manages configuration and the security features of Agilex 5 devices. The SDM is available on all Agilex 5 devices. The block diagram below provides an overview of the Agilex 5 configuration architecture which includes the following blocks:

- SDM: More information about the SDM is contained in later sections.

- Configuration network: The SDM uses this dedicated, parallel configuration network to distribute the configuration bitstream to Local Sector Managers (LSMs). You cannot access this network.
- LSMs: The LSM is a microprocessor. Each configuration sector includes an LSM. The LSM parses configuration bitstream and configures the logic elements for its sector. After configuration, the LSM performs the following functions:
 - Monitors for single event upsets at the sector level
 - Processes responses to single event upsets (SEUs)
 - Performs integrity checks in user mode

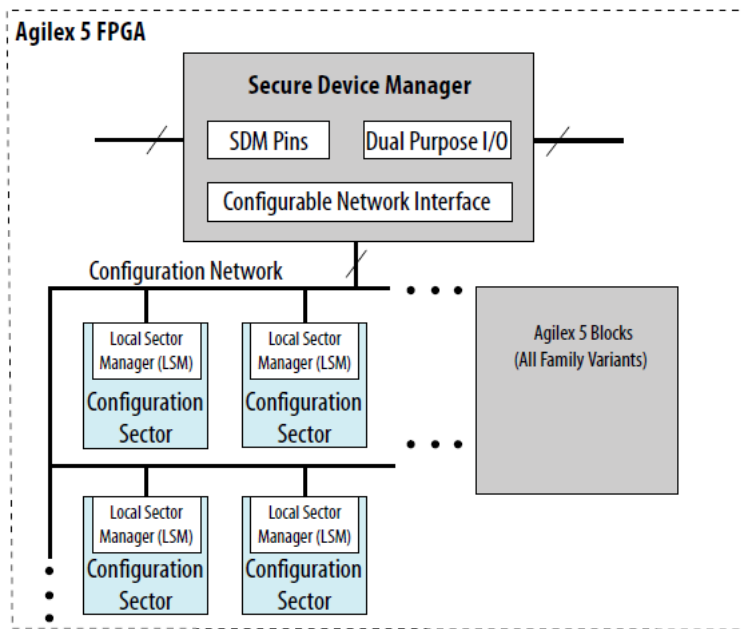


Figure 3-6 : Agilex 5 Configuration Architecture Block Diagram

Once the Agilex 5 SoC FPGA exits POR, the SDM samples the MSEL[2:0] pins to determine the boot source. Next, the device configures the SDM I/Os according to the selected boot source interface and the SDM retrieves the configuration bitstream through the interface. The typical configuration bitstream for HPS boot first mode contains:

- SDM configuration firmware HPS external memory interface (EMIF) I/O configuration data HPS FSBL code and FSBL hardware handoff binary data

The SDM completes the configuration of the HPS EMIF I/O and then copies the HPS FSBL to the HPS on-chip RAM.

3.5.3.3 First-Stage Bootloader

After the SDM releases the HPS from reset, the FSBL initializes the HPS. Initialization includes configuring clocks, HPS dedicated I/Os, and peripherals.

In HPS first boot mode, the SDM, HPS OSC and HPS EMIF clocks must be running stable and set at the correct frequency before you begin any part of the configuration sequence.

In HPS first boot mode, phase 1 configuration is successful as long as HPS OSC and HPS EMIF clocks are running stable.

You can create the FSBL from one of the following sources:

- U-Boot secondary program loader (SPL)
- Arm Trusted Firmware

3.5.3.4 Second-Stage Bootloader

The second-stage bootloader (SSBL) is the second boot stage for the HPS. The FSBL initiates the copy of the SSBL to the HPS SDRAM. The SSBL typically enables more advanced peripherals such as Ethernet and supports a command line interface.

You can create the HPS SSBL from one of the following sources:

- U-Boot secondary program loader (SPL)
- Arm Trusted Firmware

You can optionally perform FPGA core and I/O configuration in during the SSBL stage. The SSBL copies the FPGA configuration files from one of the following sources to the HPS SDRAM:

- HPS Flash
- SDM Flash
- External host via the HPS Ethernet (for example, TFTP)

After the SSBL copies the FPGA configuration files to the HPS SDRAM, the SSBL can initiate a configuration request to the SDM to begin the configuration process.

3.5.3.5 Operating System

Typically, the SSBL loads the operating system (OS) stage into SDRAM. The OS executes from SDRAM. Depending on your application requirements, you may implement a conventional OS or an RTOS.

3.5.3.6 Application

The application that runs on the OS is the last boot stage.

3.6 System Layout for HPS Boot First Mode

The following section describes the supported system layout for HPS Boot First mode. The OS is assumed to be Linux, but you may replace Linux with other supported operating systems.

3.6.1 Dual Flash System

In a dual flash system, the SDM flash stores the configuration bitstream, while the HPS flash stores the HPS SSBL and the rest of the OS files.

SDM Flash Type	HPS Flash Type
Active serial/Quad SPI	SD/eMMC

Table 2: Dual Flash Combination (SDM and HPS)

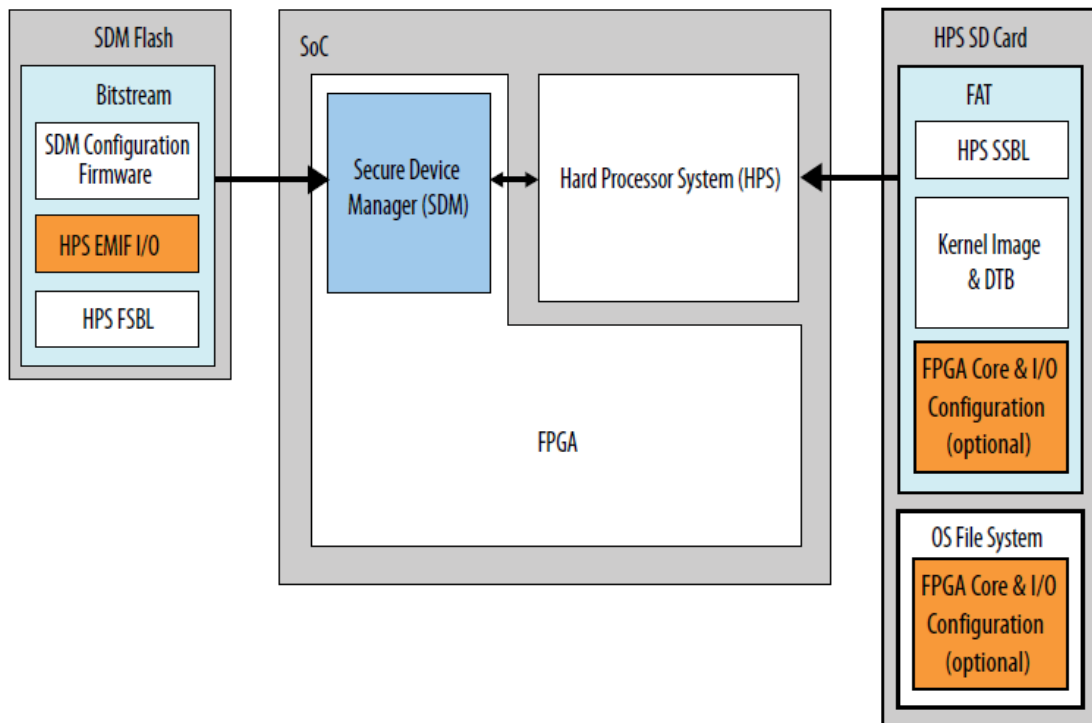


Figure 3-7 : Dual Flash Devices (SDM and HPS)

3.7 The HPS Boot Flow

The following section introduces the boot flows supported by the HPS.

There are 3 boot flows supported:

- U-Boot, ATF, Linux Boot

- U-Boot SPL ► ATF BL31 ► U-Boot ► Linux
- ATF, Linux Boot
 - ATF BL2 ► ATF BL31 ► Linux
- ATF, Zephyr Boot
 - ATF BL2 ► ATF BL31 ► Zephyr

U-Boot SPL is also known as the First Stage Boot Loader (FSBL).

ATF BL31 includes functionality known as the Secure Monitor.

U-Boot is also known as the Second Stage Boot Loader (SSBL).

The AXE5-Eagle board utilizes the U-Boot, ATF, Linux Boot flow.

The following figure shows the overview of the HPS Boot Flow using a U-Boot as HPS Bootloader to boot to the Linux OS.

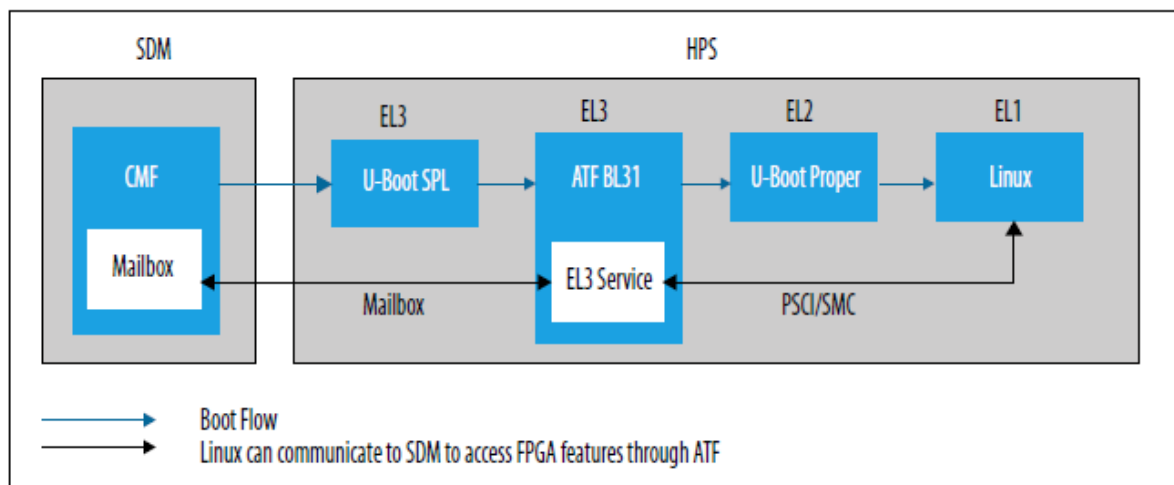


Figure 3-8 : HPS Boot flow

The boot flow is described in steps below:

- The Configuration Management Firmware (CMF), which is a part of the configuration bitstream, running on the SDM loads the FSBL, which is U-Boot SPL, into HPS On-Chip RAM and then brings the HPS boot core out from reset.
- The U-Boot SPL loads the SSBL, which is ATF BL31 and U-Boot proper (SSBL), into DDR.

- The U-Boot SPL jumps to ATF BL31.
- ATF BL31 sets up the GIC, EL3 environment, and initializes the PSCI services. PSCI services in ATF remain active or available once ATF jumps to U-Boot.
- ATF BL31 jumps to the U-Boot proper.
- U-Boot loads the Linux OS into the DDR.
- U-Boot jumps to the Linux OS.
- *Note:* U-Boot proper and the Linux OS can access the SDM FPGA features through ATF BL31 through the Arm Secure Monitor Call (SMC).

3.8 The HPS Boot Sequence

To understand the HPS boot sequence it is important to understand the concept of privilege and exception levels with respect to the Armv8-A (also known as AArch64) processor architecture.

Modern software is developed to be split into different modules, each with a different level of access to system and processor resources. An example of this is the split between the operating system kernel and user applications. The operating system needs to perform actions which we do not want a user application to be able to perform. The kernel needs a high level of access to system resources, whereas user applications need limited ability to configure the system. Privilege dictates which processor resources a software entity can see and control.

The AArch64 architectures enable this split by implementing different levels of privilege. The current privilege level can only change when the processor takes an exception or returns from an exception. Therefore, these privilege levels are referred to as Exception levels in the Arm architecture.

3.8.1 Exception Levels

The name for privilege in AArch64 is Exception level, often abbreviated to EL. The Exception levels are numbered, normally abbreviated and referred to as EL<x>, where <x> is a number between 0 and 3. The higher the level of privilege the higher the number. For example, the lowest level of privilege is referred to as EL0.

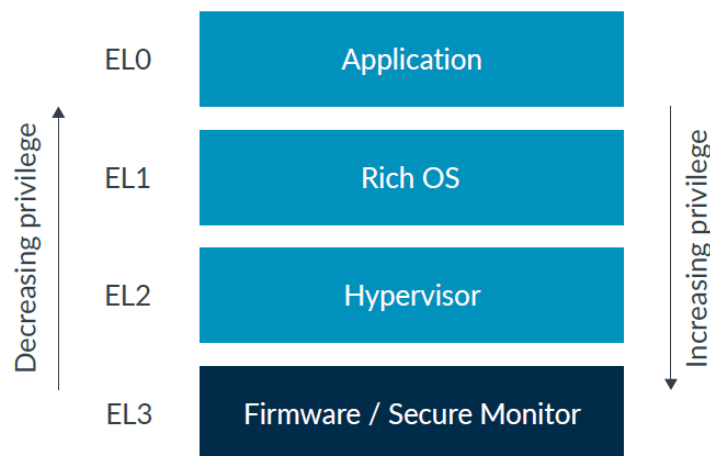


Figure 3-9 : Exception Levels

The architecture does not specify what software uses which Exception level. A common usage model is application code running at EL0, with a rich Operating

System (OS) such as Linux running at EL1. EL2 may be used by a hypervisor, with EL3 used by firmware and security gateway code.

For example, Linux can call firmware functions at EL3, using software interface standards, to abstract the intent from the lower-level details for powering on or off a core. This model means the bulk of PE processing typically occurs at EL0/1.

The Exception level can only change when any of the following occur:

- Taking an exception
- Returning from an exception
- Processor reset
- During Debug state
- Exiting from Debug state

When taking an exception, the Exception level can increase or stay the same. You can never move to a lower privilege level by taking an exception. When returning from an exception the Exception level can decrease or stay the same. You can never move to a higher privilege level by returning from an exception.

3.8.2 Arm Trustzone

TrustZone is the name of the Security architecture in the Arm A-profile architecture. First introduced in Armv6K, TrustZone is also supported in Armv7-A and Armv8-A. TrustZone provides two execution environments with system-wide hardware enforced isolation between them, as shown in this diagram:

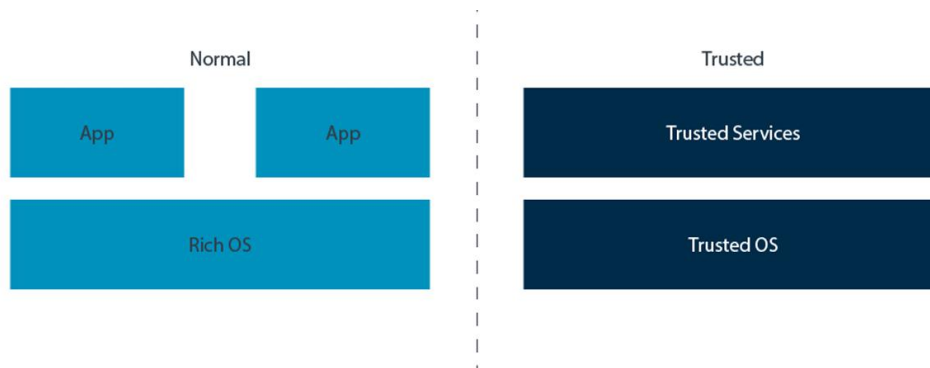


Figure 3-10 : Normal and Trusted worlds

3.8.2.1 Normal and Trusted worlds

The Normal world runs a rich software stack. This software stack typically includes a large application set, a complex operating system like Linux, and possibly a hypervisor. Such software stacks are large and complex. While efforts can be made to

secure them, the size of the attack surface means that they are more vulnerable to attack.

The Trusted world runs a smaller and simpler software stack, which is referred to as a Trusted Execution Environment (TEE). Typically, a TEE includes several Trusted services that are hosted by a lightweight kernel. The Trusted services provide functionality like key management. This software stack has a considerably smaller attack surface, which helps reduce vulnerability to attack.

3.8.2.2 Secure and Non-secure states

In the Arm architecture, there are two Security states: Secure and Non-secure. These Security states map onto the Trusted and Normal worlds.

At EL0, EL1, and EL2 the processor can be in either Secure state or Non-secure state, which is controlled by the SCR_EL3.NS bit. You often see this written as:

- NS.EL1: Non-secure state, Exception level 1
- S.EL1: Secure state, Exception level 1

EL3 is always in Secure state, regardless of the value of the SCR_EL3.NS bit. The arrangement of Security states and Exception levels is shown here:

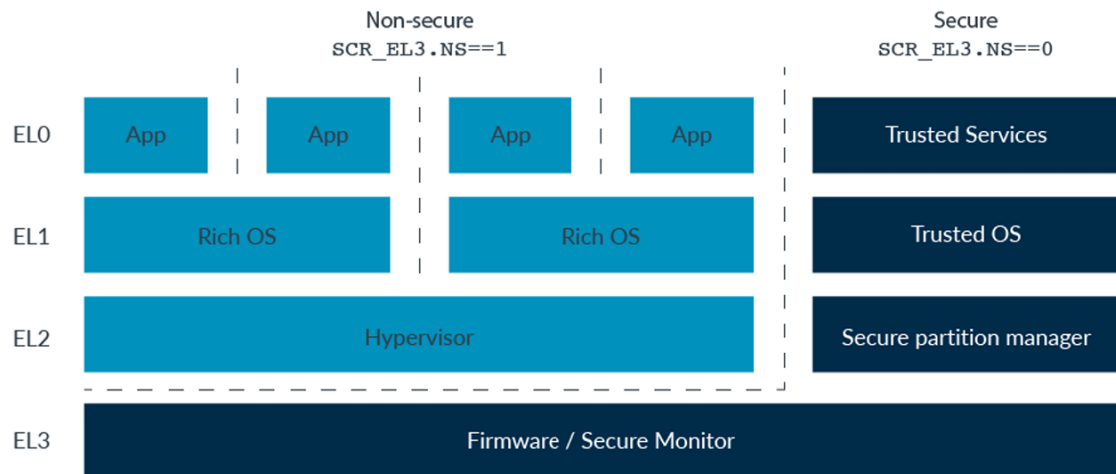


Figure 3-11 : Secure and Non-secure states

3.8.2.3 Switching between Secure States

To change Security state, in either direction, execution must pass through EL3, as shown in the following diagram:

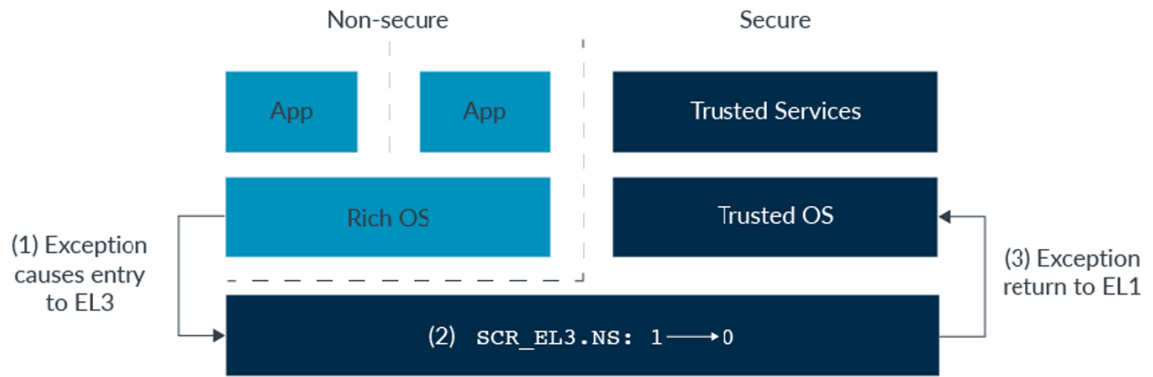


Figure 3-12 : Switching between states

The preceding diagram shows an example sequence of the steps involved in moving between Security states. Taking this one step at a time:

- Entering a higher Exception level requires an exception. Typically, this exception would be a FIQ or an SMC (Secure Monitor Call) exception.
- EL3 is entered at the appropriate exception vector. Software that is running in EL3 toggles the SCR_EL3.NS bit.
- An exception return then takes the processor from EL3 to S.EL1.

There is more to changing Security state than just moving between the Exception levels and changing the SCR_EL3.NS bit. We also must consider the processor state.

There is only one copy of the vector registers, the general-purpose registers, and most System registers. When moving between Security states it is the responsibility of software, not hardware, to save and restore register state. By convention, the piece of software that does this is called the **Secure Monitor**. This makes our earlier example look more like what you can see in the following diagram:

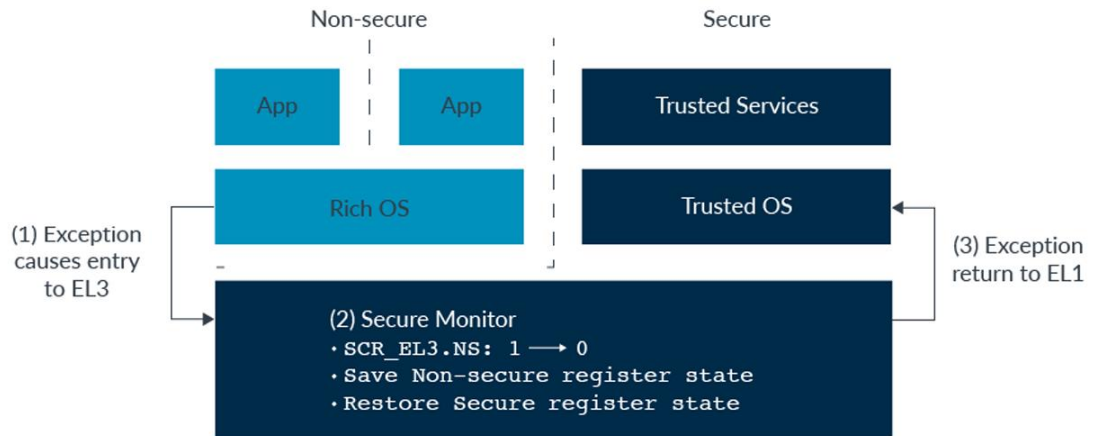


Figure 3-13 : Secure Monitor

Trusted Firmware, an open-source project that Arm sponsors, provides a reference implementation of a Secure Monitor. This is referred to as Arm Trusted Firmware (ATF).

3.8.3 Arm Trusted Firmware (ATF), Boot Loader Stage 3-1 (BL31)

The ARM Trusted Firmware implements a subset of the Trusted Board Boot Requirements (TBBR) Platform Design Document (PDD) for ARM reference platforms. The TBB sequence starts when the platform is powered on and runs up to the stage where it hands-off control to firmware running in the normal world in DRAM. This is the cold boot path.

The ARM Trusted Firmware also implements the Power State Coordination Interface ([PSCI](#)) as a runtime service. PSCI is the interface from normal world software to firmware implementing power management use-cases (for example, secondary CPU boot, hotplug and idle). Normal world software can access ARM Trusted Firmware runtime services via the ARM SMC (Secure Monitor Call) instruction.

The ARM Trusted Firmware implements a framework for configuring and managing interrupts generated in either security state.

BL31 executes solely in trusted memory. The functionality implemented by BL31 is as follows.

3.8.3.1 Architectural initialization

Architectural initialization in BL31 allows override of any previous initialization done by prior boot loader firmware. BL31 creates page tables to address the first 4GB of physical address space and initializes the MMU accordingly. It initializes a buffer of frequently used pointers, called per-CPU pointer cache, in memory for faster access. Currently the per-CPU pointer cache contains only the pointer to crash stack. It then replaces the existing exception vectors with its own. BL31 exception vectors implement more elaborate support for handling SMCs since this is the only mechanism to access the runtime services implemented by BL31 (PSCI for example). BL31 checks each SMC for validity as specified by the SMC calling convention PDD before passing control to the required SMC handler routine.

3.8.3.2 Platform initialization

BL31 performs detailed platform initialization, which enables normal world software to function correctly. It initializes a UART console, which enables access to the printf family of functions in BL31. It enables the system level implementation of the generic timer through the memory mapped interface. It initializes the following:

- GICv2 initialization
- GICv3 initialization
- Power management
- Runtime services initialization

4 HPS Customization

As mentioned in section 3.4, a hardware design must be defined for the Agilex™ SoC FPGA, if you choose to take advantage of the HPS features. This section will review the HPS customizations that were chosen for the GHRD and used on the AXE5-Eagle board.

The HPS customization is implemented by using the Platform Designer tool. This tool is launched from Quartus Prime Pro. The snapshots below were captured from the Hard Processor System Intel Agilex 5 IP component.

The Figure below shows the different sections of the HPS that are customized in the GHRD.

- FPGA to SDRAM (F2SDRAM) bridge
- FPGA to HPS (F2H) bridge
- HPS and Lightweight HPS to FPGA (LWH2F) bridge
- MPU Cluster
- HPS peripherals

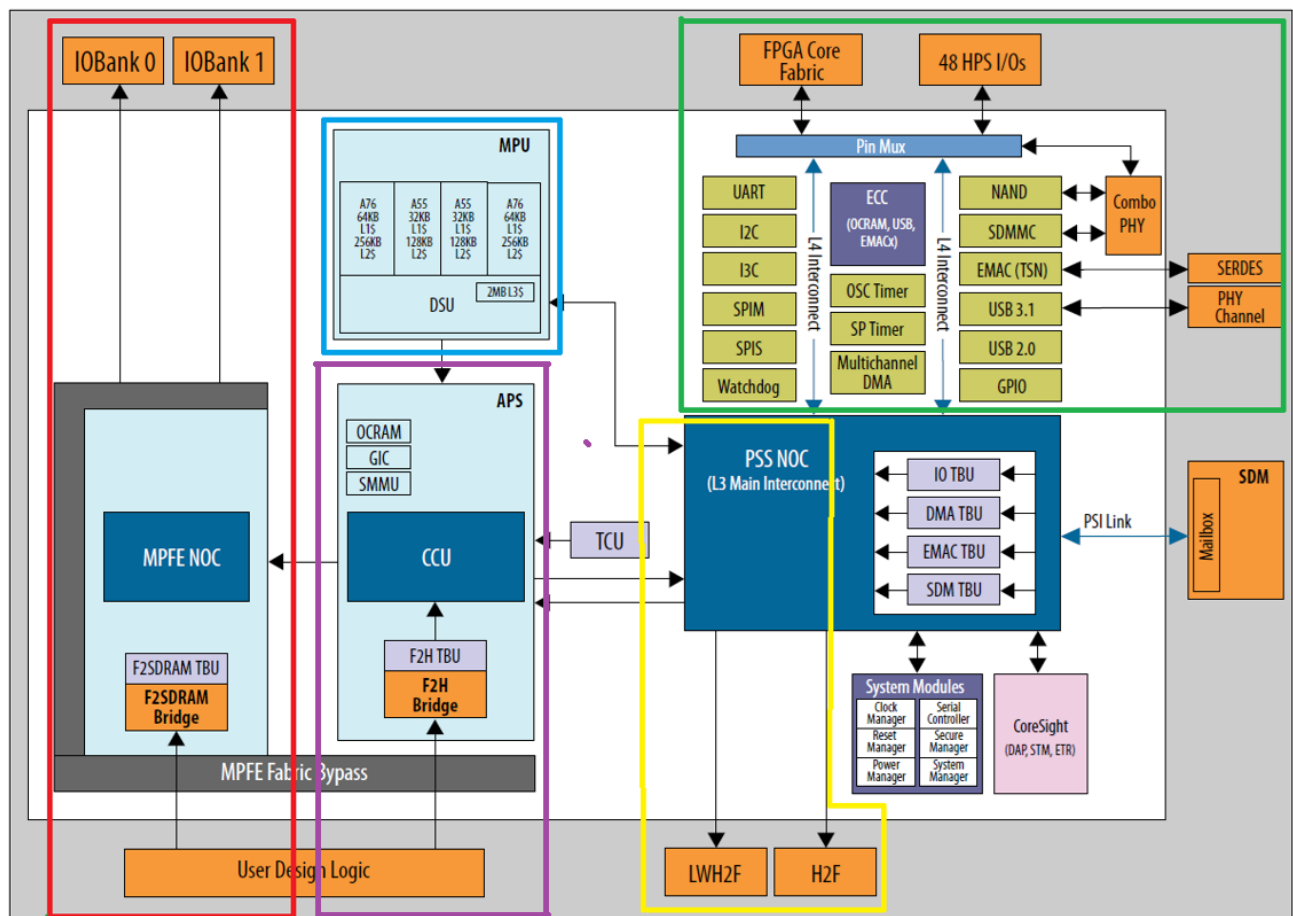


Figure 4-1 : HPS customization

Customization is implemented within the Hard Processor System Intel Agilex 5 FPGA IP Graphical User Interface seen in the figure below.

The HPS FPGA Interfaces tab is used to customize the bridges. The HPS Clocks, Resets, Power tab is used to determine which Arm cores are powered on and at which frequencies they operate. The Pin Mux and Peripherals tab is used to select which peripherals are to be used and whether they will be routed to the HPS IO banks or to FPGA IO banks.

Hard Processor System Intel Agilex 5 FPGA IP
intel_agilex_5_soc

HPS FPGA Interfaces
SDRAM
HPS Clocks, Resets, Power
IO Delays
Pin Mux and Peripherals

General

☐ Enable MPU standby & event signals
☒ Enable GP signals
☐ Enable debug APB interface
☐ Enable STM HW events
☐ Enable SWJ-DP JTAG interface
☐ Enable CTI interface
☐ Enable AMBA Trace Bus

HPS FPGA Bridges

FPGA to HPS Subordinate

Interface Specification: AXI-4
Enable/Data Width: Unused
Interface Address Width: 31-bit 2GB
☐ Enable System MMU

FPGA to SDRAM Subordinate

Enable/Data Width: 256-bit
Interface Address Width: 32-bit 4GB

HPS to FPGA Manager

Enable/Data Width: 32-bit
Interface Address Width: 30-bit 1GB

Lightweight HPS to FPGA Manager

Enable/Data Width: 32-bit
Interface Address Width: 29-bit 512 MB

Figure 4-2 : HPS Customization

4.1 HPS FPGA Interfaces

The bridges between the FPGA and the HPS allow

- HPS master access to FPGA peripherals (H2F and LWH2F)
- FPGA access to the full HPS address map including coherent access to HPS SDRAM (F2H)
- FPGA non-coherent access to HPS SDRAM (F2SDRAM)

It is useful to view the HPS system address map from the FPGA and the HPS perspectives before reviewing the bridge customization.

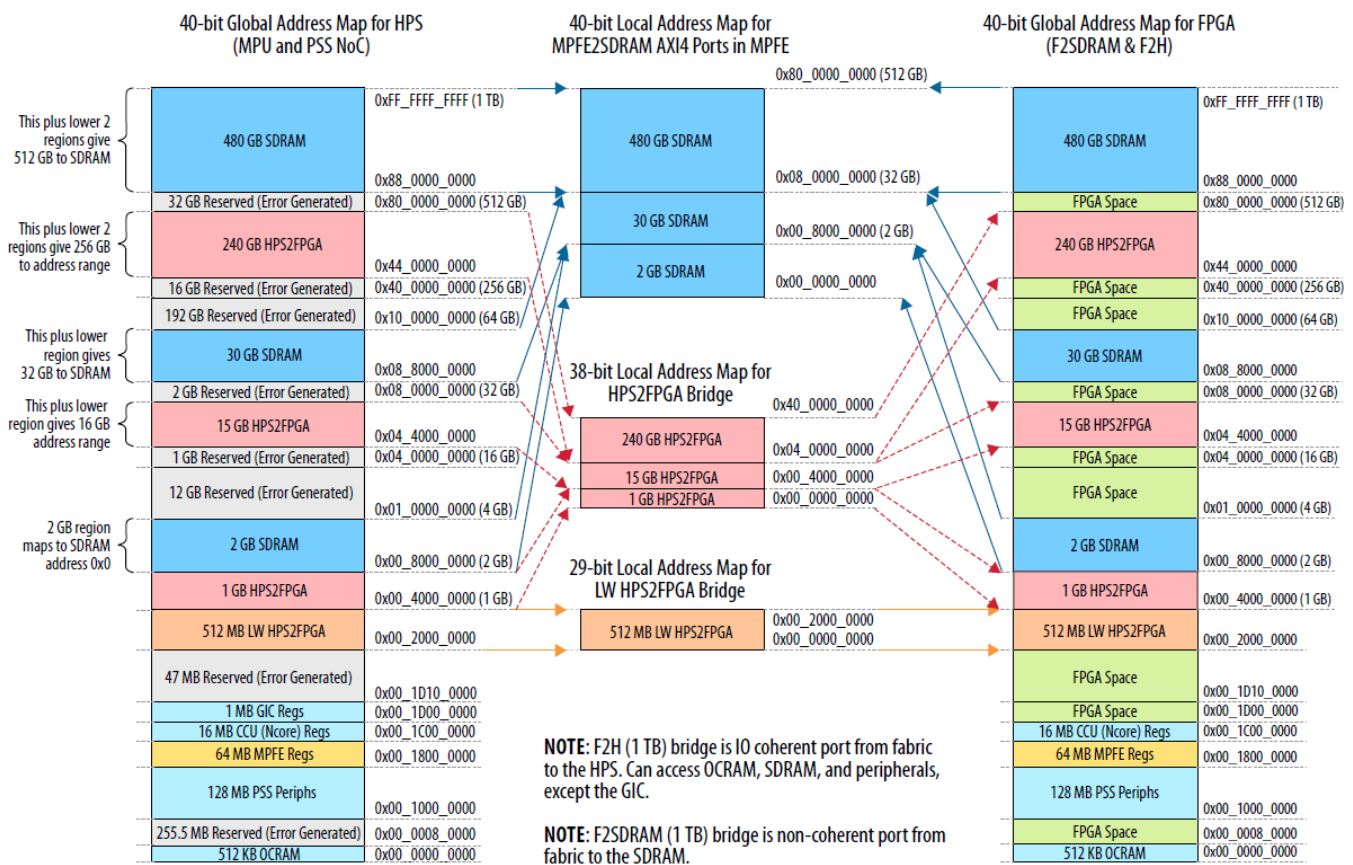


Figure 4-3 : Total System Address Map

The address map on the left represents the view as seen by HPS masters. The map on the right represents the view as seen by FPGA masters. The map in the middle shows the respective local address maps of the LWH2F, H2F and MPFE2SDRAM bridges.

4.1.1 FPGA to SDRAM Bridge (F2SDRAM)

The 40 Bit Global address map for the FPGA provides three separate address map locations for the SDRAM. The first is located at 0x00 8000 0000 and has an address span of 2GB. The next is located at 0x08 8000 0000 and has an address span of 30GB. The last is located at 0x88 8000 0000 and has an address span of 480GB. Used together these three regions provide for a total of 512 GB SDRAM. The Eagle board is populated with 1GB of LPDDR4 SDRAM connected to the HPS EMIF controller. This is mapped in the 40 Bit Global address map in the 2GB to 4GB space.

The Interface Address Width can be set anywhere from 20 to 38 bits. The Data Width can be 32, 64 or 128 bits wide. To provide access to the entire range of the LPDDR4 from the fabric, the Interface Address Width is set to 32 bits. This provides the required map range of 0 to 4GB. The data width is set to 256 bits for maximum bandwidth.

FPGA to SDRAM Subordinate	
Enable/Data Width:	256-bit ▼
Interface Address Width:	32-bit 4GB ▼
	40-bit 1TB
	39-bit 512GB
	38-bit 256GB
	37-bit 128GB
	36-bit 64GB
	35-bit 32GB
	34-bit 16GB
	33-bit 8GB
	32-bit 4GB
	31-bit 2GB
	30-bit 1GB
	29-bit 512 MB
	28-bit 256 MB
	27-bit 128 MB
	26-bit 64 MB
	25-bit 32 MB
	24-bit 16 MB
	23-bit 8 MB
	22-bit 4 MB
	21-bit 2 MB
	20-bit 1 MB

FPGA to SDRAM Subordinate	
Enable/Data Width:	256-bit ▼
Interface Address Width:	Unused
	64-bit
	128-bit
	256-bit

Figure 4-4 : F2SDRAM Customization

4.1.2 Lightweight HPS to FPGA Bridge (LWH2F)

This bridge's base address is mapped at one fixed location in the 40 Bit Global address map for the HPS (0x00 2000 0000). It has a maximum address span of 29 bits (512 MB). It has a fixed data width of 32 bits. The GHRD uses the entire 29-bit span.

The bridge is intended to be used as a low latency, control plane interface from the HPS to the FPGA.

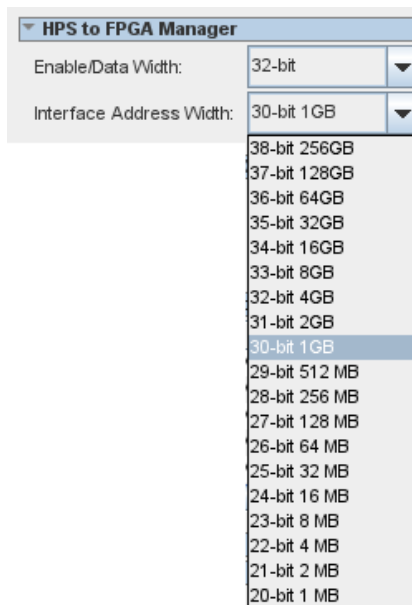
▼ Lightweight HPS to FPGA Manager	
Enable/Data Width:	32-bit ▼
Interface Address Width:	29-bit 512 MB ▼
	29-bit 512 MB
	28-bit 256 MB
	27-bit 128 MB
	26-bit 64 MB
	25-bit 32 MB
	24-bit 16 MB
	23-bit 8 MB
	22-bit 4 MB
	21-bit 2 MB
	20-bit 1 MB

Figure 4-5 : LWHPS2FPGA Customization

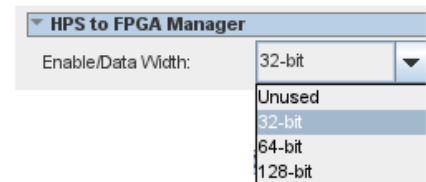
4.1.3 HPS to FPGA Bridge (H2F)

The 40 Bit Global address map for the HPS provides three separate address map locations for the H2F bridge. The first is located at 0x00 4000 0000 and has an address span of 1GB. The next is located at 0x04 4000 0000 and has an address span of 15GB. The last is located at 0x44 4000 0000 and has an address span of 240GB. The local address map for this bridge presents a contiguous 38 bit address span (256 GB).

The Interface Address Width can be set anywhere from 20 to 38 bits. The Data Width can be 32, 64 or 128 bits wide. The GHRD currently has selected a 1GB address span and 32 data width. The GHRD has the H2F bridge connected to the FPGA EMIF subsystem. The Eagle board is populated with 1GB of LPDDR4 SDRAM connected to the FPGA EMIF controller.



The screenshot shows the 'HPS to FPGA Manager' configuration window. The 'Enable/Data Width' dropdown is set to '32-bit'. The 'Interface Address Width' dropdown is open, showing a list of options from '38-bit 256GB' down to '20-bit 1 MB'. The '30-bit 1GB' option is currently selected and highlighted in blue.



The screenshot shows the 'HPS to FPGA Manager' configuration window. The 'Enable/Data Width' dropdown is open, showing a list of options: 'Unused', '32-bit', '64-bit', and '128-bit'. The '32-bit' option is currently selected and highlighted in blue.

Figure 4-6 : HPS2FPGA Customization

4.1.4 FPGA to HPS Bridge (F2H)

F2H bridge provides a way for initiators (IPs, accelerators) in the fabric to access HPS peripherals, which makes the peripherals extensions of the HPS system. The accelerator coordinates with the HPS MPU via mailboxes or semaphores in HPS memory, or via various interrupts and GPIOs exposed by the HPS to the fabric. Typical use cases involve the HPS MPU preparing space in memory for fabric accelerators to use, then allowing the accelerator to move and process large amounts of data in HPS memory. The HPS MPU can perform control functions such as inspecting headers in large streams of data to determine the next action and coordinating data movement among multiple fabric accelerators. This bridge places

fabric initiators in the same hierarchy as the MPU in the HPS subsystem. F2H bridge supports IO cache coherency with the HPS MPU caches; fabric transactions can snoop the MPU caches, but the MPU caches cannot snoop activity in the fabric. In addition, using ACE-lite, the F2H bridge goes through the system memory management unit (SMMU). This allows fabric initiators to use the same virtual memory view as the MPU.

4.2 HPS Clocks, Resets, Power

This section allows the designer the ability to customize the input clock frequency for the HPS, the Phase Lock Loop output (PLL) frequencies and which Arm cores are powered on.

4.2.1 Input Clocks

The Eagle board connects a 25 MHz external oscillator to the HPS EOSC pin.

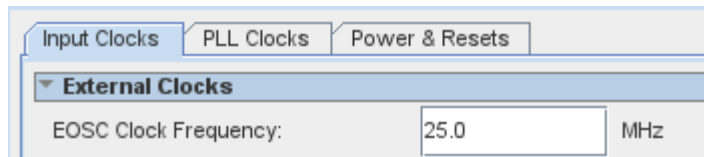


Figure 4-7 : Input Clocks

4.2.2 PLL Clocks

The individual frequencies that each Arm core operates at can be specified. Cores 0 and 1 share the same frequency.

Input Clocks PLL Clocks Power & Resets		
MPU Clocks		
Clocks	Frequencies (Mhz)	Sources
MPU	500.00	MainC2
MPU CCU	250.0	MPU
MPU Periph	125.0	MPU
Core0 Core1	875.00	PeriphC0
Core2	800.00	MainC0
Core3	800.00	MainC0

Figure 4-8 : PLL Clocks

4.2.3 Power & Resets

The user can configure which Arm cores are powered on. Cores 0 and 1 share the same selection.



Figure 4-9 : Power Configurations

4.3 Pin Mux and Peripherals

The **Auto-Place IP** tab contains a list of HPS peripherals that can be enabled and either routed to the HPS I/O or to the FPGA.

Auto-Place IP feature assists you to easily select the desired peripherals and have the tool automatically place those peripherals either among the 48 dedicated HPS I/O or on the FPGA if available.

You can enable the following types of peripherals:

- SD/MMC Controller
- USB 2.0 OTG Controller (USB0)
- USB 3.1 Gen1 Controller (USB1)
- Ethernet Media Access Controller
- SPI Master
- SPI Slave
- UART Controller
- I2C Controller
- I3C Controller
- NAND Flash Controller
- CoreSight Debug and Trace
- GPIO

The HPS has 48 dedicated IOs; therefore, not all peripherals can fit in the HPS I/O.

4.3.1 Advanced IP Placement

The Advanced IP Placement tab allows you to be more specific about the placement of each peripheral pin in the HPS dedicated I/O quadrant space. Each location has a pulldown selection menu where you can select which peripheral I/O to be routed to the pin location. Each pulldown menu corresponds to the inputs available to the Pin Mux at that location.

Pin Mux GUI

Auto-Place IP | **Advanced**

Advanced IP Placement | Advanced FPGA Placement

I/O Selections

I/O	IP Selection	I/O	IP Selection	I/O	IP Selection	I/O	IP Selection
HPS_I/OA_1	NONE	HPS_I/OA_13	USB1:CLK	HPS_I/OB_1	SDMMC:DATA0	HPS_I/OB_13	EMAC2:TX_CLK
HPS_I/OA_2	GPIO0:IO1	HPS_I/OA_14	USB1:STP	HPS_I/OB_2	SDMMC:DATA1	HPS_I/OB_14	EMAC2:TX_CTL
HPS_I/OA_3	UART0:TX	HPS_I/OA_15	USB1:DIR	HPS_I/OB_3	SDMMC:CCLK	HPS_I/OB_15	EMAC2:RX_CLK
HPS_I/OA_4	UART0:RX	HPS_I/OA_16	USB1:DATA0	HPS_I/OB_4	NONE	HPS_I/OB_16	EMAC2:RX_CTL
HPS_I/OA_5	I2C0:SDA	HPS_I/OA_17	USB1:DATA1	HPS_I/OB_5	GPIO1:IO4	HPS_I/OB_17	EMAC2:TXD0
HPS_I/OA_6	I2C0:SCL	HPS_I/OA_18	USB1:NXT	HPS_I/OB_6	SDMMC:DATA2	HPS_I/OB_18	EMAC2:TXD1
HPS_I/OA_7	GPIO0:IO6	HPS_I/OA_19	USB1:DATA2	HPS_I/OB_7	SDMMC:DATA3	HPS_I/OB_19	EMAC2:RXD0
HPS_I/OA_8	GPIO0:IO7	HPS_I/OA_20	USB1:DATA3	HPS_I/OB_8	SDMMC:CMD	HPS_I/OB_20	EMAC2:RXD1
HPS_I/OA_9	GPIO0:IO8	HPS_I/OA_21	USB1:DATA4	HPS_I/OB_9	MDIO2:MDIO	HPS_I/OB_21	EMAC2:TXD2
HPS_I/OA_10	GPIO0:IO9	HPS_I/OA_22	USB1:DATA5	HPS_I/OB_10	MDIO2:MDC	HPS_I/OB_22	EMAC2:TXD3
HPS_I/OA_11	GPIO0:IO10	HPS_I/OA_23	USB1:DATA6	HPS_I/OB_11	GPIO1:IO10	HPS_I/OB_23	EMAC2:RXD2
HPS_I/OA_12	HCLK:HPS_OSC_CLK	HPS_I/OA_24	USB1:DATA7	HPS_I/OB_12	GPIO1:IO11	HPS_I/OB_24	EMAC2:RXD3

Figure 4-10 : AXE5-Eagle board Advanced IP Placement

Figure 4-11 below shows the corresponding wiring, based on the Advanced IP Placement, on the AXE5-Eagle board, for the HPS IO bank.

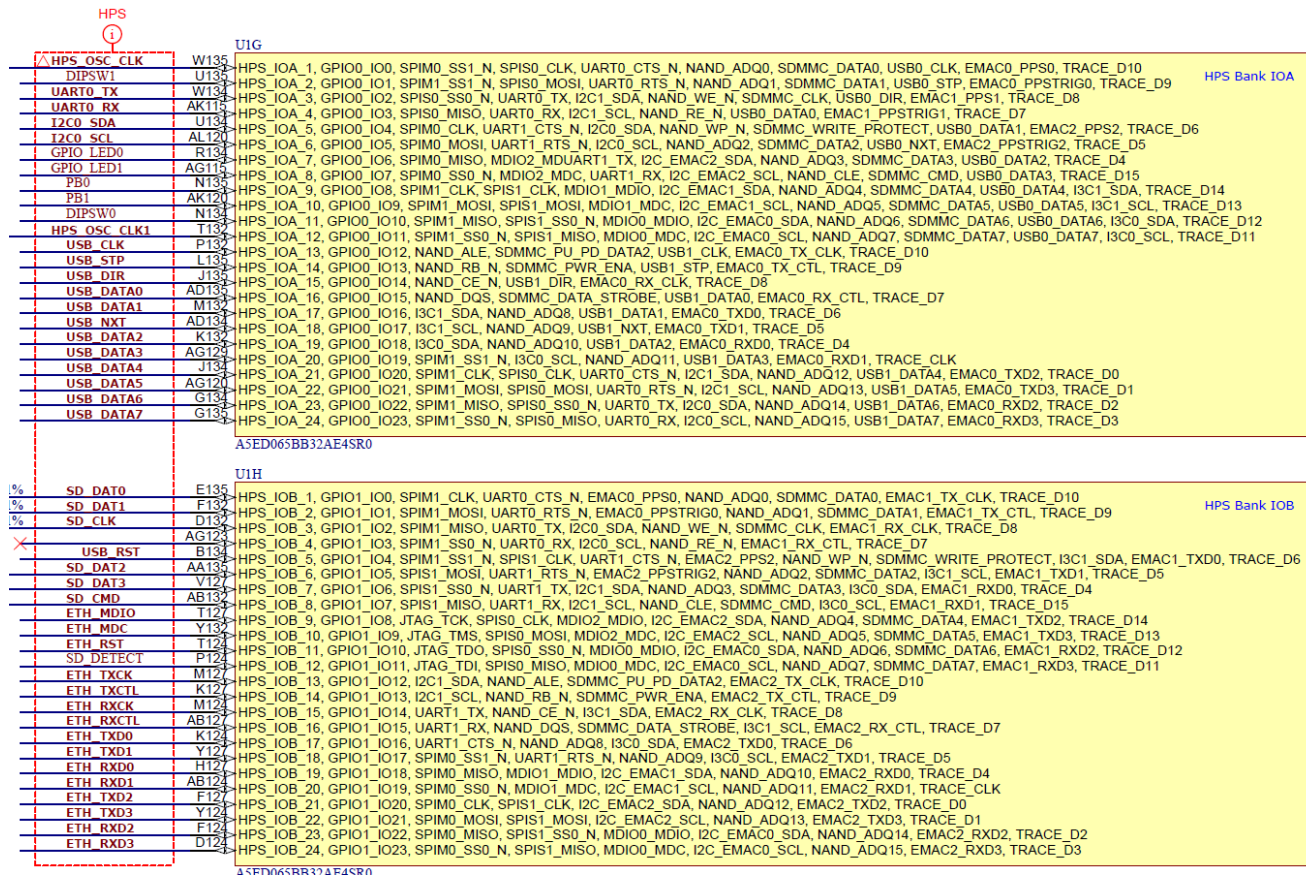


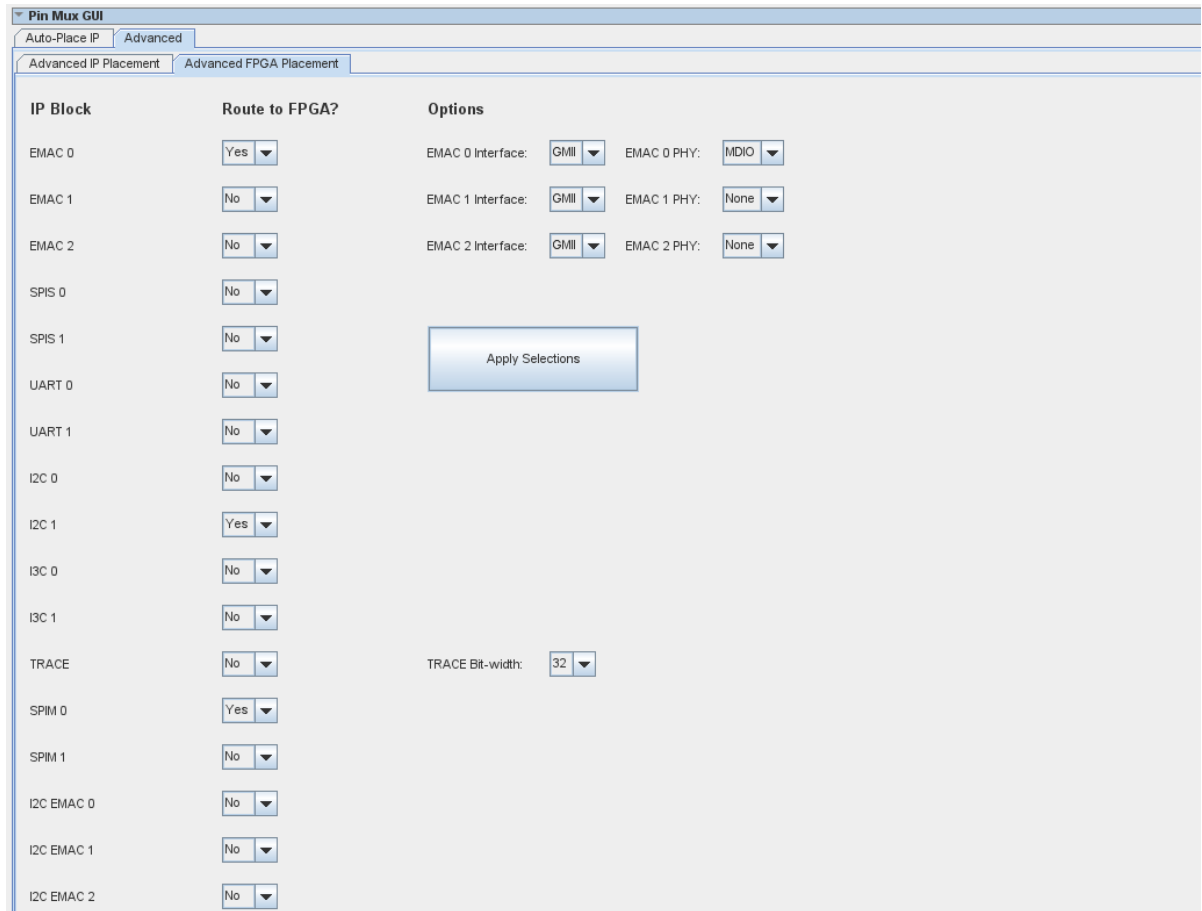
Figure 4-11 : AXE5-Eagle board schematic wiring for Advanced IP Placement

The Advanced FPGA Placement tab allows you to route specific peripherals to the FPGA.

Figure 5-12 below shows the specific peripherals that are routed to FPGA I/O banks. They are:

- EMAC0
- I2C1
- SPIM0

4.3.2 Advanced FPGA Placement



IP Block	Route to FPGA?	Options
EMAC 0	Yes	EMAC 0 Interface: GMI, EMAC 0 PHY: MDIO
EMAC 1	No	EMAC 1 Interface: GMI, EMAC 1 PHY: None
EMAC 2	No	EMAC 2 Interface: GMI, EMAC 2 PHY: None
SPIS 0	No	Apply Selections
SPIS 1	No	
UART 0	No	
UART 1	No	
I2C 0	No	
I2C 1	Yes	TRACE Bit-width: 32
I3C 0	No	
I3C 1	No	
TRACE	No	
SPIM 0	Yes	
SPIM 1	No	
I2C EMAC 0	No	
I2C EMAC 1	No	
I2C EMAC 2	No	

Figure 4-12 : AXE5-Eagle board Advanced FPGA Placement

The HPS customization information is compiled into the configuration bitstream and is later used by the FSBL to configure the HPS at boot time. It is commonly referred to as the HPS handoff data.

5 Creating a Bootable Image

In section three we explored the hardware, firmware and software necessary to boot an Agilex 5 SoC FPGA from Power-on to the Linux prompt. You will now learn how to build the required software. You will also learn how to package it on appropriate boot media.

The flow for creating a Golden System Reference Design is:

- Create a Hardware project in Quartus (GHRD)
- Customize the HPS in Platform Designer
- Compile the GHRD in Quartus to generate an FPGA configuration bitstream.
- Compile the Arm Trusted Firmware (BL31)
- Compile U-boot. This creates the FSBL and the SSBL.
- Compile a U-boot boot script.
- Compile the Linux kernel
- Use the Yocto project to create a Linux root file system (rootfs)
- Package BL31, U-boot (SSBL), boot script, and the Linux kernel in the FAT partition of an SD card.
- Package the rootfs in the Linux partition of the SD card.
- Create a custom FPGA configuration image, using the Quartus Programming File Generator. This image includes the SDM firmware, the FSBL, HPS handoff data and the FPGA configuration data.
- Write the custom FPGA configuration image into QSPI flash using the Quartus Programmer.

This lab will focus on compiling the software components required to boot Linux. You will also boot the Linux image on an AXE5-Eagle board

- The GHRD is provided as a completed, pre-built, project.
- The Yocto project can take up to a few hours to build. It is provided as a completed, pre-built, project.

The GSRD flow is shown in Figure 5-1 below.

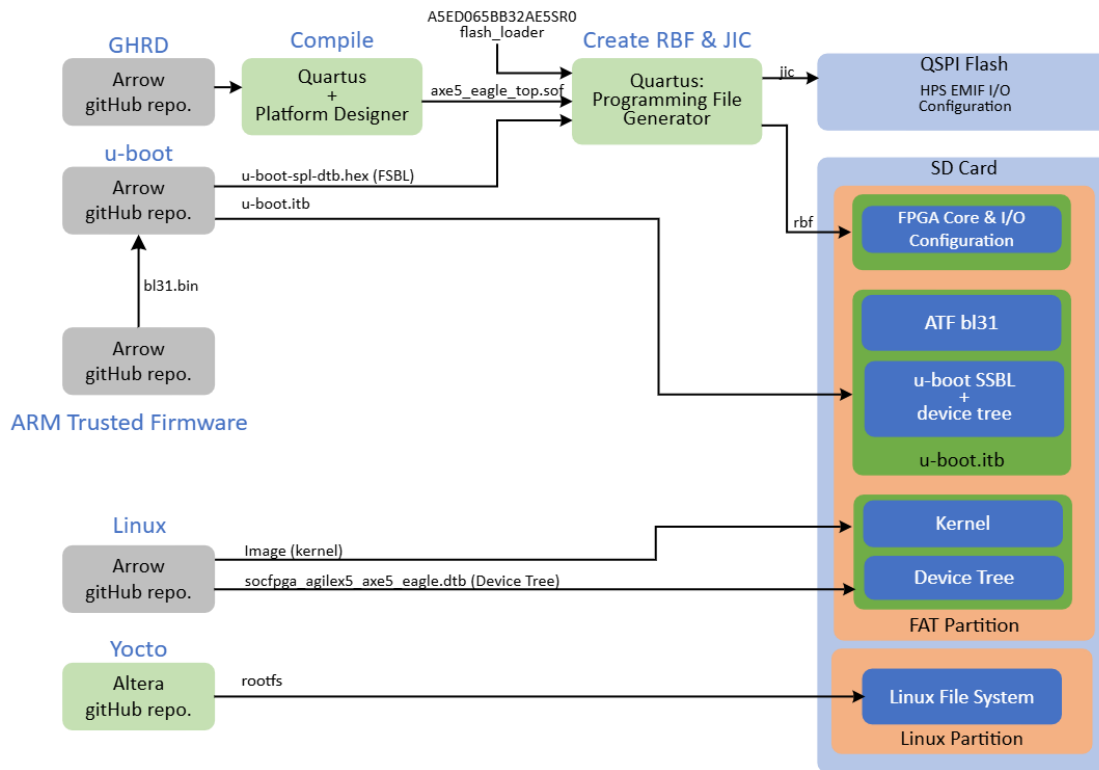


Figure 5-1 : The GSRD Build Flow

Follow the instructions in the next few sections of the Lab to create a bootable Linux image for the AXE5-Eagle board

5.1 Code repositories

Arrow hosts several repositories, required for this build, on Github. They are a mirror of the Altera repositories and are continually maintained to include the latest updates. The repositories are listed below.

- Arm Trusted Firmware – [arm-trusted-firmware](#)
- U-boot – [u-boot-socfpga](#)
- Linux – [linux_socfpga](#)

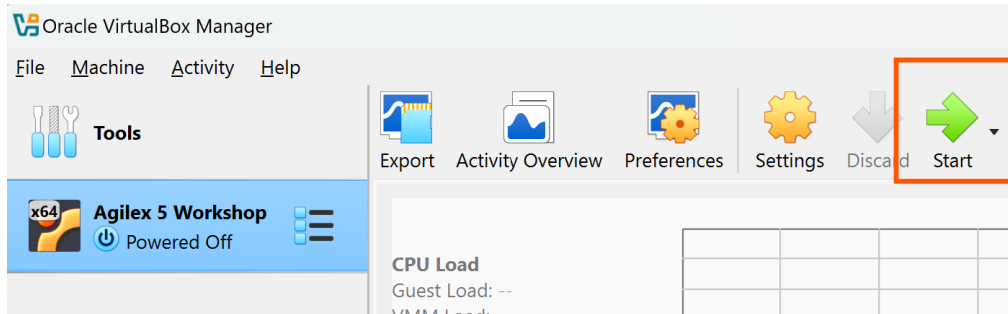
Additional repositories are hosted to support the GHRD and other reference designs

- Golden Hardware Reference Design - [ghrd-socfpga](#)
- Reference Designs – [refdes-agilex5](#)

5.2 Launch the Oracle VirtualBox Linux Virtual Machine

5.2.1 Launch the VirtualBox based Ubuntu 22.04 LTS Virtual Machine.

Select the **Agilex 5 Workshop** and then press the **Start** button

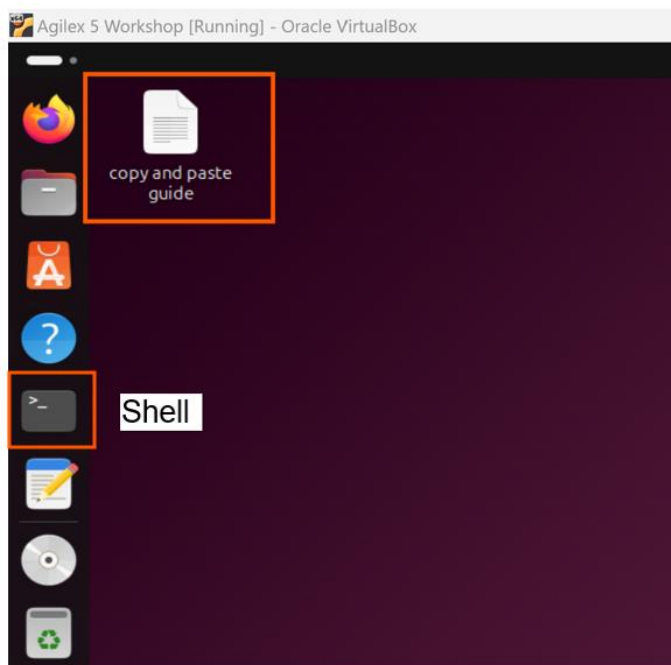


5.2.2 Copy and paste guide

A copy and paste guide is provided for convenience. These commands can be pasted into the Linux shell as directed in the Lab guide.

The following shortcuts can be used

Copy	Ctrl + C
Paste	Shift + Ctrl + V

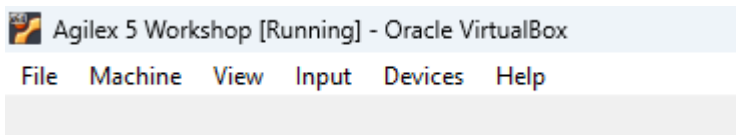


5.2.3 Host password

The password for the VirtualBox VM is **root**. This will be required by the super user command `sudo`.

5.2.4 VirtualBox Menu Bar

A menu bar is visible at the top of the VirtualBox screen. This will be referred to later in the lab document for access to specific functions.



5.3 Setup the build environment

Follow the instructions listed below to setup the build environment. First, open a shell. The instructions can be copied and pasted from the copy and paste guide, a line at a time, into the shell and executed. The instructions do the following:

- Define a TOP_FOLDER

```
$ cd agilex_5
$ export TOP_FOLDER=`pwd`
```

5.4 Arm Trusted Firmware (BL31)

The Arm Trusted Firmware code has not been modified for the AXE5-Eagle board. The code is a mirror image of the altera repository and can be compiled as is.

Follow the instructions listed below to clone the source code and complete the compilation. The instructions can be copied and pasted from the copy and paste guide, a line at a time, into the shell and executed. The instructions do the following:

- Clone the Arrow Arm Trusted Firmware repository
- Compile

```
$ git clone -b QPDS25.1_REL_GSRD_PR https://github.com/ArrowElectronics/arm-trusted-firmware arm-trusted-firmware
$ cd arm-trusted-firmware
$ make -j 48 PLAT=agilex5 bl31
```

The following file is created:

- \$TOP_FOLDER/arm-trusted-firmware/build/agilex5/release/bl31.bin

The build flow for ARM Trusted Firmware is represented in the figure below.

Arm Trusted Firmware

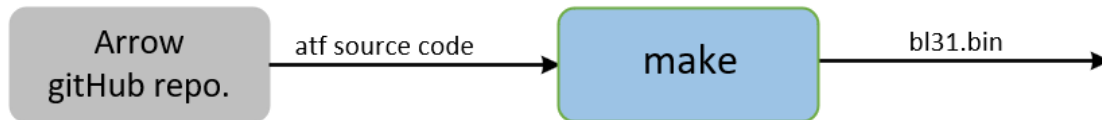


Figure 5-2 : Arm Trusted Firmware Build flow

5.5 U-boot

The build for the AXE5-Eagle board utilizes two distinct customization sources.

- U-boot build configuration (defconfig)
- devicetree

5.5.1 defconfig

U-boot utilizes the same Kconfig, Kbuild build configuration system as Linux. The build configuration is specified in a defconfig file. This determines which features and hardware support code are built into the U-boot executable.

The defconfig for the AXE5-Eagle board. Is comprised of a combination of a standard agilex 5 defconfig and a custom config fragment.

- [socfpga_agilex5_defconfig](#)
- [config-fragment-eagle](#)

5.5.2 Devicetree

The devicetrees are used to define dynamic boot selections at boot time. They define which drivers to load and which peripherals to enable.

Devicetrees for U-boot have been customized for the AXE5-Eagle board.

- [socfpga_agilex5_axe5_eagle-u-boot.dtsi](#)
- [socfpga_agilex5_axe5_eagle.dts](#)

The same set of Devicetree source files are used for FSBL and SSBL. The U-Boot devicetree is filtered by the fdtgrep tools during the build process to generate a much smaller device tree used in the FSBL.

5.5.3 Compile U-boot

Follow the instructions listed below to clone the source code and complete the compilation. The instructions can be copied and pasted from the copy and paste guide, a line at a time, into the shell and executed. The instructions do the following:

- Clone the source.
- Clean the code base of any remnants using the mrproper option. **Fun fact:** The name mrproper is derived from a reference to the cleaning product [Mr. Clean](#), which translates into Mr. Proper in other languages.
- Use the AXE5-Eagle board build configuration, socfpga_agilex5_axe5_eagle_defconfig
- Create a link to the ATF bl31 binary file. This will be packaged into the u-boot.itb image.

```
$ cd $TOP_FOLDER
$ rm -rf u-boot-socfpga
$ git clone -b QPDS25.1_REL_GSRD_PR https://github.com/ArrowElectronics/u-boot-socfpga
u-boot-socfpga
$ cd u-boot-socfpga
```

- Only boot from SD, do not try QSPI and NAND

```
$ sed -i 's/u-boot,spl-boot-order.*/u-boot\spl-boot-order = \&mmc;/g'
arch/arm/dts/socfpga_agilex5_axe5_eagle-u-boot.dtsi
```

- Disable NAND in the device tree

```
$ sed -i '/&nand {/!b;n;c\tstatus = "disabled";' arch/arm/dts/socfpga_agilex5_axe5_eagle-u-
boot.dtsi
```

- Link to ATF

```
$ ln -s ../arm-trusted-firmware/build/agilex5/release/bl31.bin
```

- Clean the build

```
$ make clean && make mrproper
```

- Create the defconfig. Combine with the custom config fragment

```
$ make socfpga_agilex5_defconfig
```

```
$ ./scripts/kconfig/merge_config.sh -O . -m .config config-fragment-eagle
```

- Compile to generate the devicetree, FSBL & SSBL
- Package the ATF bl31 binary, devicetree and SSBL into u-boot.itb

```
$ make -j 64
```

The following files are created:

- \$TOP_FOLDER/u-boot-socfpga/u-boot.itb (SSBL)
- \$TOP_FOLDER/u-boot-socfpga/spl/u-boot-spl-dtb.hex (FSBL)

Detailed build flow for U-boot is represented in the figure below.

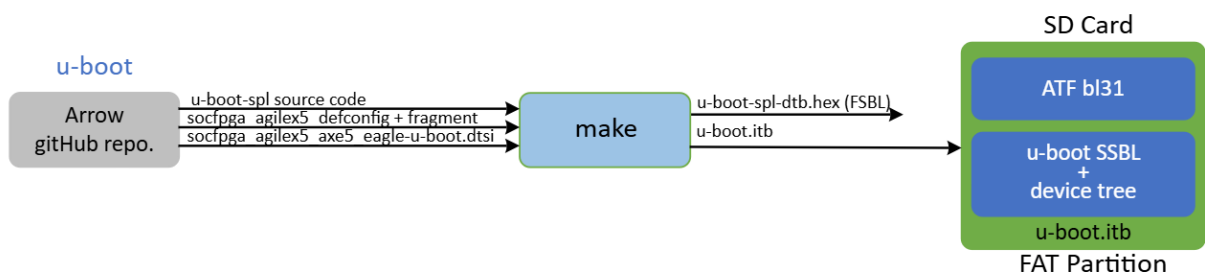


Figure 5-4 : U-boot (SSBL) Build flow

5.6 Linux

The build for the AXE5-Eagle board utilizes two distinct customization sources.

- Linux kernel build configuration (defconfig)
- Devicetree source (dts)

5.6.1 defconfig

The Linux kernel utilizes the Kconfig, Kbuild build configuration system. The build configuration is specified in a defconfig file. This determines which features and hardware support code are built into the Linux kernel.

The defconfig for the AXE5-Eagle board. Is comprised of a combination of a standard linux arch64 defconfig and a custom config fragment.

- [defconfig](#)
- [config-fragment-eagle](#)

5.6.2 Devicetree

The devicetree is used to define dynamic boot selections at boot time. They define which drivers to load and which peripherals to enable. The devicetree also defines the location of the peripherals in the memory map and their associated drivers.

The Devicetree listed below has been customized for the AXE5-Eagle board.

- [socfpga_agilex5_axe5_eagle.dts](#)

5.6.3 Compile Linux

Follow the instructions listed below to clone the source code and complete the compilation. This will take approximately 25 minutes to download and compile. The instructions can be copied and pasted from the copy and paste guide, a line at a time, into the shell and executed. The instructions do the following:

- Clone the source.
- Create the AXE5-Eagle board build configuration, socfpga_agilex5_axe5_eagle_defconfig to compile the Linux kernel
- Compile the Linux kernel and the devicetree

- Setup the build environment and clone the linux source code

```
$ cd $TOP_FOLDER
$ rm -rf linux-socfpga
$ git clone -b QPDS25.1_REL_GSRD_PR https://github.com/ArrowElectronics/linux-socfpga
linux-socfpga
$ cd linux-socfpga
```

- Add arrow dts folder as Makefile option

```
$ sed -i '$ a subdir-y += arrow' arch/arm64/boot/dts/Makefile
```

- Merge configuration fragment with the default defconfig

```
$ make defconfig
$ ./scripts/kconfig/merge_config.sh -O ./ ./config ./config-fragment-eagle
```

- Compile the Linux kernel and the devicetree

```
$ make -j 64 Image && make arrow/socfpga_agilex5_axe5_eagle.dtb
```

The following files are created:

- \$TOP_FOLDER/linux-socfpga/arch/arm64/boot/Image
- \$TOP_FOLDER/linux-socfpga/arch/arm64/boot/dts/arrow/socfpga_agilex5_axe5_eagle.dtb

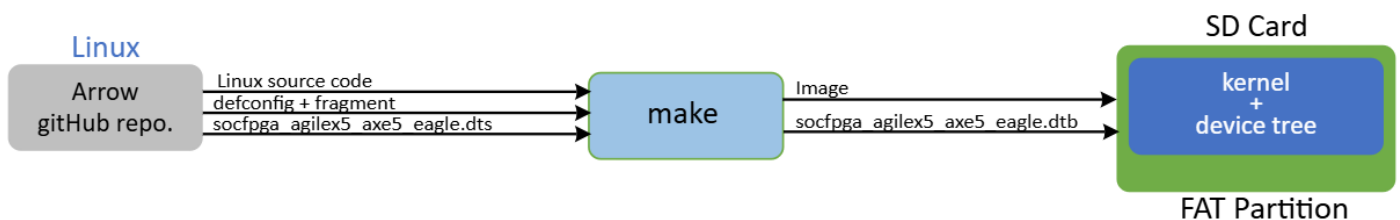


Figure 5-5 : Linux Build flow

5.7 Create a Linux Root File System (rootfs) with Yocto

The Yocto build requirements and flow are described on the Rocketboards site on the [“Building Bootloader for Agilex 5”](#) page. This can take a number of hours to complete.

The compiled rootfs image is provided for this lab..

The file is located at :

- \$TOP_FOLDER/yocto/build/tmp/deploy/images/agilex5/core-image-minimal-agilex5.rootfs.tar.gz

5.8 Create the FPGA Configuration Bitstream Images

The HPS boot first mode requires that two configuration files are generated. The first contains the HPS EMIF I/O configuration information. The SDM uses this to setup the HPS memory controller I/O before the FSBL starts running. This file is combined with the FSBL and stored as a JIC file in QSPI flash. It can also be created as an SRAM Object File (SOF) and downloaded via JTAG. This is convenient during development.

The second file is produced as a Raw Binary File (RBF) and contains the FPGA core logic and I/O configuration information. The RBF is placed on the FAT partition of the SD card. It is read during the SSBL operation and written to the FPGA.

5.8.1 Create the HPS EMIF I/O Configuration SOF file

Create a custom HPS EMIF I/O configuration image, using the Quartus Programming File Generator. This image includes the FSBL, HPS handoff data and the FPGA configuration data. It is generated as a SOF file (SRAM Object File) and is downloaded into the FPGA. This will trigger an FPGA reconfiguration which in turn will begin the process of booting the HPS.

The instructions can be copied and pasted from the copy and paste guide, a line at a time, into the shell and executed. Include the line below when cutting and pasting. The instructions do the following:

- cd to the GHRD output_files subdirectory
- Create the integrated SOF file

```
$ cd $TOP_FOLDER/ghrd-socfpga/axe5_eagle_ghrd/output_files/
$ quartus_pfg -c axe5_eagle_top.sof axe5_eagle_top_hps.sof -o hps_path=$TOP_FOLDER/u-
boot-socfpga/spl/u-boot-spl-dtb.hex
```

The following Quartus Programming File Generation option is used

- -o hps_path points to the location of the FSBL

The following file is generated :

- \$TOP_FOLDER/ghrd-socfpga/axe5_eagle_ghrd/output_files/axe5_eagle_top_hps.sof

5.8.2 Create the FPGA Core RBF file

Create the FPGA Core RBF image using the Quartus Programming File Generator. This image includes the FPGA Core logic and I/O configuration data. It is written to the FAT partition of the SD card.

- Create the RBF file

```
$ cd $TOP_FOLDER/ghrd-socfpga/axe5_eagle_ghrd/output_files/
$ quartus_pfg -c axe5_eagle_top.sof axe5_eagle_top.jic -o hps=on -o device=MT25QU02G -o
flash_loader=A5ED065BB32AE5SR0 -o hps_path=$TOP_FOLDER/u-boot-socfpga/spl/u-boot-spl-
dtb.hex -o mode=ASX4
```

The following Quartus Programming File Generation option is used

- -o hps_path points to the location of the FSBL

The following files are created :

- \$TOP_FOLDER/ghrd-socfpga/axe5_eagle_ghrd/output_files/axe5_eagle_top.hps.jic
- \$TOP_FOLDER/ghrd-socfpga/axe5_eagle_ghrd/output_files/axe5_eagle_top.core.rbf (**rename it to ghrd.core.rbf**)

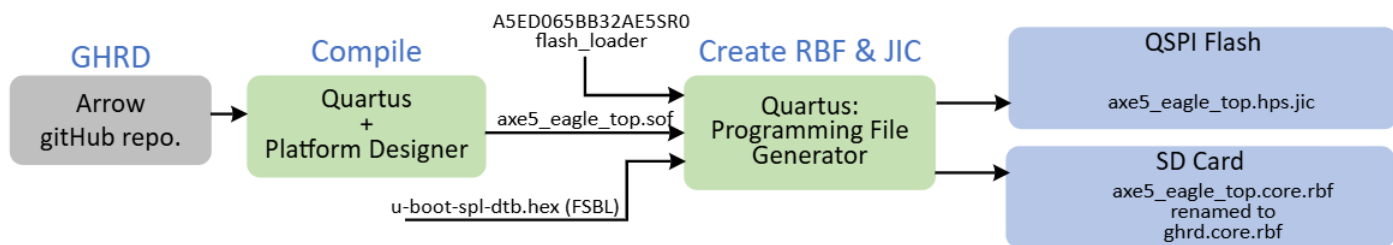


Figure 5-6 : Create the FPGA Bitstream files

5.9 Create the SD Card Image

Intel provides a python utility, *make_sdimage_p3.py*, that will create a bootable image, with partitions and content. The instructions can be copied and pasted from the copy and paste guide, a line at a time, into the shell and executed. Include the line below when cutting and pasting. The instructions do the following:

The commands below do the following

- create a directory for collecting the content for the SD card
- download the *sdimage_p3.py* utility
- copy all the files for deployment to the *fatfs* and *rootfs* directories.

```
$ cd $TOP_FOLDER
$ mkdir sd_card && cd sd_card
$ wget https://releases.rocketboards.org/release/2020.11/gsrld/tools/make_sdimage_p3.py
$ # remove mkfs.fat parameter which has some issues on Ubuntu 22.04
$ sed -i 's/\"-F 32\",//g' make_sdimage_p3.py
$ chmod +x make_sdimage_p3.py
$ mkdir fatfs && cd fatfs
$ cp $TOP_FOLDER/ghrd-socfpga/axe5_eagle_ghrd/output_files/ghrd.core.rbf .
$ cp $TOP_FOLDER/u-boot-socfpga/u-boot.itb .
$ cp $TOP_FOLDER/linux-socfpga/arch/arm64/boot/Image .
$ cp $TOP_FOLDER/linux-socfpga/arch/arm64/boot/dts/arrow/socfpga_agilex5_axe5_eagle.dtb .
$ cd ..
$ mkdir rootfs && cd rootfs
$ sudo tar xf $TOP_FOLDER/yocto/build/tmp/deploy/images/agilex5/core-image-minimal-agilex5.rootfs.tar.gz
$ cd ..
```


- Use the `sdimage_p3.py` utility to create `sdcard.img`.

```
$ sudo python3 make_sdimage_p3.py -f -P fatfs/*,num=1,format=fat32,size=512M -P
rootfs/*,num=2,format=ext3,size=512M -s 1024M -n sdcard.img
$ sudo chmod 777 sdcard.img
$ cd ..
```

The file is located at :

- `$TOP_FOLDER/sd_card/sdcard.img`

5.9.1 Write the SD Card image

The image can be written to an SD card using the `dd` command in Linux. The instructions can be copied and pasted from the windows below, a line at a time, into the shell and executed. The instructions do the following:

Caution. Care must be taken to first identify the drive letter of the SD Card. Writing an incorrect drive letter can potentially overwrite the contents of the Virtual Machines hard drive.

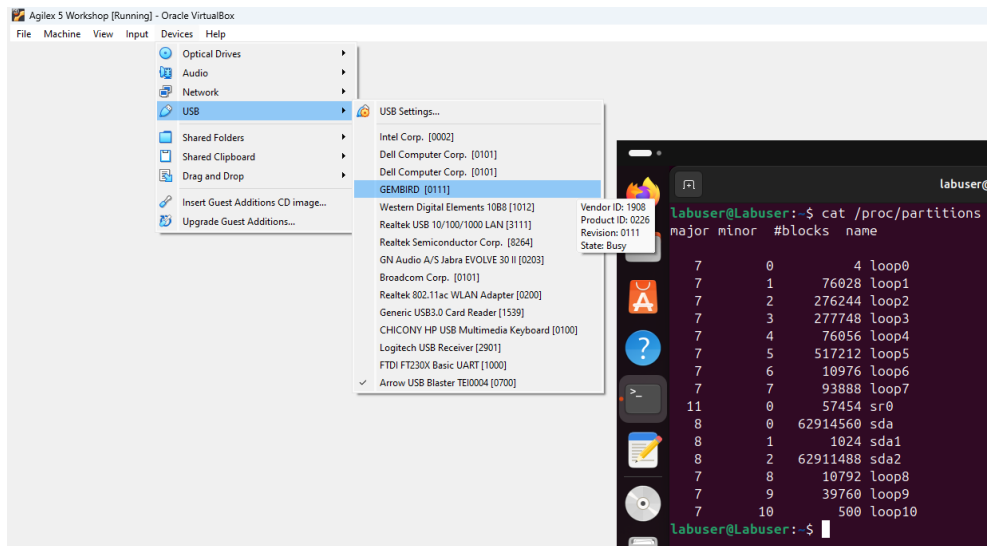
Determine the device associated with the SD card on the host. Run the command below **before and after** inserting and enabling the the SD card in the VM.

```
$ cat /proc/partitions
```

5.9.1.1 Enable the SD Card Adaptor

It is best to use a USB to micro SD card adaptor as they are recognized by VirtualBox. Use the following steps to make the adaptor available in the VM.

- On the VirtualBox menu bar press Devices → USB and then click on the adaptor.



Run the command below **again**. The new drive letter will show up as /dev/sdx/ where x represents the actual letter (a,b,c,d etc).

```
$ cat /proc/partitions
```

Use the dd utility to write the SD image to the SD card. **Substitute the letter x** with the actual drive letter discovered above. Then use the sync command to flush the changes from memory to the SD card.

```
labuser@Labuser:~$ cat /proc/partitions
major minor #blocks name

7        0          4 loop0
7        1       76028 loop1
7        2      276244 loop2
7        3      277748 loop3
7        4       76056 loop4
7        5      517212 loop5
7        6       10976 loop6
7        7       93888 loop7
11       0        57454 sr0
8        0     62914560 sda
8        1         1024 sda1
8        2     62911488 sda2
7        8        10792 loop8
7        9        39760 loop9
7       10          500 loop10
8       16     31166976 sdb
8       17        512000 sdb1
8       18       1536000 sdb2
```

```
$ sudo dd if=$TOP_FOLDER/sd_card/sdcard.img of=/dev/sdx bs=1M status=progress
```

```
$ sudo sync
```

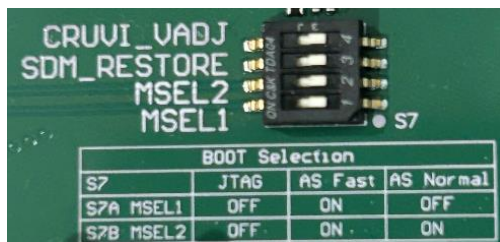
5.10 Configure the board

The following components are required for the demo:

- AXE5-Eagle (TEI0185) development board,
- 12VDC 40W power supply
- Arrow-USB-Blaster (TEI-0004-02) for downloading to the FPGA
- 2 x micro-USB Cable (one for the Arrow Blaster, one for the HPS UART)
- 8GB SD card with the sdcard.img

5.10.1 Configure the MSEL DIP Switches

The MSEL2 and MSEL1 DIP switches need to be set to the OFF position (right) for JTAG Boot selection.



5.10.2 Assemble the Hardware

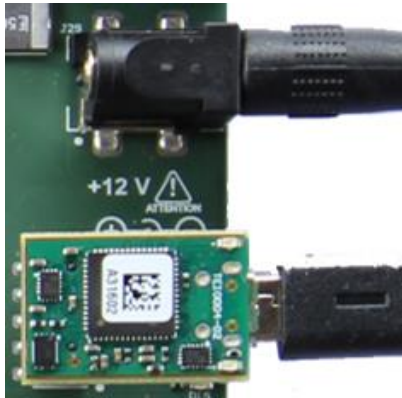
- Insert the SD card in the J24 cage, on the right hand of the board.



- Attach the micro-USB cable to UART (J5) connector



- Plug the Arrow-USB-Blaster (TEI0004-02) into J34 with the USB connector facing to the right.



- Connect both USB cables to the host computer
- Connect the power supply to the AXE5-Eagle J29 barrel connector
- Plug the AC-DC adapter into an AC outlet

5.11 Connect to the target terminal

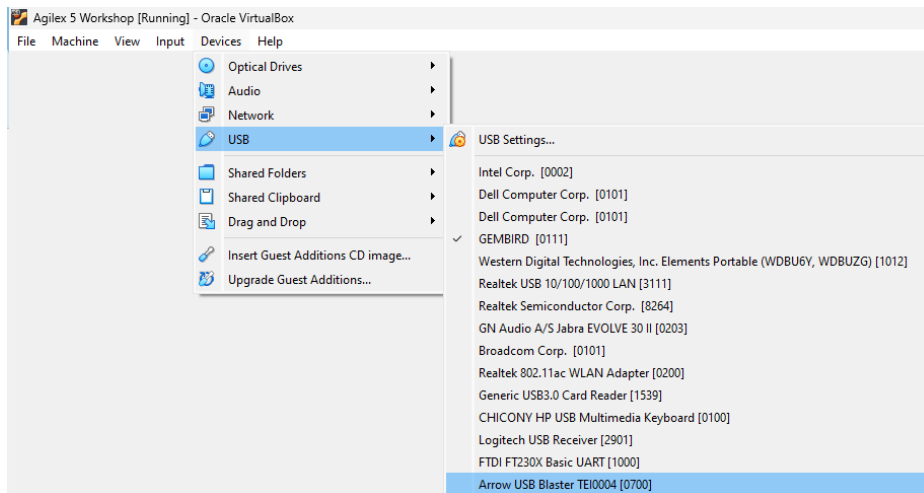
- Launch a terminal program (like *Tera Term VT* or *Putty*) and connect using **serial port**
- Select **115200** baud 8,N,1
- Select the appropriate target COM port

5.12 Boot the Linux Image

- When the integrated SOF file has been downloaded the Secure Device Manager (SDM) will initiate the FPGA configuration process.
- The SDM will read the SOF file. This contains the FPGA image, the HPS configuration data and the U-boot First Stage Boot Loader (FSBL)
- When the FPGA is configured, the SDM will release the Arm processor cluster in the Hard Processing System (HPS) from reset.
- The Arm processor cluster then boots U-boot and Linux from the SD card
- The Linux password is root

5.12.1 Enable the Arrow Blaster in the VirtualBox VM

Click on Devices → USB → Arrow USB Blaster to enable the Blaster in the VirtualBox VM.



5.12.2 Download the FPGA configuration file

The FPGA JTAG chain will expose 1 or 2 endpoints when auto-detected by the Quartus programmer. Follow the appropriate instructions to program the Agilex 5 FPGA for either scenario.

Determine the number of JTAG devices. In the terminal in VirtualBox type

```
$ jtagconfig
```

The jtag chain will respond with one or two devices in the chain. An example of each is shown below

One JTAG device

```
1) Arrow-USB-Blaster [ARA31601-TEI0004]
0364F0DD A5E(C065BB32AR0|D065BB32AR0)
```

Two JTAG devices

```
1) Arrow-USB-Blaster [ARA31601-TEI0004]
4BA06477 ARM_CORESIGHT_SOC_600
0364F0DD A5E(C065BB32AR0|D065BB32AR0)
```

5.12.2.1 Download the SOF file

The instructions can be copied and pasted from the copy and paste guide, a line at a time, into the shell and executed.

- cd to the output files directory

```
$ cd $TOP_FOLDER/ghrd-socfpga/axe5_eagle_ghrd/output_files/
```

Option 1: One Device

```
quartus_pgm -c 1 -m jtag -o "p;axe5_eagle_top_hps.sof@1"
```

Option 2: Two Devices

```
quartus_pgm -c 1 -m jtag -o "p;axe5_eagle_top_hps.sof@2"
```

```
Info: Command: quartus_pgm -c 1 -m jtag -o p;axe5_eagle_top_with_HPS.sof@1
Info (213045): Using programming cable "Arrow-USB-Blaster [ARA31601-TEI0004]"
Info (213011): Using programming file axe5_eagle_top_with_HPS.sof with checksum
0x1CBB0282 for device a5ed065bb32ae4sr0@1
Info (209060): Started Programmer operation at Sun Sep 29 12:01:34 2024
Info (18942): Configuring device index 1
Info (18943): Configuration succeeded at device index 2
Info (20104): Added ARM_CORESIGHT_SOC_600 at device index 1 after configuration
succeeded.
Info (209011): Successfully performed operation(s)
Info (209061): Ended Programmer operation at Sun Sep 29 12:01:36 2024
Info: Quartus Prime Programmer was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 1936 megabytes
Info: Processing ended: Sun Sep 29 12:01:36 2024
Info: Elapsed time: 00:00:07
Info: System process ID: 43280
```

5.13 View the Linux Boot Log

The HPS is released from reset by the SDM and the boot process begins with the FSBL. A snapshot of the initial boot log is shown below.

```

COM7 - Tera Term VT
File Edit Setup Control Window Help

U-Boot SPL 2025.01-g8f306de87f59-dirty (May 23 2025 - 06:33:32 -0400)
Reset state: Cold
MPU      1250000 kHz
L4 Main  4000000 kHz
L4 sys free 1000000 kHz
L4 MP    2000000 kHz
L4 SP    1000000 kHz
SDMMC    500000 kHz
is_ddr_csr_clkgen_locked: ddr csr io96b_0 clkgenA is successfully locked
io96b_cal_status: Calibration for IO96B instance 0x18400400 done at 0 msec!
init_mem_cal: Initial DDR calibration IO96B_0 succeed
DDR: Calibration success
io96b_mb_init: num_instance 1
io96b_mb_init: get memory interface IO96B_0
io96b_mb_init: IO96B_0 mem_interface 0: ip_type_ret: 0x1
io96b_mb_init: IO96B_0 mem_interface 0: instance_id_ret: 0x0
io96b_mb_init: IO96B_0: num_mem_interface: 0x1
LPDDR4: 1024 MiB
ecc_enable_status: ECC enable status: 0
DDR: size check success
DDR: firewall init success
DDR: init success
Bloblist at 72000 not found (err=-2)
WDI: Not starting watchdog@10d00200
Trying to boot from MMC1
## Checking hash(es) for config board-0 ... OK
## Checking hash(es) for Image atf ... crc32+ OK
## Checking hash(es) for Image uboot ... crc32+ OK
## Checking hash(es) for Image fdt-0 ... crc32+ OK

```

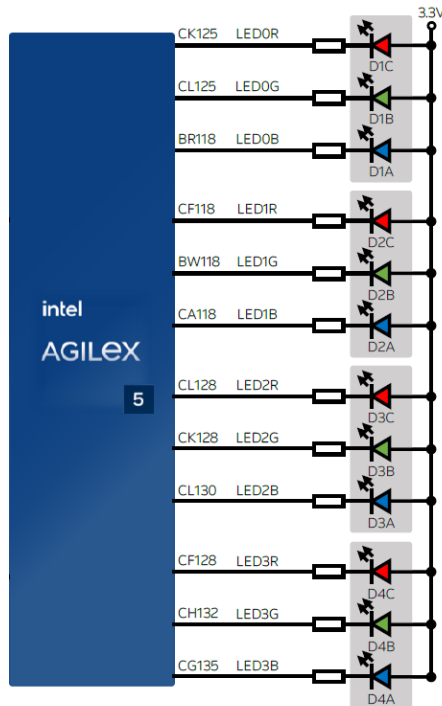
Click on the link below to view a complete Boot Log of Linux on the AXE5-Eagle board

- [AXE5-Eagle board Linux Boot Log](#)

5.14 Turn RGB LEDs On and Off

This section explains how FPGA peripherals can be connected to the HPS and where they are located in the HPS memory map. Accessing FPGA connected LEDs are an example of this.

Four RGB LEDs are directly connected to the FPGA. This is shown in the figure below.

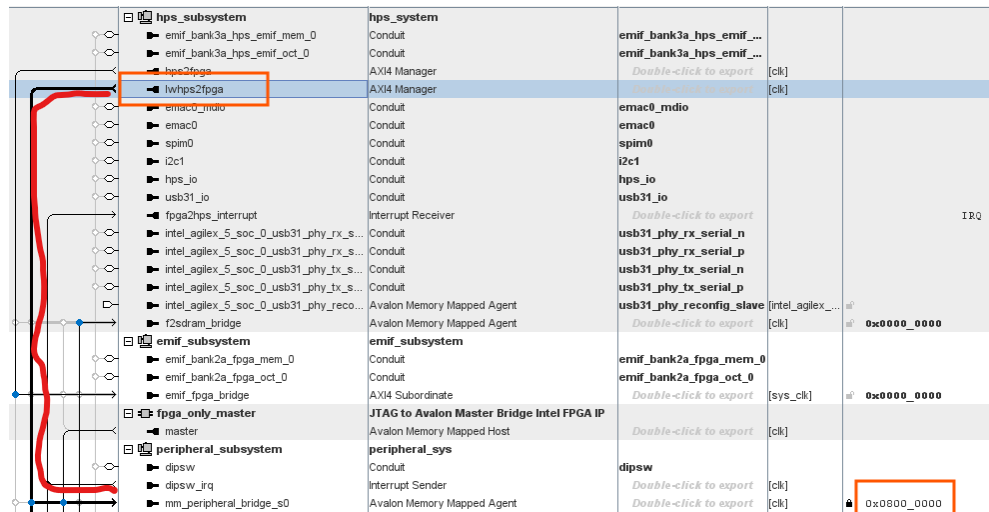


Each RGB LED is connected to a 3 bit PIO peripheral in the FPGA. These PIOs are located in the peripheral_subsystem in the GHRD. The peripheral_subsystem is connected to the HPS memory map via the Lightweight HPS to FPGA (LWH2F) bridge.

The LWH2F bridge is mapped at offset 0x20000000 within the HPS memory map

512 MB LW HPS2FPGA	0x00_2000_0000
47 MB Reserved (Error Generated)	0x00_1D10_0000
1 MB GIC Regs	0x00_1D00_0000
16 MB CCU (Ncore) Regs	0x00_1C00_0000
64 MB MPFE Regs	0x00_1800_0000
128 MB PSS Periphs	0x00_1000_0000
255.5 MB Reserved (Error Generated)	0x00_0008_0000
512 KB OCRAM	0x00_0000_0000

The peripheral_subsystem, within the GHRD, is mapped at address 0x08000000 on the FPGA side of the bridge.



A review of the peripheral_subsystem reveals the local addresses of the rgb_led PIO peripherals.

Conne...	Name	Description	Export	Clock	Base
	mm_bridge	Avalon Memory Mapped Pipeline Bridge Int...			
	s0	Avalon Memory Mapped Agent	mm_peripheral_bridge_s0	[clk]	
	m0	Avalon Memory Mapped Host	Double-click to export	[clk]	
	sys_id	System ID Peripheral Intel FPGA IP			
	control_slave	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0000
	pb	PIO (Parallel I/O) Intel FPGA IP			
	s1	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0010
	external_connection	Conduit	pb		
	irq	Interrupt Sender	pb_irq	[clk]	
	dipsw	PIO (Parallel I/O) Intel FPGA IP			
	s1	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0020
	external_connection	Conduit	dipsw		
	irq	Interrupt Sender	dipsw_irq	[clk]	
	rgb_led0	PIO (Parallel I/O) Intel FPGA IP			
	s1	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0030
	external_connection	Conduit	rgb_led0		
	rgb_led1	PIO (Parallel I/O) Intel FPGA IP			
	s1	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0040
	external_connection	Conduit	rgb_led1		
	rgb_led2	PIO (Parallel I/O) Intel FPGA IP			
	s1	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0050
	external_connection	Conduit	rgb_led2		
	rgb_led3	PIO (Parallel I/O) Intel FPGA IP			
	s1	Avalon Memory Mapped Agent	Double-click to export	[clk]	0x0060
	external_connection	Conduit	rgb_led3		

To calculate the addresses of the LED PIOs from the HPS add the LWH2F bridge address (0x20000000), the address of the peripheral_subsystem (0x08000000) and the individual LED PIO (0x30, 0x40, 0x50 or 0x60) addresses. This results in the following addresses

RGB_LED0	0x28000030
RGB_LED1	0x28000040
RGB_LED2	0x28000050
RGB_LED3	0x28000060

Each PIO has three output bits. The red LED is connected to bit2, the green LED to bit1 and the blue LED to bit0. A zero will turn the LED on and a one will turn it off.

5.14.1 Access the LEDS from Linux using devmem2

Physical addresses in hardware can be directly read or written from Linux user space using the devmem2 package.

- Press enter in the Putty or Tera Term terminal.
- Type **root** when prompted for the password

5.14.1.1 Turn on RGB_LED2

- Illuminate RGB_LED2 – red
\$ devmem2 0x28000050 w 0x03
- Illuminate RGB_LED2 – green
\$ devmem2 0x28000050 w 0x05
- Illuminate RGB_LED2 – blue
\$ devmem2 0x28000050 w 0x06
- Turn off RGB_LED2
\$ devmem2 0x28000050 w 0x07

5.14.2 Specify each LED in Linux as a device

The LEDs are typically addressed as devices from a user space application or from the command line in a shell. The Linux kernel has a device class for LEDs that allows user space to control them. The LED class includes the following features:

- LED brightness. The brightness of an LED is represented as an integer value. For LEDs connected to PWM signals, this value directly controls the brightness. For LEDs connected to GPIOs, a brightness of 0 is off and any other value is on.
- LED directory. LEDs appear in the directory `/sys/class/leds/`.
- LED mapping. GPIOs that are connected directly to LEDs are registered through the Linux LED class sysfs interface. The devicetree can be used to map GPIOs to LEDs, and to define their logic level.

5.14.3 Defining the RGB LEDs as devices on the AXE5-Eagle board

Two steps are required to enable the RGB LEDs as devices when building the embedded Linux kernel.

- Specify the PIOs and LEDS in the devicetree.
- Include and enable the PIO and LED drivers when building the Linux kernel.

5.14.3.1 Examine the AXE5-Eagle devicetree

First examine the sections of the devicetree that define the PIOs. The devicetree is parsed by the Linux kernel at boot time. The devicetree information for that peripheral is passed, at boot time, to the specified device driver.

RGB_LED2 is connected to the `rgb_led2` PIO in the peripheral subsystem. This is defined on [line 54](#) in the AXE5-Eagle devicetree. This entry declares a number of important issues.

- line 54, the name of the PIO as **led_pio2** and its associated FPGA address, `0x08000050`
- line 55, the name of the associated Linux device driver, [altr,pio-1.0](#)
- line 57, the width, in bits of the PIO device

```

246             led_pio2: led-pio2@108000050 {
247                 compatible = "altr,pio-1.0";
248                 reg = <0x00000001 0x08000050 0x10>;
249                 altr,ngpio = <3>;
250                 #gpio-cells = <2>;
251                 gpio-controller;
252             };

```

The `altr,pio-1.0` driver bindings are described in the following [Linux documentation](#).

Now that led_pio2 has been declared it can be referenced when declaring associated RGB LED devices. Linux declares a specific GPIO device and driver for LEDs. This is referred to in the [leds section](#) of the devicetree. The devicetree binding for this driver is described in the following [Linux documentation](#).

Note how each bit of the associated PIO, led_pio2 is declared as an individual device (eg. fpga2_led_red, fpga_led2_blue, fpga2_led_green).

```

69         fpga2_led_red {
70             label = "fpga_led2_red";
71             gpios = <&led_pio2 2 1>;
72         };
73
74         fpga2_led_green {
75             label = "fpga_led2_green";
76             gpios = <&led_pio2 1 1>;
77         };
78
79         fpga2_led_blue {
80             label = "fpga_led2_blue";
81             gpios = <&led_pio2 0 1>;
82         };

```

5.14.3.2 Examine the AXE5-Eagle Linux defconfig

It has been discussed that two linux devices are required in order to achieve the goal of addressing individual RGB LEDs as devices from the Linux prompt. It has already been noted how they are declared in the devicetree. Their associated drivers must be declared in the Linux defconfig file to ensure that they are available at Linux boot time.

The PIO peripheral is declared in the config fragment by the following lines of text.

- [Line 2](#), CONFIG_GPIO_ALTERA

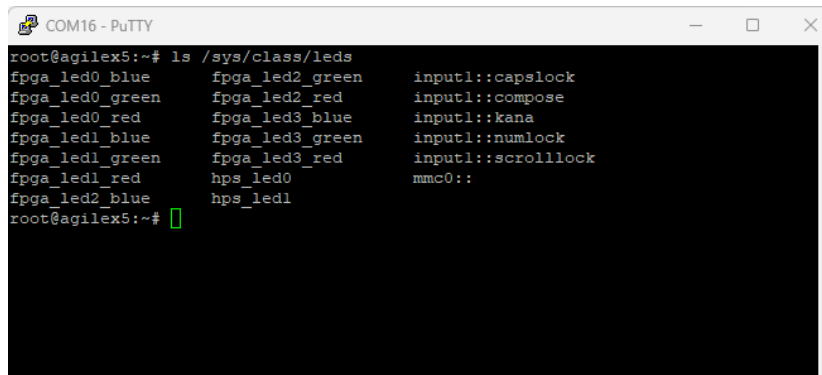
The LEDS GPIO peripheral is declared in the defconfig by the following lines of text.

- [Line 1182](#), CONFIG_LEDS_GPIO

5.14.4 Access the LEDS from Linux as devices

The LED devices can be viewed by entering the following command from the Linux prompt.

```
$ ls /sys/class/devices
```



```
COM16 - PuTTY
root@agilex5:~# ls /sys/class/leds
fpga_led0_blue      fpga_led2_green    input1::capslock
fpga_led0_green     fpga_led2_red      input1::compose
fpga_led0_red       fpga_led3_blue     input1::kana
fpga_led1_blue      fpga_led3_green    input1::numlock
fpga_led1_green     fpga_led3_red      input1::scrolllock
fpga_led1_red       hps_led0           mmc0::
fpga_led2_blue      hps_led1
```

The FPGA RGB LEDs that were declared in the devicetree are listed as individual devices. There are two additional LEDS listed that were also declared in the devicetree, hps_led0 and hps_led1. They are wired to PIOs in the HPS portion of the Agilex SoC FPGA.

- Illuminate RGB_LED2 – red
\$ echo 1 > /sys/class/leds/fpga_led2_red/brightness
- Turn off RGB_LED2 – red
\$ echo 0 > /sys/class/leds/fpga_led2_red/brightness
- Illuminate RGB_LED0 – blue
\$ echo 1 > /sys/class/leds/fpga_led0_blue/brightness
- Turn off RGB_LED0 - blue
\$ echo 0 > /sys/class/leds/fpga_led0_blue/brightness
- Turn on HPS_LED0
\$ echo 1 > /sys/class/leds/hps_led0/brightness
- Turn off HPS_LED0
\$ echo 0 > /sys/class/leds/hps_led0/brightness

6 Additional Resources

[Learn the architecture – Aarch64 Exception](#)

[Learn the architecture – TrustZone for Aarch64](#)

[Hard Processor System Technical Reference Manual](#)

[Hard Processor System Booting User Guide: Agilex 5 SoCs](#)

[Device Configuration User Guide: Agilex 5 FPGAs and SoCs](#)

[Arrow AXE5-Eagle Development Platform](#)

[Command-line Linux Reference Design](#)

[Golden Hardware Reference Design](#)

**CONGRATULATIONS! YOU HAVE SUCCESSFULLY COMPLETED
Creating a Linux Boot Image**

7 Legal Disclaimer

ARROW ELECTRONICS

EVALUATION BOARD LICENSE AGREEMENT

By using this evaluation board or kit (together with all related software, firmware, components, and documentation provided by Arrow, "Evaluation Board"), You ("You") are agreeing to be bound by the terms and conditions of this Evaluation Board License Agreement ("Agreement"). Do not use the Evaluation Board until You have read and agreed to this Agreement. Your use of the Evaluation Board constitutes Your acceptance of this Agreement.

PURPOSE

The purpose of this evaluation board is solely intended for evaluation purposes. Any use of the Board beyond these purposes is on your own risk. Furthermore, according the applicable law, the offering Arrow entity explicitly does not warrant, guarantee or provide any remedies to you with regard to the board.

LICENSE

Arrow grants You a non-exclusive, limited right to use the enclosed Evaluation Board offering limited features only for Your evaluation and testing purposes in a research and development setting. Usage in a live environment is prohibited. The Evaluation Board shall not be, in any case, directly or indirectly assembled as a part in any production of Yours as it is solely developed to serve evaluation purposes and has no direct function and is not a finished product.

EVALUATION BOARD STATUS

The Evaluation Board offers limited features allowing You only to evaluate and test purposes. The Evaluation Board is not intended for consumer or household use. You are not authorized to use the Evaluation Board in any production system, and it may not be offered for sale or lease, or sold, leased or otherwise distributed for commercial purposes.

OWNERSHIP AND COPYRIGHT

Title to the Evaluation Board remains with Arrow and/or its licensors. This Agreement does not involve any transfer of intellectual property rights ("IPR") for evaluation board. You may not remove any copyright or other proprietary rights notices without prior written authorization from Arrow or it licensors.

RESTRICTIONS AND WARNINGS

Before You handle or use the Evaluation Board, You shall comply with all such warnings and other instructions and employ reasonable safety precautions in using the Evaluation Board. Failure to do so may result in death, personal injury, or property damage.

You shall not use the Evaluation Board in any safety critical or functional safety testing, including but not limited to testing of life supporting, military or nuclear applications. Arrow expressly disclaims any responsibility for such usage which shall be made at Your sole risk.

WARRANTY

Arrow warrants that it has the right to provide the evaluation board to you. This warranty is provided by Arrow in lieu of all other warranties, written or oral, statutory, express or implied, including any warranty as to merchantability, non-infringement, fitness for any particular purpose, or uninterrupted or error-free operation, all of which are expressly disclaimed. The evaluation board is provided "as is" without any other rights or warranties, directly or indirectly.

You warrant to Arrow that the evaluation board is used only by electronics experts who understand the dangers of handling and using such items, you assume all responsibility and liability for any improper or unsafe handling or use of the evaluation board by you, your employees, affiliates, contractors, and designees.

LIMITATION OF LIABILITIES



In no event shall Arrow be liable to you, whether in contract, tort (including negligence), strict liability, or any other legal theory, for any direct, indirect, special, consequential, incidental, punitive, or exemplary damages with respect to any matters relating to this agreement. In no event shall arrow's liability arising out of this agreement in the aggregate exceed the amount paid by you under this agreement for the purchase of the evaluation board.

IDENTIFICATION

You shall, at Your expense, defend Arrow and its Affiliates and Licensors against a claim or action brought by a third party for infringement or misappropriation of any patent, copyright, trade secret or other intellectual property right of a third party to the extent resulting from (1) Your combination of the Evaluation Board with any other component, system, software, or firmware, (2) Your modification of the Evaluation Board, or (3) Your use of the Evaluation Board in a manner not permitted under this Agreement. You shall indemnify Arrow and its Affiliates and Licensors against and pay any resulting costs and damages finally awarded against Arrow and its Affiliates and Licensors or agreed to in any settlement, provided that You have sole control of the defense and settlement of the claim or action, and Arrow cooperates in the defense and furnishes all related evidence under its control at Your expense. Arrow will be entitled to participate in the defense of such claim or action and to employ counsel at its own expense.

RECYCLING

The Evaluation Board is not to be disposed as an urban waste. At the end of its life cycle, differentiated waste collection must be followed, as stated in the directive 2002/96/EC. In all the countries belonging to the European Union (EU Dir. 2002/96/EC) and those following differentiated recycling, the Evaluation Board is subject to differentiated recycling at the end of its life cycle, therefore: It is forbidden to dispose the Evaluation Board as an undifferentiated waste or with other domestic wastes. Consult the local authorities for more information on the proper disposal channels. An incorrect Evaluation Board disposal may cause damage to the environment and is punishable by the law.