Software Boot and PL Configuration
Objectives

After completing this module, you will be able to:

- Differentiate between program storage and execution memory options and when to use them

- State various mechanisms for system initialization and application loading

- Describe the Zynq® All Programmable SoC programmable logic (PL) configuration process from an FSBL software application

- Describe the flash writer utility and its requirements

- Analyze flash writing and different boot loading usage scenarios
Overview

- Overview
- Boot Loader
- Zynq All Programmable SoC PS Boot and PL Configuration
- Flash Programmer Utility
- Summary
Embedded applications can range in size from a few kilobytes to a few megabytes

Two types of external memory may be required
- Non-volatile memory for storing program and initialized data
- Memory for running the program
  - If the application size and initialized data are small enough, they can then be downloaded into internal block RAM or OCM RAM

A small application is needed to load the program from non-volatile memory into operational RAM
- Runs on power-on reset and soft resets
- Known as a boot loader

Several mechanisms are available for loading large programs and data stored in non-volatile memory
Standard Zynq All Programmable SoC Boot Model – PS is Boot Master

- **Multi-stage boot process**
  - Stage 0: Runs from ROM; loads from non-volatile memory to OCM
    - Provided by Xilinx; unmodifiable
  - Stage 1: Runs from OCM; loads from non-volatile memory to DDRx memory
    - User developed; Xilinx offers example code SDK project
    - Initiates PS boot and PL configuration
  - Stage 2: Optional; runs from DDR
    - User developed; Xilinx offers example code – Uboot
    - Sourced from flash memory or through common peripherals, programmable logic I/O, etc.
  - Programmable logic configuration can be performed in Stage 1 or 2
Zynq All Programmable SoC Program Loading and Initialization Mechanisms

- **Debug mode**: Configure PL with bitstream then run the application from XMD

- **Boot mode**: FSBL or SSBL configure PL with bitstream

- **Size of application impacts where program can run from**
  - Very small applications can run from OCM (no DDR requirement)
  - Small applications can run from BRAM (no DDR requirement)
  - Applications can run from non-volatile or DDR memory

- **Loading the application with a boot loader**
  - Use nonvolatile memory to store the application, initialize the processor memory from it, and execute it

- **May execute application directly from flash or other non-volatile memory**
Cortex-A9 Processor Memory Space in the Zynq-7000 AP SoC

- Processing system and programmable logic look the same from the processor's viewpoint

- Zynq®-7000 All Programmable SoC PS-based peripherals have a fixed address map
  - AMBA®, AXI interfaces, memory-mapped I/O, register access

Custom Peripheral

```c
int main() {
    int *data = 0x1000_0000;
    int *accel1 = 0x4000_0000;

    // Pure SW processing
    Process_data_sw(data);

    // HW Accelerator-based processing
    Send_data_to_accel(data, accel1);
    process_data_hw(accel1);
    Recv_data_from_accel(data, accel1);
}
```

Code Snippet

<table>
<thead>
<tr>
<th>Start Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000_0000</td>
<td>External DDR RAM</td>
</tr>
<tr>
<td>0x4000_0000</td>
<td>Custom Peripherals (Programmable Logic including PCIe)</td>
</tr>
<tr>
<td>0xE000_0000</td>
<td>Fixed I/O Peripherals</td>
</tr>
<tr>
<td>0xF800_0000</td>
<td>Fixed Internal Peripherals (Timers, Watchdog, DMA, Interconnect)</td>
</tr>
<tr>
<td>0xFC00_0000</td>
<td>Flash Memory</td>
</tr>
<tr>
<td>0xFC00_0000</td>
<td>On-Chip Memory</td>
</tr>
</tbody>
</table>

Programmer’s View of Custom Accelerators and Peripherals
Configuring the PL through SDK

- Download the bitstream and then the application
  - Select Xilinx Tools > Program FPGA
  - Locate and select hardware bit file
  - No BMM file as the ELF runs in PS DDR memory
  - Click Program

- The programmable logic configures and starts executing the application in DDR memory

- When launched (later), the XMD debugger will halt the processor and load the actual application for debugging
Run configurations through SDK
- Select software application in the project explorer pane
- Select **Run > Run Configurations**
- Double-click **Xilinx C/C++ ELF(GDB)** to create a run configuration for application
- Click **Run** to download executables (.elf) and run application
Software Program Loading in the Zynq All Programmable SoC (2)

- **Debug using XMD**
  - If PL is used, select **Xilinx Tools > Program FPGA**
    - Choose the .bit file and click **Program**; programmable logic configures
  - Launch **XMD** and select **Xilinx Tools > XMD Console**
    - Command console opens
    - Connect to ARM hardware
    - dow FSBL.elf to initialize PS
    - dow <executable>.elf downloads executable
Program Loading and Initialization Mechanisms (MicroBlaze Processor)

- Initializing bitstreams with the application
  - Application is small enough to be contained within the programmable logic

- Downloading an application by using XMD
  - Application resides in external memory, use XMD to download the application after the programmable logic has been configured with the bitstream
  - Bitstreams must be initialized with the BootLoop program in order to place the processor in a proper state between bitstream downloading and application downloading through XMD

- Loading the application with a boot loader
  - Use nonvolatile memory to store the application, initialize the processor memory from it, and execute it

- May be executing application from flash or other non-volatile memory
Bitstream Initialization in SDK (MicroBlaze Processor)

► Initialize bitstreams with the application
  – Select Xilinx Tools > Program FPGA
    - Locate and select hardware bit and block RAM location bmm files
    - Specify an ELF to download and click Program

► Initialize bitstreams with BootLoop
  – BootLoop is a software application that keeps the processor in a defined state until the actual application is downloaded in a controlled manner by the debugger
  – Select Xilinx Tools > Program FPGA, select BootLoop, and click Program
  – The programmable logic configures and starts executing the BootLoop program in block RAM
  – When launched (later), the XMD debugger will halt the processor and load the actual application for debugging
Software Program Loading with XMD (MicroBlaze Processor)

- Processor hardware debug unit must already be instantiated (JTAG)
  - For MicroBlaze-processor systems, the processor must be connected to a Microprocessor Debug Module (MDM)

- Select Xilinx Tools > Program FPGA, select BootLoop, and click Program; programmable logic configures in an infinite loop
  - Select Xilinx Tools > Program FPGA to download and execute the selected BootLoop

- Select Xilinx Tools > XMD Console to launch XMD
  - Command console opens
  - dow <executable>.elf downloads executable
Boot Loader

- Overview
- **Boot Loader**
- Zynq All Programmable SoC PS Boot and PL Configuration
- Flash Programmer Utility
- Summary
What is a boot loader?
- First program run
  - Runs on power up or reset
  - Copies program from non-volatile memory to DDR/OCM/block RAM
  - Could load application directly, an OS loader, or the OS itself
  - When done transfers control to selected program

Why? Final software system...
- Might not fit into ROM
- Might require some kind of run-time set up before it is launched
- Might be determined dynamically

Boot loaders tend to range from simple to quite complex systems
- May include communication drivers to download application from remote location
- May include decompression/decryption engine for compressed/encrypted images
Boot Load Scenarios

Commonly used boot load scenarios
- Booting from flash devices
- Booting from PROMs
- Booting from a serial line
- Booting from Ethernet with BootP and TFTP
- Booting from PCIe or other communications interface
- Command line-based interactive boot load

Each method has its advantages, disadvantages, and applicability
Image Formats

➤ Boot loader must understand both
  – Image format of the file (application, bitstream, or data)
  – Organization of the images in the NV storage medium

➤ Formats
  – Common: ELF, Intel MCS-86 file (.mcs), .BIN, Motorola SREC, Intel I-hex, binary, gzip/bzipped images
  – Less common: Custom formats are common as well

➤ Image formats have different processing complexities and sizes
  – ELF, SREC/iHex, binary, compressed
    • Decreasing order of size requirements
  – Compressed, ELF, SREC/iHex, binary
    • Decreasing order of processing complexity
Stage 0: ROM Based (Zynq All Programmable SoC)

- Processor boots from boot ROM (128KB)
  - Xilinx provided
  - Not viewable

- Copies First Stage Boot Loader (FSBL) from memory device to OCM static RAM (256KB)
  - Maximum size is 192KB (rest can be used as stack, BSS, or non-initialized memory)
  - Xilinx provided

- Starts executing FSBL from OCM RAM
Stage 1: First Stage Boot Loader (FSBL) (Zynq All Programmable SoC)

- Example FSBL provided by Xilinx as an SDK example project
  - Otherwise user developed

- Copies next stage of code into
  - DDRx or static memory (OCM)
  - And/or enables an external device for Stage 2

- Further initialization of PS components and peripherals

- Optionally configures programmable logic

- Upon completion, launches application or Second Stage Boot Load
Stage 2: Second Stage Boot Loader (SSBL) (Optional) (Zynq All Programmable SoC)

➤ Example U-Boot provided by Xilinx
   – Otherwise user developed

➤ Loaded from user-selected external device

➤ Flexibility in boot sources
   – Static memory
   – Dynamic memory
   – PS peripherals such as
     • USB, Ethernet, or SD
   – Programmable logic I/Os

➤ Initializes rest of PS

➤ Optionally configures PL
Boot Loader Flow in SDK (MicroBlaze Processor)

➤ Build the target software application as an ELF executable file

➤ Use the Flash Writer utility to
  – Convert the application ELF to a Motorola S-Record format, an industry-standard format mostly used for flash programming
  – Generate boot loader application
    • Explained in the next section

➤ Select Xilinx Tools > Program FPGA, specify the boot loader application to be used, and click Program

➤ The programmable logic configures with the launch of the boot loader application
  – On system start up, the boot loader fetches image from the boot load target
  – Copies/unpacks the image to external memory
  – Performs other initialization, if needed
  – Transfers control to the entry point of the final application
Processor Boot Sequence (MicroBlaze Processor)

/* file: boot.S */
_boot: jump_boot0
_boot0: jump_start

/* file: _start.c */
_start: clear .bss
_crt0: clear_sbss
_init: init_stack
_clear_TBR
_call: main

/* file: hello.c */
main: call __eabi

do program

/* file: eabi.S */
__eabi: init EABI
return
Zynq All Programmable SoC PS Boot and PL Configuration

- Overview
- Boot Loader
- Zynq All Programmable SoC PS Boot and PL Configuration
- Flash Programmer Utility
- Summary
Zynq-7000 All Programmable SoC Boot and Configuration

Zynq-7000 All Programmable SoC devices can be booted and/or configured in
- Secure mode via static memories only (JTAG excluded)
  - Ability to have secure software and protects bitstream and IP
- Non-secure mode via JTAG or static memories (debug and development environment)
  - Standard boot model

Three master boot devices
- QSPI: serial memory, linear addressing
- NAND: complex parallel memory
- NOR: parallel memory, linear addressing
- SD: Flash memory card
- JTAG: download cable

Secondary boot devices
- USB, Ethernet, and most other peripherals
Non-Secure OS Boot Example

Stage 2
- Uboot runs from Second Stage Source
  - Responsible for loading OS kernel
  - Can be from any source

Stage 1
- First Stage Bootloader runs from OCM
  - PS Boot Data is loaded into specified memory
  - Bitstream is loaded and PL is configured (optional)
  - Can enable Second stage boot (optional)

- Power up Zynq-7000 EPP
  - Boot Mode Pins Identifies Boot Device
  - ROM Code runs
  - Copies First Stage Boot Loader to OCM
Secure Boot Example

Stage 1
- First Stage Boot Loader runs from OCM
  - PS Boot Data is decrypted using AES engine and put into specified memory
  - Bitstream is decrypted using AES and PL is configured
- Power up Zynq-7000 EPP
  - Boot Mode Pins Identifies Boot Device
  - ROM Code runs
  - First Stage Boot Loader is decrypted using AES and then copied into OCM

Stage 2
- UBoot from Xilinx does not support encryption at this point

OS (Kernel & Drivers)
- UBoot
- FSBL
Secure Linux Boot Example

- **Stage 0**: Power up Zynq-7000 EPP
  - Boot Mode Pins Identifies Boot Device
  - ROM Code runs
  - Decrypts and authenticates FSBL using AES/SHA and then copied into OCM (PL powered on)

- **Stage 1**: First Stage Boot Loader runs from OCM
  - Decrypts and authenticates PS Boot Data using AES/SHA engine and puts into specified memory OR
  - Enables Second stage boot (optional)
  - Decrypts and authenticates Bitstream using AES/SHA and configures PL (optional)

- **Stage 2**: User developed code can be secure

- **uBoot For Linux**
  - Common Boot Path
  - PS Boot Path
  - PL Configuration Path

- **Secure Boot Process**
  - Step 1: CPU Boots From Internal ROM
  - Step 2: PS Boot Image Decryption and Authentication
  - Step 3: Decrypted, Authenticated Image Stored in OCM
  - Step 4: PL Configuration (Optional)
The Zynq All Programmable SoC is a processor first, programmable logic second

Most options, features, and configurations are controlled by software setup
- Clock generation
- MIO usage
- Processor cache and memory configuration

The Vivado® Design Suite Export to SDK utility generates the PS configuration code
- Used by FSBL
- Various *.c and *.h files
Software Drivers

- Xilinx provides drivers for all primary boot interfaces
  - Linux driver
  - Standalone driver

- Example FSBL provided

- Example SSBL provided
PL Device Configuration Services

- Used by FSBL
- Set of Standalone library services
- The device configuration interface has three main functionalities
  - AXI-PCAP
  - Security policy
  - System monitor
    - Currently not implemented
- Supports the downloading of the programmable logic bitstream and readback of the decrypted image
- Services are detailed in the *Software Developers Guide (UG821)*
SDK FSBL Support

- **SDK software project**

- **Complete FSBL boot application**
  - Software application load
  - PL configuration from bit file
  - Support for *golden* image

- **Requires *.bif file for image generation**

- **All source code is included**
  - Can be modified for other boot sources
    - Ethernet
    - USB
    - Serial
Building the FSBL Software Application Project

- Created with an SDK software project template
- Automatically loads program and bit file images from flash
- Requires a special FSBL partition image in flash
- BootGen tool builds flash image binary
Programmable the PL from SDK

- Configure PL from SDK

- Select Xilinx Tools > Program FPGA

- Specify hardware BIT file
  - Files typically located in the Vivado Design Suite project
  - No BMM file as the ELF file runs in DDR memory

- Once PS is configured, use RUN command to download new ELF file
Creating a Zynq All Programmable SoC boot image file using SDK

- Right-click the software application project and select Create Boot Image
- Add the FSBL ELF file
- Add the PL bitstream file (if PL design available)
- Add the software application ELF file
- Select the output file name
- Click Create Image to create the image file

Output files (based on extension .bin or .mcs)

- Creates Intel MCS-86 file (.mcs) for programming flash
- Creates .BIN image for booting the application from the SD card
1. PS runs Stage 0 boot from OCM-ROM, which loads the First Stage Boot Loader from Flash into OCM-RAM
2. PL bitstream may be loaded into the PL under FSBL control at this time or may be deferred until later
3. FSBL loads Second Stage Boot Loader or Application into DDRx memory and runs SSBL / Application
   - When SSBL is complete, the OS begins execution out of DDRx memory
   - Optionally, PL may be configured from SSBL now or even later under user software control
1. PS runs Stage 0 boot from OCM-ROM, which loads the First Stage Boot Loader from Flash into OCM-RAM
2. PL bitstream may come from SD card or Flash and is loaded into the PL under FSBL control at this time or may be deferred until later
3. FSBL loads Second Stage Boot Loader or Application from the SD card into DDRx memory and runs SSBL / Application
   - When SSBL is complete, the OS begins execution out of DDRx memory
   - Optionally, PL may be configured from SSBL now or even later under user software control
1. PS runs Stage 0 boot from OCM-ROM, which loads the First Stage Boot Loader from Flash into OCM-RAM
2. FSBL may configure the PL at this time or defer configuration
3. FSBL configures Ethernet MAC and loads Second Stage Boot Loader or Application from the remote server into DDRx memory and runs SSBL / Application
   - When SSBL is complete, the OS begins execution out of DDRx memory
   - Optionally, PL may be configured during FSBL or SSBL or even later under user software control
Configuring and Re-Configuring the PL

- The PL is configured via the device configuration interface module.

- Accessed via a software application using an AXI port in the PS
  - Supported by Xilinx-provided APIs in SDK
  - Recommended methodology

- Separate DMA port into the Central interconnect for simultaneous PL configuration with software download

- Accessed from the PL via a GPx master AXI port
  - Not recommended
Flash Programmer Utility

- Overview
- Boot Loader
- Zynq All Programmable SoC PS Boot and PL Configuration
- Flash Programmer Utility
- Summary
Flash Image Generation Flow (Zynq All Programmable SoC)
Use of the Vivado hardware manager to program flash

- Assign flash image to flash memory attached to the Zynq device
  - Connect to the JTAG chain containing the ARM® DAP of the Zynq device
  - Download the flash programming application into the PS of the Zynq device using the hardware manager
- Procedure
  - Erase: the hardware manager tells the Zynq device application to erase flash memory
  - Program: the hardware manager loads the flash image into the PS buffer of the Zynq device; Zynq device application writes the image to flash memory
  - Verify: Zynq device reads the flash contents and writes into buffer; reads the buffer and compares against the original flash image

Use of SDK to program flash

- Creates a flash Image using a flash image generator
- The flash writer uses the same programming engine as hardware manager
Creating a First Stage Bootload Software Application (Zynq All Programmable SoC)

> SDK supports an example FSBL software application project
  - Target application must be in Zynq device FSBL format in flash
  - RAM must exist at location targeted by flash image format
  - Selection of target hardware processor(s)

> FSBL
  - May configure the programmable logic with hardware bitstream
  - May load OS image or standalone image or SSBL image from the non-volatile memory to RAM (DDR)
  - Transfers program control to the newly loaded application/OS

> Xilinx FSBL supports multiple partitions; each partition can be a code image or a bitstream
Select Xilinx Tools > Program Flash

Select the image file and offset
- A full flash image offset will always be 0

Click Program to "flash" the image
Parallel Flash Memory Boot (MicroBlaze Processor)

- SDK provides a Program Flash Memory dialog box for programming flash devices
- Supports flash devices that support Intel command sets
- Supports most all physical flash arrangements
- Supports an executable file as an input format and provides the option of converting it into SREC format
- Supports flash bootloader software application creation
- Flash devices must interface through the EMC peripheral
- Requires XMD to execute a Tcl file to perform programming and verification functions
Creating a Boot Load Software Application (MicroBlaze Processor)

SDK supports an example boot load software application project
- Target application must be in SREC format in flash (as provided by the flash programming utility)
- RAM must exist at location targeted by SREC format

All source is provided and can be modified
- Address of flash memory must be specified in `blconfig.h`
- Boot messages sent to STDOUT if VERBOSE symbol defined
Flash Writer Utility (MicroBlaze Processor)

- Object file to be programmed
- Object in ELF or SREC format
- Hardware instance of flash memory
  - Base address, size, and data width automatically determined from hardware platform specification
  - Storage offset from beginning of flash memory
- Programming scratchpad RAM is required
Summary

- Overview
- Boot Loader
- Zynq All Programmable SoC PS Boot and PL Configuration
- Flash Programmer Utility

Summary
1. Where are the different places in which a user application can reside and when will you use them?

2. What is the Flash Programmer utility? What are some of the requirements?
SDK supports various mechanisms for initializing an application. The choice depends on the size of the application, how it will be stored, and the memory technology environment in which it will execute.

FSBL application provided in SDK features
- PS boot
- PL configuration

SDK provides a Flash Programmer utility that you can use to program flash devices

SDK provides a sample bootloader software application project
Lab 6: Boot Loading from Flash Memory (Zynq All Programmable SoC)

Introduction

- The lab illustrates the steps involved in booting an application from QSPI flash

Objectives

- Open the Vivado Design Suite project and re-configure to Zynq All Programmable SoC PS settings
- Export to SDK and launch the software project
- Create a Zynq All Programmable SoC boot image file using the ELF and BIT files
- Program the flash by copying the MCS file to flash
- Boot the system by loading the actual application stored in flash