Linux on Zynq

ECE 699 Hardware/Software Codesign

Jeremy Trimble

Linux

- A clone of the Unix operating system.
  - Started by Linus Torvalds in 1991.
    - Originally for i386 architecture.
  - Now supports 20 different architectures.
    - Including ARM.
  - Open-source
    - GNU General Public License (GPL)
- Arguably the world's largest software project.
  - >3700 developers around the world.
  - 1 code change accepted every 7.5 minutes.
  - ~359 lines of code added each hour.
    - Source: https://github.com/gregkh/kernel-history

A few technicalities

- Linux is a kernel.
  - Not an operating system.
  - Typically run with “GNU” Operating System
    - So the correct name is actually “GNU/Linux”
    - But most folks just say “Linux.”
- GNU project
  - Started by Richard Stallman
  - Provides common “userspace” utilities:
    - ls, make, gcc, emacs

Advantages of Linux on Zynq

- Flexibility
  - More like a general-purpose computer.
  - Multitasking, filesystems, networking, hardware support.
- Ease of development
  - Kernel protects against certain types of software errors.
  - Vast ecosystem of open-source tools and languages.
  - Faster time-to-market.
- Graphics support
  - “X Windows”

Disadvantages

- Complexity
  - More hoops to jump through for some tasks.
    - Memory management makes interacting with PL cores more involved.
- Overhead
  - For very simple programs, bare metal can be faster.

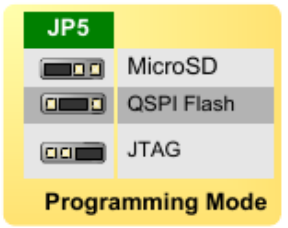
Key parts of a (Zynq) Linux System

- Bootloader
  - Zynq FSBL – “First Stage Bootloader”
  - U-Boot
- Kernel
- PL Image
- Device Tree File
  - A file describing the computer where Linux will run.
- Root Filesystem
  - Linux Distributions

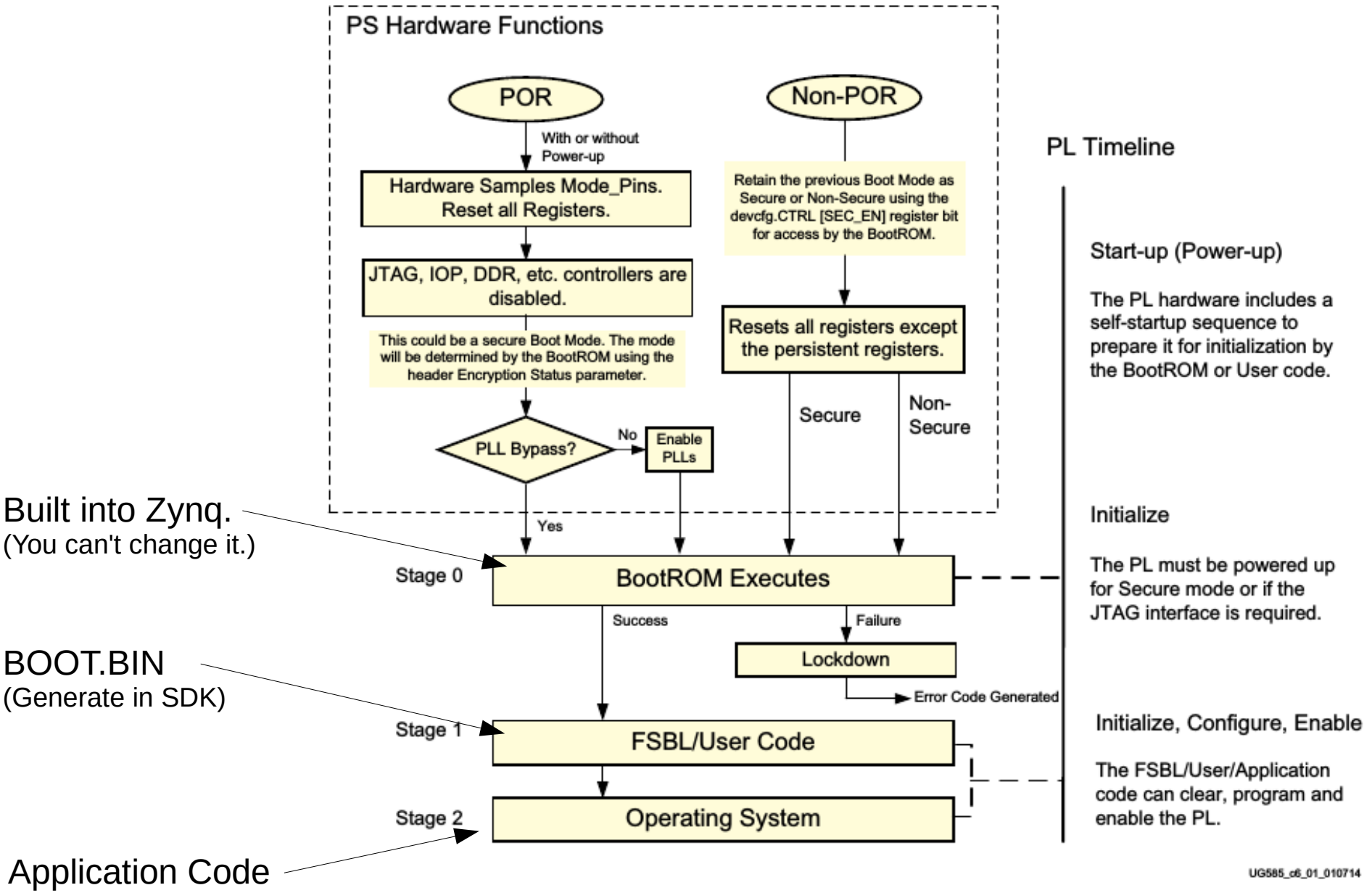
How to boot the Zynq

- Zynq supports multiple boot sources:
  - NAND/NOR Flash, SD Card, Quad-SPI, JTAG.
  - SD boot supports FAT16/FAT32.
    - BOOT.BIN, first partition
  - Refer to TRM (UG-585) Chapter 6.

- Zybo
  - Boot Mode Jumper
    - JP5



- ZedBoard
  - MIO2-6 Jumpers
    - JP7-11



Built into Zynq.
(You can't change it.)

BOOT.BIN
(Generate in SDK)

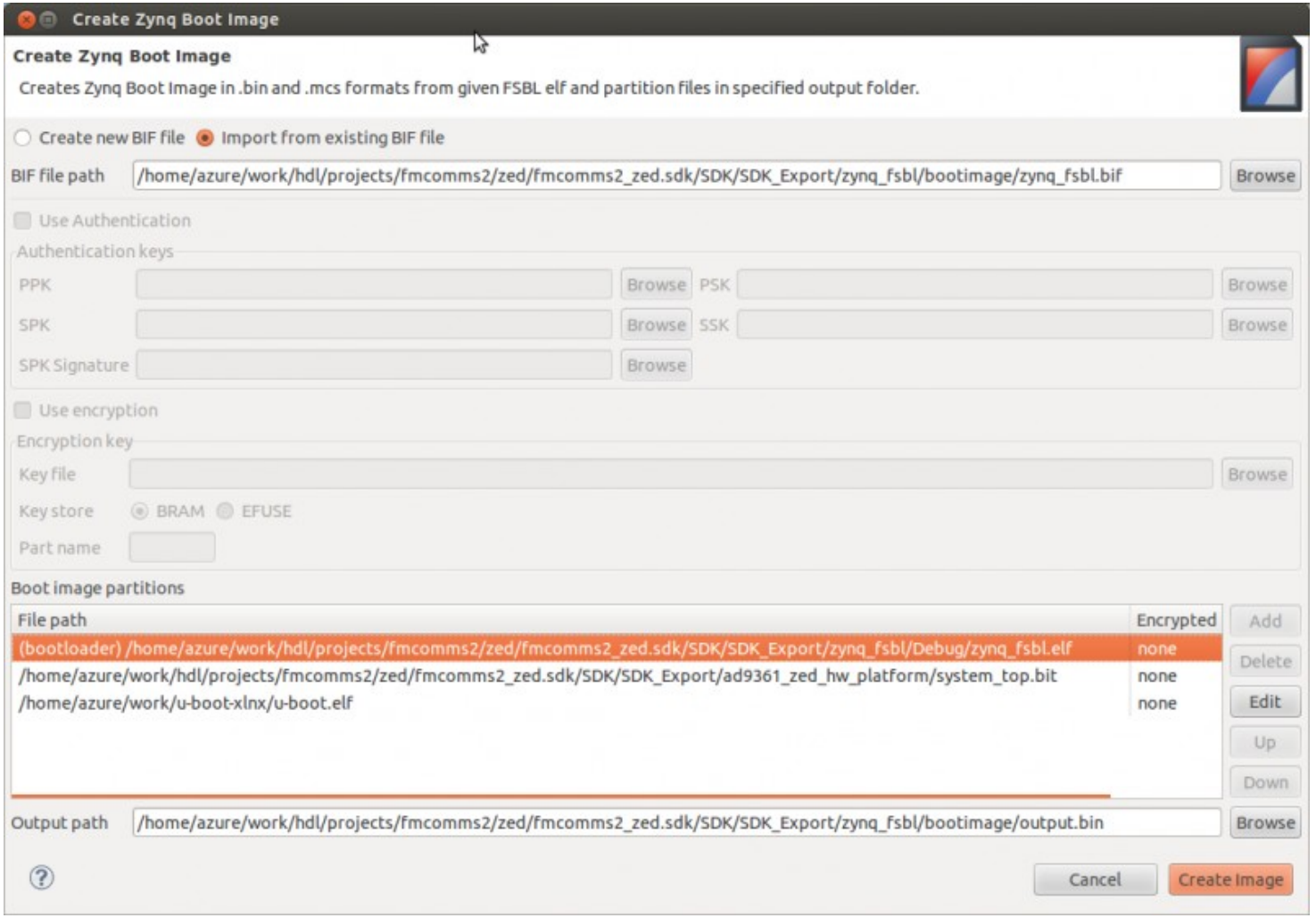
Application Code
(Or U-Boot & Linux)

Figure 6-1: **PS/PL Boot Process for Hardware and Software**

Zynq FSBL

("fuzzball", anyone?)

- "First Stage BootLoader"
  - Executed by BootROM
  - Sets up MIO, clocks
    - As configured in the PS7 IP core in Vivado.
  - Optionally:
    - Can load a PL image (bitstream) of your choosing.
    - Can run a PS image of your choosing.
  - Generate BOOT.BIN in SDK.
    - Xilinx Tools > Create Zynq Boot Image


Create Zynq Boot Image
Create Zynq Boot Image
Creates Zynq Boot Image in .bin and .mcs formats from given FSBL elf and partition files in specified output folder.
Create new BIF file
Import from existing BIF file
BIF file path
/home/azure/work/hdl/projects/fmcomms2/zed/fmcomms2_zed.sdk/SDK/SDK_Export/zynq_fsbl/bootimage/zynq_fsbl.bif
Browse
Use Authentication
Authentication keys
PPK
Browse
PSK
Browse
SPK
Browse
SSK
Browse
SPK Signature
Browse
Use encryption
Encryption key
Key file
Browse
Key store
BRAM
EFUSE
Part name
Boot image partitions
File path
Encrypted
(bootloader) /home/azure/work/hdl/projects/fmcomms2/zed/fmcomms2_zed.sdk/SDK/SDK_Export/zynq_fsbl/Debug/zynq_fsbl.elf
none
/home/azure/work/hdl/projects/fmcomms2/zed/fmcomms2_zed.sdk/SDK/SDK_Export/ad9361_zed_hw_platform/system_top.bit
none
/home/azure/work/u-boot-xlnx/u-boot.elf
none
Add
Delete
Edit
Up
Down
Output path
/home/azure/work/hdl/projects/fmcomms2/zed/fmcomms2_zed.sdk/SDK/SDK_Export/zynq_fsbl/bootimage/output.bin
Browse
Cancel
Create Image

Das U-Boot

- Full-featured open-source bootloader.
  - Started by Wolfgang Denk in 2000.
    - Source Code: http://git.denx.de/?p=u-boot.git;a=summary
  - Features:
    - Loading images from SD/MMC Cards, USB devices
      - Numerous filesystems (FAT)
    - Loading images over a network (ethernet support)
      - TFTP, NFS, DHCP/bootp
    - Loading Zynq PL images at boot time
      - Using “fpga” command.
    - Interactive command prompt
    - Scriptable
    - ...and it boots things!

Linux Kernel

- Compile your own!
  - Source code: https://kernel.org/, mirrored on GitHub
- Written in C.
  - You can cross-compile your own kernel in minutes.
  - Customize the features you want.
  - Quick-Start (Ubuntu):
    - `sudo apt-get install build-essential bc gcc-arm-linux-gnueabi git`
    - `git clone https://github.com/torvalds/linux`
    - `cd linux`
    - `export ARCH=arm; export CROSS_COMPILE=arm-linux-gnueabihf-`
    - `make xilinx_zynq_defconfig`
    - `make -j4 uImage LOADADDR=0x00008000 modules`
- Or, use a pre-built kernel from Xilinx, Digilent, Xillybus...

Device Tree

- A machine-readable description of the hardware
  - Passed to the kernel at boot time
  - Same kernel can run on different machines
    - Any differences in the hardware are captured in the device tree.
- Two forms of file:
  - *.dts – Human-readable.
  - *.dtb – “Binary blob” passed to kernel @ boot.
    - Device tree compiler ***dtc*** converts dts <-> dtb.
  - Shows up in /proc on running kernel.

Device Tree

- Heirarchical data
  - Nodes bounded by {}
  - Attributes defined inside nodes.
    - "**compatible**" attribute specifies driver
    - "**reg**" attribute specifies memory-mapped address ranges
- Example...

```
/* From Zybo Device Tree: */

ps7_uart_1: serial@e0001000 {
  clock-names = "uart_clk", "pclk";
  clocks = <&clkc 24>, <&clkc 41>;
  compatible = "xlnx,xuartps",
               "cdns,uart-r1p8";
  current-speed = <115200>;
  device_type = "serial";
  interrupt-parent = <&ps7_scugic_0>;
  interrupts = <0 50 4>;
  port-number = <0>;
  reg = <0xe0001000 0x1000>;
  xlnx,has-modem = <0x0>;
} ;
```

Device Tree Generator

- Xilinx SDK Plugin
  - Automatically generates device tree
    - Based on HW Platform Spec
  - https://github.com/Xilinx/device-tree-xlnx
    - Under active development
  - I haven't had good luck with it.

Root Filesystem

- Contains the rest of the Operating System
  - Mounted by the kernel during boot.
    - Typically separate partition from the bootloader and kernel.
  - Linux “Distributions”/Flavors
    - Many to choose from, some with different focuses.
    - Linaro (based on Ubuntu) is a popular choice.
      - I like Debian and Arch.

Kernel Modules

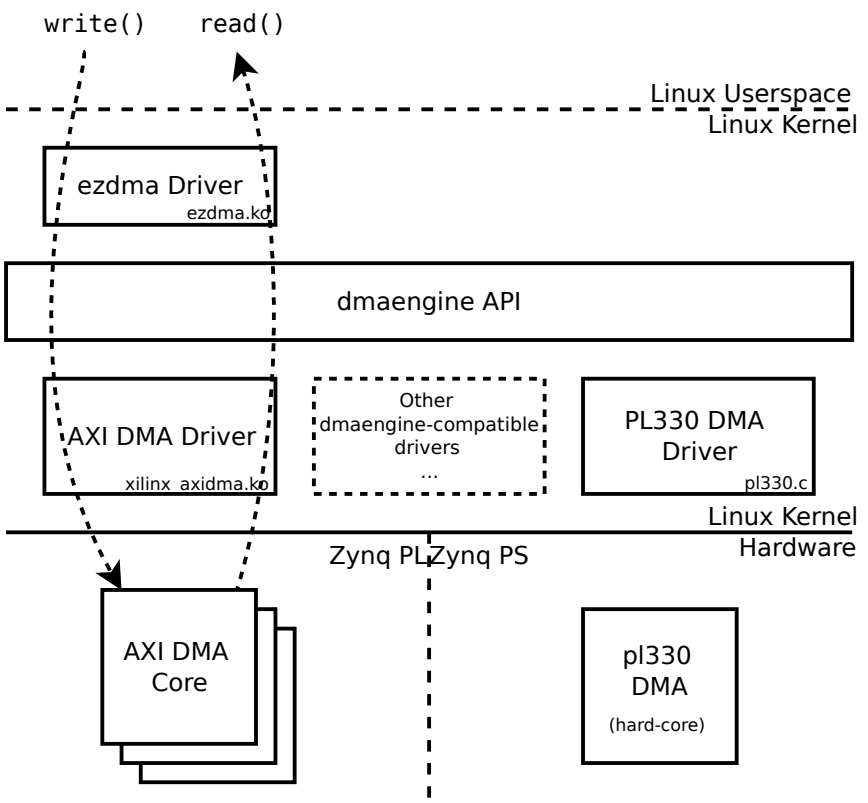
- Insert/remove code into running kernel
  - .ko files – “Kernel Object”
  - Dynamically linked at runtime
  - Device drivers often compiled as modules
    - Automatically inserted when new devices are plugged in
  - Other uses:
    - Support for cryptographic algorithms, IPv6, KVM.
  - Commands:
    - `insmod` – Insert a module into the kernel.
    - `rmmod` – Remove a module from the kernel.
    - `modprobe` – Automatically resolves dependencies.

Challenges to writing Device Drivers

- The kernel is complex.
  - Memory-managed.
    - Separation between “user memory” and “kernel memory.”
    - “User memory” may not always be present in RAM.
  - Multithreaded.
  - Only allowed to sleep/block in certain contexts.
  - Easily crash the entire system, or corrupt data.
  - No floating-point operations.
- The kernel is always changing.
  - Documentation is almost instantly out-of-date.
  - There is no “stable API” design tenet.

ezdma

- DMA can be tricky (especially in the kernel)
- ezdma tries to make it easier
  - simply read()/write()
- Support:
  - Uses “dmaengine” API.
  - At least ~40 dmaengine drivers
    - ezdma should work with them all
    - AXI DMA
    - AXI CDMA
    - AXI VDMA
    - PL330 DMA



- Released as GPL: https://github.com/jeremytrimble/ezdma
- I hope to contribute this to the Linux kernel.

ezdma

- Usage:
  - See the README:
    - https://github.com/jeremytrimble/ezdma

ezdma Caveats

- May need some DMA core-specific tweaks, depending on which driver you're using.
  - For my project, I had to set the DelayIRQ register in the AXI DMA. This is device-specific.
- As of 4/30/2015:
  - xilinx_axidma.ko has a few bugs:
    - Doesn't register with of_dma system.
    - Early callbacks cause data corruption
      - https://github.com/Xilinx/linux-xlnx/issues/54
      - I've submitted a fix for this but Xilinx hasn't accepted yet.
  - Run with my fork and you should be good:
    - https://github.com/jeremytrimble/linux-xlnx

Resources

- Tutorial on device trees: http://xillybus.com/tutorials/device-tree-zynq-1
- Linux kernel documentation: https://github.com/torvalds/linux/tree/master/Documentation
- Linux Weekly News: http://lwn.net/
- Linux Device Drivers, 3rd Ed. (LDD3): https://lwn.net/Kernel/LDD3/
- Haifa Linux Club: http://haifux.org/index.html

Resources

- Xilinx Linux tree:
  https://github.com/Xilinx/linux-xlnx
  - With my bugfixes:
    https://github.com/jeremytrimble/linux-xlnx
- Analog Devices Linux tree:
  https://github.com/analogdevicesinc/linux