Embedded Linux

Device driver development

Sébastien Bilavarn
Outline

- Ch1 – Introduction to Linux
- Ch2 – Linux kernel overview
- Ch3 – Linux for Embedded Systems
- Ch4 – Embedded Linux distributions
- Ch5 – Case study: Xilinx PowerPC Linux
- Ch5 bis – Case study: Xilinx Zynq-7000 Linux
- Ch6 – Device driver development
Case study: Xilinx Zynq-7000 Linux

- Introduction
- Target platform: Xilinx Zynq-7000
  - Platform configuration
  - Kernel configuration and compilation
  - Kernel startup and execution
Target platform: Xilinx Zynq-7000

- **Processing System**
  - Hard IP processor: implemented on the silicon layout (where a soft IP would be implemented using FPGA logic).

- **Programmable logic**
  - FPGA
  - Provides the ability to add custom logic
Development of a Linux kernel on a reconfigurable platform:

- Hardware platform configuration
  - Peripherals use on demand (UART, Timer, Ethernet, etc.)
  - Processor architecture itself can be configured according to the needs (frequency, on-chip/off-chip memory, coprocessors, etc.)
  - Compared to a standard kernel development (i.e. for a fixed platform architecture), information on the platform (BSP), in particular on the processor, must be known in order to configure properly the kernel. This process is more complex than on standard platforms where the architecture is fixed (memory size, enabling cache, etc.).

- Kernel development steps
  - Configuration of the hardware platform
  - Kernel configuration and compilation
  - Kernel startup and test
Linux 4.19.0

- Distributions
  - www.kernel.org
  - www.uclinux.org
  - Xilinx Zynq Linux
  - Petalinux
  - Zynq Yocto

- Cross development tools for ARM
  - Provided with Xilinx tools
  - Sourcery CodeBench (Mentor)

- Zynq platform (ZedBoard / ARM CortexA9)
  - Xilinx Vivado, SDK

- Support of an « open source » community
  - Kernel.org, u-boot, Xilinx, etc
Xilinx Zynq-7000 Linux

- Introduction
- Target platform: Xilinx Zynq-7000
  - Platform configuration
  - Kernel configuration and compilation
  - Kernel startup and execution
Xilinx Zynq-7000

ZedBoard (DIGILENT)
- Zynq®-7000 AP SoC
  XC7Z020-CLG484-1
- Memory (512 MB DDR3, 256 Mb Spansion® Quad-SPI Flash, 4 GB SD card)
- Onboard USB-JTAG
- 10/100/1000 Ethernet
- USB OTG 2.0, USB-UART
- PS & PL I/O expansion
- Display (1080p HDMI, 12-bit VGA, 128 x 32 OLED)
- I2S Audio CODEC
Extensible Processing Platform

ZedBoard (DIGILENT)

- Processing System
  - ARM dual CortexA9 MPCore
    - 667MHz, L1/L2 cache, FPU, NEON, cache coherency, etc.
  - Common peripherals
    - SPI, I2C, ...
  - Memory interfaces
    - Flash, DRAM
- Programmable Logic
  - FPGA: Xilinx Artix-7
  - Hardware accelerators
  - Custom peripherals
  - AMBA interconnect (AXI)
Hardware platform configuration

- Xilinx Vivado

  - Processing System
    - Application processor
      - Cortex A9 MPCore @667MHz
    - 32KB I + 32 KB D Cache
    - 512 KB L2 Cache
    - 256 KB on-chip memory
    - NEON/FPU
    - MMU
    - Snoop control Unit (cache coherency)
  - Common peripherals
    - UART, USB, GPIO, Enet, SD (Flash),
  - Programmable Logic
    - Hardware accelerators
    - Custom peripherals
      - Block RAM
    - GPIO LEDs
Xilinx Zynq-7000 Linux

- Introduction
- Target platform: Xilinx Zynq-7000
  - Platform configuration
  - Kernel configuration and compilation
  - Kernel startup and execution
Hardware platform configuration

- Xilinx Vivado
  - Used for all hardware developments

The following basic configuration is used for our labs:
- Processing System
  - Standard configuration
  - 32-bit General Purpose AXI Master port
- Programmable Logic
  - Block RAM
  - GPIO
  - LED_8Bits
- AXI Interconnect

© Copyright 2013 Xilinx
XILINX ALL PROGRAMMABLE.
Hardware platform configuration

- **Xilinx Vivado**
  - Used for all hardware developments

The following basic configuration is used for our labs:

- **Processing System**
  - Standard configuration
  - 32-bit General Purpose AXI Master port

- **Programmable Logic**
  - Block RAM
  - GPIO
  - LED_8Bits

- **AXI Interconnect**
Software development

- Xilinx Software Development Kit (SDK)

Used for all software developments:
- Standalone application code
- Board Support Package
  - Files related to the hardware platform (required for instance for Linux kernel configuration)
- Kernel related developments
  - Board Support Package (BSP, device tree)
  - First Stage Boot Loader (FSBL)

- Hardware platform description is exported from Vivado to SDK to let further application code development
Xilinx Software Development Kit (SDK)

Kernel related developments:

- Board Support Package
  - A board support package is the implementation of specific support code for a given platform that conforms to a given operating system. It is commonly built with a bootloader that contains the minimal device support to load the operating system and device drivers for all the devices on the platform.

- Device Tree (DT)
  - Device Tree is a data structure and language for describing hardware layout. More specifically, it is a description of hardware that is readable by an operating system so that the operating system doesn't need to hard code details of the machine.
Kernel related developments:

- **First Stage Boot Loader (FSBL)**
  - When the computer is powered on, it typically does not have an operating system in RAM. The computer first executes a small program stored in persistent memory (ROM) along with a small amount of needed data, to access the nonvolatile device or devices from which the operating system programs and data can be loaded into RAM.
  - Generated with SDK

- **Second Stage Boot Loader (U-boot)**
  - Second-stage boot loaders (e.g. U-boot) are able to load an operating system properly and transfer execution to it.

![Software development diagram](image)
Software development

- Kernel related developments
  - Requiring Xilinx SDK:
    - Board Support Package
      - Device Tree
    - First Stage Boot Loader
  - Requiring Xilinx ARM Linux cross compiler
    - U-Boot
    - Filesystem (RAMDISK)
    - Kernel configuration and compilation
    - Application and driver development
Xilinx Zynq-7000 Linux

- Introduction
- Target platform: Xilinx Zynq-7000
  - Platform configuration
  - Kernel configuration and compilation
  - Kernel startup and execution
U-Boot

Das U-Boot (Universal Bootloader) is an open source boot loader used in embedded devices.

Ramdisk

- a fraction of the RAM memory that can be used as a virtual disk.
- temporary Filesystem solution to test quickly a Linux system.

Xilinx Design Flow

SDK: Build & Compile Application Code

SDK/HSI: Create FSL

Vivado IPI: Configure PS
  - Develop RTL/IP
  - Add/Integrate IP
  - Generate Bitstream
  - Export to SDK

Install Xilinx Tools

System Design

Hardware Design

System Software Development

Hardware handoff (hdif)

SDK/Bootgen: Create Boot Image

Bitstream FSL

Build U-boot

Build Linux

Create RamDisk

Fetch Sources: U-boot, Linux, RamDisk

Build U-boot

u-boot

uimage

uRamDisk

Booting and Running Linux
Target Platform

Application Code Binary

Polytech’Nice Sophia - Département Electronique - Université de Nice Sophia Antipolis - S. Bilavarn - 19 -
Kernel options configuration

- The kernel first requires information on the hardware platform
  - File: xilinx.dts
  - arch/arm/boot/dts/

- Make a default configuration:
  $ make xilinx_zynq_defconfig
  - Produces a working default configuration for a Zynq platform.

- Kernel options configuration:
  $ make menuconfig
  - Check kernel options for RAMDISK

When leaving the kernel configuration utility, a .config file is produced.
Description

- A Linux file system (Root Filesystem)
  - Standard Linux file hierarchy: /bin /dev /etc /home...

- Utilities
  - Basic commands: ls cd more cp...

Generation of a RAMDISK

- Busybox
  - Provided in some distributions
Compilation takes 5 to 10 minutes

**uImage** is to the kernel image produced

- Kernel image will be loaded at address **0x8000** in system RAM at boot time

**Xilinx.dtb** is the device tree generated from **xilinx.dts**

```
$ make ARCH=arm uImage
xilinx.dtb LOADADDR=0x8000
```
Xilinx Zynq-7000 Linux

- Introduction
- Target platform: Xilinx Zynq-7000
- Platform configuration
- Kernel configuration and compilation
- Kernel startup and execution
Kernel execution

**SD Card setup**
- **BOOT partition (FAT32)**
  - `BOOT.bin`, `uImage`, `devicetree.dtb`, `uramdisk_image.gz`
- **Rootfs partition (ext2)**
  - Put your files here
  - `mount /dev/mmcblk0p2 /media`

**Platform setup**
- **Switches**
- **Serial terminal**
  - baud rate = 115200, data bits = 8, stop bits = 1, flow control = none

JP9 – JP11 setup to boot from SD
JP6 shorted to allow SD boot

Serial cable
Kernel execution

ROM startup code
- Architecture dependent initialization

Bootloader
- Loads kernel to RAM and starts it

Kernel execution
- Initializes hardware devices and kernel subsystems
- Mounts the root Filesystem
- Starts the init process

/sbin/init
- starts other userspace services and applications

shell
- other application

Root Filesystem
Kernel execution

ROM startup code
Architecture dependent initialization

Bootloader
Loads kernel to RAM and starts it

Kernel execution
Initializes hardware devices and kernel subsystems
Mounts the root Filesystem
Starts the init process

/sbin/init
starts other userspace services and applications

shell
other application

Root Filesystem
Kernel execution

ROM startup code
Architecture dependent initialization

Bootloader
Loads kernel to RAM and starts it

Kernel execution
Initializes hardware devices and kernel subsystems
Mounts the root Filesystem
Starts the init process

/sbin/init
starts other userspace services and applications

shell
other application

Root Filesystem
What is a Linux initial RAM disk (*initrd*)? 

In a standard Linux system:

- Historically, the initial ramdisk was a temporary file system mounted and used *before* the final Root Filesystem was available.
- The reason of this is that when a system is switched on, there is not even a driver able to read the hard drive where all the drivers are.
- The initial ramdisk is associated to the kernel. Loading the initial ramdisk is included in the boot process of the kernel.
- It is composed of directories, drivers and binaries that will allow hardware setup, and especially to mount the final Root Filesystem.
- It can be seen as a small (temporary) *file system* containing key applications (like *nash script interpreter*, *insmod module loader*, *lvm logical volume manager*) that are needed to fully boot the Linux kernel with the Root Filesystem.
Root Filesystem: RAMDISK

Why using an initial RAM disk for the Root Filesystem?

- In an embedded system
  - Embedded systems usually do not include hard disk drives.
  - A RAMDISK is a fraction of the RAM memory that can be used as a virtual disk.
  - Consequently, the Root Filesystem can be implemented using a RAMDISK.

- Benefits
  - Access
  - Easy to setup

- Drawbacks
  - Volatile memory.
  - Less RAM memory available for users and applications.

- Interest for Linux systems
  - Provides a temporary solution for the root Filesystem, allowing to test easily and quickly a Linux system at an early stage of development.
Root Filesystem: Compact FLASH

Why using a Compact FLASH for the Root Filesystem?

- Compact device and easy to setup.

Benefits
- Non volatile
- Easy to setup
- Suited to a Linux system at final stages of development

Drawbacks
- Relatively slow
- Not easy to use for a system under current development (still easier than a RAMDISK).
Root Filesystem: Network filesystem

- **NFS**
  - Filesystem is placed on a distant machine, usually the host, and accessed via the network.
    - The developer can directly cross-compile applications to the Root Filesystem (no need to copy the binaries to the filesystem each time an application is compiled).

- **Benefits**
  - Non volatile
  - Very useful for systems in early or current development stages

- **Drawbacks**
  - Requires network
  - More difficult to setup (compared to a Compact FLASH)