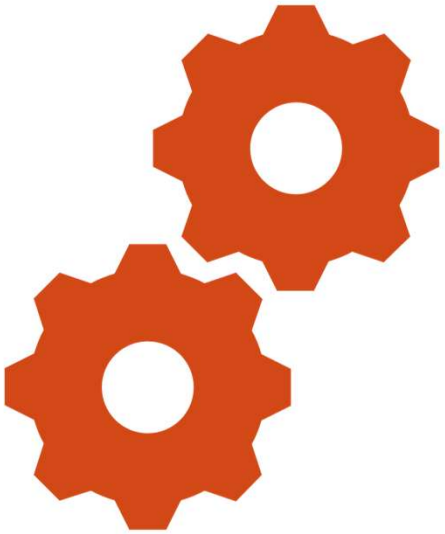
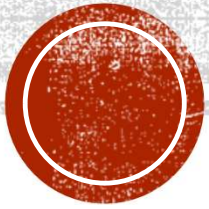


EMBEDDED LINUX SYSTEM DEVELOPMENT WITH QEMU

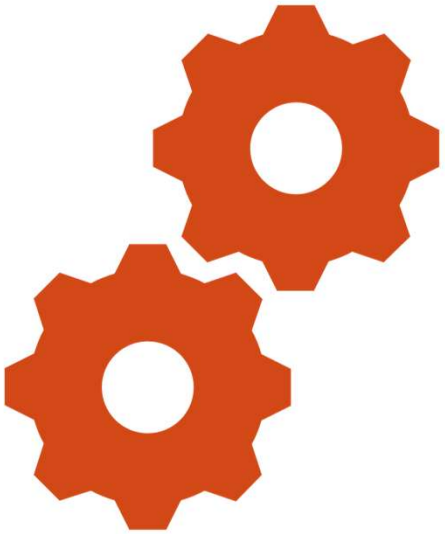


BHARATH G

Senior Lead Engineer | Collins Aerospace

- 9+ Years of Experience in Firmware Development
- Medical, Aerospace and IoT device Design
- Embedded C++ | RTOS | Bluetooth Low Energy | Embedded Linux |Technical Training | Career Guidance

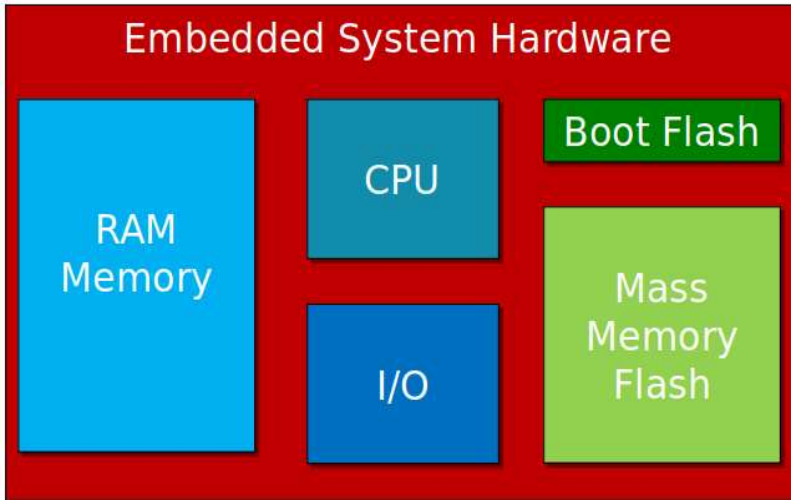
TABLE OF CONTENTS



EMBEDDED SYSTEM COMPONENTS

Hardware platform of Embedded Systems

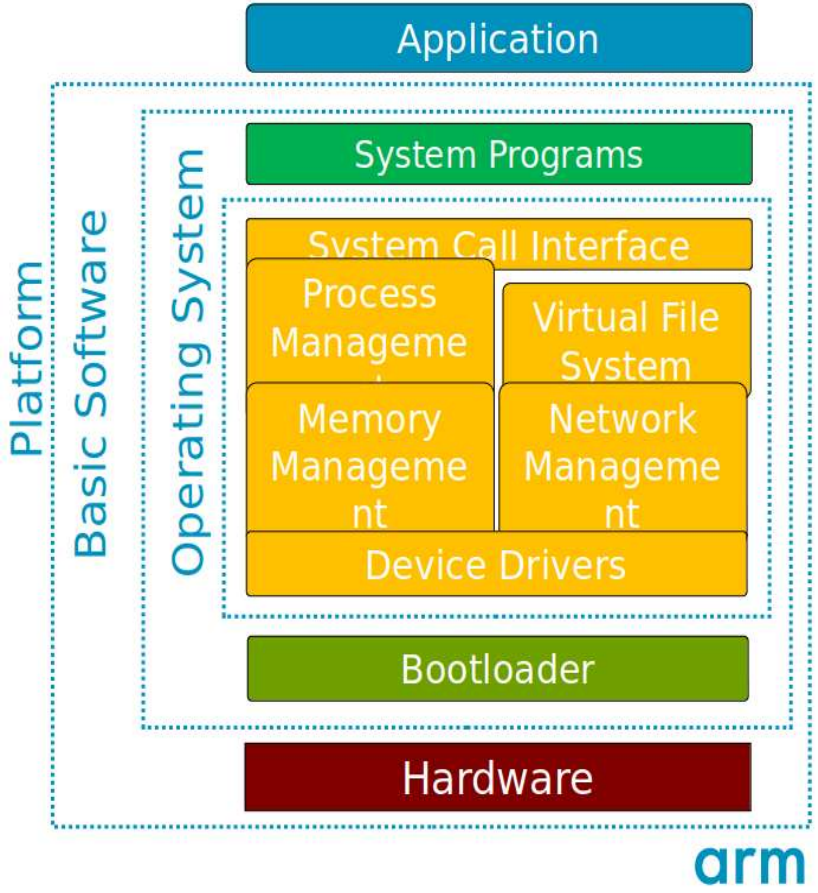
- RAM memory: volatile memory storing data/code
- CPU: processor running software
- I/O: peripherals to get inputs from the user, and to provide outputs to the user
- Boot Flash: small non-volatile memory needed at power-up
- Mass Memory Flash: large non-volatile memory



Source: Embedded Linux – Arm®

Linux Based Embedded System Components

- **Application**
  - Software that implements the functionalities for which the embedded system is intended (e.g., to control an Internal Combustion Engine)
- **Platform**
  - Combination of hardware and basic software components that provides the services needed for the application to run
  - Basic software may include system programs, operating system, bootloader

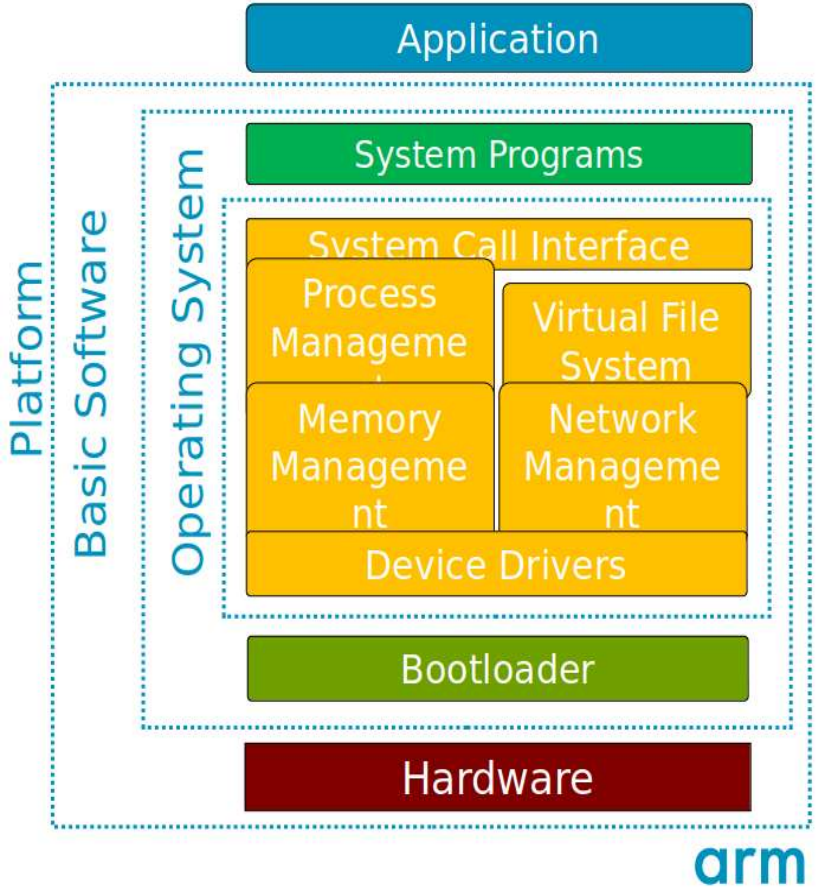


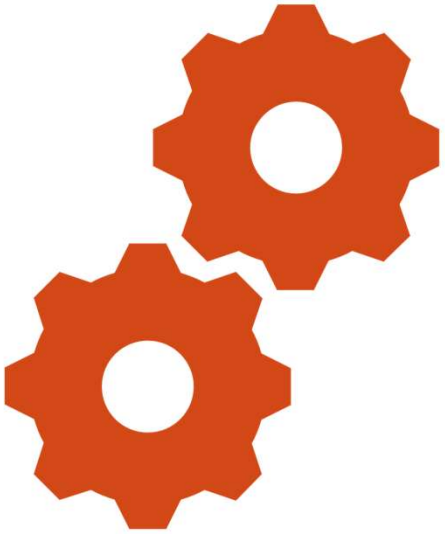
Source: Embedded Linux – Arm®

Linux Based Embedded System Components

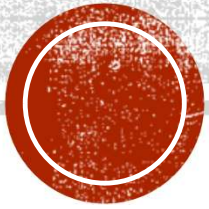
- **Bootloader:** Software executed at power-up to set-up the hardware to run the operating system
- **Device tree:** A tree data structure with nodes that describe the physical devices in the hardware needed by the Linux kernel to initialize properly the device drivers
- **Linux Kernel:** The operating system code providing all the services to manage the hardware resources
- **Root filesystem:** Container for the Linux Kernel configuration files, the system programs, and the application
- **System programs:** User-friendly utilities to access operating system services
- **Application:** Software implementing the functionalities to be delivered to the embedded system user

Source: Embedded Linux – Arm®



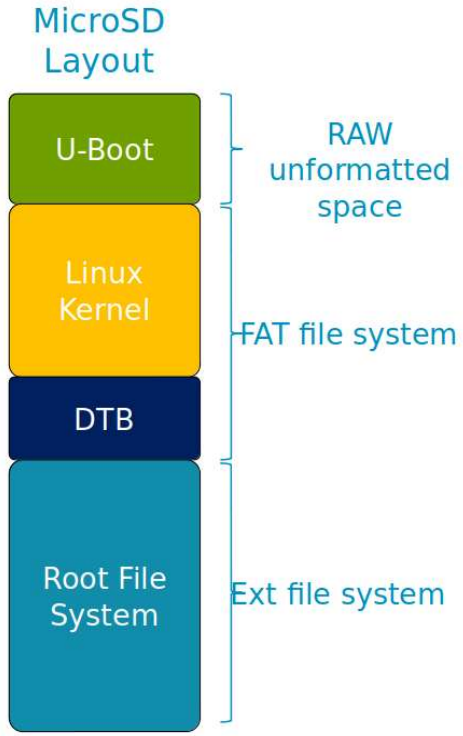


EMBEDDED LINUX & ITS COMPONENTS



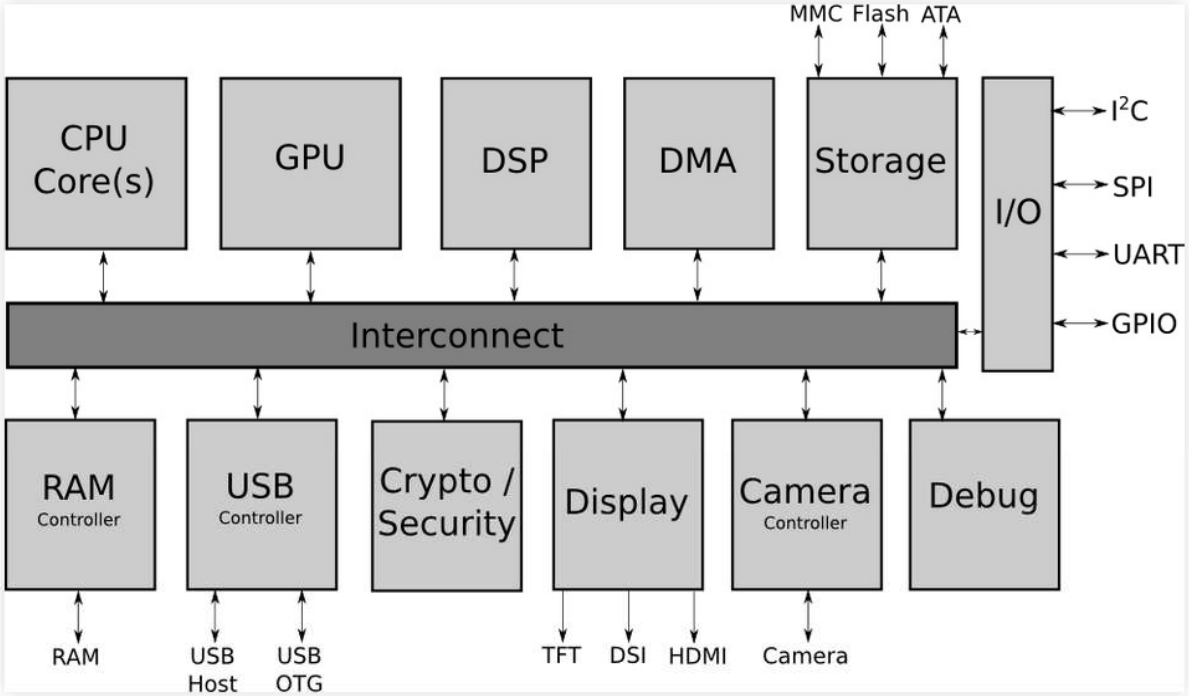
Components required to build Embedded Linux on target

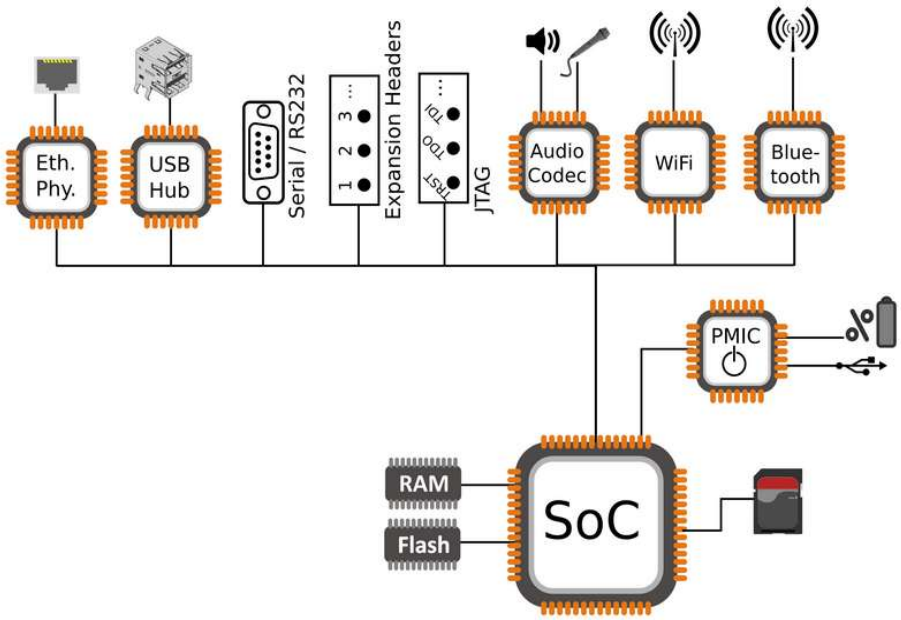
- **An embedded Linux system requires the following components to operate:**
  - The bootloader
  - The Linux Kernel
  - The device tree blob
  - The Root File System
- **All these components shall be:**
  - Configured for the embedded system hardware platform
  - Compiled and linked into an executable format
  - Deployed into the embedded system persistent storage for booting and operations



Source: Embedded Linux – Arm®

Embedded Linux Target Hardware

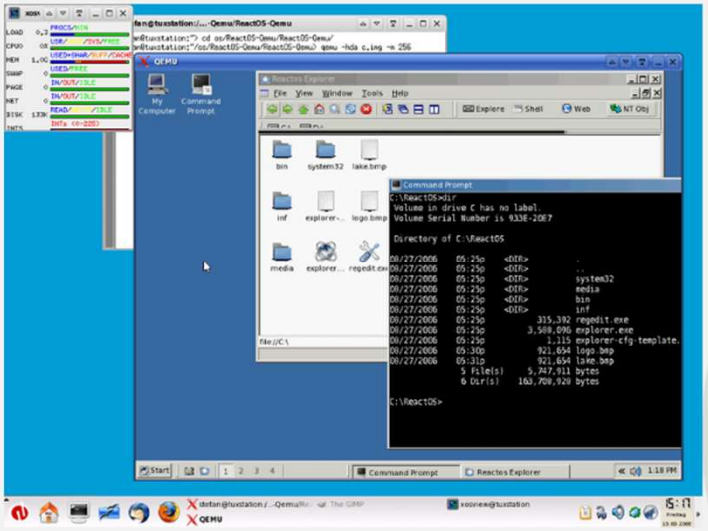




Source: Embedded Linux (opersys.com)

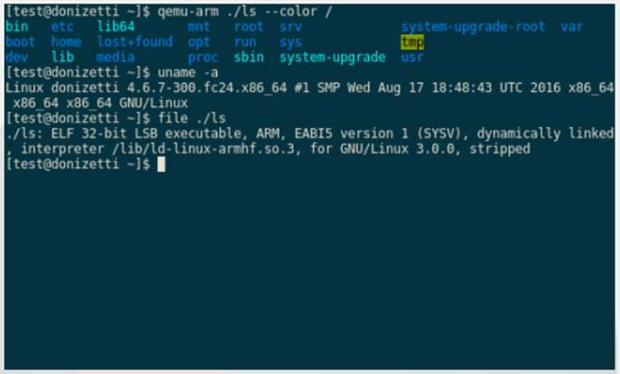
QEMU – Quick Emulator

Generic and open-source machine emulator and virtualizer



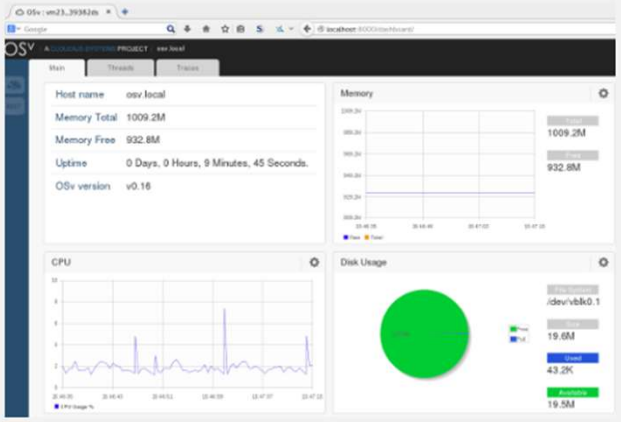
Full-system emulation

Run operating systems for any machine, on any supported architecture



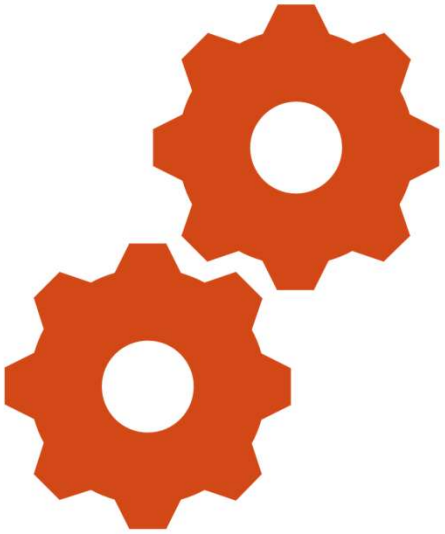
User-mode emulation

Run programs for another Linux/BSD target, on any supported architecture

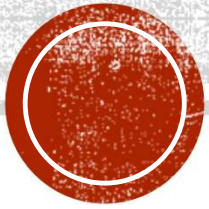


Virtualization

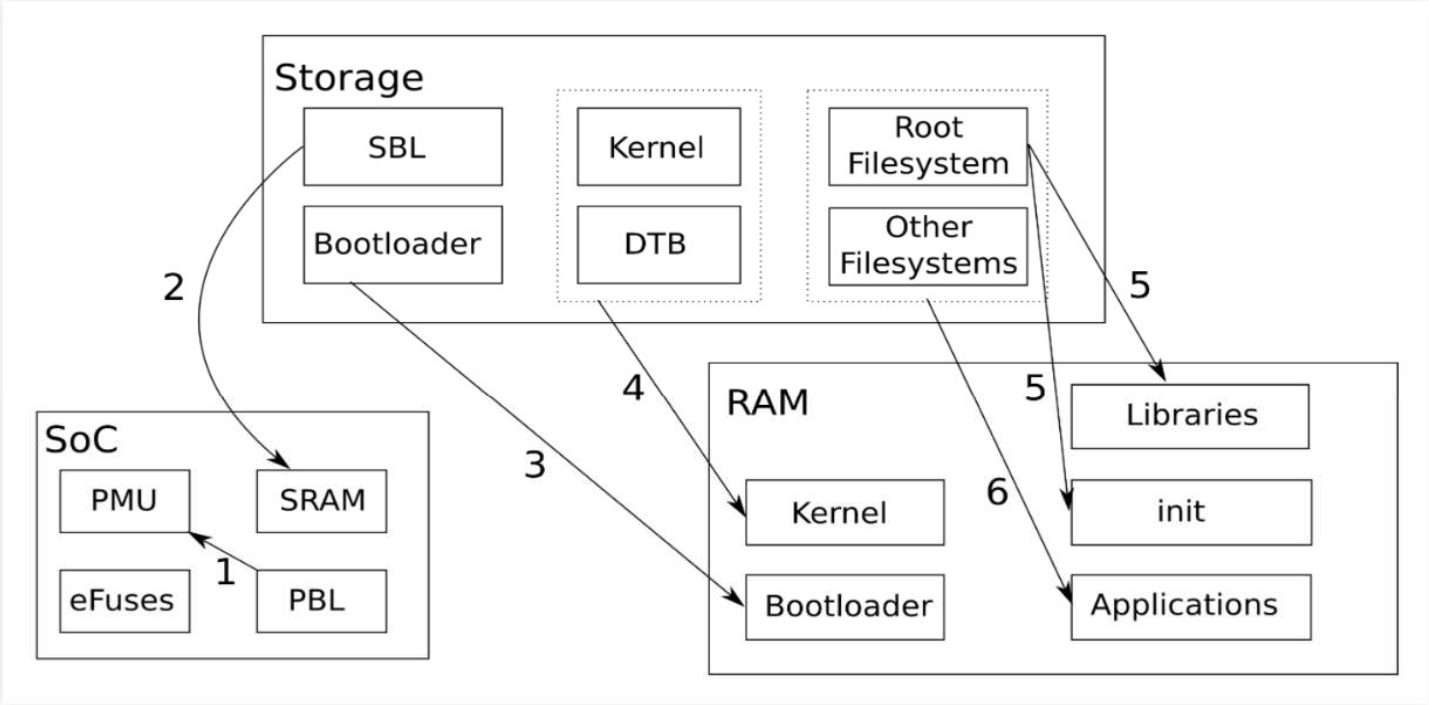
Run KVM and Xen virtual machines with near native performance



BOOTING LINUX KERNEL ON QEMU



Boot Sequence in Linux



Source: Embedded Linux (opersys.com)

Lab Setup for QEMU

Note: We are using Linux for the Demo & all the steps outlined here are for Linux OS only

- **Install QEMU**

```
sudo apt-get install qemu-system-arm
```

- **Download and extract the Linux Kernel from source**

```
wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.6.11.tar.xz
tar xvf linux-6.6.11.tar.xz
cd linux-6.6.11
```

- **Configure Kernel**

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- vexpress_defconfig
```

- **Build Kernel**

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
```

Note: The above step may take time upto 1 hour depending on your CPU speed

Let's Boot Linux With QEMU

- **Install the Toolchain**

```
sudo apt-get install gcc-arm-linux-gnueabi
```

- **Download and rename RootFS**

```
wget https://downloads.yoctoproject.org/releases/yocto/yocto-2.5/machines/qemu/qemuarm/core-image-minimal-qemuarm.ext4
mv core-image-minimal-qemuarm.ext4 rootfs.img
e2fsck –f rootfs.img
resize2fs rootfs.img 16M
```

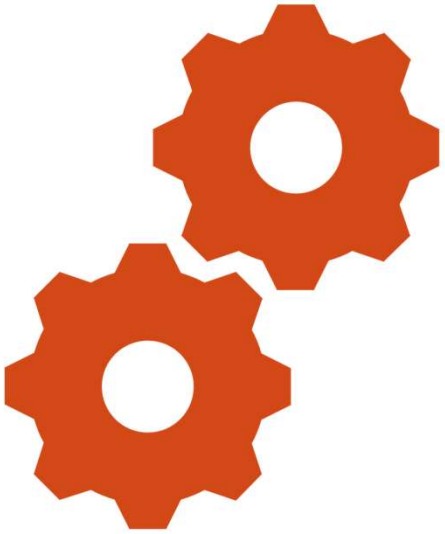
Alternatively, download pre-built binaries -
https://github.com/Bharathgopal/Simulators_Demo

- **Run Kernel on QEMU**

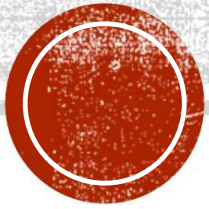
```
qemu-system-arm -M vexpress-a9 -kernel arch/arm/boot/zImage -dtb arch/arm/boot/dts/arm/vexpress-v2p-ca9.dtb -append "console=ttyAMA0,115200" –nographic
```

Verify the Booting of Linux on QEMU

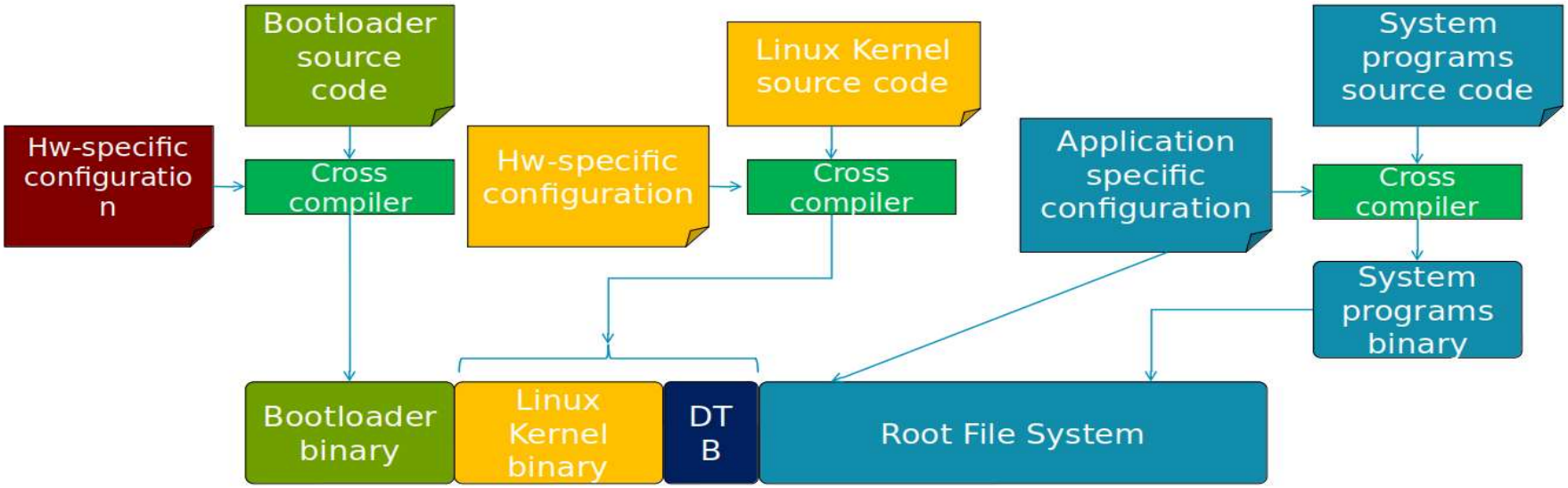
- **Run basic commands to check the correct bootup**
  - uname –a
  - whoami
  - date
  - time
  - ls /
  - Etc..



CROSS COMPILE & RUN APPLICATION ON QEMU



Cross compilation Workflow



arm

Source: Embedded Linux – Arm®

Cross compiling Application

- **Install the Toolchain**

```
sudo apt-get install gcc-arm-linux-gnueabi
```

- **Cross compile application**

```
arm-linux-gnueabi-gcc hello.c–o h1.out
```

- **Copy the Binary onto the target rootfs**

```
sudo mount –o loop,rw.sync rootfs.img /mnt/rootfs
sudo cp h1.out /mnt/rootfs/home/root
sudo umount /mnt/rootfs
```

- **Reboot QEMU**
- **Run the application**

```
./h1.out
```

```
#include “stdio.h”
int main()
{
   printf("Hello World\n");
   return 0;
}
```

Can we have the Library as part of Binary?

Analyze Binary file

Type of the File

file h1.out

ls –lh h1.out

Linked Libraries

ldd h1.out

Cross compiling Application – Static Binary

- **Cross compile application**

```
arm-linux-gnueabi-gcc hello.c–o h2.out -static
```

- **Analyze the file**
  - **Type of the File**
    - file h1.out h2.out
    - ls –lh h1.out h2.out
  - **Linked Libraries**
    - ldd h1.out
    - ldd h2.out
- **Copy the Binary onto the target rootfs**
- **Reboot QEMU**
- **Run the application**

```
./h2.out
```

```
#include "stdio.h"
int main()
{
   printf("Hello World\n");
   return 0;
}
```

How to do it for a Multifile Based Program?

Cross compiling Application – Multifile

- **Cross compile application**

```
arm-linux-gnueabi-gcc main.c
arm-linux-gnueabi-gcc file1.c
arm-linux-gnueabi-gcc file2.c
arm-linux-gnueabi-gcc main.o file1.o file2.o
```

- **Copy the Binary onto the target rootfs**
- **Reboot and run the application**

```
#include "stdio.h"
int main()
{
    int result1 = function1(10, 20);
    int result2 = function2(10, 20);
   printf("%d\n%d", result1, result2);
    return 0;
}
```

```
int function1(int a, int b)
{
   return a + b;
}
```

```
int function2(int a, int b)
{
   return a * b;
}
```

Cross compiling Application – Static Library

- **Cross compile application and generate object files**
- **Create a Static Library**

```
arm-linux-gnueabi-ar rc file1.o file2.o libtest1.a
```

- **Build Binary with Static Library**

```
arm-linux-gnueabi-gcc –L. main.o -ltest1 –o static_final.out
```

- **Copy the Binary onto the target rootfs**
- **Reboot and run the application**

What all should be copied to rootfs?

Cross compiling Application – Dynamic Library

- **Cross compile application and generate object files**
- **Create a Dynamic Library**

```
arm-linux-gnueabi-gcc –shared –o libtest2.so file1.o file2.o
```

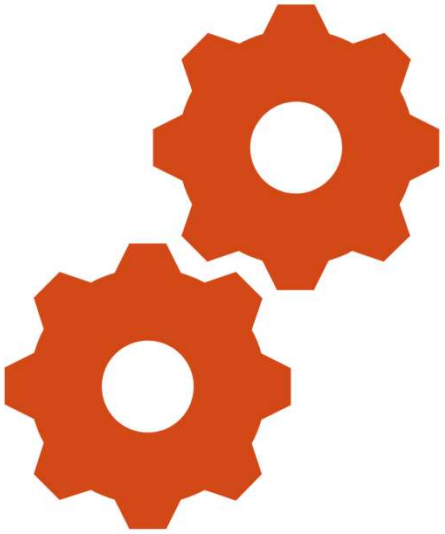
- **Build Binary with Dynamic Library**

```
arm-linux-gnueabi-gcc –L. main.o –ltest2 –o dynamic_final.out
```

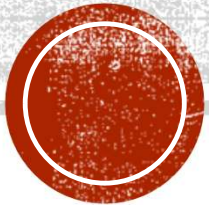
- **Reboot QEMU**
- **Run the application**

```
LD_LIBRARY_PATH=. ./dynamic_final.out
```

What all should be copied to rootfs?



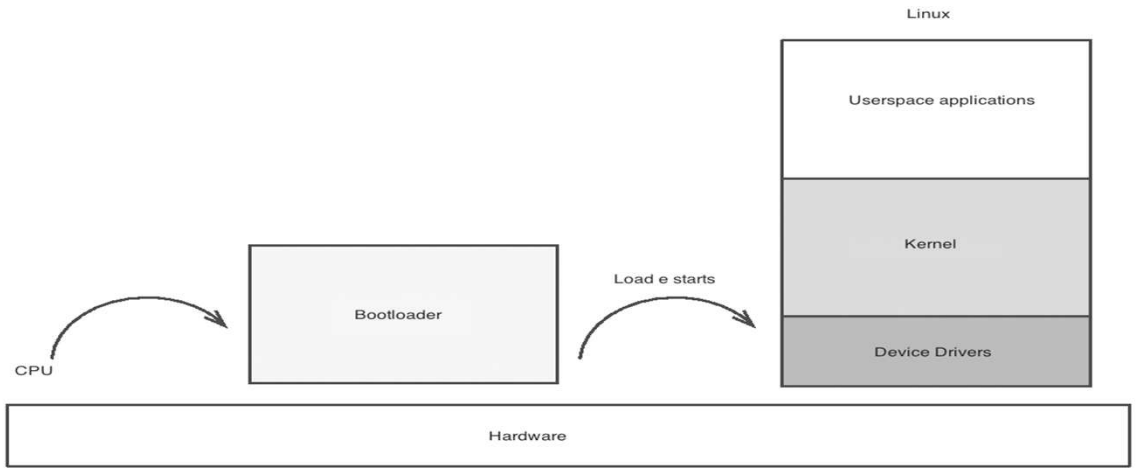
BOOTING TECHNIQUES IN EMBEDDED LINUX

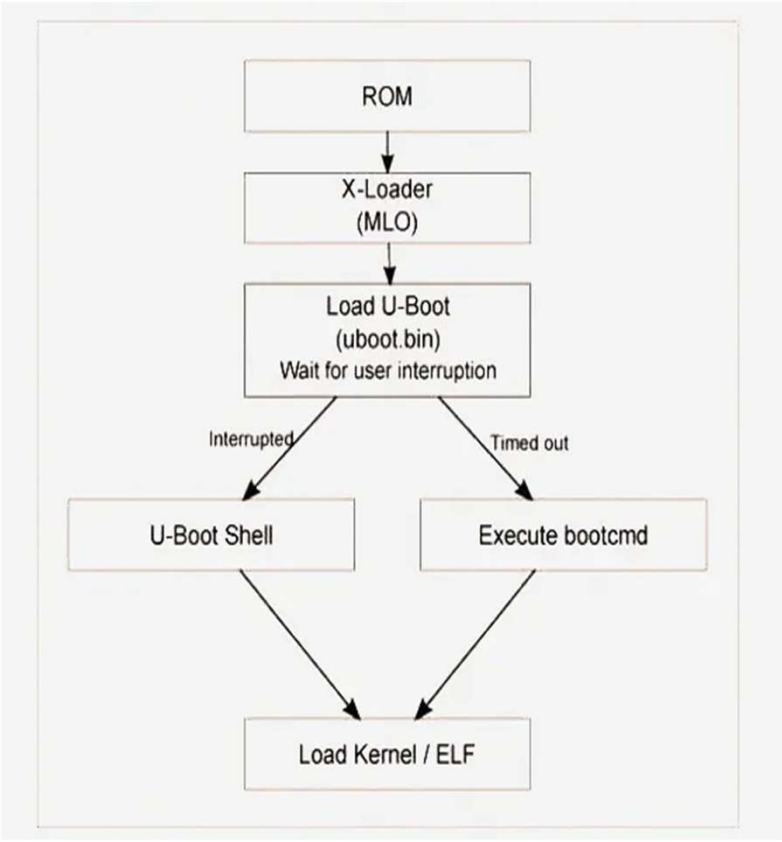


Booting Techniques

- **Storage Based Boot**
- **Network Boot**
  - **Serial or USB etc.**
  - **Wireless Communication ex: TFTP**

- **All the booting methods require a Boot loader**





Source: Embedded Linux Design: File System and Bootloader - EEWeb

NEXT STEPS

Resources to learn about QEMU

Welcome to QEMU's documentation! — QEMU documentation

Compiling linux kernel for qemu arm emulator GitHub

Q & A

STAY CONNECTED

- Bharathgopal75@gmail.com
- Linkedin.com/in/Bharathgopal
- LinkedIn- QR Code