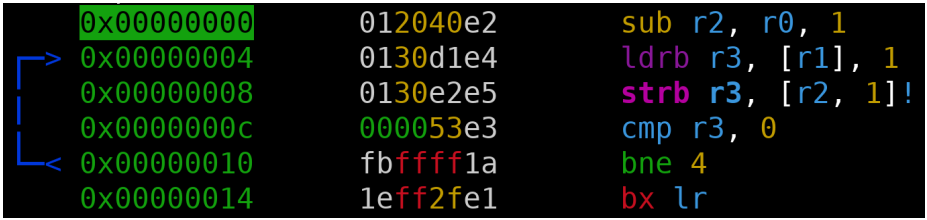
ENGR-E 399/599: Embedded Systems Reverse Engineering

Lecture 4: Introduction to embedded Linux systems

Austin Roach
ahroach@iu.edu

February 3, 2022

Mystery function



- How many arguments does the function take?
- What are the types of the arguments?
- What does the function do?
- What does the function return?

Mystery function revealed

```
char *strcpy(char *dest, const char *src)
```

Description

The `strcpy()` function copies the string pointed to by `src`, including the terminating null byte, to the buffer pointed to by `dest`. The strings may not overlap, and the destination string dest must be large enough to receive the copy. *Beware of buffer overruns!*

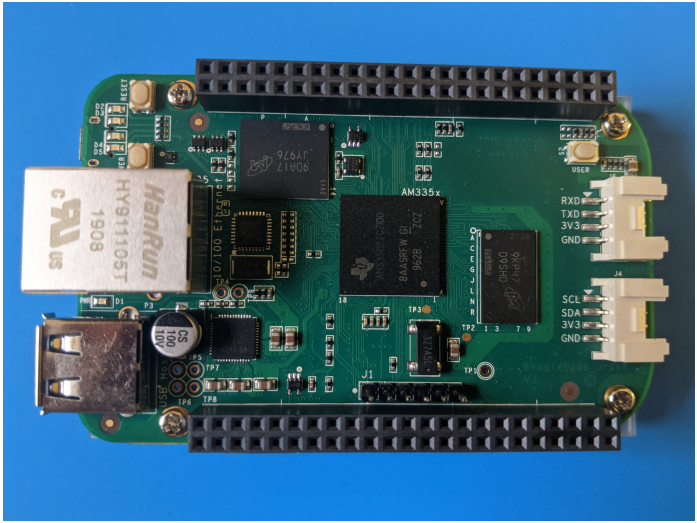
Return value

`strcpy()` returns a pointer to the destination string `dest`.

Today's plan

- Embedded Linux boot process
- Examining filesystem contents
- Emulating an embedded Linux system
- Basic interaction with an emulated system
- Reverse-engineering a system service
- Assignment 2 introduction

BeagleBone Green



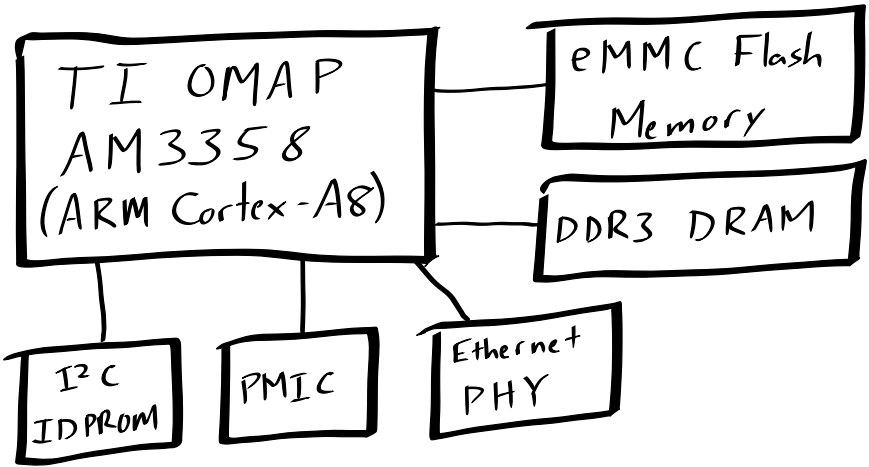
What are we doing with it?

Awesome embedded systems RE tool!

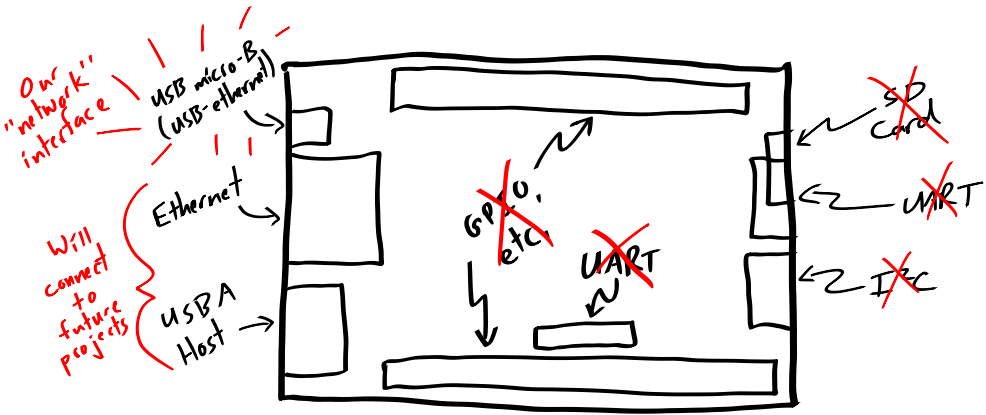
- Loaded with software for interacting with other projects
  - Eliminate OS compatibility issues
  - Eliminate awkward VM hardware passthrough problems
- Ethernet port and USB A host port – no dongles needed!

Today: well-documented example of an (almost) embedded Linux system

BeagleBone Green Architecture



BeagleBone Green Interfaces



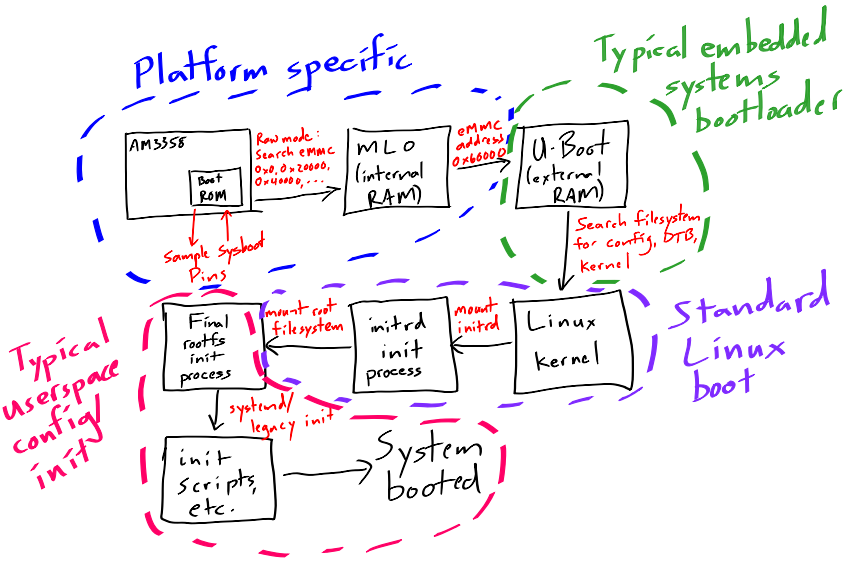
Firmware update file examination

```
$ file BBG-fwupd-2019-12-13.unxz
BBG-fwupd-2019-12-13.unxz: DOS/MBR boot sector; partition 1 : ID=0x83, active,
start-CHS (0x0,130,3), end-CHS (0x1ca,238,36), startsector 8192, 7364608 sectors

$ xxd -a BBG-fwupd-2019-12-13.unxz | less
...
```

Examine with binwalk

```
$ binwalk BBG-fwupd-2019-12-13.unxz

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
202468        0x316E4         CRC32 polynomial table, little endian
206825        0x327E9         Unix path: /lib/firmware/BB-BONE-eMMC1-01-00A0.dtbo
398592        0x61500         CRC32 polynomial table, little endian
627176        0x991E8         device tree image (dtb)
641984        0x9CBC0         SHA256 hash constants, little endian
694259        0xA97F3         Android bootimg, kernel size: 1684947200 bytes,
                              kernel addr: 0x64696F72, ramdisk size: 1763734311 bytes,
                              ramdisk addr: 0x6567616D,
                              product name: "ddr 0x%08x size %u KiB"
709226        0xAD26A         Unix path: /home/userid/targetNFS
734923        0xB36CB         Unix path: /lib/firmware/BB-BONE-eMMC1-01-00A0.dtbo
759037        0xB94FD         LZO compressed data
4194304       0x400000        Linux EXT filesystem, blocks count: 920576,
                              image size: 942669824, rev 1.0, ext4 filesystem data,
                              UUID=ce2457a0-4d2f-4de1-a186-85e98c938c93, volume name "rootfs"
```

Boot process

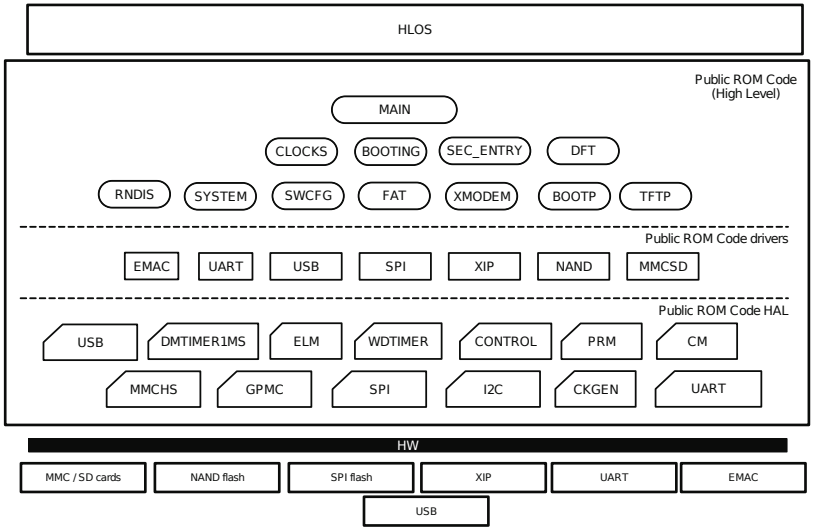


Platform specific references

AM335x Technical Reference Manual:
http://www.ti.com/lit/pdf/spruh73

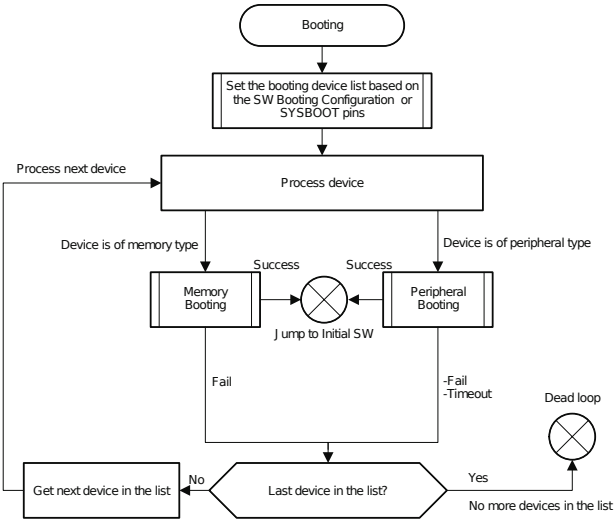
AM335X TRM: BootROM Architecture

Figure 26-1. Public ROM Code Architecture



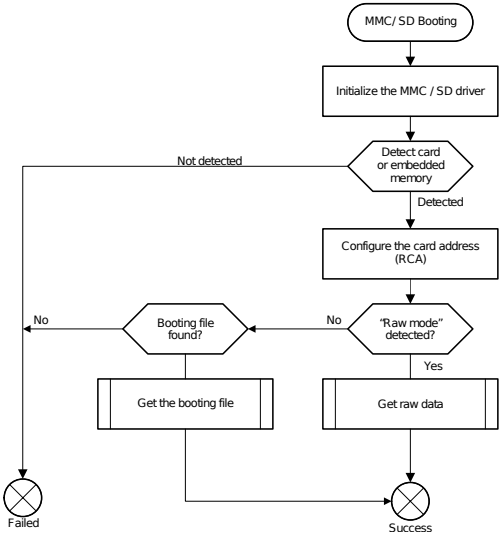
AM335X TRM: BootROM Flowchart

Figure 26-10. ROM Code Booting Procedure



AM335X TRM: MMC Boot

Figure 26-22. MMC/SD Booting



26.1.8.5.5 MMC/SD Read Sector Procedure in Raw Mode

In raw mode the booting image can be located at one of the four consecutive locations in the main area: offset 0x0 / 0x20000 (128KB) / 0x40000 (256KB) / 0x60000 (384KB). For this reason, a booting image shall not exceed 128KB in size. However it is possible to flash a device with an image greater than 128KB starting at one of the aforementioned locations. Therefore the ROM Code does not check the image size. The only drawback is that the image will cross the subsequent image boundary.

The ROM will check the first sector (offset 0x0) for the presence of the TOC structure as described in Section 26.1.10. If the sector contains a valid CHSETTINGS item, the ROM reads the GP header in the next 512-byte section, uses its size and destination information to download the image to the destination address, and jumps to the destination address to complete the boot. If the TOC structure is missing or invalid, the ROM checks for a redundant image in the next location (0x20000), and repeats the process to find a valid TOC. The ROM continues to attempt to boot from each of the 4 redundant locations as mentioned above. The TOC is considered invalid if the first location is either 0xFFFFFFFF or 0x00000000 (which is typical of a missing card or an erased image), or if the CHSETTINGS item is missing or corrupt.

AM335X TRM: MMC Raw Boot TOC

Table 26-38. The TOC Item Fields

Offset	Field	Size (bytes)	Description
0000h	Start	4	0x00000040
0004h	Size	4	0x0000000C
0008h	Flags	4	Not used, should be zero.
000Ch	Align	4	Not used, should be zero.
0010h	Load Address	4	Not used, should be zero.
0014h	Filename	12	12 character long name of sub image, including the zero ('\0') terminator. The ASCII representation is "CHSETTINGS".

Table 26-39. Magic Values for MMC RAW Mode

Offset	Value
40h	0xC0C0C0C1
44h	0x00000100

The ROM Code recognizes the TOC based on the filename described in Table 26-40.

Table 26-40. Filenames in TOC for GP Device

Filename	Description
CHSETTINGS	Magic string used by ROM

AM335X TRM: MMC Raw Boot GP headers

Table 26-37. GP Device Image Format

Field	Non-XIP Device Offset	XIP Device Offset	Size[bytes]	Description
Size	0000h	-	4	Size of the image
Destination	0004h	-	4	Address where to store the image / code entry point
Image	0008h	0000h	x	Executable code

MMC raw boot TOC

```
$ xxd -a -s 0x20000 -l 0x260 BBG-fwupd-2019-12-13.unxz
00020000: 4000 0000 0c00 0000 0000 0000 0000 0000  @...............
00020010: 0000 0000 4348 5345 5454 494e 4753 0000  ....CHSETTINGS..
00020020: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00020030: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00020040: c1c0 c0c0 0001 0000 0000 0000 0000 0000  ................
00020050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
*
00020200: 5468 0100 0004 2f40 0f00 00ea 14f0 9fe5  Th..../@........
00020210: 14f0 9fe5 14f0 9fe5 14f0 9fe5 14f0 9fe5  ................
00020220: 14f0 9fe5 14f0 9fe5 4004 2f40 4004 2f40  ........@./@@./@
00020230: 4004 2f40 4004 2f40 4004 2f40 4004 2f40  @./@@./@@./@@./@
00020240: 4004 2f40 efbe adde feff ffea 3300 00ea  @./@........3...
00020250: 0000 0fe1 1f10 00e2 1a00 31e3 1f00 c013  ..........1.....
```

U-Boot references

Official references:

- https://www.denx.de/wiki/U-Boot
- https://github.com/u-boot/u-boot

But many, many unofficial guides

Examine U-Boot

```
$ xxd -a -s 0x60000 -l 0xa0 BBG-fwupd-2019-12-13.unxz
00060000: 2705 1956 7e6f 23e7 5dbc 3dd1 0007 4598  '..V~o#.].=...E.
00060010: 8080 0000 0000 0000 650d 5e3a 1102 0500  ........e.^:....
00060020: 552d 426f 6f74 2032 3031 392e 3034 2d30  U-Boot 2019.04-0
00060030: 3030 3033 2d67 6637 3537 3737 6461 3163  0003-gf75777da1c
00060040: b800 00ea 14f0 9fe5 14f0 9fe5 14f0 9fe5  ................
00060050: 14f0 9fe5 14f0 9fe5 14f0 9fe5 14f0 9fe5  ................
00060060: 6000 8080 c000 8080 2001 8080 8001 8080  `....... .......
00060070: e001 8080 4002 8080 a002 8080 efbe adde  ....@...........
00060080: dec0 ad0b 00f0 20e3 00f0 20e3 00f0 20e3  ...... ... ... .
00060090: 00f0 20e3 00f0 20e3 00f0 20e3 00f0 20e3  .. ... ... ... .
```

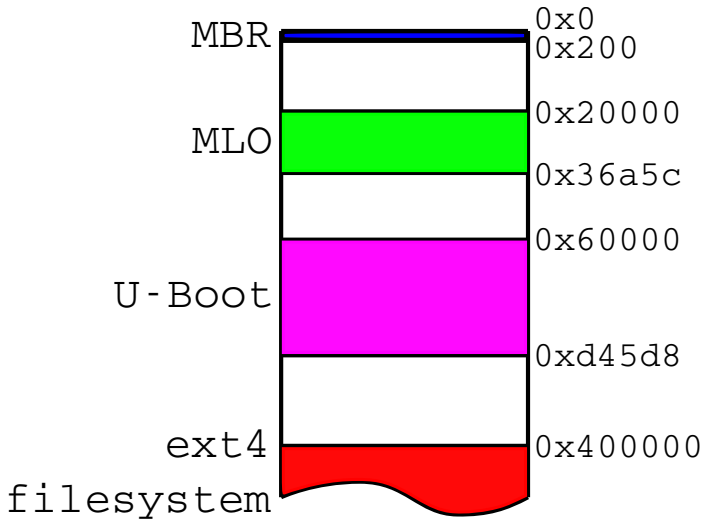
uimage dumpimage

```
$ carve BBG-fwupd-2019-12-13.unxz 0x60000 +0x745d8 > uimage
$ dumpimage -l uimage
Image Name:   U-Boot 2019.04-00003-gf75777da1c
Created:      Fri Nov  1 10:14:41 2019
Image Type:   ARM U-Boot Firmware (uncompressed)
Data Size:    476568 Bytes = 465.40 KiB = 0.45 MiB
Load Address: 80800000
Entry Point:  00000000
```

Linux boot references

References are innumerable, for example:
https://en.wikipedia.org/wiki/Linux_startup_process

eMMC layout


MBR
MLO
U-Boot
ext4
filesystem
0x0
0x200
0x20000
0x36a5c
0x60000
0xd45d8
0x400000

Filesystem examination

```
$ /sbin/fdisk -l BBG-fwupd-2019-12-13.unxz
Disk BBG-fwupd-2019-12-13.unxz: 3.53 GiB, 3774873600 bytes, 7372800 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xcc9ae073

Device                        Boot Start     End Sectors  Size Id Type
BBG-fwupd-2019-12-13.unxz1 *     8192 7372799 7364608  3.5G 83 Linux

$ sudo mount -o offset=0x400000,ro BBG-fwupd-2019-12-13.unxz /mnt/blah
```

Segue

From this point on, the Beaglebone Green is not typical of an embedded OS:

- Read-write filesystem
- Featureful set of userspace software
  - Editors
  - Compilers
  - Development files (header files and such)
  - Debuggers
  - Various language interpreters
  - Package manager

A more typical filesystem

SquashFS, (or CramFS, etc.)

- Compressed
  - Non-volatile memory is expensive
  - Many NVM buses are slow
    - ⋆ Compressed filesystem speeds up boot time
- Read-only
  - The firmware should change only rarely
  - Less chance of corruption with sudden power-off
  - Special NVRAM storage areas for configuration variables and the like

Typical embedded systems userspace

- uClibc (instead of GNU libc)
  - C standard library
  - Very small!
- BusyBox (instead of GNU coreutilities)
  - Multi-call binary providing standard Unix utilities
  - Very small!
- BusyBox init (instead of systemd)
  - Look at `/etc/inittab`
  - Often more init scripts linked from `/etc/rc*.d`

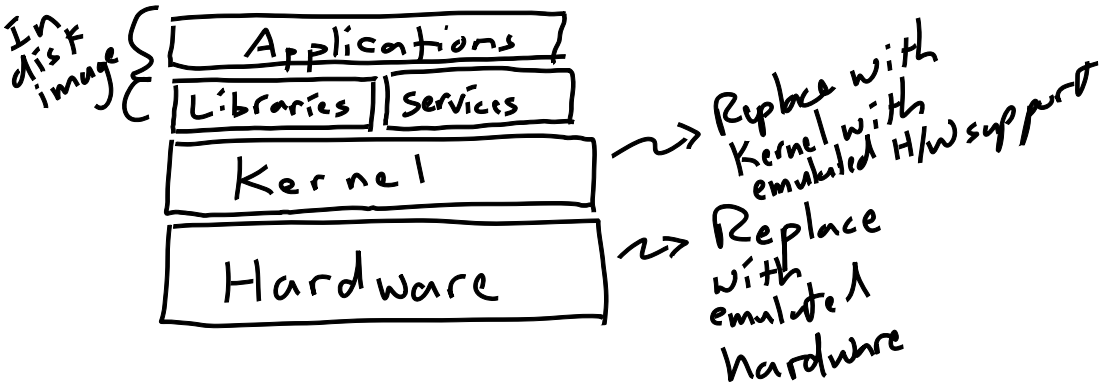
References:

- https://www.uclibc.org
- https://www.uclibc-ng.org
- https://busybox.net

Filesystem examination

```
$ unsquashfs demo-roots.squashfs
...
$ cat squashfs-root/etc/inittab
...
$ ls squashfs-root/etc/init.d
...
$ cat squashfs-root/etc/S99demo-service
...
$ file squashfs-root/usr/bin/demo-service
...
```

System abstraction layers



Emulation

Powerful approach for dynamically analyzing system operation:

- Replace physical system with emulated hardware
- Replace kernel with a kernel supporting emulated peripherals
- Execute software contained in filesystem image
- Allows greater access for examination than a physical system might provide
  - Rapidly modify filesystems and test
  - Low-level debug interfaces to emulated system
  - Opportunities for parallelization

QEMU is the go-to tool for full system emulation

QEMU system mode demonstration

```
$ sudo ./launch-emulated-system.sh demo-rootfs.squashfs
...

Welcome to the Embedded Systems RE demo system
esre login:
```

Interacting with network services

```
$ nmap -A -T4 -p- 169.254.15.2
...

Web server

$ nc 169.254.15.2 31330
...

Python sockets module
```

Now let's reverse-engineer a service...

Assignment 2 introduction