

U-Boot – Bootloader for IoT Platform?

Alexey Brodkin

Embedded Linux Conference Europe 2018, Edinburgh



Alexey Brodkin

Engineering manager at Synopsys

- Maintainer of ARC architecture in U-Boot
- Contributor to:
 - Linux kernel
 - Buildroot
 - OpenEmbedded
 - OpenWrt
 - uClibc etc

Agenda

U-Boot for IoT device

Shrinking memory footprint

Execution from ROM

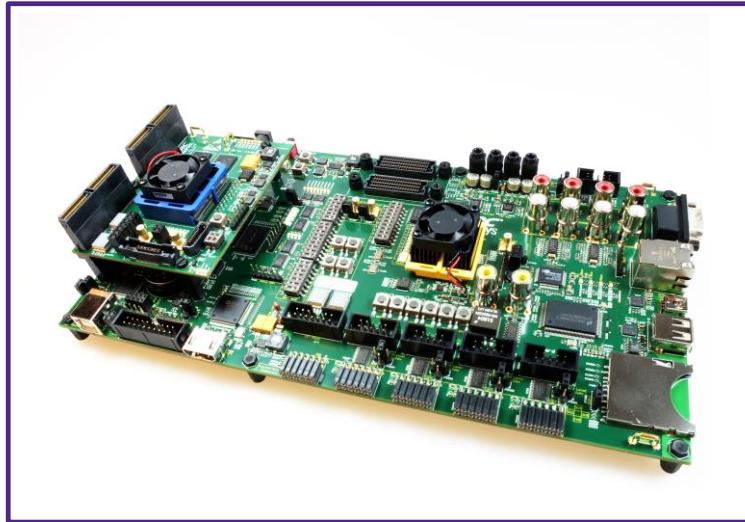
Run-time issues

Previous ARC boards with U-Boot

Single-board computers with Linux

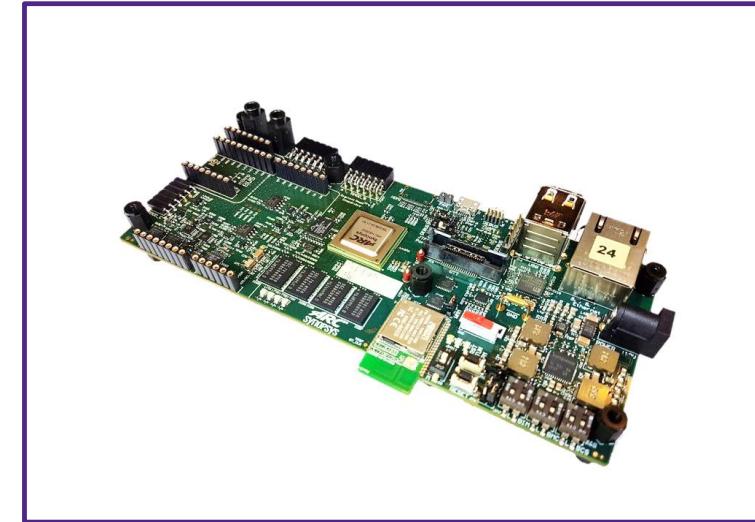
AXS103

- Dual-core ARC HS38 @ 100 MHz in FPGA
- BootROM
- 2 GiB of DDR



HSDK

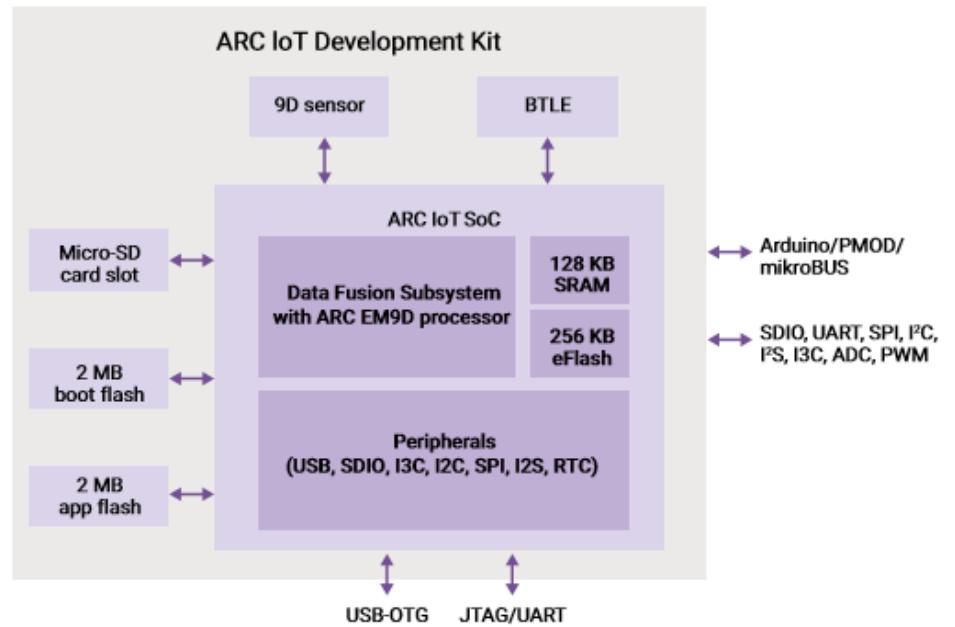
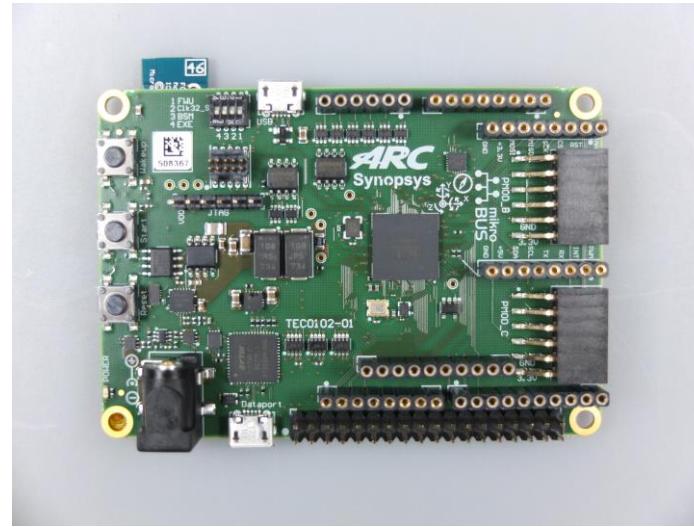
- Quad-core ARC HS38 @ 1 GHz in silicon
- BootROM
- 4 GiB of DDR



New board – new fun

Meet IoT development kit

- ARC EM9D @ 150 MHz
- Memories:
 - eFlash 256 KiB @ 0x0000_0000
 - ICCM 256 KiB @ 0x2000_0000
 - SRAM 128 KiB @ 0x3000_0000
 - DCCM 128 KiB @ 0x8000_0000
 - SPI flash 2 MiB
- Peripherals:
 - SD-card (DW MobileStorage)
 - USB OTG (DW USB OTG)
 - Serial port (DW APB UART)



* The board features many more peripherals but we are only listing those relevant to the bootloader here

Why U-Boot

Add support of a new board in the blink of an eye

- Mature and well-known bootloader
- Supports:
 - 12 CPU architectures
 - Lots of peripherals: UART, SPI, I2C, Ethernet, SD, USB ...
 - File-systems: Fat, Ext, Yaffs2, Btrfs, ubifs ...
 - Networking protocols: TFTP, NFS, DHCP
- “Device-tree” used for drivers initialization
- Allows for flexible scripting in “hush” shell
- Allows re-use of its stdio and C run-time libs by user applications

```
$ git show --stat --pretty=oneline 5396e8b
5396e8b arc: Add support for IoT development kit
 arch/arc/Kconfig | 5 +++
 arch/arc/dts/Makefile | 1 +
 arch/arc/dts/iot_devkit.dts | 45 ++++++=====
 board/synopsys/iot_devkit/Kconfig | 12 ++++++
 board/synopsys/iot_devkit/MAINTAINERS | 5 +++
 board/synopsys/iot_devkit/Makefile | 7 +////
 board/synopsys/iot_devkit/config.mk | 2 ++
 board/synopsys/iot_devkit/iot_devkit.c | 168 ++++++=====
 board/synopsys/iot_devkit/u-boot.lds | 77 ++++++=====
 configs/iot_devkit_defconfig | 38 ++++++=====
 include/configs/iot_devkit.h | 84 ++++++=====
 11 files changed, 444 insertions(+)
```

Starting point: 422 KiB total

- DW USB OTG & DW MMC drivers
- Read/write FAT file-system
- Built-in .dtb

```
$ size u-boot
text      data      bss      dec      hex filename
257129    9372   155928  422429  6721d u-boot
```

```
CONFIG_ARC=y
CONFIG_ISA_ARCV2=y
CONFIG_CPU_ARCEM6=y
CONFIG_TARGET_IOT_DEVKIT=y
CONFIG_CMD_MMC=y
CONFIG_CMD_USB=y
CONFIG_CMD_FAT=y
CONFIG_OF_CONTROL=y
CONFIG_OF_EMBED=y
CONFIG_ENV_IS_IN_FAT=y
CONFIG_ENV_FAT_INTERFACE="mmc"
CONFIG_ENV_FAT_DEVICE_AND_PART="0:1"
CONFIG_DM=y
CONFIG_MMCA=y
CONFIG_MMCA_DW=y
CONFIG_DM_SERIAL=y
CONFIG_SYS_NS16550=y
CONFIG_USB=y
CONFIG_DM_USB=y
CONFIG_USB_DWC2=y
CONFIG_USB_STORAGE=y
```

Shrinking memory footprint

We only have 256 KiB of ROM and 128 KiB of RAM

Disable useless options: 366 KiB total

As simple as deselecting items in *menuconfig*

- No plans to load OS
- No plans to use flash (as of now)
- There's no Ethernet controller
- No memory to load application from Elf
 - Load Elf
 - Copy sections from Elf to RAM
- No need to load via serial port

```
# CONFIG_CMD_BOOTD is not set
# CONFIG_CMD_BOOTM is not set
# CONFIG_CMD_ELF is not set
# CONFIG_CMD_XIMG is not set
# CONFIG_CMD_FLASH is not set
# CONFIG_CMD_LOADB is not set
# CONFIG_CMD LOADS is not set
# CONFIG_NET is not set
```

```
$ size u-boot
text      data      bss      dec      hex filename
216009    7544   142468  366021  595c5 u-boot
```

Get rid of dead code: 311 KiB total

Might require changes in sources

- Put all functions, global & static variables in their own sections and strip unused on final linkage.
- Should be enabled by default for all architectures and boards in U-Boot except toolchain doesn't support it.
- Was not the case for ARC – fixed now
[fac4790491f6 \("arc: Eliminate unused code and data with GCC's garbage collector"\)](#)

```
CPPFLAGS += -ffunction-sections -fdata-sections  
LDFLAGS += --gc-sections
```

```
$ size u-boot  
text      data      bss      dec      hex filename  
163532    6948   140928  311408  4c070 u-boot
```

Shrink statically allocated buffers: 188 KiB total

- tmpbuf_cluster & get_contents_vfatname_block of 65 KiB each!
- CONFIG_FS_FAT_MAX_CLUSTSIZE=4096
- Save 120 KiB of memory!

```
$ nm --size-sort --reverse-sort u-boot | head -n 5  
00010000 b tmpbuf_cluster  
00010000 B get_contents_vfatname_block  
00001414 b hist_lines  
00000a96 t do_fdt  
00000812 t set_contents  
  
#define MAX_CLUSTSIZE CONFIG_FS_FAT_MAX_CLUSTSIZE  
static __u8 tmpbuf_cluster[MAX_CLUSTSIZE]  
__aligned(ARCH_DMA_MINALIGN);  
__u8 get_contents_vfatname_block[MAX_CLUSTSIZE]
```

```
$ size u-boot  
text      data      bss      dec      hex filename  
163532    6948    18048   188528  2e070  u-boot
```

Compile-time optimizations summary: /2 size

Memory foot-print reduced from 422 to 188 KiB (saved 234 KiB)

- Analyze
 - Main contributors were huge statically allocated buffers
 - Primary tools:
 - bloat-o-meter (<https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/scripts/bloat-o-meter>)
 - size
 - nm
- Be practical
 - Unused options might add to memory usage significantly (56 KiB in our case)
- Use advanced features of the toolchain
 - 5% size reduction due to dead code elimination
 - Link Time Optimization (LTO) might help a bit more

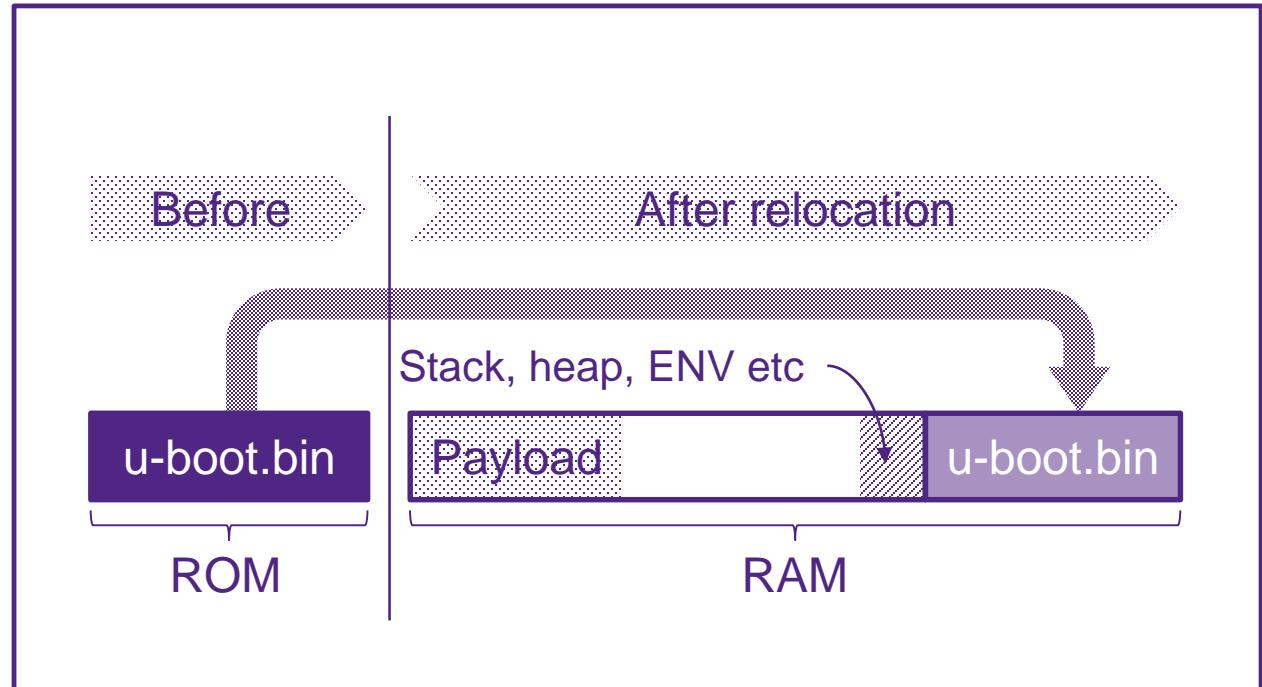
Execution from ROM

Relocation & memory partitioning

[Self-] relocation

Fundamental feature of U-Boot

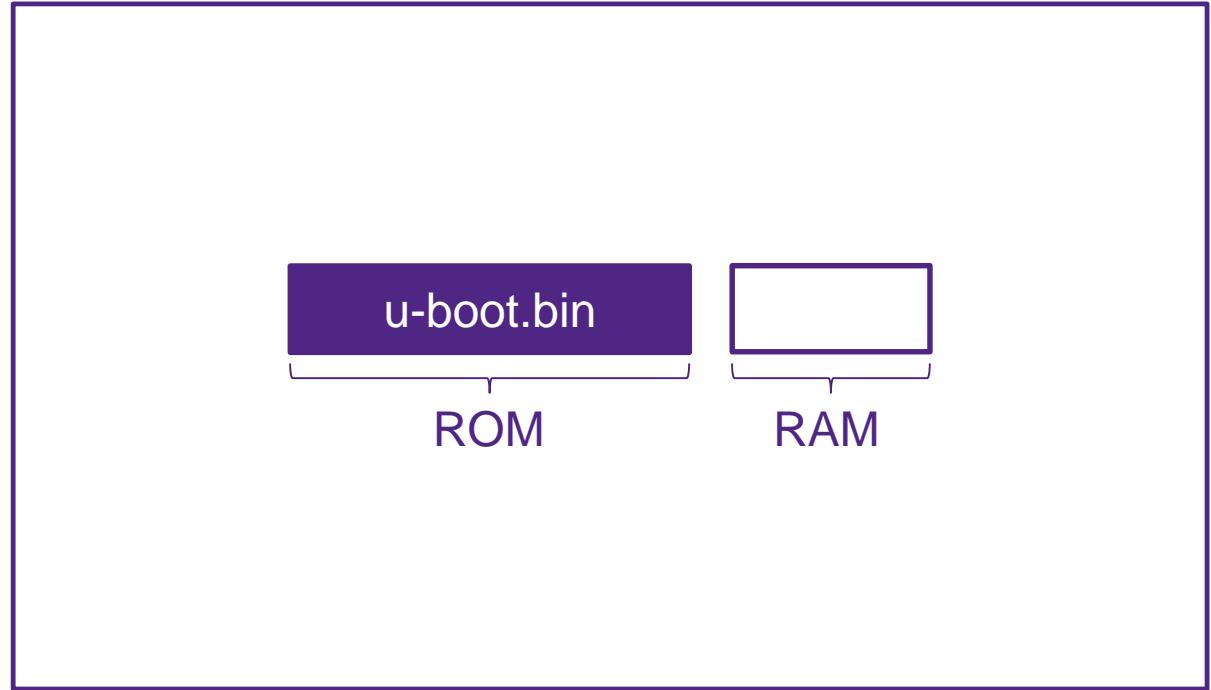
- Why relocation?
 - RAM is much larger
 - DDR might require initialization before use
 - We'll need RAM anyways so why not?
- 2 major stages:
 - Pre-relocation (`common/board_f.c`)
Execute code from ROM/flash with limited RAM options:
 - On-chip SRAM
 - Locked D\$ lines (x86)
 - DDR (sometimes)
 - After-relocation (`common/board_r.c`)
Executing from RAM (usually DDR)



Skip relocation*

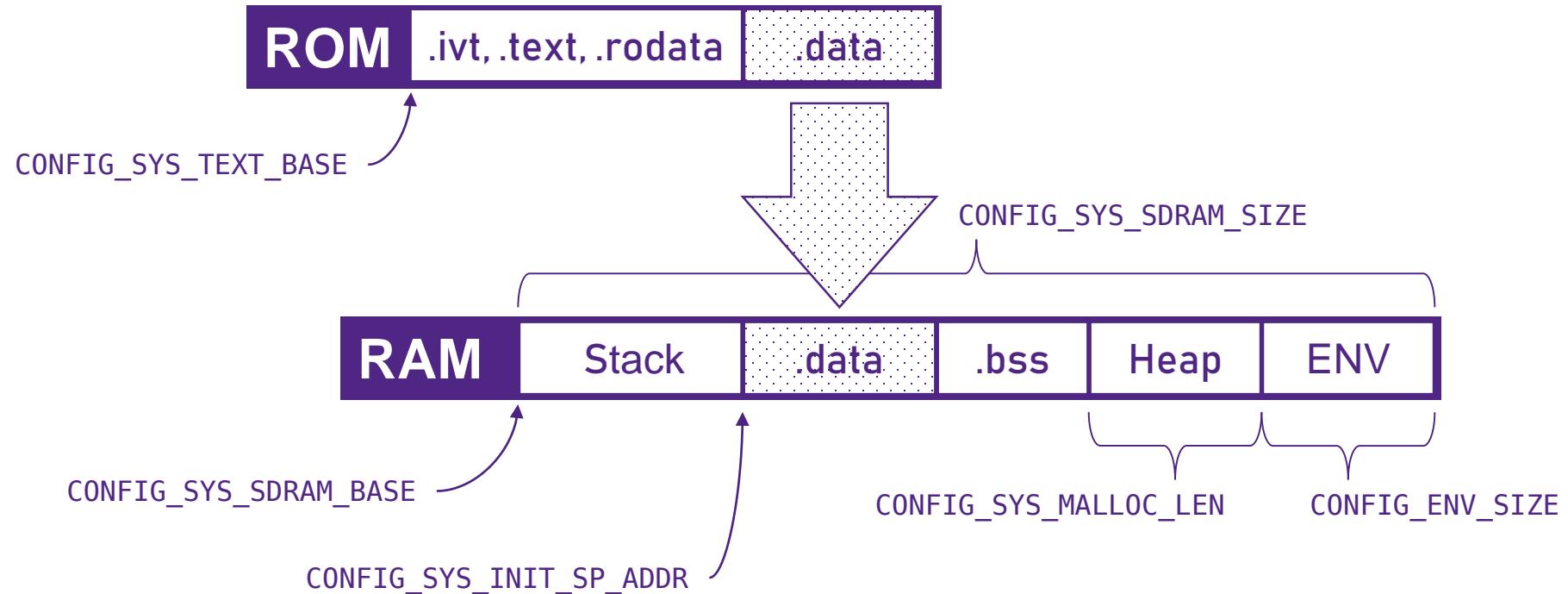
What if RAM size < ROM size = u-boot.bin

- Only supported for ARC as of today
- Add support for your architecture:
[264d298fda39 \("arc: Introduce a possibility to not relocate U-boot"\)](#)
- In platform/board code signal your intention
- Copy .data from ROM to RAM
- Keep executing code from ROM/flash
- Use RAM only for data
 - Heap
 - Stack
 - .data
 - Environment
 - Payload



Memory partitioning

Standard U-Boot *CONFIG_XXX* constants



Definition of derived constants

configs/iot_devkit.h

```
#define CONFIG_SYS_MONITOR_BASE           CONFIG_SYS_TEXT_BASE\n\n#define SRAM_BASE                         0x30000000\n#define SRAM_SIZE                          SZ_128K\n#define DCCM_BASE                         0x80000000\n#define DCCM_SIZE                          SZ_128K\n\n#define CONFIG_SYS_SDRAM_BASE             DCCM_BASE\n#define CONFIG_SYS_SDRAM_SIZE            DCCM_SIZE\n#define CONFIG_SYS_INIT_SP_ADDR          (CONFIG_SYS_SDRAM_BASE + SZ_32K)\n#define CONFIG_SYS_MALLOC_LEN            SZ_64K\n#define CONFIG_SYS_BOOTM_LEN             SZ_128K\n#define CONFIG_SYS_LOAD_ADDR             SRAM_BASE\n#define ROM_BASE                          CONFIG_SYS_MONITOR_BASE\n#define ROM_SIZE                           SZ_256K\n#define RAM_DATA_BASE                     CONFIG_SYS_INIT_SP_ADDR\n#define RAM_DATA_SIZE                    CONFIG_SYS_SDRAM_SIZE - \  
                                      (CONFIG_SYS_INIT_SP_ADDR - \  
                                       CONFIG_SYS_SDRAM_BASE) - \  
                                       CONFIG_SYS_MALLOC_LEN - \  
                                         CONFIG_ENV_SIZE
```

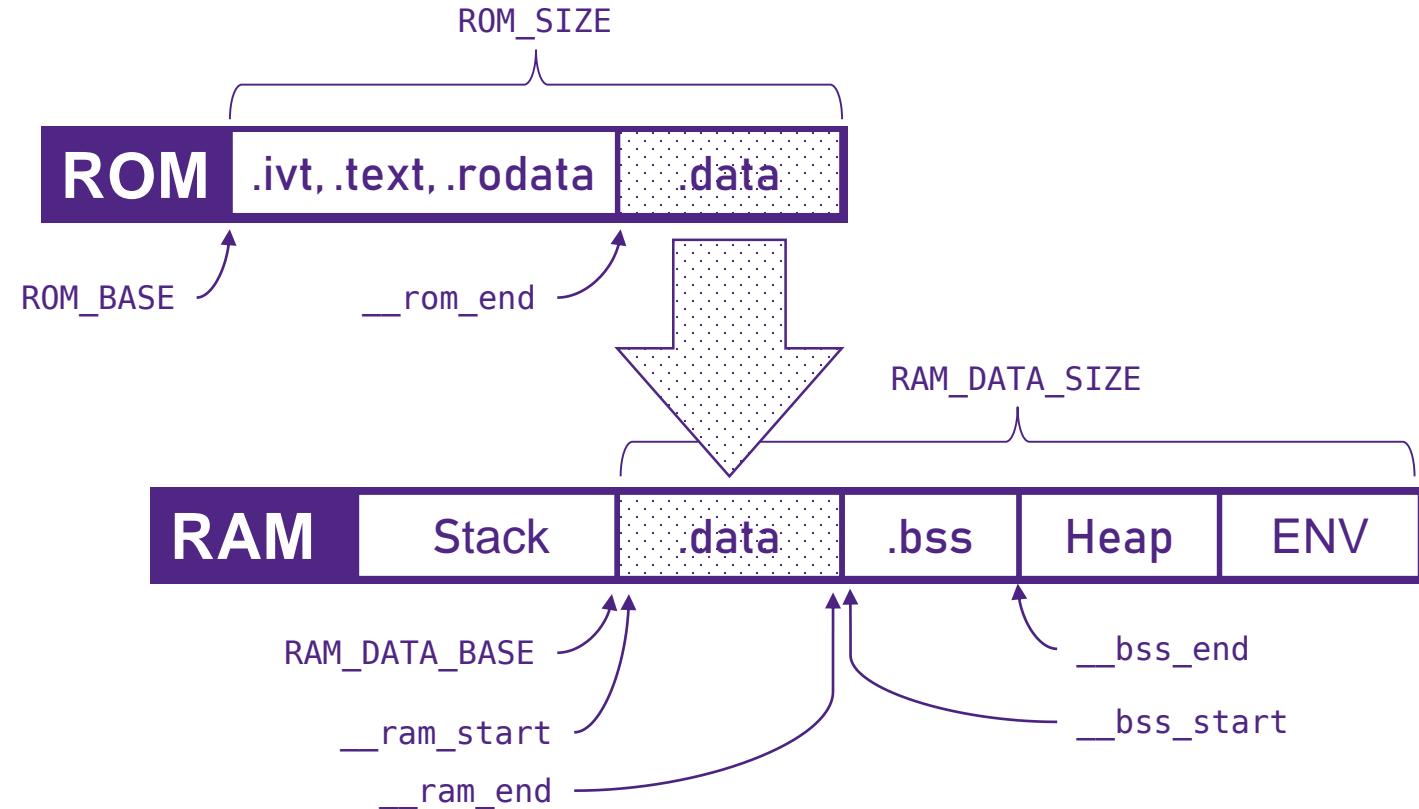
board/synopsys/iot_devkit/u-boot.lds

```
MEMORY {\n    ROM : ORIGIN = ROM_BASE, LENGTH = ROM_SIZE\n    RAM : ORIGIN = RAM_DATA_BASE, LENGTH = RAM_DATA_SIZE\n}\n\nSECTIONS {\n    . = CONFIG_SYS_MONITOR_BASE;\n    .ivt : { *(.ivt); } > ROM\n    .text : { *(.text*); } > ROM\n    .rodata : { *(.rodata*); } > ROM\n    __rom_end = .;\n    .data : {\n        __ram_start = .;\n        *(.data*)\n        __ram_end = .;\n    } > RAM AT > ROM\n    .bss : {\n        __bss_start = .;\n        *(.bss*)\n        __bss_end = .;\n    } > RAM
```

<https://sourceware.org/binutils/docs/ld/Output-Section-LMA.html#Output-Section-LMA>

Memory partitioning

Derived constants



Required quirks

They are not too many

- Signal intention to skip relocation
 - Set `GD_FLG_SKIP_REL0C` flag
- Copy `.data` section from ROM to RAM
- Zero `.bss` as usual in `clear_bss()`

```
/* 1. Don't relocate U-Boot */
gd->flags |= GD_FLG_SKIP_REL0C;

/* 2. Copy data from ROM to RAM */
u8 *src = __rom_end;
u8 *dst = __ram_start;
while (dst < __ram_end)
    *dst++ = *src++;

/* 3. Zero .bss as usual in clear_bss() */
size_t len = (size_t)&__bss_end - (size_t)&__bss_start;
memset((void *) &__bss_start , 0x00, len);
```

Run-time issues

-ENOMEM

Even though we boot to command prompt “usb start” fails

- Problem
 - Driver attempts to allocate 64 KiB buffer
- Fix
 - [42637fdae833 \(“usb: dwc2: Allow selection of data buffer size”\)](#)
 - Set CONFIG_USB_DWC2_BUFFER_SIZE = 16 (instead of default 64)
- Hint
 - Check malloc() return value early!

```
starting USB...
USB0: probe failed, error -12
USB error: all controllers failed lowlevel init
```

```
[14] malloc(bytes = 66328) = dlmalloc.c!1241
[13] memalign()+0x78 = dlmalloc.c!1922
[12] alloc_priv()+0x1a = device.c!269
[11] device_probe()+0x9c = device.c!325+0x6
[10] usb_init()+0xa2 = usb-uclass.c!276
[ 9] do_usb_start()+0xc = usb.c!586+0x4
[ 8] do_usb()+0x4e = usb.c!657
...
```

```
#define DWC2_DATA_BUF_SIZE          (64 * 1024)
struct dwc2_priv {
    uint8_t aligned_buffer[DWC2_DATA_BUF_SIZE];
    ...
}
```

Stack overflow

Compared to malloc we don't control stack size

- Problem
 - Instead of 78 bytes for `struct legacy_mbr` we allocate $78 * 512$ ("blksz") = 40KiB on stack
- Fix
 - [8639e34d2c5e \("part: Allocate only one legacy_mbr buffer"\)](#)
- Hints
 - `ALLOC_ALIGN_BUFFER()`,
`ALLOC_CACHE_ALIGN_BUFFER()` allocate buffers on stack
 - Use Memory Protection Unit (MPU) if possible
 - Locate stack right after non-existing memory or at least read-only region to get early exception

```
IoTDK# usb start
starting USB...
USB0: scanning bus 0 for devices...

[20] part_test_dos() = part_dos.c!90
[19] part_init()+0x30 = part.c!241
[18] usb_stor_probe_device()+0xfc = usb_storage.c!280
[17] device_probe()+0x84 = device.c!416

#define ALLOC_CACHE_ALIGN_BUFFER(type, name, size) \
    char name[size * sizeof(type)]

static int part_test_dos(struct blk_desc *dev_desc)
{
    ALLOC_CACHE_ALIGN_BUFFER(legacy_mbr, mbr,
                            dev_desc->blksz);
    ...
}
```

Conclusions

U-Boot could be ported on very memory-constrained system

- 200 KiB of ROM and 128 KiB of RAM is enough for full-scale U-Boot
 - USB and MMC drivers
 - FAT file-system with write support
- With tools, trials & errors it's possible to shrink memory footprint a lot
 - With vary basic tools it's possible to identify large statically-allocated objects
 - Allocations happen in run-time as well
 - Fixes and improvements to generic code might be required
- Special measures required to skip relocation
 - Requires changes in generic & architecture-specific code
- Run-time issues are mostly due to attempts to allocate more memory than available

Thank You

