Cambridge Consultants

EMBEDDED RUST ON THE BEAGLEBOARD X15

MEETING EMBEDDED

Jonathan Pallant

14 November 2018

OUR LOCATIONS

What is Rust?

- Statically compiled programming language
- Backed by Mozilla
- Being used to re-write Firefox
  - e.g. new multi-threaded CSS engine
- Uses LLVM as the backend
  - Supports x86, AMD64, PowerPC, MIPS, SPARC, Arm (Cortex-M, -R and -A).
- Strong on type safety and memory safety
- First class tooling:
  - build system, package manager, docs, code formatter, etc
- **Zero cost abstractions**
  - **fast, reliable, productive: pick three**

Cambridge Consultants

Shortest Rust intro ever..

- We have Generics and Traits (like interfaces)

```
– fn test<T>(thing: T) where T: Debug { … }
```

- We have heap allocation and type inference:

```
– let x = Box::new(thing);
– let r = Rc::new(thing);
```

- We have struct and enum and closures.

```
– struct Uart { … }
– enum Interupts { … }
– access(|r| { r.field() });
```

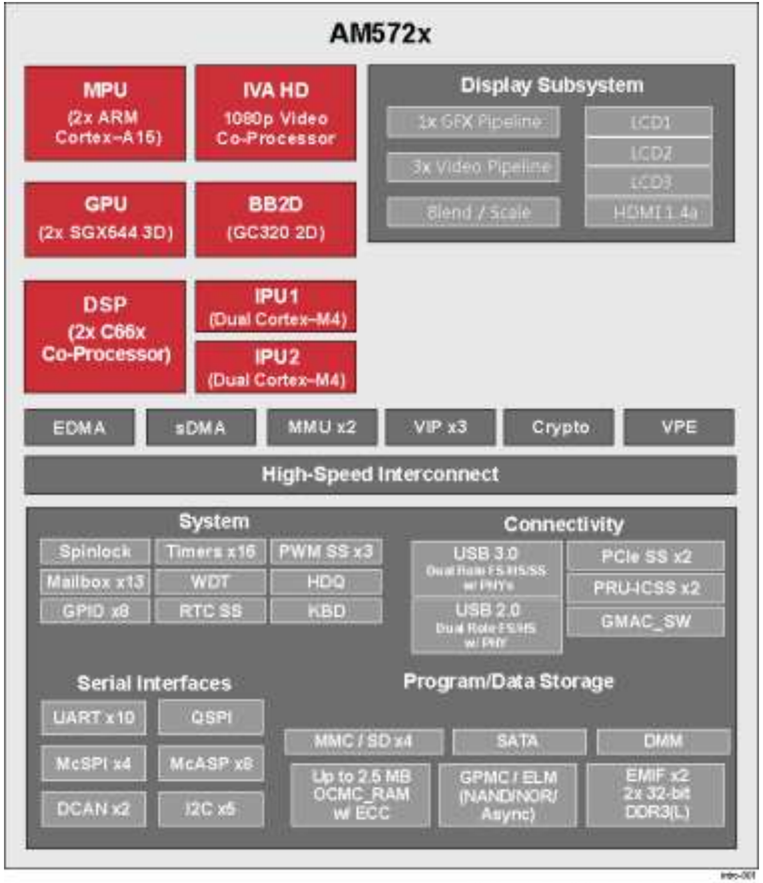
- We have collections:

```
let h: Hashmap<u32, Uart> = Hashmap::new();
```

- We have two libraries: *std* and *core*

The Beagleboard X15

- Biggest member of the Beagleboard family
- Not a Beaglebone…
- Texas Instruments AM5728 SoC
- 2 GiB DDR3 @ 533 MHz
- 4 GiB eMMC
- 2x Gigabit Ethernet
- 1x eSATA
- 1x microSD
- 1x HDMI (1080p)
- Line In/Out



AM5728

- Dual-core Cortex A15 MPU @ 1.5 GHz
- Dual-core SGX544 GPU @ 533 MHz
- 2x C66x DSPs @ 700 MHz
- 2x Dual-Core Cortex-M4 IPUs @ 213 MHz
- 2x Dual-Core 32-bit PRU-ICSS*
- Costs $50

** Programmable Real-Time Unit and Industrial Communication Sub-System*

RemoteProc

- Linux kernel feature
- Allows Linux to program, boot and control ‘remote processors’
- Controlled by the Device Tree
- RemoteProcs run ELF files:
  - live in /lib/firmware
  - must have a magic “.resource_table” section
  - Special linker script
  - Can run RTOS or bare-metal
- Memory is allocated from System RAM (Carveouts)
- Peripherals can be handed over (Device Memory)
- Shared text buffer for debug (Trace)
- Ring Buffers (VirtIO vrings)

Software Support

- The AM5728 is a “vayu” class SoC
  - In the Sitara family but from the OMAP5 lineage
  - Heavily related to (and often referred to as) the DRA7x automotive infotainment SoC family
- Kernel support (omap-remoteproc) in TI’s tree for loading IPU, PRU and DSP
- Example code in the TI SDK
  - IPC examples for Linux and QNX MPU talking to TI-RTOS on the IPU/DSP
  - Examples use TI’s Javascript based build system
  - Serious quantities of autogenerated code, magic numbers and deep macro indirection
  - Incredibly difficult to work out what’s going on:
    - How do these processors talk to each other?
    - How does the firmware get into RAM?
    - What’s a vring?
    - Who configures each of the three(?) MMUs?
    - How does the IPU even boot?

Cambridge Consultants

Booting the IPU

- Both cores boot from 0x0000_0000 at the same time.

- Which is which?
  - Magic register which returns 0 on Core 0 and 1 on Core 1
  - Not in the 8,500 page datasheet…

- Need to write ARM Assembler as we can't use the stack pointer
  - Both cores have the same stack pointer!

- Sleep the core we don't want with **`wfi`**

- Configure the L1 AMMU

Boot Code

```
vecbase: .long   0                 @ sp = not used
         .long   ti_sysbios_family_arm_ducati_Core_reset
core1sp: .long   0                 @ Core 1 sp
core1vec:.long   0                 @ Core 1 resetVec
ti_sysbios_family_arm_ducati_Core_reset:
        ldr     r0, coreid         @ point to coreid reg
        ldr     r0, [r0]           @ read coreid
        cmp     r0, #0
        bne     core1
core0:
        ...
core1:
        ...
coreid: .word   0xE00FFFE0
```

Cambridge Consultants

Boot Code

```
vecbase: .long   0                   @ sp = not used
         .long   ti_sysbios_family_arm_ducati_Core_reset
core1sp: .long   0                   @ Core 1 sp
core1vec:.long   0                   @ Core 1 resetVec
ti_sysbios_family_arm_ducati_Core_reset:
        ldr     r0, coreid          @ point to coreid reg
        ldr     r0, [r0]            @ read coreid
        cmp     r0, #0
        bne     core1
core0:
        ...
core1:
        ...
coreid: .word   0xE00FFFE0
```

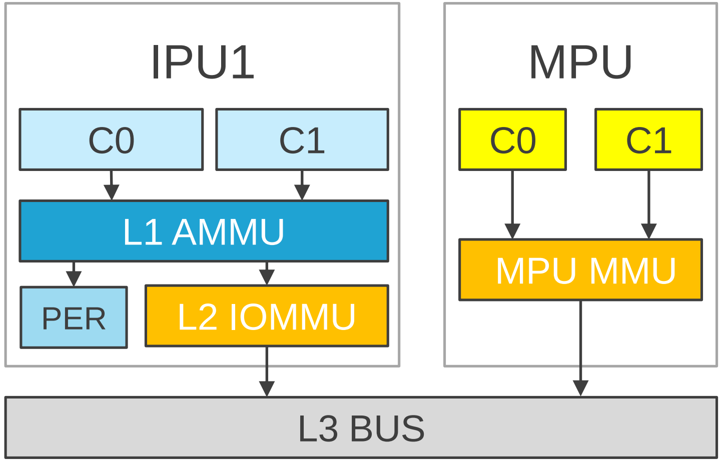
```
#[doc(hidden)]
#[link_section = ".vector_table.reset_vector"]
#[no_mangle]
pub static __RESET_VECTOR: unsafe extern "C" fn() -> ! = Reset;

#[no_mangle]
pub unsafe extern "C" fn Reset() -> ! {
    const AM5728_IPU_PERIPHERAL_ID0: *const u32 =
        0xE00FFFE0 as *const u32;
    if read_volatile(AM5728_IPU_PERIPHERAL_ID0) != 0 {
        loop {
            asm!("wfi");
        }
    }
    r0::zero_bss(&mut __sbss, &mut __ebss);
    main();
}
```

Cambridge Consultants

Managing Memory Management Units

- Linux controls the MPU MMU and the IPUx L2 IOMMU.
  - The IPU L2 IOMMU is configured using the resource table
- The IPU must configure its own L1 AMMU
  - Also called the “Unicache MMU”
  - Is also an L1 cache
  - Has default mappings to allow boot code to run
  - Mostly straight-through, but need to ensure addresses that come in/out of the top of the IPU L2 IOMMU are mapped to addresses the Cortex-M4 cores can access.
  - Cortex-M4s have ‘bit-banding’ functionality on certain address ranges, make use of them or avoid them.
- You can add 2x DSPs, IPU2, the 3D GPU, the 2D GPU, 2x PCI-Express subsystems and 2x EDMA controllers to this picture as they all have one or more MMUs…



Cambridge Consultants

Resource Tables

- A series of structures in memory
- Array of offsets to each structure
- Common header for each resource
- Describes:
  - Carve Outs
  - Device Memory
  - Trace Buffers
  - VirtIO devices

```
#[link_section = ".resource_table"]
#[no_mangle]
#[repr(C)]
pub static RESOURCE_TABLE: ResourceTable = ResourceTable {
    base: rt::Header { ver: 1, num: NUM_ENTRIES, reserved: [0, 0], },
    offsets: [...],
    rpmsg_vdev: rt::Vdev {
        rtype: rt::ResourceType::VDEV,
        id: vring::VIRTIO_ID_RPMSG,
        notifyid: 0,
        dfeatures: 1,
        gfeatures: 0,
        config_len: 0,
        status: 0,
        num_of_vrings: 2,
        reserved: [0, 0],
    },
    rpmsg_vring0: rt::VdevVring {
        da: 0x60000000,
        align: 4096,
        num: 256,
        notifyid: 1,
        reserved: 0,
    },
    ...
};
```

Cambridge Consultants

Writing to the Trace Buffer

- Address specified in resource table.
- Null-terminated text buffer – probably UTF-8.
- RemoteProc needs to append to this buffer
- If we run out of space … just erase everything and go back to the start
- Userland can obtain buffer with:

```
$ cat /sys/kernel/debug/remoteproc/remoteproc0/trace
```

- Generally just run:

```
$ watch tail -n 30 /sys/kernel/debug/remoteproc/remoteproc0/trace
```

Cambridge Consultants

Reading/Writing VirtIO vrings

- Transliterating kernel structures into Rust
- Couldn't find much documentation
  - First used by hypervisors for paravirtualised device drivers
- https://www.ibm.com/developerworks/library/l-virtio/index.html

```
#[repr(C)]
#[derive(Debug, Clone, Copy)]
pub struct DescriptorEntry {
    addr: u64,
    len: u32,
    pub flags: DescriptorFlags,
    pub next: u16,
}

#[repr(C)]
pub struct UsedRing {
    pub flags: UsedFlags,
    pub idx: u16,
    pub ring: UsedEntry,
}

#[repr(C)]
pub struct AvailableRing {
    pub flags: AvailableFlags
    pub idx: u16,
    pub ring: AvailableEntry,
}
```

```
pub struct GuestVring {
    descriptors: &'static mut DescriptorRing,
    available: &'static mut AvailableRing,
    used: &'static mut UsedRing,
    entries: usize,
    last_seen_available: u16,
    addr_map: &'static Fn(u64) -> u64
}
```

Cambridge Consultants

Using the Mailbox

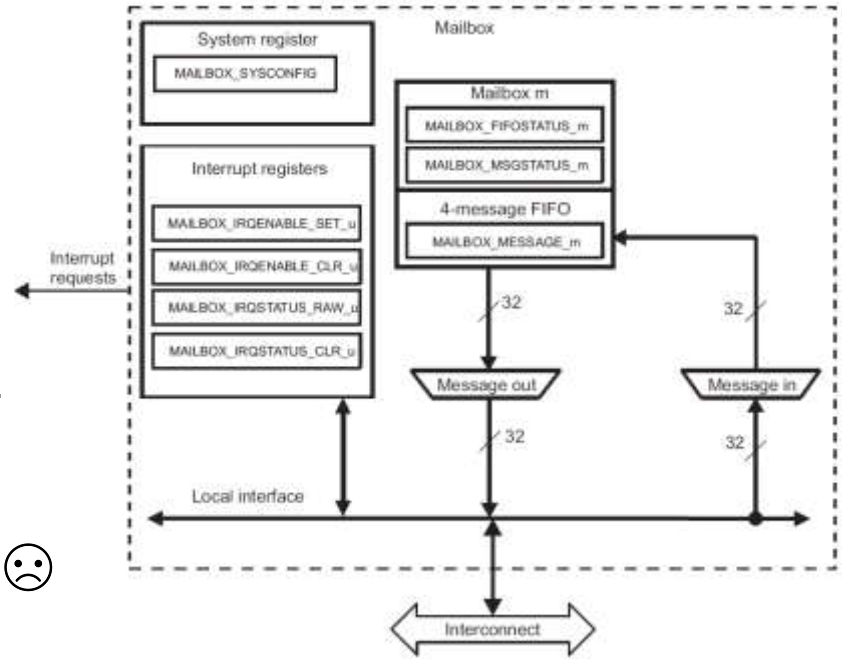
- The Am5728 has 13 System Mailbox peripherals

- Each mailbox has:
  - 3 or 4 ‘users’
  - 8 or 12 individual FIFOs
  - Up to 4 messages (each 32-bits) per FIFO

- Each FIFO should have one writing user and one reading user.
- Each user gets their own interrupts.
- Routing the interrupts is done through the Interrupt Crossbar.
- The documentation unhelpfully refers to a FIFO as a ‘mailbox’ ☹

- Allocation of Mailboxes to processor cores is deep magic.

Figure 19-8. Mailbox Block Diagram



Cambridge Consultants

Making the Firmware

- Need to ensure the resource table goes into ".resource_table"
  - This configures the IPU's L2 IOMMU (but not the L1 AMMU)
- Vector table must be at 0x0000_0000, followed by code.
- Data lives at 0x8000_0000.
- Unclear if DDR or internal SRAM mapped to 0x0000_0000.
  - Probably first 64 KiB is SRAM and rest is DDR?
- For compatibility with the toolchain we call the 0x0000_0000 segment "Flash" even though it's just RAM.
- Don't need to copy .data from Flash to RAM, do need to zero .bss
- Special section for IPC data – address also specified in resource table and in MMU config as un-cachable

- `$ cargo build --release` *`[--target=thumbv7em-none-eabi ]`*
-
```
$ scp ./target/thumbv7em-none-eabi/release/ipu-demo \
            root@beagleboard:/lib/firmware/dra7-ipu1-fw.xem4
```

Cambridge Consultants

Access from Linux user-space

- RemoteProc is a protocol that uses VirtIO Vrings as a transport, and Mailboxes as a notification mechanism.
- In user-space, RemoteProc is access using a socket of type AF_RPMSG.
  - RemoteProc messages have source and destination addresses.
  - In your application you *bind* a socket to receive from the IPU, and *connect* a second to send to the IPU.
- The IPU informs Linux of its address on start-up using a well-known Name Server address (53).
- Each TX socket write becomes goes on the VirtIO Queue A available ring
- Each packet placed in the VirtIO Queue B used ring appears when RX socket when read.

Cambridge Consultants

Use the source, Luke!

https://github.com/cambridgeconsultants/rust-beagleboardx15-demo (coming soon...)

linux: user-space AF_RPMSG socket using code

bare-metal: Rust code for IPU1_Core0 *(contains a fork of cortex-m-rt)*

Get in touch:

- https://keybase.io/thejpster
- https://github.com/rust-embedded and @rustembedded on Twitter.
- @thejpster in #rust-embedded on Mozilla IRC
- **We have jobs! See cambridgeconsultants.com/careers**

Cambridge Consultants

UK ▪ USA ▪ SINGAPORE ▪ JAPAN

www.CambridgeConsultants.com

Registered no. 01036298 England and Wales