AMP
or
Linux and Zephyr meet STM32MP1

ECC 3. September 2019, © Markus Kappeler

Agenda

- Applikationen mit AMP
- Was ist embedded Linux – Yocto Project
- Was ist Zephyr
- Was ist OpenAMP
  - Remoteproc
  - RPMsg
- Demo
- Zusammenfassung

Applikationen mit AMP

- AMP = asymetric multi processor
- Im gleichen SoC kann es verschiedene CPU Architekturen und Kombinationen geben:
  - Application Core: z.B. ARM Cortex A7
  - Digital Signal Processing DSP
  - Computing Power z.B: FPGA
  - Realtime Systeme: PRU
  - Low Power und Realtime performance: ARM Cortex M4

Applikationen mit AMP

- Real-time
- Performance Optimierungen
- Reduktion Energieverbrauch (Vorschriften Standby-Mode)
- System Security
- Zertifizierte SW (z.B. Misra)
- Wiederverwendung von Legacy Software

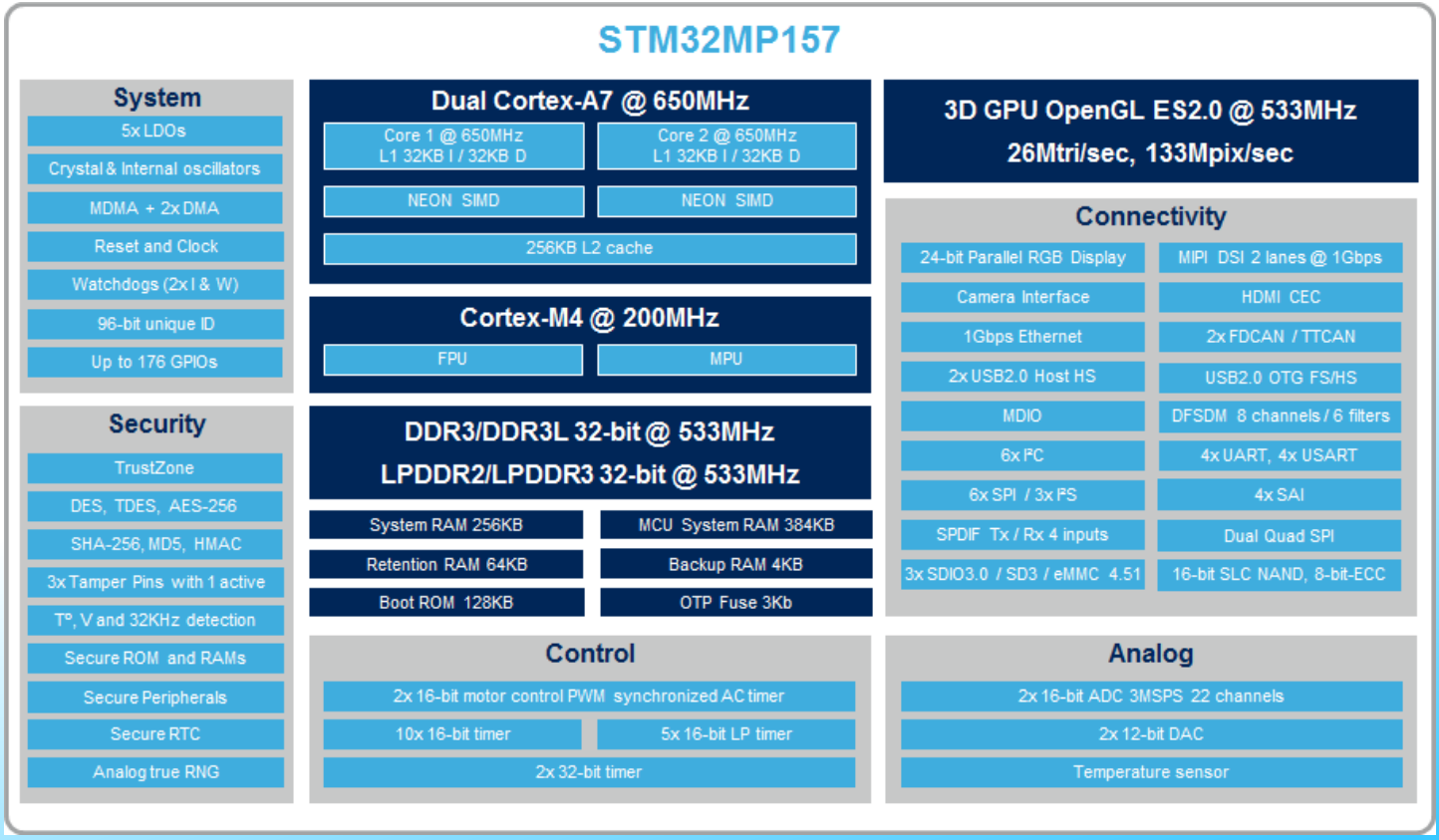
Realtime Application

- Linux Kernel (mit PREEMPT_RT) erfüllt (teilweise) die Anforderungen. Aber Tuning, Customising, Debugging, Maintaining und Life-Cycle darf man nicht unterschätzen in Bezug auf Know-How, Zeit und Geld.

- AMP ist ein möglicher Lösungsansatz
  - Trennung der Software-Domains
  - BOM Reduktion

- AMP hat aber auch Herausforderungen:
  - Interprozessor Synchronization und Kommunikation
  - Power Management
  - Shared Ressources

AMP mit STM32MP1



bytesatwork

Yocto project

- Das Yocto project ist keine embedded Linux Distribution –
  Es kreiert eine für Dich!

- Das yocto Projekt ist nicht ein einzelnes open source Projekt – es ist ein Ecosystem.

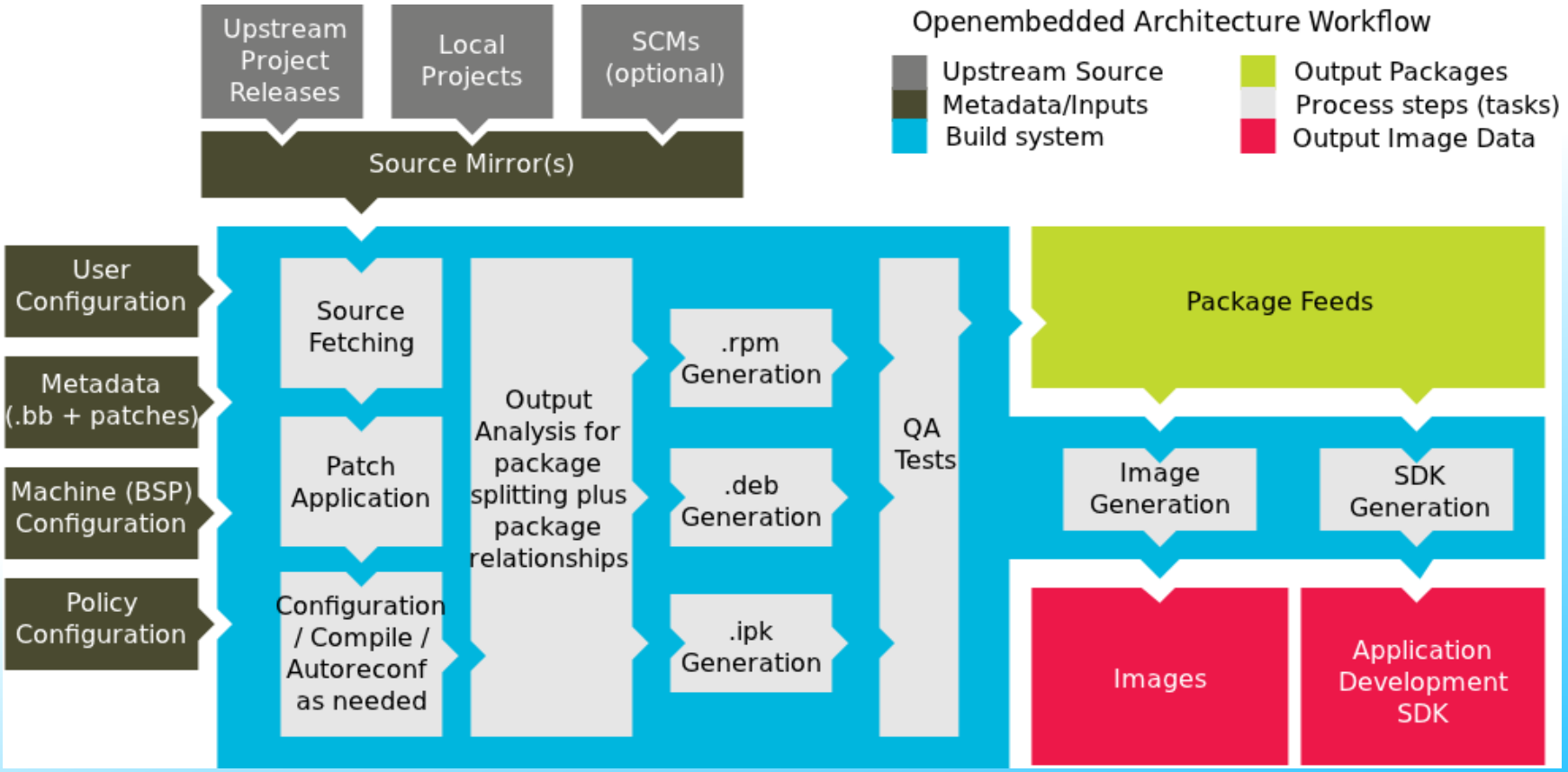
- yocto ist eine Kombination von ready to run Linux-Distro mit der Flexibilität eines custom Linux-OS.

Was ist das Yocto Project

- Opensource Projekt mit aktiver Community
- Ein Sammlung von embedded Projekten und Tools:
  - Board Support Packages
  - Application Development Tools (Eclipse Plugins)
- Referenz Distro Poky
  - Kompletes Build-System basierend auf bitbake
  - Release alle 6 Monate (April und Oktober) mit aktuellem Kernel LTS, Toolchain und Package Versionen.

It's not an embedded Linux distribution,
It creates a custom one for you.

yocto build process



source yocto project

yocto project auf den Punkt gebracht

1) yocto von yoctoproject.org herunterladen

   git clone http://git.yoctoproject.org/git/poky

2) source oe-init-build-env
3) Herunterladen weiterer Layer (meta-bytesatwork)

   layers.openembedded.org

4) Layer in conf/bblayers.conf eintragen
5) Anpassen von conf/local.conf (MACHINE=”bytedevkit”)
6) Starte build Prozess

   bitbake dev-image

7) Kaffee trinken oder besser Mittag essen, ev. Schlafen gehen….

Auswahl von „freien“ RTOS für IoT

Azure IoT

arm MBED

Zephyr Ecosystem

Zephyr OS

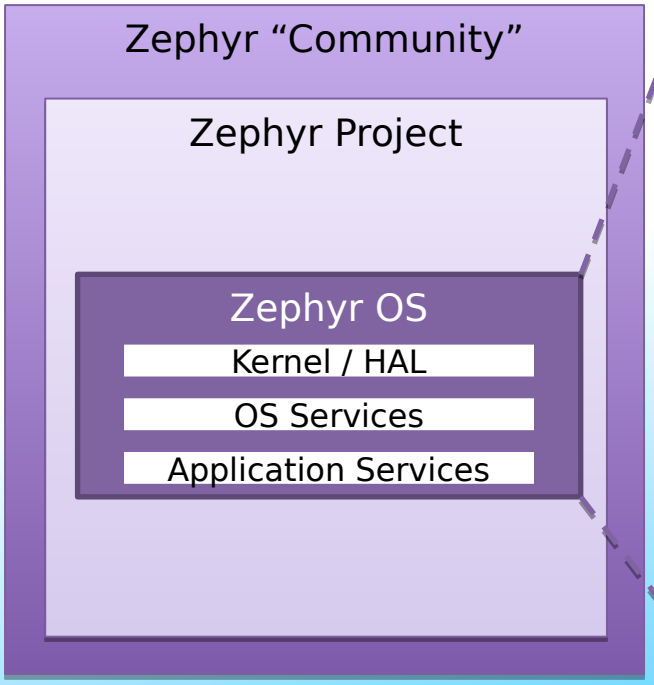
- Kernel und HAL
- OS Services wie **IPC**, Logging, file systems, crypto

Zephyr Project

- SDK, Tools und Entwicklungsumgebung
- Zusätzliche Middleware
- Device Management und Bootloader

Zephyr Community

- 3rd Party Module und Bibliotheken
- Support für Zephyr in 3rd Party Projekte: Jerryscript, Micropython, Iotivity



Kernel / HAL

- Scheduler
- Kernel objects und services
- low-level Architektur und BSP
- Power Management und low level Interfaces zur Hardware

OS Services and Low level APIs

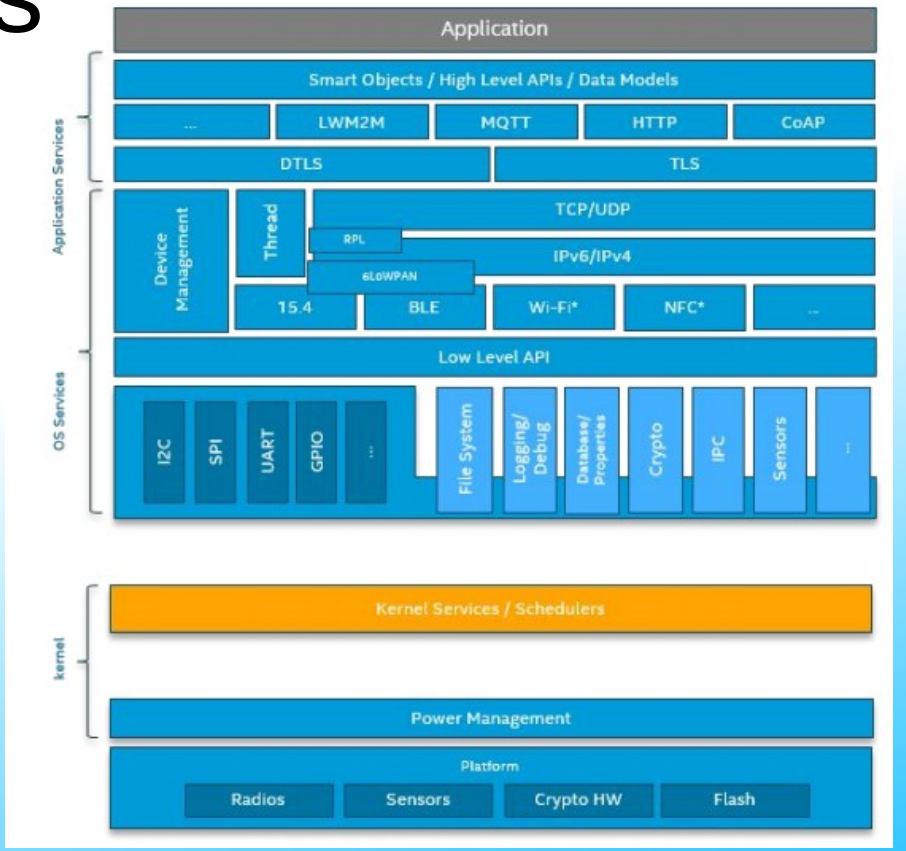
- Plattform spezifische Treiber
- Generisches I/O API
- File systems, Logging, Debugging und IPC
- Cryptography Services
- Networking und Connectivity

Application Services

- High Level APIs
- Zugang zu standardisierten Daten Modellen
- High Level Netzwerk Protokoll

Keyfeatures

- Cooperative und Pre-emptive Threading
- Speicher wird typischerweise statisch alloziert, auf Heap auch möglich
- Memory Protection: stack overflow Protection, Kernel object und Driver tracking, Thread isolation
- Integrierte Device Driver Schnittstelle
- Native und integrierter Netzwerk Stack
- Mehrere Architekturen: ARC, ARM, RISC-V, Tensilica, x86
- Tolerante Lizenz



Zephyr ist unter Apache 2.0 Lizenz

- Man darf Software unter dieser Lizenz frei in jedem Umfeld verwenden, modifizieren und verteilen.
- Eine Kopie der Lizenz muss dem Paket beiliegen.
- Änderungen am Quellcode der unter der Apache-Lizenz stehenden Software müssen nicht zum Lizenzgeber zurückgeschickt werden.
- Eigene Software, die unter Apache-Lizenz stehende Software verwendet, muss nicht unter der Apache-Lizenz stehen.

Zephyr Projekt Mitglieder

Februar 2016

2019

Platinum Members

Silver Members

and others….

Momentum	2016	2019
Entwickler	80	453
Commits	2806	29568
Repositories	5	22

bytesatwork

Zephyr- erster Eindruck

- cmake basiert
  - Build unter Windows möglich
- Gute HW Abstraktion
  - Applikation einfach portierbar
- Mehr als nur OS
  - Services wie HTTP-Server, COAP, MQTT
- Gute Dokumentation
  - Einfacher Einstieg
  - Viele Samples
- Gute Integration von BLE5
  - Mesh Networking
- Ähnlich wie Linux und doch anders
  - Coding style, Kbuild, Kconfig
  - west und ninja
- Vertraute Directory Struktur
- Sensor Abstraktion
- Junges Projekt
  - Schnelle Entwicklung
  - Häufige API Änderungen
  - Neue Tools

OpenAMP

- **Standard** von MCA (The Multicore Association)
  - https://www.multicore-association.org
  - eSOL, TI, Xlinx, Renesas, Poly-Core Software

- Implementiert im **Linux** Kernel und **Zephyr**
  - auch in Xlinx FPGA, TI PRUSS,

- Lifecycle Operations via **Remoteproc** (Remote Processor):
  - Framework um Remote Prozessor zu kontrollieren (power on/off, reset, load firmware, start/stop)

- Messaging via **RPMsg** (Remote Processor Messaging):
  - Framework für Inter-Prozessor Kommunikation (IPC) basierend auf **VirtIO** (Standard Linux Virtualization-Component) für Shared-Memory Management (sending/receiving data)

- Proxy channels:
  - rpmsg-chrdev: Remote Zugriff zu System-Services wie Filesystem
  - tty-channel: ttyRPMSG0 wie eine UART

remoteproc – Lifecycle Management

- Muss in Kernel-Config und devicetree aktiviert werden
- Sysfs Interface für Kommandos
  - echo zephyr.elf > /sys/class/remoteproc/remoteproc0/firmware
  - echo start > /sys/class/remoteproc/remoteproc0/state
  - echo stop > /sys/class/remoteproc/remoteproc0/state
  - cat /sys/class/remoteproc/remoteproc0/state

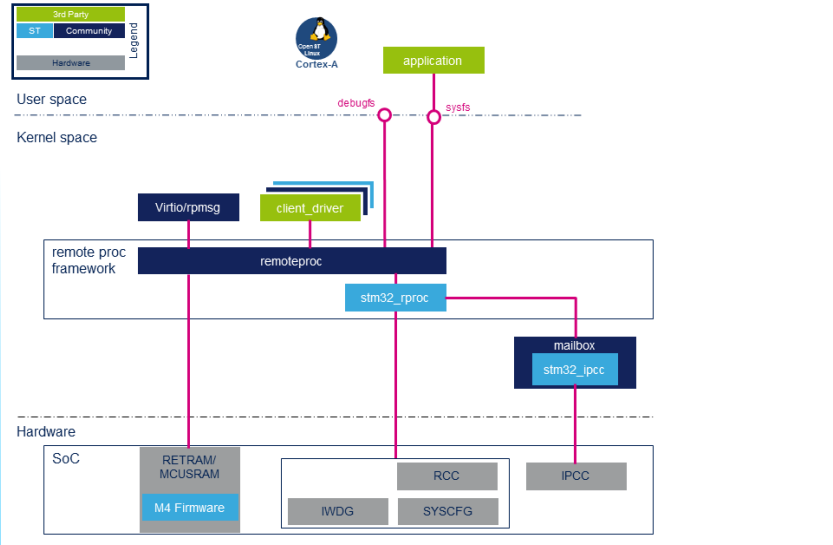
```
are @stm32mp1-disco:~# echo zephyr.elf > /sys/class/remoteproc/remoteproc0/firmwa
root@stm32mp1-disco:~#
root@stm32mp1-disco:~#
root@stm32mp1-disco:~# cat /sys/class/remoteproc/remoteproc0/state
offline
root@stm32mp1-disco:~# echo start > /sys/class/remoteproc/remoteproc0/state
[27430.385313] remoteproc remoteproc0: powering up m4
[27430.419888] remoteproc remoteproc0: Booting fw image zephyr.elf, size 578516
[27430.425835] remoteproc remoteproc0: remote processor m4 is now up
root@stm32mp1-disco:~# cat /sys/class/remoteproc/remoteproc0/state
running
root@stm32mp1-disco:~#
```

remoteproc

- Parsed Resource-Table der Firmware
- Stellt benötigte Ressourcen (IPC, memory, etc.) bereit
- Stellt Services für Debug und Überwachung der Firmware zur Verfügung.

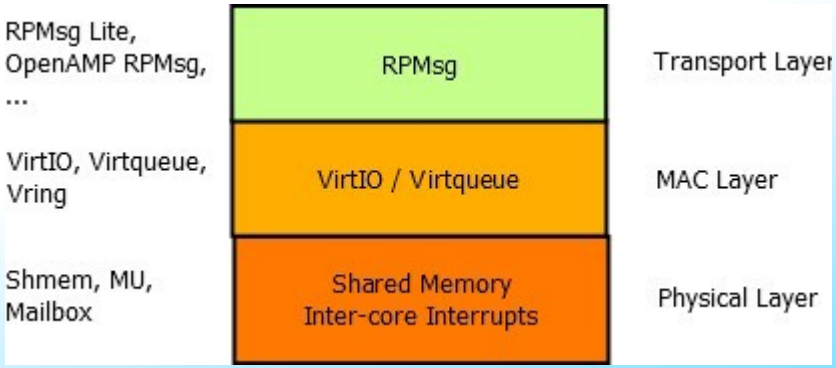
```
root@stm32mp1-disco:/sys/kernel/debug/remoteproc/remoteproc0# ls
carveout_memories  name  recovery  resource_table  trace0
root@stm32mp1-disco:/sys/kernel/debug/remoteproc/remoteproc0# cat trace0
***** Booting Zephyr OS build zephyr-v1.14.0-1669-g9f857c3ba804 *****
Hello World! stm32mp157c_dk2
root@stm32mp1-disco:/sys/kernel/debug/remoteproc/remoteproc0#
```

Demo remoteproc on STM32MP1

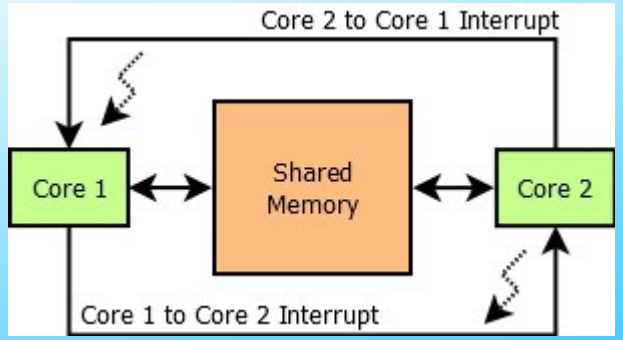


RPMsg

- Standardisierte Kommunikation zwischen Prozessoren
- Kommunikation kann in 3 OSI Layer abgebildet werden:

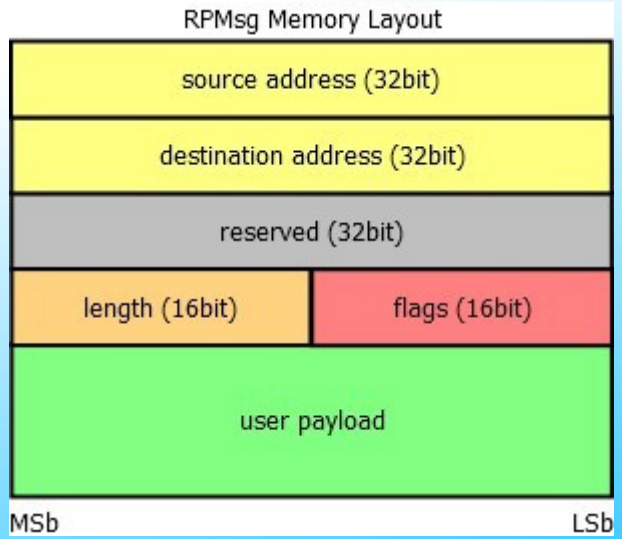


- Konzept basiert auf Shared-Memory zwischen den Prozessoren und Inter-Prozessor Interrupts



RPMsg

- Überträgt Nutzdaten im Shared-Memory mit Ring-Buffer
  - 2 Ring Buffers für jede Richtung (tx,rx)
  - Ring Buffers enthalten die Adressen vom Shared-Memory mit den RPMsg Daten
- RPMsg Message selber ist ein Buffer-Eintrag im Shared-Memory



RPMsg in Linux

- Enable rpmsg im Kernel
  - Device Driver → Rpmsg drivers →Virtio
- Define Memory regions im devicetree

```
/* Memory region declaration, containing vring and rpmsg buffers */
 reserved-memory {
        #address-cells = <1>;
        #size-cells = <1>;
        ranges;
        retram: retram@0x38000000 {
                compatible = "shared-dma-pool";
                reg = <0x38000000 0x10000>;
                no-map;
        };
        mcusram: mcusram@0x10000000 {
                compatible = "shared-dma-pool";
                reg = <0x10000000 0x40000>;
                no-map;
        };
        mcusram2: mcusram2@0x30000000 {
                compatible = "shared-dma-pool";
                reg = <0x30000000 0x40000>;
                no-map;
        };
        /* memory region reserved for virtio ring buffer descriptor 0*/
        vdev0vring0: vdev0vring0@10040000 {
                compatible = "shared-dma-pool";
                reg = <0x10040000 0x2000>;
                no-map;
        };
        /* memory region reserved for virtio ring buffer descriptor 1*/
        vdev0vring1: vdev0vring1@10042000 {
                compatible = "shared-dma-pool";
                reg = <0x10042000 0x2000>;
                no-map;
        };
        /* memory region reserved for virtio buffers associated to the vrings*/
        vdev0buffer: vdev0buffer@10044000 {
                compatible = "shared-dma-pool";
                reg = <0x10044000 0x4000>;
                no-map;
        };
};
```

```
        [ ] Platform support for Goldfish virtual devices  ----
        -*- Platform support for Chrome hardware  --->
        [ ] Platform support for Mellanox hardware  ----
            Common Clock Framework  --->
        [*] Hardware Spinlock drivers  --->
            Clock Source drivers  --->
        -*- Mailbox Hardware Support  --->
        [*] IOMMU Hardware Support  --->
            Remoteproc drivers  --->
        [ ] Rpmsg drivers  --->
        [ ] SoundWire support  ----
            SOC (System On Chip) specific Drivers  --->
        [ ] Generic Dynamic Voltage and Frequency Scaling (DVFS) support  ----
        -*- External Connector Class (extcon) support  --->
        [ ] Memory Controller drivers  ----
```

```
    < > RPMSG device interface
    < > Qualcomm RPM Glink driver
    <*> Virtio RPMSG bus driver
    <*> RPMSG tty driver
```

```
/* stm32 M4 remoteproc device */
  m4_rproc: m4@0 {
        compatible = "st,stm32mp1-rproc";
        ranges = <0x00000000 0x38000000 0x10000>,
                 <0x30000000 0x30000000 0x60000>,
                 <0x10000000 0x10000000 0x60000>;
        memory-region = <&retram>, <&mcusram>, <&mcusram2>, <&vdev0vring0>,
                        <&vdev0vring1>, <&vdev0buffer>;
        mboxes = <&ipcc 0>, <&ipcc 1>, <&ipcc 2>;
        resets = <&rcc_rst MCU_R>;
        reset-names = "mcu_rst";
        st,hold_boot = <&rcc 0x10C 0x1>;
        st,smc_boot;
        mboxes = <&ipcc 0>, <&ipcc 1>, <&ipcc 2>;
        status = "okay";
 };
```

bytesatwork

RPMsg in Linux

- #include <linux/rpmsg.h>
- struct rpmsg_driver rpmsg_sample_client = {

```
        .drv.name   = KBUILD_MODNAME,
        .id_table   = rpmsg_driver_sample_id_table,
        .probe      = rpmsg_sample_probe,
        .callback   = rpmsg_sample_cb,
        .remove     = rpmsg_sample_remove,
};
module_rpmsg_driver(rpmsg_sample_client);
```

- Send messages

```
rpmsg_send(rpdev->ept, MSG, strlen(MSG));
```

RPMsg in Linux

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/rpmsg.h>

static void rpmsg_sample_cb(struct rpmsg_channel *rpdev, void *data, int len, void *priv, u32 src){
      print_hex_dump(KERN_INFO, "incoming message:", DUMP_PREFIX_NONE, 16, 1, data, len, true);
}

static int rpmsg_sample_probe(struct rpmsg_channel *rpdev){
      int err;

      dev_info(&rpdev->dev, "chnl: 0x%x -> 0x%x\n", rpdev->src, rpdev->dst);
      /* send a message on our channel */
      err = rpmsg_send(rpdev, "hello!", 6);
      if (err) {
              pr_err("rpmsg_send failed: %d\n", err);
              return err;
      }
      return 0;
}

static void rpmsg_sample_remove(struct rpmsg_channel *rpdev){
      dev_info(&rpdev->dev, "rpmsg sample client driver is removed\n");
}

static struct rpmsg_device_id rpmsg_driver_sample_id_table[] = {
      { .name = "rpmsg-client-sample" },
      { },
};
MODULE_DEVICE_TABLE(rpmsg, rpmsg_driver_sample_id_table);

static struct rpmsg_driver rpmsg_sample_client = {
      .drv.name       = KBUILD_MODNAME,
      .id_table       = rpmsg_driver_sample_id_table,
      .probe          = rpmsg_sample_probe,
      .callback       = rpmsg_sample_cb,
      .remove         = rpmsg_sample_remove,
};
```

RPMsg in Zephyr

- Enable Module open-amp

- Abhängig von libmetal
  - Glue zwischen OS und IPC/OpenAMP

- Shared-Memory device definieren

- VRING erzeugen und registrieren

- Endpoint erzeugen

```
int rpmsg_create_ept(struct rpmsg_endpoint *ept, struct rpmsg_device *rdev,
                     const char *name, uint32_t src, uint32_t dest,
                     rpmsg_ept_cb cb, rpmsg_ns_unbind_cb ns_unbind_cb);
```

- Messages empfangen via Callback
  - int endpoint_cb(struct rpmsg_endpoint *ept, void *data, size_t len, u32_t src, void *priv)

- Messages versenden

```
rpmsg_sendto(struct rpmsg_endpoint *ept, const void *data, int len, uint32_t dst)
```

RPMsg als virtuelle UART

- Generischer Ansatz, kein eigener Treiber notwendig
- Enable RPMsg tty driver im Kernel (CONFIG_RPMSG_TTY)
- Erzeugt ein virtuelle UART in /dev/ttyRPMSG0
- Kann genutzt werden wie jede andere UART

RPMsg als char-device

- Generischer Ansatz, kein eigener Treiber notwendig
- Enable RPMsg Device Interface im Kernel (CONFIG_RPMSG_CHAR)
- Erzeugt ein Control device /dev/rpmsg_ctrl0
- Via ioctrl eigentliches character device /dev/rpmsg0 erzeugen
- Open, read, write wie bei jedem anderen File

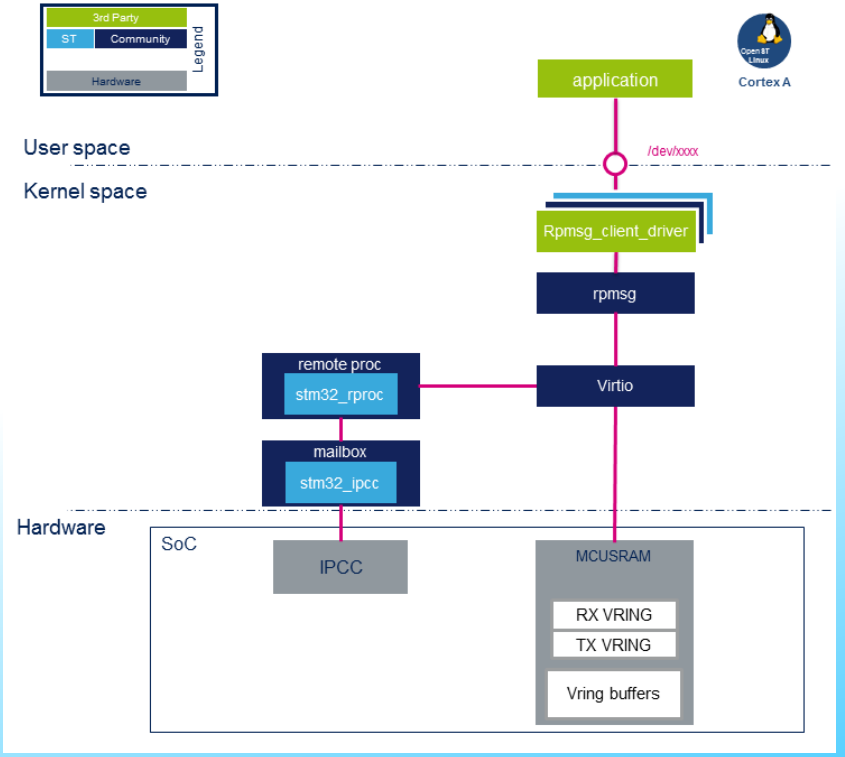
```
 int main(int argc, char *argv[])
{

        struct rpmsg_endpoint_info ept_info = {"rpmsg-demo-channel", 0x2, 0x1};
        int fd = open("/dev/rpmsg_ctrl0", O_RDWR);
        if (fd < 0) {
                printf("Error opening rpmsg crtl0: %s.\n", strerror(errno));
                return -1;
        }

        /* create endpoint interface */
        ioctl(fd, RPMSG_CREATE_EPT_IOCTL, &ept_info);  // /dev/rpmsg0 is created

        /* create endpoint */
        int fd_ept = open("/dev/rpmsg0", O_RDWR);
```

Demo RPMsg on STM32MP1



Zusammenfassung

- AMP = Asymetric Multi Processor

- Yocto baut custom-Linux Distribution
  - Ecosystem – nicht einzelnes Projekt

- Zephyr RTOS von Linux Foundation
  - Security, Safety, IoT

- Open AMP ein Standard von MCA
  - Gibt es für verschiedene Systeme (FPGA, PRUSS, Cortex-M4)
  - Linux und Zephyr haben Implementationen

- remoteproc
  - Lifecycle Management: load, start und stop

- RPMsg
  - Daten-Austausch via virtio (Shared-Memory) je in eine Richtung (rx/tx)
  - RPMsg tty: virtuelle UART
  - RPMsg chrdev: character device /dev/rpmsg0

- STM32MP1 hat AMP: Dual Cortex-A7 und Cortex-M4
  - Gute Beispiele und Doku von ST

markus.kappeler@bytesatwork.ch
http://www.bytesatwork.io