Software Defined Hardware on Heterogenous SoCs Using OpenAMP

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

Who am I?

Peter Fecher

- Finished bachelors degree few months ago
- Working at PHYTEC Messtechnik GmbH
- On IoT and low power projects

PHYTEC

Timeline

1. Introduction
2. What is SDH?
3. Live Demo
4. Conclusion

Introduction

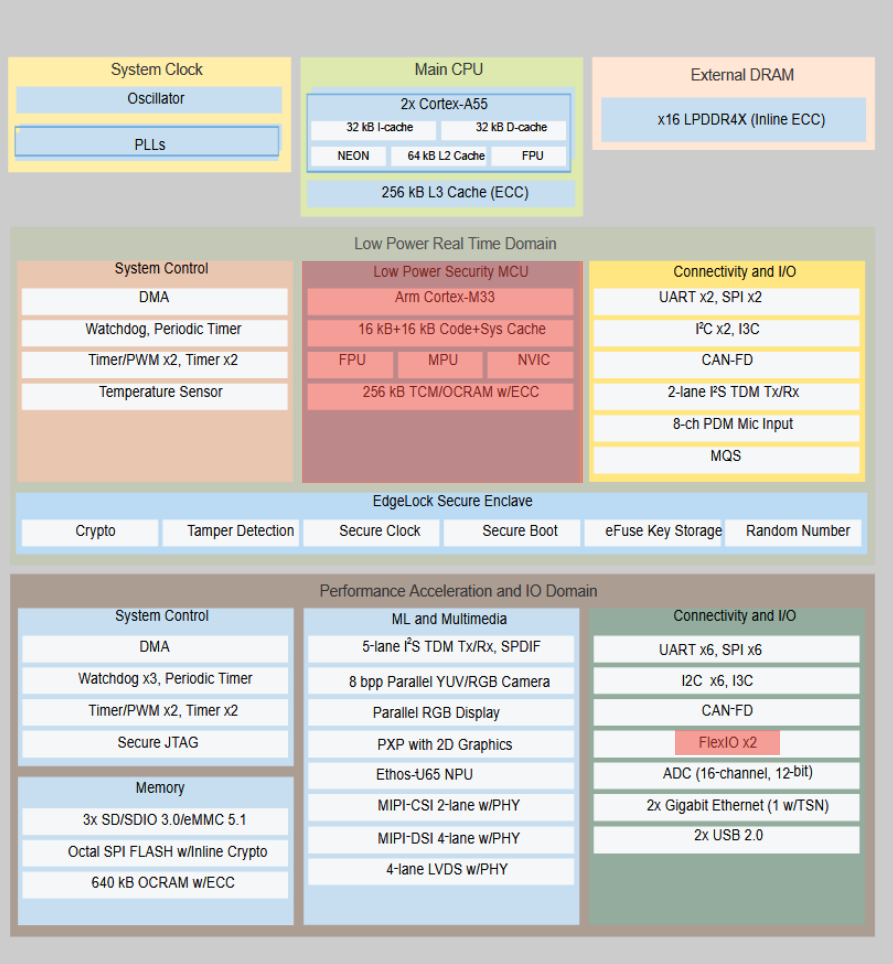
Most modern SoCs include flexible interfaces

e.g. NXPs **FlexIO**, RPIs **PIO**, TIs **PRU**s

Can be used to **emulate** almost **any interface** without using main CPU resources

For example I2C, SPI, CAN, Modbus
68K Bus

About which SoCs are we talking?



i.MX93 RM

Introduction

Why do we need that?

Agile Projects

„Interface Updates“

More flexible products

Introduction

Some interfaces need real-time capabilities

→ e.g. NXPs FlexIO

→ One word buffer

→ On use with fast UART: Linux is too slow to copy the word before it gets lost

Vendor-specific API

→ Likely differs between platforms

→ Maybe not well maintained

How can we overcome these Problems?

→ Buffer incoming data

→ Abstraction

→ Simple configuration

What is SDH?

→ **Software Defined Hardware**

→ Describing the type and configuration
of hardware interfaces of a system

Differentiation from configuring hardware

→ Normally configuring just operating parameters

→ SDH is thought so that the **interface type itself can be configured**

What is SDH?

How to achieve the goals?

→ Most of the infrastructure exists already

The main ingredient: **co-processors**

→ **And most modern SoCs feature some**

What is SDH?

How to make use of the co-processors?

→ Buffering incoming data (e.g. Ringbuffer)

→ Communicating to Linux via IPC

→ Some sort of RTOS running on coprocessor

What is SDH?

Which RTOS on co-processor?

We could do it **Bare Metal**

→ That decreases modularity

We could use a **vendor SDK + FreeRTOS**

→ Leads to reimplementation for each platform

We could use **Zephyr RTOS**

→ **High portability**, but maybe some hardware features missing

Benefits of Zephyr

→ Advanced and lightweight RTOS

→ Great portability (ARM, x86, RISC-V)

→ Devicetree & Kconfig

→ OpenAMP support

Zephyr® Project

What is SDH?

OpenAMP:

→ Framework for managing (heterogeneous) multi processor systems

Remoteproc:

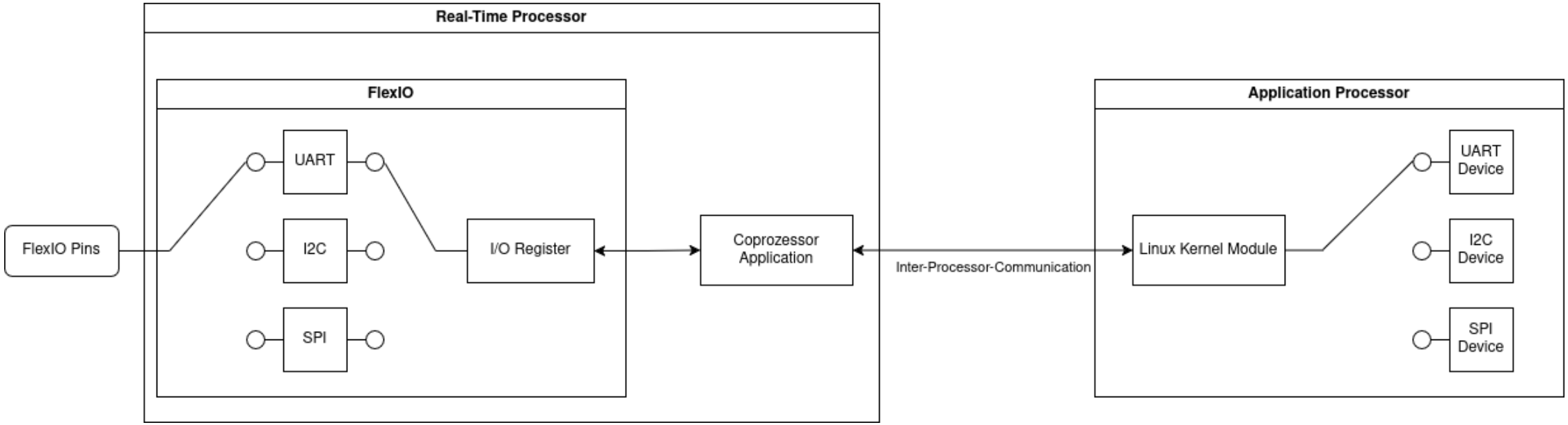
→ Lifecycle management

→ Resource management

RPMsg:

→ Bidirectional communication via shared memory

What is SDH?


Real-Time Processor
FlexIO
UART
I2C
SPI
I/O Register
FlexIO Pins
Coprozessor Application
Inter-Processor-Communication
Application Processor
Linux Kernel Module
UART Device
I2C Device
SPI Device

What is SDH?

Zephyr Side:

→ Every interface is implemented as own module

→ Every module runs in at least one separate thread

→ Each module opens its own RPMsg endpoint

→ Leading to great modularity

Linux Side:

→ Each interface is implemented as kernel module

→ Presents device in userland

→ Instance is created for every RPMsg endpoint

→ Building a bridge between co-processor and userland

→ Leads to a **single point of configuration**

What is SDH?

Configuration

→ On Zephyr side

→ Via a single Devicetree file

```
devices: sdh-devices {
    compatible = "phytec,sdh-devices";

    device1: sdh_device1 {
        compatible = "phytec,sdh-uart";
        status = "okay";
        device = <&uart4>;
    };
};
```

Abstracting..

Minimum structure of zephyr-module

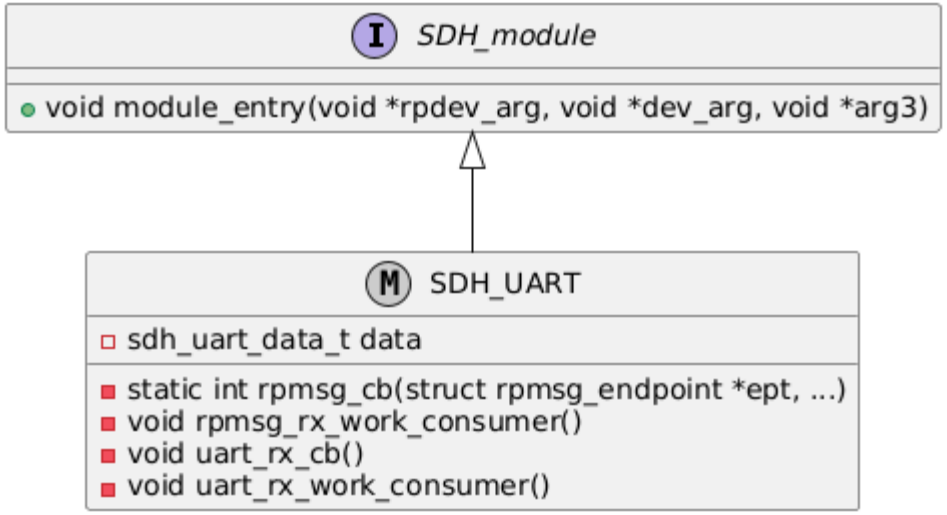
→ Entry point so it can be started as new thread

→ Init function that sets everything up

→ RPMsg callback and work consumer

→ If needed, interface callback and consumer

→ Instance based state data



Abstracting..

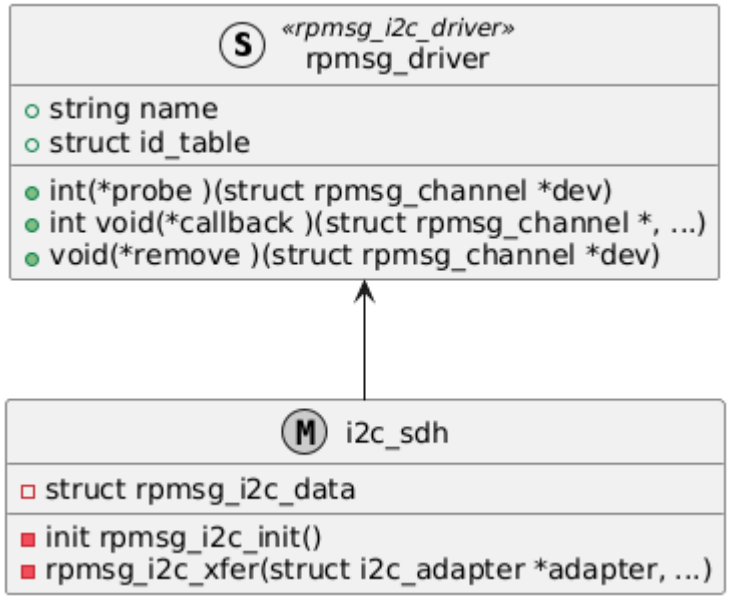
Minimum structure of linux-module

→ Entry points: **probe, callback, remove**

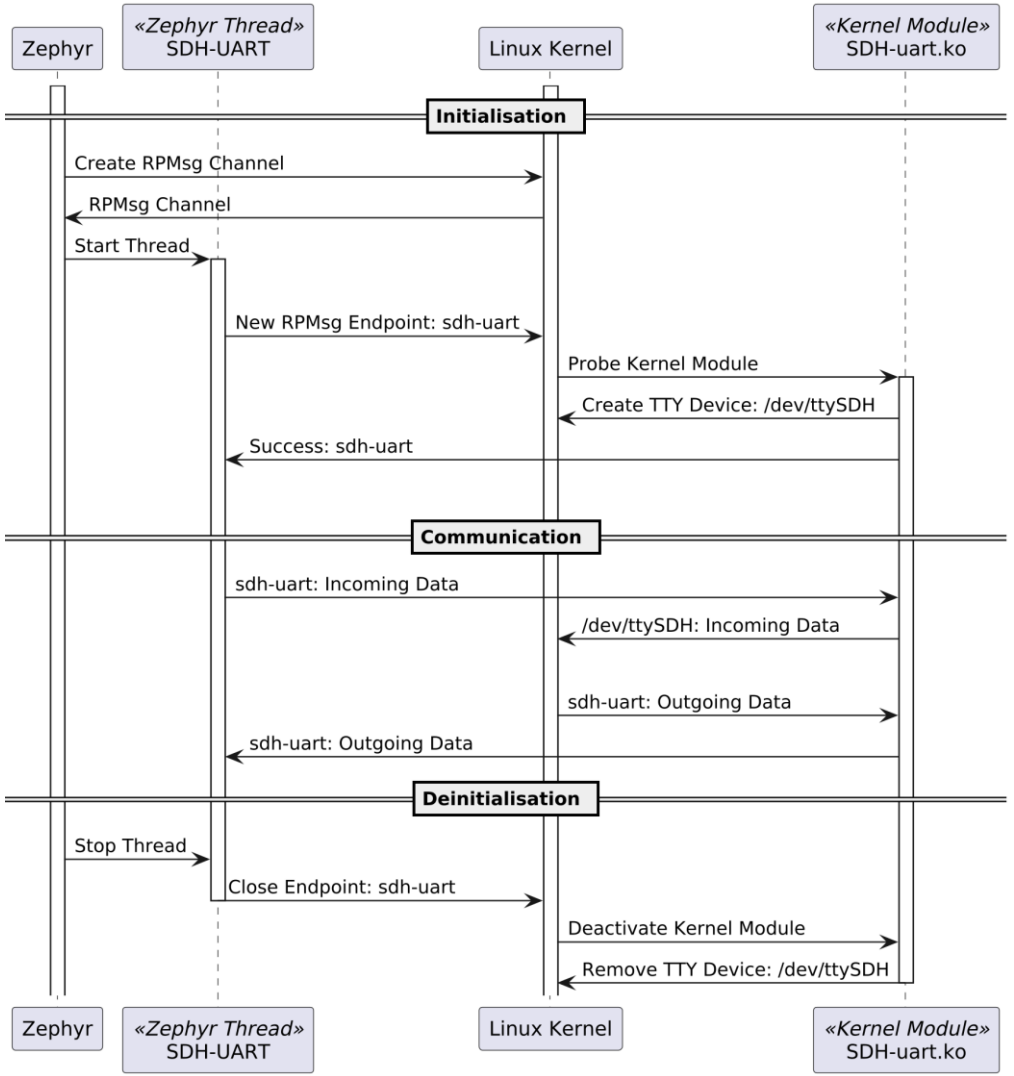
→ **RPMsg endpoint** name as “id”

→ Needs to setup device & callbacks in userland

→ Instance based state data



What is SDH?


Zephyr
«Zephyr Thread»
SDH-UART
Linux Kernel
«Kernel Module»
SDH-uart.ko
Initialisation
Create RPMsg Channel
RPMsg Channel
Start Thread
New RPMsg Endpoint: sdh-uart
Probe Kernel Module
Create TTY Device: /dev/ttySDH
Success: sdh-uart
Communication
sdh-uart: Incoming Data
/dev/ttySDH: Incoming Data
sdh-uart: Outgoing Data
sdh-uart: Outgoing Data
Deinitialisation
Stop Thread
Close Endpoint: sdh-uart
Deactivate Kernel Module
Remove TTY Device: /dev/ttySDH
Zephyr
«Zephyr Thread»
SDH-UART
Linux Kernel
«Kernel Module»
SDH-uart.ko

How are interfaces displayed in Linux?

Interface Type	Userspace
UART	/dev/ttySDHx
I²C	/dev/i2c-x

Live Demo

Conclusion

Pros:

→ “as many hardware interfaces as you want”

→ Single point of configuration

→ High level of abstraction

→ Every interface can be implemented if co-processor is fast enough

Cons:

→ More complicated: different software architectures on one device

→ Possibly slower than direct hardware access

→ Zephyr support for flexible interfaces lags behind

→ Cannot replace a FPGA solution

Conclusion

How can the Prototype evolve?

→ The co-processor is able to do more than just bridging data

It could…

Preprocess data

Wake up a sleeping Linux on an event

→ The Prototype is open source and can ideas an easily added

Conclusion

How can the Prototype evolve?

→ At the moment any interface is bridged via a kernel module

A better realization could be done by **using VirtIO virtual devices**:

→ VirtIO is normally used within virtualization of guest OS

→ Basically the remote processor is just a “guest” that presents interfaces

→ **This could make the system faster and simpler**

Conclusion

What I’ve picked up along the way:

→ Debugging multi processor applications can be really difficult

→ Configuring Kernel / Devicetree for OpenAMP is difficult

→ Trial and error can be really tricky, because of lacking debug possibilities

→ If OpenAMP works, it works like a charm

Thanks for listening!

https://github.com/phytec/zephyr-software-defined-hardware