PROJECT PRESENTATION

SoC Architecture: Inter-processor Communication

Matteo Fragassi, Davide Giuffrida,
Massimiliano Di Todaro, Balint Bujtor

Outline

- Project objectives
- Background
- Proposed solution
- Future work
- Conclusions

Project objectives

Main goal: Developement of a sensor node on a multi-processor platform

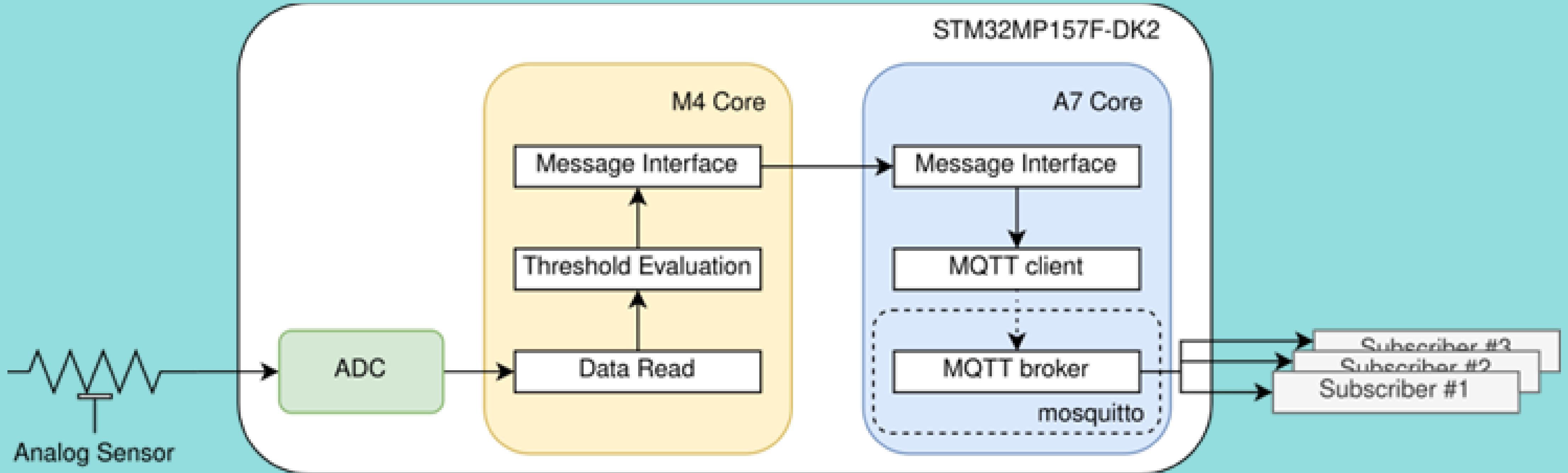
Specific tasks:

- Data acquisition via ADC
- Inter-Processor Communication
- Network interactions with MQTT
- Low energy consumption

Development board:

Based on **STM32MP157F-DK2 board**:

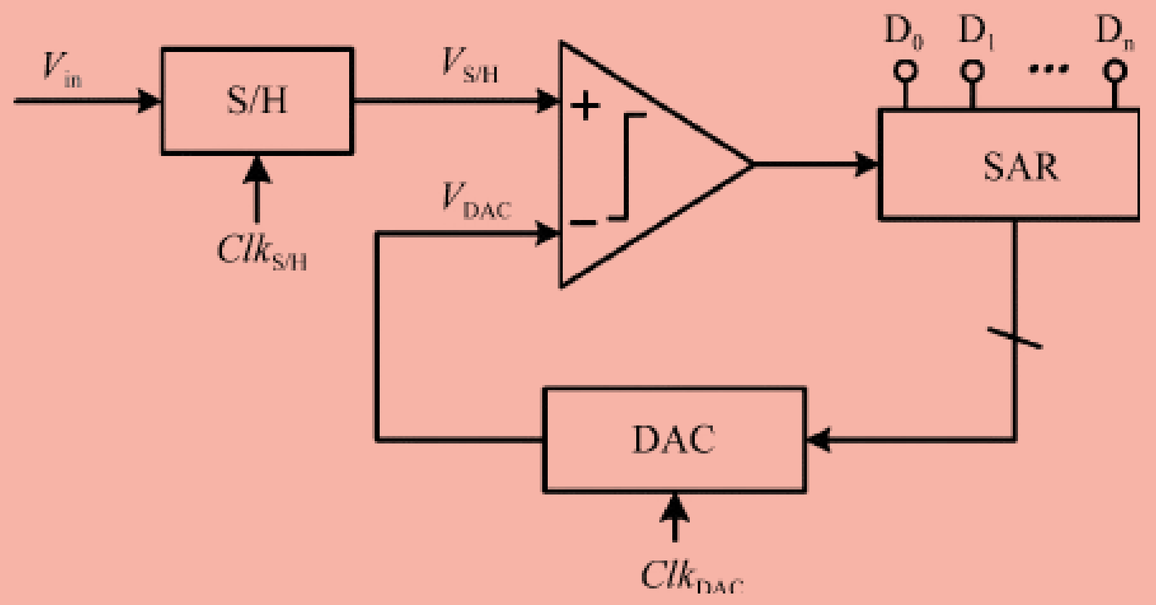
- Dual Cortex-A7 800 MHz 32-bit processor (MPU)
- Cortex-M4 32-bit co-processor (MCU)



Background

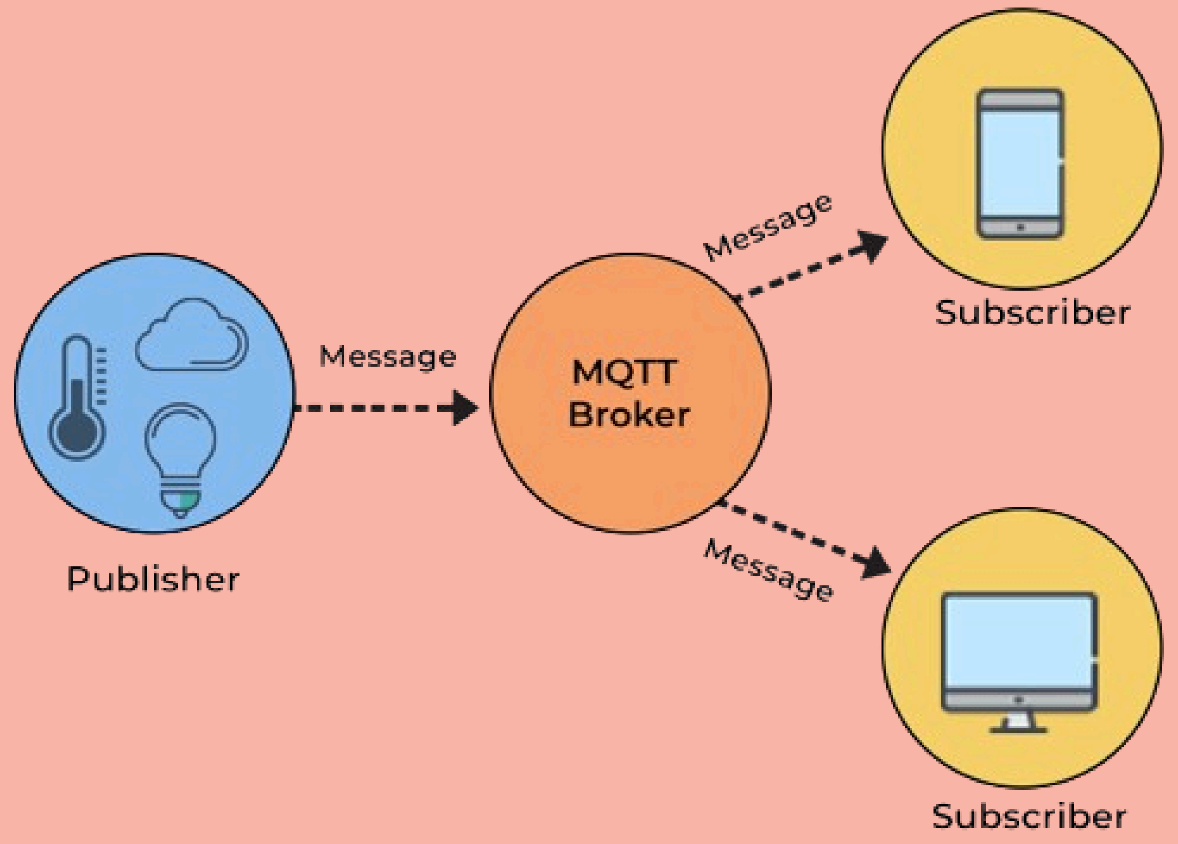
Analog-to-Digital Converter (ADC2)

- SAR with 0 V - 3.6 V **operation range**
- **20 channels** (6 internal) configured as Regular or Injected
- A variety of hardware and software **conversion triggers**
- Many **conversion modes**
  - Single, Continuous, Discontinuous
  - Single channel or Multiple channels (Scan mode)
- **Analog watchdogs** (AWD) to monitor input values



MQTT protocol

- **Publish/Subscribe** architecture
- **Topics** are the communication channels of the protocol
- The **Broker** is the node that delivers the messages published on a topic
- The **Quality of Service** (QoS) defines the reliability of a delivery



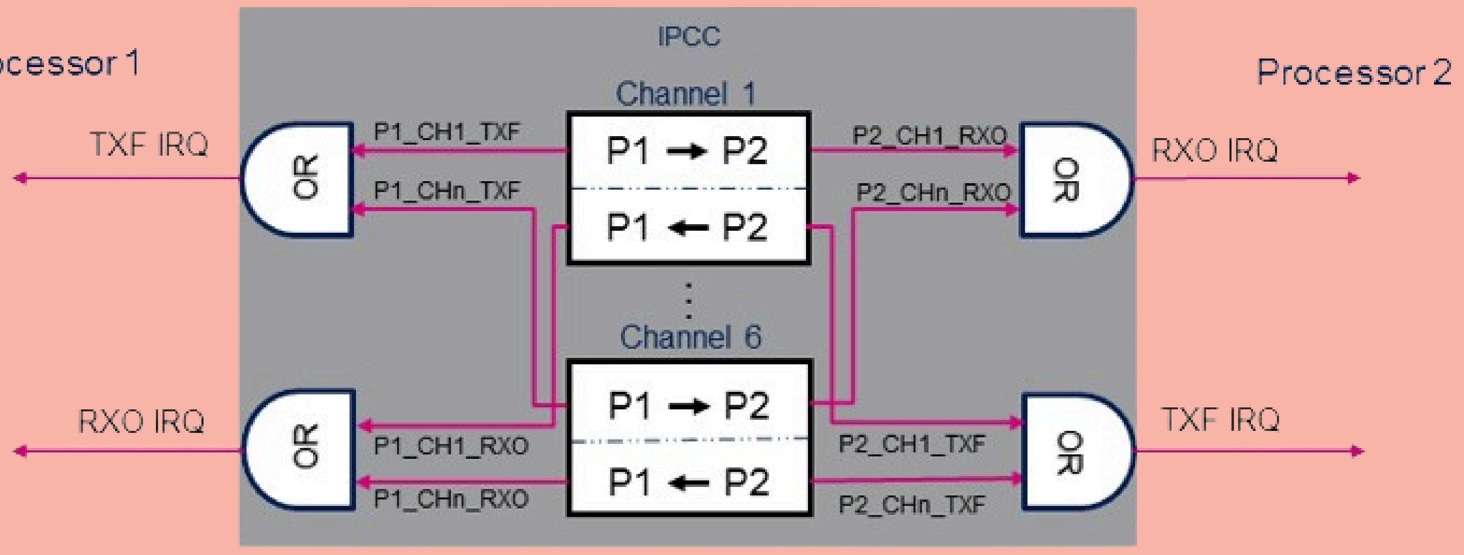
Background

Inter-processor communication (IPC)

The **IPC** is the set of hardware and software components that enables the communication between the MPU and the MCU subsystem.

There are mainly two **hardware** modules that support this feature:

- **IPC Controller** (IPCC)
  - 6 channels in total
  - 2 interrupts (RXO and TXF) for each processor
- **Shared Memory** (MCUSRAM)
  - 64KB in the MCUSRAM3 for the IPC buffers



The predefined **IPCC configuration** on STM32MP15x boards is a full-duplex communication on channel 1 and 2. In particular, channel 1 is used by the MCU to set RXO and by the MPU to set TXF, viceversa on channel 2.

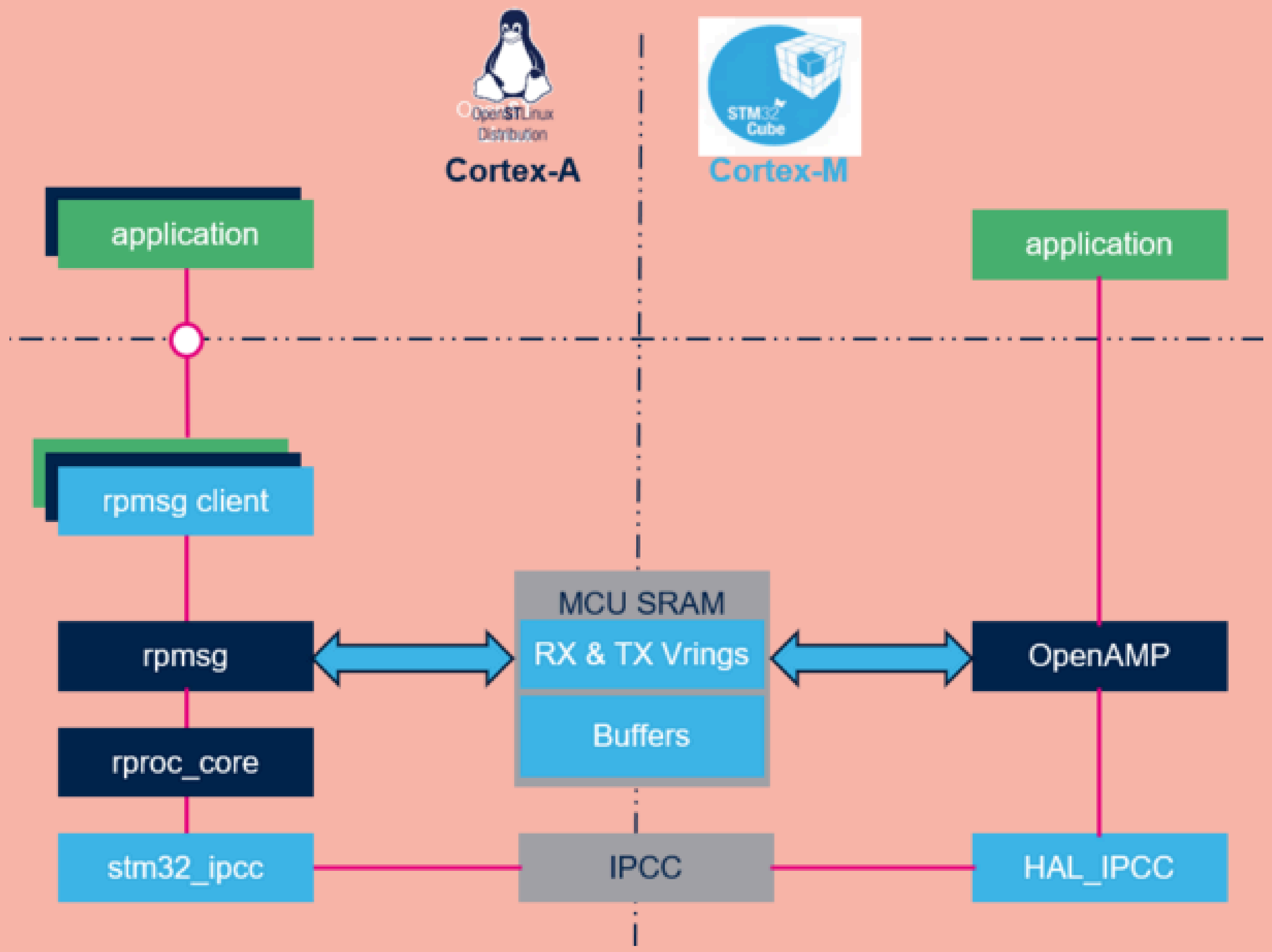
Background

Inter-processor communication (IPC)

The **software** stack is built with the following protocols:

- The **Mailbox** protocol drives the IPCC
- Remote Processor Messaging (**RPMsg**) offers a high-level communication interface
- **Remoteproc** enables the IPC on Linux depending on the information in the firmware resource table

The implementation of each protocol depends on the considered subsystem.



Background

Low-power states

The power consumption of a system is generally controlled by acting on two parameters that influence it the most: **clock frequency** and **supply voltage**.

The board allows to act on these parameters by controlling **clock** and **power** domains through different low-power operating modes:

- **Subsystem operating modes**: CRun, CSleep, CStop, CStandby (CA7 only)
- **System operating modes**: Run, Stop, LP-Stop, LPLV-Stop, Standby

Each **subsystem** can independently enter a subsystem certain subsystem operating mode while the entire **system** enters a system operating mode depending on the low-power state of both MCU and MPU.

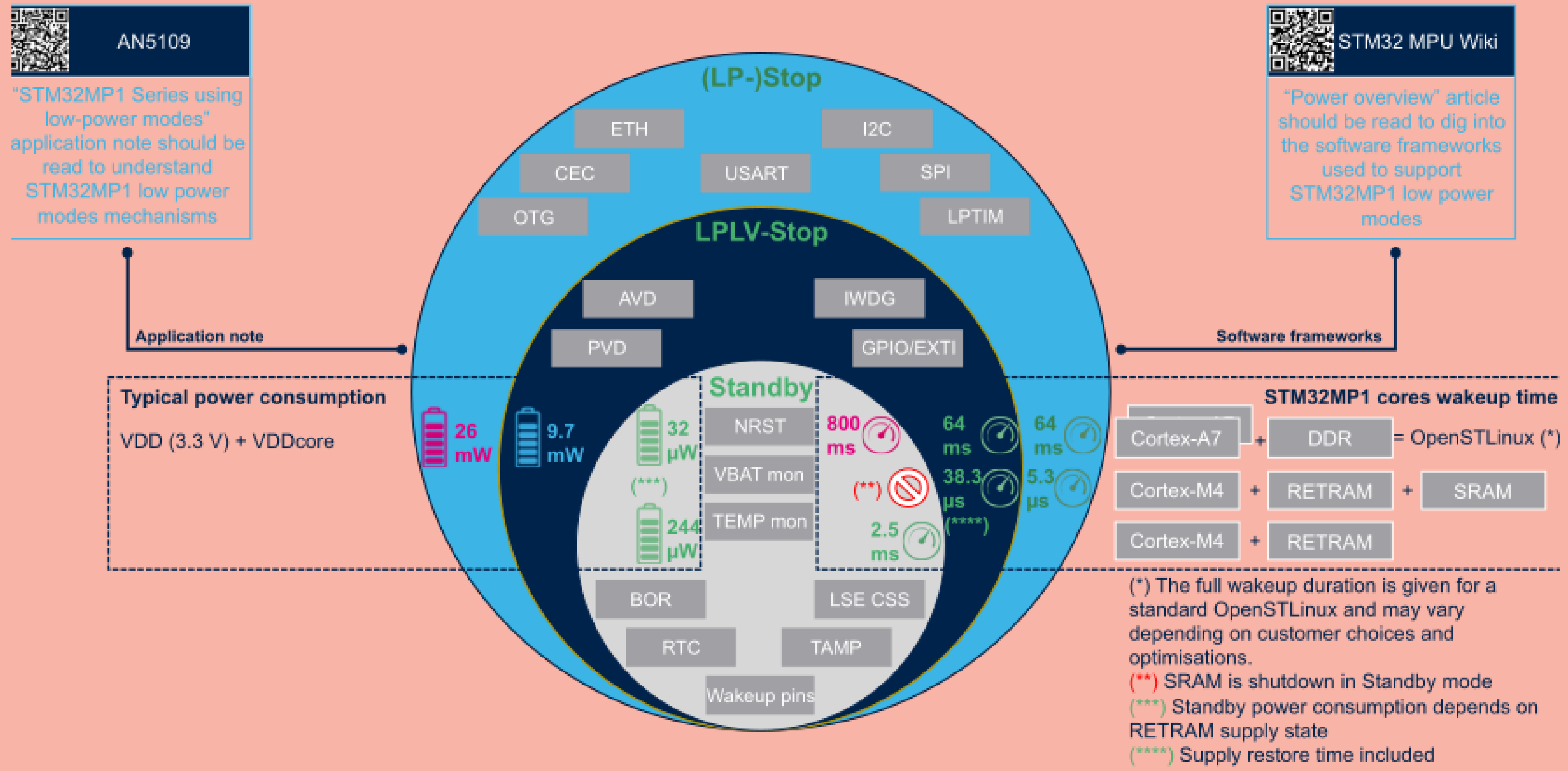
Background

Low-power states

Many **wake-up** sources are available for each subsystem depending on the targeted low-power mode.

The **software components** that manage the described low-power features are:

- HAL_PWR drivers on the MCU
- Linux Suspend framework (GenPD framework and PSCI standard) on the MPU



"STM32MP1 Series using low-power modes" application note should be read to understand STM32MP1 low power modes mechanisms

"Power overview" article should be read to dig into the software frameworks used to support STM32MP1 low power modes

(*) The full wakeup duration is given for a standard OpenSTLinux and may vary depending on customer choices and optimisations.
(**) SRAM is shutdown in Standby mode
(***) Standby power consumption depends on RETRAM supply state
(****) Supply restore time included

IMPORTANT!
When a suspension is requested on the MPU, the entered system operating mode depends on the power domains that cannot be switched off because of the active wake-up sources

Proposed solution

Proposed solution

Data acquisition on the CM4

- **DAC1 generates the analog signal** to feed ADC2
  - DAC1 output channel 1 internally connected to ADC2 channel 16
  - 5 different values set by *user_button_1*
  - The button triggers the EXTI_14 interrupt to change the DAC1 value

- ADC2, 12-bit resolution, Continuous mode, 0-2.9V input voltage
  - **AWD1** enabled **in interrupt mode**
  - Interrupt generated if the input value is outside the programmed window (high/low threshold)
  - Data retrieved inside the HAL callback

- **Data transmission to the MPU** if there are enough elements
  - Minimum number of data is configurable
  - **Only communication started by the MCU**
  - The ADC is stopped during this operation

Proposed solution

CM4-CA7 communication

Part of the **configurations** needed **to implement the IPC** are already present in the *OpenAMP_TTY_echo_wakeup* example used as starting point for the application:

- The IPC is implemented in **Direct Buffer Exchange Mode** (DBEM)
  - Low data transfer rate
  - RPMsg buffers contain effective data so no other memory allocation is needed

- **IPC Interfaces**
  - MCU: **Virtual UART** with “rpmsg-tty” as RPMsg service
  - MPU: **RPMsg TTY driver** @ /dev/ttyRPMsg<x> with x in [0,31]

The **remaining setup steps** are done at the startup of the kernel:

1. *setup.service* flashes the MCU
2. The MCU initializes OpenAMP and Virtual UART and waits for a message
3. *comm_setup.service* sends a message to communicate to the MCU its RPMsg endpoint address

Proposed solution

CM4-CA7 communication

On the **MCU** each **received message** is treated depending on its type:

- **THR**: New threshold for the threshold update process.
  In case of a succesful threshold update, the MCU sends an acknowledge to the MPU.
- **DATA_ACK**: The MPU acknowledges that the previously sent out-of-window data have been processed. It is also used to re-enable the data transmission after a suspension or threshold update operation.
- **WAIT_THR**: The user requests a threshold update
  a. If no unprocessed data are on the channel, the MCU informs the MPU that it is ready to receive the threshold
  b. Otherwise, the MCU waits for the MPU to process the data and send the new threshold
- **WAIT_SUSP**: Warns that the MPU is entering into a low-power state (currently not used)

Proposed solution

CM4-CA7 communication

On the **MPU** the communication channel is polled at each iteration of the main loop and, in case **new out-of-window data** are found, they are processed and transmitted to the user.

In case the other key operation of the application, i.e. the **threshold update**, is requested by the user the steps followed are:

1. A WAIT4THR message is sent to the MCU
2. The MPU reads the channel and proceeds, depending on the message
    a. If there are new data, they are processed and transmitted to the user client before continuing with the update process
    b. If RDY4OP is read, the process can continue normally
3. The new threshold is delivered to the MCU
4. An acknowledge (THR_SET) informs the MPU of the successful threshold update
5. The MPU acknowledges the threshold update process and re-enables the MCU to send out-of-window data

Proposed solution

Interaction on the network (MQTT)

Actors involved:

- **Broker**: a standard MQTT broker (e.g. Mosquitto).
- **Board Client**: interface executed on the CA7 that handles the CA7-CM4 interactions and the board-user network communication.
- **User client**: interface used by the final user of the application to request threshold updates and gather out-of-window data.

The current version of the application does not support any kind of **low-power functionality** on the CA7 subsystem because of a conflict with the MQTT client. However, an additional client is provided in case this function was available and the Wake-on-Lan was configured to act on magic packet activity:
the **Proxy client**

Proposed solution

Interaction on the network (MQTT)

MQTT topics:

- **BOARD_TOPIC_STATUS**: used by the board client to inform the user about its status
- **BOARD_TOPIC_ERROR**: used by the board client to warn the user about protocol errors
- **THR_TOPIC_ACK**: topic where the board notifies the user about a successful threshold update
- **THR_TOPIC_UPDATE**: topic where the user client publishes the threshold when it needs to be updated.
- **DATA_TOPIC**: the user receives out-of-window data through this topic

Additional topics w/Proxy client:

- **PROXY_TOPIC_STATUS**: used by the proxy to inform the user about its status
- **PROXY_TOPIC_ERROR**: used by the proxy to warn the user about internal errors
- **PROXY_TOPIC_WAKEUP**: topic where the user client request to the proxy to wake up the boards targeted by the incoming threshold update process

Proposed solution

Interaction on the network (MQTT)

The steps followed by the **user client** during the **threshold update** process are:

1. The user client publishes the new threshold and the list of boards to update on *THR_TOPIC_UPDATE*
2. The board client proceeds as previously described while the user client waits to receive acknowledges on *THR_TOPIC_ACK*
3. The client terminates the update process after all the boards have sent a message (FINISHED) or after a period of time set by the user (ABORTED)

The role of the user client in the **out-of-window data exchange** is simply to listen on *DATA_TOPIC* and log the received messages.

IMPORTANT!
The **threshold update** process may require additional steps if **low-power** features were active for the MPU subsystem

Proposed solution

Power states

The **MCU can enter the CSleep state** when it does not communicate with the MPU:

- Not waiting for MPU messages (e.g. DATA_ACK, THR)
- Not enough data in the buffer to start a transmission

The MCU is woken up with the following interrupts:

- IPCC_RXO interrupt when the MPU is requesting to start an operation
- AWD interrupt when a new out-of-window data is detected

In general, the CSleep mode is *not deep* and it is accessed through a WFI so the MCU can be woken up with every interrupt in the NVIC.

Even though the code that implements **low-power features on the MPU** is provided (*ipcc-interface.py*), this feature is not integrated in the final version because of an **error** of the Ethernet DMA initialization after a wake-up.

Because the subsystems mode are CSleep and CRun, the whole system always remains in **Run mode.**

The future of the sensor node

The application could be further developed in order to meet the targeted requirements and improve its functionalities. Some of the points that can be addressed are:

- The error of the Ethernet DMA after waking up from suspension

- Change the network architecture according to the specific application environment
    - Mix a publish/subscribe and a client/server approach
    - Use the Wake-on-Lan in unicast or magic packet mode
    - Adapt the provided nodes and use some additional ones (e.g. the proxy client)

- Integrate the MCU error handling in the MPU-MCU and board-user communication protocols

A more detailed explanation and additional points are reported in the documentation of the application.

Conclusions

Despite the early state of the application, the objectives achieved with the proposed sensor node are:

- A generic, customizable and highly scalable data acquisition architecture
- Exploitation of most of the features offered by the STM32MP157F-DK2 board
- Simple and easily extendable inter-processors communication protocol
- Scalable network communication with MQTT
- Low energy consumptions thanks to the board operating modes and the low power MQTT clients

THANK YOU FOR THE ATTENTION!