Shared system resources on multi-processor system

Lionel DEBIEVE

#lfelc  @twitterhandle
Lionel DEBIEVE
Software Engineer @ STMicroelectronics
- STM32 MPU development : STM32MP15
- Boot and Security focus
- Contributing to open source projects:
  - Trusted Firmware A
  - OP-TEE OS
  - U-Boot
  - Linux kernel (Crypto drivers)
Agenda

• Multi-processor design
• Shared resources
  • What are shared resources?
  • Manage shared resources
• The SCMI protocols
  • Description
  • Associated implementations
Multi-processor design
A standard SoC architecture

- A complex IC that integrates multiples functional elements into a single chip
- Many embeds multiple heterogeneous processors
- Assets:
  - Isolate functions on specialized and more efficient accelerators
  - Increase performance
  - Decrease power consumption
  - Reduce over all cost
- Implies to manage shared resources
Where do you find it?

• **Mobile**
  – Modem (5G) / Application processor and GPU (Android)
    • Ex: Mediatek Dimensity, Samsung Exynos…

• **IoT gateway**
  – Connectivity (Ethernet, CAN, …) / Web gateway
    • Ex: ST STM32MP1, NXP i.MX6, …

• **AI**
  – Sensors (Camera) / AI (image recognition) / Web gateway
    • Ex: Nvidia Xavier, …

• …
Shared resources
What are shared resources?

- **Exclusive resources**: A peripheral clock, interrupt, reset,... which is assigned and controlled by an entity without conflict.

- **Shared resources**: central SoC resources shared by several processors or peripherals and so that can be used by several entities at the same time:
  - GPIOs, regulators, clocks, resets,…
  - Common registers banks (platform dependent)
What are shared resources?

- Shared resources are used by CPUs:
  - For their own management:
    - DVFS
    - Power control
    - ...
  - To manage their peripherals:
    - Runtime power optimization
    - Reset
    - ...

![Diagram showing shared resources](image-url)
Shared resources in a SoC

- **Ex: STM32MP1**
Managed shared resources

• How to manage shared resources in such complex system?
  – Dedicate a single entity that is responsible for the shared resources: a system controller.
  – A system controller must centralize the global knowledge of the shared resources state.
The system controller

• The system controller must follow some rules to properly managed shared resources:
  – Reliability:
    • Dedicated execution context
    • Control and identify access to shared resources
  – Flexibility:
    • Must be adapted depending on the platform
    • Must manage peripheral assignment changes depending on the executed use case
  – Accessibility:
    • Stable API: Standard interface to access the system controller
The SCMI Protocols: Access to the system controller
Arm® defined SCMI protocols (System Control and Management Interface)

Standard specification:
Defines messages exchanged to discover and expose services between a client (agent) and a Server(platform).

Two layers:
- SCMI Protocols (Clocks, …)
- SCMI Transport (Mailbox, …)
• **Messages**
  - Agent to Platform (A2P): request messages
  - Platform to Agent (P2A): synchronous responses, notifications or delayed responses

• **Channels:**
  - One or more dedicated channels per agent
  - Standard channel: use to transmit exchange requests and responses between an agent and the system controller.
  - FastChannel: Unidirectional channel specific to performance management protocol (low latency).
• SCMI specifications version 3.0 is available:
  – Linux kernel (v5.9) support SCMI 2.0:
    • First introduced in v4.17 (early 2018)
    • Clock, reset, power, performance, sensor
    • Transport Mailbox
    • SMC, Notifications: recently added in v5.9
  – Change in 3.0:
    • Voltage regulator, sensor extensions
  – Next: QoS, …
Possible integration schemes (1/3)

• **System controller on a dedicated processor**
  - **Ex: Arm® Juno:**
    - Dedicated Cortex-M3 with SCP firmware implementing SCMI server
      - Control PMIC (regulators)
      - Clock control, voltage, power gating
    - Secure: independent execution context
    - Increase SoC footprint (cost)

SCMI Server implementation
Possible integration schemes (2/3)

- System controller on the application processor
  - SCMI Virtualization support: SCMI Server in a dedicated VM
  - Stratos: [https://projects.linaro.org/projects/STR](https://projects.linaro.org/projects/STR)
Possible integration schemes (3/3)

- System controller on the application processor
  - Ex: STM32MP15
    - Cortex-A7 based
    - TrustZone® usage
    - Secure execution context
  - Pros:
    - Reduced SoC footprint
    - Cost reduction
  - Cons:
    - Applicative processor becomes the master in the system
SCMI: STM32MP1 implementation
SCMI on STM32MP1

- **Implementation overview:**
  - Messaging based on shared memory and SMC calls
  - Shared resources accesses only by secure context

- **Supported implementations:**
  - OP-TEE OS integration
  - Trusted Firmware-A

**Legend:**
- **PL:** Privilege Level
- **PMIC:** Power Management Integrated Circuit
- **SMC:** Secure Monitor Call

**Diagram Notes:**
- SCMI Clock
- SCMI Reset
- SCMI Power domain
- SCMI Server
- SCMI Driver
- RCC Driver
- PWR Driver
- PMIC Driver
- SCMI Clock Driver
- SCMI Reset Driver
- SCMI Voltage Driver
- SMC Transport
- Mailbox Transport

**Diagram Components:**
- Cortex-A7 Secure (TZ)
- Cortex-A7 Non-Secure
- User space
- Linux Applications
- Application frameworks
- Secure OS
- Linux Kernel space
- User space
- Trusted Applications
- Secure OS
- OP-TEE OS
  - / Trusted Firmware-A
- Trusted Applications
- Linux Applications
- Application frameworks
- User space
- Secure OS
- SCMI Server
- SMC handler

**Additional Notes:**
- SCMI on STM32MP1
- Implementation overview
- Supported implementations
- Privilege Level (PL)
- Power Management Integrated Circuit (PMIC)
- Secure Monitor Call (SMC)
SCMI on STM32MP1: Clock sequence

Clock ID correspond to SCMI clock provider

Synchronous command
- Add Clock Protocol ID
- Add Clock Message ID

Write in shared memory and call SMCC

Parse the message and detect clock protocol ID

Update shared memory with return status

return from SMCC
Linux kernel SCMI usage on STM32MP1

Shared memory definition

```c
scmi_sram: sram@2ffff000 {
    compatible = "mmio-sram";
    reg = <0x2ffff000 0x1000>;
    #address-cells = <1>;
    #size-cells = <1>;
    ranges = <0 0x2ffff000 0x1000>;
}

scmi0_shm: scmi_shm@0 {
    reg = <0 0x80>;
}
```

SCMI channel description

```c
firmware {
    scmi0: scmi-0 {
        compatible = "arm,scmi-smc";
        #address-cells = <1>;
        #size-cells = <0>;
        arm,smc-id = <0x82002000>;
        shmem = <&scmi0_shm>;

        scmi0_clk: protocol@14 {
            reg = <0x14>;
            #clock-cells = <1>;
        }

        scmi0_reset: protocol@16 {
            reg = <0x16>;
            #reset-cells = <1>;
        }
    }
}
```

Clock Protocol

Reset Protocol
STM32MP15 DTS bindings using SCMI protocol

dsi: dsi@5a000000 {
    compatible = "st,stm32-dsi";
    reg = <0x5a000000 0x800>;
    clocks = <&rcc DSI_K>, <&scmi0_clk CK_SCMI0_HSE>, <&rcc DSI_PX>;
    clock-names = "pclk", "ref", "px_clk";
    resets = <&rcc DSI_R>;
    (...)
};

m4_rproc: m4@10000000 {
    compatible = "st,stm32mp1-m4";
    reg = <0x10000000 0x40000>,
         <0x30000000 0x40000>,
         <0x38000000 0x10000>;
    resets = <&scmi0_reset RST_SCMI0_MCU>;
    (...)
};
Status on STM32MP1

• **Current Status:**
  – Clocks, resets are fully implemented
  – Possibility to embed SCMI server in OP-TEE or Trusted Firmware-A
  – OP-TEE: SCMI server merged upstream
  – TF-A: Merged in June 2020 (ready for v2.4)
  – U-Boot: SCMI agent driver merged next for v2021.01

• **Next steps:**
  – Regulators
  – Performance
  – Coprocessor SCMI agent for M4
    • Current implementation is using another resource manager:
Links

- STM32MP1 SCMI: [https://wiki.st.com/stm32mpu/wiki/SCMI_on_STM32MP1](https://wiki.st.com/stm32mpu/wiki/SCMI_on_STM32MP1)
- Arm SCP Firmware: [https://github.com/ARM-software/SCP-firmware](https://github.com/ARM-software/SCP-firmware)

- SCMI upstream:

- Logos are propriety of their respective owners
Thank you!