Introduction to the System Controller Firmware on i.MX 8 Application Processor

Manuel Rodriguez

Automotive Field Applications Engineer

October 2019 | Session #AMF-AUT-T3889

SECURE CONNECTIONS FOR A SMARTER WORLD

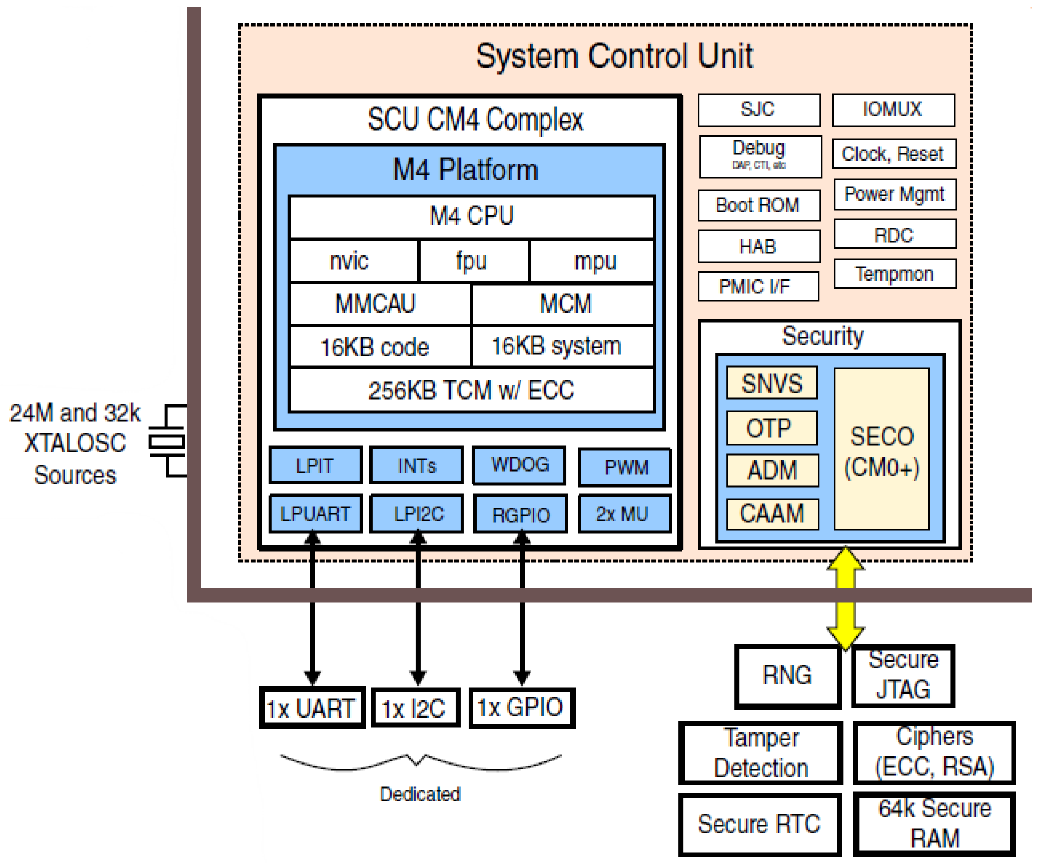
Agenda

- System Controller Unit and I.MX8 Architecture
- System Controller Firmware Overview and Getting Started
- System Controller Firmware Services
  - Power Management Service
  - Resource Management Service
  - Pad Configuration Service
  - Timer Service
  - Miscellaneous Service

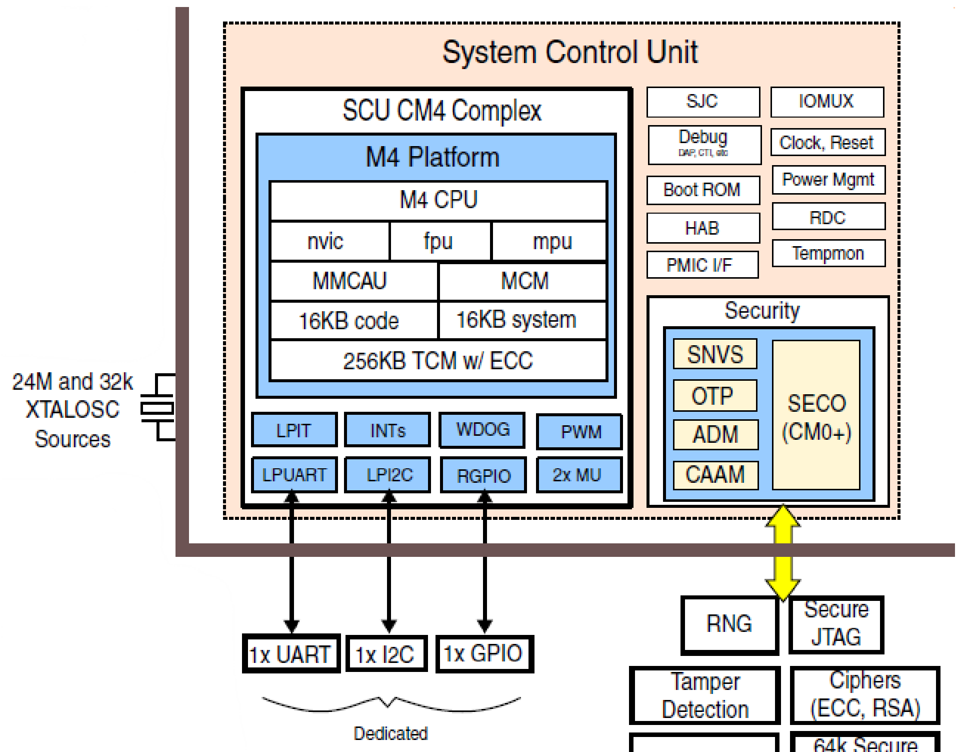
System Controller Unit and I.MX8 Architecture

Introduction

- i.MX8 features a module dedicated to:
  - Boot Management
  - Power Management
  - Clock and Reset Management
  - IO Configuration
  - Resource Partitioning



SCU – Architecture Overview

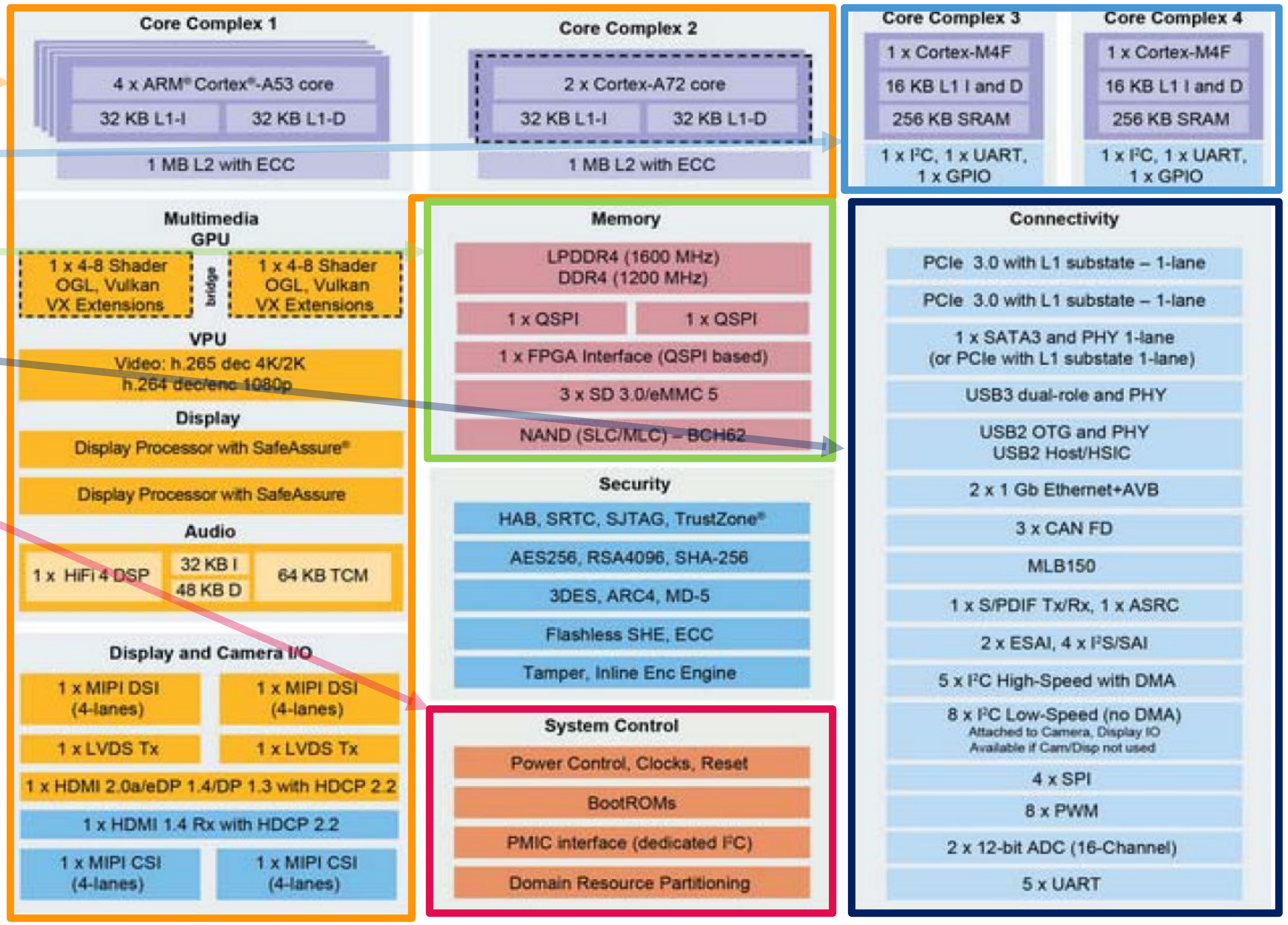


- System Controller Unit subsystem is comprised of
  - 1x Cortex-M4 processor
  - A set of peripherals
    - 1x TPM, 1x UART, 1x I2C, 8x GPIOs, 4x MUs
  - This is the first processor to boot in the design

- Security subsystem is made of
  - 1x Cortex-M0 processor running at 133MHz
    - Handles security services for other processors
  - A set of cryptographic related hardware accelerators
    - RSA, ECC, AES, DES/3DES, SHA-1, SHA-2, MD5, HMAC, RNG

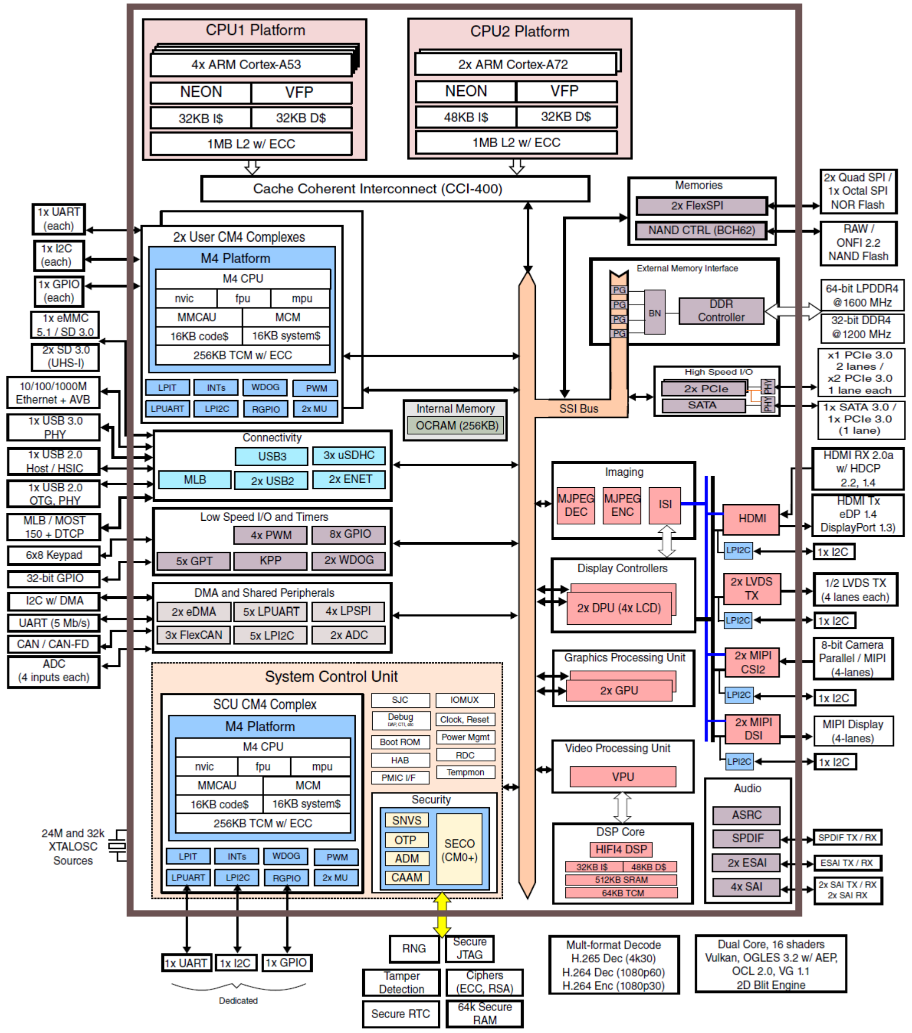
High-level Block Diagram i.MX8QM

- Application domain
- Real time domain
- Memory
- Connectivity peripherals
- System Control Unit

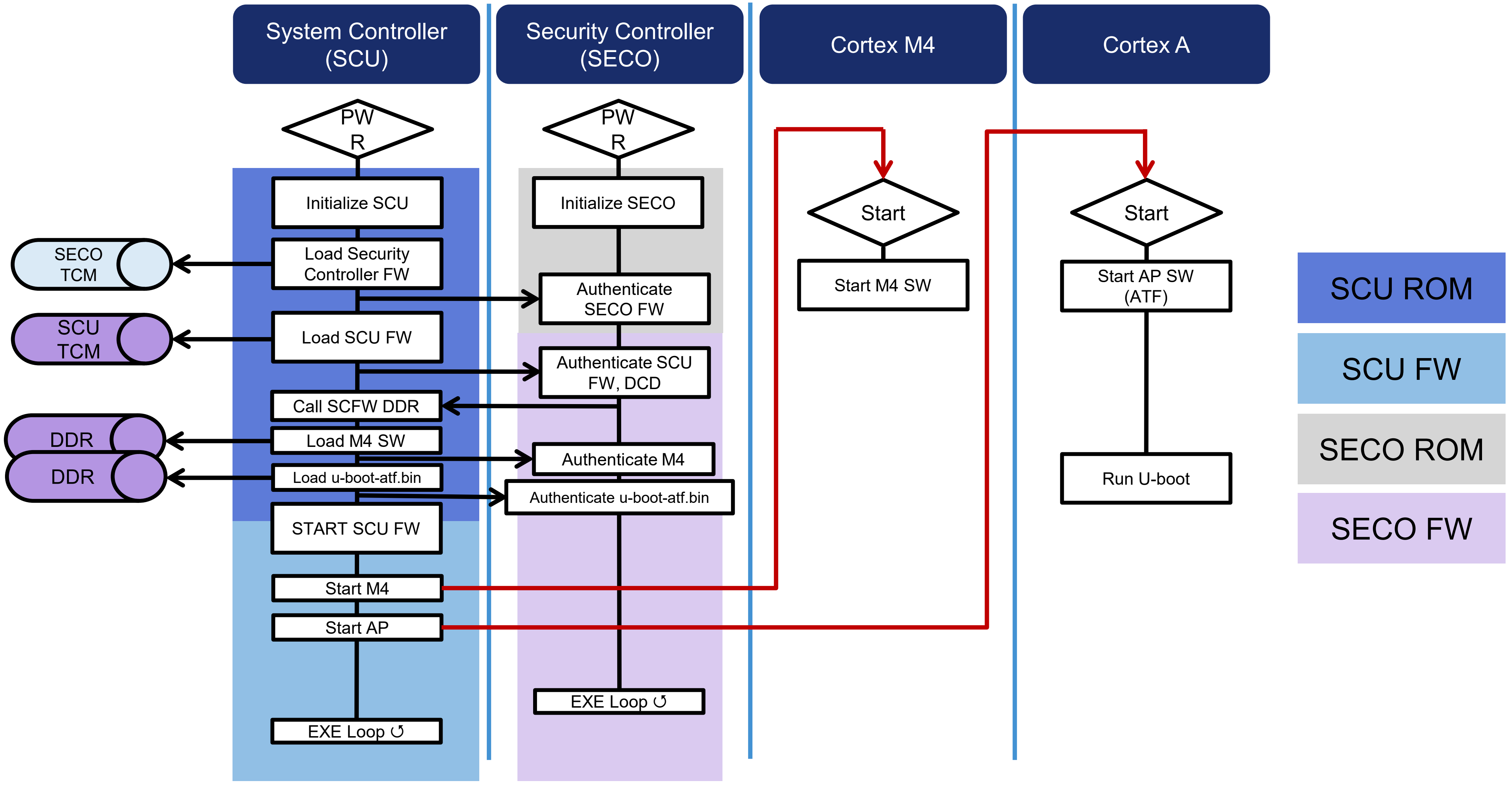

Core Complex 1
4 x ARM® Cortex®-A53 core
32 KB L1-I
32 KB L1-D
1 MB L2 with ECC
Core Complex 2
2 x Cortex®-A72 core
32 KB L1-I
32 KB L1-D
1 MB L2 with ECC
Core Complex 3
1 x Cortex-M4F
16 KB L1 I and D
256 KB SRAM
1 x I²C, 1 x UART, 1 x GPIO
Core Complex 4
1 x Cortex-M4F
16 KB L1 I and D
256 KB SRAM
1 x I²C, 1 x UART, 1 x GPIO
Multimedia
GPU
1 x 4-8 Shader OGL, Vulkan VX Extensions
bridge
1 x 4-8 Shader OGL, Vulkan VX Extensions
VPU
Video: h.265 dec 4K/2K h.264 dec/enc 1080p
Display
Display Processor with SafeAssure®
Display Processor with SafeAssure
Audio
1 x HiFi 4 DSP
32 KB I
48 KB D
64 KB TCM
Display and Camera I/O
1 x MIPI DSI (4-lanes)
1 x MIPI DSI (4-lanes)
1 x LVDS Tx
1 x LVDS Tx
1 x HDMI 2.0a/eDP 1.4/DP 1.3 with HDCP 2.2
1 x HDMI 1.4 Rx with HDCP 2.2
1 x MIPI CSI (4-lanes)
1 x MIPI CSI (4-lanes)
Memory
LPDDR4 (1600 MHz) DDR4 (1200 MHz)
1 x QSPI
1 x QSPI
1 x FPGA Interface (QSPI based)
3 x SD 3.0/eMMC 5
NAND (SLC/MLC) – BCH62
Security
HAB, SRTC, SJTAG, TrustZone®
AES256, RSA4096, SHA-256
3DES, ARC4, MD-5
Flashless SHE, ECC
Tamper, Inline Enc Engine
System Control
Power Control, Clocks, Reset
BootROMs
PMIC interface (dedicated I²C)
Domain Resource Partitioning
Connectivity
PCIe 3.0 with L1 substate – 1-lane
PCIe 3.0 with L1 substate – 1-lane
1 x SATA3 and PHY 1-lane (or PCIe with L1 substate 1-lane)
USB3 dual-role and PHY
USB2 OTG and PHY USB2 Host/HSIC
2 x 1 Gb Ethernet+AVB
3 x CAN FD
MLB150
1 x S/PDIF Tx/Rx, 1 x ASRC
2 x ESAI, 4 x I²S/SAI
5 x I²C High-Speed with DMA
8 x I²C Low-Speed (no DMA) Attached to Camera, Display IO Available if Cam/Disp not used
4 x SPI
8 x PWM
2 x 12-bit ADC (16-Channel)
5 x UART

Why Do We Need a System Control Unit?

- Each one of these peripherals requires access to a subset of resources on the chip (clocks, connectivity, memory).
- The System Control Unit simplifies software development by providing a high-level abstraction to key system functionality.
- The SCU takes care of managing these resources.
  - Resource allocation/deallocation
    - Imagine two peripherals using the same clock, peripheral one is shut down and it asks for the clock to be disabled while peripheral 2 is still using it, the SCU takes care of handling this situation and keeps the clock alive.
  - Resource partition
    - A system can be configured in such a way that the application subsystem (e.g. Linux) is oblivious of the existence of the real time domain (e.g. FreeRTOS). Just like having two different chips in the same package.



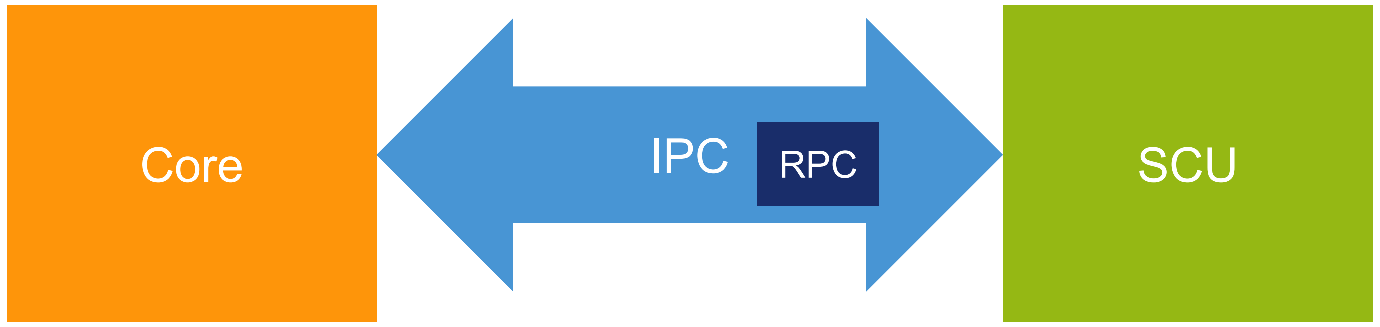
i.MX8 High-level Boot Sequence



System Controller Firmware Overview and Getting Started

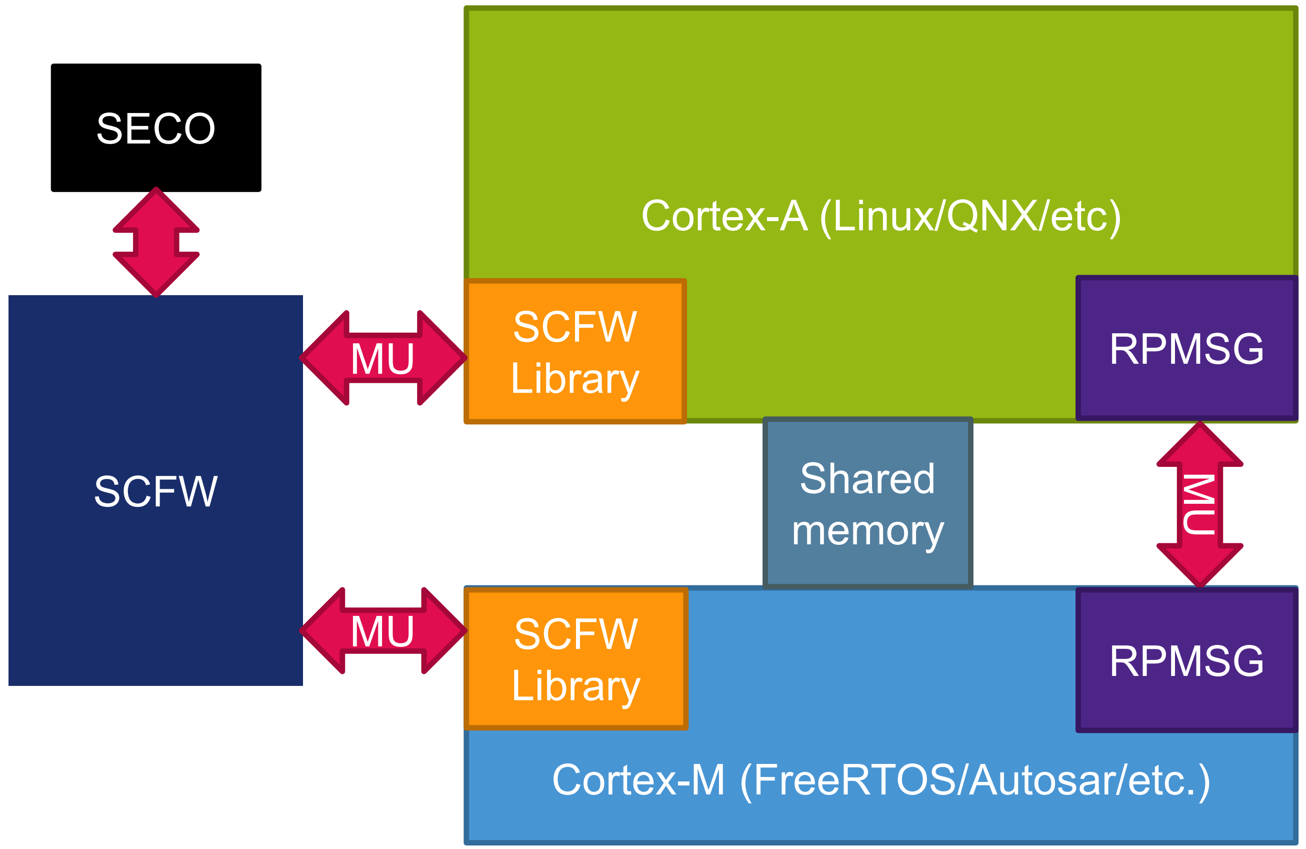
Introduction

- The System Controller Firmware (SCFW) runs on the System Controller Unit and is responsible for managing requests from other cores in the system.
  - Other software components communicate with System Controller Unit (SCU) via an Application Programming Interface (API) library
  - This library makes Remote Procedure Calls (RPC) via an underlying Inter-Processor Communication (IPC) mechanism



System Controller Firmware and Application

- Operating Systems can configure the device through an Application Programming Interface (API).
- Development is simplified thanks to the API that provides control of key system functionality.
- Each OS features a library that allows it to interact with the SCFW.

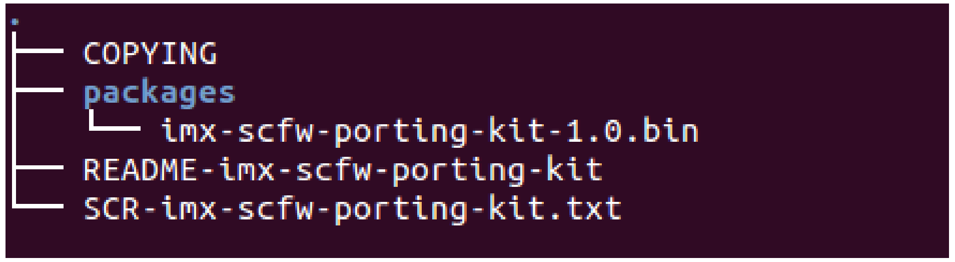


System Controller Firmware Porting Kit – Introduction

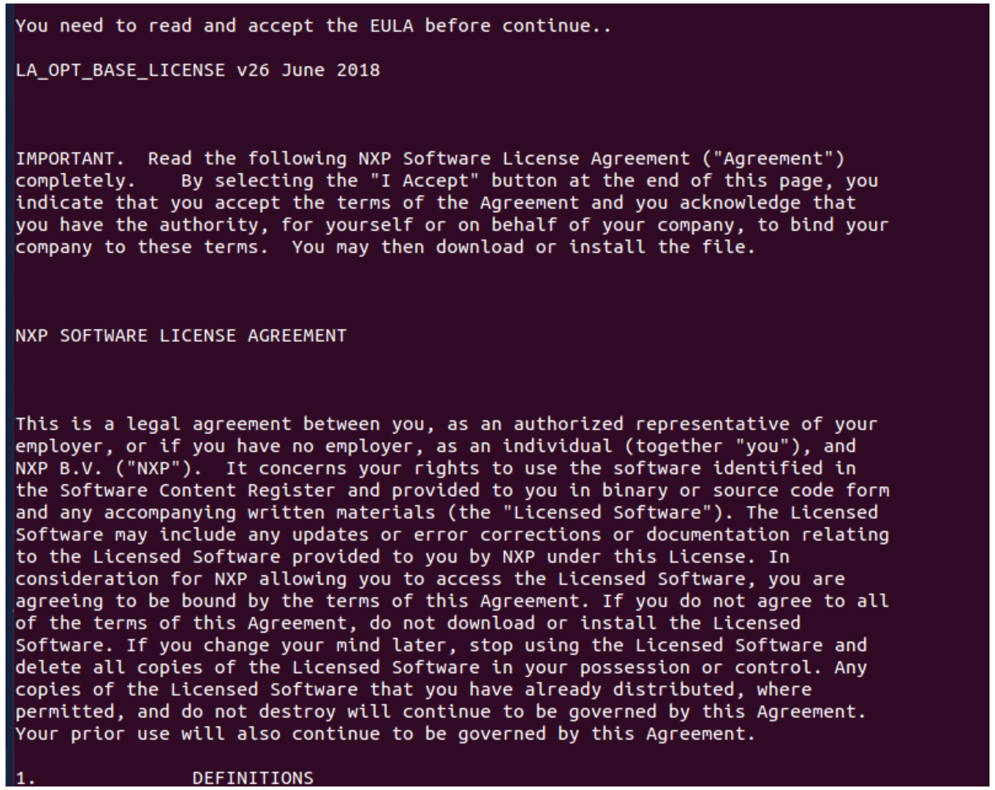
- The SCFW is distributed through a “Porting Kit”
- This porting kit allows you to build the SCFW and modify the board dependent parts of it such as:
  - Power Management Integrated Circuit (PMIC) information
  - CPU/GPU Power supplies
  - Board components that need to be powered by the PMIC such as cameras, deserializers, sensors, etc…
- A Linux host is required to build it
- The porting kit is downloaded as a package on the Yocto BSP
- This is the first step to bring-up a board

System Controller Firmware Porting Kit – Introduction

- The README file contains the instructions to extract the porting kit
  - $ cd packages
  - $ chmod a+x imx-scfw-porting-kit-1.0.bin
  - $ ./imx-scfw-porting-kit-1.0.bin
- You will be prompted to accept the End-User License Agreement or EULA



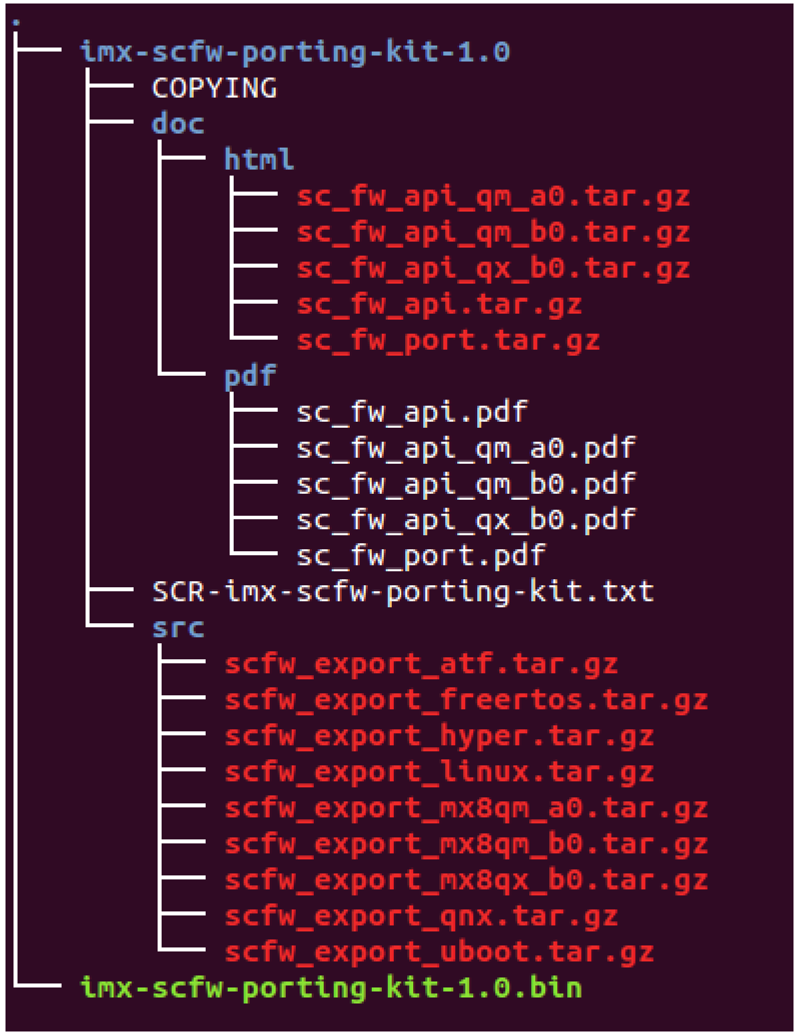
```
.
├── COPYING
├── packages
│   └── imx-scfw-porting-kit-1.0.bin
├── README-imx-scfw-porting-kit
└── SCR-imx-scfw-porting-kit.txt
```



```
You need to read and accept the EULA before continue..

LA_OPT_BASE_LICENSE v26 June 2018



IMPORTANT.  Read the following NXP Software License Agreement ("Agreement")
completely.    By selecting the "I Accept" button at the end of this page, you
indicate that you accept the terms of the Agreement and you acknowledge that
you have the authority, for yourself or on behalf of your company, to bind your
company to these terms.  You may then download or install the file.



NXP SOFTWARE LICENSE AGREEMENT



This is a legal agreement between you, as an authorized representative of your
employer, or if you have no employer, as an individual (together "you"), and
NXP B.V. ("NXP").  It concerns your rights to use the software identified in
the Software Content Register and provided to you in binary or source code form
and any accompanying written materials (the "Licensed Software"). The Licensed
Software may include any updates or error corrections or documentation relating
to the Licensed Software provided to you by NXP under this License. In
consideration for NXP allowing you to access the Licensed Software, you are
agreeing to be bound by the terms of this Agreement. If you do not agree to all
of the terms of this Agreement, do not download or install the Licensed
Software. If you change your mind later, stop using the Licensed Software and
delete all copies of the Licensed Software in your possession or control. Any
copies of the Licensed Software that you have already distributed, where
permitted, and do not destroy will continue to be governed by this Agreement.
Your prior use will also continue to be governed by this Agreement.

1.              DEFINITIONS
```

System Controller Firmware Porting Kit – Introduction

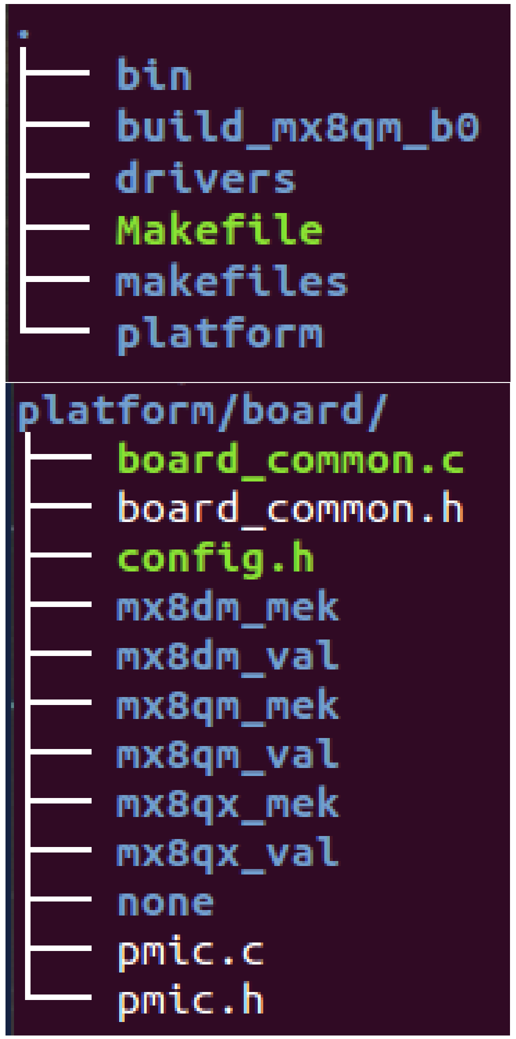
- The porting kit contains the documentation and required source to build the SCFW
- The sc_fw_api documents contains usage information on the API and resources/clock/controls information for each supported SoC
- The libraries for each supported OS as well as the source to build the SCFW can be found under the "src" directory



```
.
├── imx-scfw-porting-kit-1.0
│   ├── COPYING
│   ├── doc
│   │   ├── html
│   │   │   ├── sc_fw_api_qm_a0.tar.gz
│   │   │   ├── sc_fw_api_qm_b0.tar.gz
│   │   │   ├── sc_fw_api_qx_b0.tar.gz
│   │   │   ├── sc_fw_api.tar.gz
│   │   │   └── sc_fw_port.tar.gz
│   │   └── pdf
│   │       ├── sc_fw_api.pdf
│   │       ├── sc_fw_api_qm_a0.pdf
│   │       ├── sc_fw_api_qm_b0.pdf
│   │       ├── sc_fw_api_qx_b0.pdf
│   │       └── sc_fw_port.pdf
│   ├── SCR-imx-scfw-porting-kit.txt
│   └── src
│       ├── scfw_export_atf.tar.gz
│       ├── scfw_export_freertos.tar.gz
│       ├── scfw_export_hyper.tar.gz
│       ├── scfw_export_linux.tar.gz
│       ├── scfw_export_mx8qm_a0.tar.gz
│       ├── scfw_export_mx8qm_b0.tar.gz
│       ├── scfw_export_mx8qx_b0.tar.gz
│       ├── scfw_export_qnx.tar.gz
│       └── scfw_export_uboot.tar.gz
└── imx-scfw-porting-kit-1.0.bin
```

System Controller Firmware Porting Kit – Introduction

- The build directory is where the SCFW binary will be stored after building.
- The platform directory contains all the required source to build.
- Under "platform/board/" is where all board configurations are kept, to create a new board variant simply copy and rename the folder of the board that more closely resembles yours.

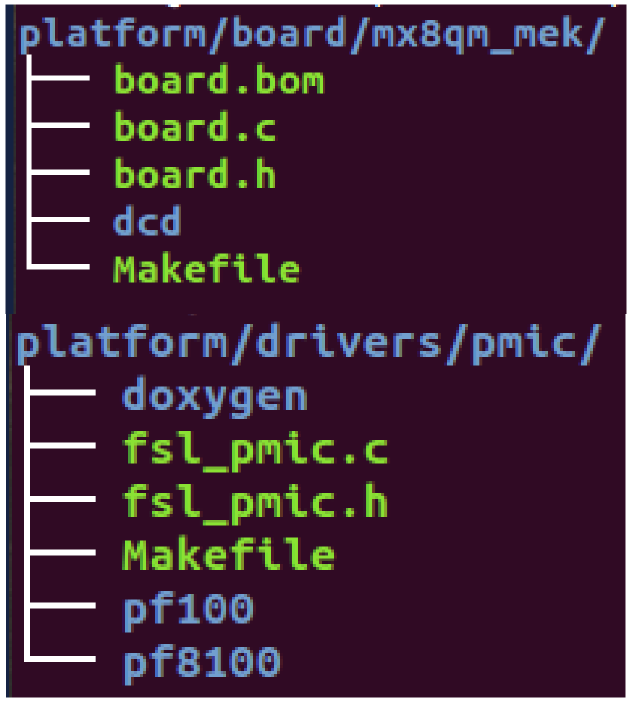


```
.
├── bin
├── build_mx8qm_b0
├── drivers
├── Makefile
├── makefiles
└── platform
```

```
platform/board/
├── board_common.c
├── board_common.h
├── config.h
├── mx8dm_mek
├── mx8dm_val
├── mx8qm_mek
├── mx8qm_val
├── mx8qx_mek
├── mx8qx_val
├── none
├── pmic.c
└── pmic.h
```

System Controller Firmware Porting Kit – Introduction

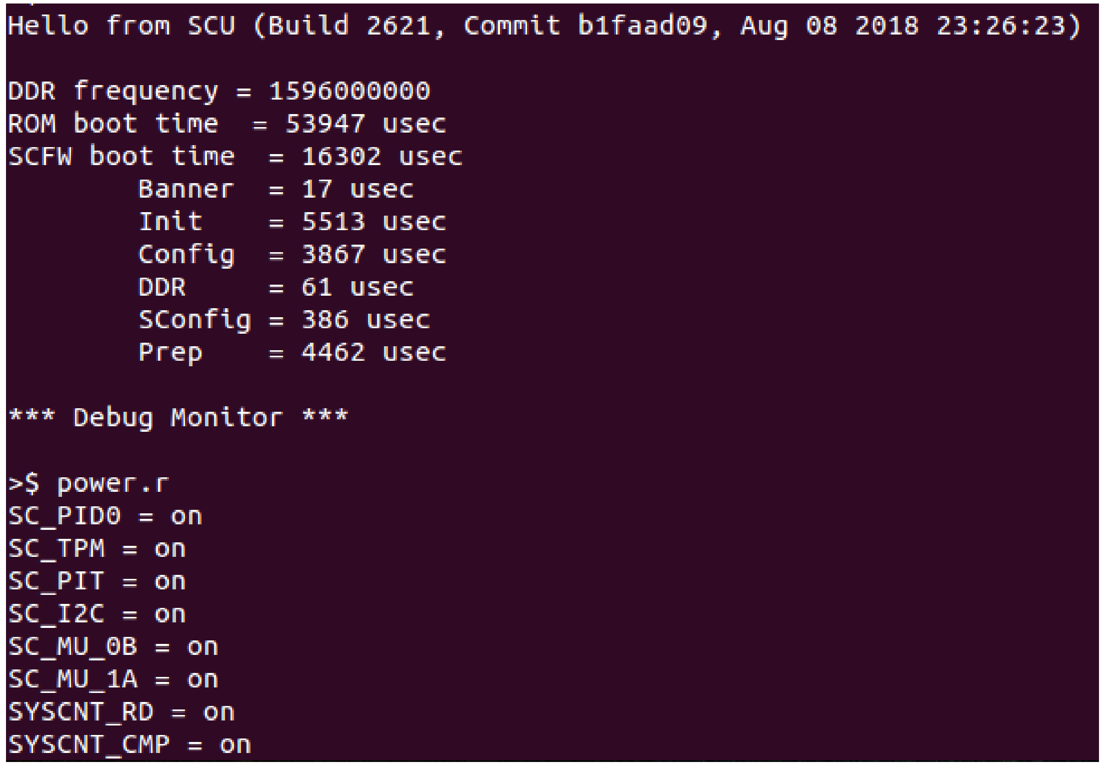
- The board.c file is where the board definition occurs, for details into all the functions that need to be modified please refer to the sc_fw_port.pdf.
  - PMIC SW and LDO connections are defined in here, this information is required by the SCU to power up/down CPUs and GPUs
  - "Board resources" can be mapped with different PMIC/power supplies in here to allow power up/down of them through the application
  - UART configuration for the SCU can be configured in here as well (optional)
- Device Configuration Data (used to bring up DDR mostly) can also be contained within the SCFW
- Both PF100 and PF8100 PMICs are supported and the source of the drivers is provided under "platform/drivers/pmic/pfX"



```
platform/board/mx8qm_mek/
├── board.bom
├── board.c
├── board.h
├── dcd
└── Makefile
platform/drivers/pmic/
├── doxygen
├── fsl_pmic.c
├── fsl_pmic.h
├── Makefile
├── pf100
└── pf8100
```

System Controller Firmware Porting Kit – Introduction

- The SCFW features a "Debug Monitor" option that allows you to perform the following from the SCU UART terminal:
  - Read and write fuses
  - Obtain device information (unique ID, SECO information and SCFW information)
  - Read and write PMIC registers
  - Read resources power modes



```
Hello from SCU (Build 2621, Commit b1faad09, Aug 08 2018 23:26:23)

DDR frequency = 1596000000
ROM boot time  = 53947 usec
SCFW boot time  = 16302 usec
        Banner  = 17 usec
        Init    = 5513 usec
        Config  = 3867 usec
        DDR     = 61 usec
        SConfig = 386 usec
        Prep    = 4462 usec

*** Debug Monitor ***

>$ power.r
SC_PID0 = on
SC_TPM = on
SC_PIT = on
SC_I2C = on
SC_MU_0B = on
SC_MU_1A = on
SYSCNT_RD = on
SYSCNT_CMP = on
```

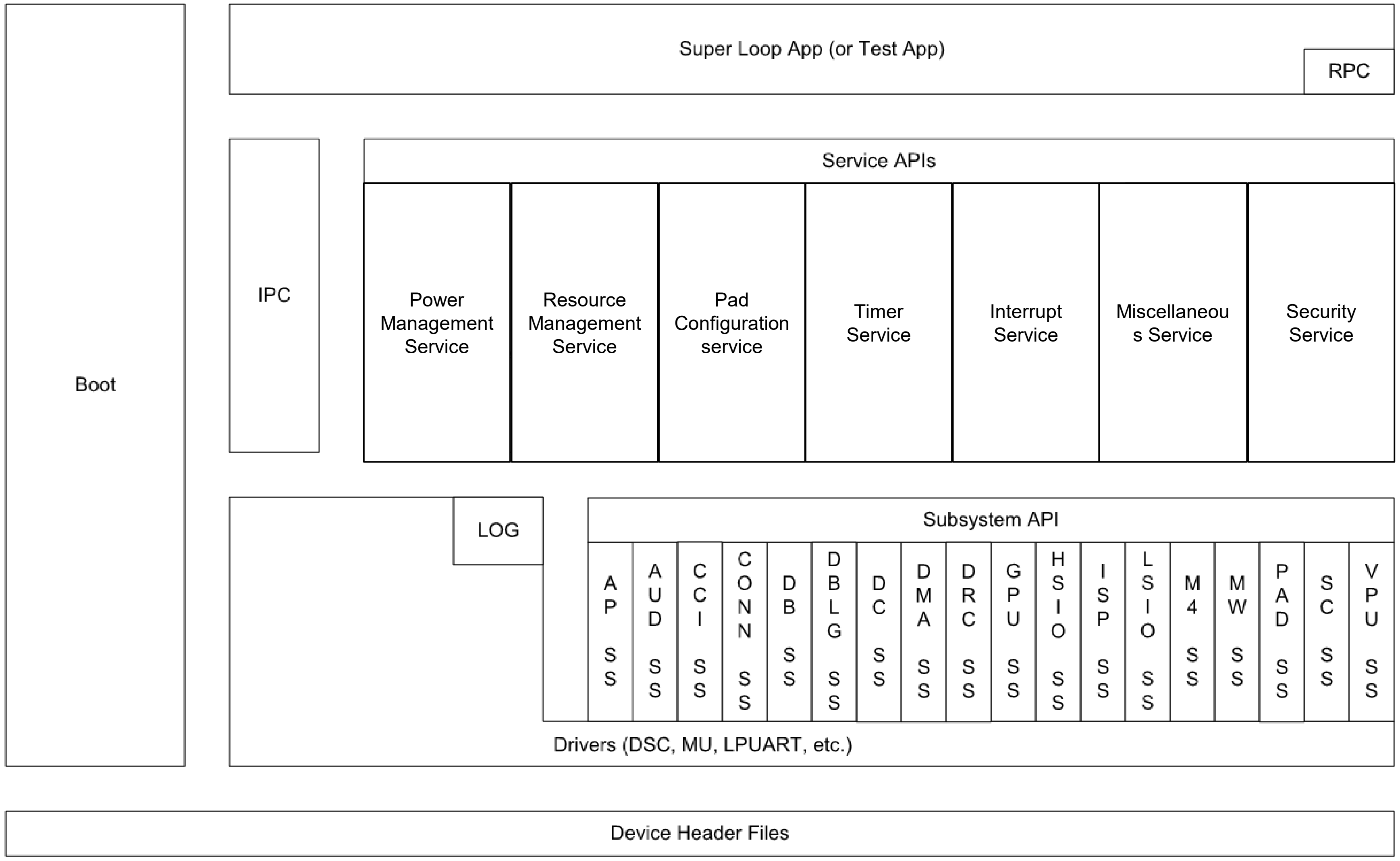
System Controller Firmware Services

System Controller Services

- The functionality offered by the System Controller Firmware is divided into “services” or sets of API calls

- The SCFW provides access to the following services:
  - Power Management Service
  - Resource Management Service
  - Pad Configuration Service
  - Timer Service
  - Interrupt Service
  - Miscellaneous Service

System Controller Firmware Architecture



Power Management Service

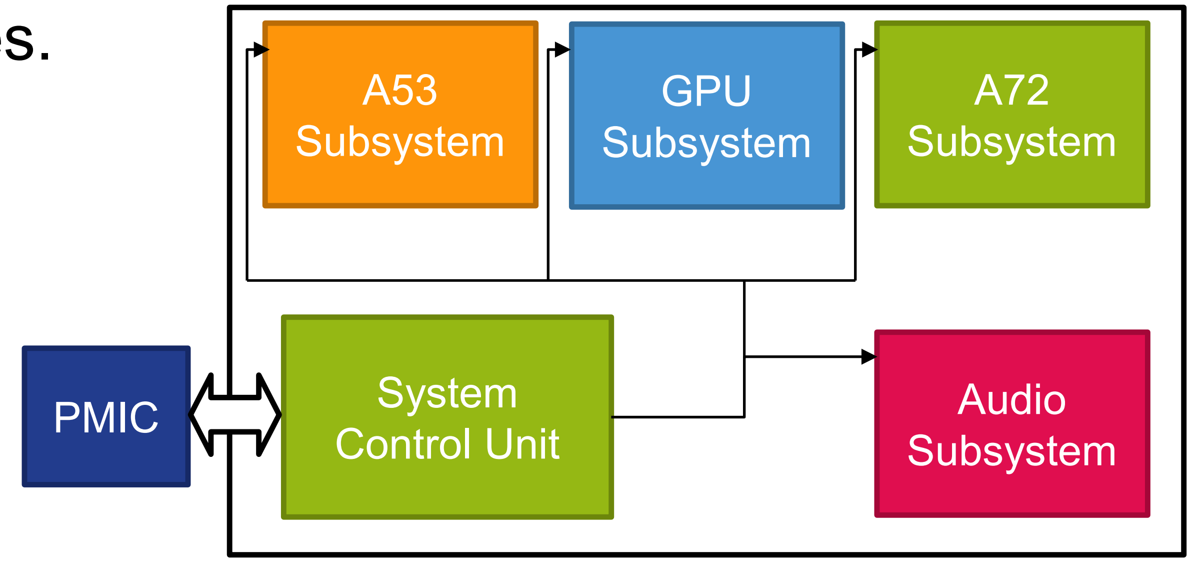
Power Management Service

- The SCFW is responsible for centralized management of power controls both within the device and with external power management devices

- The power management service is in charge of:
  - Power control
  - Clock control
  - Reset control
  - Wake-up event monitoring

Power Management Introduction

- The device is subdivided in subsystems
- Subsystems are groups of resources.
- These subsystems share power domains and clocks
- Power control in the i.MX8 is managed in a distributed manner
- SCU interfaces with each of the subsystems to control the power state of its resources
- SCU has a dedicated interface to the Power Management Integrated Circuit (PMIC)



Power Management Introduction

- Resources can be in either of four power modes:

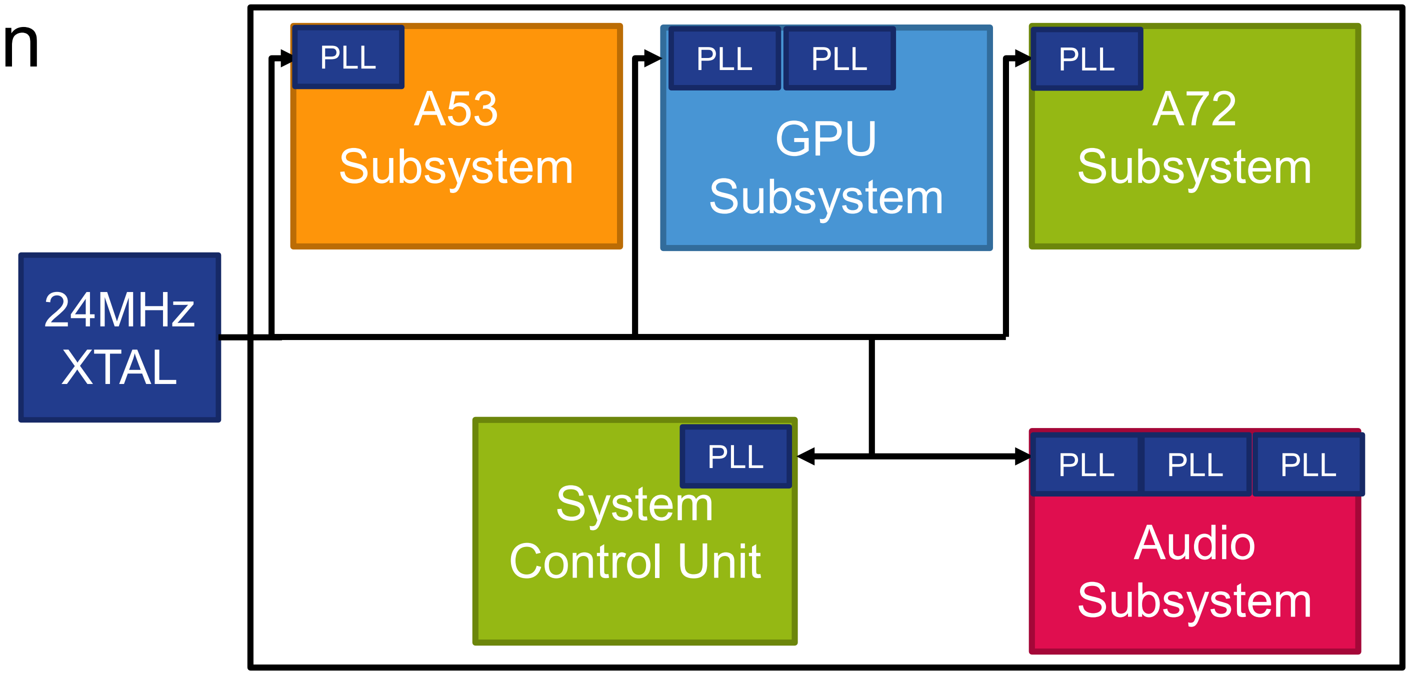
Power Mode	Voltage	Clock Configuration
SC_PM_PW_MODE_OFF	OFF	All clocks Off
SC_PM_PW_MODE_STBY	ON	All clocks Off
SC_PM_PW_MODE_LP	ON	PLLs Off module running from XTAL
SC_PM_PW_MODE_ON	ON	PLLs On

Power Management – API Usage

- The power mode of all resources can be set/get by calling the SCFW API.
- Before accessing a resource it has to be turned on, attempts to access a powered off resource will result in a bus error.
- To set/get a power mode call the following function:
  - sc_pm_set_resource_power_mode (sc_ipc_t ipc, sc_rsrc_t resource, sc_pm_power_mode_t mode)
  - sc_pm_get_resource_power_mode (sc_ipc_t ipc, sc_rsrc_t resource, sc_pm_power_mode_t *mode)
  - Where:
    - ipc – Interprocessor Communication Channel (it can be obtained by calling sc_ipc_open)
    - resource – Resource to set/get the power mode to
    - mode – Power mode to be configured

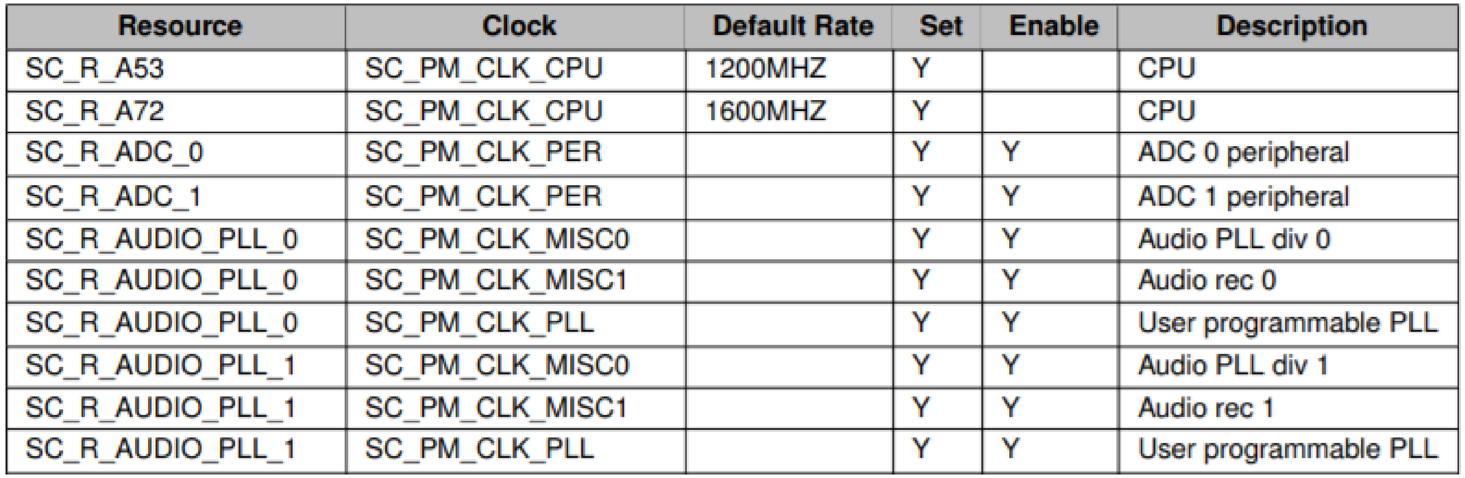
Clock Control Introduction

- The clock control features supported are:
  - Set/Get Clock rate
  - Enable/Disable Clock
  - Set/Get Clock parent (i.e. source, PLL, XTAL…)
- Each subsystem has it’s own PLLs and can have up to three PLLs
- Clocks on all subsystems are derived from the same 24MHz XTAL source



Clock Control – API Usage

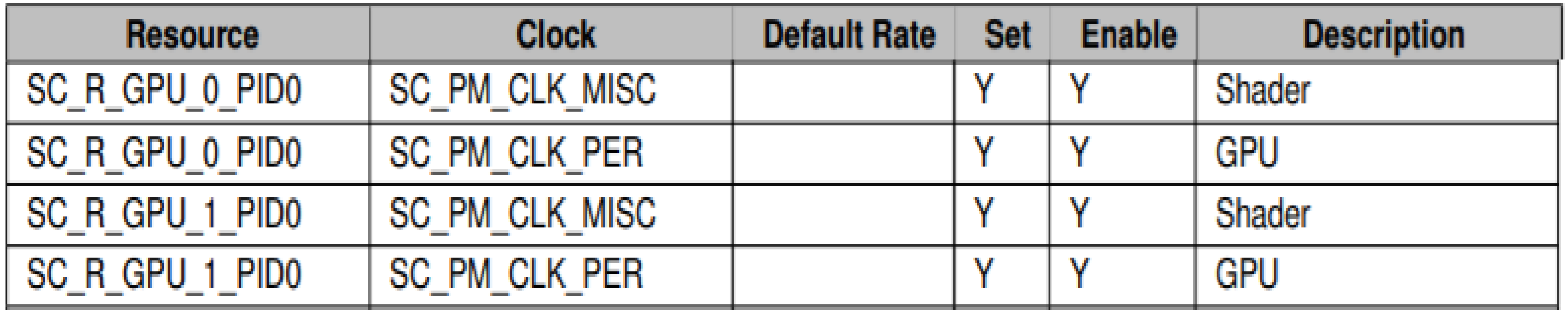
- To set/get a clock rate simply call:
  - sc_pm_set_clock_rate (sc_ipc_t ipc, sc_rsrc_t resource, sc_pm_clk_t clk, sc_pm_clock_rate_t *rate)
  - sc_pm_get_clock_rate (sc_ipc_t ipc, sc_rsrc_t resource, sc_pm_clk_t clk, sc_pm_clock_rate_t *rate)
  - Where:
    - ipc – Interprocessor Communication Channel (it can be obtained by calling sc_ipc_open)
    - resource – Resource to set/get the power mode to
    - clk – Clock to be configured (resources can have multiple clocks and this is a way of identifying them)
    - rate – Desired clock rate



Resource	Clock	Default Rate	Set	Enable	Description
SC_R_A53	SC_PM_CLK_CPU	1200MHZ	Y		CPU
SC_R_A72	SC_PM_CLK_CPU	1600MHZ	Y		CPU
SC_R_ADC_0	SC_PM_CLK_PER		Y	Y	ADC 0 peripheral
SC_R_ADC_1	SC_PM_CLK_PER		Y	Y	ADC 1 peripheral
SC_R_AUDIO_PLL_0	SC_PM_CLK_MISC0		Y	Y	Audio PLL div 0
SC_R_AUDIO_PLL_0	SC_PM_CLK_MISC1		Y	Y	Audio rec 0
SC_R_AUDIO_PLL_0	SC_PM_CLK_PLL		Y	Y	User programmable PLL
SC_R_AUDIO_PLL_1	SC_PM_CLK_MISC0		Y	Y	Audio PLL div 1
SC_R_AUDIO_PLL_1	SC_PM_CLK_MISC1		Y	Y	Audio rec 1
SC_R_AUDIO_PLL_1	SC_PM_CLK_PLL		Y	Y	User programmable PLL

Clock Control – API Usage Example

- As an example the rate of the GPU_0 shader clock will be configured.
- The clock type for the GPU shader is SC_PM_CLK_MISC.
- The set column indicates if the clock can be configured by using SCFW API calls. (The only clock that is not settable is the SCU one)
- The “Enable” column indicates if the clock needs to be enabled by calling sc_pm_clock_enable before usage or if it is auto enabled.
- All information required can be found at the SC_FW_API.pdf Chapter 5 Clock List.
- The following snippet configures the GPU_0 shader clock:
  - sc_clock_rate_t shader_clk=700000000; // 700 MHz
  - sc_pm_set_clock_rate(ipc, SC_R_GPU_0_PID0, SC_PM_CLK_MISC, &shader_clk);



Resource	Clock	Default Rate	Set	Enable	Description
SC_R_GPU_0_PID0	SC_PM_CLK_MISC		Y	Y	Shader
SC_R_GPU_0_PID0	SC_PM_CLK_PER		Y	Y	GPU
SC_R_GPU_1_PID0	SC_PM_CLK_MISC		Y	Y	Shader
SC_R_GPU_1_PID0	SC_PM_CLK_PER		Y	Y	GPU

Reset Control

- The SCFW is responsible for reset control. The available features are:
  - Power up/down of subsystems
  - Manage resets triggered by watchdog timeout events.
  - Request subsystem resets from software
  - Obtaining previous reset source
  - Starting/stopping CPUs

Wake-up Event Monitoring

- The SCU is located in an always-on power domain and is responsible for wakeup event monitoring.
- Wakeup events arrive at the SCU in one of the following ways:
  - Dedicated wakeup inputs (e.g. pad with edge detection)
  - IRQs from subsystems connected to the respective DSC module
- For wake events that require service from other subsystems, the SCU will apply clocks and power to the applicable subsystems.
- The wake event monitoring supports the following:
  - Registration of clock/power/reset sequence to be followed for wake events
  - Deregistration of clock/power/reset sequence for wake events
  - Collaborates with SCU power management firmware to ensure registered wake events can be received

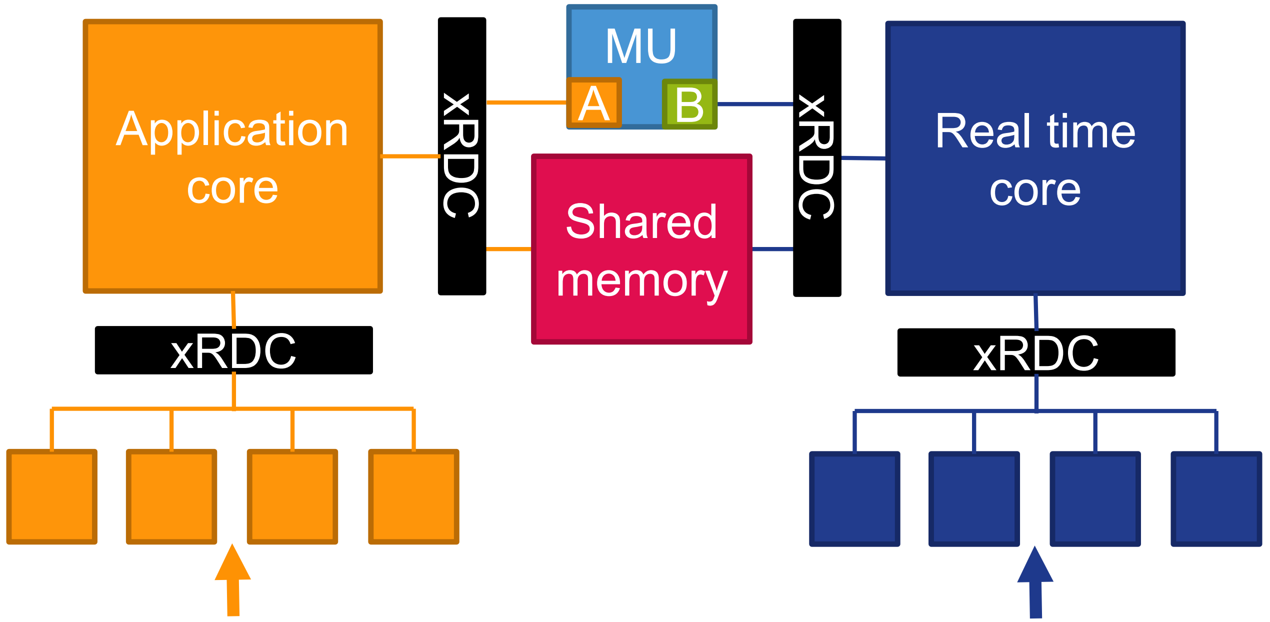
Resource Management Service

Resource Management – Introduction

- SCFW is responsible for managing ownership and access permissions to system resources.
- The resource management functions of the SCU are used to divide up the System on a Chip (SoC) resources.
- The features supported by the SCFW are:
  - Management of system resources such as SoC peripherals, memory regions and pads.
  - Allows resources to be partitioned into different ownership groupings that are associated with different execution environments.
  - Allows owners to configure access permissions to resources.
  - **Provide hardware enforced isolation.**

Partioning on the i.MX8

- The i.MX8 contains a module called the eXtended Resource Domain Controller (xRDC) to enforce hardware portioning on the system.
- The xRDC is configured through the SCFW API.
- The resource management in the i.MX8 allows the creation of domains/clusters/partitions of resources.
- All of these are considered resources:
  - SoC Peripherals
  - Memory regions
  - Pads
- If a peripheral tries to access a resource outside its domain the transaction is blocked by the xRDC and a bus error is generated.

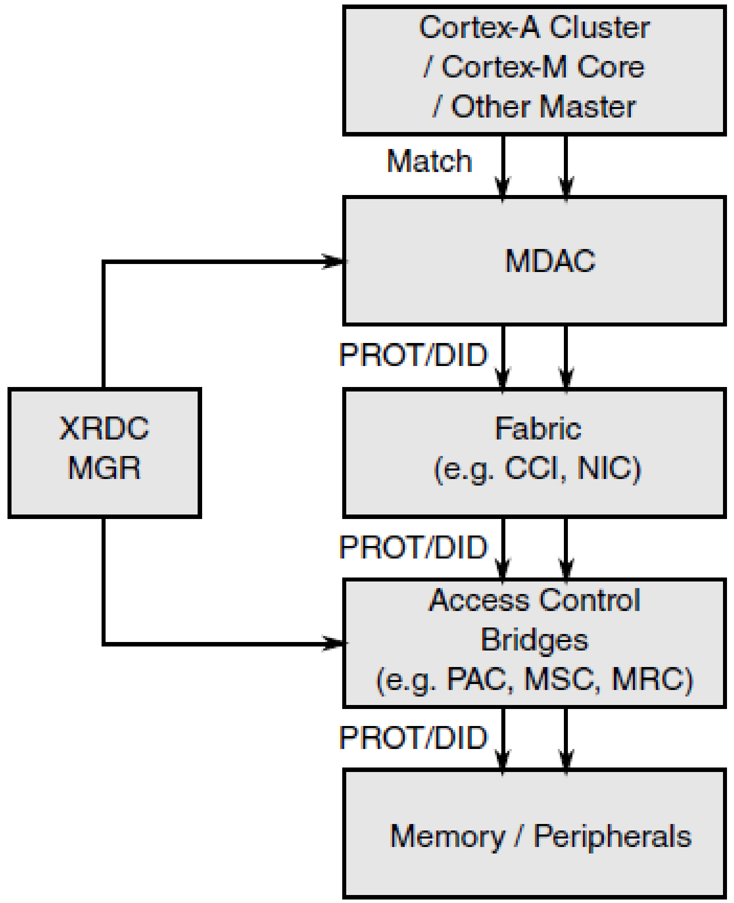


Dedicated app peripherals, memory and pads

Dedicated real time peripherals, memory and pads

xRDC Interaction on Bus Transactions

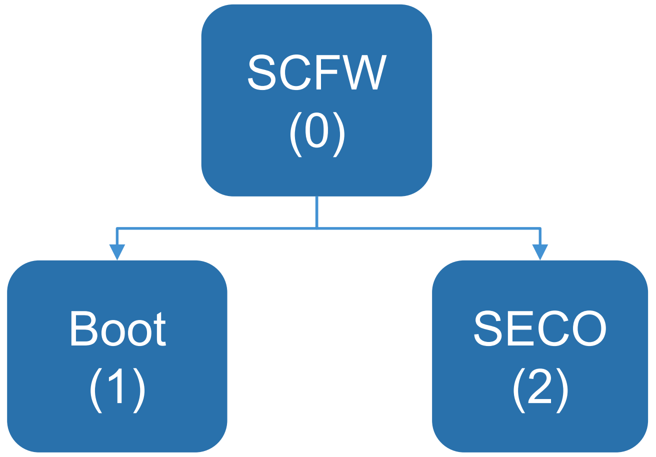
- xRDC MGR (Manager)
  - Provides the programming interface for the entire xRDC.
  - It dispatches incoming accesses to the different xRDC blocks implemented in a subsystem
- MDAC (Master Domain Assignment Controller)
  - It adds Domain ID, Stream ID, Secure/Non-Secure information to out-going transactions of a subsystem
- PAC (Peripheral Access Controller)
  - Control access to peripherals of a subsystem
  - Each peripheral can have different permissions
- MSC (Memory Slot Controller)
  - Used in subsystem when it's protected as a whole (GPU for instance)
- MRC (Memory Region Controller)
  - It controls accesses to memory regions defined by a start address and an end address
  - It can handle up to 32 regions


Cortex-A Cluster / Cortex-M Core / Other Master
Match
MDAC
PROT/DID
XRDC MGR
Fabric (e.g. CCI, NIC)
PROT/DID
Access Control Bridges (e.g. PAC, MSC, MRC)
PROT/DID
Memory / Peripherals

Resource Management – Introduction

- SCFW divides HW into resource partitions (i.e. logical machines). Max of 32.
- HW that is divided includes resources (IP blocks), memory regions, and pads.
- Partitioning affects both SCFW API permissions as well as HW access permissions
- Definition mechanism
  - Based on an assignment method, not an allocation method
  - Owner of resources creates a new resource partition and assigns resources, i.e. users create and define new “logical machines”
  - Similar in concept to virtualization and VMs

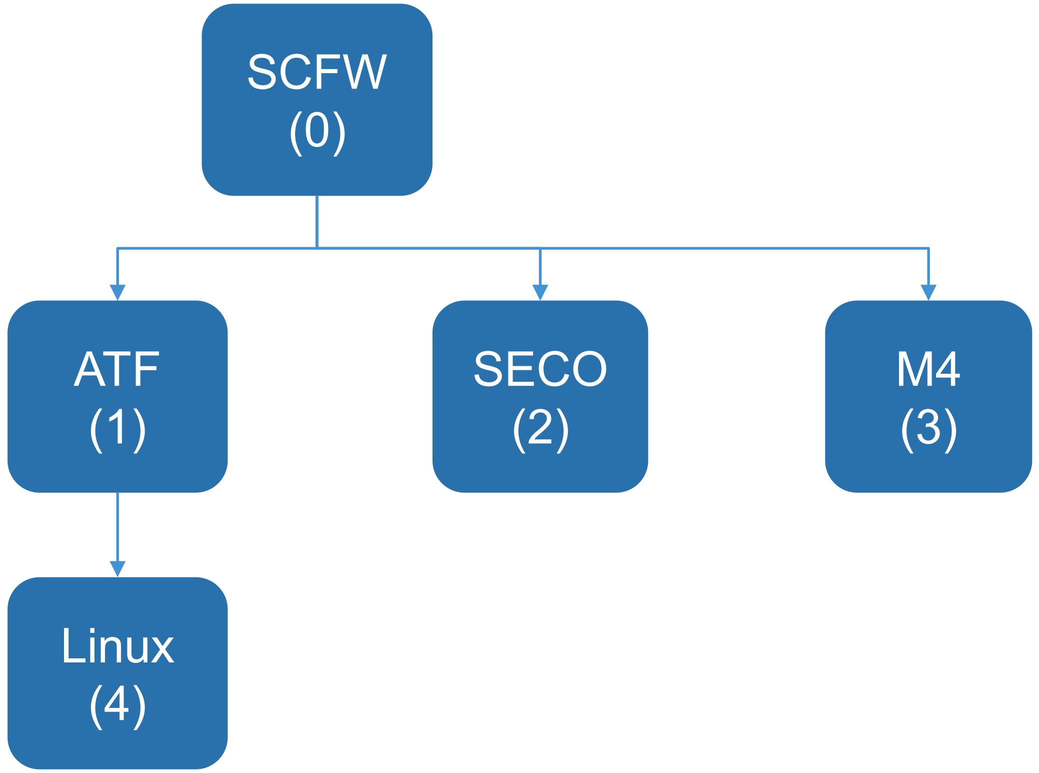
Resource Management – Introduction



- Initial RM partition state:
  - Partition 0: SCFW
  - Partition 1: Boot partition
  - Partition 2: SECO
  - Partitions have a parent/child relationship; determines who can delete, restart, etc.
  - Point(s) of additional configuration depends on system needs
    - Usually, partitioning for SW that is loaded/booted as part of the boot process is done in the SCFW itself via `board_system_config()` in board.c (customer porting layer)
    - Partitioning of SW that is launched by OSes (e.g. an M4 used as sensor fusion and loaded/started by Linux) is done in the OS itself; ATF creates a partition for Linux and assigns all resource TZ does not need

Resource Management – Typical Use Case

- Initial RM partition state:
  - Partition 0: SCFW
  - Partition 1: Boot partition (becomes ATF)
  - Partition 2: SECO
  - Partition 3: M4
  - Partition 4: Linux

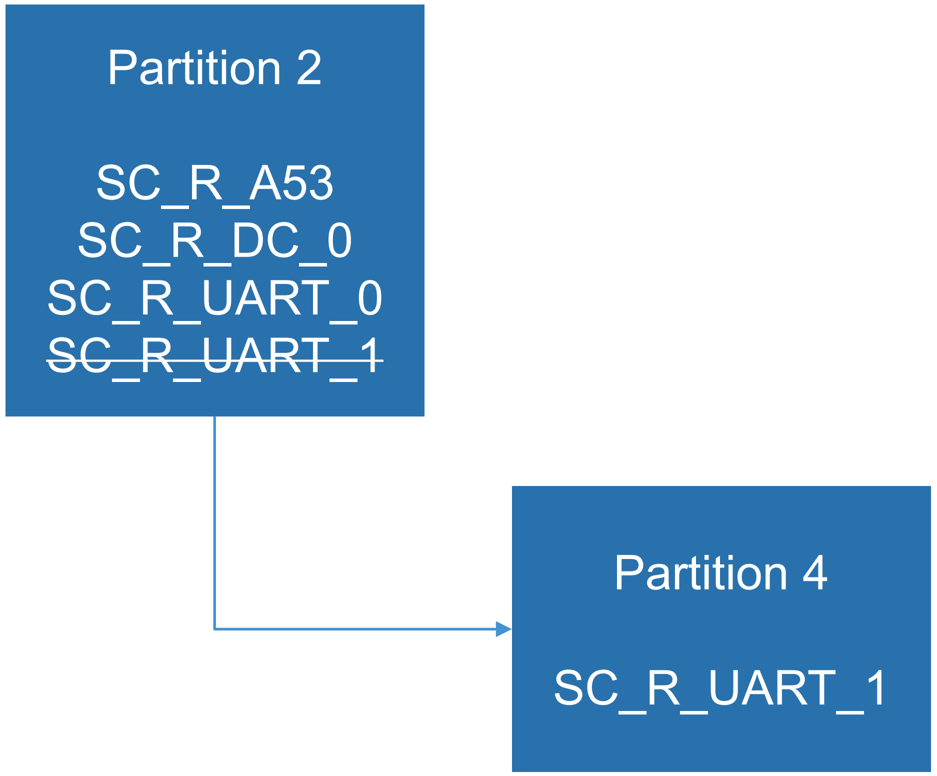


Resource Management – Resources/Pads

- Resources and pads always have an owner (partition)
  - Owner can assign/move to another partition
  - Owner can configure resource/pad attributes
  - Owner can power up/down, control clocks, etc. for resources

- Resource Master Attributes
  - Security
  - Privilege
  - SMMU bypass
  - Stream ID

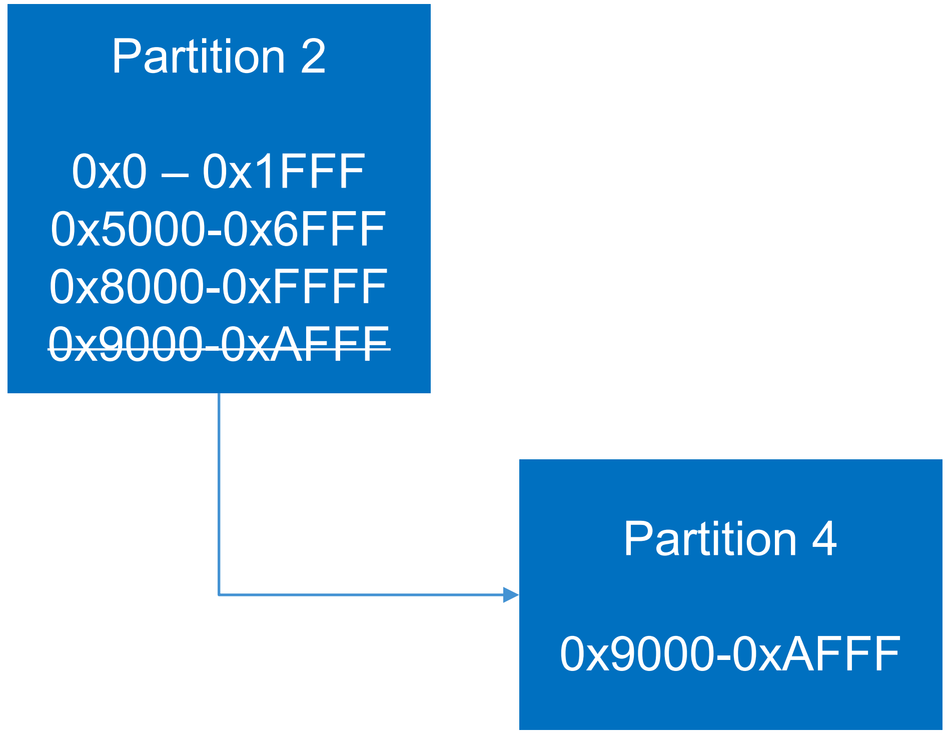
- Resource Peripheral Attributes
  - Access permissions



Resource Management – Memory Regions

- Memory Regions always have an owner (partition)
  - Owner can assign/move to another partition
  - Owner can create a new memory region within the bounds of an existing owned region
  - Owner can split a region
  - Owner can configure access permissions

- Limited to 16 total regions!



Resource Management – Example

```
sc_rm_pt_t pt_m4;
sc_rm_mr_t mr_m4;
/* Create M4 partition*/
/* ipc - Interprocess Communication Channel
   pt_m4 - NEW_Partition
   secure - Indicates if this partition should be secure, only valid if caller is secure
   isolated - Indicates if partition should be HW isolated
   restricted - Indicates if partition should be restricted,
               i.e. masters in this partition cannot create new partitions
   grant - Indicates if partition should always grant access and control to the parent
   coherent - Indicates if partition is coherent, set to true only if this partition will
              contain both AP clusters and they will be coherent via CCI
*/
sc_rm_partition_alloc(ipc, &pt_m4, SC_FALSE, SC_TRUE, SC_TRUE, SC_FALSE, SC_FALSE);
```

Resource Management – Example

```
sc_rm_pt_t pt_m4;
sc_rm_mr_t mr_m4;
/* Create M4 partition*/
/* IPC, NEW_Partition, secure_pt, isolated_pt, restricted_pt, grant_pt, coherent_pt*/
sc_rm_partition_alloc(ipc, &pt_m4, SC_FALSE, SC_TRUE, SC_TRUE, SC_FALSE, SC_FALSE);

/* Assign some resources */
sc_rm_assign_resource(ipc, pt_m4, SC_R_M4_0_PID0);
sc_rm_assign_resource(ipc, pt_m4, SC_R_UART_1);
sc_rm_assign_resource(ipc, pt_m4, SC_R_M4_0_MU_1A);
sc_rm_assign_resource(ipc, pt_m4, SC_R_DMA_0_CH14);
sc_rm_assign_resource(ipc, pt_m4, SC_R_DMA_0_CH15);

/* Allocate and assign memory */
sc_rm_memreg_alloc(ipc, &mr_m4, 0xD0000000, 0xDFFFFFFF);
sc_rm_assign_memreg(ipc, pt_m4, mr_m4);
```

Pad Configuration Service

Pad Configuration Service – Introduction

- All the functionality needed to configure a pad is supported by the SCFW:
  - Common features:
    - Mux selection
    - Mode of operation
    - Low-power isolation mode
    - Pull select
  - Technology specific features:
    - Drive strength
    - Compensation configuration for pad groups with dual voltage capability

Pad Configuration Service – Introduction

- There are 3 main types of CMOS I/Os
  - 1.8V only I/Os
  - 3.3V only I/Os
  - 1.8V / 3.3V I/Os (Dual Voltage) → biggest part of QM IOs

- USB High Speed Inter-Chip (HSIC) and Ethernet (ENET) interfaces have specific integration scheme with dedicated features
  - HSIC I/Os are specific to sustain 480Mbps data rate
  - ENET I/Os have the capability to support 2.5V operations

Pad Configuration Service – I/Os Common Features

- Muxing capability of up to 4 signals
- Mode of operation
  - It defines how the I/O will behave, e.g. open drain
- Low power configuration
- Wake-up capability
  - Defines whether or not the I/O can be used to wake-up the system
  - Defines on which event system should be woken-up (falling edge, rising edge, …)

Pad Configuration Service – Modes of Operation

- There are 4 modes of operation
  - Normal mode
    - While output is enabled, input is disabled and vice-versa
  - Open drain
    - IO switches between high-Z state and drive low
  - Open drain and input
    - Same as open drain with input enabled unconditionally
  - Output and input
    - Same as normal mode with input enabled unconditionally

Pad Configuration Service – Low Power Configuration

- This is a mechanism that is used to ensure that no pads are modified during low-power state and to isolate the input/output signals
- It is a latch that retains the value of the pad while entering low-power mode
- There are 4 different configurations
  - ISO_OFF latch is transparent
    - Latch is off
  - ISO_EARLY latch is driven by EARLY_ISO signal
    - The value in the pad is latched when EARLY_ISO signal is asserted
  - ISO_LATE latch is driven by LATE_ISO
    - The value in the pad is latched when LATE_ISO signal is asserted
  - ISO_ON latch latches the data
    - The value in the pad is latched when this configuration is selected

Pad Configuration Service – Wake-up Configuration

- Each IO has following wake-up capabilities
  - OFF
    - IO cannot wake-up the system
  - Low detect
    - Generate wake-up event when the pad remains in low level for a specific time amount
  - High detect
    - Generate wake-up event when the pad remains in high level for a specific time amount
  - Rising edge detect
    - Generate wake-up event on rising edge detection
  - Falling edge detect
    - Generate wake-up event on falling edge detection

- Wake-up event detection is completely asynchronous
  - It is based on delay elements
  - The structure allows filtering out glitches

Pad Configuration Service – Pull Select

- The pull select available options are the same for all I/O types.

Pull select options
Bus-keeper (only available for 1.8V)
Pull-up
Pull-down
No Pull (Disabled)

Pad Configuration Service – Technology Specific Features

- Drive strength options vary within I/O types
- The available options are:

1.8V Drive strength options	3.3V Drive strength options	Dual Voltage drive strength options
Drive strength of 1mA	Drive strength of 2mA	Low drive strength
Drive strength of 2mA	Drive strength of 4mA	High drive strength
Drive strength of 4mA	Drive strength of 8mA	
Drive strength of 6mA	Drive strength of 12mA	
Drive strength of 8mA		
Drive strength of 10mA		
Drive strength of 12mA		
High-speed drive strength		

Pad Configuration Service – Technology Specific Features

- The compensation feature is only available on Dual Voltage I/Os
- Dual Voltage I/Os have a different implementation, they require:
  - Voltage reference generator – provides voltage references to supply detector and compensation cell
  - Supply detector – detects whether the I/O is being supplied with 1.8V or 3.3V
  - Compensation cell – adjusts drive strength of dual voltage I/Os depending on Process Voltage and Temperature (PVT) conditions
- The SCFW features functions to configure the compensation functionality on dual voltage I/Os

Pad Configuration Service – Example

- Configure the mux alternative, the mode of operation and the low-power isolation

```
sc_err_t sci_err;

/* Configure UART0 pads */
sci_err = sc_pad_set_mux(ipc, SC_P_UART0_RX, IMX_PAD_ALT_0, SC_PAD_CONFIG_NORMAL, SC_PAD_ISO_OFF);
if (sci_err != SC_ERR_NONE) {
        SCI_ERROR_OCCURRED();
}
```

- Then we configure the technology specific features, Drive Strength and Pull Select

```
sci_err = sc_pad_set_gp_28fdsoi(ipc, SC_P_UART0_RX, SC_PAD_28FDSOI_DSE_DV_LOW, SC_PAD_28FDSOI_PS_PD);
if (sci_err != SC_ERR_NONE) {
    SCI_ERROR_OCCURRED();
}
```

Timer Service

Timer Service – Introduction

- The System Controller Firmware supports the following timer functions via the timer service.
  - Watchdog
  - RTC

Timer Service – Watchdog

- As it is not practical to have physical watchdog timers for all execution environments, the SCFW uses a physical timer (Low Power Timer) to expose a "virtual" watchdog timer for all resource partitions.
- The SC manages a set of software watchdog timeout events and refresh requests.
- The watchdog features supported by SC firmware include:
  - Update watchdog timeout
  - Start/stop watchdog
  - Refresh watchdog
  - Return watchdog status such as maximum watchdog timeout that can be set, watchdog timeout interval, and watchdog timeout interval remaining

Note: the SC subsystem includes a physical watchdog timer that is used to insure the correct operation of the SC firmware. It isn't used to implement the timer service's watchdog function.

Timer Service – RTC

- The RTC API supports
  - Setting the time
  - Getting the time
  - Setting alarms.

Note: Only the SW partition that owns the SC_R_SYSTEM resource is allowed to set the time. Time is maintained by the Secure Non-Volatile Storage (SNVS) hardware. All partitions have access to the same time.

Watchdog Functions

- **sc_err_t sc_timer_set_wdog_timeout (sc_ipc_t ipc, sc_timer_wdog_time_t timeout)**
  - sets the watchdog timeout in milliseconds.
- **sc_err_t sc_timer_set_wdog_pre_timeout (sc_ipc_t ipc, sc_timer_wdog_time_t pre_timeout)**
  - sets the watchdog pre-timeout in milliseconds.
- **sc_err_t sc_timer_start_wdog (sc_ipc_t ipc, bool lock)**
  - starts the watchdog.
- **sc_err_t sc_timer_stop_wdog (sc_ipc_t ipc)**
  - stops the watchdog if it is not locked.
- **sc_err_t sc_timer_ping_wdog (sc_ipc_t ipc)**
  - pings (services, kicks) the watchdog resetting the time before expiration back to the timeout.
- **sc_err_t sc_timer_get_wdog_status (sc_ipc_t ipc, sc_timer_wdog_time_t timeout, sc_timer_wdog_time_t max_timeout, sc_timer_wdog_time_t remaining_time)**
  - gets the status of the watchdog.
- **sc_err_t sc_timer_pt_get_wdog_status (sc_ipc_t ipc, sc_rm_pt_t pt, bool enb, sc_timer_wdog_time_t *timeout, sc_timer_wdog_time_t *remaining_time)**
  - gets the status of the watchdog of a partition.
- **sc_err_t sc_timer_set_wdog_action (sc_ipc_t ipc, sc_rm_pt_t pt, sc_timer_wdog_action_t action)**
  - configures the action to be taken when a watchdog expires.

Real-Time Clock (RTC) Functions

- **sc_err_t sc_timer_set_rtc_time (sc_ipc_t ipc, uint16_t year, uint8_t mon, uint8_t day, uint8_t hour, uint8_t min, uint8_t sec)**
  - sets the RTC time.
- **sc_err_t sc_timer_get_rtc_time (sc_ipc_t ipc, uint16_t *year, uint8_t mon, uint8_t *day, uint8_t *hour, uint8_t *min, uint8_t *sec)**
  - gets the RTC time.
- **sc_err_t sc_timer_get_rtc_sec1970 (sc_ipc_t ipc, uint32_t *sec)**
  - gets the RTC time in seconds since 1/1/1970.
- **sc_err_t sc_timer_set_rtc_alarm (sc_ipc_t ipc, uint16_t year, uint8_t mon, uint8_t day, uint8_t hour, uint8_t min, uint8_t sec)**
  - sets the RTC alarm.
- **sc_err_t sc_timer_set_rtc_calb (sc_ipc_t ipc, int8_t count)**
  - sets the RTC calibration value.

Interrupt Service

Interrupt Service

- Interrupt Service provides the method for SC to inform users about **asynchronous notification events**.
- IRQ API export to user the following functionalities:
  - Enable/Disable Interrupts
  - Read Status of pending interrupts
    - **Note: Reading status of pending interrupts automatically clears any pending state**

- sc_err_t sc_irq_enable (sc_ipc_t ipc, sc_rsrc_t resource, sc_irq_group_t group, uint32_t mask, bool enable)
  - enables/disables interrupts.
- sc_err_t sc_irq_status (sc_ipc_t ipc, sc_rsrc_t resource, sc_irq_group_t group, uint32_t *status)
  - returns the current interrupt status (regardless if masked).

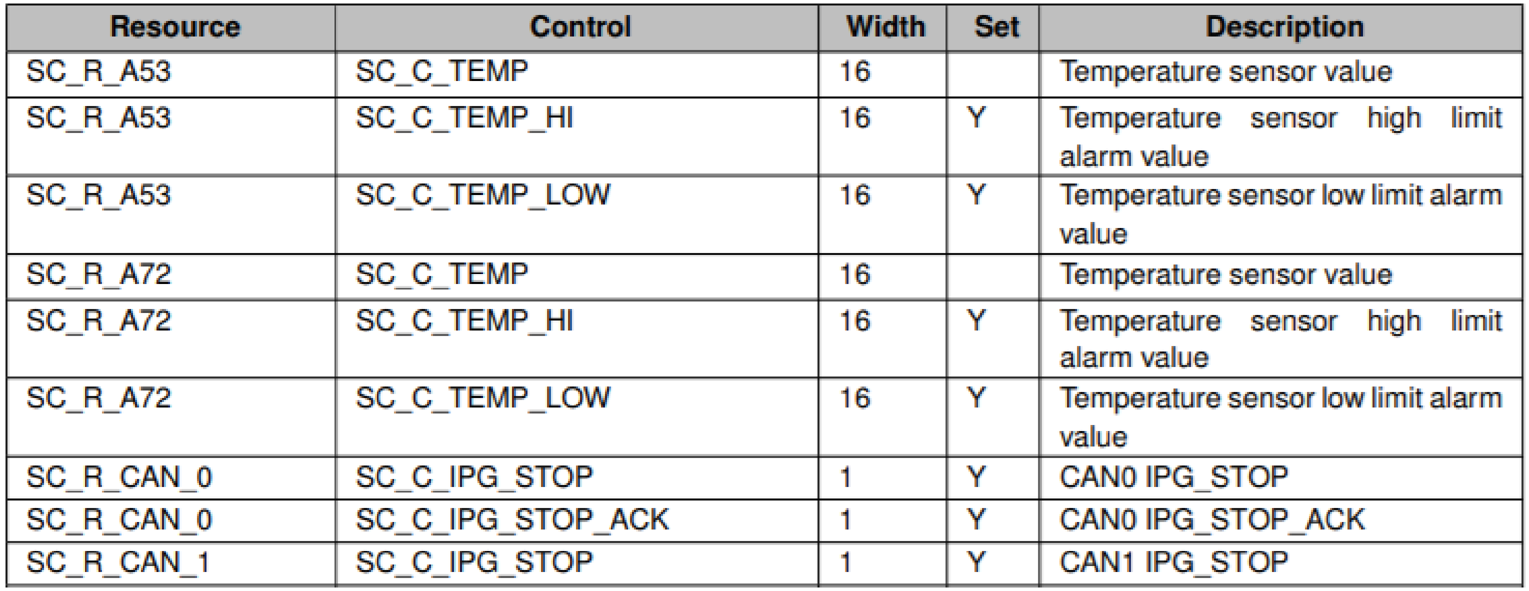
Miscellaneous Service

Miscellaneous Service – Introduction

- The miscellaneous service is in charge of providing access to all features not handled by the other services some examples include the following features:
- Subsystems Controls
  - Some subsystems have settings that can be configured through the SCFW. For instance, it is possible to set thresholds for a temperature alarm and get the temperature value of the sensor in different resources.
    - The SCFW provides functions to set/get these controls.
- DMA
  - The SCFW provides access to DMA grouping and priority functions.
- Debug Features
  - The SCFW provides some debug functionality through its miscellaneous service, some examples include:
    - Output a character through the SCU UART port
    - SCFW build information (SCFW version)
    - Device Unique ID

Miscellaneous Service – Controls Examples

- For a complete list of all available controls refer to the Control List in the sc_fw_api document for your SoC.
- The list showcases all available controls and their respective resources, the set column indicates if the control can be written/set.



Resource	Control	Width	Set	Description
SC_R_A53	SC_C_TEMP	16		Temperature sensor value
SC_R_A53	SC_C_TEMP_HI	16	Y	Temperature sensor high limit alarm value
SC_R_A53	SC_C_TEMP_LOW	16	Y	Temperature sensor low limit alarm value
SC_R_A72	SC_C_TEMP	16		Temperature sensor value
SC_R_A72	SC_C_TEMP_HI	16	Y	Temperature sensor high limit alarm value
SC_R_A72	SC_C_TEMP_LOW	16	Y	Temperature sensor low limit alarm value
SC_R_CAN_0	SC_C_IPG_STOP	1	Y	CAN0 IPG_STOP
SC_R_CAN_0	SC_C_IPG_STOP_ACK	1	Y	CAN0 IPG_STOP_ACK
SC_R_CAN_1	SC_C_IPG_STOP	1	Y	CAN1 IPG_STOP

Miscellaneous Service – Controls Examples

- To set/get a control call:
  - sc_misc_set_control (sc_ipc_t ipc, sc_rsrc_t resource, sc_ctrl_t ctrl, uint32_t val)
  - sc_misc_get_control (sc_ipc_t ipc, sc_rsrc_t resource, sc_ctrl_t ctrl, uint32_t *val)
  - Where:
    - ipc – Interprocessor Communication Channel (it can be obtained by calling sc_ipc_open)
    - resource – Resource to set/get the control
    - val – Value to set the control to

Control Functions

- **sc_err_t sc_misc_set_control (sc_ipc_t ipc, sc_rsrc_t resource, sc_ctrl_t ctrl, uint32_t val)**
  - sets a miscellaneous control value.
- **sc_err_t sc_misc_get_control (sc_ipc_t ipc, sc_rsrc_t resource, sc_ctrl_t ctrl, uint32_t *val)**
  - gets a miscellaneous control value.

DMA Functions

- **sc_err_t sc_misc_set_max_dma_group (sc_ipc_t ipc, sc_rm_pt_t pt, sc_misc_dma_group_t max)**
  - configures the max DMA channel priority group for a partition.
- **sc_err_t sc_misc_set_dma_group (sc_ipc_t ipc, sc_rsrc_t resource, sc_misc_dma_group_t group)**
  - configures the priority group for a DMA channel.

Debug Functions

- **void sc_misc_debug_out (sc_ipc_t ipc, uint8_t ch)**
  - used output a debug character from the SCU UART.
- **sc_err_t sc_misc_waveform_capture (sc_ipc_t ipc, bool enable)**
  - starts/stops emulation waveform capture.
- **void sc_misc_build_info (sc_ipc_t ipc, uint32_t *build, uint32_t *commit)**
  - used to return the SCFW build info.

Other Functions

- **sc_err_t sc_misc_set_ari (sc_ipc_t** ipc, **sc_rsrc_t** resource, **sc_rsrc_t** resource_mst, **uint16_t** ari, **bool** enable**)**
  - configures the ARI match value for PCIe/SATA resources.
- **void sc_misc_boot_status (sc_ipc_t** ipc, **sc_misc_boot_status_t** status**)**
  - reports boot status.
- **sc_err_t sc_misc_boot_done (sc_ipc_t** ipc, **sc_rsrc_t** cpu**)**
  - tells the SCFW that a CPU is done booting.
- **sc_err_t sc_misc_otp_fuse_read (sc_ipc_t** ipc, **uint32_t** word, **uint32_t** *val**)**
  - reads a given fuse word index.
- **sc_err_t sc_misc_otp_fuse_write (sc_ipc_t** ipc, **uint32_t** word, **uint32_t** val**)**
  - writes a given fuse word index.
- **sc_err_t sc_misc_set_temp (sc_ipc_t** ipc, **sc_rsrc_t** resource, **sc_misc_temp_t** temp, **int16_t** celsius, **int8_t** tenths**)**
  - sets a temp sensor alarm.
- **sc_err_t sc_misc_get_temp (sc_ipc_t** ipc, **sc_rsrc_t** resource, **sc_misc_temp_t** temp, **int16_t** *celsius, **int8_t** *tenths**)**
  - gets a temp sensor value.
- **void sc_misc_get_boot_dev (sc_ipc_t** ipc, **sc_rsrc_t** *dev**)**
  - returns the boot device.
- **void sc_misc_get_button_status (sc_ipc_t** ipc, **bool** *status**)**
  - returns the current status of the ON/OFF button.

Security Service

Security Service – Introduction

- The SC firmware provides access to many security functions including:
  - Image Authentication
  - Generating, exporting, and loading key blobs
  - Fuse programming
  - Lifecycle management
  - Attestation
- Interactions with the Security Controller (SECO) Firmware are managed through the SCFW API.

Commonly Used Funtions

- sc_err_t sc_seco_forward_lifecycle (sc_ipc_t ipc, uint32_t change)
  - This function updates the lifecycle of the device.
- void sc_seco_build_info (sc_ipc_t ipc, uint32_t ∗version, uint32_t ∗commit)
  - This function is used to return the SECO FW build info.
- sc_err_t sc_seco_chip_info (sc_ipc_t ipc, uint16_t ∗lc, uint16_t ∗monotonic, uint32_t ∗uid_l, uint32_t ∗uid_h)
  - This function is used to return SECO chip info.
- sc_err_t sc_seco_get_event (sc_ipc_t ipc, uint8_t idx, uint32_t ∗event)
  - This function is used to return an event from the SECO error log.

Questions

SECURE CONNECTIONS
FOR A SMARTER WORLD