2 System

2.1 Memory Mapping

Module	Address	Size(Bytes)
BROM & SRAM		
S_BROM	0x0000 0000---0x0000 AFFF	44 KB
M_BROM	0x0001 0000---0x0001 8FFF	36 KB
MCU Local SRAM	0x0002 0000---0x0003 FFFF	128 KB The local SRAM is used for system boot.
SRAM A2	0x0004 0000---0x0006 3FFF	16 KB+128 KB
GPU_SYS		
GPU	0x0180 0000---0x0183 FFFF	256 KB
VE_SYS		
VE	0x01C0 E000---0x01C0 EFFF	4 KB
SP0		
GPIO	0x0200 0000---0x0200 07FF	2 KB
SPC	0x0200 0800---0x0200 0BFF	1 KB
PWM0	0x0200 0C00---0x0200 0FFF	1 KB
CCU	0x0200 1000---0x0200 1FFF	4 KB
CIR_TX	0x0200 3000---0x0200 33FF	1 KB
CIR_RX	0x0200 5000---0x0200 53FF	1 KB
LEDC	0x0200 8000---0x0200 83FF	1 KB
GPADC_CTRL0	0x0200 9000---0x0200 93FF	1 KB
THS1	0x0200 9400---0x0200 97FF	1 KB
LRADC	0x0200 9800---0x0200 9BFF	1 KB
GPADC_CTRL1	0x0200 9C00---0x0200 9FFF	1 KB
THS0	0x0200 A000---0x0200 A3FF	1 KB
IOMMU	0x0201 0000---0x0201 FFFF	64 KB
NSI	0x0202 0000---0x0202 FFFF	64 KB
CPUX_WDT	0x0205 0000---0x0205 0FFF	4 KB
PWM1	0x0205 1000---0x0205 13FF	1 KB
NSI_CPU	0x0207 1000---0x0207 13FF	1 KB
SP1		
UART0	0x0250 0000---0x0250 03FF	1 KB
UART1	0x0250 0400---0x0250 07FF	1 KB
UART2	0x0250 0800---0x0250 0BFF	1 KB
UART3	0x0250 0C00---0x0250 0FFF	1 KB

Module	Address	Size(Bytes)
UART4	0x0250 1000---0x0250 13FF	1 KB
UART5	0x0250 1400---0x0250 17FF	1 KB
UART6	0x0250 1800---0x0250 1BFF	1 KB
UART7	0x0250 1C00---0x0250 1FFF	1 KB
TWI0	0x0250 2000---0x0250 23FF	1 KB
TWI1	0x0250 2400---0x0250 27FF	1 KB
TWI2	0x0250 2800---0x0250 2BFF	1 KB
TWI3	0x0250 2C00---0x0250 2FFF	1 KB
TWI4	0x0250 3000---0x0250 33FF	1 KB
TWI5	0x0250 3400---0x0250 37FF	1 KB
SH0		
SYSCTRL	0x0300 0000---0x0300 0FFF	4 KB
CPUX_TIMER	0x0300 8000---0x0300 83FF	1 KB
DMAC	0x0300 2000---0x0300 2FFF	4 KB
CPUX_MSGBOX	0x0300 3000---0x0300 3FFF	4 KB
SPINLOCK	0x0300 5000---0x0300 5FFF	4 KB
SID	0x0300 6000---0x0300 6FFF	4 KB
CE_NS	0x0304 0000---0x0304 07FF	2 KB
CE_S	0x0304 0800---0x0304 0FFF	2 KB
SECURE_DEBUG_CFG	0x0304 2000---0x0304 23FF	1 KB
MEMC	0x0310 2000---0x0330 1FFF	2 M
MEMC_SMC	0x0311 0000---0x0311 FFFF	64 KB
MEMC_COMMON	0x0312 0000---0x0312 FFFF	64 KB
MEMC_DDRC	0x0313 0000---0x0313 FFFF	64 KB
MEMC_PHY	0x0314 0000---0x0314 FFFF	64 KB
GIC	0x0340 0000---0x034E FFFF	15*64 KB
SH2		
NDFC	0x0401 1000---0x0401 1FFF	4 KB
SMHC0	0x0402 0000---0x0402 0FFF	4 KB
SMHC1	0x0402 1000---0x0402 1FFF	4 KB
SMHC2	0x0402 2000---0x0402 2FFF	4 KB
SPI0	0x0402 5000---0x0402 5FFF	4 KB
SPI1	0x0402 6000---0x0402 6FFF	4 KB
SPI2	0x0402 7000---0x0402 7FFF	4 KB
USB0	0x0410 0000---0x041F FFFF	1 MB
USB1	0x0420 0000---0x042F FFFF	1 MB
GMAC0	0x0450 0000---0x0450 FFFF	64 KB
GMAC1	0x0451 0000---0x0451 FFFF	64K
SPIFC	0x047F 0000---0x047F 0FFF	4 KB
PCIE	0x0480 0000---0x04CF FFFF	5 MB

Module	Address	Size(Bytes)
USB3	0x04D0 0000---0x04EF FFFF	2 MB
PCIE_USB3_TOP_APP	0x04F0 0000---0x04F7 FFFF	512 KB
DE_SYS		
DE	0x0500 0000---0x053F FFFF	4 MB
DI	0x0540 0000---0x0543 FFFF	256 KB
G2D	0x0544 0000---0x0547 FFFF	256 KB
VIDEO0_OUT_SYS		
DISPLAY0_TOP	0x0550 0000---0x0550 0FFF	4 KB
TCON_LCD0	0x0550 1000---0x0550 1FFF	4 KB
TCON_LCD1	0x0550 2000---0x0550 2FFF	4 KB
TCON_TV0	0x0550 3000---0x0550 3FFF	4 KB
TCON_TV1	0x0550 4000---0x0550 4FFF	4 KB
COMBOPHY_DSI0	0x0550 6000---0x0550 7FFF	8 KB
COMBOPHY_DSI1	0x0550 8000---0x0550 9FFF	8 KB
HDMI	0x0552 0000---0x0561 FFFF	1 MB
EDP	0x0572 0000---0x0572 3FFF	16 KB
VIDEO1_OUT_SYS		
DISPLAY1_TOP	0x0573 0000---0x0573 0FFF	4 KB
TCON_LCD2	0x0573 1000---0x0573 1FFF	4 KB
VIDEO_IN_SYS		
CSI	0x0580 0000---0x058F FFFF	1 MB
ISP	0x0590 0000---0x05CF FFFF	4 MB
APBS0		
S_PPU1	0x0700 1400---0x0700 17FF	1 KB
S_SPC	0x0700 2000---0x0700 23FF	1 KB
STBY_PRCM	0x0701 0000---0x0701 FFFF	64 KB
CPUS_WDT	0x0702 0400---0x0702 07FF	1 KB
S_TWD	0x0702 0800---0x0702 0BFF	1 KB
S_PWM0	0x0702 0C00---0x0702 0FFF	1 KB
S_INTC	0x0702 1000---0x0702 13FF	1 KB
S_GPIO	0x0702 2000---0x0702 27FF	2 KB
CPUS_CFG	0x0703 1000---0x0703 1FFF	4 KB
S_CIRRX	0x0704 0000---0x0704 03FF	1 KB
PCK600_CPU	0x0705 0000---0x0705 FFFF	64 KB
PCK600_QCHANNEL(S_PPU)	0x0706 0000---0x0706 7FFF	32 KB
APBS1		
S_UART0	0x0708 0000---0x0708 03FF	1 KB
S_UART1	0x0708 0400---0x0708 07FF	1 KB
S_TWI0	0x0708 1400---0x0708 17FF	1 KB
S_TWI1	0x0708 1800---0x0708 1BFF	1 KB

Module	Address	Size(Bytes)
S_TWI2	0x0708 1C00---0x0708 1FFF	1 KB
AHBS		
RTC	0x0709 0000---0x0709 03FF	1 KB
CPUS_TIMER	0x0709 0400---0x0709 07FF	1 KB
S_SPI0	0x0709 2000---0x0709 2FFF	4 KB
S_SPINLOCK	0x0709 3000---0x0709 3FFF	4 KB
CPUS_MSGBOX	0x0709 4000---0x0709 4FFF	4 KB
MCU_APB0		
MCU_PRCM	0x0710 2000---0x0710 2FFF	4 KB
MCU_PWM0	0x0710 3000---0x0710 33FF	1 KB
AUDIO CODEC	0x0711 0000---0x0711 0FFF	4 KB
DMIC	0x0711 1000---0x0711 13FF	1 KB
I2S0	0x0711 2000---0x0711 2FFF	4 KB
I2S1	0x0711 3000---0x0711 3FFF	4 KB
I2S2	0x0711 4000---0x0711 4FFF	4 KB
I2S3	0x0711 5000---0x0711 5FFF	4 KB
OWA	0x0711 6000---0x0711 63FF	1 KB
MCU_AHB		
MCU_DMAC	0x0712 1000---0x0712 1FFF	4 KB
MCU_TIMER	0x0712 3000---0x0712 33FF	1 KB
RISCV_SYS		
RISCV_CFG	0x0713 0000---0x0713 0FFF	4 KB
RISCV_WDT	0x0713 2000---0x0713 2FFF	4 KB
RISCV_LCNT	0x0713 4000---0x0713 4FFF	4 KB
RISCV_MSGBOX	0x0713 6000---0x0713 6FFF	4 KB
MCU_SRAM		
SRAMA3_0	0x0728 0000---0x072B FFFF	256 KB (SRAMA3_0 cannot be used cross 256 KB boundary)
SRAMA3_1	0x072C 0000---0x072F FFFF	256 KB (SRAMA3_1 cannot be used cross 256 KB boundary)
SRAMA3_2	0x0730 0000---0x0737 FFFF	512 KB
CPUX Related		
CPU_SUBSYS_CTRL	0x0800 0000---0x0800 0FFF	4 KB
TIMESTAMP_STA	0x0801 0000---0x0801 0FFF	4 KB
TIMESTAMP_CTRL	0x0802 0000---0x0802 0FFF	4 KB
CPU_PLL_CFG	0x0881 7000---0x0881 7FFF	4 KB
PCIE		
PCIE_SLV	0x2000 0000---0x2FFF FFFF	256 MB
RISCV Related (Only RISC-V access)		
RISCV_CLINT	0xE000 0000---0xE000 FFFF	64 KB

Module	Address	Size(Bytes)
RISCV_CLIC	0xE080 0000---0xE080 4FFF	20 KB
RISCV_SYSMAP	0xEFFF F000---0xEFFF FFFF	4 KB
DRAM Space		
DRAM SPACE	0x4000 0000---0x13FFF FFFF	4 GB RISC-V core accesses the DRAM address: 0x4004 0000---0x7FFF FFFF

2.2 ARM Cortex™-A55 System(CPUX)

2.2.1 Overview

A527 CPU architecture adopts DynamIQ technology. The CPUX system includes DynamIQ Shared Unit (DSU), DynamIQ cluster, GIC600 distributor, CoreSight subsystem, and timestamp module. The features of the CPUX cores and DSU in DynamIQ cluster are as follows.

CPUX Cores

- Two sets of ARM Cortex™-A55 cores in a DynamIQ big. LITTLE configuration
- Memory subsystem features
  - 32 KB L1 I-cache and D-cache
  - Optional 64KB L2 cache for 'LITTLE' cores
  - Optional 128KB L2 cache for 'big' cores
  - Separate L1 instruction side memory subsystem with a Memory Management Unit (MMU)
- A64, A32, and T32 instruction sets running on ARMv8-A architecture ISA
- Both the AArch32 and AArch64 execution states at all Exception levels (EL0 to EL3).
- In-order pipeline with direct and indirect branch prediction.
- Optional Data Engine Unit implementing the advanced Single Instruction Multiple Data (SIMD) and floating-point architecture
- Optional Cryptography extensions
- Separate L1 instruction side memory system with a Memory Management Unit (MMU)
- ARM TrustZone® technology
- Generic Interrupt Controller (GIC) interface connecting an external distributor
- Generic Timers interface supporting 64-bit count input from an external system counter
- Reliability, Availability, and Serviceability (RAS) extension
- Debug and trace capabilities

NOTE

Cryptography extensions are available only when Data Engine unit is present.

DSU

DSU comprises the L3 cache, the Snoop Control Unit (SCU), internal interfaces to the cores, and external interfaces to the SoC.

- Memory subsystem features
  - 1024 KB L3 cache

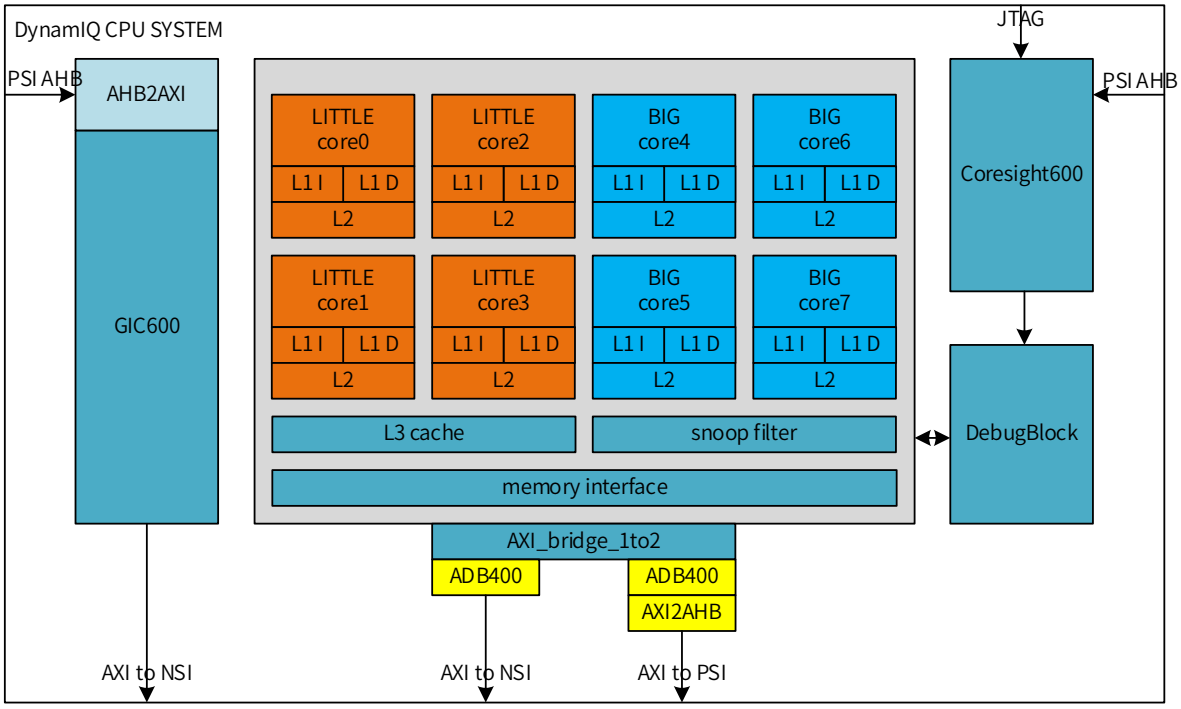
- Optional 16-way set-associative L3 cache, 64-byte cache line
- L3 memory system can be clocked at a rate synchronous to the external system interconnect or at integer multiples.
- L3 cache partial power down
- Optional cache protection in the form of Error Correcting Code (ECC) on L3 cache RAM instances
- 40-bit, 44-bit, and 48-bit physical addresses

- Main bus interface adopting AMBA 5 ACE protocol or AMBA 5 CHI protocol
- Optional 128-bit wide and I/O-coherent Accelerator Coherency Port (ACP)
- Optional 64-bit wide peripheral port
- ARMv8.2 debug logic
- Supports RAS

2.2.2 Block diagram

The following figure shows the block diagram of CPUX system.

Figure 2-1 CPUX System Block Diagram



The following table describes the components of A527 DynamIQ big. LITTLE cluster.

Table 2-1 CPUX DynamIQ Cluster Components

Components	Description
CPU bridges	For communication between cores and DSU buffers.

Components	Description
SCU	The SCU maintains coherency and cache-to-cache transmission for all CPUX cores.
Debug and trace components	Each core allows tracing supported by Embedded Trace Macrocell (ETM). The trigger events from CPUX cores are transmitted through debug APB master/slave interface.
Clock and power management	The cluster supports low power mode and is controlled by a low power control module outside the cluster power-down domain. DSU and each CPUX core has independent P-channels. They could control the power mode through P-channels.
L3 memory interfaces	To access memory and peripherals.
DSU system control registers	Include information related to CPUX core configuration, such as: • Power management of the cluster • QOS and ID control of CHI bus • DSU hardware configuration information • L3 cache hit and miss count information

2.2.3 Functional Descriptions

2.2.3.1 Power Block System

Power Domain

The following table describes the power domain of the CPUX.

Table 2-2 CPUX Power domain

Power Domain	Power Switch	Description
VDD-CPUB	Yes	Power source of Core4-Core7. It is controlled by the PPU for each core.
VDD-CPUL	Yes	Power source of the cluster top and Core0-Core3. It is controlled by the PPU for the cluster top and each core.
VDD-SYS	No	Power source of CPUX system excluding the cluster top and CPUX cores. It is the same power supply of the SoC system.

Power Mode

CPUX cores support four power modes:

- Debug Recovery
- ON
- OFF (emulated)
- OFF

DSU supports the following power modes

- ON: SFONLY_ON、1/4 ON、1/2 ON、3/4 ON、FULL ON
- Functional Retention: SFONLY FUNC_RET, ¼FUNC_RET, ½FUNC_RET, ¾FUNC_RET, FULL FUNC_RET
- OFF and OFF_EMU

2.2.3.2 CPU PLL Distribution and Clock Sources

The CPUX system contains three linear frequency modulation PLLs: CPU_L_PLL, CPU_DSU_PLL, and CPU_B_PLL. The following table shows the clock sources of CPUX cores and DSU.

Table 2-3 Clock Sources of CPUX Cores and DSU

CPUX Cores	Clock Sources	Description
Core0-Core3	CLK32K	• Generally, CPU_L_PLL is the main clock source of Core0-Core3. For all clock sources of Core0-Core3, refer to CPUA_CLK_REG register. • Generally, CPU_B_PLL is the main clock source of Core4-Core7. For all clock sources of Core4-Core7, refer to CPUB_CLK_REG register. • Generally, CPU_DSU_PLL is the main clock source of DSU. For all clock sources of DSU, refer to DSU_CLK_REG register.
	CLK16M_RC	
	HOSC	
	PERI0_600M	
	CPU_L_PLL	
Core4-Core7	CLK32K	
	CLK16M_RC	
	HOSC	
	PERI0_600M	
	CPU_PLL3CPU_B_PLL	
DSU	CLK32K	
	CLK16M_RC	
	HOSC	
	PERI0_600M	
	PLL_PERI0(2X)	
	CPU_DSU_PLL	

2.2.3.3 CPUX Reset System

The following table shows the input reset signal of the whole CPUX system.

Table 2-4 Reset signal description of CPUX System

Reset signal	Source	Description
DBGSYS_RST	CCU	For detailed information, please refer to the description of DBGSYS_RST in section 2.6.6.70 0x078C DBGSYS Bus Gating Reset Register (Default Value: 0x0000_0000).
DSU_RSTN	PPU	Whether to reset is controlled by PPU according to the power mode.

Reset signal	Source	Description
CORE_RSTN	PPU	Whether to reset is controlled by PPU according to the power mode.

2.2.4 Programming Guidelines

The following takes CPU_L_PLL as an example, CPU_DSU_PLL and CPU_B_PLL are the same.

NOTE

It is not suggested to enable or disable the PLLs during usage. When the clock is not required, it is recommended to configure the PLL_OUTPUT_EN bit of PLL control register as 0.

2.2.4.1 Enabling the Linear Frequency Modulation PLLs

Step 1 Write 1 to the PLL_SSC_CLK_SEL bit (bit [29]) of CPU_L_PLL_SSC_REG register.

Step 2 Configure the N, M, and P factors of the CPU_L_PLL_CTRL_REG register.

Step 3 Write 1 to the PLL_PLL_EN bit (bit [31]) and the PLL_LDO_EN bit (bit [30]) of the CPU_L_PLL_CTRL_REG register.

Step 4 Write 1 to the LOCK_ENABLE bit (bit [29]) of the CPU_L_PLL_CTRL_REG register.

Step 5 Write 1 to the PLL_UPDATE bit (bit [26]) of the CPU_L_PLL_CTRL_REG register.

Step 6 Wait for the value of the PLL_UPDATE bit to change to 0.

Step 7 Wait for the status of the Lock to change to 1.

Step 8 Delay 10 ms.

Step 9 Write 0 to the PLL_SSC_CLK_SEL bit (bit [29]) of CPU_L_PLL_SSC_REG register.

2.2.4.2 Configuring the Frequency of Linear Frequency Modulation PLLs

Step 1 Configure the PLL_SSC_STEP bit (bit [3:0]) of the CPU_L_PLL_SSC_REG register to select required frequency modulation slope.

Step 2 Configure the PLL_SSC bit (bit [28:12]) of the CPU_L_PLL_SSC_REG register to set the SSC amplitude.

Step 3 Write 1 to the PLL_SSC_MODE bit (bit [31]) of the CPU_L_PLL_SSC_REG register to enable linear frequency modulation.

Step 4 Write 1 to the PLL_UPDATE bit (bit [26]) of the CPU_L_PLL_CTRL_REG register to update PLL configuration parameters.

Step 5 Wait for the value of the PLL_UPDATE bit to change to 0.

Step 6 Configure the N factor of the CPU_L_PLL_CTRL_REG register.

Step 7 Write 1 to the PLL_UPDATE bit (bit [26]) of the CPU_L_PLL_CTRL_REG register to update PLL configuration parameters.

Step 8 Wait for the value of the PLL_UPDATE bit to change to 0.

Step 9 Write 0 to the PLL_SSC_MODE bit (bit [31]) of the CPU_L_PLL_SSC_REG register to disable linear frequency modulation.

Step 10 Write 1 to the PLL_UPDATE bit (bit [26]) of the CPU_L_PLL_CTRL_REG register to update PLL configuration parameters.

Step 11 Wait for the value of the PLL_UPDATE bit to change to 0.

2.2.4.3 Disabling the Linear Frequency Modulation PLLs

Follow the steps below to disable the PLL:

Step 1 Write 0 to the LOCK_ENABLE bit (bit [29]) of the CPU_L_PLL_CTRL_REG register.

Step 2 Write 0 to the PLL_PLL_EN bit (bit [31]) and the PLL_LDO_EN bit (bit [30]) of the CPU_L_PLL_CTRL_REG register.

Step 3 Write 1 to the PLL_SSC_CLK_SEL bit (bit [29]) of CPU_L_PLL_SSC_REG register.

Step 4 Write 1 to the PLL_UPDATE bit (bit [26]) of the CPU_L_PLL_CTRL_REG register.

Step 5 Wait for the value of the PLL_UPDATE bit to change to 0.

Step 6 Write 0 to the PLL_SSC_CLK_SEL bit (bit [29]) of CPU_L_PLL_SSC_REG register.

2.2.5 Register list

Module Name	Base Address	Description
CPU_SUBSYS_CTRL	0x08000000	CPU Subsystem Control (4KB)
TIMESTAMP_STA	0x08010000	Timestamp Status Registers (4KB)
TIMESTAMP_CTRL	0x08020000	Timestamp Control Registers (4KB)
CPU_PLL_CFG	0x08817000	Cluster PLL configure (4KB)

2.2.5.1 CPU_SUBSYS_CTRL Register List

Register Name	Offset	Description
GENER_CTRL_REG0	0x0000	General Control Register0
GIC_JTAG_RST_CTRL	0x000C	GIC and JTAG reset control Register
DBG_STATE	0x001C	Debug State Register
CPU0_CTRL_REG	0x0020	CPU0 Control Register

2.3 RISC-V System (RISCV)

2.3.1 Overview

The RISC-V system includes RISC IP core and related peripheral devices (AHB_Decoder, AHB2APB, RISCV_CFG, RISC_TIMESTAMP, and so on), which is able to be interconnected to MCU system by MCU AHB Matrix.

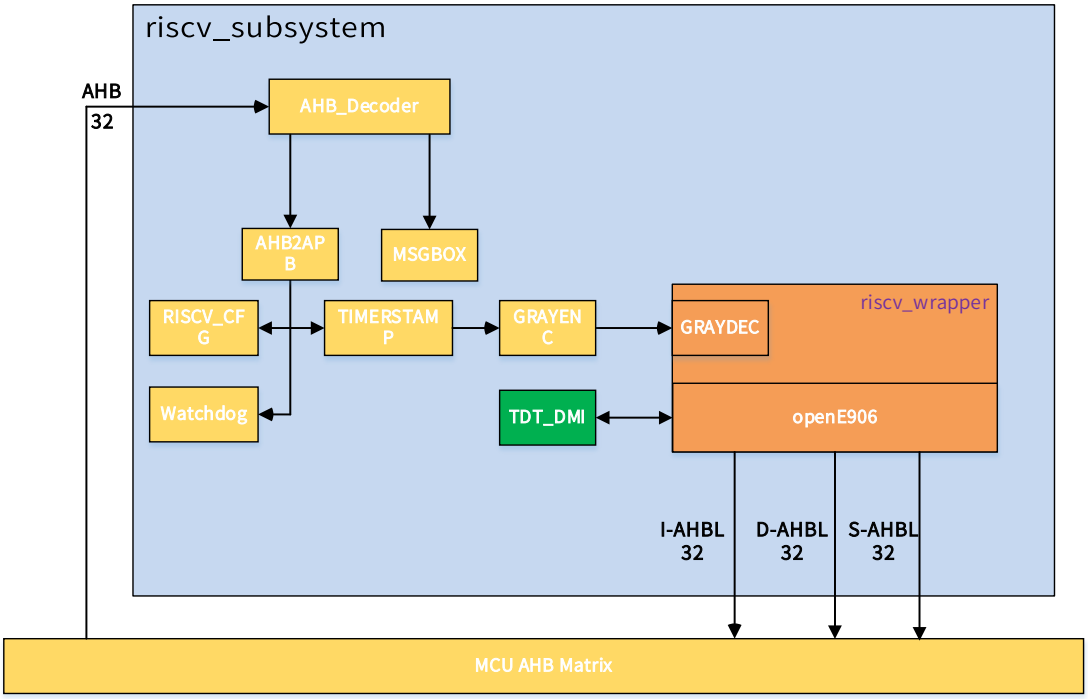
The RISC-V system has the following features:

- Configurable start address via software
- Supports standby in low-power mode and wake-up through external interrupts
- Separate TIMERSTAMP supports timing immediately after reset is released
- Separate watchdog supports to reset the SoC system when the RISC-V system malfunctions
- Separate message box supports communicating with other modules
- Supports separate PMU check module

2.3.2 Block Diagram

The following figure shows the block diagram of RISC-V system.

Figure 2-2 RISC-V System Block Diagram



2.3.3 Register List

Module Name	Base Address
RISCV_CFG	0x0713_0000

2.4 GPU

The GPU has the following features:

- ARM G57 MC1 GPU
- Supports OpenGL ES 3.2/2.0/1.1, Vulkan1.1/1.2/1.3, and OpenCL2.2
- Output and input format: 8-bit, 10-bit, and 16-bit YUV
- Anti-aliasing algorithm
- High memory bandwidth and low power consumption in 3D graphics processing
- AMBA4 AXI slave interface
- Latency tolerance
- Texture compression
- Configurable power management
- AFBC1.3
- Supports Digital Rights Management (DRM)

2.5 BROM System

2.5.1 Overview

The system has several ways to boot. It has an integrated on-chip Boot ROM (BROM) that is considered the primary program-loader. On the startup process, the A527 starts to fetch the first instruction from address 0x0, where is the BROM located at.

The BROM system is divided into two parts: the firmware exchange launch (FEL) module and the Medium Boot module. FEL is responsible for writing the external data to the local NVM, and Medium Boot is responsible for loading an effective and legitimate BOOT0 from NVM and running.

The BROM system includes the following features:

- Supports CPU0 (the Core0 of ARM CPU) boot process
- Supports mandatory upgrade process through USB and SD card
- Supports GPADC0 pin and eFuse module to select the boot media type
- Supports normal booting and secure booting
- Secure BROM loads only certified firmware
- Ensures that the secure boot is in a trusted environment

2.5.2 Functional Description

2.5.2.1 Selecting the Boot Medium

The BROM system supports the following boot media:

- SD Card
- eMMC
- RAW NAND Flash
- SPI NOR Flash (Quad Mode and Single Mode)
- SPI NAND Flash

There are two ways to select the boot medium: GPADC Pin Select and eFuse Select. The BROM will read the state of BOOT_MODE first, and then select the boot medium according to the state of BOOT_MODE. The BOOT_MODE is the BROM_Config in the eFuse mapping.

The following table shows the BOOT_MODE setting:

Table 2-5 BOOT_MODE Setting

BOOT_MODE[0]	Boot Select type
0	GPADC Selection
1	eFuse Selection

NOTE

The BOOT_MODE BIT is bit [0] of the eFuse register 0x03006210.

GPADC Boot Selection

If the state of the BOOT_MODE is 0, choose the GPADC Boot Selection.

If BROM failed to boot from the selected medium, it will try other media with the following priority:

EMMC_USR -> EMMC_BOOT -> SLC_NAND -> MLC_NAND -> SPI_NOR -> SPI_NAND

And the medium selected by GPADC will be skipped.

For example, when BROM failed to boot from SPI NOR, it will try other media with the following priority:

SPI NOR (selected by GPADC) -> EMMC_USR -> EMMC_BOOT -> SLC_NAND -> MLC_NAND -> SPI_NOR (try at first, skipped) -> SPI_NAND

The following table shows GPADC Boot Select setting.

Table 2-6 GPADC Boot Select Setting

KEY_VALUE	Boot Select type
0x00-0xB6	SD Card->MLC NAND->SLC NAND->try (except SPI in PJ)
0xB7-0x22B	SD Card->SLC NAND->MLC NAND->try (except SPI in PJ)
0x22C-0x3AF	SD Card->EMMC_USER->EMMC_BOOT->try (except SPI in PJ)
0x3B0-0x57B	SD Card->EMMC_BOOT->EMMC_USER->try (except SPI in PJ)
0x57C-0x73C	SD Card->SPI NOR->try (except SPI in PJ)
0x73D-0x8CC	SD Card->SPI NAND->try (except SPI in PJ)
0x8CD-0xB49	SD Card->SPI NOR in PJ->try
0xB4A-0xE7C	SD Card->SPI NAND in PJ->try
0x8CD-0xFFF	Reserved

NOTE

When trying SPI NOR, BROM try 4 wire mode first, then 1 wire mode.

eFuse Boot Selection

If the state of the BOOT_MODE is 1, choose the eFuse Boot Selection.

The eFuse_Boot_Select_Cfg is divided into 3 groups and each group is 4-bit. The following table shows the groups of eFuse_Boot_Select.

Table 2-7 Groups of eFuse_Boot_Select

eFuse_Boot_Select_Cfg [11:0]	Description
eFuse_Boot_Select[3:0]	eFuse_Boot_Select_1

eFuse_Boot_Select_Cfg [11:0]	Description
eFuse_Boot_Select[7:4]	eFuse_Boot_Select_2
eFuse_Boot_Select[11:8]	eFuse_Boot_Select_3

These three groups take effect with the following priority:

eFuse_Boot_Select_1 -> eFuse_Boot_Select_2 -> eFuse_Boot_Select_3

For example, eFuse_Boot_Select_2 will not take effect unless eFuse_Boot_Select_1 is set as 0b1111(skip), eFuse_Boot_Select_3 will not take effect unless eFuse_Boot_Select_2 is set as 0b1111(skip), etc. If all three groups are set to 0b1111, no other groups can be used for boot select, BROM assume "try" is selected.

In the Try mode, the BROM follows the order below to select the boot medium:

SD Card -> eMMC -> NAND FLASH -> SPI NOR -> SPI NAND

The following table shows the boot medium priority for the different values of eFuse_Boot_Select_n, where n = [3:1].

Table 2-8 eFuse Boot Select Setting

eFuse_Boot_Select_n	Boot media
0000	Try (except SPI in PJ)
0001	SLC NAND -> MLC NAND
0010	EMMC_USER -> EMMC_BOOT
0011	SPI NOR
0100	SPI NAND
0101	MLC NAND -> SLC NAND
0110	MMC_BOOT -> EMMC_USER
1011	SPI NOR in PJ
1100	SPI NAND in PJ
1111	When n is 1 or 2: The boot medium is decided by the value of **eFuse_Boot_Select_(n + 1)**. When n is 3: Select the boot medium in Try mode.

NOTE

The status of the eFuse boot select pin is the bit [11:0] of the eFuse register 0x03006212.

2.5.2.2 Selecting the Boot Mode

For SoCs that have implemented and enabled the ARM TrustZone technology, there are two boot modes: Normal BROM Mode and Secure BROM Mode.

Secure BROM Mode is designed to protect against attackers modifying the code or data areas in the programmable memory.

During the startup process, the BROM will select the boot mode according to the value of the Secure Enable bit. If the value of Secure Enable bit is 0, the system will boot in Normal BROM Mode. Otherwise, it will boot in Secure BROM Mode.

NOTE

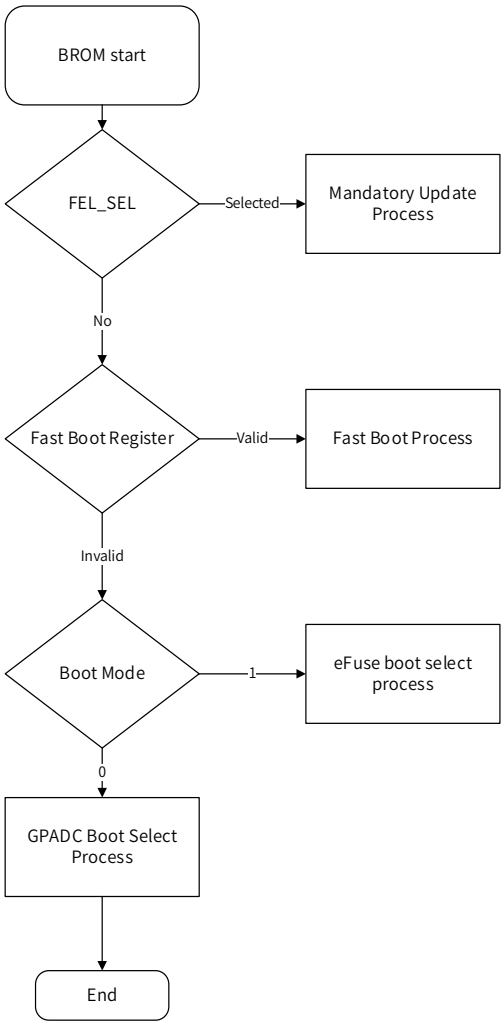
The System on Chip (SoC) supports the ARM TrustZone technology. If the Secure Enable Bit is enabled, the BROM will be safely booted based on this ARM TrustZone technology.

Normal BROM Mode

In Normal BROM Mode, the system boot starts from CPU0, and then the BROM will read the state of the FEL pin. If the FEL pin is high, the system will jump to the fast boot process. If it is low, the system will jump to the mandatory upgrade process.

The following figure shows the boot process in Normal BROM Mode.

Figure 2-3 Boot Process in Normal BROM Mode



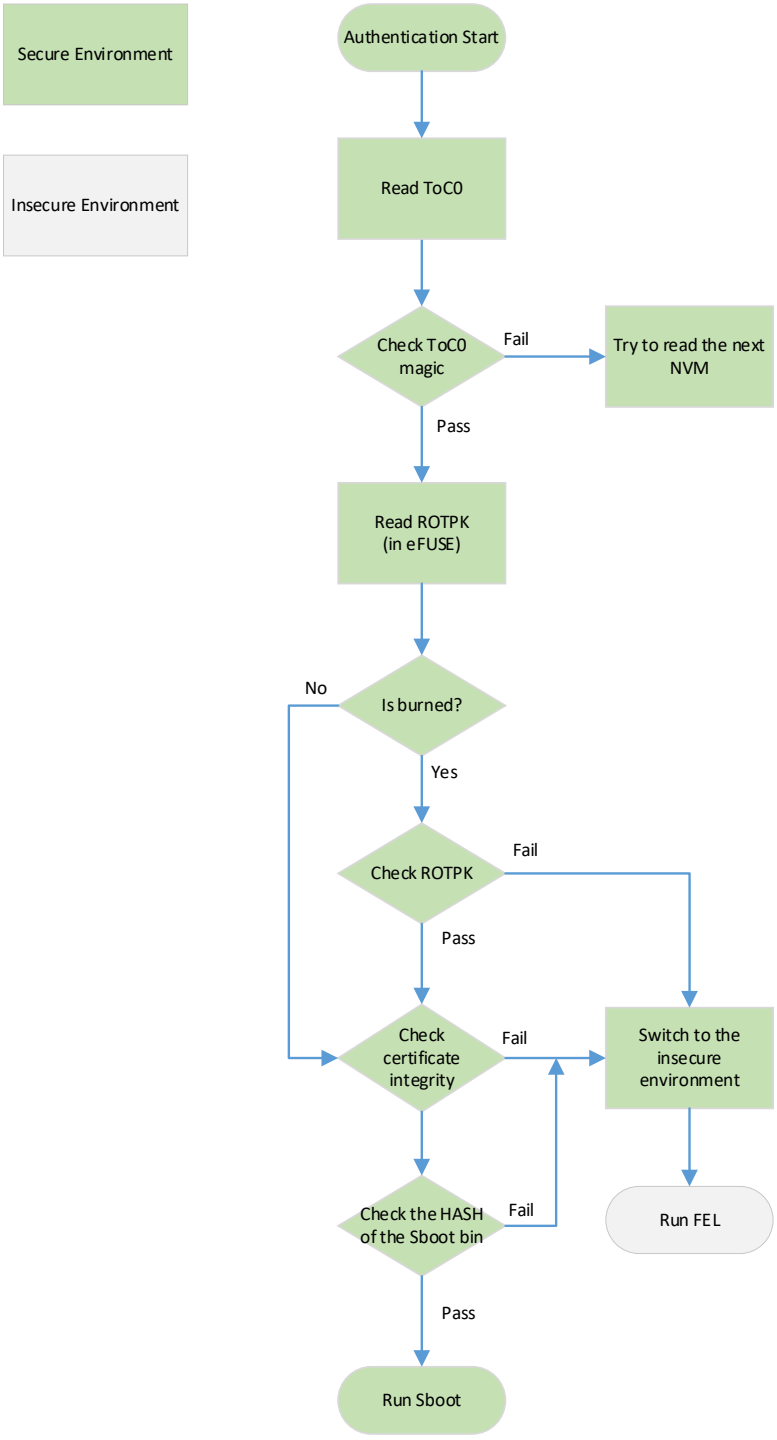
Secure BROM Mode

The process of selecting the boot medium in Secure Boot Mode is the same as that in Normal Boot Mode.

In Secure Boot Mode, after the boot medium is selected, the system additionally runs the Security Boot software to authenticate the Sboot bin file.

The following figure shows the authentication process.

Figure 2-4 Authentication Process in Secure BROM Mode



Secure BROM Requirements

The Secure Boot has the following some requirement:

- Supports X509 certificate

  The certificate is used to check whether the Security Boot software is modified or replaced. Before running the Security Boot software, the system checks the integrity of the certificate make sure the software has not been modified or replaced.

- Supports cryptographic algorithms
  - AES-128
  - SHA-256
  - RSA-2048
  - AES, DES

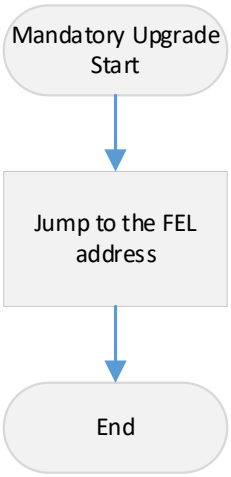
  The system uses the Crypto Engine (CE) hardware module to accelerate the speed of encryption and decryption. The standard cryptography ensures the reliability of the firmware images. The reliable firmware image ensures that the system security state can be as expected.

- Support OTP/eFuse

2.5.2.3 Mandatory Upgrade Process

If the FEL pin is detected to pull low, the system will jump to the mandatory upgrade process. The following figure shows the mandatory upgrade process.

Figure 2-5 Mandatory Upgrade Process



NOTE

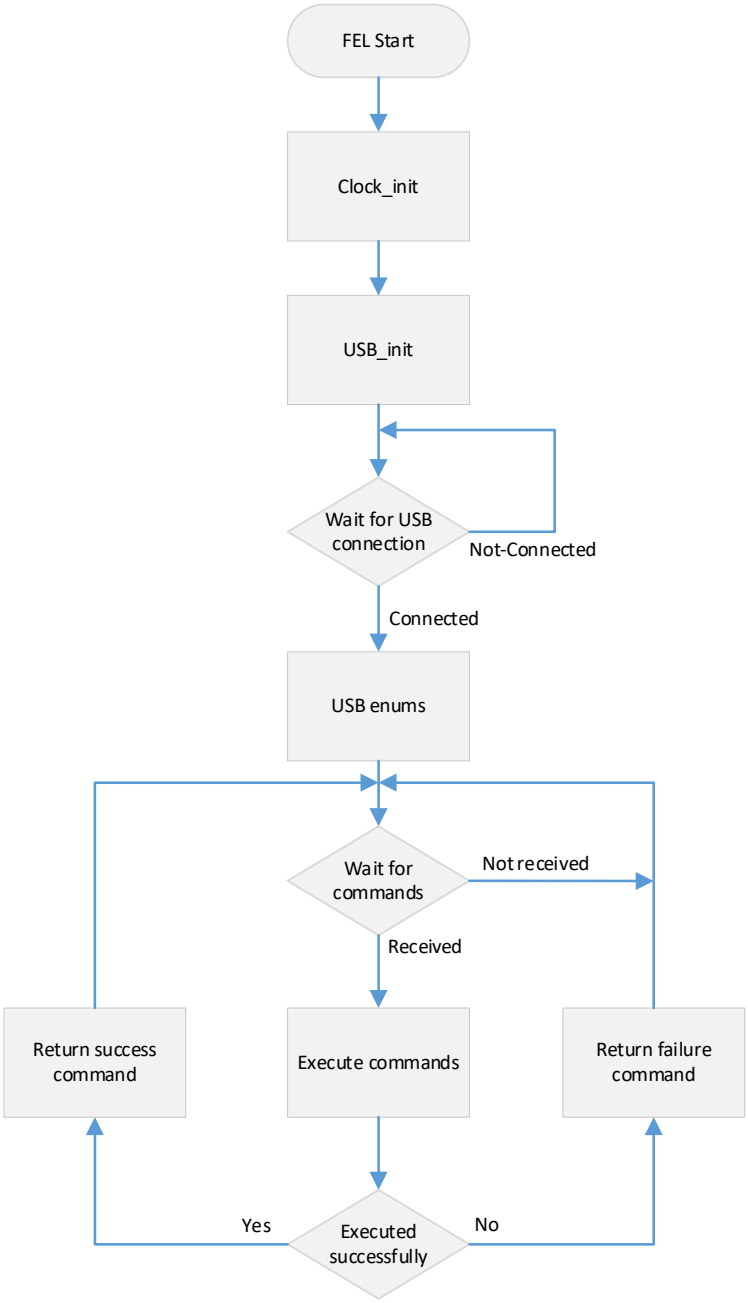
The FEL address of the Normal BROM is 0x20.

FEL Process

When the system enters mandatory upgrade process, it will jump to the FEL process.

The following figure shows the FEL upgrade process.

Figure 2-6 USB FEL Process



2.5.2.4 Fast Boot Process

If the value of the Fast Boot register (0x07090120) in RTC module is not zero, the system will enter the fast boot process. The following table shows the boot medium priority for different values of the Fast Boot register.

Table 2-9 Fast Boot Select Setting

Reg_bit[31:28]	Boot Select type
1	SD Card->MLC NAND -> SLC NAND -> TRY
2	SD Card->EMMC_USER -> EMMC_BOOT -> TRY
3	SD Card->SPI NOR(1 wire)-> SPI NOR(4 wire)-> TRY
4	SD Card->SPI NAND -> TRY
5	SD Card->EMMC_BOOT -> EMMC_USER -> TRY
6	SD Card->SLC NAND -> MLC NAND -> TRY
7	Reserved
8	SD Card->SPI NOR(4 wire)-> SPI NOR(1 wire)-> TRY
10	SD Card->SPI NOR(4 wire) in PJ -> TRY
11	SD Card->SPI NAND in PJ-> TRY

NOTE

- The Fast Boot register bit [27:0] is used record the media information.
- Unused value like 7 regarded as “TRY”.

2.6 Clock Controller Unit (CCU)

2.6.1 Overview

The clock controller unit (CCU) controls the PLL configurations and most of the clock generation, division, distribution, synchronization, and gating. The input signals of the CCU include the external clock for the reference frequency (24 MHz). The outputs from the CCU are mostly clocks to other blocks in the system.

The CCU includes the following features:

- 11 PLLs
- Bus source and divisions
- Clock output control
- Configuring modules clock
- Bus clock gating
- Bus software reset

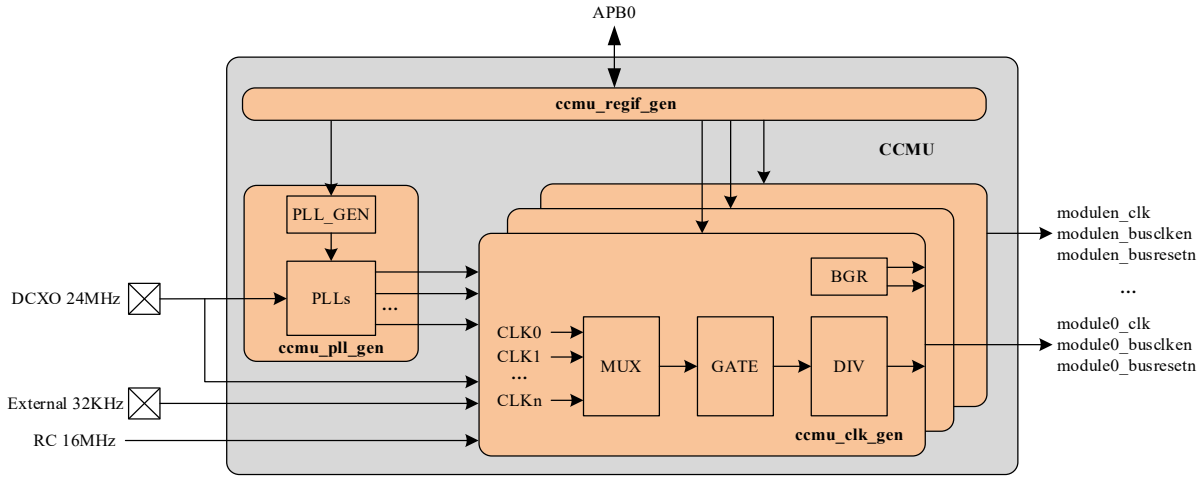
NOTE

- There are 16 PLLs in A527. 11 PLLs in CCU, 4 PLLs in CPUX system, and 1 PLL in MCU_PRCM.
- CCU describes module clocks in CPUX domain excluding the clock of CPUX system.
- For clock description of CPUX system, please refer to section 2.2.3.2 CPU PLL Distribution and Clock Sources.
- For module clocks in CPUS domain, please refer to section 2.12 Power Reset Clock Management (PRCM).

2.6.2 Block Diagram

The following figure shows the functional block diagram of the CCU.

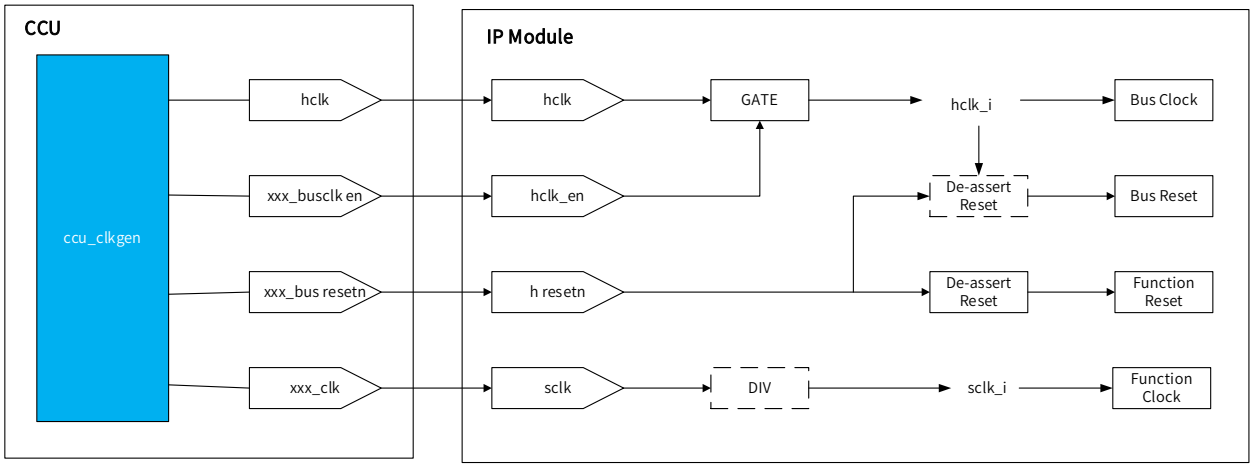
Figure 2-7 CCU Block Diagram



2.6.3 Functional Description

2.6.3.1 Typical Application

Figure 2-8 CCU Typical Application Diagram



CCU outputs bus clock, bus reset, function clock, and function reset to each IP module.

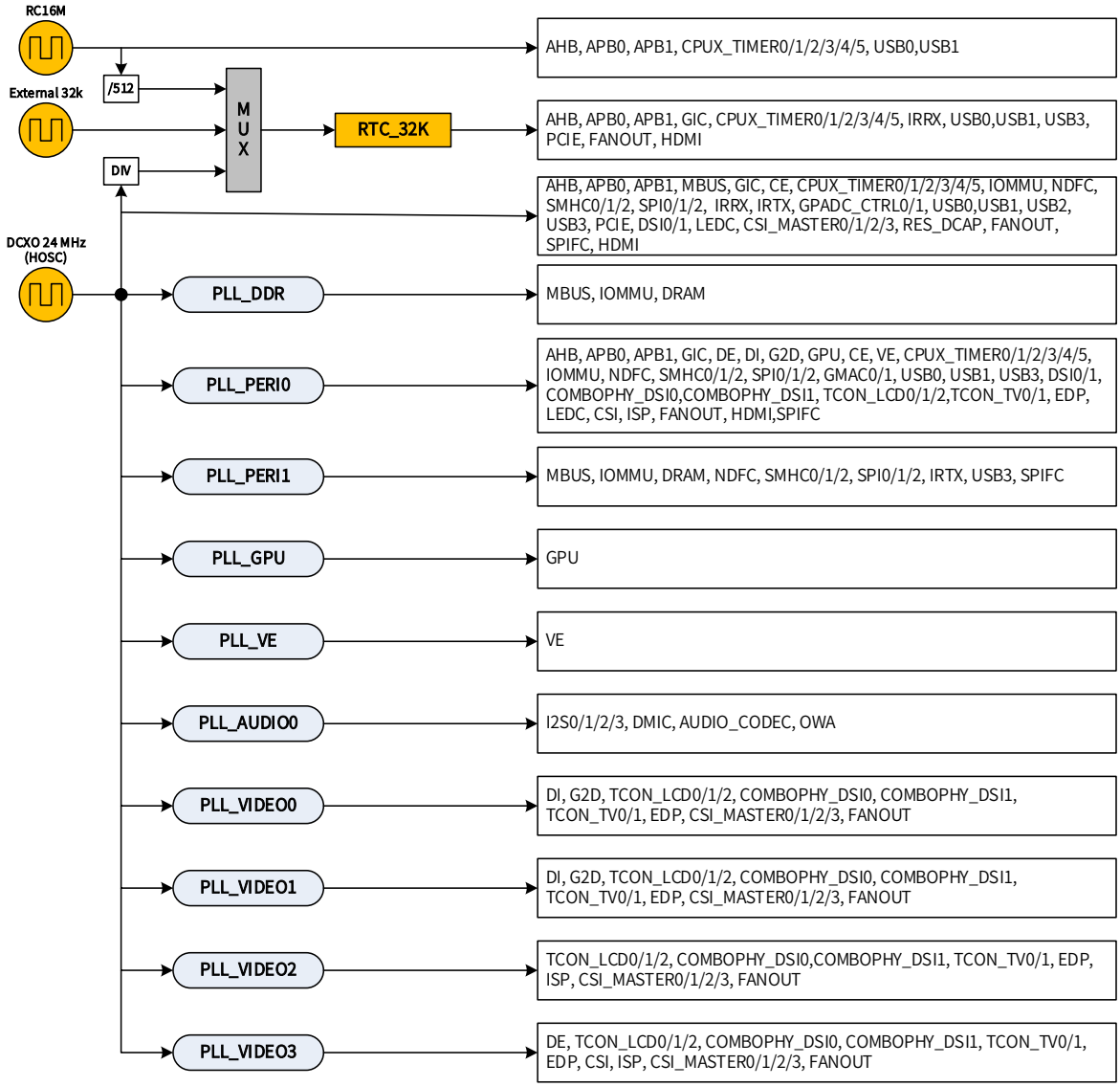
It is needed to enable the bus clock gating signal before using the bus clock. For some subsystems, CCU outputs special bus clock with clock gating added. You also need to enable the bus clock gating signal before using the special bus clock.

The IP reset sources from the synchronous release of the input reset signal. To ensure the synchronous release is implemented in every module, you need to release the reset signal before enabling the clock gating signal of the function clock.

2.6.3.2 PLL Distribution

The following figure shows the block diagram of the PLL distribution.

Figure 2-9 PLL Distribution



2.6.3.3 PLL Features

The following table shows the PLL features.

Table 2-10 PLL Features

PLL	Stable Operating Frequency	Actual Operating Frequency	Spread Spectrum	Linear FM	Pk-Pk	Lock Time
PLL_DDR	1.26 GHz～2.52 GHz	< 2.5 GHz	Yes	No	< 200ps	500us

PLL	Stable Operating Frequency	Actual Operating Frequency	Spread Spectrum	Linear FM	Pk-Pk	Lock Time
PLL_GPU	1.26 GHz～2.52 GHz	< 1.5 GHz	Yes	No	< 200ps	500us
PLL_PERI0	1.26 GHz～2.52 GHz	2x：1.2 GHz 3x：800 MHz 5x：480 MHz	Yes	No	< 200ps	500us
PLL_PERI1	1.26 GHz～2.52 GHz	2x：1.248 GHz/1.2 GHz 3x：832 MHz/800 MHz 5x：499.2 MHz/480 MHz	Yes	No	< 200ps	500us
PLL_VE	1.26 GHz～2.52 GHz	< 1.5 GHz	Yes	No	< 200ps	500us
PLL_AUDIO0	1.26 GHz～2.52 GHz	1x：22.5792 MHz 4x：22.5792*4 MHz	Yes	No	< 200ps	500us
PLL_VIDEO	1.26 GHz～2.52 GHz	3x:792 MHz 4x:1188 MHz	Yes	No	< 200ps	500us

2.6.4 Programming Guidelines

NOTE

It is not suggested to enable or disable the PLLs frequently during usage. Because the enabling and disabling of PLL will cause a mutual interference between PLLs, which will affect system stability. When the clock is not required, it is recommended to configure the PLL_OUTPUT_GATE bit of PLL control register as 0 instead of writing 0 to the enable bit.

2.6.4.1 Enabling the PLL

Follow the steps below to enable the PLL:

Step 1 Configure the N, M, and P factors of the PLL control register.

Step 2 Write 1 to the PLL_EN bit (bit [31]) and the PLL_LDO_EN bit (bit [30]) of the PLL control register, write 0 to the PLL_OUTPUT_GATE bit (bit [27]) of the PLL control register.

Step 3 Write 1 to the LOCK_ENABLE bit (bit [29]) of the PLL control register.

Step 4 Wait for the status of the Lock to change to 1.

Step 5 Delay 20 us.

Step 6 Write the PLL_OUTPUT_GATE bit (bit [27]) of the PLL control register to 1 and then the PLL will be available.

2.6.4.2 Configuring the Frequency of General PLLs

Step 1 Make sure the PLL is enabled. If not, refer to section 2.6.4.1 Enabling the PLL to enable the PLL.

Step 2 Configure the PLL_OUTPUT_GATE bit (bit [27]) of the PLL control register as 0 to disable the output gate of the PLL. Because, general PLLs are unavailable in the process of frequency modulation.

Step 3 Configure the N and M factors. (It is not suggested to configure M1 factor)

Step 4 Write 0 and then write 1 to the LOCK_ENABLE bit (bit [29]) of the PLL control register.

Step 5 Wait for the LOCK bit (bit [28]) of the PLL control register to 1.

Step 6 Configure PLL_OUTPUT_GATE bit (bit [27]) of the PLL control register to 1.

2.6.4.3 Configuring the Frequency of PLL_AUDIO0

The frequency configuration formula of PLL_AUDIO0:

PLL_AUDIO0 = 24 MHz*N/M0/M1/P

PLL_AUDIO0 does not support dynamic adjustment because changing any parameter of N, M0, M1, and P will affect the normal work of PLL, and the PLL will need to be relocked.

Generally, PLL_AUDIO0 only needs two frequency points: 24.576*4 MHz or 22.5792*4 MHz. For these two frequencies, there are usually special recommended matching factors. To implement the desired frequency point of PLL_AUDIO0, you need to use the decimal frequency-division function, so follow the steps below:

Step 1 Configure the N, M0, M1 and P factors.

Step 2 Write 1 to the PLL_SDM_EN bit (bit [24]) of PLL_AUDIO0_CTRL register.

Step 3 Configure PLL_AUDIO0_PAT0_CTRL register to enable the digital spread spectrum.

Step 4 Write 0 and then write 1 to the LOCK ENABLE bit (bit [29]) of PLL_AUDIO0_CTRL register.

Step 5 Write 1 to the LOCK bit (bit [28]) of PLL_AUDIO0_CTRL register.

NOTE

- When the P factor of PLL_AUDIO0 is an odd number, the clock output is an unequal-duty-cycle signal.
- A527 includes PLL_AUDIO0 and PLL_AUDIO1. For detailed description of PLL_AUDIO1, please refer to section 2.12 Power Reset Clock Management (PRCM).

2.6.4.4 Disabling the PLL

Follow the steps below to disable the PLL:

Step 1 Write 0 to the PLL_EN bit (bit [31]) and the PLL_LDO_EN bit (bit [30]) of the PLL control register.

Step 2 Write 0 to the LOCK_ENABLE bit (bit [29]) of the PLL control register.

2.6.4.5 Implementing Spread Spectrum

The spread spectrum technology is to convert a narrowband signal into a wideband signal. It helps to reduce the effect of electromagnetic interference (EMI) associated with the fundamental frequency of the signal.

For the general PLL frequency, the calculation formula is as follows:

$$f = \frac{N+1+X}{P\cdot(M0+1)\cdot(M1+1)}\cdot 24MHz,\ 0<X<1$$

Where,

P is the frequency division factor of module or PLL;

M0 is the post-frequency division factor of PLL;

M1 is the pre-frequency division factor of PLL;

N is the frequency doubling factor of PLL;

X is the amplitude coefficient of the spread spectrum.

The parameters N, P, M1, and M0 are for the frequency division.

When M1 = 0, M0 = 0, and P = 1 (no frequency division), the calculation formula of PLL frequency can be simplified as follows:

$$f = (N+1+X)\cdot 24MHz,\ 0<X<1$$

$$[f_1, f_2] = (N+1+[X_1, X_2])\cdot 24MHz$$

$$SDM\_BOT = 2^{17}\cdot X_1$$

$$\mathrm{WAVE\_STEP} = 2^{17}\cdot(X_2 - X_1)/(24\ \mathrm{MHz}/PREQ)\cdot 2$$

Where, SDM_BOT and WAVE_STEP are bits of the PLL pattern control register, and PREQ is the frequency of the spread spectrum.

NOTE

Different PLLs have different calculate formulas, refer to the CTRL register of the corresponding PLL in section 2.6.6 Register Description.

Follow the steps below to implement the spread spectrum:

Step 1 Configure the control register of the corresponding PLL

a) Calculate the factor N and decimal value X according to the PLL frequency and PLL frequency formula. Refer to the control register of the corresponding PLL (named PLL_xxx_CTRL_REG, where xxx is the module name) in 2.6.6 Register Description for the corresponding PLL frequency formula.

b) Write M0, M1, N, and PLL frequency to the PLL control register.

c) Configure the PLL_SDM_EN bit (bit [24]) of the PLL control register to 1 to enable the spread spectrum function.

Step 2 Configure the pattern control register of the corresponding PLL

a) Calculate the SDM_BOT and WAVE_STEP of the pattern control register according to decimal value X and spread spectrum frequency (the bit [18:17] of the PLL pattern register)

b) Configure the spread spectrum mode (SPR_FREQ_MODE) to 2 or 3.

c) If the PLL_INPUT_DIV2 of the PLL control register is 1, configure the spread spectrum clock source select bit (SDM_CLK_SEL) of the PLL pattern control register to 1. Otherwise, configure SDM_CLK_SEL to the default value 0.

d) Write SDM_BOT, WAVE_STEP, PREQ, SPR_FREQ_MODE, and SDM_CLK_SEL to the PLL pattern control register, and configure the SIG_DELT_PAT_EN bit (bit [31]) of this register to 1.

Step 3 Delay 20 us

2.6.4.6 Configuring Bus Clock

The bus clock supports dynamic switching, but the process of switching needs to follow the following two rules.

- From a higher frequency to a lower frequency: switch the clock source first, and then set the frequency division factor;
- From a lower frequency to a higher frequency: configure the frequency division factor first, and then switch the clock source.

The bus frequency for each bus is as follows:

- AXI: It is suggested to be configured as the CPU clock frequency divided by 3. when the CPU clock frequency is less than 1.2 GHz, AXI frequency could be configured as the CPU clock frequency divided by 2.
- AHB: Maximum 200 MHz
- APB0: Maximum 100 MHz
- APB1: Maximum 160 MHz

- MBUS: Maximum 700 MHz
- IOMMU: Maximum 600 MHz

2.6.4.7 Configuring Module Clock

For the Bus Gating Reset register of a module, the reset bit is de-asserted first, and then the clock gating bit is enabled to avoid potential problems caused by the asynchronous release of the reset signal.

For all module clocks except the DDR clock, configure the clock source and frequency division factor first, and then release the clock gating (that is, set to 1). For the configuration order of the clock source and frequency division factor, follow the rules below:

- With the increasing of the clock source frequency, configure the frequency division factor before the clock source;
- With the decreasing of the clock source frequency, configure the clock source before the frequency division factor.

2.6.5 Register List

NOTE

- Before switching the glitch-free MUX, ensure that
  - Every clock source is in use.
  - The switching time is longer than two clock periods of the slowest clock source.
- Before switching the normal MUX, ensure that the clock sources are closed.

Module Name	Base Address
CCU	0x0200 1000

Register Name	Offset	Description
PLL_DDR_CTRL_REG	0x0010	PLL_DDR Control Register
PLL_PERI0_CTRL_REG	0x0020	PLL_PERI0 Control Register
PLL_PERI1_CTRL_REG	0x0028	PLL_PERI1 Control Register
PLL_GPU_CTRL_REG	0x0030	PLL_GPU Control Register
PLL_VIDEO0_CTRL_REG	0x0040	PLL_VIDEO0 Control Register
PLL_VIDEO1_CTRL_REG	0x0048	PLL_VIDEO1 Control Register
PLL_VIDEO2_CTRL_REG	0x0050	PLL_VIDEO2 Control Register
PLL_VE_CTRL_REG	0x0058	PLL_VE Control Register
PLL_VIDEO3_CTRL_REG	0x0068	PLL_VIDEO3 Control Register
PLL_AUDIO0_CTRL_REG	0x0078	PLL_AUDIO0 Control Register

2.7 DMA Controller (DMAC)

2.7.1 Overview

The Direct Memory Access (DMA) is a method of transferring data between peripherals and memories (including the SRAM and DRAM) without using the CPU. It is an efficient way to offload data transfer duties from the CPU. Without DMA, the CPU has to control all the data transfers. While with DMA, the DMAC directly transfers data between a peripheral and a memory, between peripherals, or between memories.

The DMAC has the following features:

- Up to 16-ch DMA in CPUX domain and 16-ch DMA in CPUS domain
- Provides 53 peripheral DMA requests for data reading and 53 peripheral DMA requests for data writing
- Transferring data with linked list
- Flexible data width: 8 bits, 16 bits, or, 32 bits
- Programmable DMA burst length
- DRQ response includes waiting mode and handshake mode
- Supports non-aligned transform for memory devices
- DMA channels that support the following:
  - Pausing DMA
  - BMODE and I/O speed mode
  - DMA timeout

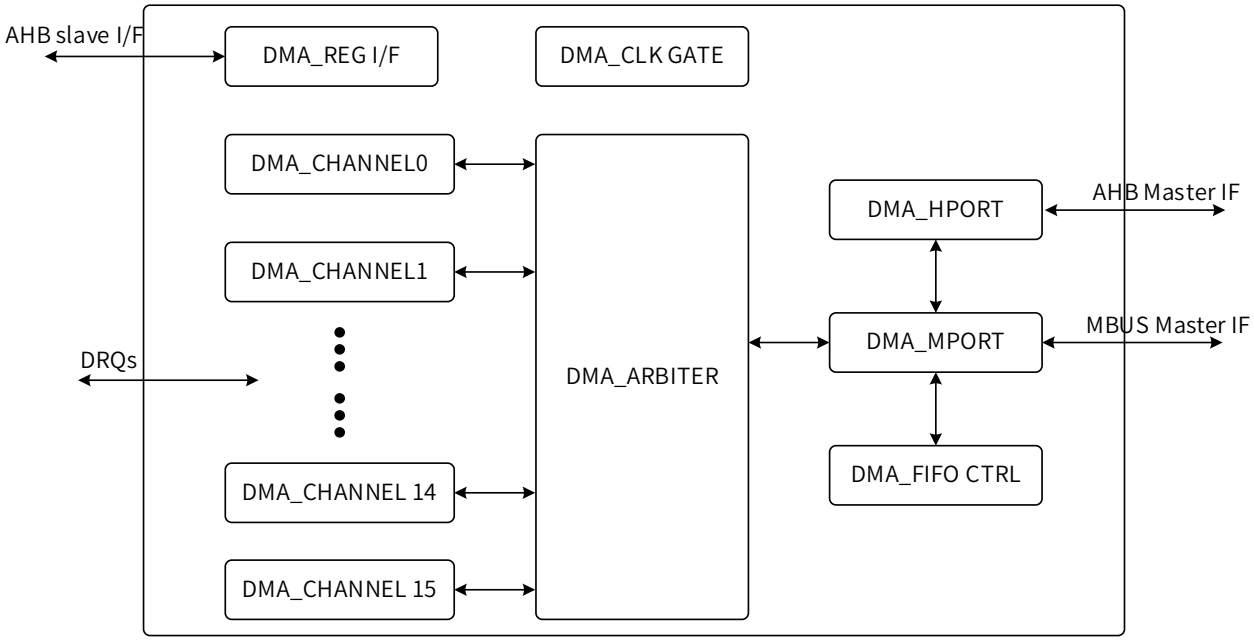
NOTE

The following description focuses on the DMA of the CPUX domain.

2.7.2 Block Diagram

The following figure shows a block diagram of DMAC.

Figure 2-10 DMAC Block Diagram



DMAC contains the following sub-blocks:

Table 2-11 DMAC Sub-blocks

Sub-block	Description
DMA_ARBITER	Arbitrates the DMA read/write requests from all channels, and converts the requests to the read/write requests of ports.
DMA_CHANNELs	DMA transfer engine. Each channel is independent. When the DMA requests from multiple peripherals are valid simultaneously, the channel with the highest priority starts data transfer first. The system uses the polling mechanism to decide the priorities of DMA channels. When DMA_ARBITER is idle, channel 0 has the highest priority, whereas channel 15 has the lowest priority. When DMA_ARBITER is busy processing the request from channel n, channel (n+1) has the highest priority. For n = 15, the channel (n + 1) should be channel 0.
DRQs	DMA requests. Peripherals use the DMA request signals to request a data transfer.
DMA_MPORT	Receives the read/write requests from DMA_ARBITER, and converts the requests to the corresponding MBUS access requests. It is mainly used for accessing the DRAM.
DMA_HPORT	The port for accessing the AHB Master. It is mainly used for accessing the SRAM and IO devices.
DMA_FIFO CTRL	Internal FIFO cell control module.
DMA_REG Interface	DMA_REG is the common register module that is mainly used to resolve AHB commands.

Sub-block	Description
DMA_CLKGATE	The control module for hardware auto clock gating.

The DMAC integrates 16 independent DMA channels and each channel has an independent FIFO controller. When the DMA channel starts, the DMAC gets a DMA descriptor from the DMA_DESC_ADDR_REG and uses it as the configuration information for the data transfer of the current DMA package. Then the DMAC can transfer data between the specified devices. After transferring a DMA package, the DMAC judges if the current channel transfer is finished via the linked address in the descriptor. If the linked address shows all the packages are transferred, the DMAC will end the chain transmission and close the channel.

2.7.3 Functional Description

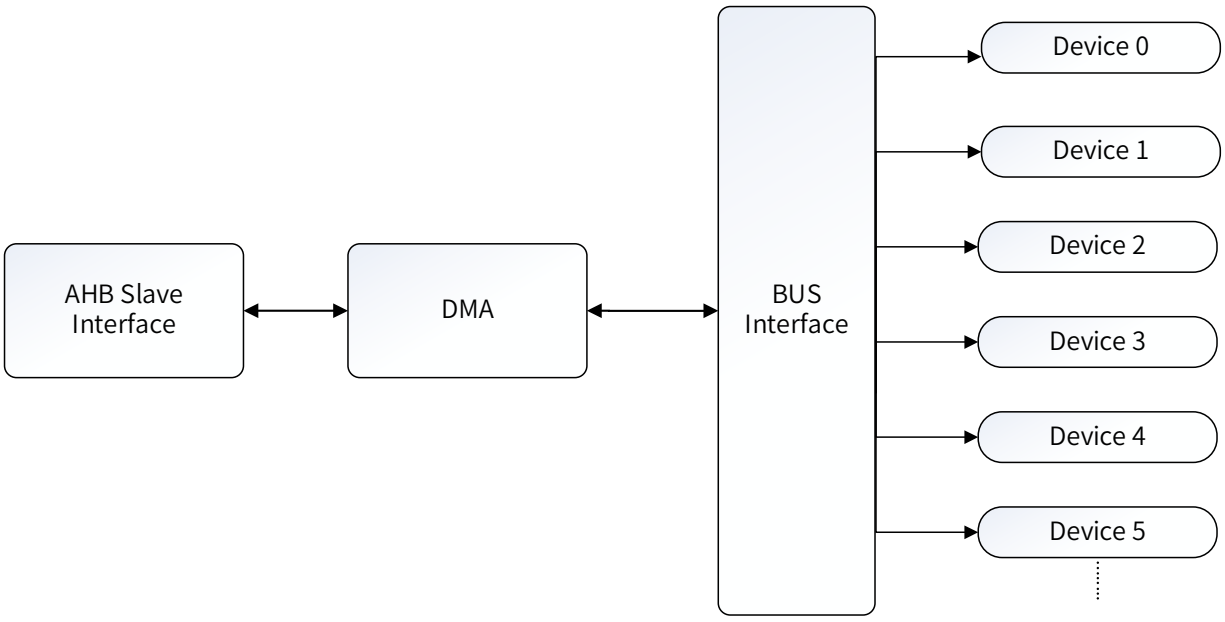
2.7.3.1 Clock

The DMAC is on AHB or MBUS. The clocks of AHB and MBUS influence the transfer efficiency of the DMAC.

2.7.3.2 Typical Application

The following figure shows a typical application of the DMAC.

Figure 2-11 DMAC Typical Application Diagram



2.7.3.3 DRQ Port of Peripherals

The following tables show the source DRQ types and destination DRQ types of different ports.

Table 2-12 DMA DRQ Type

Source DRQ Type		Destination DRQ Type	
port0	SRAM	port0	SRAM
port1	DRAM	port1	DRAM
port2		port2	
port3		port3	
port4		port4	
port5		port5	
port6		port6	
port7		port7	
port8		port8	
port9		port9	
port10	NDFC	port10	NDFC
port11		port11	
port12	GPADC_CTRL0	Port12	
port13	GPADC_CTRL1	port13	CIR_TX
port14	UART0_RX	port14	UART0_TX
port15	UART1_RX	port15	UART1_TX
port16	UART2_RX	port16	UART2_TX
port17	UART3_RX	port17	UART3_TX
port18	UART4_RX	port18	UART4_TX
port19	UART5_RX	port19	UART5_TX
port20	UART6_RX	Port20	UART6_TX
port21	UART7_RX	port21	UART7_TX
port22	SPI0_RX	port22	SPI0_TX
port23	SPI1_RX	port23	SPI1_TX
port24	SPI2_RX	port24	SPI2_TX
port25		port25	
port26		port26	
port27		port27	
port28		port28	
port29		port29	
Port30	USB0_EP1	Port30	USB0_EP1
Port31	USB0_EP2	Port31	USB0_EP2
Port32	USB0_EP3	Port32	USB0_EP3
Port33	USB0_EP4	Port33	USB0_EP4
Port34	USB0_EP5	Port34	USB0_EP5
Port35		Port35	
Port36		Port36	

Source DRQ Type		Destination DRQ Type	
Port37		Port37	
Port38		Port38	
Port39		Port39	
Port40		Port40	
Port41		Port41	
Port42		Port42	LEDC
Port43	TWI0	Port43	TWI0
Port44	TWI1	Port44	TWI1
Port45	TWI2	Port45	TWI2
Port46	TWI3	Port46	TWI3
Port47	TWI4	Port47	TWI4
Port48	TWI5	Port48	TWI5
Port49	S_TWI0	Port49	S_TWI0
Port50	S_TWI1	Port50	S_TWI1
Port51	S_UART0	Port51	S_UART0
Port52	S_UART1	Port52	S_UART1
Port53	S_SPI0	Port53	S_SPI0

Table 2-13 DMA DRQ Type of MCU_DMAC

Source DRQ Type		Destination DRQ Type	
port0	SRAM	port0	SRAM
port1	DRAM	port1	DRAM
port2	OWA	port2	OWA
port3	I2S0_RX	port3	I2S0_TX
port4	I2S1_RX	port4	I2S1_TX
port5	I2S2_RX	port5	I2S2_TX
port6	I2S3_RX	port6	I2S3_TX
port7	AUDIO_CODEC	port7	AUDIO_CODEC
port8	DMIC	port8	
port9	S_TWI0	port9	S_TWI0
port10	S_TWI1	port10	S_TWI1
port11	S_UART0	port11	S_UART0
port12	S_UART1	port12	S_UART1
Port13	S_SPI0	Port13	S_SPI0
Port14	S_TWI2	Port14	S_TWI2

2.7.3.4 DMA Descriptor

The DMAC descriptor is the configuration information of DMA transfer that decides the DMA working mode. Each descriptor includes 6 words: Configuration, Source Address, Destination

Address, Byte Counter, Parameter, and Link. The following figure shows the structure of the DMA descriptor.

Figure 2-12 DMA Descriptor

- **Configuration**: Configure the following information.
  - DRQ type: DRQ type of the source and destination devices.
  - Address counting mode: For both the source and destination devices, there are two address counting modes: the IO mode and linear mode. The IO mode is for IO devices whose address is fixed during the data transfer and the linear mode is for the memory whose address is increasing during the data transfer.
  - Transferred block length: How many times can non-memory peripherals transfer in a valid DRQ. The block length supports 1 time, 4 times, 8 times, and 16 times.
  - Transferred data width: The data width of operating the non-memory peripherals. The data width supports 8 bits, 16 bits, and 32 bits.

    NOTE

    The configuration supports BMODE mode. The BMODE is used in the following scenario: the source is an IO device, and the destination is a memory device. Setting the BMODE mode can limit the amount of block data transferred in DMA block transmission to the amount of data transferred when the DRQ threshold of the source IO device is 1.
- **Source Address**: Configure the address of the source device.
- **Destination Address**: Configure the address of the destination device.

  DMA reads data from the source address and then writes data to the destination address.

  Both the DMA source and destination addresses have 34 bits. In the descriptor, because there are only 32 bits in the Source/Destination Address field, another 2 bits are stored in the Parameter field.

The following table shows the details of the related fields in the descriptor.

Table 2-14 Source/Destination Address Distribution

Descriptor Group	Bit	Description
Source Address	31:0	DMA transfers the lower 32 bits of the 34-bit source address
Destination Address	31:0	DMA transfers the lower 32 bits of the 34-bit destination address
Parameter	31	TIMEOUT Enable TIMEOUT only can be enabled in BMODE and IOspeed should be disabled when using this function.
	30:29	TIMEOUT Configuration 00: Not use sub-functions 01: Generate an interrupt and suspend the transmission after timeout. 10: Generate an interrupt and end the transmission after timeout. 11: Generate an interrupt and skip to the next descriptor after timeout.
	28:20	TIMEOUT Configuration Timer time of channels.
	19:18	DMA transfers the higher 2 bits of the 34-bit destination address
	17:16	DMA transfers the high 2 bits of the 34-bit source address
	15:9	Reserved
	8	I/O Speed Mode Enable If this bit is enabled, DMA will transmit the data of the I/O device from the source device or the destination device, or both of them at a faster speed. **Note: IOspeed and BMODE cannot be enabled at the same time.**
	7:0	Wait Clock Cycles Set the waiting time in DRQ mode
Link	31:2	The address of the next group descriptor, the lower 30 bits of the word address
	1:0	The address of the next group descriptor, the higher 2 bits of the word address

From the above table, you can get:

Real DMA source address (in byte mode) = {Parameter [17:16], Source Address [31:0]};

Real DMA destination address (in byte mode) = {Parameter [19:18], Destination Address [31:0]};

Real link address (in byte mode) = {Link [1:0], Link [31:2], 2'b00}.

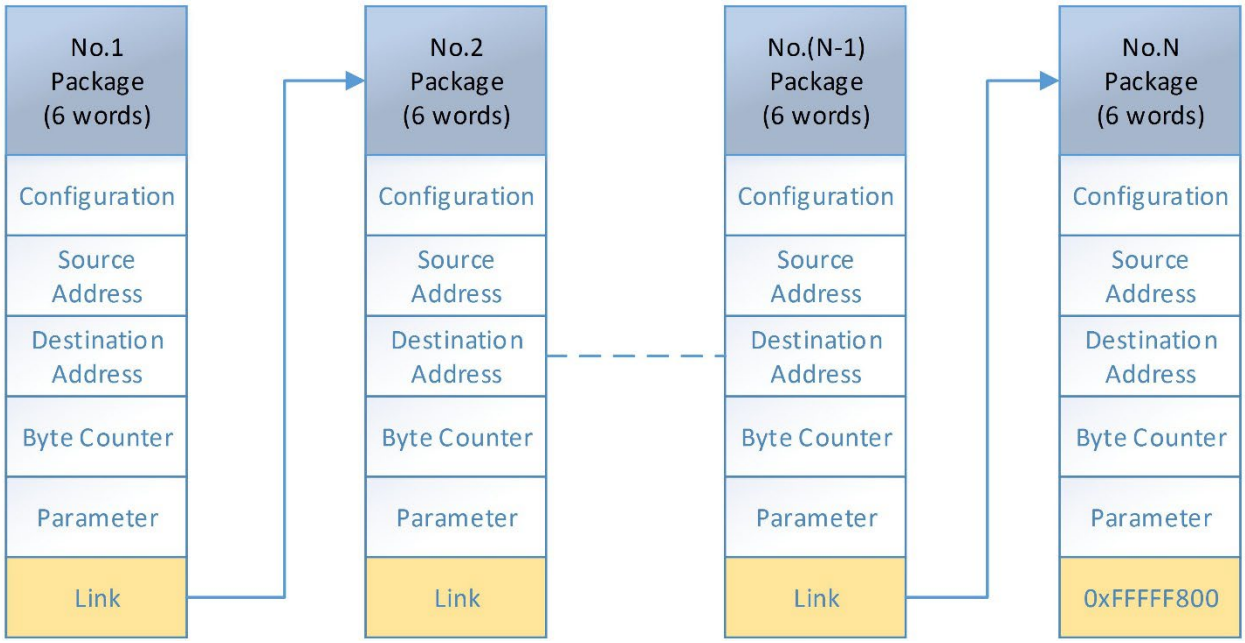
- **Byte counter**: Configure the data amount of a package. The maximum value is (2^25-1) bytes. If the data amount of the package reaches the maximum value, even if DRQ is valid, the DMA will stop the current transfer.

- **Parameter**: Configure the interval between the data block. The parameter is valid for non-memory peripherals. When DMA detects that the DRQ is high, the DMA transfers the data block and ignores the status changes of the DRQ until the data transfer finishes. After that, the DMA waits for certain clock cycles (WAIT_CYC) and executes the next DRQ detection. In addition, the Parameter is responsible for enabling and configuring TIMEOUT. In the case that the source device is an I/O device and the destination device is a memory device, the waiting time of a DRQ signal triggered by the source device can be set when TIMEOUT is enabled. When time is out, an interrupt signal of TIMEOUT will be generated by DMA. There are three sub-functions of TIMEOUT to be enabled (TIMEOUT will only generate interrupts if they are disabled): suspend the transmission of the current channel after an interrupt is generated; end the transmission of the current channel after an interrupt is generated; skip to the next descriptor for transmission after an interrupt is generated. (TIMEOUT only can be enabled in BMODE.)

  The Parameter also configures whether the IOspeed is enabled or not. If IOspeed is enabled, DMA will transmit the data of the I/O device from the source device or the destination device, or both of them at a faster speed. The larger block indicates a faster speed. However, when the block is 1, the speed won't change a lot even if the IO speed is enabled.

- **Link**: If the value of the link is 0xFFFFF800, the current package is at the end of the linked list. The DMAC will stop the data transfer after transferring the package; otherwise, the value of the link is considered as the descriptor address of the next package.

Figure 2-13 DMA Chain Transfer



2.7.3.5 Interrupts

There are four kinds of DMA interrupts: the half package interrupt, package end interrupt, and queue end interrupt.

- Half package interrupt

When enabled, the DMAC sends out a half package interrupt after transferring half of a package.

- Package end interrupt

When enabled, the DMAC sends out a package end interrupt after transferring a complete package.

- Queue end interrupt

When enabled, the DMAC sends out a queue end interrupt after transferring a complete queue.

- Timeout interrupt

When TIMEOUT is enabled, DMA will generate a timeout interrupt after timeout.

Notice that when CPU does not respond to the interrupts timely, or two DMA interrupts are generated very closely, the later interrupt may override the former one. That is, from the perspective of the CPU, the DMAC has only a system interrupt source.

2.7.3.6 Clock Gating

The DMA_CLK_GATE module is a hardware module for controlling the clock gating automatically. It provides clock sources for sub-modules in DMAC and the module local circuits.

The DMA_CLK_GATE module consists of two parts: the channel clock gate and the common clock gate.

Channel clock gate: Controls the DMA clock of the DMA channels. When the system accesses the register of the current DMA channel and the DMA channel is enabled, the channel clock gate automatically opens the DMA clock. With a 16-HCLK-cycle delay after the system finishes accessing the register or the DMA data transfer is completed, the channel clock gate automatically closes the DMA clock. Also, the clock for the related circuits, such as for the channel control and FIFO control modules, will be closed.

Common clock gate: Controls the clocks of the DMA common circuits. The common circuits include the common circuit of the FIFO control module, MPORT module, and MBUS. When all the DMA channels are disabled, the common clock gate automatically closes the clocks for the above circuits.

The DMA clock gating can support all the functions stated above or not by software.

2.7.3.7 Transfer Mode

The peripherals initiate data transfer by transmitting DMA request signals to the DMAC. After receiving the request signal, the DMAC converts it to the internal DRQ signal and controls the DMA data transfer.

The DMAC supports two data transfer modes: the waiting mode and handshake mode.

The principle of waiting mode

- When the DMAC detects a valid external request signal, the DMAC starts to operate the peripheral device. The internal DRQ always holds high before the transferred data amount reaches the transferred block length.
- When the transferred data amount reaches the transferred block length, the internal DRQ pulls low automatically.
- The internal DRQ holds low for certain clock cycles (WAIT_CYC), and then the DMAC restarts to detect the external requests. If the external request signal is valid, then the next transfer starts.

The principle of handshake mode

- When the DMAC detects a valid external request signal, the DMAC starts to operate the peripheral device. The internal DRQ always holds high before the transferred data amount reaches the transferred block length.
- When the transferred data amount reaches the transferred block length, the internal DRQ will be pulled down automatically. For the last data transfer of the block, the DMAC sends a DMA Last signal with the DMA commands to the peripheral device. The DMA Last signal will be packed as part of the DMA commands and transmitted on the bus. It is used to inform the peripheral device that it is the end of the data transfer for the current DRQ.
- When the peripheral device receives the DMA Last signal, it can judge that the data transfer for the current DRQ is finished. To continue the data transfer, it sends a DMA Active signal to the DMAC.

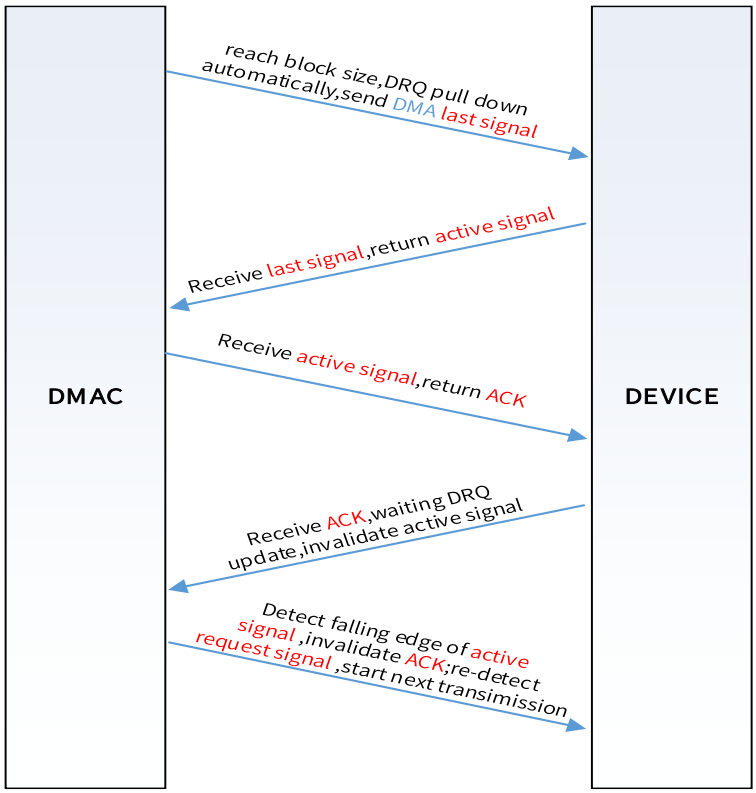
NOTE

One DMA Active signal will be converted to one DRQ signal in the DMA module. To generate multiple DRQs, the peripheral device needs to send out multiple DMA Active signals via the bus protocol.

- When the DMAC received the DMA Active signal, it sends back a DMA ACK signal to the peripheral device.
- When the peripheral device receives the DMA ACK signal, it waits for all the operations on the local device completed, and both the FIFO and DRQ status refreshed. Then it invalidates the DMA Active signal.
- When the DMAC detects the falling edge of the DMA Active signal, it invalidates the corresponding DMA ACK signal, and restarts to detect the external request signals. If a valid request signal is detected, the next data transfer starts.

The following figure shows the workflow of the handshake mode.

Figure 2-14 Workflow of the DMAC Handshake Mode



2.7.3.8 Address Auto-Alignment

For the non-IO devices whose start address is not 32-byte-aligned, the DMAC will adjust the address to 32-byte-aligned through the burst transfer within 32 bytes. Adjusting address to 32-byte-aligned improves the DRAM access efficiency.

The following example shows how the DMAC adjusts the address: when the peripheral device of a DMA channel is a non-IO device whose start address is 0x86 (not 32-byte-aligned), the DMAC firstly uses a 26-byte burst transfer to align the address to 0xA0 (32-byte-aligned), and then transfers data by 64-byte burst (the maximum transfer amount that MBUS allows).

The IO devices do not support address alignment, so the bit width of IO devices must match the address offset; otherwise, the DMAC will ignore the inconsistency and directly transmit data of the corresponding bit width to the address.

The address of the DMA descriptor does not support the address auto-alignment. Make sure the address is word-aligned; otherwise the DMAC cannot identify the descriptor.

2.7.3.9 DMAC Clock Control

- The DMAC clock is synchronous with the AHB clock. Make sure that the DMAC gating bit of AHB clock is enabled before accessing the DMAC register.

- The reset input signal of the DMAC is asynchronous with AHB and is low valid by default. Make sure that the reset signal of the DMAC is de-asserted before accessing the DMA register.
- To avoid the indefinite state within registers, de-assert the reset signal first, and then open the gating bit of AHB.
- The DMAC supports Clock Auto Gating function to reduce power consumption, the system will automatically disable the DMAC clock in the DMAC idle state. Clock Auto Gating is enabled by default.

2.7.4 Programming Guidelines

2.7.4.1 Using DMAC Transfer Process

The DMAC transfer process is as follows.

Step 1 Configure DMAC_IRQ_CPU_EN_REG register and DMAC_IRQ_MCU_EN_REG register to control whether the channel interrupt siganl is transferred to CPU field and MCU field or not. If both of the two registers are not configured, the IRQ will be transferred to CPU field and MCU field by default.

Step 2 Request DMA channel, and check if the DMA channel is idle by checking if it is enabled. A disabled channel indicates it is idle, while an enabled channel indicates it is busy.

Step 3 Write the descriptor with 6 words into the memory. The descriptor must be word-aligned. For more details, refer to section 2.7.3.4 DMA Descriptor.

Step 4 Write the start address of the descriptor to DMAC_DESC_ADDR_REG.

Step 5 Enable the DMA channel, and write the corresponding channel to DMAC_EN_REG.

Step 6 The DMA obtains the descriptor information.

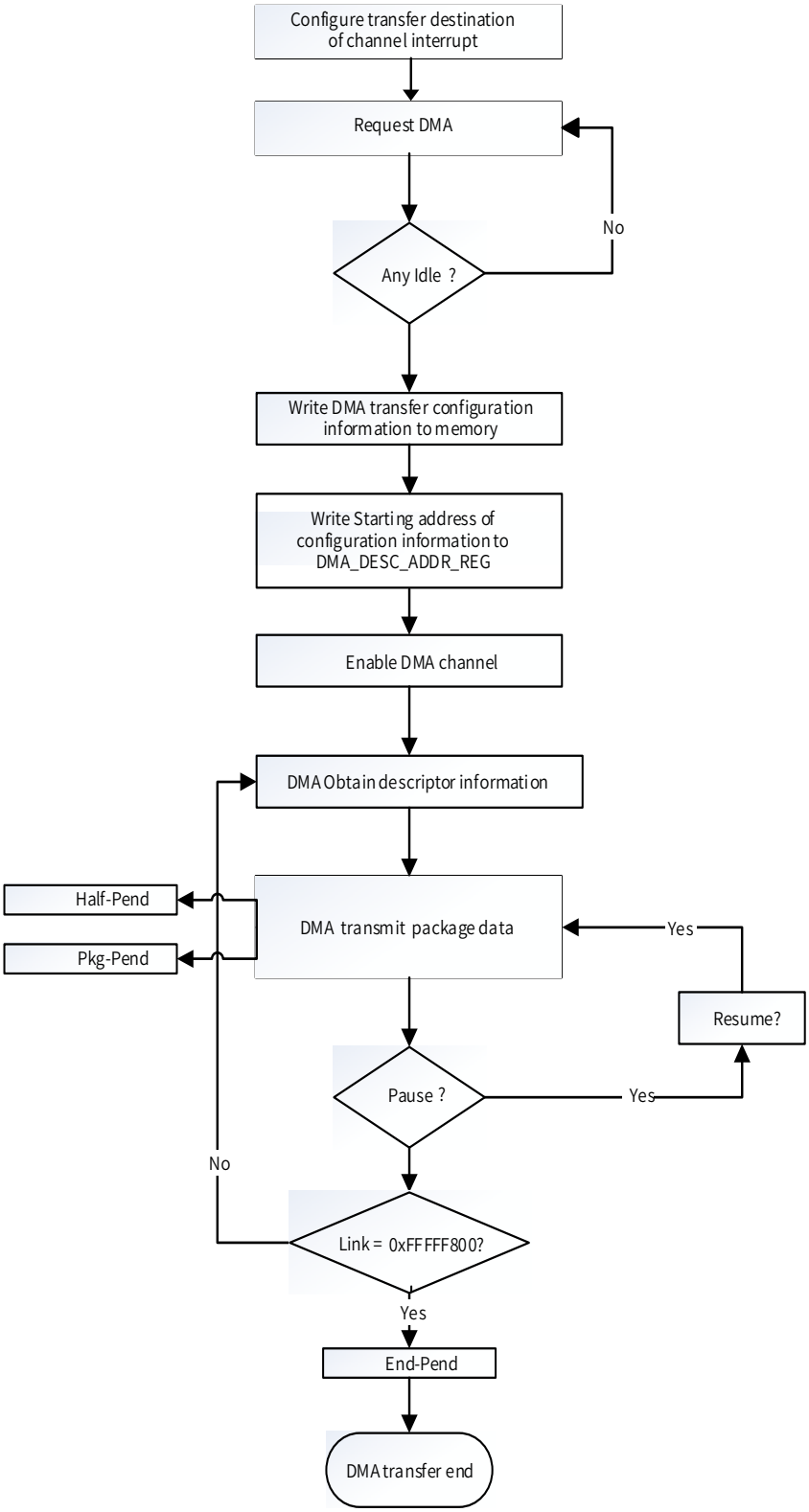
Step 7 Start to transmit a package. When half of the package is completed, the DMA sends a Half Package Transfer Interrupt; when a total package is completed, the DMA sends a Package End Transfer Interrupt. This interrupt status can be read by DMAC_IRQ_PEND_REG0.

Step 8 Set DMAC_PAU_REG to pause or resume the data transfer.

Step 9 After completing a total package transfer, the DMA decides to start the next package transfer or end the transfer by the link of the descriptor. If the link is 0xFFFFF800, the transfer ends; otherwise, the next package starts to transmit. When the transfer ends, the DMA sends a Queue End Transfer Interrupt.

Step 10 Disable the DMA channel.

Figure 2-15 DMAC Transfer Process



2.7.4.2 Processing DMAC Interrupt

Follow the steps below to process the DMAC interrupt:

Step 1 Enable interrupt: write the corresponding interrupt enable bit of DMAC_IRQ_EN_REG0. The system generates an interrupt when the corresponding condition is satisfied.

Step 2 After entering the interrupt process, write to clear the interrupt pending and execute the process of waiting for the interrupt.

Step 3 Resume the interrupt and continue to execute the interrupted process.

2.7.4.3 Configuring DMAC

To configure the DMAC, follow the guidelines below:

- Make sure the transfer bit width of IO devices is consistent with the offset of the start address.
- The MBUS protocol does not support the read operation of non-integer words. For the devices whose bit width is not word-aligned, after receiving the read command, they should resolve the read command according to their FIFO bit width instead of the command bit width, and ignore the redundant data caused by the inconsistency of the bit width.
- When the DMA transfer is paused, this is equivalent to invalid DRQ. Because there is a certain time delay between DMA transfer commands, the DMAC will not stop data transfer until the DMAC finishes processing the current command and the commands in Arbiter (at most 128 bytes' data).
- IOspeed and BMODE cannot be enabled at the same time.
- As DMA will transmit interrupts to DMAC_IRQ_CPU_EN_REG and DMAC_IRQ_MCU_EN_REG by default simultaneously, it should be configured which one receives interrupts before transmission, and the other one ought to be disabled.
- Timeout Programming Instruction
  - Similar to the register configuration of other channels, configure by descriptors and read through registers.
  - Enable BMODE before using TIMEOUT functions.
  - Before enabling TIMEOUT functions, if the source device configures the descriptor Config [7:6] as 0, set the DMAC_MODE_REG [4] as 1, and then start the DMA transfer.
  - After enabling the pause function of TIMEOUT, resume the data transfer by configuring the DMAC_PAU_REG.
  - After enabling the stop function of TIMEOUT, when TIMEOUT generates an interrupt, DMA will not generate the interrupt end signal of package and write-back. To restart the channel, configure the descriptors again.
  - After enabling the jump function of TIMEOUT, when TIMEOUT generates an interrupt, DMA will not generate the interrupt end signal of package and write-back but skip to the

next descriptor. Whether to start the TIMEOUT functions is decided by the descriptor being executed at that time. If there is no descriptor, transmission ends.

- The entered count number= the time to be counted * clock frequency (DMA clock frequency)/4096
- Take an example of 200MHz clock frequency for DMA. If the step size of timer is 20.48 us, and the maximum count is 511, the longest counting time will be the maximum count*step size=10.46 ms, and the shortest counting time will be 1*step size=20.48 us.

DMAC application example:

```
writel(0x00000000, mem_address + 0x00);  //Set configurations. The mem_address must be word-aligned.
writel (0x00001000, mem_address + 0x04);   // Set the start address for the source device.
writel (0x20000000, mem_address + 0x08);  //Set the start address for the destination device.
writel (0x00000020, mem_address + 0x0C);  // Set the data package size.
writel (0x00000000, mem_address + 0x10);  //Set the parameters.
writel (0xFFFFF800, mem_address + 0x14);  //Set the start address for the next descriptor.
writel (mem_address, 0x03002000+ 0x100 + 0x08); //Set the start address for the DMA channel0 descriptor.
do {
If (mem_address == readl (0x03002000+ 0x100 + 0x08));
break;
} while (1);                  //Make sure that the writing operation is valid.
writel (0x000000001, 0x03002000+ 0x100 + 0x00);      // Enable DMA channel0 transfer.
```

The DMAC supports increasing data package in transfer, pay attention to the following points:

- The 0xFFFFF800 value of DMAC_FDESC_ADDR_REG indicates that the DMA channel has got back the descriptor of the last package. The DMA channel will automatically stop the data transfer after transferring the current package.
- To add a package during the data transfer, check if the DMA channel has got back the descriptor of the last package. If yes, you cannot add any package in the current queue. Request another DMA channel with a new DRQ to transfer the package. Otherwise, you can add the package by modifying the DMAC_FDESC_ADDR_REG of the last package from 0xFFFFF800 to the start address of the to-be-added package.

  To ensure that the modification is valid, read the value of DMAC_FDESC_ADDR_REG after the modification. The value 0xFFFFF800 indicates the modification fails and the other values indicate you have successfully added packages to the queue.

  Another problem is, the system needs some time to process the modification, during which the DMA channel may get back the descriptor of the last package. You can read DMAC_CUR_SRC_REG and DMAC_CUR_DEST_REG and check if the increasing memory

address accords with the information of the added package. If yes, the package is added successfully; otherwise, the modification failed.

To ensure a higher rate of success, it is suggested that you add the package before the half package interrupt of the penultimate package.

2.7.5 Register List

Module Name	Base Address	Comments
DMAC	0x0300 2000	
MCU_DMAC	0x0712 1000	MCU_DMAC register is the same with DMAC

Register Name	Offset	Description
DMAC_IRQ_EN_REG0	0x0000	DMAC IRQ Enable Register 0
DMAC_IRQ_EN_REG1	0x0004	DMAC IRQ Enable Register 1
DMAC_IRQ_PEND_REG0	0x0010	DMAC IRQ Pending Status Register 0
DMAC_IRQ_PEND_REG1	0x0014	DMAC IRQ Pending Status Register 1
DMAC_SEC_REG	0x0020	DMAC Security Register
DMAC_AUTO_GATE_REG	0x0028	DMAC Auto Gating Register
DMAC_STA_REG	0x0030	DMAC Status Register
DMAC_IRQ_CPU_EN_REG	0x0034	DMAC IRQ Transfer to CPU Field Enable Register
DMAC_IRQ_MCU_EN_REG	0x0038	DMAC IRQ Transfer to MCU Field Enable Register
DMAC_EN_REG	0x0100+N*0x0040 (N=0-15)	DMAC Channel Enable Register
DMAC_PAU_REG	0x0104+N*0x0040 (N=0-15)	DMAC Channel Pause Register
DMAC_DESC_ADDR_REG	0x0108+N*0x0040 (N=0-15)	DMAC Channel Descriptor Address Register
DMAC_CFG_REG	0x010C+N*0x0040 (N=0-15)	DMAC Channel Configuration Register
DMAC_CUR_SRC_REG	0x0110+N*0x0040 (N=0-15)	DMAC Channel Current Source Address Register
DMAC_CUR_DEST_REG	0x0114+N*0x0040 (N=0-15)	DMAC Channel Current Destination Address Register
DMAC_BCNT_LEFT_REG	0x0118+N*0x0040 (N=0-15)	DMAC Channel Byte Counter Left Register
DMAC_PARA_REG	0x011C+N*0x0040 (N=0-15)	DMAC Channel Parameter Register
DMAC_MODE_REG	0x0128+N*0x0040 (N=0-15)	DMAC Mode Register

2.8 Generic Interrupt Controller (GIC)

2.8.1 Overview

The GIC-600 is a generic interrupt controller that handles interrupts from peripherals to the cores and interrupts between cores. The GIC-600 supports a distributed microarchitecture containing several individual blocks that are used to provide a flexible GIC implementation.

All the GIC-600 blocks communicate through fully credited AXI4-Stream interface channels. This means that the interface exerts transient backpressure only on their ic<xy>tready signals, enabling packets to be routed over any free-flowing interconnect. Channels can be routed over dedicated AXI4-Stream buses, or over any available free-flowing transport layer in the system. A channel is described as free-flowing if all transactions on that channel complete without a non-transient dependency on any other transaction.

The GIC-600 includes build scripts that can create appropriate levels of hierarchy for any particular configuration. In small configurations, the distribution can be hidden and internally optimized.

the GIC has the following features:

- Interrupt services and masking:
  - Supports the following interrupt types:
    - Up to 56000 LPIs. A peripheral generates these interrupts by writing to a memory-mapped register in the GIC-600.
    - Up to 960 SPIs in groups of 32.
    - Up to 16 PPIs that are independent for each core and can be programmed to support either edge triggered or level-sensitive interrupts.
    - Up to 16 SGIs that are generated through the GIC CPU interface of a core.
  - Up to 16 ITS modules that provide device isolation and ID translation for message-based interrupts and enable virtual machines to program devices directly.
  - Interrupt masking and prioritization with 32 priority levels, five bits per interrupt.
- Registers and programming
  - Flexible affinity routing, using the Multiprocessor Identification Register (MPIDR) addresses, including support for all four affinity levels.
  - Single ACE-Lite slave port on each chip for programming of all GIC Distributor (GICD) registers, GIC Interrupt Translation Service (GITS) registers, and GIC Redistributor (GICR) registers. Each ITS has an optional ACE-Lite slave port for programming the GITS_TRANSLATER register.
  - Coherent view of SPI register data across multiple chips.
- Security
  - A global Disable Security (DS) bit. This bit enables support for systems without security.
  - The following interrupt groups allow interrupts to target different exception levels:

- Group 0.
- Non-secure Group 1.
- Secure Group 1.

- Performance Monitoring Unit (PMU) counters with snapshot functionality.
- Error correction: ARMv8.2 Reliability Accessibility Serviceability (RAS) architecture-compliant error reporting for the following:
  - Software access errors
  - ITS command and translation errors
  - Error Correcting Code (ECC) errors

2.8.2 Functional Description

the following table describes the details of interrupt sources:

Table 2-15 Interrupt Source in CPUX Domain

Interrupt Number	Interrupt Source	Interrupt Vector	Description
0	SGI 0	0x0000	SGI 0 interrupt
1	SGI 1	0x0004	SGI 1 interrupt
2	SGI 2	0x0008	SGI 2 interrupt
3	SGI 3	0x000C	SGI 3 interrupt
4	SGI 4	0x0010	SGI 4 interrupt
5	SGI 5	0x0014	SGI 5 interrupt
6	SGI 6	0x0018	SGI 6 interrupt
7	SGI 7	0x001C	SGI 7 interrupt
8	SGI 8	0x0020	SGI 8 interrupt
9	SGI 9	0x0024	SGI 9 interrupt
10	SGI 10	0x0028	SGI 10 interrupt
11	SGI 11	0x002C	SGI 11 interrupt
12	SGI 12	0x0030	SGI 12 interrupt
13	SGI 13	0x0034	SGI 13 interrupt
14	SGI 14	0x0038	SGI 14 interrupt
15	SGI 15	0x003C	SGI 15 interrupt
16	PPI 0	0x0040	PPI 0 interrupt
17	PPI 1	0x0044	PPI 1 interrupt
18	PPI 2	0x0048	PPI 2 interrupt
19	PPI 3	0x004C	PPI 3 interrupt
20	PPI 4	0x0050	PPI 4 interrupt
21	PPI 5	0x0054	PPI 5 interrupt
22	PPI 6	0x0058	PPI 6 interrupt
23	PPI 7	0x005C	PPI 7 interrupt
24	PPI 8	0x0060	PPI 8 interrupt

Interrupt Number	Interrupt Source	Interrupt Vector	Description
25	PPI 9	0x0064	PPI 9 interrupt
26	PPI 10	0x0068	PPI 10 interrupt
27	PPI 11	0x006C	PPI 11 interrupt
28	PPI 12	0x0070	PPI 12 interrupt
29	PPI 13	0x0074	PPI 13 interrupt
30	PPI 14	0x0078	PPI 14 interrupt
31	PPI 15	0x007C	PPI 15 interrupt
32	CPUX_MSGBOX_R	0x0080	CPUX MSGBOX read IRQ for CPUX
33	CPUS _MSGBOX_W	0x0084	CPUS MSGBOX write IRQ for CPUX
34	UART0	0x0088	
35	UART1	0x008C	
36	UART2	0x0090	
37	UART3	0x0094	
38	UART4	0x0098	
39	UART5	0x009C	
40	UART6	0x00A0	
41	UART7	0x00A4	
42	TWI0	0x00A8	
43	TWI1	0x00AC	
44	TWI2	0x00B0	
45	TWI3	0x00B4	
46	TWI4	0x00B8	
47	TWI5	0x00BC	
48	SPI0	0x00C0	
49	SPI1	0x00C4	
50	SPI2	0x00C8	
51	PWM0	0x00CC	
52	SPIFC	0x00D0	
53		0x00D4	
54		0x00D8	
55		0x00DC	
56		0x00E0	
57		0x00E4	
58	CIR_TX	0x00E8	
59	CIR_RX	0x00EC	
60	LEDC	0x00F0	
61	USB0_DEVICE	0x00F4	
62	USB0_EHCI	0x00F8	
63	USB0_OHCI	0x00FC	
64	USB1_EHCI	0x0100	

Interrupt Number	Interrupt Source	Interrupt Vector	Description
65	USB1_OHCI	0x0104	
66		0x0108	
67	USB2/USB3	0x010C	
68		0x0110	
69		0x0114	
70	NDFC	0x0118	
71	THS0	0x011C	
72	SMHC0	0x0120	
73	SMHC1	0x0124	
74	SMHC2	0x0128	
75	NSI	0x012C	
76	SMC	0x0130	
77		0x0134	
78	GMAC0	0x0138	
79	GMAC1	0x013C	
80	CCU_FERR	0x0140	
81	AHB_HREADY_TIME_OUT	0x0144	
82	DMAC_CPUX_NS	0x0148	DMAC channel 0-15 non-secure interrupt
83	DMAC_ CPUX_S	0x014C	DMAC channel 0-15 secure interrupt
84	CE_NS	0x0150	
85	CE_S	0x0154	
86	SPINLOCK	0x0158	
87	CPUX_TIMER0	0x015C	
88	CPUX_TIMER1	0x0160	
89	CPUX_TIMER2	0x0164	
90	CPUX_TIMER3	0x0168	
91	CPUX_TIMER4	0x016C	
92	CPUX_TIMER5	0x0170	
93	GPADC_CTRL0	0x0174	
94	THS1	0x0178	
95	CPUX_WDT	0x017C	
96	GPADC_CTRL1	0x0180	
97	IOMMU	0x0184	
98	LRADC	0x0188	
99	GPIOA_NS	0x018C	
100	GPIOA_S	0x0190	
101	GPIOB_NS	0x0194	
102	GPIOB_S	0x0198	

Interrupt Number	Interrupt Source	Interrupt Vector	Description
103	GPIOC_NS	0x019C	
104	GPIOC_S	0x01A0	
105	GPIOD_NS	0x01A4	
106	GPIOD_S	0x01A8	
107	GPIOE_NS	0x01AC	
108	GPIOE_S	0x01B0	
109	GPIOF_NS	0x01B4	
110	GPIOF_S	0x01B8	
111	GPIOG_NS	0x01BC	
112	GPIOG_S	0x01C0	
113	GPIOH_NS	0x01C4	
114	GPIOH_S	0x01C8	
115	GPIOI_NS	0x01CC	
116	GPIOI_S	0x01D0	
117	GPIOJ_NS	0x01D4	
118	GPIOJ_S	0x01D8	
119	DE	0x01DC	
120	DI	0x01E0	
121	HDMI_PHY	0x01E4	
122	TCON_LCD0	0x01E8	
123	TCON_TV0	0x01EC	
124	TCON_LCD1	0x01F0	
125	HDMI	0x01F4	
126	DSI0	0x01F8	
127	DSI1	0x01FC	
128	TCON_TV1	0x0200	
129	TCON_LCD2	0x0204	
130	PCIE_EDMA[0]	0x0208	
131	PCIE_EDMA[1]	0x020C	
132	PCIE_EDMA[2]	0x0210	
133	PCIE_EDMA[0]	0x0214	
134	PCIE_EDMA[4]	0x0218	
135	PCIE_EDMA[5]	0x021C	
136	PCIE_EDMA[6]	0x0220	
137	PCIE_EDMA[7]	0x0224	
138	PCIE_SII	0x0228	
139	PCIE_MSI	0x022C	
140	PCIE_EDMA[8]	0x0230	
141	PCIE_EDMA[9]	0x0234	
142	PCIE_EDMA[10]	0x0238	
143	PCIE_EDMA[11]	0x023C	

Interrupt Number	Interrupt Source	Interrupt Vector	Description
144	PCIE_EDMA[12]	0x0240	
145	PCIE_EDMA[13]	0x0244	
146	PCIE_EDMA[14]	0x0248	
147	PCIE_EDMA[15]	0x024C	
148	GPU_EVENT	0x0250	
149	GPU_JOB	0x0254	
150	GPU_MMU	0x0258	
151	GPU	0x025C	
152	VE3	0x0260	
153	MEMC_DFS	0x0264	
154	CSI_DMA0	0x0268	
155	CSI_DMA1	0x026C	
156	CSI_DMA2	0x0270	
157	CSI_DMA3	0x0274	
158	CSI_VIPP0	0x0278	
159	CSI_VIPP1	0x027C	
160	CSI_VIPP2	0x0280	
161	CSI_VIPP3	0x0284	
162	CSI_PARSER0	0x0288	
163	CSI_PARSER1	0x028C	
164	CSI_PARSER2	0x0290	
165	CSI_ISP0	0x0294	
166	CSI_ISP1	0x0298	
167	CSI_ISP2	0x029C	
168	CSI_ISP3	0x02A0	
169	CSI_CMB	0x02A4	
170	CSI_TDM	0x02A8	
171	CSI_TOP_PKT	0x02AC	
172	GPIOK_NS	0x02B0	
173	GPIOK_S	0x02B4	
174	PWM1	0x02B8	
175	G2D	0x02BC	
176	EDP	0x02C0	
177		0x02C4	
178		0x02C8	
179	CSI_PARSER3	0x02CC	
180	NMI	0x02D0	
181	S_PPU	0x02D4	
182	S_PPU1	0x02D8	
183	S_TWD	0x02DC	
184	CPUS_WDT	0x02E0	

Interrupt Number	Interrupt Source	Interrupt Vector	Description
185	CPUS_TIMER0	0x02E4	
186	CPUS_TIMER1	0x02E8	
187	CPUS_TIMER2	0x02EC	
188	S_TWI2	0x02F0	
189	ALARM	0x02F4	
190	GPIOL_S	0x02F8	
191	GPIOL_NS	0x02FC	
192	GPIOM_S	0x0300	
193	GPIOM_NS	0x0304	
194	S_UART0	0x0308	
195	S_UART1	0x030C	
196	S_TWI0	0x0310	
197	S_TWI1	0x0314	
198		0x0318	
199	S_CIRRX	0x031C	
200	S_PWM0	0x0320	
201		0x0324	
202	AHBS_HREADY_TIME_OUT	0x0328	
203	CPUIDLE(PCK600_CPU)	0x032C	
204	S_SPI	0x0330	
205	S_SPINLOCK	0x0334	
206	CPUS_MSGBOX_CPUX	0x0338	
207		0x033C	
208		0x0340	
209		0x0344	
210		0x0348	
211		0x034C	
212		0x0350	
213		0x0354	
214		0x0358	
215		0x035C	
216		0x0360	
217	MCU_TIMER0	0x0364	
218	MCU_TIMER1	0x0368	
219	MCU_TIMER2	0x036C	
220	MCU_AHB0_TO	0x0370	
221	MCU_AHB1_TO	0x0374	
222	AUDIO CODEC	0x0378	
223	DMIC	0x037C	

Interrupt Number	Interrupt Source	Interrupt Vector	Description
224	I2S0	0x0380	
225	I2S1D	0x0384	
226	I2S2	0x0388	
227	I2S3	0x038C	
228	OWA	0x0390	
229	MCU_DMAC_NS	0x0394	
230	MCU_DMAC_S	0x0398	
231		0x039C	
232		0x03A0	
233	MCU_TIMER3	0x03A4	
234	MCU_TIMER4	0x03A8	
235	MCU_TIMER5	0x03AC	
236	RISCV_MSGBOX_CPUS	0x03B0	
237	RISCV_MSGBOX_CPUX	0x03B4	
238	RISCV_WDT	0x03B8	
239	MCU_PWM0	0x03BC	
240		0x03C0	
241		0x03C4	
242		0x03C8	
243		0x03CC	
244		0x03D0	
245		0x03D4	
246		0x03D8	
247		0x03DC	
248		0x03E0	
249		0x03E4	
250		0x03E8	
251		0x03EC	
252		0x03F0	
253		0x03F4	
254		0x03F8	
255		0x03FC	
CPUX Related			
256	nERRIRQ[0]	0x0400	L3 ECC error that causes potential data corruption or loss of coherency
257	nERRIRQ[1]	0x0404	Core0 ECC error that causes potential data corruption or loss of coherency

Interrupt Number	Interrupt Source	Interrupt Vector	Description
258	nERRIRQ[2]	0x0408	Core1 ECC error that causes potential data corruption or loss of coherency
259	nERRIRQ[3]	0x040C	Core2 ECC error that causes potential data corruption or loss of coherency
260	nERRIRQ[4]	0x0410	Core3 ECC error that causes potential data corruption or loss of coherency
261	nERRIRQ[5]	0x0414	Core4 ECC error that causes potential data corruption or loss of coherency
262	nERRIRQ[6]	0x0418	Core5 ECC error that causes potential data corruption or loss of coherency
263	nERRIRQ[7]	0x041C	Core6 ECC error that causes potential data corruption or loss of coherency
264	nERRIRQ[8]	0x0220	Core7 ECC error that causes potential data corruption or loss of coherency
265	nFAULTIRQ[0]	0x0424	L3 detected 1-bit or 2-bit ECC or Parity error in the RAMs
266	nFAULTIRQ[1]	0x0428	Core0 detected 1-bit or 2-bit ECC or Parity error in the RAMs
267	nFAULTIRQ[2]	0x042C	Core1 detected 1-bit or 2-bit ECC or Parity error in the RAMs
268	nFAULTIRQ[3]	0x0430	Core2 detected 1-bit or 2-bit ECC or Parity error in the RAMs

Interrupt Number	Interrupt Source	Interrupt Vector	Description
269	nFAULTIRQ[4]	0x0434	Core3 detected 1-bit or 2-bit ECC or Parity error in the RAMs
270	nFAULTIRQ[5]	0x0438	Core4 detected 1-bit or 2-bit ECC or Parity error in the RAMs
271	nFAULTIRQ[6]	0x043C	Core5 detected 1-bit or 2-bit ECC or Parity error in the RAMs
272	nFAULTIRQ[7]	0x0440	Core6 detected 1-bit or 2-bit ECC or Parity error in the RAMs
273	nFAULTIRQ[8]	0x0444	Core7 detected 1-bit or 2-bit ECC or Parity error in the RAMs
274	nCLUSTERPMUIRQ	0x0448	Cluster PMU interrupt request
275	GIC_FAULT_INT	0x044C	
276	GIC_ERR_INT	0x0450	
277	GIC_PMU_INT	0x0454	
278		0x0458	
279		0x045C	
280		0x0460	
281		0x0464	
282		0x0468	
283		0x046C	
284		0x0470	
285		0x0474	
286		0x0478	
287		0x047C	

2.8.3 Register List

Module Name	Base Address	Comments
GIC		
GIC600_MON_4	0x03400000	General interrupt controller(23*64KB)
GITS_TRANSLATER	0x03450000	

2.9 Core-Local Interrupt Controller (CLIC)

2.9.1 Overview

The Core-Local Interrupt Controller (CLIC) is only used for sampling, priority arbitration and distribution for external interrupt sources.

- supports RISC-V Core-Local Interrupt Controller Version 0.8 specification
- Up to 144 interrupt source sampling, supporting level interrupt and pulse interrupt
- 32 levels of interrupt priority
- 4 memory-mapped control registers for each interrupt
- Each attribute of this interrupt source can be configured by writing the control register of the corresponding interrupt source.

2.9.2 Functional Description

The following table describes the detail of interrupt sources.

Table 2-16 Interrupt Sources

Interrupt Number	Interrupt Source	Interrupt Vector	Description
0-15	Reserved	0x0000-0x003C	Not Used
16	RISCV_WDT	0x0040	RISCV watchdog interrupt
17	RISCV_MSGBOX_RISCV	0x0044	RISCV MSGBOX read IRQ
18		0x0048	
19		0x004C	
20		0x0050	
21		0x0054	
22		0x0058	
23		0x005C	
24		0x0060	
25	MCU_TIMER0	0x0064	
26	MCU_TIMER1	0x0068	
27	MCU_TIMER2_	0x006C	
28	AHB0_HREADY_TIME_OUT	0x0070	MCU AHB decoder0 timer out interrupt
29	AHB1_HREADY_TIME_OUT	0x0074	MCU AHB decoder1 timer out interrupt
30	AUDIO CODEC	0x0078	Audio Codec IRQ
31	DMIC	0x007C	DMIC IRQ
32	I2S0	0x0080	I2S0 IRQ
33	I2S1	0x0084	I2S1 IRQ
34	I2S2	0x0088	I2S2 IRQ

Interrupt Number	Interrupt Source	Interrupt Vector	Description
35	I2S3	0x008C	I2S3 IRQ
36	OWA	0x0090	OWA IRQ
37	MCU_DMAC_NS	0x0094	MCU DMAC channel IRQ non-secure to MCU
38	MCU_DMAC_S	0x0098	MCU DMAC channel IRQ secure to MCU
39		0x009C	
40		0x00A0	
41	MCU_TIMER3	0x00A4	
42	MCU_TIMER4	0x00A8	
43	MCU_TIMER5	0x00AC	
44	MCU_PWM0	0x00B0	MCU PWM0 Interrupt
45		0x00B4	
46		0x00B8	
47		0x00BC	
48		0x00C0	
49		0x00C4	
50		0x00C8	
51		0x00CC	
52	NMI	0x00D0	STBY NMI interrupt
53	S_PPU	0x00D4	STBY PCK600 Q-channel Interrupt
54	S_PPU1	0x00D8	/
55	S_TWD	0x00DC	STBY TWD interrupt
56	CPUS_WDT	0x00E0	STBY WDT interrupt
57	CPUS_TIMER0	0x00E4	CPUS_TIMER0 interrupt
58	CPUS_TIMER1	0x00E8	CPUS_TIMER1 interrupt
59	CPUS_TIMER2	0x00EC	CPUS_TIMER2 interrupt
60	S_TWI2	0x00F0	
61	ALARM	0x00F4	RTC ALARM0 interrupt
62	GPIOL_S	0x00F8	
63	GPIOL_NS	0x00FC	
64	GPIOM_S	0x0100	
65	GPIOM_NS	0x0104	
66	S_UART0	0x0108	S_UART0 interrupt
67	S_UART1	0x010C	
68	S_TWI0	0x0110	
69	S_TWI1	0x0114	
70		0x0118	
71	S_CIRRX	0x011C	S_CIRRX interrupt
72	S_PWM0	0x0120	S_PWM0 interrupt

Interrupt Number	Interrupt Source	Interrupt Vector	Description
73		0x0124	
74	AHBS_HREADY_TOUT	0x0128	STBY AHBS TIMEOUT interrupt
75	PCK600_CPU	0x012C	CPUIDLE(PCK600_CPU) interrupt
76	S_SPI	0x0130	S_SPI interrupt
77	S_SPINLOCK	0x0134	S_SPINLOCK interrupt
78	CPUS_MSGBOX_CPUX	0x0138	CPUS MSGBOX write IRQ for CPUX
79		0x013C	
80		0x0140	
81	CPUS_MSGBOX_RISCV	0x0144	CPUS MSGBOX write IRQ for RISCV
82		0x0148	
83	INT_SCRI[0]	0x014C	CPUX_MSGBOX_IRQ_RISCV
84	INT_SCRI[1]	0x0150	CPUX_MSGBOX_IRQ_CPUS
85	INT_SCRI[2]	0x0154	SPLOCK_IRQ
86		0x0158	
87	INT_SCRI[4]	0x015C	
88	INT_SCRI[5]	0x0160	DMAC_IRQ1_NS
89	INT_SCRI[6]	0x0164	DMAC_IRQ1_S
90	INT_SCRI[7]	0x0168	GIC IRQ 32-39 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG0[7:0] in S_INTC. Group GIC IRQ bit [33:32] are fixed to be masked.
91	INT_SCRI[8]	0x016C	GIC IRQ 40-47 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG0[15:8] in S_INTC.
92	INT_SCRI[9]	0x0170	GIC IRQ 48-55 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG0[23:16] in S_INTC.

Interrupt Number	Interrupt Source	Interrupt Vector	Description
93	INT_SCRI[10]	0x0174	GIC IRQ 56-63 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG0[31:24] in S_INTC.
94	INT_SCRI[11]	0x0178	GIC IRQ 64-71 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG1[7:0] in S_INTC.
95	INT_SCRI[12]	0x017C	GIC IRQ 72-79 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG1[15:8] in S_INTC.
96	INT_SCRI[13]	0x0180	GIC IRQ 80-87 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG1[23:16] in S_INTC. Group GIC IRQ bit [83:82] are fixed to be masked.
97	INT_SCRI[14]	0x0184	GIC IRQ 88-95 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG1[31:24] in S_INTC.
98	INT_SCRI[15]	0x0188	GIC IRQ 96-103 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG2[7:0] in S_INTC.

Interrupt Number	Interrupt Source	Interrupt Vector	Description
99	INT_SCRI[16]	0x018C	GIC IRQ 104-111 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG2[15:8] in S_INTC.
100	INT_SCRI[17]	0x0190	GIC IRQ 112-119 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG2[23:16] in S_INTC.
101	INT_SCRI[18]	0x0194	GIC IRQ 120-127 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG2[31:24] in S_INTC.
102	INT_SCRI[19]	0x0198	GIC IRQ 128-135 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG3[7:0] in S_INTC.
103	INT_SCRI[20]	0x019C	GIC IRQ 136-143 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG3[15:8] in S_INTC.
104	INT_SCRI[21]	0x01A0	GIC IRQ 144-151 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG3[23:16] in S_INTC.
105	INT_SCRI[22]	0x01A4	GIC IRQ 152-159 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG3[31:24] in S_INTC.

Interrupt Number	Interrupt Source	Interrupt Vector	Description
106	INT_SCRI[23]	0x01A8	GIC IRQ 160-167 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG4[7:0] in S_INTC.
107	INT_SCRI[24]	0x01AC	GIC IRQ 168-175 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG4[15:8] in S_INTC.
108	INT_SCRI[25]	0x01B0	GIC IRQ 176-183 Group Interrupt, the corresponding interrupt group mask register is GINTC_CONFIG_REG4[23:16] in S_INTC. Group GIC IRQ bit [183:180] are fixed to be masked.
109		0x01B4	
110		0x01B8	
111		0x01BC	
112		0x01C0	
113		0x01C4	
114		0x01C8	
115		0x01CC	
116		0x01D0	
117		0x01D4	
118		0x01D8	
119		0x01DC	
120		0x01E0	
121		0x01E4	
122		0x01E8	
123		0x01EC	
124		0x01F0	
125		0x01F4	
126		0x01F8	
127		0x01FC	
128		0x0200	
129		0x0204	

Interrupt Number	Interrupt Source	Interrupt Vector	Description
130		0x0208	
131		0x020C	
132		0x0210	
133		0x0214	
134		0x0218	
135		0x021C	
136		0x0220	
137		0x0224	
138		0x0228	
139		0x022C	
140		0x0230	
141		0x0234	
142		0x0238	
143		0x023C	
144		0x0240	

2.9.3 Register List

2.9.3.1 CLIC Register List

Module Name	Base Address
RISCV CLIC	0xE080_0000

Register Name	Offset	Description
CLIC_CFG_REG	0x0000	CLIC Configuration Register
CLIC_MINTTHRESH_REG	0x0008	CLIC MINTTHRESH Register
CLIC_INT_REGn	0x1000+n*4	CLIC Interrupt Register n

2.9.3.2 S_INTC Register List

Module Name	Base Address
S_INTC	0x0702_1000

Register Name	Offset	Description
GINTC_CONFIG_REG0	0x00C0	Group Interrupt Configuration Register 0
GINTC_CONFIG_REG1	0x00C4	Group Interrupt Configuration Register 1
GINTC_CONFIG_REG2	0x00C8	Group Interrupt Configuration Register 2
GINTC_CONFIG_REG3	0x00CC	Group Interrupt Configuration Register 3

2.10 I/O Memory Management Unit (IOMMU)

2.10.1 Overview

IOMMU (I/O Memory management unit) is designed for the specific memory requirements. It maps the virtual address (sent by the peripheral access memory) to the physical address. IOMMU allows multiple ways to manage the location of the physical address. It can use the physical address which has the potentially conflict mapping for different processes to allocate the memory space, and also allow application of non-continuous address mapping to the continuous virtual address space.

The IOMMU has the following features:

- Supports virtual address to physical address mapping by hardware implementation
- Supports ISP, CSI, VE_MBUS0, VE_MBUS1, G2D, DE, and DI parallel address mapping
- Supports ISP, CSI, VE_MBUS0, VE_MBUS1, G2D, DE, and DI bypass function independently
- Supports ISP, CSI, VE_MBUS0, VE_MBUS1, G2D, DE, and DI pre-fetch independently
- Supports ISP, CSI, VE_MBUS0, VE_MBUS1, G2D, DE, and DI interrupt handing mechanism independently
- Supports 2 levels TLB (level1 TLB for special using, and level2 TLB for sharing)
- Supports TLB Fully cleared and Partially disabled
- Supports trigger PTW behavior when TLB miss
- Supports checking the permission

2.10.2 Block Diagram

The internal module of IOMMU mainly includes the following parts.

Micro TLB: Level1 TLB, 64 words. Each peripheral corresponds to a TLB, which caching the level2 page table for the peripheral.

Macro TLB: Level2 TLB, 4K words. Each peripheral shares a level2 TLB for caching the level2 page table.

Pre-fetch Logic: Each Micro TLB corresponds to a Pre-Fetch Logic. By monitoring each master device to predict the bus access, the secondary page table corresponding to the address to be accessed can be read from the memory and stored in the secondary TLB to improve the hit ratio.

PTW Logic: Page Table Walk, mainly contains PTW Cache and PTW. The PTW Cache is used to store the level1 page table; when the virtual address is missed in the level1 and level2 TLB, it will trigger the PTW. PTW Cache can store 512 level1 page tables, that is, 512 words.

PMU: Performance Monitoring Unit, which is used to count the hit efficiency and the latency.

APB Interface: IOMMU register instantiation module. CPU reads and writes the IOMMU register by APB bus.

The following figure shows the internal block diagram of IOMMU.

Figure 2-16 IOMMU Block Diagram

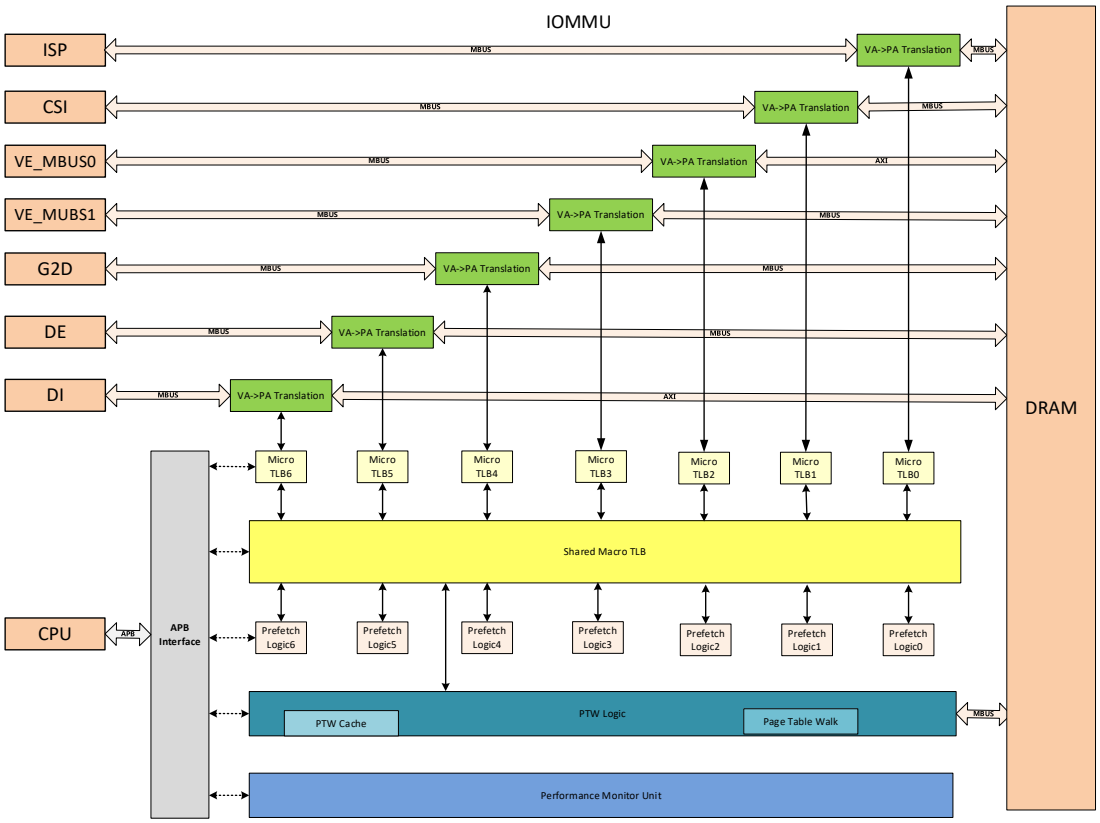


Table 2-17 Correspondence Relation between Master and Module

Master number	module
Master0	ISP
Master1	CSI
Master2	VE_MBUS0
Master3	VE_MBUS1
Master4	G2D
Master5	DE
Master6	DI

2.10.3 Function Descriptions

2.10.3.1 Initialization

- Release the IOMMU reset signal by writing 1 to the bit[31] of the IOMMU_RESET_REG (Offset: 0x0010);
- Write the base address of the first TLB to the IOMMU_TTB_REG (Offset: 0x0050);
- Set the IOMMU_INT_ENABLE_REG (Offset: 0x0100);
- Enable the IOMMU by configuring the IOMMU_ENABLE_REG (Offset: 0x0020) in the final.

2.10.3.2 Address Translation

In the process of address mapping, the peripheral virtual address [31:12] are retrieved in the Level1 TLB. When TLB is hit, the mapping is finished. Otherwise, they are retrieved in the Level2 TLB in the same way. If TLB is hit, the hit mapping will be written to the Level1 TLB, and hit in Level1 TLB. If Level1 and Level2 TLB are retrieved fail, the PTW will be triggered. After opening the peripheral bypass function by setting IOMMU_BYPASS_REG (Offset: 0x0030), IOMMU will not map the address typed by this peripheral, and it will output the virtual address as the physical address. The typical applications are as follows.

Micro TLB hit

Step 1 The master device sends a transfer command, and also sends the address to the corresponding Micro TLB to search the Level2 page table related to the virtual address;

Step 2 If Micro TLB is hit, it will return a Level2 page table containing the corresponding physical addresses and the permission Index;

Step 3 The address translation module converts the virtual address into the physical address, and checks the permissions at the same time. If it is passed, the transfer is completed.

Micro TLB miss, Macro TLB hit

Step 1 The master device sends a transfer command, and also sends the address to the corresponding Micro TLB to search the Level2 page table related to the virtual address;

Step 2 If Micro TLB is missed, continue to search Macro TLB;

Step 3 If Macro TLB is hit, it will return the Level2 page table to Micro TLB;

Step 4 Micro TLB receives this page table, puts it in Micro TLB (If this Micro TLB is full, the replace activities will happen), and sends the page table to the address translation module at the same time;

Step 5 The address translation module converts the virtual address into the physical address, and checks the permissions at the same time. If it is passed, the transfer is completed.

Micro TLB miss, Macro TLB miss, PTW Cache hit

Step 1 The master device sends a transfer command, and also sends the address to the corresponding Micro TLB to search the Level2 page table related to the virtual address;

Step 2 If Micro TLB is missed, continue to search Macro TLB;

Step 3 If Macro TLB is missed, send the request to the PTW to return the corresponding page table;

Step 4 PTW first accesses PTW Cache. If the required Level1 page table exists in the PTW Cache, send the page table to PTW logic;

Step 5 PTW logic returns the corresponding Level2 page table from the memory page table according to the Level1 page table, checks the effectiveness, and sends it to Macro TLB;

Step 6 Macro TLB stores the Level2 page table (the replace activities may happen), and returns the Level2 page table to Micro TLB;

Step 7 Micro TLB receives this page table, puts it in the Micro TLB (if this Micro TLB is full, the replace activities will happen), and sends the page table to the address translation module at the same time;

Step 8 The address translation module converts the virtual address into the physical address, and checks the permissions at the same time. If it is passed, the transfer is completed.

Micro TLB miss, Macro TLB miss, PTW Cache miss

Step 1 The master device sends a transfer command, and also sends the address to the corresponding Micro TLB to search the Level2 page table related to the virtual address;

Step 2 If Micro TLB is missed, continue to search Macro TLB;

Step 3 If Macro TLB is missed, send the request to the PTW to return the corresponding page table;

Step 4 PTW accesses PTW Cache, there is no necessary Level1 page table;

Step 5 PTW accesses the memory, gets the corresponding Level1 page table and stores it in the PTW Cache (the replace activities may happen);

Step 6 PTW logic returns the corresponding Level2 page table from the memory page table according to the Level1 page table, checks the effectiveness, and sends it to Macro TLB;

Step 7 Macro TLB stores the Level2 page table (the replace activities may happen), and returns the Level2 page table to Micro TLB;

Step 8 Micro TLB receives this page table, puts it in the Micro TLB (if this Micro TLB is full, the replace activities will happen), and sends the page table to the address translation module at the same time;

Step 9 The address translation module converts the virtual address into the physical address, and checks the permissions at the same time. If it is passed, the transfer is completed.

Permission error

Step 1 The permission checking is always performed during the process of translating the address;

Step 2 Once the permission checking makes mistake, the new access of the master suspends, but the access before this checking can be continued;

Step 3 Set the error status register;

Step 4 Trigger the interrupt.

Invalid Level1 page table

Step 1 The invalid Level1 page table is checked when PTW logic reads the new level page table from the memory;

Step 2 The PTW reads two sequential page table entries from the memory (64-bit data, a complete cache line), and stores them in the PTW cache;

Step 3 If the current page table is invalid, the error flag is set and the interrupt is triggered. The cache line needs to be invalidated.

 NOTE

- Invalid page table has two situations: the reading target page table from the memory is invalid, or the page table stored in PTW Cache with target page table is found to be invalid after using;
- If a page table is invalid, invalidate the total cache line (that is two page tables).

Invalid Level2 page table

Step 1 The invalid Level2 page table is checked when Macro TLB reads the new level page table from the memory;

Step 2 The Macro TLB reads two sequential page table entries from the memory (64-bit data, a complete cache line), and stores them in the Macro TLB;

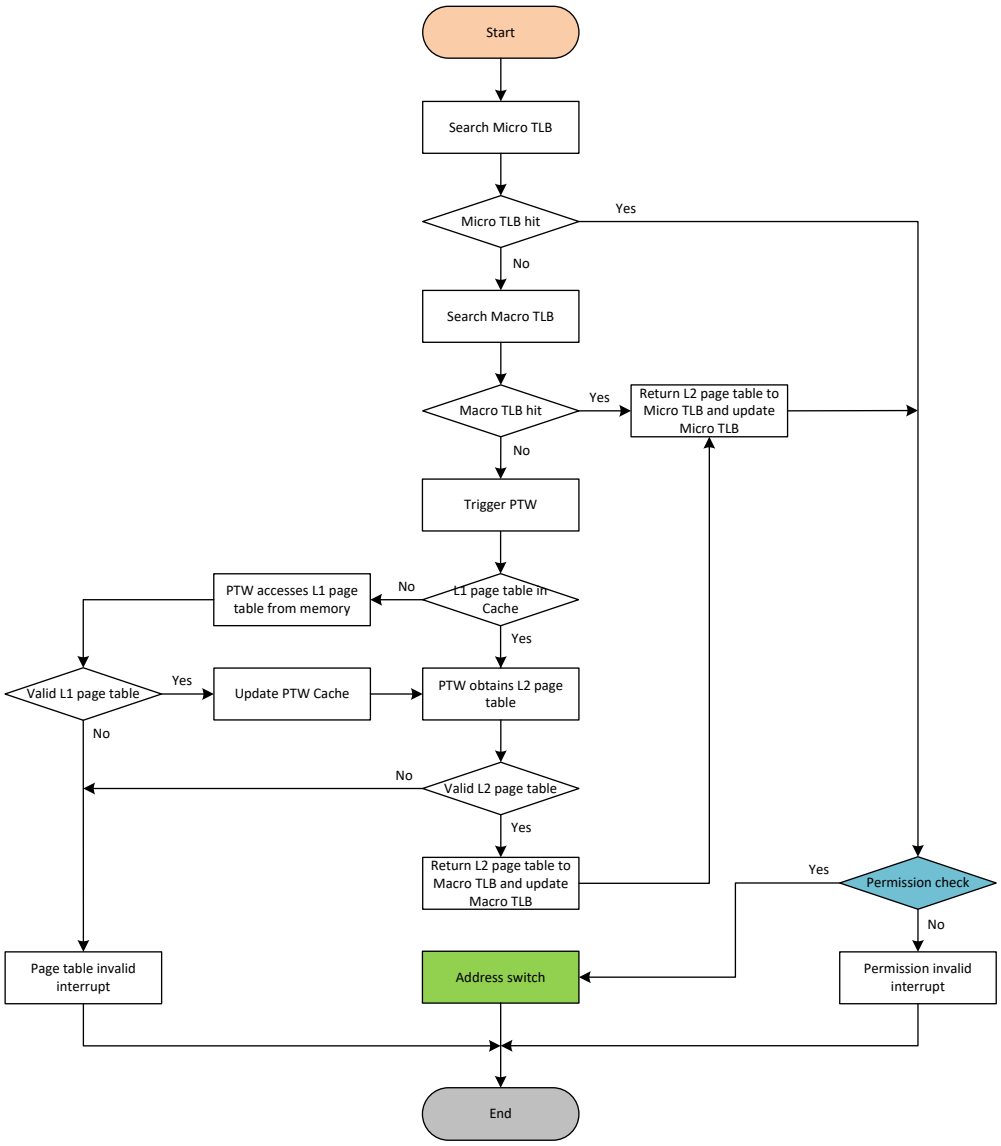
Step 3 If the current page table is invalid, the error flag is set and the interrupt is triggered. The cache line needs to be invalidated.

 NOTE

- Invalid page table has two situations: the reading target page table from the memory is invalid, or the page table stored in Macro TLB with target page table is found to be invalid after using;
- If a page table is invalid, invalidate the total cache line (that is two page tables).

The internal address translation process is shown in the following figure.

Figure 2-17 Internal Switch Process



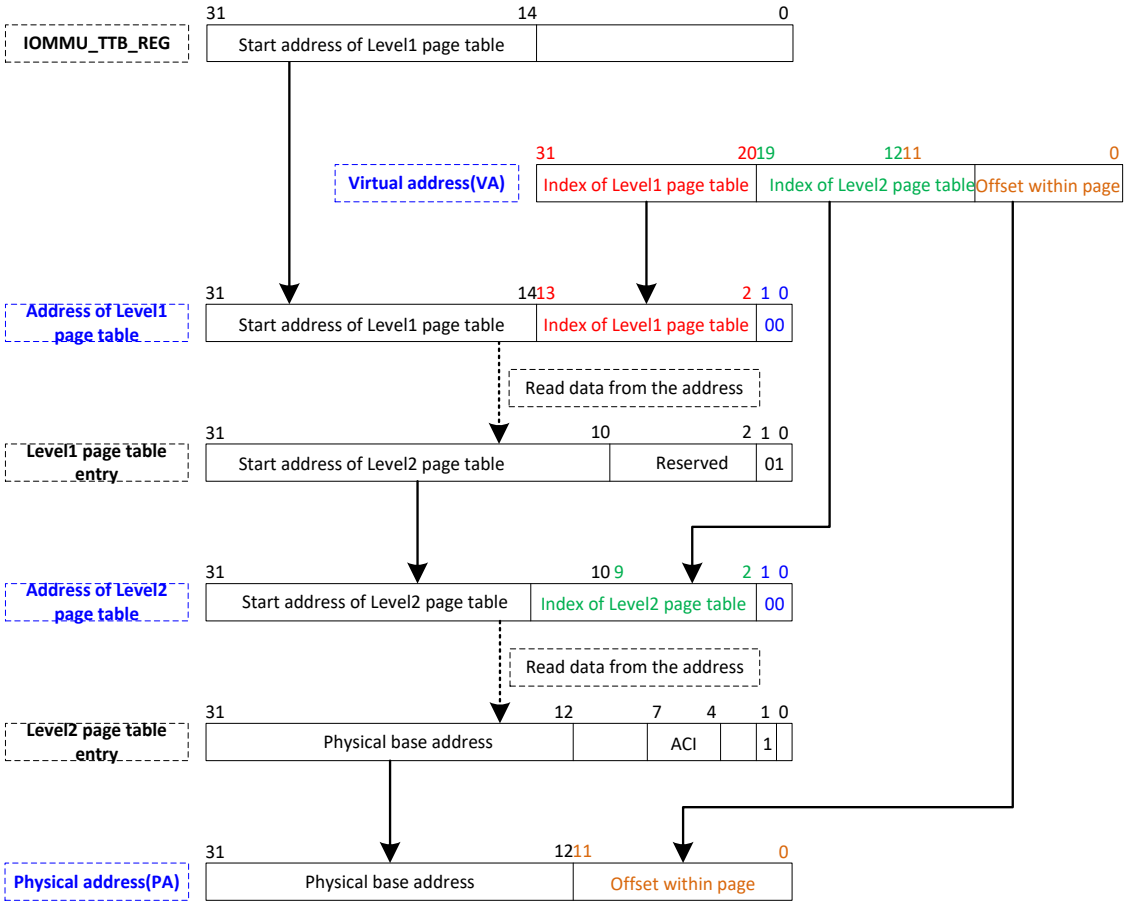
2.10.3.3 VA-PA Mapping

IOMMU page table is defined as the Level2 mapping. The first level is 1M address space mapping, the second level is 4K address space. This version does not support 1K, 16K and other page table sizes. IOMMU only supports a page table, the meaning is:

- All peripherals connected to IOMMU use the same virtual address space;
- The virtual address space of the peripherals can overlap;
- Different virtual addresses can map to the same physical address space;

Base address of this page table is defined by the software, and it needs 16 KB address alignment. The page table of the Level2 table item needs 1 KB address alignment. A complete VA-PA address translation process is shown in the following figure.

Figure 2-18 VA-PA Switch Process



2.10.3.4 Clearing and Invalidating TLB

When multi page table contents are refreshed or table address changes, all VA-PA mappings which have been cached in TLB will be invalid. You need to configure IOMMU_TLB_FLUSH_ENABLE_REG (Offset: 0x0080) to clear the TLB or PTW Cache according to the following steps:

Step 1 Suspend the access to TLB or Cache.

Step 2 Configure the corresponding Flush bit of IOMMU_TLB_FLUSH_ENABLE_REG (Offset: 0x0080).

Step 3 After the operation takes effect, the related peripherals can continue to send the new access memory operations.

When some page table is invalid or the mapping is incorrect, you can set the TLB Invalidation relevant register to invalidate TLB VA-PA mapping pairs. The invalid TLB supports the following two modes:

- Mode0

Step 1 Set IOMMU_TLB_IVLD_MODE_SEL_REG (Offset: 0x0084) to 0 and select mode0;

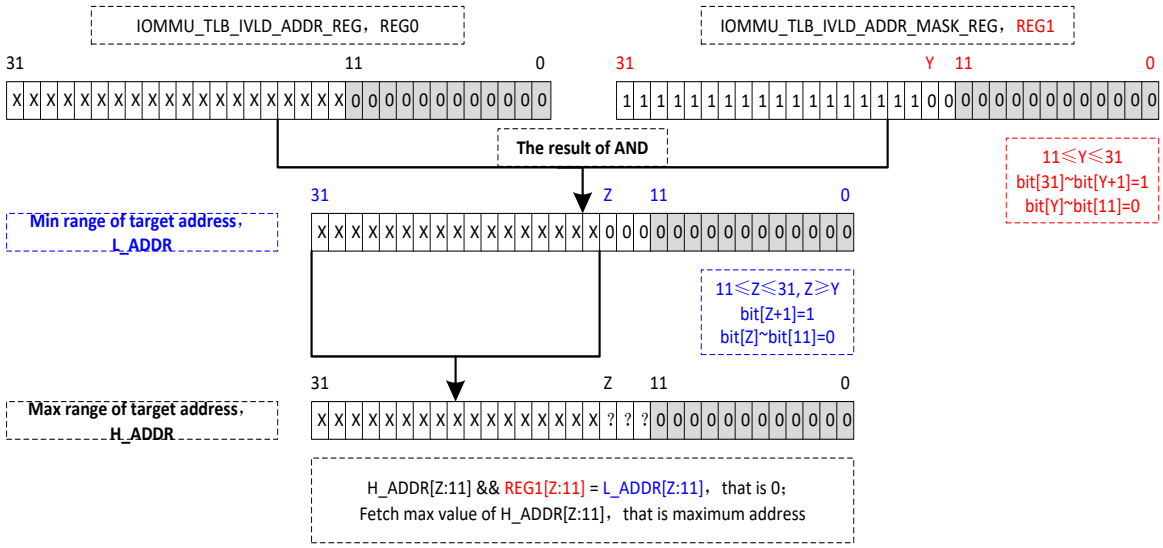
Step 2 Write the target address to IOMMU_TLB_IVLD_ADDR_REG (Offset: 0x0090);

Step 3 Set the configuration values to IOMMU_TLB_IVLD_ADDR_MASK_REG (Offset: 0x0094), the requirements are as follows:

- The value of IOMMU_TLB_IVLD_ADDR_MASK_REG (Offset: 0x0094) cannot be less than the IOMMU_TLB_IVLD_ADDR_REG (Offset: 0x0090).
- The higher bit of IOMMU_TLB_IVLD_ADDR_MASK_REG (Offset: 0x0094) must be continuous 1, the lower bit must be continuous 0. For example, 0xFFFFF000, 0xFFFFE000, 0xFFFFC000, 0xFFFF8000, and 0xFFFF0000 are legal values; while 0xFFFFD000, 0xFFFFB000, 0xFFFFA000, 0xFFFF9000, and 0xFFFF7000 are illegal values.

Step 4 Configure IOMMU_TLB_IVLD_ENABLE_REG (Offset: 0x0098) to enable the invalid operation. Among the way to determine the invalid address is to get the maximum valid bit and determine the target address range by the target address AND the mask address. The process is shown as follows.

Figure 2-19 Invalid TLB Address Range



The examples are shown below:

- When the value of IOMMU_TLB_IVLD_ADDR_MASK_REG (Offset: 0x0094) is 0xFFFFF000 by default, the result of AND is target address. That is, only the target address is invalid.
- When the value of IOMMU_TLB_IVLD_ADDR_MASK_REG (Offset: 0x0094) is 0xFFFF0000, the value of IOMMU_TLB_IVLD_ADDR_REG (0x0090) is 0xEEEE1000, then target address range is from 0xEEEE0000 to 0xEEEEF000.
- When the value of IOMMU_TLB_IVLD_ADDR_MASK_REG (Offset: 0x0094) is 0xFFFFC000, the value of IOMMU_TLB_IVLD_ADDR_REG (0x0090) is 0xEEEE8000, then target address range is from 0xEEEE8000 to 0xEEEEB000.
- When the value of IOMMU_TLB_IVLD_ADDR_MASK_REG (Offset: 0x0094) is 0xFFFF8000, the value of IOMMU_TLB_IVLD_ADDR_REG (0x0090) is 0xEEEEC000, then target address range is from 0xEEEE8000 to 0xEEEEF000.

- When the value of IOMMU_TLB_IVLD_ADDR_MASK_REG (Offset: 0x0094) is 0xFFFFC000, the value of IOMMU_TLB_IVLD_ADDR_REG (0x0090) is 0xEEEE0000, then target address range is from 0xEEEE0000 to 0xEEEE3000.

- Mode1

Step 1 Set IOMMU_TLB_IVLD_MODE_SEL_REG (Offset: 0x0084) to 1 and select mode1;

Step 2 Set the starting address and the ending address of the invalid TLB by IOMMU_TLB_IVLD_STA_ADDR_REG (Offset: 0x0088);

Step 3 Configure IOMMU_TLB_IVLD_ENABLE_REG (Offset: 0x0098) to enable the invalid operation, then the TLB invaliding operation can be completed.

2.10.3.5 Clearing and Invalidating PTW Cache

- Mode0

Step 1 Set IOMMU_PC_IVLD_MODE_SEL_REG (Offset: 0x009C) to 0 and select mode0.

Step 2 Invalid the IOMMU_PC_IVLD_ADDR_REG (Offset: 0x00A0), 1MB aligned.

Step 3 Configure IOMMU_PC_IVLD_ENABLE_REG (Offset: 0x00A8) to enable the invalid operation, then you can invalid one piece of CacheLine.

- Mode1

Step 1 Set IOMMU_PC_IVLD_MODE_SEL_REG (Offset: 0x009C) to 1 and select mode1.

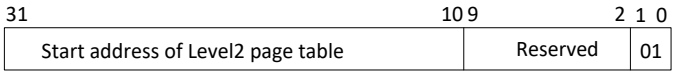
Step 2 Set the starting address and the ending address of the invalid TLB by IOMMU_PC_IVLD_STA_ADDR_REG (Offset: 0x00A4).

Step 3 Configure IOMMU_PC_IVLD_ENABLE_REG (Offset: 0x00A8) to enable the invalid operation, then you can invalid a period of sections.

2.10.3.6 Level1 Page Table

The format of Level1 page table is as follows.

Figure 2-20 Level1 Page Table Format



Bit [31:10]: Base address of Level2 page table;

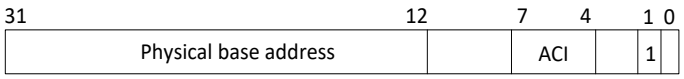
Bit [9:2]: Reserved;

Bit [1:0]: 01 is a valid page table; other values are fault;

2.10.3.7 Level2 Page Table

The format of Level2 page table is as follows.

Figure 2-21 Level2 Page Table Format



Bit [31:12]: Physical address of 4K address;

Bit [11:8]: Reserved;

Bit [7:4]: ACI, permission control index; correspond to permission control bit of IOMMU Domain Authority Control Register;

Bit [3:2]: Reserved;

Bit [1]: 1 is a valid page table; 0 is fault;

Bit [0]: Reserved

2.10.4 Programming Guidelines

2.10.4.1 Resetting IOMMU

Before the IOMMU module software reset operation, make sure IOMMU is never opened, or all bus operations are completed, or DRAM and peripherals already open the corresponding switch, to shield the influence of IOMMU reset.

2.10.4.2 Enabling IOMMU

Before opening the IOMMU address mapping function, IOMMU_TTB_REG (Offset: 0x0050) should be correctly configured, or all the masters are in the bypass state, or all the masters do not send the bus command.

2.10.4.3 Configuring TTB

Operating the register must close IOMMU address mapping function, namely IOMMU_ENABLE_REG (Offset: 0x0020) is 0; or Bypass function of all masters is set to 1, or no the state of transfer bus commands.

2.10.4.4 Clearing TTB

In the Flush operation, all TLB/Cache access will be suspended; but the operation entered the TLB will continue to complete before the Flush starts.

2.10.4.5 Reading/Writing VA Data

For the virtual address, read and write the corresponding physical address data to make sure whether IOMMU module address mapping function is normal. First, make sure to read or write, and then configure the target virtual address or write data, then start to read or write function, after the operation is finished, check if the results are as expected.

2.10.4.6 PMU Statistics

When PMU function is used for the first time, set IOMMU_PMU_ENABLE_REG (Offset: 0x0200) to enable statistics function; when reading the relevant Register, clear the enable bit of IOMMU_PMU_ENABLE_REG (Offset: 0x0200); when PMU function is used next time, first IOMMU_PMU_CLR_REG (Offset: 0x0210) is set, after counter is cleared, set the enable bit of IOMMU_PMU_ENABLE_REG (Offset: 0x0200).

Given a Level2 page table administers continuous 4KB address, if Micro TLB misses in continuous virtual address, a Level2 page table needs to be returned from Macro TLB to hit; but the hit number is not recorded in the Macro TLB hit and Micro TLB hit related register. So the true hit rate calculation is as follows:

```
Hit Rate = N1/M1 + (1-N1/M1)*N2/M2

N1: Micro TLB hit number
M1: Micro TLB access number
N2: Macro TLB hit number
M2: Macro TLB access number
```

2.10.5 Register List

Module Name	Base Address
IOMMU	0x0201_0000

Register Name	Offset	Description
IOMMU_RESET_REG	0x0010	IOMMU Reset Register
IOMMU_ENABLE_REG	0x0020	IOMMU Enable Register
IOMMU_BYPASS_REG	0x0030	IOMMU Bypass Register
IOMMU_AUTO_GATING_REG	0x0040	IOMMU Auto Gating Register
IOMMU_WBUF_CTRL_REG	0x0044	IOMMU Write Buffer Control Register
IOMMU_OOO_CTRL_REG	0x0048	IOMMU Out Of Order Control Register
IOMMU_4KB_BDY_PRT_CTRL_REG	0x004C	IOMMU 4KB Boundary Protect Control Register
IOMMU_TTB_REG	0x0050	IOMMU Translation Table Base Register
IOMMU_TLB_ENABLE_REG	0x0060	IOMMU TLB Enable Register
IOMMU_TLB_PREFETCH_REG	0x0070	IOMMU TLB Prefetch Register

2.11 Message Box (MSGBOX)

2.11.1 Overview

The Message Box (MSGBOX) provides interrupt communication mechanism for on-chip processor.

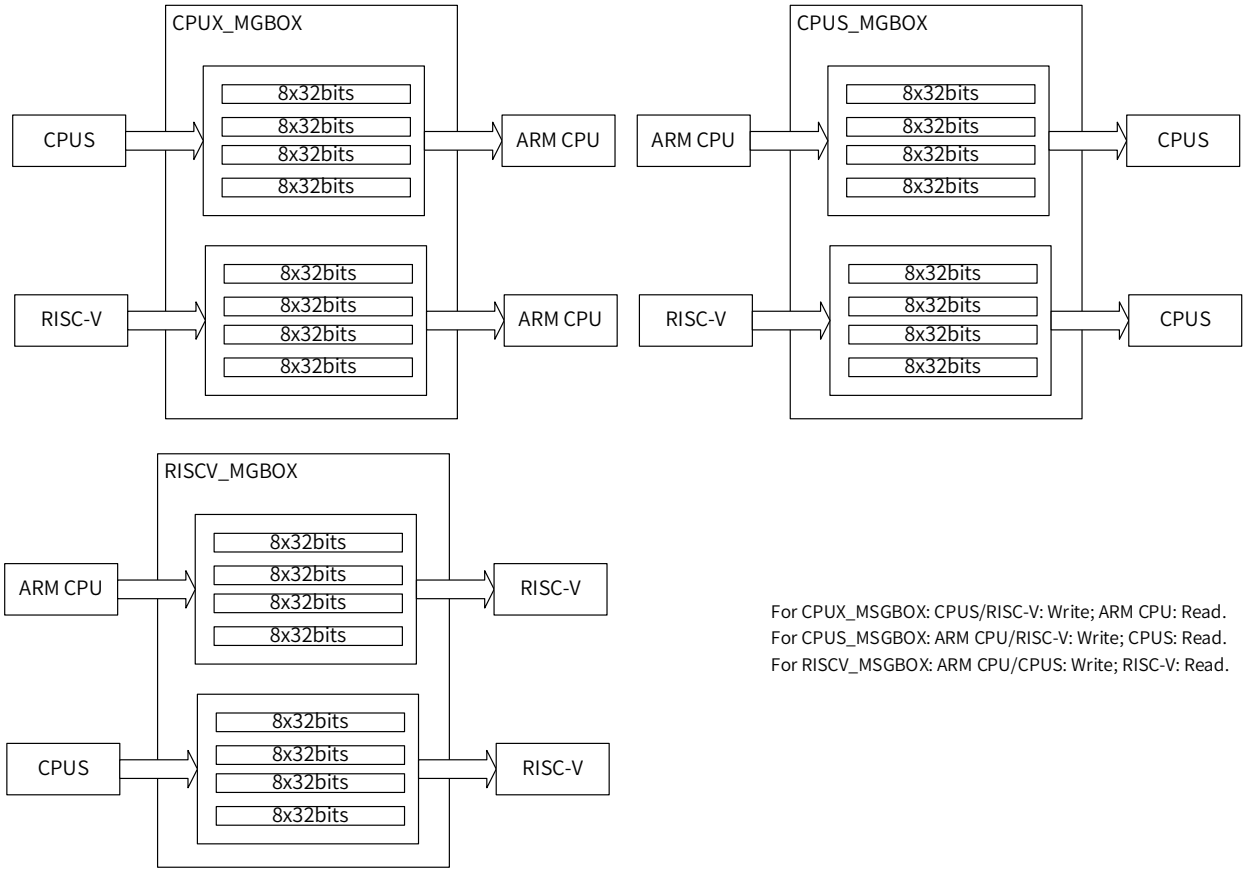
The MSGBOX has the following features:

- Supports communication between two CPUs through one way channels. Each CPU has one MSGBOX and can only read or write in one communication
  - CPUX_MSGBOX: CPUS/RISC-V write; ARM CPU read
  - CPUS_MSGBOX: ARM CPU/RISC-V write; CPUS read
  - RISCV_MSGBOX: ARM CPU/CPUS write; RISC-V read
- The channel between two CPU has 4 channels, and the FIFO depth of a channel is 8 x 32 bits
- Supports interrupts

2.11.2 Block Diagram

The following figure shows the block diagram of the message box.

Figure 2-22 Message Box Block Diagram



Each CPU has 4 channels. The two channels can be configured to be secure by software, the other two channels can be configured to be non-secure by software. The two secure channels or two

non-secure channels can be configured as one synchronous box (Sending a message requires a response) or one asynchronous box (Sending a message does not require a response).

2.11.3 Functional Description

2.11.3.1 Clock and Reset

The MSGBOX is mounted on AHB. Before accessing the MSGBOX registers, you need to de-assert the MSGBOX reset signal on AHB bus and then open the MSGBOX gating signal on AHB bus.

2.11.3.2 Transmitter/Receiver Mode

At the same channel, user1 is fixed as transmitter, user0 is fixed as receiver.

2.11.3.3 Typical Application

Several masters can build communication by configuring the MSGBOX. The communication parties have 4 channels. In a channel, the user1 is fixed as the transmitter and the user0 is fixed as the receiver. During the communication process, the current status can be judged through the interrupt or FIFO status.

2.11.3.4 Interrupt

Each channel can configure independently the interrupt enable bit, a read interrupt will be generated when the channel is empty, a write interrupt will be generated when the channel is non-full. For each CPU, all channels generate a read interrupt together, that is, if only a channel is non-full, the read interrupt will be generated, this channel can be obtained by querying the interrupt status register.

2.11.3.5 FIFO Status

When channel FIFO is non-full, the FIFO_NOT_AVA_FLAG is 0, at the moment the FIFO can be written.

When channel FIFO is full, the FIFO_NOT_AVA_FLAG is 1, at the moment if FIFO is written again, the first data of FIFO can be covered.

See MSGBOX_FIFO_STATUS_REG for FIFO status.

2.11.4 Programming Guidelines

2.11.4.1 Checking the Transfer Status via the Interrupt

Follow the steps below to check the transfer status:

Step 1 Enable the interrupt for the channel: Configure the interrupt enable bits of transmitter/receiver through MSGBOX_WR_IRQ_EN_REG/MSGBOX_RD_IRQ_EN_REG. (user0: RX interrupt enable; user1: TX interrupt enable)

Step 2 Check the IRQ status of the corresponding queue through MSGBOX_WR_IRQ_STATUS_REG/MSGBOX_RD_IRQ_STATUS_REG.

- If the FIFO is not full, the channel generates a transmission interrupt to remind the transmitter to transmit data. Write data to the FIFO in the interrupt handler, then clear the pending bit of the transmitter in MSGBOX_WR_IRQ_STATUS_REG and the enable bit of the transmitter in MSGBOX_WR_IRQ_EN_REG.
- If the FIFO has new data, the channel generates a reception interrupt to remind the receiver to receive data. Read data from the FIFO in interrupt handler, then clear the pending bit of the receiver in MSGBOX_RD_IRQ_STATUS_REG and the enable bit of the receiver in MSGBOX_RD_IRQ_EN_REG.

2.11.4.2 Checking the Transfer Status via the FIFO

Follow the steps below to check the FIFO status of the corresponding queue:

- If the FIFO is not full, the transmitter fills the FIFO to 8*32 bits.
- If the FIFO is full, the receiver reads the FIFO data, and reads MSGBOX_FIFO_STATUS_REG to acquire the current FIFO data amount and the FIFO data amount before reading, which means no data is dropped.

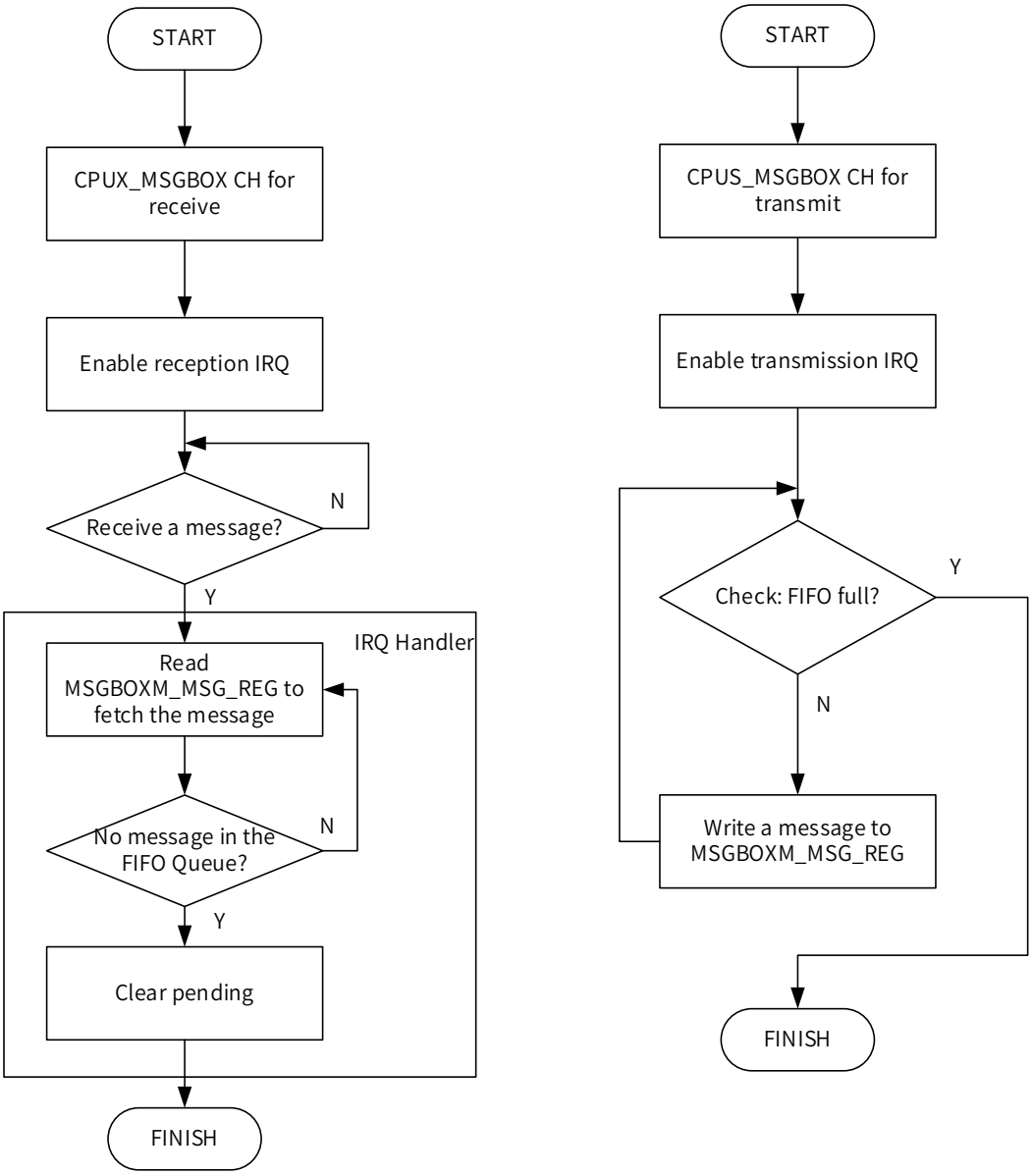
2.11.4.3 Transmitting/Receiving Message

The following figure shows the communication process between CPUX_MSGBOX and CPUS_MSGBOX.

CPUX_MSGBOX: Receiving message

CPUS_MSGBOX: Transmitting message

Figure 2-23 The Communication Process between CPUX_MSGBOX and CPUS_MSGBOX



2.11.5 Register List

Module Name	Base Address
CPUX_MSGBOX	0x0300 3000
CPUS_MSGBOX	0x0709 4000
RISCV_MSGBOX	0x0713 6000

Parameter	Description	Value
N	The CPU numbers that communicates with the current CPU	0 or 2
P	The channel numbers between two communication CPU	0-3

2.12 Power Reset Clock Management (PRCM)

2.12.1 Overview

The Power Reset Clock Management (PRCM) module is one of the most import design aspects in this system. It provides a versatile supporting multiple power-management techniques. And it also manages the gating and enabling of the clocks to the device modules.

The system-level reset management provides correct reset routing and sequencing when one or more devices are stacked together in the same package. The device-level reset management provides reset routing to relevant devices, such as CPUS_TIMER, S_UART and so on.

The PRCM has the following features:

- Two PRCMs in CPUS domain: STBY_PRCM and MCU_PRCM
- 1 PLL
- CPUS Clock Configuration
- APBS Clock Configuration
- CPUS Module Clock Configuration
- CPUS Module BUS Gating and Reset
- RAM configure Control for STBY_PRCM

NOTE

- There are 15 PLLs in A527. 10 PLLs in CCU, 4 PLLs in CPUX system, and 1 PLL in MCU_PRCM.
- PRCM describes module clocks in CPUS domain.
- For clock description of CPUX system, please refer to section 2.2.3.2 CPU PLL Distribution and Clock Sources.
- For module clocks in CPUX domain (excluding the clocks of CPUX system.), please refer to section 2.6 Clock Controller Unit (CCU).

2.12.2 Functional Description

2.12.2.1 System Bus Tree

The following figures show the diagram of the System Bus Tree.

Figure 2-24 System Bus Tree of STBY_PRCM

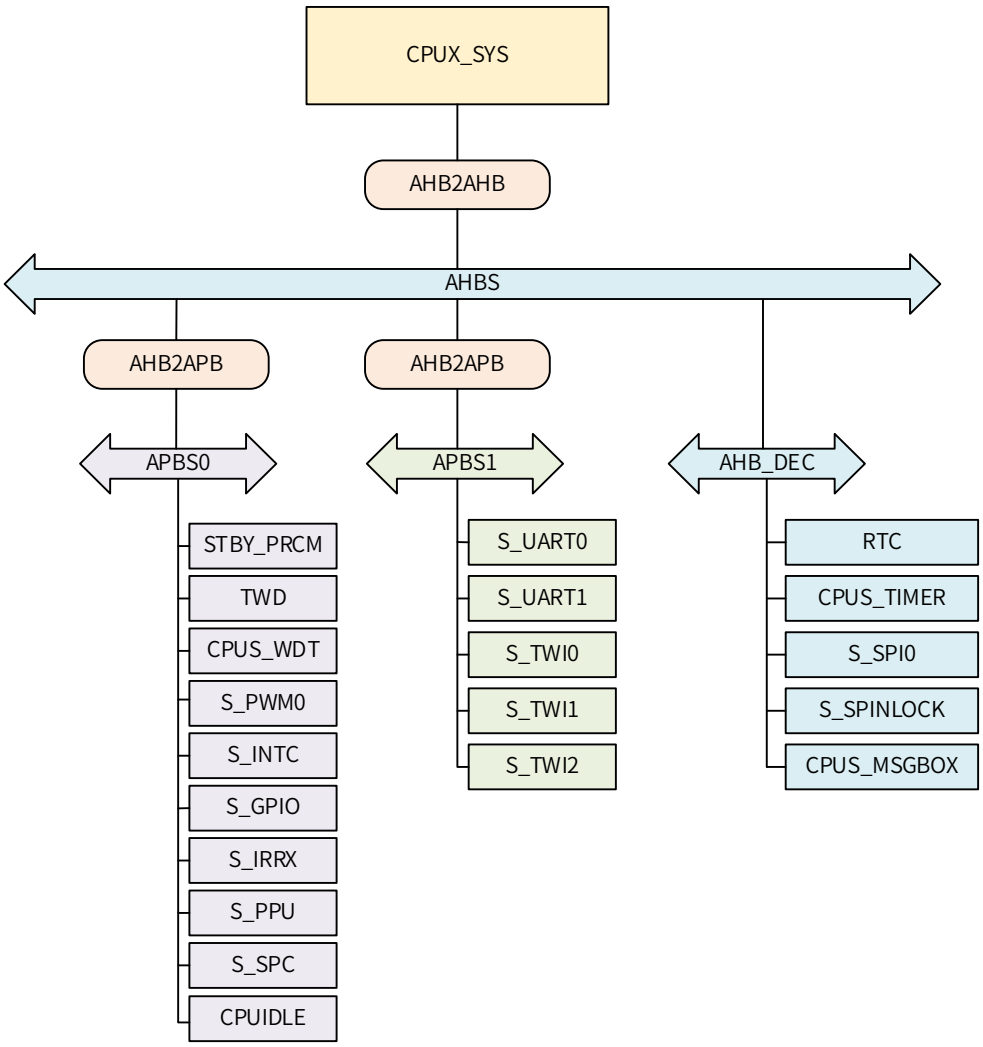
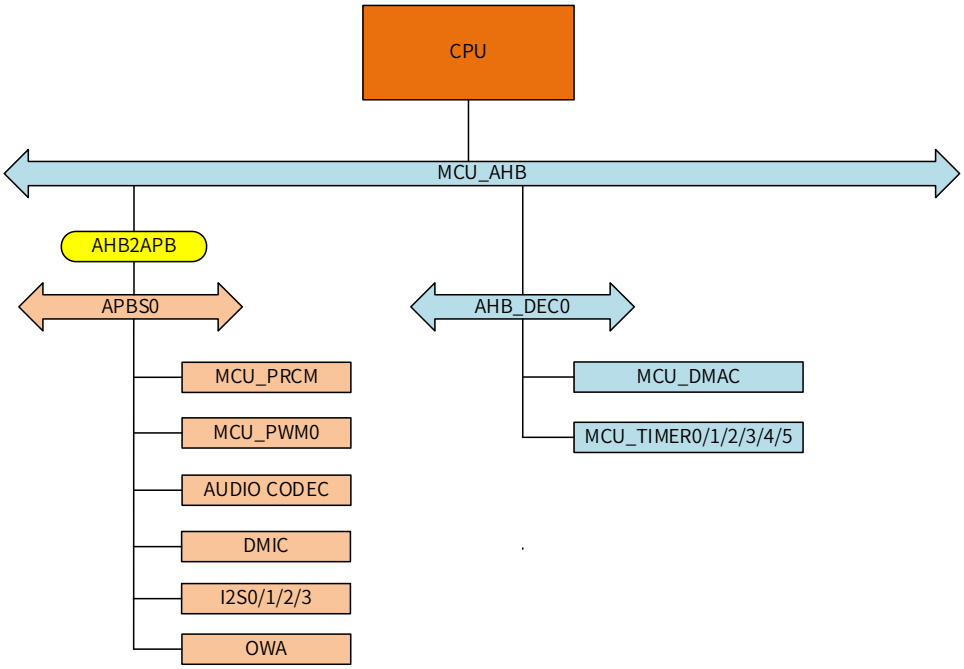


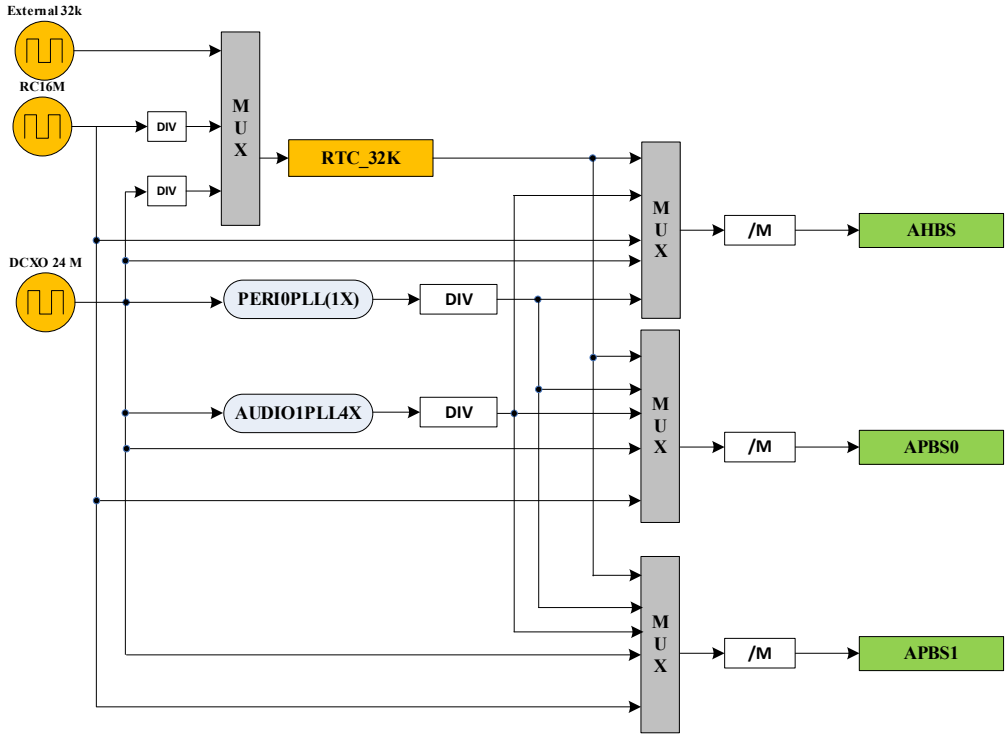
Figure 2-25 System Bus Tree of MCU_PRCM



2.12.2.2 Bus Clock Tree

The following figures show a block diagram of the clock tree Diagram in CPUS domain.

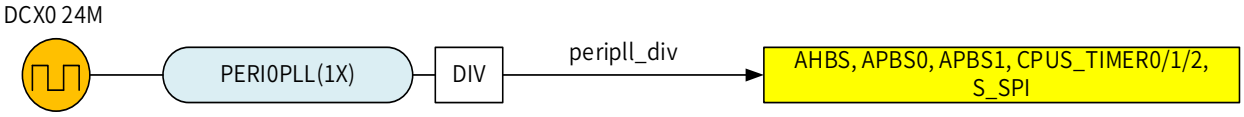
Figure 2-26 Bus Clock Tree



2.12.2.3 PLL Distribution

The following figures show the block diagram of the PLL distribution.

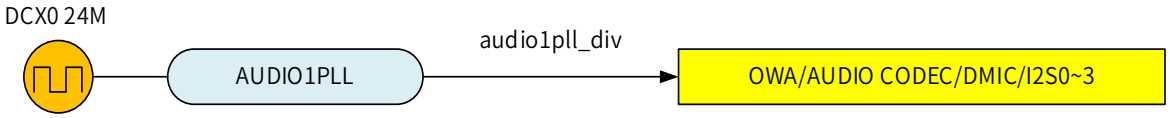
Figure 2-27 PLL Distribution of STBY_PRCM



NOTE

PERI0PLL(1X) is PERI0_600M = PERI0PLL2X/2. For detailed information of PLL_PERI0, see section 2.6.3.3 PLL Features.

Figure 2-28 PLL Distribution of MCU_PRCM



2.12.2.4 PLL Features

The following table shows the PLL features.

Table 2-18 PLL Features

PLL	Stable Operating Frequency	Actual Operating Frequency	Spread Spectrum	Linear FM	Pk-Pk	Lock Time
PLL_AUDIO1	1.52 GHz-3.1GHz	Integer mode: 1/2x: 1.536 GHz 1/5x: 614.4 MHz Decimal mode: 1/2x: 1.1179648 GHz 1/5x: 471.8592 MHz	Yes	No	< 200ps	500us

2.12.3 Programming Guidelines

2.12.3.1 Enabling the PLL

Follow the steps below to enable the PLL:

Step 1 Configure the N, M, and P factors of the PLL control register.

Step 2 Write 1 to the PLL_EN bit (bit [31]) and the PLL_LDO_EN bit (bit [30]) of the PLL control register, write 0 to the PLL_OUTPUT_GATE bit (bit [27]) of the PLL control register.

Step 3 Write 1 to the LOCK_ENABLE bit (bit [29]) of the PLL control register.

Step 4 Wait for the status of the Lock to change to 1.

Step 5 Delay 20 us.

Step 6 Write the PLL_OUTPUT_GATE bit (bit [27]) of the PLL control register to 1 and then the PLL will be available.

2.12.3.2 Configuring the Frequency of PLL_AUDIO1

The frequency configuration formula of PLL_AUDIO1:

PLL_AUDIO1 = 24 MHz*N/M0/M1/P

PLL_AUDIO1 does not support dynamic adjustment because changing any parameter of N, M0, M1, and P will affect the normal work of PLL, and the PLL will need to be relocked.

Generally, PLL_AUDIO1 only needs two frequency points: 24.576*4 MHz or 22.5792*4 MHz. For these two frequencies, there are usually special recommended matching factors. To implement the desired frequency point of PLL_AUDIO1, you need to use the decimal frequency-division function, so follow the steps below:

Step 1 Configure the N, M0, M1 and P factors.

Step 2 Write 1 to the PLL_SDM_EN bit (bit [24]) of PLL_AUDIO1_CTRL register.

Step 3 Configure PLL_AUDIO1_PAT0_CTRL register to enable the digital spread spectrum.

Step 4 Write 0 and then write 1 to the LOCK ENABLE bit (bit [29]) of PLL_AUDIO1_CTRL register.

Step 5 Write 1 to the LOCK bit (bit [28]) of PLL_ AUDIO1_CTRL register.

NOTE

When the P factor of PLL_AUDIO1 is an odd number, the clock output is an unequal-duty-cycle signal.

2.12.3.3 Disabling the PLL

Follow the steps below to disable the PLL:

Step 1 Write 0 to the PLL_EN bit (bit [31]) and the PLL_LDO_EN bit (bit [30]) of the PLL control register.

Step 2 Write 0 to the LOCK_ENABLE bit (bit [29]) of the PLL control register.

2.12.3.4 Implementing Spread Spectrum

The spread spectrum technology is to convert a narrowband signal into a wideband signal. It helps to reduce the effect of electromagnetic interference (EMI) associated with the fundamental frequency of the signal.

For the general PLL frequency, the calculation formula is as follows:

$$f = \frac{N+1+X}{P\cdot(M0+1)\cdot(M1+1)}\cdot 24MHz,\ 0<X<1$$

Where,

P is the frequency division factor of module or PLL;

M0 is the post-frequency division factor of PLL;

M1 is the pre-frequency division factor of PLL;

N is the frequency doubling factor of PLL;

X is the amplitude coefficient of the spread spectrum.

The parameters N, P, M1, and M0 are for the frequency division.

When M1 = 0, M0 = 0, and P = 1 (no frequency division), the calculation formula of PLL frequency can be simplified as follows:

$$f = (N+1+X)\cdot 24MHz,\ 0<X<1$$

$$[f_1, f_2] = (N+1+[X_1, X_2])\cdot 24MHz$$

$$SDM\_BOT = 2^{17}\cdot X_1$$

$$\mathrm{WAVE\_STEP} = 2^{17}\cdot(X_2 - X_1)/(24\ \mathrm{MHz}/PREQ)\cdot 2$$

Where, SDM_BOT and WAVE_STEP are bits of the PLL pattern control register, and PREQ is the frequency of the spread spectrum.

Follow the steps below to implement the spread spectrum:

Step 1 Configure the control register of the corresponding PLL

a) Calculate the factor N and decimal value X according to the PLL frequency and PLL frequency formula. Refer to the control register of the corresponding PLL (named PLL_xxx_CTRL_REG, where xxx is the module name) in 3.3.6 Register Description for the corresponding PLL frequency formula.

b) Write M0, M1, N, and PLL frequency to the PLL control register.

c) Configure the PLL_SDM_EN bit (bit [24]) of the PLL control register to 1 to enable the spread spectrum function.

Step 2 Configure the pattern control register of the corresponding PLL

a) Calculate the SDM_BOT and WAVE_STEP of the pattern control register according to decimal value X and spread spectrum frequency (the bit [18:17] of the PLL pattern register)

b) Configure the spread spectrum mode (SPR_FREQ_MODE) to 2 or 3.

c) If the PLL_INPUT_DIV2 of the PLL control register is 1, configure the spread spectrum clock source select bit (SDM_CLK_SEL) of the PLL pattern control register to 1. Otherwise, configure SDM_CLK_SEL to the default value 0.

d) Write SDM_BOT, WAVE_STEP, PREQ, SPR_FREQ_MODE, and SDM_CLK_SEL to the PLL pattern control register, and configure the SIG_DELT_PAT_EN bit (bit [31]) of this register to 1.

Step 3 Delay 20 us

2.12.3.5 Configuring Bus Clock

The bus clock supports dynamic switching, but the process of switching needs to follow the following two rules.

- From a higher frequency to a lower frequency: switch the clock source first, and then set the frequency division factor;
- From a lower frequency to a higher frequency: configure the frequency division factor first, and then switch the clock source.

The typical bus frequency for each bus in STBY_PRCM and MCU_PRCM is as follows:

- AHBS: 200 MHz
- APBS0: 100 MHz
- APBS1: 24 MHz

2.12.3.6 Configuring Module Clock

For the Bus Gating Reset register of a module, the reset bit is de-asserted first, and then the clock gating bit is enabled to avoid potential problems caused by the asynchronous release of the reset signal.

For all module clocks except the DDR clock, configure the clock source and frequency division factor first, and then release the clock gating (that is, set to 1). For the configuration order of the clock source and frequency division factor, follow the rules below:

- With the increasing of the clock source frequency, configure the frequency division factor before the clock source.
- With the decreasing of the clock source frequency, configure the clock source before the frequency division factor.

2.13 RTC

2.13.1 Overview

The Real Time Clock (RTC) is used to implement the time counter and the timing wakeup functions. The RTC can display the year, month, day, week, hour, minute, second in real time. The RTC has the independent power to continue to work in system power-off.

The RTC has the following features:

- Provides a 16-bit counter for counting day, 5-bit counter for counting hour, 6-bit counter for counting minute, 6-bit counter for counting second
- Timer frequency is 1 kHz
- Configurable initial value by software anytime
- Supports timing alarm, and generates interrupt and wakeup the external devices
- Supports fanout function of internal 32K clock
- 8 general purpose registers for storing the power-off information
- Multiple special registers for recording the BROM information

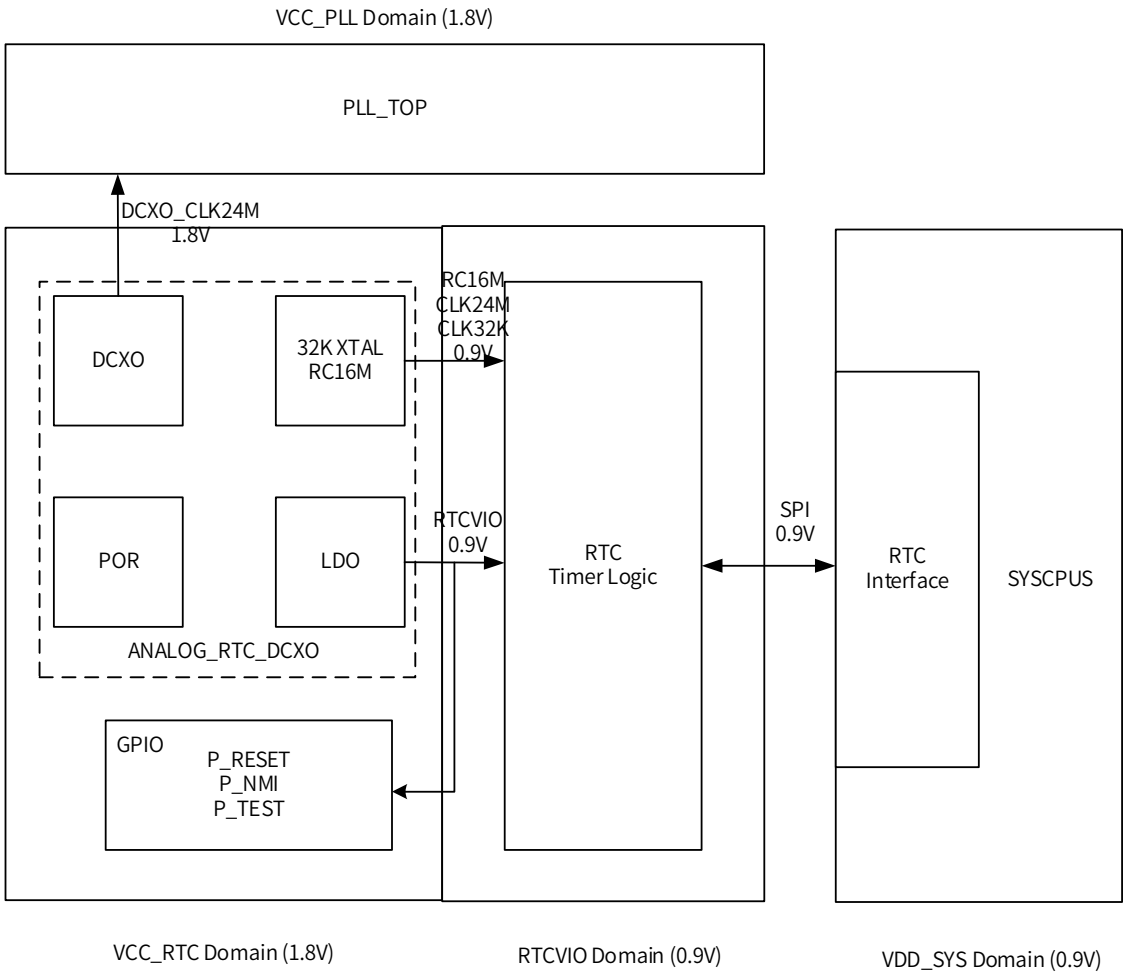
 NOTE

The register configuration of RTC is AHB bus, it only can support word operation, not byte operation and half-word operation.

2.13.2 Block Diagram

The following figure shows the block diagram of the RTC.

Figure 2-29 RTC Block Diagram



2.13.3 Functional Descriptions

2.13.3.1 External Signals

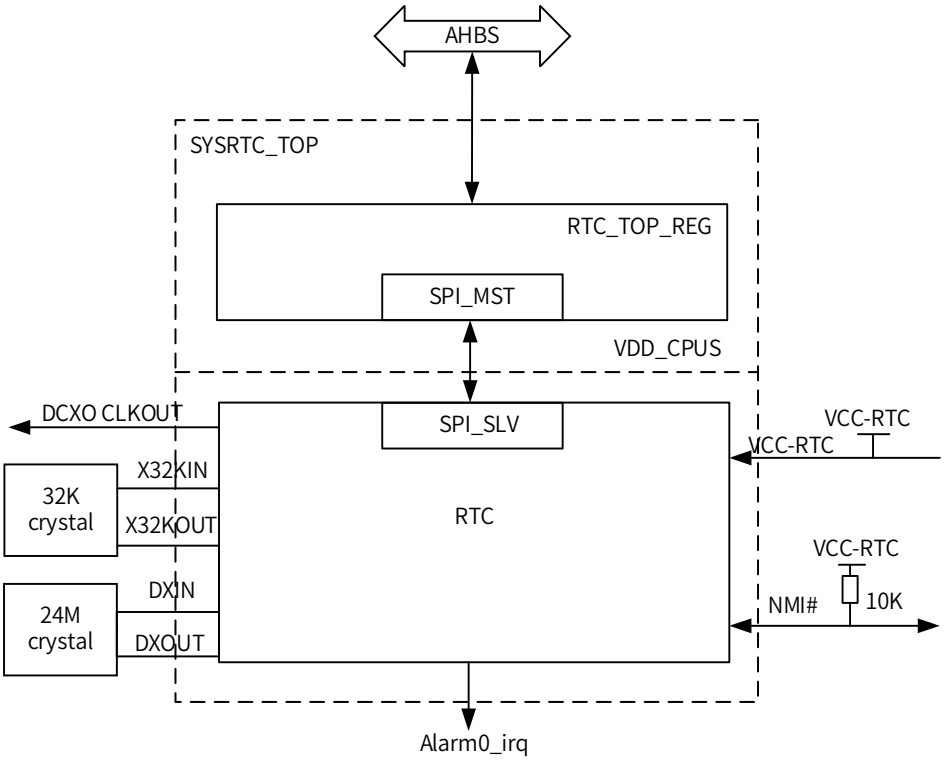
Table 2-19 RTC External Signals

Signal Name	Description	Type
X32KFOUT	32.768 kHz clock Fanout Provides low frequency clock for external devices	AO,OD
X32KIN	Clock Input of 32.768 kHz Crystal	AI
X32KOUT	Clock Output of 32.768 kHz Crystal	AO
DXIN	Digital Compensated Crystal Oscillator Input	AI
DXOUT	Digital Compensated Crystal Oscillator Output	AO
REFCLK-OUT	Digital Compensated Crystal Oscillator Clock Fanout	AO

Signal Name	Description	Type
WREQIN	Request signal of REFCLK_OUT	AI
NMI	Non-Maskable Interrupt	I/O, OD
RESET	Reset Signal (Low Active)	I/O, OD
VCC-DCXO	Digital Compensated Crystal Oscillator Power Supply	P
VCC-RTC	RTC Power	P

2.13.3.2 Typical Application

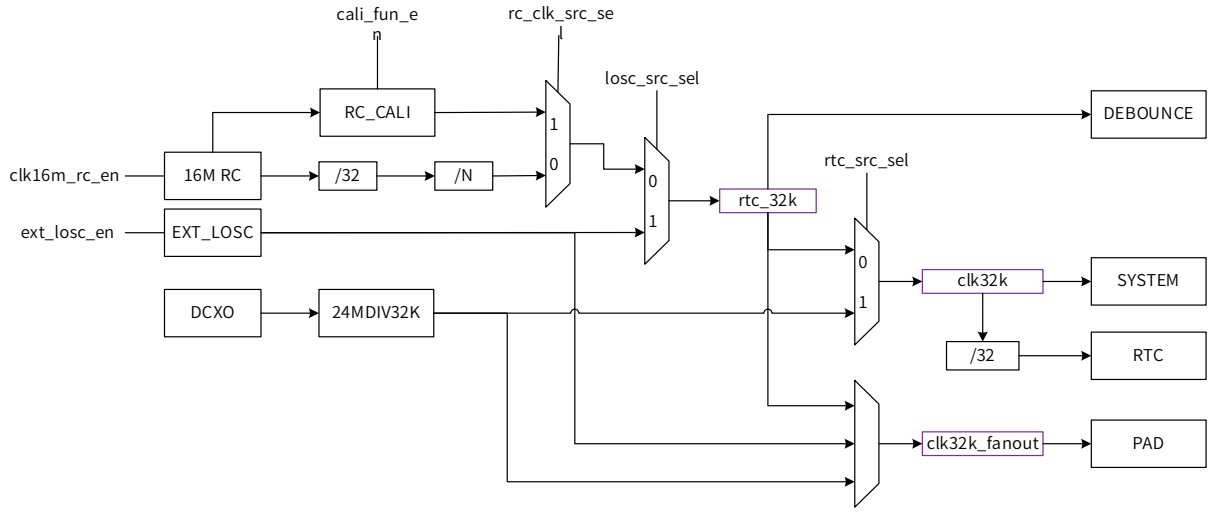
Figure 2-30 RTC Application Diagram



2.13.3.3 Clock Tree

The following figure shows the clock tree of the RTC.

Figure 2-31 RTC Clock Tree



RTC

The RTC has three clock sources:

- 32K divided by internal 16 MHz RC
- 32K divided by external DCXO
- external 32.768 kHz crystal

The RTC selects the internal RC by default, when the system starts, the RTC can select the external low frequency crystal to provide much accurate clock by software. The clock accuracy of the RTC is related to the accuracy of the external low frequency crystal. Usually 32.768 kHz crystal with ±50 ppm frequency tolerance is selected as the clock source. When using internal RC, the clock can be changed by changing division ratio. When using external clock, the clock cannot be changed.

System 32K

The system32K has three clock sources:

- 32K divided by the internal 16 MHz RC
- 32K divided by external DCXO
- external 32.768 kHz crystal

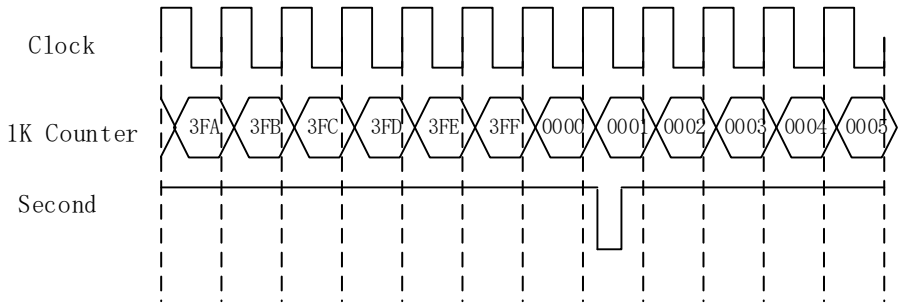
RTC_32K_FANOUT

The RTC_32K_FANOUT has three clock sources:

- 32K divided by the internal 16 MHz RC
- 32K divided by external DCXO
- external 32.768 kHz crystal

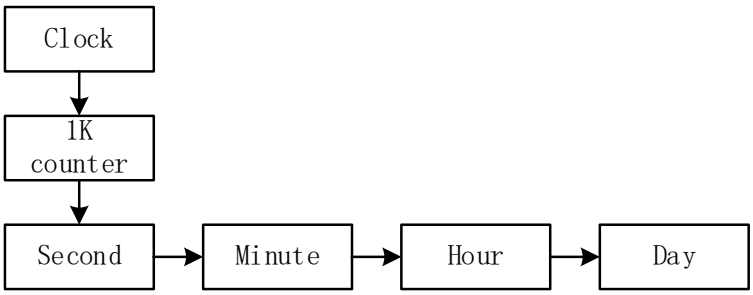
2.13.3.4 Real Time Clock

Figure 2-32 RTC Counter



The 1 kHz counter adds 1 on each rising edge of the clock. When the clock number reaches 0x3FF, 1 kHz counter starts to count again from 0, and the second counter adds 1. The step structure of 1 kHz counter is as follows.

Figure 2-33 RTC 1 kHz Counter Step Structure



According to above implementation, the changing range of each counter is as follows.

Table 2-20 RTC Counter Changing Range

Counter	Range
Second	0 to 59
Minute	0 to 59
Hour	0 to 23
Day	0 to 65535 (The year, month, day need be transformed by software according to day counter)

CAUTION

Because there is no error correction mechanism in the hardware, note that each counter configuration should not exceed a reasonable counting range.

2.13.3.5 Alarm 0

The principle of alarm0 is a comparator. When RTC timer reaches scheduled time, the RTC generates the interrupt, or outputs low level signal by NMI pin to wakeup power management chip.

The RTC only generates one interrupt when RTC timer reached the scheduled day, hour, minute and second counter, then the RTC need set a new scheduled time, the next interrupt can be generated.

2.13.3.6 RTC-VIO

The RTC module has a LDO, the input source of the LDO is VCC-RTC, the output of the LDO is RTC-VIO, the value of RTC-VIO is adjustable, the RTC_VIO is mainly used for internal digital logic.

2.13.3.7 RC Calibration

Figure 2-34 Calibration Circuit

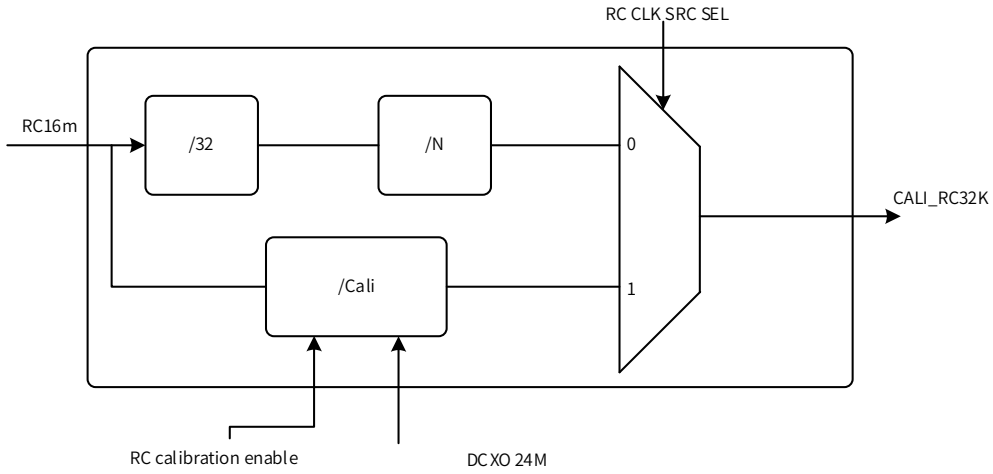
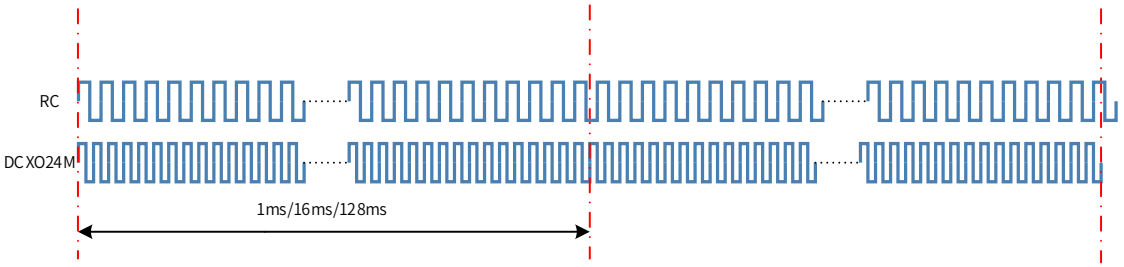


Figure 2-35 RC Waveform



The basic circuit of RC calibration is shown in Figure 2-34. Whether to output the calibrated RC clock can be selected by the RC_Cali_SEL control bit, the calibration principle is as follows:

Step 1 As shown in Figure 2-35, with DCXO 24M as the reference clock, calculate the counter number M of RC clock within 1 ms/16 ms/128 ms to obtain the accurate frequency of internal RC.

Step 2 Divide the accurate frequency by 32.768 kHz and the frequency divider(K) from RC clock to 32.768 kHz is obtained.

Step 3 Divide RC16M by the frequency divider(K) to obtain 32.768 kHz frequency.

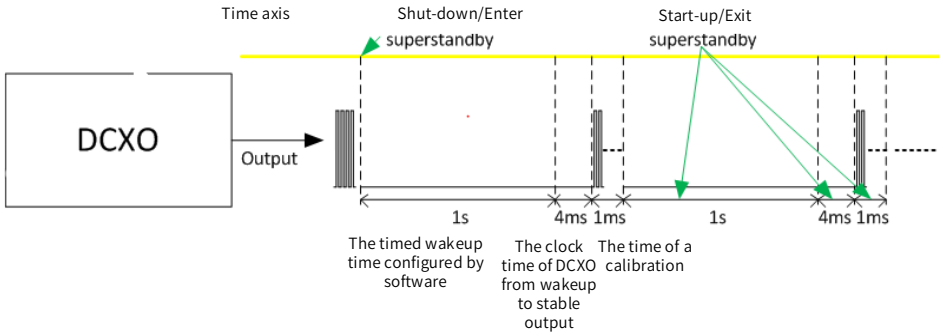
NOTE

The calibration principle is to output 32.768 kHz, not to input 16 MHz.

2.13.3.8 DCXO Timed Wakeup

The logic of DCXO timed wakeup circuit includes two controls: timed wakeup hardware automatic enable and timed wakeup time length (software configuration). The timed wakeup means that DCXO circuit is required to wakeup the output clock once every second (1s-60s, usually the ambient temperature changes little in a few seconds) for 32K calibration in the super standby or shutdown scenario, after calibration, DCXO circuit is closed, the closed time is timed wakeup time length (software configuration). The time of DCXO circuit from wakeup starting to stable output is 3 ms-4 ms. Although the timed wakeup function is closed, DCXO circuit always had worked. The process of timed wakeup is shown in the following figure.

Figure 2-36 DCXO Timed Wakeup Waveform



The time of a calibration in shutdown or super standby:

the timed wakeup time configured by software + the clock time of DCXO from wakeup to stable output + the time of a calibration.

The timed wakeup time configured by the software in the figure is 1 s, and can be configured by software in application. It is the theoretical maximum value for DCXO from wakeup to stable output clock in 4 ms, the specific value is subject to IC measured results. In the any time of these three periods, the startup or exit of the super standby action will not cause DCXO abnormal.

The enable signal of DCXO and the enable signal of timed wakeup DCXO is “OR” logic, and they do not contradict each other.

The interval between continuous DCXO enable operation and disable operation is at least greater than 4 ms.

2.13.3.9 RC Calibration Usage Scenario

Power-on: Select non-accurate 32K divided by internal RC.

Normal scenario: Select external accurate 32K, or external calibrated 32K.

Standby or power-off scenario: Select external accurate 32K, or external calibrated 32K.

2.13.4 Programming Guidelines

2.13.4.1 RTC Clock Control

Step 1 Select clock source: Select clock source by the bit0 of LOSC_CTRL_REG, the clock source is the internal RC oscillator by default. When the system starts, the clock source can be switched to the external 32K oscillator by software.

Step 2 Auto switch: After enabled the bit [15:14] of LOSC_CTRL_REG, the RTC automatically switches clock source to the internal oscillator when the external crystal could not output waveform, the switch status can query by the bit [1] of LOSC_AUTO_SWT_STA_REG.

NOTE

If only configuring the bit [15] of LOSC_CTRL_REG, the clock source status bit cannot be changed after the auto switch is valid, because the two functions are independent.

Here are the basic code samples.

```
Write (0x16aa4000, LOSC_Ctrl); //Write key field
Write (0x16aa4001, LOSC_Ctrl); //Select the external 32K clock
```

2.13.4.2 RTC Calendar

Step 1 Write time initial value: Write the current time to RTC_DAY_REG and RTC_HH_MM_SS_SET_REG.

Step 2 After updated time, the RTC restarts to count again. The software can read the current time anytime.

NOTE

- The RTC can only provide day counter, so the current day counter need be converted to year, month, day and week by software.
- Ensure the bit [8:7] of LOSC_CTRL_REG is 0 before the next time configuration is performed.

Here are the basic code samples.

For example: set time to 21st, 07:08:09 and read it.

```
RTC_DAY_REG = 0x00000015;
```


RTC_HH_MM_SS_REG = 0x00070809; //0000 0000 000|0 0000(Hour) 00|00 0000(Minute) 00|00 0000(Second)

Read (RTC_DAY_REG);

Read (RTC_HH_MM_SS_REG);

2.13.4.3 Alarm0

Step 1 Enable alram0 interrupt by writing ALARM0_IRQ_EN.

Step 2 Set the counter comparator, write the count-down day, hour, minute, second number to ALARM0_DAY_SET_REG and ALARM0_CUR_VLU_REG.

Step 3 Enable alarm0 function by writing ALARM0_ENABLE_REG, then the software can query alarm count value in real time by ALARM0_DAY_SET_REG and ALARM0_CUR_VLU_REG. When the setting time reaches, ALARM0_IRQ_STA_REG is set to 1 to generate interrupt.

Step 4 After enter the interrupt process, write ALARM0_IRQ_STA_REG to clear the interrupt pending, and execute the interrupt process.

Step 5 Resume the interrupt and continue to execute the interrupted process.

Step 6 The power-off wakeup is generated via SoC hardware and PMIC, the software only needs to set the pending condition of alarm0, and set ALARM0_CONFIG_REG to 1.

2.13.4.4 Fanout

CLK32K Fanout

Set the LOSC_OUT_GATING bit (bit [0]) of CLK32K_FOUT_CTRL_GATING_REG register to 1, and ensure external pull-up resistor, voltage, and clock source are normal, then 32.768kHz square wave can be output.

CLK24M Fanout

To fanout CLK24M clock though REFCLK-OUT pin, configure the CLK_REQ_ENB bit (bit [31]) of DCXO_CTRL_REG register.

2.13.5 Register List

Module Name	Base Address
RTC	0x0709 0000

Register Name	Offset	Description
VDD_RTC Power Domain		
LOSC_CTRL_REG	0x0000	LOSC Control Register

2.14 Spinlock

2.14.1 Overview

The spinlock provides hardware synchronization mechanism in multi-core systems. With the lock operation, the spinlock prevents multiple processors from handling the sharing data simultaneously and thus ensure the coherence of data.

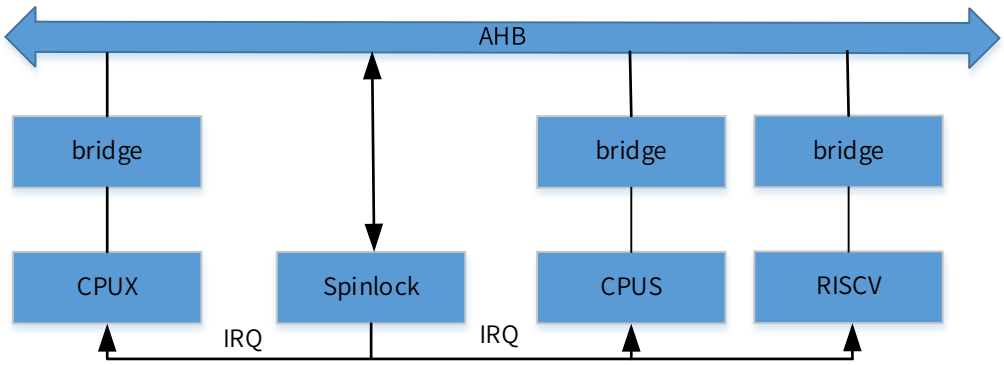
The spinlock has the following features:

- Supports 32 lock units
- Two kinds of lock status: locked and unlocked
- Lock time of the processor is predictable (less than 200 cycles)

2.14.2 Block Diagram

The following figure shows the block diagram of the spinlock.

Figure 2-37 Spinlock Block Diagram



2.14.3 Functional Description

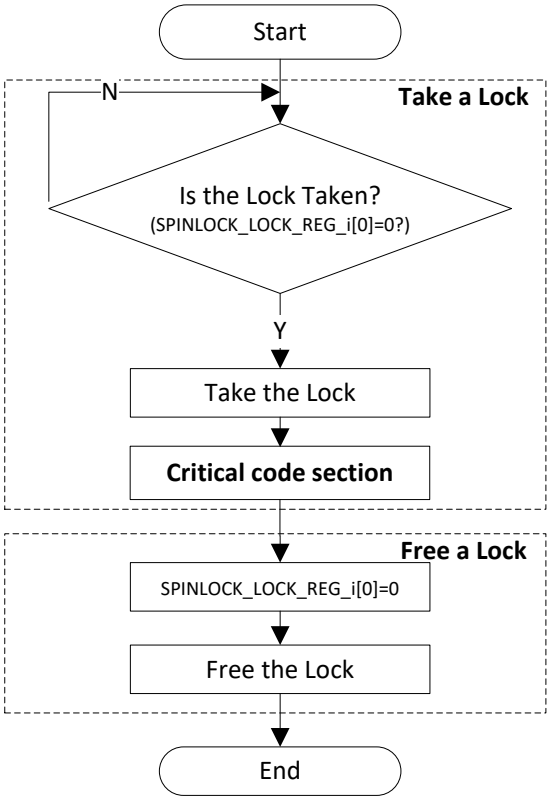
2.14.3.1 Clock and Reset

The spinlock is mounted on AHB. Before accessing the spinlock registers, you need to de-assert the reset signal on AHB bus and then open the corresponding gating signal on AHB bus.

2.14.3.2 Typical Application

The following figure shows a typical application of the spinlock. A processor locks spinlock0 before executing specific codes, and then unlocks the codes. After the lock is freed, other processors can read or write the data.

Figure 2-38 Spinlock Typical Application Diagram



2.14.3.3 Spinlock State Machine

When a processor uses spinlock, it needs to acquire the spinlock status through SPINLOCK_STATUS_REG.

Reading operation

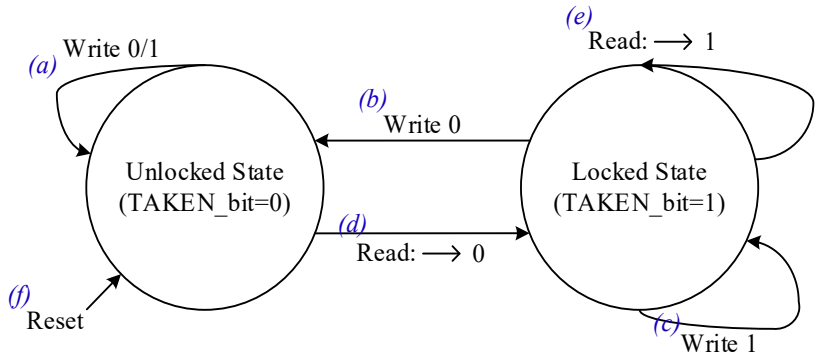
when the return value is 0, it indicates that the spinlock enters the locked status; reading this status bit again can return 1, it indicates that the spinlock is the locked status.

Writing operation

when the spinlock is in the locked status, writing 0 can convert the spinlock to the unlocked status, the writing operation for other status is invalid.

The following figure shows the spinlock state machine.

Figure 2-39 Spinlock State Machine



- When the spinlock is in the unlocked state, writing 0/1 has no effect;
- When the spinlock is in the locked state, writing 0 can convert the corresponding spinlock to the unlocked state;
- When the spinlock is in the locked state, writing 1 has no effect;
- When the spinlock is in the unlocked state, reading the bit can return 0 (it indicates spinlock enters into the locked state);
- When the spinlock is in the locked state, reading the bit can return 1 (it indicates spinlock is in the locked state);
- After reset, the spinlock is in the unlock state by default.

2.14.4 Programming Guidelines

2.14.4.1 Switching the Status

Follow the steps below to switch the lock status of a spinlock.

Step 1 When the read value from SPINLOCKN_LOCK_REG (N=0–31) is 0, the spinlock comes into the locked status.

Step 2 Execute the application codes, and the status of SPINLOCK_STATUS_REG is 1.

Step 3 Write 0 to SPINLOCKN_LOCK_REG (N=0–31), the spinlock converts into the unlocked status, and the corresponding spinlock is released.

2.14.4.2 Processing the Interrupt

The spinlock generates an interrupt when a lock is freed (the lock status converts from the locked status to the unlocked status).

Follow the steps below to process the interrupt:

Step 1 Configure the interrupt enable bit of the corresponding spinlock in SPINLOCK_IRQ_EN_REG to enable the interrupt.

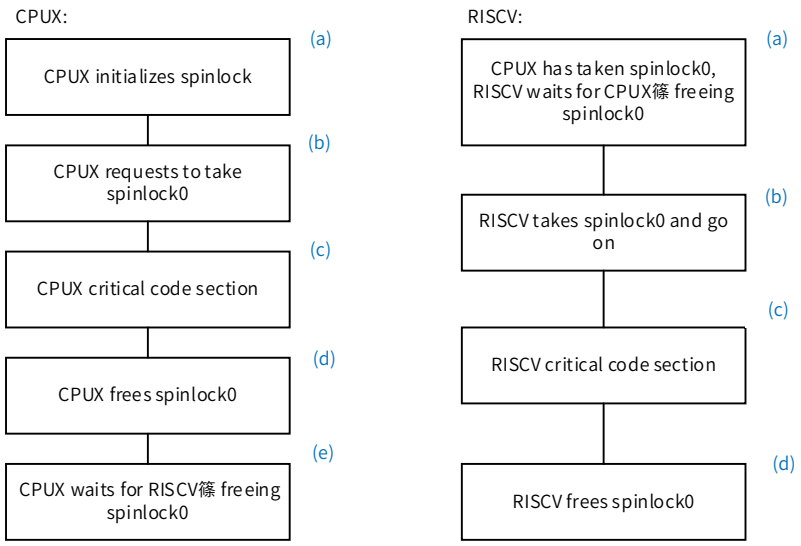
Step 2 The spinlock generates an interrupt when its status converts from the locked status to the unlocked status, and the corresponding bit of the SPINLOCK_IRQ_STA_REG turns to 1.

Step 3 Execute the interrupt handle function and clear the pending bit.

2.14.4.3 Taking/Freeing Spinlock

Take the synchronization between CPUX and RISCV with Spinlock0 as an example, the CPUX and RISCV perform the following steps.

Figure 2-40 CPUX and RISCV Taking/Freeing Spinlock0 Process



CPUX:

a) The CPUX initializes Spinlock.

b) Check lock register0 (SPINLOCK_STATUS_REG0) status. If it is taken, check until CPUX frees spinlock0 and then request to take spinlock0. Otherwise, retry until the lock register0 is taken.

c) Execute CPUX critical code.

d) After executing CPUX critical code, the CPUX frees spinlock0.

The CPUX waits for RISCV to free spinlock0.

RISCV:

a) If the CPUX has taken spinlock0, the RISCV waits for CPUX to free spinlock0.

b) The RISCV requests to take spinlock0. If it fails, retry until the lock register0 is taken.

c) Execute RISCV critical code.

d) After executing RISCV critical code, the RISCV frees spinlock0.

The following codes are for reference.

----------CPUX----------

Step 1 CPUX initializes Spinlock

```
put_wvalue(SPINLOCK_BGR_REG,0x00010000);
put_wvalue(SPINLOCK_BGR_REG,0x00010001);
```

Step 2 CPUX requests to take spinlock0

```
rdata=readl(SPINLOCK_STATUS_REG0);                  //Check lock register0 status
if (rdata != 0) writel (0, SPINLOCK_LOCK_REG0); //If it is taken, check till CPUX frees spinlock0
rdata=readl(SPINLOCK_LOCK_REG0);                    //Request to take spinlock0
if (rdata != 0) rdata=readl(SPINLOCK_LOCK_REG0);    //If it fails, retry till lock register0 is taken
```

---------- **CPUX critical code section** ----------

Step 3 CPUX frees spinlock0

```
writel (0, SPINLOCK_LOCK_REG0);                     //CPUX frees spinlock0
```

Step 4 CPUX waits for RISCV' freeing spinlock0

```
writel(readl(SPINLOCK_STATUS_REG0) == 1);       //CPUX waits for RISCV' freeing spinlock0
```

----------RISCV----------

Step 1 CPUX has taken spinlock0, RISCV waits for CPUX' freeing spinlock0

```
while(readl(SPINLOCK_STATUS_REG0) == 1);        //RISCV waits for CPUX' freeing spinlock0
```

Step 2 RISCV takes spinlock0 and go on

```
rdata=readl(SPINLOCK_LOCK_REG0);                    //Request to take spinlock0
if (rdata != 0) rdata=readl(SPINLOCK_LOCK_REG0);    //If it fails, retry till lock register0 is taken
```

---------- **RISCV critical code section** ----------

Step 3 RISCV frees spinlock0

```
writel (0, SPINLOCK_LOCK_REG0);                     //RISCV frees spinlock0
```

2.14.5 Register List

Module Name	Base Address
SPINLOCK	0x03005000
S_SPINLOCK	0x07093000

2.15 Thermal Sensor Controller (THS)

2.15.1 Overview

The thermal sensors are common elements in wide range of modern system on chips (SoCs) platform. The thermal sensors are used to constantly monitor the temperature on the chip.

The thermal sensor controller (THS) embeds five thermal sensors. THS1_0 is located in the 'big' cores of CPUX; THS1_1 is located in the 'LITTLE' cores of CPUX; THS1_2 is located in the GPU; THS1_3 is located in the MCU; THS0_0 is located in the DDR. When the temperature reaches a certain thermal threshold, the thermal sensor can generate interrupts to the software to lower the temperature via the dynamic voltage and frequency scaling (DVFS) technology.

The THS has the following features:

- Two THS controllers
  - THS0, including THS0_0
  - THS1, including THS1_0, THS1_1, THS1_2, and THS1_3
- Temperature accuracy: ±5°C from -40°C to +60°C, ±3°C from +60°C to +125°C
- Averaging filter for thermal sensor reading
- Supports over-temperature protection interrupt and over-temperature alarm interrupt

2.15.2 Block Diagram

The following figures show the block diagrams of the THS0 and THS1.

Figure 2-41 THS0 Block Diagram

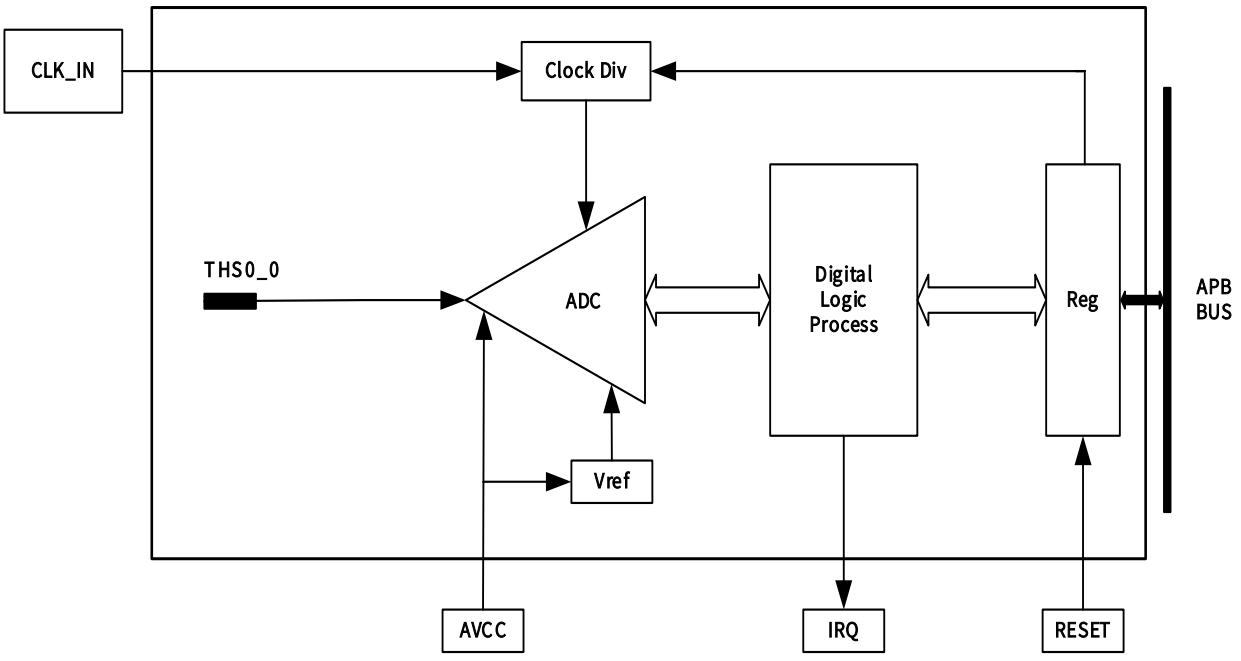
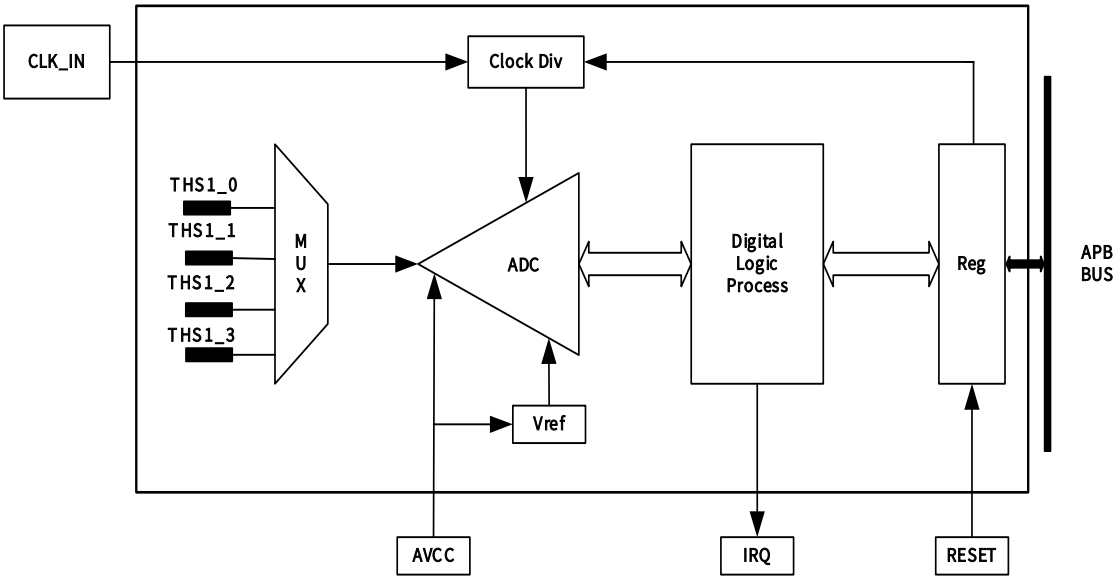


Figure 2-42 THS1 Block Diagram



2.15.3 Functional Description

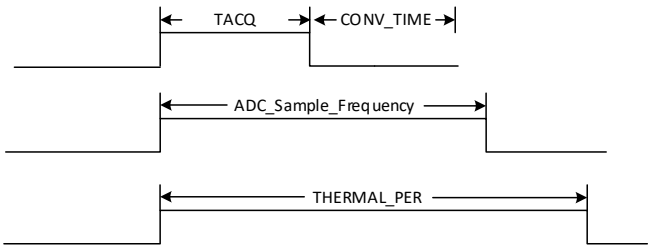
2.15.3.1 Clock Source

Both of THS0 and THS1 get two clock sources: DCXO24M and PCLK. For details about clock configurations, refer to section 2.6 Clock Controller Unit (CCU).

2.15.3.2 Timing Requirements

The following figure shows the timing requirements for the THS module.

Figure 2-43 Thermal Sensor Timing Requirement



CLK_IN = 24 MHz

CONV_TIME (Conversion Time) = 1/24 MHz x 14 Cycles = 0.583 us

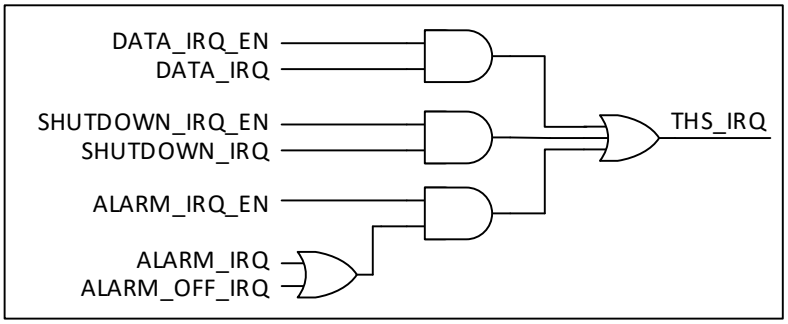
TACQ > 1/24 MHz x 24 Cycles

THERMAL_PER > ADC_Sample_Frequency > TACQ + CONV_TIME

2.15.3.3 Interrupts

The THS module has four interrupt sources: DATA_IRQ, SHUTDOWN_IRQ, ALARM_IRQ, and ALARM_OFF_IRQ. The following figure shows thermal sensor interrupt sources.

Figure 2-44 Thermal Sensor Controller Interrupt Source



DATA_IRQ

The interrupt is generated when the measured sensor_data is updated.

SHUTDOWN_IRQ

The interrupt is generated when the temperature is higher than the shutdown threshold.

ALARM_IRQ

The interrupt is generated when the temperature is higher than the Alarm_Threshold.

ALARM_OFF_IRQ

The interrupt is generated when the temperature drops to lower than the Alarm_Off_Thershold. It is triggered at the fall edge.

2.15.3.4 THS Temperature Conversion Formula

-40°C to +55°C: T=(sensor_data-2736)/ (-13.54)

+55°C to 125°C: T= (sensor_data-2825)/ (-15.33)

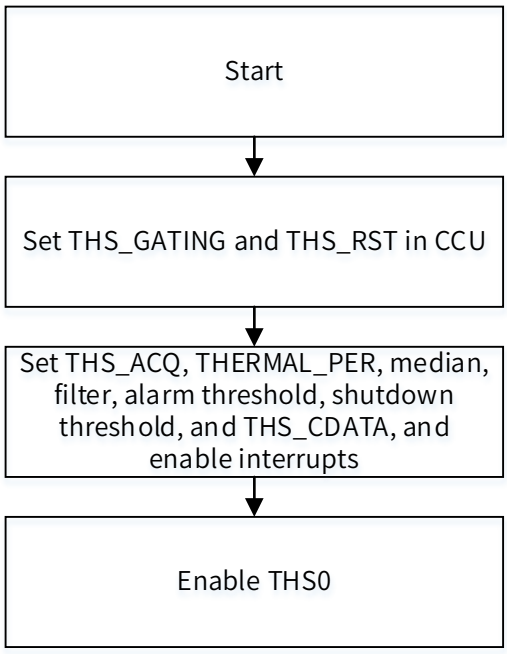
Unit of T: Celsius degree (°C).

The sensor_data is read from the sensor data register.

2.15.4 Programming Guidelines

The initial process of the THS is as follows.

Figure 2-45 THS Initial Process



In the final test (FT) stage, the THS is calibrated through the ambient temperature, and the calibration value is written in the SID module. The following table shows the THS0 and THS1 information in the SID.

Table 2-21 THS Information in the SID

eFuse Name	Base Address	Bit	Description
T-sensor Calibration	0x3B-0x3F (72 bits)	35-24	The calibration value of THS1_0
		23-12	The calibration value of THS1_1
		11-0	ROOM
	0x44-0x48 (72 bits)	35-24	The calibration value of THS0_0
		23-12	The calibration value of THS1_3
		11-0	The calibration value of THS1_2

Before enabling THS, read eFuse value and write the value to THSn_CDATA (n=0, 1, 2, or 3).

Query Mode

the following takes THs0 as an example, THS0 and THS1 are the same.

Step 4 Write 0x1 to the bit [16] of THS_BGR_REG to dessert the reset.

Step 5 Write 0x1 to the bit [0] of THS_BGR_REG to open the THS clock.

Step 6 Write 0x2F to the bit [15:0] of THS_CTRL to set the ADC acquire time.

Step 7 Write 0x1DF to the bit [31:16] of THS_CTRL to set the ADC sample frequency divider.

Step 8 Write 0x3A to the bit [31:12] of THS_PER to set the THS work period.

Step 9 Write 0x1 to the bit [2] of THS_FILTER to enable the temperature convert filter.

Step 10 Write 0x1 to the bit [1:0] of THS_FILTER to select the filter type.

Step 11 Read THS eFuse value from SID, then write the eFuse value to THS0&THS1_CDATA to calibrate THS.

Step 12 Write 0x1 to the bit [0] 0f THS_EN to enable THS.

Step 13 Read the bit [0] of THS_DATA_INTS. If it is 1, the temperature conversion is complete.

Step 14 Read the bit [11:0] of THS0_DATA, and calculate the THS temperature based on section 1.1 .

Interrupt Mode

the following takes THs0 as an example, THS0 and THS1 are the same.

Step 15 Write 0x1 to the bit16 of THS_BGR_REG to dessert the reset.

Step 16 Write 0x1 to the bit0 of THS_BGR_REG to open the THS clock.

Step 17 Write 0x2F to the bit [15:0] of THS_CTRL to set the ADC acquire time.

Step 18 Write 0x1DF to the bit [31:16] of THS_CTRL to set the ADC sample frequency divider.

Step 19 Write 0x3A to the bit [31:12] of THS_PER to set the THS work period.

Step 20 Write 0x1 to the bit2 of THS_FILTER to enable the temperature convert filter.

Step 21 Write 0x1 to the bit [1:0] of THS_FILTER to select the filter type.

Step 22 Read THS eFuse value from SID, and then write the eFuse value to THS0&THS1_CDATA to calibrate THS.

Step 23 Write 0x1 to the bit [0] of THS_DATA_INTC to enable the interrupt of THS.

Step 24 Set GIC interface based on IRQ 71.

Step 25 Put the interrupt handler address into the interrupt vector table.

Step 26 Write 0x1 to the bit [0] 0f THS_EN to enable THS.

Step 27 Read the bit [0] of THS_DATA_INTS. If it is 1, the temperature conversion is complete.

Step 28 Read the bit [11:0] of THS0_DATA, and calculate the THS temperature based on section 1.1 .

2.15.5 Register List

THS module includes two groups of registers:

Module Name	Base Address
THS0	0x0200A000
THS1	0x02009400

2.16 Timer

2.16.1 Overview

The Timer module implements the timing and counting functions, which includes CPUX_TIMER, CPUS_TIMER, and MCU_TIMER. There are 6 timers in TIMER, 3 timers in CPUS_TIMER, and 6 timers in MCU_TIMER.

The Timer module has the following features:

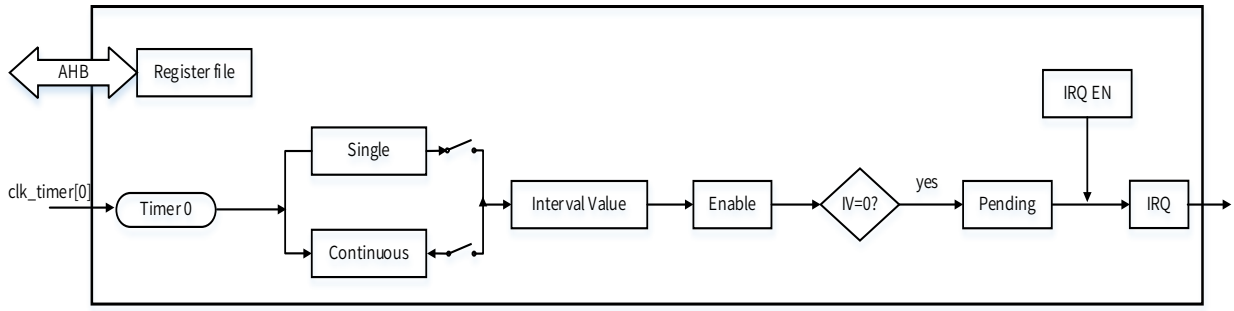
- The AHB port is used to configure the timer register
- Configurable count clock: PRCM/CCU can be switched to 32 kHz, 24 MHz, 16 MHz, and 200 MHz
- Programmable 56-bit down timer
- Supports two timing modes: periodic mode and single counting mode
- Generates an interrupt when the count is decreased to 0

2.16.2 Block Diagram

The timer is a 56-bit down counter. The counter value is decremented by 1 on each rising edge of the timer clock.

The following figure shows the block diagram for the timer.

Figure 2-46 Block Diagram for the Timer



NOTE

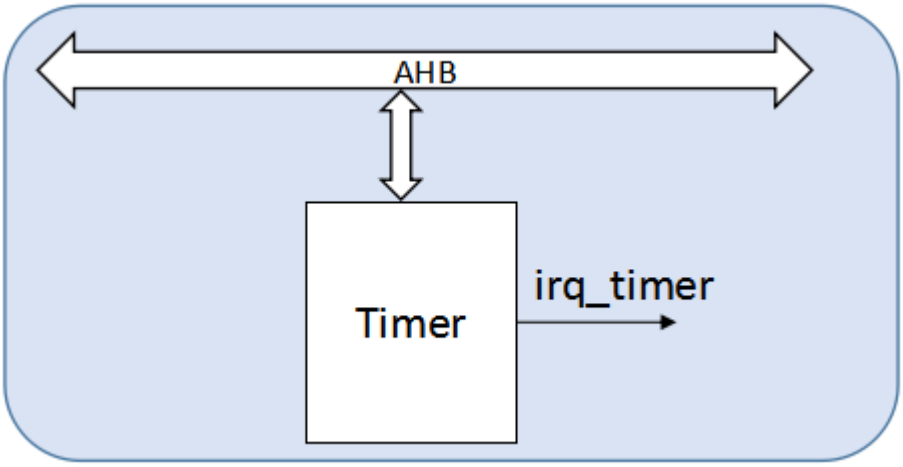
timer0 is used for illustration here. Block diagrams for other timers are the same.

2.16.3 Functional Descriptions

2.16.3.1 Typical Application

The following figure shows the typical application of the Timer module.

Figure 2-47 Timer Typical Application



The Timer is mounted at the AHB bus. The system configures and controls the Timer via the AHB bus.

2.16.3.2 Formula for Calculating the Timer Time

The following formula describes the relationship among timer parameters.

$$T = \frac{\{\text{TIMER\_IVH}, \text{TIMER\_IVL}\} - \{\text{TIMER\_CVH}, \text{TIMER\_CVL}\}}{f_{clk\_timer}}$$

Where,

TIMER_IVH=the higher 24 bits of the interval value, which could be configured by the lower 24 bits of the TIMER_IVH_REG register;

TIMER_IVL=the lower 32 bits of the interval value, which could be configured by the TIMER_IVL_REG register;

TIMER_CVH=the higher 24 bits of the current value, which could be configured by the lower 24 bits of the TIMER_CVH_REG register;

TIMER_CVL=the lower 32 bits of the current value, which could be configured by the TIMER_CVL_REG register;

fclk_timer=the frequency of the timer clock source;

{TIMER_IVH, TIMER_IVL} = 56-bit interval value of the timer;

{TIMER_CVH, TIMER_CVL} = 56-bit current value of the timer.

2.16.3.3 Timing Modes

The timer has two timing modes: the single counting mode and the periodic mode. You can configure the timing mode via the bit[7] of TIMER_CTRL_REG. The value 0 is for the period mode and value 1 is for the single counting mode.

- Single Counting Mode

In the single counting mode, the timer starts counting from the interval value and generates an interrupt after the counter decreases to 0, and then stops counting. It starts to count again only when the interval value is reloaded.

- Periodic Mode

In the periodic mode, the timer restarts another round of counting after generating the interrupt. It reloads data from the Timer Interval Value and then continues to count.

2.16.4 Programming Guidelines

2.16.4.1 Initializing the Timer

Refer to the following steps to initialize the timer:

Step 1 Configure the timer parameters including the clock source and timing mode by writing TIMER_CTRL_REG. There is no sequence requirement of configuring these parameters.

Step 2 Write the interval value.

a) Write TIMER_IVL bit of TIMER_IVL_REG register and TIMER_IVH bit of TIMER_IVH_REG register to configure the interval value for the timer.

b) Write bit [1] of TIMER_CTRL_REG to load the interval value to the timer. The value of the bit will be cleared automatically after the interval value is loaded.

Step 3 Write bit [0] of TIMER_CTRL_REG to start the timer. Read TIMER_CVL bit of TIMER_CVL_REG register and TIMER_CVH bit of TIMER_CVH_REG register to get the current value of the timer.

NOTE

When performing read or write operations on the current register, operate TIMER_CVL_REG register before TIMER_CVH_REG register.

2.16.4.2 Processing the Interrupt

Refer to the following steps to process the interrupt:

Step 1 Enable interrupts for the timer: write the enable bit of the corresponding interrupt in TIMER_IRQ_REG for the timer. The timer will generate an interrupt once the count value reaches 0.

Step 2 After the software program enters the interrupt process, write the pending bit of the corresponding interrupt in TIMER_STA_REG to clear the interrupt pending.

Step 3 Resume the interrupt and continue to execute the interrupted process.

2.16.5 Register List

Module Name	Base Address	The value of N
CPUX_TIMER	0x0300 8000	0-5
CPUS_TIMER	0x0709 0400	0-2
MCU_TIMER	0x0712 3000	0-5

Register Name	Offset	Description
TIMER_IRQ_REG	0x0000	Timer IRQ Enable Register
TIMER_STA_REG	0x0004	Timer Status Register
TIMER_SEC_REG	0x0008	Timer Secure Register
TIMER_CTRL_REG	0x0020+0x0020*N	Timer Control Register
TIMER_IVL_REG	0x0024+0x0020*N	Timer Interval Value bit[31:0] Register
TIMER_CVL_REG	0x0028+0x0020*N	Timer Current Value [31:0]bit Register
TIMER_IVH_REG	0x002C+0x0020*N	Timer Interval Value bit[55:32] Register
TIMER_CVH_REG	0x0030+0x0020*N	Timer Current Value [55:32]bit Register

2.16.6 Register Description

2.16.6.1 0x0000 Timer IRQ Enable Register (Default: 0x0000_0000)

Offset: 0x0000			Register Name: TIMER_IRQ_REG
Bit	Read/Write	Default/Hex	Description
31:NUM	/	/	/
NUM-1:0	R/W	0x0	Timer (0-NUM-1) Interrupt Enable 1: Enable 0: Disable

2.17 Watchdog Timer (WDT)

2.17.1 Overview

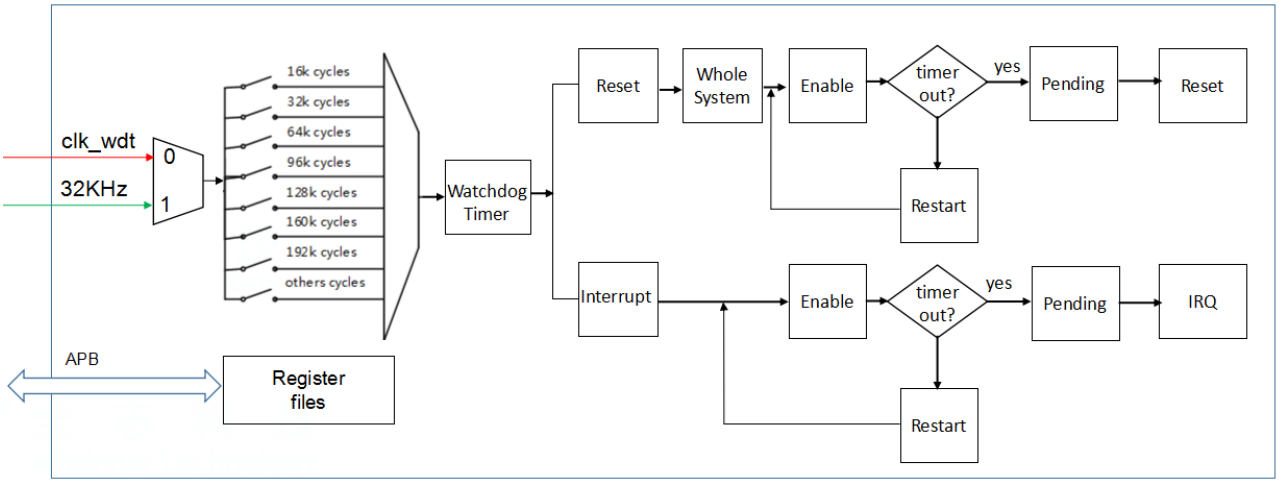
Watchdog is used to transmit a reset signal to reset the entire system after an exception occurs in the system. It has the following features:

- Three watchdog timers: CPUX_WDT in CPUX domain, CPUS_WDT and RISCV_WDT in CPUS domain
- 12 initial values to configure
- Generation of timeout interrupts
- Generation of reset signal
- Watchdog restart the timing

2.17.2 Block Diagram

The following figure shows the functional block diagram of the watchdog module.

Figure 2-48 Watchdog Block Diagram



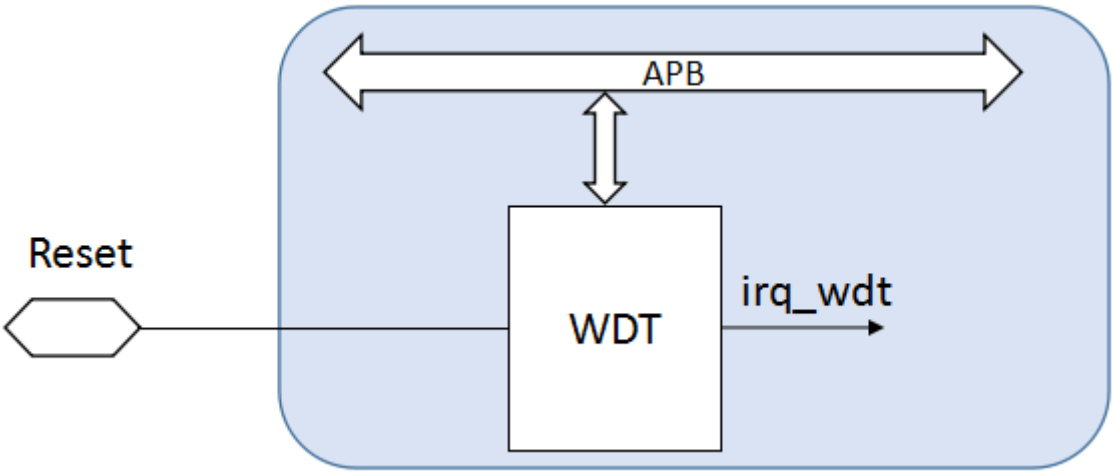
2.17.3 Functional Descriptions

2.17.3.1 Clock Sources

The clock source of the watchdog is either LOSC (32 kHz) or HOSC/750 (24 MHz/750). Configure the bit [8] of the WDT_CFG_REG to select a clock source.

2.17.3.2 Typical Application

Figure 2-49 Watchdog Application Diagram



Watchdog configures register by APB bus.

The system configures the time of watchdog, if the system has no timing for restart watchdog (such as bus hang dead), then watchdog sends out watchdog reset external signal to reset system; meanwhile watchdog outputs signal to RESET pad to reset PMIC.

2.17.3.3 Operating Modes

The watchdog is a 32-bit down counter; the counter value is decreased by 1 on each rising edge of the count clock. The watchdog has two operating modes.

- Interrupt mode

The bit [1:0] of the WDT_CFG_REG is set to 0x2, when the counter value reaches 0 and the bit [0] of the WDT_IRQ_EN_REG is written to 1, the watchdog generates an interrupt, the watchdog enters into interrupt mode.

- Reset mode

The bit [1:0] of the WDT_CFG_REG is set to 0x1, when the counter value reaches 0, the watchdog generates a reset signal to reset the entire system.

2.17.4 Programming Guidelines

2.17.4.1 Initializing the Watchdog

Follow the steps below to initialize the watchdog:

Step 1 Configure the bit [1:0] of the WDT_CFG_REG to configure the generation of the interrupts or the output of reset signal.

Step 2 Configure the bit [7:4] of the WDT_MODE_REG to configure the initial count value.

Step 3 Write the bit [0] of the WDT_MODE_REG to 1 to enable the watchdog.

2.17.4.2 Processing the Interrupt

Follow the steps below to process the interrupt:

Step 1 Write the bit [0] of the WDT_IRQ_EN_REG to 1 to enable the interrupt.

Step 2 After enter the interrupt process, write the bit [0] of the WDT_IRQ_STAT to 1 to clear the interrupt pending, and execute the process of waiting for the interrupt.

Step 3 Resume the interrupt and continue to execute the interrupted process.

2.17.4.3 Resetting the Watchdog

In the following instance making configurations for WDT: configure clock source as HOSC/750, configure Interval Value as 1s and configure Watchdog Configuration as to whole system. This instance indicates that reset system after 1s.

```
writel (0x16AA_0001, WDT_CFG_REG);                          //To whole system
writel (0x16AA_0010, WDT_MODE_REG);                         //Interval Value set 1s
writel(readl(WDT_MODE_REG) |(1<<0)|(0x16AA<<16), WDT_MODE_REG);      //Enable WDT
```

2.17.4.4 Restarting the Watchdog

In the following instance making configurations for WDT: configure clock source as HOSC/750, configure Interval Value as 1s and configure Watchdog Configuration as to whole system. In the following instance, if the time of other codes is larger than 1s, watchdog will reset the whole system. If the sentence of restart watchdog is implemented inside 1s, watchdog will be restarted.

```
write (0x16AA_0001, WDT_CFG_REG);                           //To whole system
write (0x16AA_0010, WDT_MODE_REG);                          //Interval Value set 1s
writel(readl(WDT_MODE_REG) |(1<<0)|(0x16AA<<16), WDT_MODE_REG);     //Enable WDT
----other codes---
writel(readl(WDT_CTRL_REG) |(0xA57<<1) | (1<<0), WDT_CTRL_REG); //Write 0xA57 at Key Field and Restart WDT
```

2.17.5 Register List

Module Name	Base Address
CPUX_WDT	0x0205 0000
CPUS_WDT	0x0702 0400
RISCV_WDT	0x0713 2000