TCC70xx MCU BSP

User Guide

Rev. 1.00 [A]
2023-03-31

※ The information in this document is subject to change without notice and should not be construed as a commitment by Telechips, Inc.

Kindly visit www.telechips.com for more information.

TABLE OF CONTENTS

Contents

Figures

Tables

1 INTRODUCTION

This document describes TCC70xx MCU BSP. The TCC70xx MCU BSP means all contents including source code, build environment, and tools that you downloaded via git.

1.1 Terminology

Table 1.1 Terminology

BL	Boot Loader
BSP	Board Support Package
Cortex-R5	Cortex-R5 processor
EVB	Evaluation Board
FWDN	Firmware Downloader
F/W	Firmware
MCU	Micro Controller Unit
MPU	Memory Protection Unit
OS	Operating System
pFlash	Program Flash memory
RTOS	Real Time OS
S-NOR	Serial NOR Flash memory
SAL	System Adaptation Layer
XIP	eXecute In Place

2 SOFTWARE ARCHITECTURE

Figure 2.1 shows software components in TCC70xx MCU BSP. The TCC70xx MCU BSP is composed of device driver, Real Time Operating System (RTOS), System Adaptation Layer (SAL), and sample application. The TCC70xx MCU BSP is categorized into three types (Sample, 3rd party and TC Support) by usage.

- Sample is a sample code for testing device driver. You can refer to the sample code while implementing your own solution, but there is no guarantee for quality.
- The TCC70xx MCU BSP has third-party software such as RTOS, but this TCC70xx MCU BSP does not provide technical support for the RTOS itself.
- The device driver and SAL are supported by Telechips (TC Support).

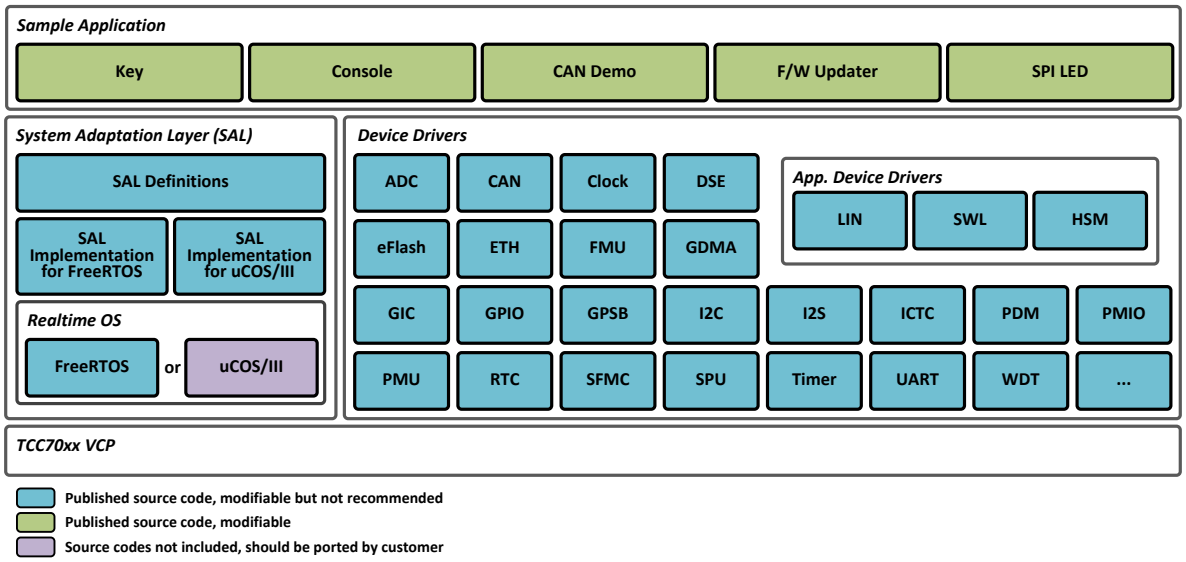

Figure 2.1 Block Diagram of Software Components in TCC70xx MCU BSP

2.1 Device Driver

Certain device drivers in the TCC70xx MCU BSP are supported by Telechips, but complex device driver such as CAN device driver can be added and developed by the third party or Tier 1. For more detailed information on each device driver, refer to "*TCC70xx MCU BSP-API Specification for xxx*" [3].

2.2 Operating System (OS)

The TCC70xx MCU BSP basically uses FreeRTOS which can be replaced by your own OS. If you have the source code and license of uC/OS-III, you can easily replace it.

Note 1: The uC/OS-III source code is not included in the TCC70xx MCU BSP, but some portable components are included. The TCC70xx MCU BSP is optimized for uC/OS-III v4.04.03 and FreeRTOS v10.3.1. The uC/OS-III is supported based on the contract with Micrium. For more detailed information on uC/OS-III, refer to Micrium's website [5].
Note 2: FreeRTOS is distributed in the hopes that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. Full license text is available on the following link: http://www.freertos.org/a00114.html

2.3 System Adaptation Layer (SAL)

SAL is an adaptation layer to change OS component and system resources. For detailed information on SAL, refer to "*TCC70xx MCU BSP-API Specification for System Adaptation Layer*" [3].

2.4 Sample Application

The TCC70xx MCU BSP provides a sample application for Telechips EVB with sample code applied. However, according to the OEM specifications, Tier 1 should develop its own application. If you have an existing solution, you can apply it to your application.

3 DIRECTORY STRUCTURE

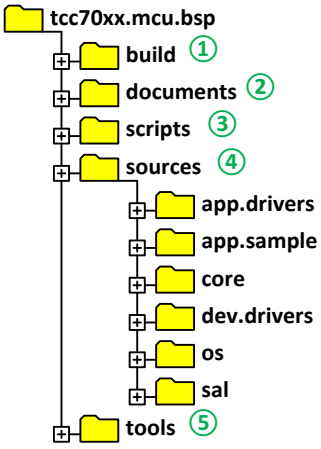

Figure 3.1 Directory Structure of TCC70xx MCU BSP

① build: Build environment & utilities.
② documents: All of documents for TCC70xx MCU BSP
③ scripts: Debug script for JTAG debugger such as Trace32
④ sources: Source code
- app.drivers: Application drivers. These drivers do not access hardware directly.
- app.sample: Sample applications
- core: Assembly core files
- dev.drivers: Device drivers
- os: Operating System
- sal: System Adaptation Layer

⑤ tools: Tools for FWDN and F/W upgrade

4 BOOTUP PROCESS

Three elements are required to initialize MCU: TCC70xx MCU Boot Code (MCU-BL0), MCU firmware (F/W), and HSM Boot Code (HSM F/W). Figure 4.1 shows the simple MCU bootup process. MCU-BL0 is in the internal Boot ROM and executed when the power is turned on. MCU F/W and HSM F/W are in the pFlash (eFlash or S-NOR) memory. MCU-BL0 loads HSM F/W. And HSM F/W verifies MCU F/W. If MCU F/W is valid, MCU-BL0 executes MCU F/W. MCU F/W is executed as XIP on pFlash memory. This document does not cover HSM F/W. For more information on HSM F/W, contact Telechips [1].

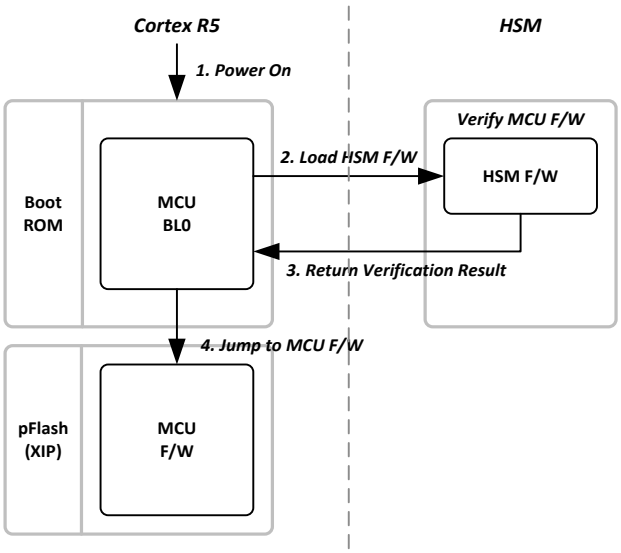

Figure 4.1 Simple MCU Bootup Process

Figure 4.2 shows detailed MCU bootup process.

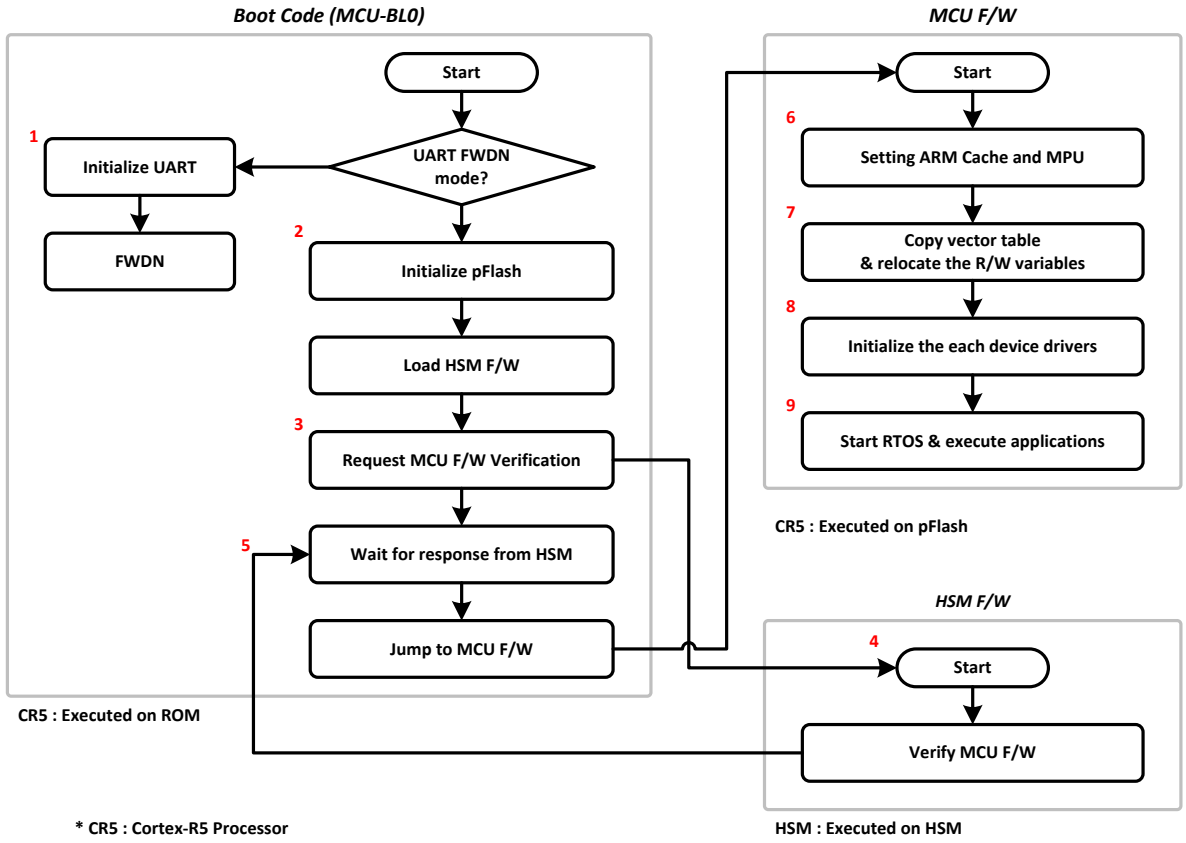

Figure 4.2 Detailed MCU Bootup Process

1. Execute internal boot code (MCU-BL0):
   1) If the boot mode is UART, execute the FWDN process. ----- (1)
   2) If the boot mode is normal, initialize the storage pFlash. ----- (2)
   3) Load HSM F/W, request MCU F/W image verification to HSM and wait for response from HSM. ----- (3)
2. Execute HSM H/W:
   1) Verify the MCU F/W image. ----- (4)
3. Execute MCU F/W:
   1) Jump to MCU F/W ----- (5)
   2) Initialize core elements of Cortex-R5 such as cache, MPU, ARM stack, and so on. - "startup.s" ----- (6)
   3) Copy the vector table and relocate the R/W (.DATA) variable and initialize the Zero Initialize (.ZI) area. ----- (7)
   4) Initialize each device driver. ----- (8)
   5) Start the RTOS and run the application. ----- (9)

Figure 4.3 shows the F/W image layout.

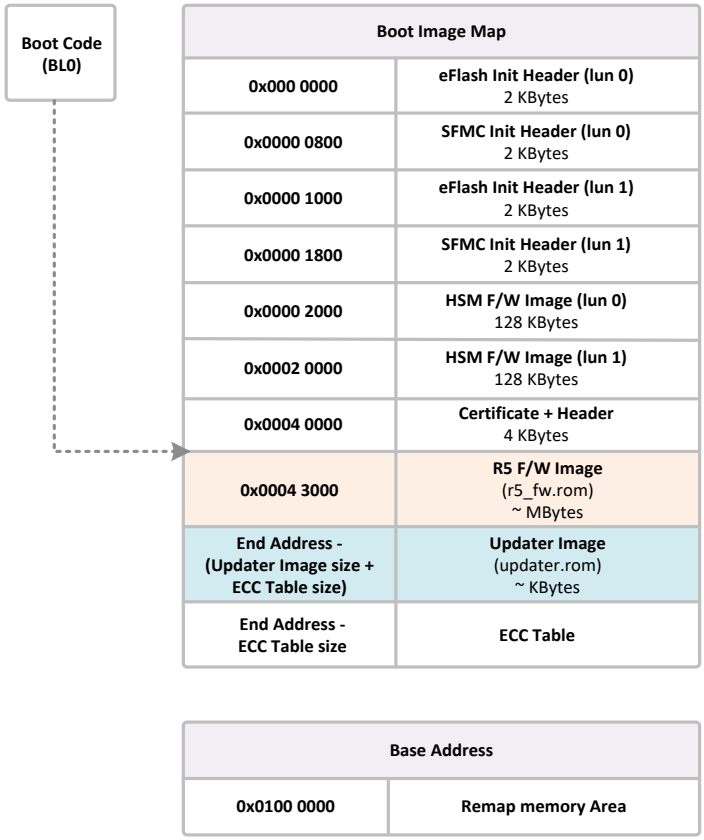

Figure 4.3 Memory Layout of Flash Image

5 RESET PROCESS

The TCC70xx MCU BSP can reset by itself.

5.1 Cold Reset

The TCC70xx MCU BSP can perform a cold reset. Refer to the following function. You can call this function in your application.

tcc70xx_mcu_bsp/sources/dev.drivers/pmu/tcc70xx/pmu.c

```
void PMU_ColdReset(void)
{
    PMU_REG_SET
    (
        PMU_VA_WR_PASS,
        PMU_ADDR_PMU_WR_PW_FIELD_PMU_WR_PW,
        PMU_ADDR_PMU_WR_PW_FIELD_PMU_WR_PW_MASK,
        PMU_ADDR_PMU_WR_PW
    );

    PMU_REG_SET
    (
        0x1,
        PMU_ADDR_COLD_RST_REQ_FIELD_COLD_RST_REQ,
        PMU_ADDR_COLD_RST_REQ_FIELD_COLD_RST_REQ_MASK,
        PMU_ADDR_COLD_RST_REQ
    );
}
```

5.2 Warm Reset

The TCC70xx MCU BSP can perform a warm reset. Refer to the following function. You can call this function in your application.

tcc70xx_mcu_bsp/sources/dev.drivers/pmu/tcc70xx/pmu.c

```
void PMU_WarmReset(void)
{
    PMU_REG_SET
    (
        PMU_VA_WR_PASS,
        PMU_ADDR_PMU_WR_PW_FIELD_PMU_WR_PW,
        PMU_ADDR_PMU_WR_PW_FIELD_PMU_WR_PW_MASK,
        PMU_ADDR_PMU_WR_PW
    );

    PMU_REG_SET
    (
        0x1,
        PMU_ADDR_WARM_RST_REQ_FIELD_WARM_RST_REQ,
        PMU_ADDR_WARM_RST_REQ_FIELD_WARM_RST_REQ_MASK,
        PMU_ADDR_WARM_RST_REQ
    );
}
```

5.3 Watchdog Reset

The TCC70xx MCU BSP can perform a watchdog reset. For more detailed information on how to use watchdog reset, refer to "*TCC70xx MCU BSP-API Specification for Windowed Watchdog Timer*" [3].

6 DEVELOPMENT ENVIRONMENT

For information on TCC70xx MCU BSP development environment, refer to Chapter 4 Download MCU BSP and Chapter 5 Build Guide in *"TCC70xx MCU BSP-Getting Stared"* [3].

7 MEMORY MAP

In the TCC70xx EVB environment, the memories available for the MCU are assumed to be exclusive to pFlash memory for code operation and internal SRAM. The memory layout can be changed according to target product and requirements.

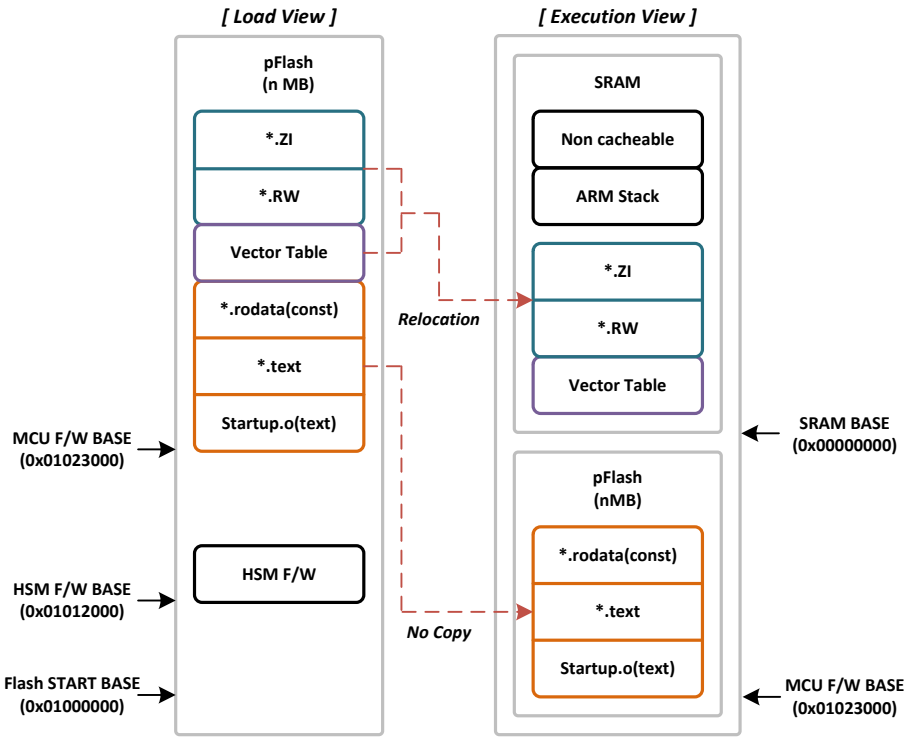

Figure 7.1 Memory Map of TCC70xx MCU BSP

7.1 Load View

Load view in Figure 7.1 above schematizes where the firmware image is written in the memory. The target firmware image is generated after compiling, and the linking process is saved in the pFlash memory. Each area is classified as .text, .rodata, .RW, and .ZI based on the property of data.

7.2 Execution View

Execution view in Figure 7.1 above shows the status of memory used when the system operates after booting. Since the pFlash can execute the instruction at once, it does not need to copy data (.text and .rodata) to the separate RAM. Others except .text and .rodata should be copied to the RAM because the data (.RW and .ZI) must be able to read and write. However, .ZI data actually does not need to be copied because the value of .ZI should be 0 (zero). Therefore, .ZI is initialized to 0 instead of being copied. That is, the code is executed in the pFlash and the data is executed in the RAM. The internal SRAM (0x00000000) can be remapped to ARM vector address for the Cortex-R5, and the vector table must be located on the logical zero. Therefore, you should copy the vector table code to the internal SRAM.

7.3 Define Memory Layout Base Address

In the case of Green Hills IDE, assignment of load base address and data copy base address (SRAM area) can be adjusted in the linker script file (.ld) of Green Hills IDE. The load base address is defined as `flash_memory`, while the data copy base address is defined as `dram_memory`.

- Define Memory Layout (Green Hills IDE)

tcc70xx_mcu_bsp/build/TCC70xx/ghc/tgt/standalone_romcopy_TCC70xx_snorwithPreload.ld

```
MEMORY
{
    // Reserve 0x8000 bytes for use by target-resident code or debug monitor
    dram_memory   : ORIGIN = 0x00000000, LENGTH = (SRAM_TOTAL_SIZE - ARM_STACK_SIZE - DMA_NC_SIZE
- CAN_NC_SIZE - LOADCODE_SIZE)

// for preload code on ram
dram_memory    : ORIGIN = 0x00000000 + (SRAM_TOTAL_SIZE - ARM_STACK_SIZE - DMA_NC_SIZE - CAN_NC_SIZE
- LOADCODE_SIZE),
LENGTH = LOADCODE_SIZE

    // Reserve 0x680 bytes for arm stack
    stack_memory     : ORIGIN = 0x00000000 + (SRAM_TOTAL_SIZE - ARM_STACK_SIZE - DMA_NC_SIZE -
CAN_NC_SIZE),
    LENGTH = ARM_STACK_SIZE

    // Reserve CAN_NC_SIZE bytes for non-cacheable area
    can_memory    : ORIGIN = 0x00000000 + (SRAM_TOTAL_SIZE - DMA_NC_SIZE - CAN_NC_SIZE), LENGTH =
CAN_NC_SIZE

    // Reserve DMA_NC_SIZE bytes for non-cacheable area
    dma_memory    : ORIGIN = 0x00000000 + (SRAM_TOTAL_SIZE - DMA_NC_SIZE), LENGTH = DMA_NC_SIZE

    // Flash code area XIP
    flash_memory     : ORIGIN = 0x01043000, LENGTH = 0x00200000 //2MByte
}
```

In the case of GCC environment, the memory layout is defined by project's linker script file (linker_512.ld) as shown below. The load base address is defined as REMAP_FLASH, while the data copy base address is defined as SRAM.

- Define Memory Layout (GCC Environment)

tcc70xx_mcu_bsp/build/TCC70xx/gcc/linker_512_withPreload.ld

```
MEMORY
{
    /*=========================================================
      Region for Executing at SRAM with cacheable: 256KB
      =========================================================*/
    SRAM (rwx) : ORIGIN = 0x00000000, LENGTH = (SRAM_TOTAL_SIZE - ARM_STACK_SIZE - DMA_NC_SIZE -
CAN_NC_SIZE)
    /* total(0x80000) - ARM_STACK_SIZE(0x680) - NC_SRAM(0x2000 or...config) - CAN_NC_SRAM(0x8000)
*/

    /*=========================================================
      Region for ARM STACK area: 0x680Byte
      =========================================================*/

    SRAM_CODELOAD  (rwx)  : ORIGIN = 0x00000000 + (SRAM_TOTAL_SIZE - ARM_STACK_SIZE - DMA_NC_SIZE
- CAN_NC_SIZE - LOADCODE_SIZE), LENGTH = LOADCODE_SIZE
    SRAM_ARM_STACK (rwx) : ORIGIN = 0x00000000 + (SRAM_TOTAL_SIZE - ARM_STACK_SIZE - DMA_NC_SIZE
- CAN_NC_SIZE),
    LENGTH = ARM_STACK_SIZE

    /*=========================================================
      Region for Executing at SRAM with non-cacheable for CAN : 32KB
      =========================================================*/
    CAN_NC_SRAM (rwx)    : ORIGIN = 0x00000000 + (SRAM_TOTAL_SIZE - DMA_NC_SIZE - CAN_NC_SIZE),
LENGTH = CAN_NC_SIZE

    /*=========================================================
      Region for Executing at SRAM with non-cacheable : 8KB
      =========================================================*/
    NC_SRAM (rwx) : ORIGIN = 0x00000000 + (SRAM_TOTAL_SIZE - DMA_NC_SIZE), LENGTH = DMA_NC_SIZE

    /*=========================================================
      Region for Executing at remap Flash area with cacheable
      =========================================================*/
    REMAP_FLASH (rx) : ORIGIN = 0x01043000, LENGTH = 0x00200000
}
```

7.4 Relocate RW and ZI

Copying RW and initializing ZI area are executed in the `InitCVar()` of "startup.s". In the `InitCVar()`, the location and size of each area, which are referred for initiation of the RW and ZI area, are based on variables defined by linker script file (.ld) function of Green Hills IDE. The name, location, and size of variables can be changed arbitrarily depending on the use-cases. Refer to the following linker script file (.ld):

- Variable definition of each area (Green Hills IDE)

tcc70xx_mcu_bsp/build/TCC70xx/ghc/tgt/standalone_romcopy_TCC70xx_snorwithPreload.ld

```
// These special symbols mark the bounds of RAM and ROM memory.
    __ghs_romstart    = MEMADDR(flash_memory);
    __ghs_romend      = ADDR(.ROM.data);
    __ghs_ramstart    = ADDR(.data);
    __ghs_ramend      = ENDADDR(.bss);
    __ghs_bssstart    = ADDR(.bss);
    __arm_stack       = MEMADDR(stack_memory);
    __nc_canstart     = MEMADDR(can_memory);
    __nc_dmastart     = MEMADDR(dma_memory);
    _end_of_nc_can    = __nc_canstart + CAN_NC_SIZE;
    _end_of_nc_dma    = __nc_dmastart + DMA_NC_SIZE;
    __PRECODE_START_LOAD    = ADDR(.preload_code);
    __PRECODE_RAM_START__   = MEMADDR(SRAM_CODELOAD);
    __PRECODE_SIZE__        = __PRECODE_RAM_START__+ SIZEOF(.preload_code);
```

■ Variable definition of each area (GCC Environment)

tcc70xx_mcu_bsp/build/TCC70xx/gcc/linker_512_withPreload.ld

```
/* .vector section which is used for vector table */
.vector:
{
    __VECTOR_START_LOAD = LOADADDR (.vector);
    __VECTOR_START__ = .;
    *vector.o (.vector .text .rodata .rodata*)
    . = ALIGN(0x20);
    __VECTOR_END__ = .;
} > SRAM AT > REMAP_FLASH

.codeonsram :
  {
    __PRECODE_START_LOAD = LOADADDR (.codeonsram);
    __PRECODE_SIZE__ = SIZEOF (.codeonsram);
    __PRECODE_RAM_START__ = .;
    *preload.o (.text .rodata .rodata* .rwdata*)
    . = ALIGN(0x20);
  } > SRAM_CODELOAD AT > REMAP_FLASH

/* .data section which is used for initialized data */
.data:
    AT ( LOADADDR (.codeonsram) + SIZEOF (.codeonsram) )
{
    __DATA_START_LOAD = LOADADDR (.data);
    __DATA_START__ = .;
    *(.data)
    *(.data.*)
    . = ALIGN(0x20);
    __DATA_END__ = .;
} > SRAM

/* .bss section which is used for uninitialized data */
.bss (NOLOAD):
{
    . = ALIGN(0x20);
    __BSS_START__ = .;
    *(.bss)
    *(.bss.*)
    . = ALIGN(0x20);
    __BSS_END__ = .;
} > SRAM
```

The RW and ZI variables are manually relocated to main RAM of system by software while the system starts up.

- Variable reference of each area (Green Hills IDE)

tcc70xx_mcu_bsp/sources/core/GHC/startup.asm

```
InitCVar
    ldr r0, =__ghs_romend
    ldr r1, =__ghs_ramstart
    ldr r2, =__ghs_bssstart

    cmp r0, r1
    beq 5f
4:                  ; Copy RW Region
    ldm r0!, {r4-r11}
    stm r1!, {r4-r11}
    cmp r1, r2
    bne 4b
5:                  ; Clear BSS Region
    ldr r0, =__ghs_bssstart
    ldr r1, =__ghs_ramend
    add r1, r1, #0x1F
    bic r1, r1, #0x1F
    cmp r0, r1
    moveq   pc, lr

    mov r4, #0
    mov r5, #0
    mov r6, #0
    mov r7, #0
    mov r8, #0
    mov r9, #0
    mov r10, #0
    mov r11, #0
6:
    stm r0!, {r4-r11}
    cmp r0, r1
    bne 6b

    mov pc, lr
```

■ Variable reference of each area (GCC Environment)

tcc70xx_mcu_bsp/sources/core/GCC/startup.S

```
InitCVar:
    ldr r0, =(__DATA_START_LOAD) @ load address
    ldr r1, =(__DATA_START__)
    ldr r2, =(__DATA_END__)

    cmp r0, r1
    beq 5f
4:  /* Copy RW Region */
    ldm r0!, {r4-r11}
    stm r1!, {r4-r11}
    cmp r1, r2
    bne 4b
5:  /* Clear BSS Region */
    ldr r0, =(__BSS_START__)
    ldr r1, =(__BSS_END__)
    cmp r0, r1
    moveq   pc, lr

    mov r4, #0
    mov r5, #0
    mov r6, #0
    mov r7, #0
    mov r8, #0
    mov r9, #0
    mov r10, #0
    mov r11, #0
6:
    stm r0!, {r4-r11}
    cmp r0, r1
    bne 6b

    mov pc, lr
```

7.5 Define Memory Attribute for Each Region Using MPU

Since Cortex-R5 is a processor that supports Memory Protection Unit (MPU), the TCC70xx MCU BSP uses the MPU, which can be set as follows:

- MPU table for setting region

tcc70xx_mcu_bsp/sources/dev.drivers/mpu/TCC70xx/mpu.c

```
MPUConfig_t sMPUTable[ MPU_MAX_REGION ] =
{
    { MPU_REGION_ENABLE, MPU_SRAM0_ADDR, MPU_REGION_512KB, MPU_NORM_SHARED_WB_WA |
MPU_PRIV_RW_USER_RW },
    { MPU_REGION_ENABLE, 0, 0, MPU_STRONG_ORDERED_SHARED | MPU_PRIV_RW_USER_RW },
    { MPU_REGION_ENABLE, 0, 0, MPU_STRONG_ORDERED_SHARED | MPU_PRIV_RW_USER_RW },
    { MPU_REGION_ENABLE, MPU_REMAP_ADDR, MPU_REGION_2MB, MPU_NORM_NSHARED_WB_NWA | MPU_PRIV_RO_USER_RO },
    { MPU_REGION_ENABLE,  MPU_PFLASH_ADDR, MPU_REGION_2MB, MPU_NORM_OUT_POLICY_WB_NWA |   MPU_NORM_IN_POLICY_NCACHE |
    MPU_PRIV_RO_USER_RO },
    { MPU_REGION_ENABLE, MPU_DFLASH_ADDR, MPU_REGION_128KB, MPU_NORM_OUT_POLICY_WB_NWA |MPU_NORM_IN_POLICY_NCACHE |
    MPU_PRIV_RW_USER_RW },
    { MPU_REGION_ENABLE, MPU_SNOR_ADDR, MPU_REGION_256MB, MPU_NORM_OUT_POLICY_WB_NWA |MPU_NORM_IN_POLICY_NCACHE |
    MPU_PRIV_RO_USER_RO },
    { MPU_REGION_ENABLE, MPU_PERI_ADDR, MPU_REGION_16MB, MPU_STRONG_ORDERED_SHARED | MPU_PRIV_RW_USER_RW },
    { MPU_REGION_ENABLE, MPU_PFLASHCON_ADDR, MPU_REGION_64KB, MPU_STRONG_ORDERED_SHARED | MPU_PRIV_RW_USER_RW },
    { MPU_REGION_ENABLE, MPU_PFLASH_OPTAREA_ADDR, MPU_REGION_16KB, MPU_STRONG_ORDERED_SHARED |    MPU_PRIV_RW_USER_RO },
    { MPU_REGION_ENABLE, MPU_PFLASH_INT_ADDR, MPU_REGION_32B, MPU_STRONG_ORDERED_SHARED | MPU_PRIV_RW_USER_RW },
    { MPU_REGION_ENABLE, MPU_DFLASHCON_ADDR, MPU_REGION_64KB, MPU_STRONG_ORDERED_SHARED |     MPU_PRIV_RW_USER_RW },
    { MPU_REGION_ENABLE, MPU_DFLASH_OPTAREA_ADDR, MPU_REGION_2KB, MPU_STRONG_ORDERED_SHARED | MPU_PRIV_RW_USER_RO },
    { MPU_REGION_ENABLE, MPU_DFLASH_INT_ADDR, MPU_REGION_32B, MPU_STRONG_ORDERED_SHARED | MPU_PRIV_RW_USER_RW },
    { MPU_REGION_ENABLE, MPU_SFMC_ADDR, MPU_REGION_4KB, MPU_STRONG_ORDERED_SHARED | MPU_PRIV_RW_USER_RW },
};
```

The TCC70xx MCU BSP sets the attribute of each region. After setting the attribute, you can access region only by the attribute that is set by TCC70xx MCU BSP. For example, if you set S-NOR address attribute to read-only, you cannot write to S-NOR. To check the setting of MPU, refer to `MPU_Init()` in "tcc70xx_mcu_bsp/sources/dev.drivers/mpu/TCC70xx/mpu.c". For more detailed information on MPU, refer to ARM web site: https://developer.arm.com/documentation/den0042/a/The-Memory-Protection-Unit?lang=en

7.6 Use Fixed DMA Buffer

To use a DMA buffer in non-cacheable area, you should check the size of the DMA buffer first before using it. Then, you can configure the size by modifying the definition in linker script file (.ld) as shown below. By default, in TCC70xx MCU BSP, the size of DMA buffer area is set to 16 KB and the size of SRAM area is set to 512 KB. Refer to the default DMA size of TCC70xx MCU BSP below.

- Definition for DMA size (GCC Environment)

```
tcc70xx_mcu_bsp/build/TCC70xx/gcc/linker_512_withPreload.ld

DMA_NC_SIZE      = 0x4000;
CAN_NC_SIZE      = 0x2000;
SRAM_TOTAL_SIZE  = 0x80000; /* 512KB */
ARM_STACK_SIZE   = 0x680;
STACK_BASE_ADDR  = __VECTOR_START + SRAM_TOTAL_SIZE - DMA_NC_SIZE - CAN_NC_SIZE;
```

- Definition for DMA size (Green Hills IDE)

The TCC70xx MCU BSP sets DMA area to non-cacheable as much as DMA size. If you use Green Hills IDE, you need to modify the predefinition (`DMA_NC_SIZE`, default value:0x4000) as follows. This predefinition is used in linker script file (.ld). If you use Green Hills IDE, refer to Define Memory Layout (Green Hills IDE) in Chapter 7.3.

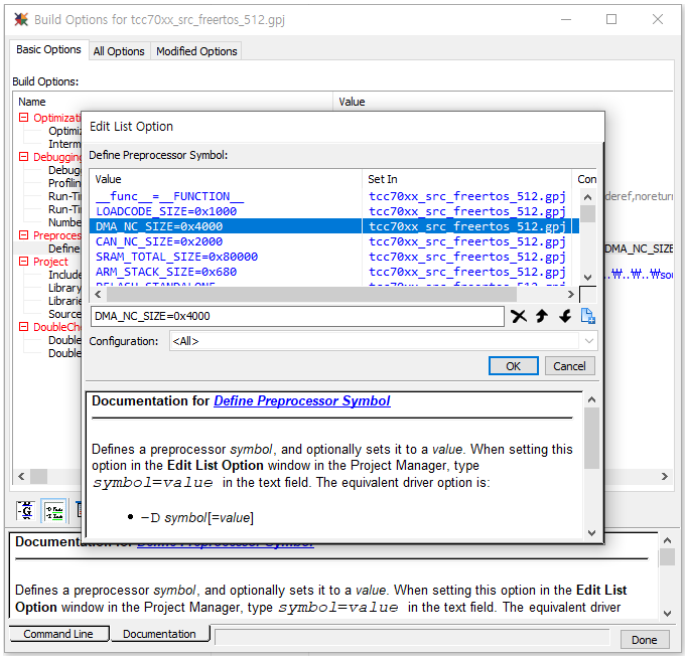

Figure 7.2 Predefinition for DMA (Green Hills IDE)

You can use the DMA buffer address that is returned from `MPU_GetDMABaseAddress()`.

- MPU Function to get DMA Base Address

```
tcc70xx_mcu_bsp/sources/dev.drivers/mpu/TCC70xx/mpu.c

uint32 MPU_GetDMABaseAddress( void )
{
    uint32  uiDMAStart;

    uiDMAStart= ( uint32 )( &__nc_dmastart );

    return uiDMAStart;
}
```

The `MPU_GetDMABaseAddress()` returns only the DMA based address. If you want to provide different types of DMA buffers

for your application, you need to implement the DMA buffer management depending on the use-cases.

Note: The TCC70xx MCU BSP does not limit how this DMA buffer area is partitioned. The use-case and total size of DMA buffer should be strictly classified.

7.7 Detailed View of SRAM

SRAM is categorized into seven areas as follows: .vector area, .data area, .bss area, User stack area, ARM stack area, Non-cacheable CAN message RAM area and Non-cacheable DMA area.

- The vector table is prepared for each ARM exception entry table.
- .bss (ZI) size and .data (RW) size vary depending on the function of application.
- The size of user stack area depends on user applications.
- The ARM stack area is reserved stack for each ARM execution mode, and it starts at the end of the Non-cacheable CAN message RAM area.
- The size of the non-cacheable CAN message RAM area is configured by redefining the `CAN_NC_SIZE` parameter.

The size of the non-cacheable DMA area is configured by redefining the `DMA_NC_SIZE` parameter.

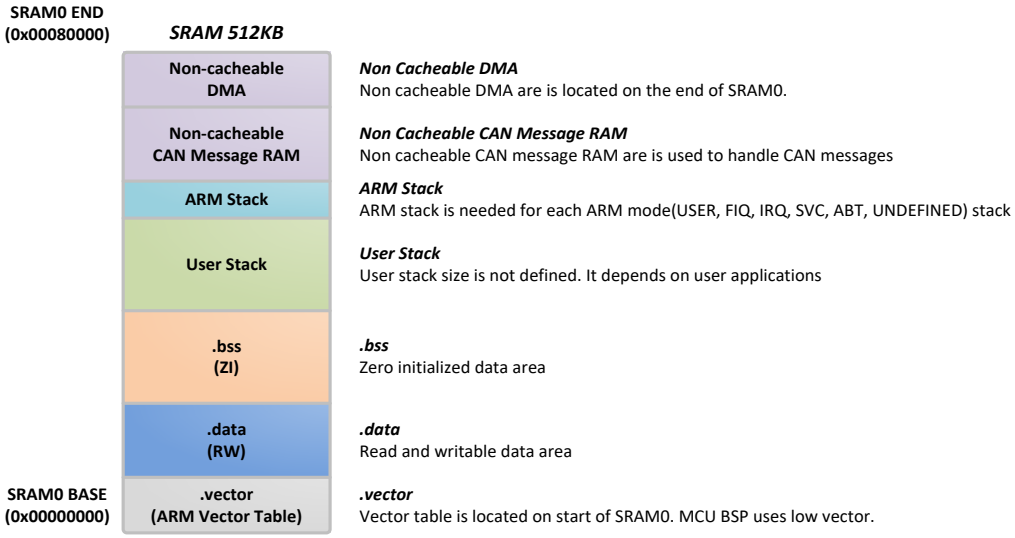

Figure 7.3 SRAM Map

8 MCU BSP RTOS

The TCC70xx MCU BSP can use Real Time Operating System (RTOS) function via System adaptation Layer (SAL) which originally supports simple RTOS such as FreeRTOS, uC/OS-III, and so on. The SAL of TCC70xx MCU BSP currently supports FreeRTOS and uC/OS-III. The TCC70xx MCU BSP uses FreeRTOS by default.

Figure 8.1 shows the position of the FreeRTOS in TCC70xx MCU BSP. For the uC/OS-III, the uC/OS-III is supportable based on the contract with Micrium [5].

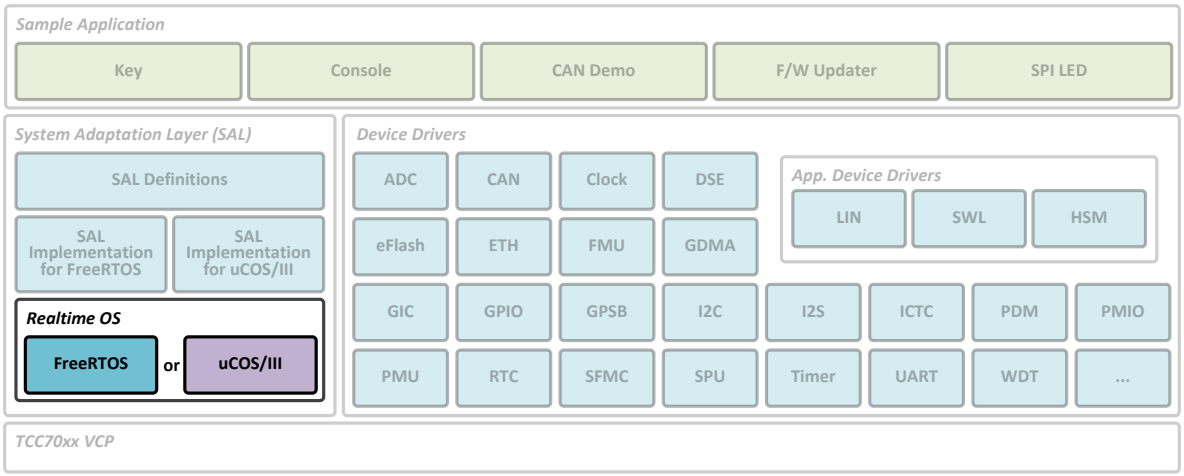

Figure 8.1 RTOS in TCC70xx MCU BSP

The SAL supports simple Operating System (OS) functions such as task, queue, semaphore, and event flags management. The TCC70xx MCU BSP application uses some of the full OS functions and some of these functions are defined in "sal_com.h" file. The header file contains prototype of SAL APIs, error value, and some configuration definitions. This chapter describes FreeRTOS because the TCC70xx MCU BSP uses FreeRTOS by default.

8.1 Initiation Procedure

The following procedure initiates variables and necessary internal tasks for initializing OS:

1. Generate User Start Task: To generate the first task after starting OS, the system calls `Main_StartTask()` in the "tcc70xx_mcu_bsp/sources/app.sample/app.base/main.c".
2. Start OS: The MCU BSP RTOS is operated by calling `SAL_OsStart()`. This function sets timer0 for OS tick. OS tick is a periodic interrupt by a timer and enables the OS to execute the priority-based task schedules. OS Tick interval is 1 ms.
   Refer to `vConfigureTickInterrupt()` in "tcc70xx_mcu_bsp/sources/os/freertos/portable/ARM_CR5".
3. Generate user application task: You can add a function to generate task for each application.

Refer to `AppTaskCreate()` in "tcc70xx_mcu_bsp/sources/app.sample/app.base/main.c".

8.2 SAL

SAL is required for implementation according to the OS. If you want to use FreeRTOS, the SAL should be implemented as a function supported by FreeRTOS in "sal_freertos_impl.c" file. For more detailed information on SAL, refer to "*TCC70xx MCU BSP-API Specification for System Adaptation Layer*" [3].

8.3 FreeRTOS OS Kernel

The TCC70xx MCU BSP uses FreeRTOS v10.3.1 and you can download FreeRTOS v10.3.1 from FreeRTOS download page [4].
FreeRTOS OS Kernel layer contains FreeRTOS kernel codes that should be used via SAL APIs. Telechips does not guarantee any modification of FreeRTOS kernel codes.

8.3.1 FreeRTOS Directory Structure

FreeRTOS of TCC70xx MCU BSP is structured as follows:

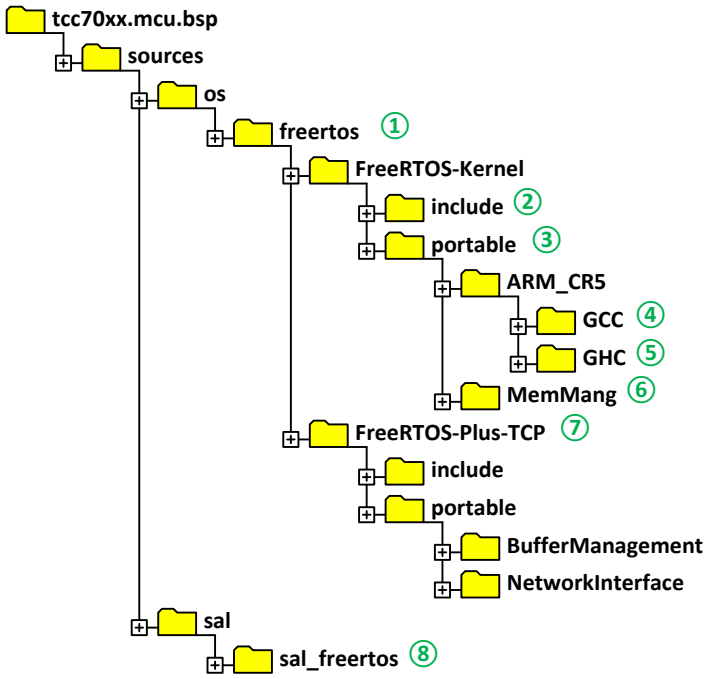

Figure 8.2 Directory Structure of FreeRTOS

① The core FreeRTOS kernel files
② The core FreeRTOS kernel header files
③ Processor specific code
④ Supported ports for Cortex-R5 processor on GCC environments
⑤ Supported ports for Cortex-R5 processor on GHC environments
⑥ The sample heap Implementation
⑦ The core FreeRTOS TCP
⑧ SAL implementation for FreeRTOS

8.3.1.1 OS

The OS directory contains the OS specific directories as follows:

- **`freertos`**: This directory contains the FreeRTOS specific kernel files, which are called as "tasks.c", "queue.c", "list.c", and "event_group.c". These files are required for FreeRTOS kernel operation.
  - **`include:`** This directory contains the FreeRTOS kernel header files including "FreeRTOSConfig.h". You can customize FreeRTOS via "FreeRTOSConfig.h" file.
  - **`portable`**: This directory contains specific processor and builds environment-related codes. These codes can be modified to use FreeRTOS according to the development environment.
    - **`ARM_CR5`**: The TCC70xx MCU BSP operates based on Cortex-R5. FreeRTOS supports the "port.c" and "portASM.S" files. You can port FreeRTOS by modifying these files.
    - **`MemMang`**: FreeRTOS supports memory allocation implementations which have five sample codes: heap_1,

heap_2, heap_3, heap_4, and heap_5. For more information on memory allocation implementations, refer to the FreeRTOS web page or "ReadMe" file in this directory. However, the TCC70xx MCU BSP uses **MemMang** only for queue buffer allocation case. Therefore, you must not use the memory allocation.

8.3.1.2 SAL

The SAL directory contains the SAL-related codes.

- **sal_freertos**: This directory contains the "sal_freertos_impl.c" file. The APIs in this file call FreeRTOS specific kernel function. SAL can use FreeRTOS kernel function via these APIs and you can use the SAL.

For detailed information on SAL, refer to "*TCC70xx MCU BSP-API Specification for System Adaptation Layer*"[3].

8.3.2 Configuration for FreeRTOS

You should configure a Makefile to use FreeRTOS. The following shows configuration of Makefile in GCC environment.

- Makefile Flags for FreeRTOS

```
tcc70xx_mcu_bsp/build/TCC70xx/gcc/Makefile

# Build Flags
MCU_BSP_BUILD_FLAGS_TARGET_OS_FREERTOS   ?= 1
MCU_BSP_BUILD_FLAGS_TARGET_OS_UCOS       ?= 0
```

In Green Hills IDE, you can use the GreenHills project for FreeRTOS.

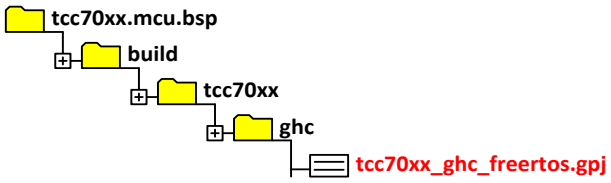

Figure 8.3 GreenHills Project for FreeRTOS

8.3.3 Configuration for Task Priorities

Each task is assigned by a priority from 0 to 'configMAX_PRIORITIES - 1' and configMAX_PRIORITIES is defined in "FreeRTOSConfig.h". The default value of configMAX_PRIORITIES is 16.
In the case of FreeRTOS, the priority number represents the task priority. For example, if the task priority is 0, it means the lowest task priority. The task priority can be duplicated without any restrictions, and the same number means the same task priority. These task priorities are defined in "sal_com.h" file. Each task priority can be adjusted according to your system. The following shows the task priorities of FreeRTOS.

- RTOS Task Priorities

tcc70xx_mcu_bsp/sources/sal/sal_com.h

```
typedef enum _SALTaskPriority_
{
    SAL_PRIO_APP_CFG        = 2,      /**< APP TASK START task          */
    SAL_PRIO_POWER_MANAGER  = 3,      /**< Power Manager APP task    */
    SAL_PRIO_IPC_CONTROL    = 3,      /**< IPC Control task             */
    SAL_PRIO_SYSTEM_MON     = 4,      /**< System Monitoring APP task      */
    SAL_PRIO_CAN_DEMO       = 4,      /**< CAN DEMO APP task         */
    SAL_PRIO_AUDIO_TEST     = 4,      /**< Audio Test APP task          */
    SAL_PRIO_VSDR_MONITOR   = 5,      /**< Vehicle Signal DEMO APP task */
    SAL_PRIO_SPILED_DEMO    = 6,      /**< SPILED Demo task             */
    SAL_PRIO_IPC_PARSER     = 6,      /**< IPC Parser task           */
    SAL_PRIO_KEY_APP        = 6,      /**< KEY APP task                 */
    SAL_PRIO_UART_TEST      = 6,      /**< UART Test APP task           */
    SAL_PRIO_LIN_TEST       = 6,      /**< Lin Test APP task            */
    SAL_PRIO_ETH_TASK       = 6,      /**< Ethernet TX/RX task          */
    SAL_PRIO_CONSOLE_DEMO   = 6,      /**< Console DEMO APP task        */
    SAL_PRIO_NVM_MAMAGER    = 8,      /**< NVM Manager APP task         */
    SAL_PRIO_FWUG_APP       = 10,     /**< FWUG APP task             */
    SAL_PRIO_LOWEST         = 12,     /**< APP TASK START task          */
} SALTaskPriority_t;
```

8.3.4 Customizing "FreeRTOSConfig.h" File

FreeRTOS is customized by using the configuration file, "FreeRTOSConfig.h". This file is located in the path src/OS/freertos/include. The following shows the definitions configured for FreeRTOS.

- FreeRTOS configuration File

```
tcc70xx_mcu_bsp/sources/os/freertos/FreeRTOS-Kernel/include/FreeRTOSConfig.h

#define    configMAX_API_CALL_INTERRUPT_PRIORITY       0  // (16 -2)

#define    configCPU_CLOCK_HZ                          100000000UL
#define    configUSE_PORT_OPTIMISED_TASK_SELECTION     0
#define    configUSE_TICKLESS_IDLE                     0
#define    configTICK_RATE_HZ                          ( (TickType_t ) 1000 )
#define    configPERIPHERAL_CLOCK_HZ                   ( 33333000UL )
#define    configUSE_PREEMPTION                        1
#define    configUSE_IDLE_HOOK                         0

#ifdef TCC_FREERTOS_DEMO
    #define    configUSE_TICK_HOOK                     1  // TCC_DEMO
#else
    #define    configUSE_TICK_HOOK                     0  // org
#endif

#define    configMAX_PRIORITIES                        ( 16 )
#define    configMINIMAL_STACK_SIZE                    ( (unsigned short ) 250 )

/* Large in case configUSE_TASK_FPU_SUPPORT is 2 in which case all tasks have an FPU context. */
#ifdef TCC_FREERTOS_DEMO
    #define    configTOTAL_HEAP_SIZE                   ( 100 * 1024 )
#else
    #define    configTOTAL_HEAP_SIZE                   ( 1 * 1024 )
#endif

#define    configMAX_TASK_NAME_LEN                     ( 30 )
#define    configUSE_TRACE_FACILITY                    0
#define    configUSE_16_BIT_TICKS                      0
#define    configIDLE_SHOULD_YIELD                     1
#define    configUSE_MUTEXES                           1
#define    configQUEUE_REGISTRY_SIZE                   8
#define    configCHECK_FOR_STACK_OVERFLOW              2
#define    configUSE_RECURSIVE_MUTEXES                 1
#define    configUSE_MALLOC_FAILED_HOOK                0
#define    configUSE_APPLICATION_TASK_TAG              0
#define    configUSE_COUNTING_SEMAPHORES               1
#define    configUSE_QUEUE_SETS                        1
#define    configUSE_TASK_NOTIFICATIONS                1
```

FreeRTOS is defined in "FreeRTOSConfig.h" file. Refer to "*Customization section*" in FreeRTOS web page for more information.

8.3.5 Memory Resource Function

FreeRTOS uses standard library such as `memset()`, `memcpy()`, `strcpy()`, and so on.
Refer to "tcc70xx_mcu_bsp/sources/sal/sal_freertos/sal_freertos_impl.c" file.

8.3.6 Exception Handler

Exception handlers are defined in "tcc70xx_mcu_bsp/sources/core/GCC/vector.S" file in the case of GCC environment. The handlers below are executed when an exception occurs. The following shows the defined exception handlers.

- Vector Table for ARM Exception

tcc70xx_mcu_bsp/sources/core/GCC/vector.S

```
_vector:
    ldr pc, =reset_handler
    ldr pc, =ARM_ExceptUndefInstrHndlr
    ldr pc, =ARM_ExceptSwiHndlr
    ldr pc, =ARM_ExceptPrefetchAbortHndlr
    ldr pc, =ARM_ExceptDataAbortHndlr
    ldr pc, =arm_reserved
    ldr pc, =ARM_ExceptIrqHndlr
    ldr pc, =ARM_ExceptFiqHndlr
```

The exception handlers vary depending on the OS. For example, when an undefined exception occurs, `FreeRTOS_ARM_ExceptUndefinestrHndlr()` handler is executed for FreeRTOS. The following shows the implemented exception handlers.

- SAL Exception Handler

tcc70xx_mcu_bsp/sources/sal/sal.S

```
/*********************************************************************************************************
*****
*                           UNDEFINED INSTRUCTION EXCEPTION HANDLER
*********************************************************************************************************
****/

ARM_ExceptUndefInstrHndlr:
#ifdef OS_FREERTOS
    B   FreeRTOS_ARM_ExceptUndefInstrHndlr
#else
    B   OS_CPU_ARM_ExceptUndefInstrHndlr
#endif
```

9 USE TECHNOLOGY BY THE THIRD PARTY WITHIN MCU BSP

Technologies of Telechips may include the technologies of the third party (for example, uC/OS-III). The third party is responsible for all or any of intellectual property licenses for any special, indirect, incidental, or consequential damage or loss whatsoever resulting from the use of this application for other purposes. Technologies are provided as they are without any other implied or express warranty. The third party does not provide any implied warranty including suitability or marketability for a specific use.

10 REFERENCES

[1] Contact Telechips for more details: sales@telechips.com
[2] TCC70xx Full Specification
[3] TCC70xx MCU BSP-API Specification for [Driver]
[4] http://www.freertos.org/a00104.html
[5] http://www.micrium.com/

11 REVISION HISTORY

Rev. 1.00: 2023-03-31

- Updated
  - Chapter 4: Figure 4.3
  - Chapter 7.3: Linker script file
  - Chapter 7.4: Linker script file

Rev. 0.10: 2021-09-03

- Preliminary version release

DISCLAIMER

All information and data contained in this material are without any commitment, are not to be considered as an offer for conclusion of a contract, nor shall they be construed as to create any liability. Any new issue of this material invalidates previous issues. Product availability and delivery are exclusively subject to our respective order confirmation form; the same applies to orders based on development samples delivered. By this publication, Telechips, Inc. does not assume responsibility for patent infringements or other rights of third parties that may result from its use.

Further, Telechips, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.
No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of Telechips, Inc.
This product is designed for general purpose, and accordingly customer be responsible for all or any of intellectual property licenses required for actual application. Telechips, Inc. does not provide any indemnification for any intellectual properties owned by third party.
Telechips, Inc. cannot ensure that this application is the proper and sufficient one for any other purposes but the one explicitly expressed herein. Telechips, Inc. is not responsible for any special, indirect, incidental, or consequential damage or loss whatsoever resulting from the use of this application for other purposes.

COPYRIGHT STATEMENT

Important Notice

This material may include technology owned by the 3rd party licensor and the technology may be subject to its associated licenses. It is solely customer's responsibility to identify and comply with such licenses. No other licenses are granted or implied by Telechips with making available this material.

For customers who use licensed Codec ICs and/or licensed codec firmware of mp3:
"Supply of this product does not convey a license nor imply any right to distribute content created with this product in revenue-generating broadcast systems (terrestrial. Satellite, cable and/or other distribution channels), streaming applications (via internet, intranets and/or other networks), other content distribution systems (pay-audio or audio-on-demand applications and the like) or on physical media (compact discs, digital versatile discs, semiconductor chips, hard drives, memory cards and the like). An independent license for such use is required. For details, please visit http://mp3licensing.com".

For customers who use other firmware of mp3:
"Supply of this product does not convey a license under the relevant intellectual property of Thomson and/or Fraunhofer Gesellschaft nor imply any right to use this product in any finished end user or ready-to-use final product. An independent license for such use is required. For details, please visit http://mp3licensing.com".

For customers who use Digital Wave DRA solution:
"Supply of this implementation of DRA technology does not convey a license nor imply any right to this implementation in any finished end-user or ready-to-use terminal product. An independent license for such use is required."

For customers who use DTS technology:
"This product made under license to certain U.S. patents and/or foreign counterparts."
"© 1996 – 2011 DTS, Inc. All rights reserved."

For customers who use Dolby technology:
"Supply of this Implementation of Dolby technology does not convey a license nor imply a right under any patent, or any other industrial or intellectual property right of Dolby Laboratories, to use this Implementation in any finished end-user or ready-to-use final product. It is hereby notified that a license for such use is required from Dolby Laboratories."

For customers who use Google technology:
"Copyright © 2013 Google Inc. All rights reserved."

For customers who use MS technology:
"This product is subject to certain intellectual property rights of Microsoft and cannot be used or distributed further without the appropriate license(s) from Microsoft."