应用指南

从 Cortex-M4 到芯来 N308
应用移植说明
V1.0

目录

1、 概述

CM32M4xxR 是芯昇科技 RSIC-V MCU 系列，该系列采用芯来 N308 内核，并搭载丰富的片上外设功能。本文档针对如何将程序从 Cortex-M4 内核移植到 CM32M4xxR 的 N308 内核的一些关键点进行了总结，以期帮助用户快速完成芯片替换和软件移植的工作。

2、 移植步骤

2.1 工程准备

首先需要安装芯来官方 IDE 工具——Nuclei Studio IDE，安装完成后导入 SDK 中提供的模板工程，路径为：CM32M4xxR_SDK_1.0.0\CMIOT.CM32M4xxR_Library\Projects\CM32M4xxR_LQFP128_STB\Templates\Baremetal。模板工程中已经关联了 Drivers 中的 NMSIS 和 BSP 目录，应用程序可以直接使用。

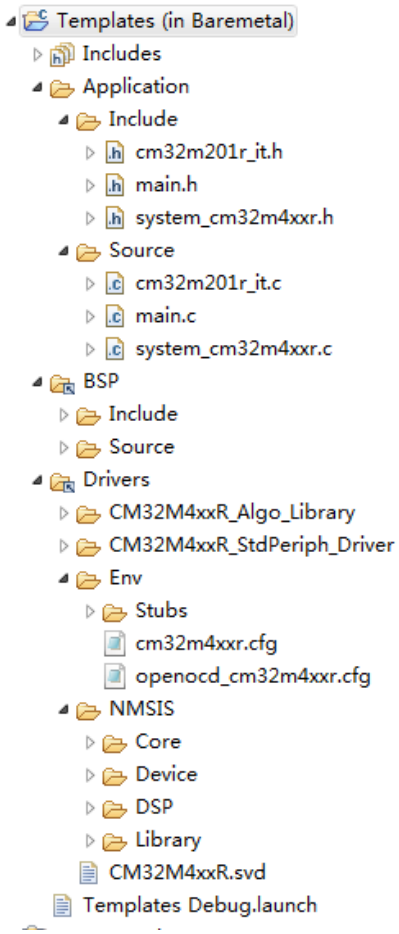

图 1 导入模板工程

基于此工程用户可以添加应用程序代码，构建自己的应用工程。

2.2 芯片相关文件替换

芯片相关文件的替换主要包括 NMSIS 的通用文件和 CM32M4xxR 的处理器文件。NMSIS 是芯来基于 ARM CMSIS 修改的一套适配 RSIC-V 的处理器硬件抽象层，最大程度保持了与 CMSIS 的兼容。在应用移植的过程中用户需要对 CMSIS 的内容进行替换，并修改应用中对其的引用，如果应用中有直接使用到 CMSIS 的接口，需要逐一在 NMSIS 中确认其兼容性并进行替换。

CM32M4xxR 的处理器文件是根据 NMSIS 提供的模板进行实现的，也需要检查替换，并且修改所有对其的引用。

关于 NMSIS 的详细说明，用户可以参考芯来提供的官方说明文档 https://doc.nucleisys.com/nmsis/introduction/introduction.html 。

关于芯来 NMSIS/Core 下的文件简要说明如下：

core_compatiable.h	兼容 ARM 的 API 定义头文件，包括存储器屏障。饱和运算和数据处理接口
core_feature_base.h	芯来 N/NX 系列核的基本特性 API，定义了所有核的共用寄存器和访问函数。
core_feature_cache.h	芯来 N/NX 系列核的 Cache API，定义了 I-Cache 和 D-Cache 操作接口
core_feature_dsp.h	芯来 N/NX 系列核的 DSP API，主要定义了 SIMD 指令。
core_feature_eclic.h	芯来 N/NX 系列核的 eclic API，用于中断配置。
core_feature_fpu.h	芯来 N/NX 系列核的 FPU API，用于浮点运算。
core_feature_pmp.h	芯来 N/NX 系列核的 PMP API，用于物理内存保护。
core_feature_timer.h	芯来 N/NX 系列核的系统 timer API，用于配置系统节拍及产生软件中断和软件复位。
nmsis_compiler.h	编译器通用头文件，目前只支持 gcc
nmsis_core.h	NMSIS 核通用头文件（顶层文件）
nmsis_gcc.h	gcc 编译器头文件
nmsis_version.h	NMSIS 版本定义
riscv_bits.h	bit 操作相关指令定义
riscv_encoding.h	处理器和寄存器赋值所用到的宏定义

关于 CM32M4xxR 的处理器文件简要说明如下：

cm32m4xxr_conf.h	外设启用配置等
cm32m4xxr_def.h	通用类型定义头文件
cm32m4xxr.h	中断向量、外设寄存器、处理器相关宏的定义。
nuclei_sdk_soc.h	顶层头文件，包含了 cm32m4xxr_conf.h、cm32m4xxr_def.h 和 cm32m4xxr.h

system_cm32m4xxr.h	系统初始化实现（时钟、中断向量），启动文件里调用
system_cm32m4xxr.c	系统初始化头文件
intexc_cm32m4xxr.S	中断和异常处理文件
startup_cm32m4xxr.S	启动文件

2.3 中断移植

Cortex-M4 的中断控制器为 NVIC，而芯来 N308 的中断控制器为 ECLIC。二者有比较大的区别，移植过程中该部分需要进行修改。关于 ECLIC 的详细说明可以参考《Nuclei_N 级别指令架构手册中》，这里仅做简要总结：

（1）在硬件方面，Cortex-M4 由 NVIC 中断控制器统一管理异常、NMI 和中断，而 N308 的 ECLIC 仅管理中断，异常和 NMI 单独处理。

（2）在软件流程上，Cortex-M4 的异常和中断处理均是向量方式，而 N308 既支持向量处理，也支持非向量处理，二者中断的响应流程不同；

（3）ELCIC 需要根据单独配置中断源的电平或边沿属性，NVIC 中不需要配置；

（4）在中断服务函数上，由于 Cortex-M4 在处理异常和中断时硬件自动压栈和出栈，因此可以像普通函数一样编写中断服务函数，而 N308 没有硬件自动压栈和出栈机制，因此需要在中断服务函数中手动压栈和出栈，对中断服务函数的实现有特殊要求。该部分功能已经在 intexc_cm32m4xxr.S 中实现，非向量中断用户无需处理；向量中断需要添加 interrupt 修饰。一般情况写向量中断不支持嵌套，如果需要支持嵌套，则需要在中断处理函数中手动添加 SAVE_IRQ_CSR_CONTEXT()和 RESTORE_IRQ_CSR_CONTEXT()进行上下文的保存和恢复。

向量中断嵌套示例代码：

```
/**
 * @brief  TIM3 IRQ Handler Vector Interrupt
 * @note Vector interrupt does not support nesting.
 * If you need to support nesting, add SAVE_IRQ_CSR_CONTEXT() / RESTORE_IRQ_CSR_CONTEXT()
 * in the interrupt response.
 */
__INTERRUPT void TIM3_IRQHandler() {
    // Save necessary CSRs into variables for vector interrupt nesting
    SAVE_IRQ_CSR_CONTEXT();

```

```
11.     if (TIM_GetIntStatus(TIM3, TIM_INT_UPDATE) != RESET) {
12.         TIM_ClrIntPendingBit(TIM3, TIM_INT_UPDATE);
13.
14.         LedBlink(LED2_PORT, LED2_PIN);
15.     }
16.
17.     //Restore necessary CSRs from variables for vector interrupt nesting
18.     RESTORE_IRQ_CSR_CONTEXT();
19. }
```

ECLIC 的中断仲裁机制是由四个因素决定：中断 level、中断 priority、中断 ID 号、中断阈值，这四个因素的判定顺序是中断 level>中断 priority>中断 ID 号>中断阈值，其中前三个因素都是数字越大表明仲裁优先级高，最后一个因素只有当中断 level 的数值大于中断阈值时，中断才会生效。

NVIC 与 ECLIC 函数接口对比：

NVIC	ECLIC	ECLIC 功能说明
NVIC_SetPriorityGrouping	ECLIC_SetCfgNlbits	配置 level 域的 bit 数
NVIC_GetPriorityGrouping	ECLIC_GetCfgNlbits	配置 level 域的 bit 数
NVIC_EnableIRQ	ECLIC_EnableIRQ	中断使能
NVIC_GetEnableIRQ	ECLIC_GetEnableIRQ	获取中断使能状态
NVIC_DisableIRQ	ECLIC_DisableIRQ	中断去使能
NVIC_GetPendingIRQ	ECLIC_GetPendingIRQ	获取中断等待标识
NVIC_SetPendingIRQ	ECLIC_SetPendingIRQ	设置中断等待标识
NVIC_ClearPendingIRQ	ECLIC_ClearPendingIRQ	清除中断等待标识
NVIC_GetActive		
	ECLIC_SetPriorityIRQ	设置中断 Priority
	ECLIC_GetPriorityIRQ	获取中断 Priority
NVIC_SystemReset		
NVIC_SetVector	ECLIC_SetVector	设定中断处理地址
NVIC_GetVector	ECLIC_GetVector	获取中断处理地址
	ECLIC_SetMth	设定中断 level 阈值
	ECLIC_GetMth	获取中断 level 阈值
	ECLIC_SetTrigIRQ	设置中断源的电平或边沿属性

	ECLIC_GetTrigIRQ	获取中断源的电平或边沿属性
	ECLIC_SetShvIRQ	修改中断向量/非向量处理模式
	ECLIC_GetShvIRQ	修改中断向量/非向量处理模式
NVIC_SetPriority	ECLIC_SetCtrlIRQ	统一设置中断 level 和 priority
NVIC_GetPriority	ECLIC_GetCtrlIRQ	统一获取中断 level 和 priority
	ECLIC_SetLevelIRQ	设置中断 level
	ECLIC_GetLevelIRQ	获取中断 level
	ECLIC_GetInfoCtlbits	获取 level 和 priority 的总 bit 域个数，返回值为 4，硬件配置无法修改。

NVIC 中断代码示例：

```
/* Configure the preemption priority and subpriority:
   - 1 bits for pre-emption priority: possible value are 0 or 1
   - 3 bits for subpriority: possible value are 0...7
   - Lower values gives higher priority
*/
NVIC_PriorityGroupConfig(0x600);

/*Set key input interrupt priority*/
NVIC_InitStructure.NVIC_IRQChannel                   = KEY_INPUT_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority        = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd                = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

N308 移植代码修改：

```
/* Configure the ECLIC level and priority  Bits */
ECLIC_SetCfgNlbits(1);  /* 1 bits for level, 3 bits for priority */

/* Enable the Key Interrupt */
ECLIC_SetLevelIRQ(EXTI0_IRQn, 1);       //interrupt level 1
ECLIC_SetPriorityIRQ(EXTI0_IRQn, 1);    //interrupt priority 0
ECLIC_SetTrigIRQ(EXTI0_IRQn, ECLIC_LEVEL_TRIGGER);  //level interrupt
ECLIC_EnableIRQ(EXTI0_IRQn);            //Enable interrupt
```

2.4 异常和 NMI 移植

由于两个内核的架构差异，其异常类型也有较大的差异，用户需要根据 N308 的特性对异常和 NMI 的处理代码进行重构。

下表对比了 CortexM4 和 N308 的异常和 NMI 说明：

CortexM4	**说明**	**N308**	**说明**
RESET 复位	复位为特殊类型的异常		
NMI	非可屏蔽中断	NMI	非可屏蔽中断
HardFault	硬件异常		
MemMange	内存管理异常	指令访问错误 读存储器访问错误 写存储器和 AMO 访问错误	取指令访问错误 Load 指令访存错误 Store 或 AMO 指令访存错误
BusFault	总线故障异常		
UsageFault	指令执行异常	非法指令 读存储器地址非对齐 写存储器和 AMO 地址非对齐	非法指令 Load 指令访存地址非对齐 Store 或 AMO 指令访存地址非对齐
SVCall	SVC 指令触发异常	用户模式环境调用 机器模式环境调用	UserMode 下执行 ecall 指令 MachineMode 下执行 ecall 指令
PendSV	中断驱动的系统级服务请求		
SysTick	Systick 异常		
Interrupt（ISR）	外设中断		
		断点	断点异常，调试器使用。

N308 的异常和 NMI 的处理函数通过 system_cm32m4xxr.c 文件中的 Exception_Register_EXC、core_exception_handler 和 Exception_Get_EXC 三个函数进

行注册、获取和执行。用户只需要关心注册和获取注册两个函数的使用即可。

示例代码如下：

```
1.  /**
2.   * @brief IllegalAccess Exception Handler.
3.   */
4.  void IllegalAccess_Exception_Handler(unsigned long mcause, unsigned long sp)
5.  {
6.      printf("illegal memory @ 0x%08X access!\r\n", __RV_CSR_READ(CSR_MBADADDR));
7.      while(1);
8.  }
9.
10. /**
11.  * @brief Ecall(System Software Call) Handler
12.  */
13. void ECALL_Exception_Handler(unsigned long mcause, unsigned long sp)
14. {
15.     uint32_t saved_regs = sp;
16.     uint32_t mepc = ((uint32_t *)saved_regs)[12];
17.     printf("ECALL Exception Trigger\r\n");
18.     /* Since the mepc is the ecall address when the exception is returned,
19.      * there will be an endless loop in the return. The ecall is a
20.      * 4-byte instruction. Modify the mepc to try to skip it. */
21.     ((uint32_t *)saved_regs)[12] = mepc + 4;
22. }
23.
24. /**
25.  * @brief  Main function.
26.  */
27. int main(void) {
28.     /* SystemInit() function has been called by startup file startup_cm32m4xxr.s */

29.     log_init();
30.
31.     /* Register exception handling functions for system calls and illegal access */
32.     Exception_Register_EXC(MmodeEcall_EXCn, (unsigned long)ECALL_Exception_Handler);

33.     Exception_Register_EXC(LdFault_EXCn, (unsigned long)IllegalAccess_Exception_Hand
ler);
34.
35.     /* Call ecall to enter exception mode */
36.     __ECALL();
37.     printf("ECALL Process Completed! \r\n");
38.
39.     /* Attempt to access an illegal address to trigger an exception */
40.     uint32_t trap = *(uint32_t *)0xFFFFFFFF;
41.     printf("trap : %d\r\n",trap);
42.
43.     while (1) {
44.
45.     }
46. }
```

2.5 内核 Timer 移植

CortexM4 内核包含一个 24 位向下计数的 SysTick 定时器，时钟源可以选择 AHB/8 或者 AHB，当计数到 0 时，将从 reload 寄存器中自动重装载定时器初值。

N308 内核同样提供一个定时器 mtimer，时钟源可以选择 RTCCLK 或者 HCLK，可以替代 SysTick。该计数器为 64 位向上计数，默认开启，可以一直计数无需重载，内置一个比较寄存器和软件中断寄存器，可用于产生计时器中断和软件中断。

关于内核 timer 的详细说明可以参考：《Nuclei_N 级别指令架构手册中》。

N308 内核 timer 配置代码示例：

```
uint32_t SystimerCtrl;

/* use HCLK as system timer clock */
SystimerCtrl = SysTimer_GetControlValue();
SysTimer_SetControlValue(SystimerCtrl | SysTimer_MTIMECTL_CLKSRC_Msk);

SysTimer_SetLoadValue(0);                 //set the timer load value
SysTimer_SetCompareValue(ticks);          //set the compare value
ECLIC_SetShvIRQ(SysTimer_IRQn, ECLIC_NON_VECTOR_INTERRUPT);   //set the system timer compare interrupt
ECLIC_SetLevelIRQ(SysTimer_IRQn,  0);    //set interrupt level
ECLIC_EnableIRQ(SysTimer_IRQn);          //enable interrupt
```

N308 内核 timer 使用示例：

```
/**
 * @brief  Function of delay nus.
 * @param  u32 nus:Delay number of us.
 */
void delay_us(uint32_t nus)
{
    uint64_t temp;
    uint64_t cmpvalue;

    cmpvalue = SysTimer_GetLoadValue() + nus*(SystemCoreClock / 1000000);

    do
    {
        temp = SysTimer_GetLoadValue();
    } while (temp < cmpvalue); // waiting time arrives
}
```

2.6 移植 MCU 外设功能

CM32M4xxR 的外设功能与 MCU 的外设的差异无法尽述，具体可以自行查阅芯片手册进行比对。

在模板工程中，所有外设的驱动位于 Drivers 目录下，可以通过#include “nuclei_sdk_soc.h”文件，调用所有驱动接口。

2.7 其他移植

移植过程中，还有一些方面需要关注：

1、 编译工具链选项和语法扩展移植

CM32M4xxR 目前提供使用 RSIC-V GNU 编译工具链进行编译，由于编译工具链的不同，如果应用代码中使用的编译选项和编译器语法扩展，也需要逐项进行修改。例如：ARM Compiler 5 支持__forceinline 语法，但是在使用 GNU 编译工具时需要修改为__attribute__((always_inline))。

RSIC-V GNU 工具链手册请参考：https://gcc.gnu.org/onlinedocs/9.2.0/

2、 内嵌汇编代码移植

由于内核平台的变化，其汇编语言也不相同。如果应用代码中使用了内嵌汇编代码，需要逐项进行修改。

RISC-V 的汇编说明可以参考：https://riscv.org/technical/specifications/

3、 位带操作移植

N308 目前不支持位带（Bit Band）操作，但是支持原子操作指令，原子操作指令也具有不可分割的特性，同样适用于多任务场景。原子操作指令包括加、与、或、异或等操作，具体可以参考 NMSIS core_feature_base.h 文件中的__AMO 相关函数。

4、 链接脚本移植

SDK 的中提供了链接脚本 gcc_cm32m4xxr_flashxip.ld 文件，一般情况下可以直接使用，如应用程序中需要改变输出文件的存储布局，则需要对 LD 文件进行修改。

RSIC-V GNU LD 手册请参考：https://sourceware.org/binutils/docs/ld/

3、 版本历史

版本	日期	修改内容
V1.0	20211020	新建