FPGA Cortex-M1 软核编程应用指南

(AN02005 V1.0)

(2020.10.28)

深圳市紫光同创电子有限公司

文档版本修订记录

版本号	发布日期	修订记录
V0.1	2019/09/19	初始版本
V0.2	2019/10/10	增加I2C、memory和FreeRTOS部分
V0.3	2019/12/12	增加DDR、Systick、打印函数、动态内存操作和Bootloader部分
V0.4	2020/03/02	增加Bootloader部分的ICACHE和DCACHE使用描述
V0.4-1	2020/03/09	增加DCACHE地址映射范围到16MB
V0.5	2020/03/25	硬件已经不支持DDR3外设访问，删掉DDR3模块相关描述，并且增加DCACHE地址映射范围到256MB
V0.6	2020/04/24	增加CACHE一致性接口函数
V0.7	2020/07/30	增加DMAC描述以及LwIP移植
V0.9	2020/09/28	增加用户自定义APB总线接口以及MAC地址配置接口
V1.0	2020/10/28	增加UDP硬件加速功能及接口配置函数，增加Timer1和UART1外设模块，增加GPIO支持中断数量

名词术语解释

Abbreviations 缩略语	Full Spelling 英文全拼	Chinese Explanation 中文解释
UART	Universal Asynchronous Receiver-Tranamitter	通用异步接收-发射器
Timer	Timer	定时器
WDT	Watchdog Timer	看门狗
GPIO	General Purpose I/O	通用输入输出接口
SPI	Serial Peripheral Interface	串行外设接口
I2C	Inter-Integrated Circuit	集成电路互联在线
FreeRTOS		开源实时操作系统
DMA	Direct Memory Access	可以指内存和外设直接存取数据
LwIP	Light weight IP	轻量化的TCP/IP 协议
UDP	User Datagram Protocol	用户数据报协议

目录

图目录

表目录

1 软件编程

1.1 Cortex-M1 软件编程库

PANGO Cortex-M1 软件编程库中提供的 Cortex-M1 软件驱动，如下表所示：

表 1 Cortex-M1 软件编程

文件	描述
startup_PANGO_M1.s	Cortex-M1 启动引导程序
core_cm1.h	Cortex-M1 内核定义
systick.c	Systick 函数定义
PANGO_M1.h	中断向量表、寄存器和地址映射定义
system_PANGO_M1.c	Cortex-M1 系统初始化和系统时钟定义
PANGO _M1_gpio.c	GPIO 驱动函数定义
PANGO _M1_timer.c	Timer 驱动函数定义
PANGO _M1_wdog.c	Watch Dog 驱动函数定义
PANGO _M1_uart.c	UART 驱动函数定义
PANGO _M1_spi.c	SPI 驱动函数定义
PANGO _M1_i2c.c	I2C 驱动函数定义
PANGO _M1_memory.c	Memory 驱动函数定义
PANGO _M1_printk.c	打印驱动函数定义
PANGO _M1_my_mem.c	动态内存驱动函数定义
PANGO _M1_it.c	中断向量表函数定义
PANGO _M1_icache.c	ICACHE 接口函数定义
PANGO _M1_dcache.c	DCACHE 接口函数定义
PANGO_M1_eth_dmac.c	DMAC 配置函数定义
PANGO_M1_eth_buffer.c	网络接收/发送接口函数定义
PANGO_M1_udp_speed_up.c	UDP 硬件加速接口函数定义

上表中的各个外设模块 gpio、timer0、timer1、wdog、uart0、uart1、spi、i2c、memory、systick、printk、my_mem、Ethernet 以及在线调试功能，均可在 RTL 代码中通过宏定义包含或不包含该模块，灵活实现，模块添加或删除的具体的宏定义接口描述以及定义请参考《FPGA Cortex-M1 SoC DEMO 应用指南》。

1.2 标准外设内存映射

PANGO Cortex-M1 标准外设内存映射地址如下表：

表 2 PANGO Cortex-M1 标准外设地址映射

标准外设	类型	地址映射	描述
ITCM		0x00000000	1KB，2KB，4KB，8KB，16KB，32KB，64KB，128KB，256KB，1MB
ICACHE		0x10000000	寻址空间 0x10000000-0x1fffffff
DTCM		0x20000000	1KB，2KB，4KB，8KB，16KB，32KB，64KB，128KB，256KB，1MB
DCACHE		0x30000000	寻址空间 0x30000000-0x3fffffff
TIMER0	TIMER_TypeDef	0x50000000	定时器 0
TIMER1	TIMER_TypeDef	0x50001000	定时器 1

UART0	UART_TypeDef	0x50004000	串口 0
UART1	UART_TypeDef	0x50005000	串口 1
Watch Dog	WDOG_TypeDef	0x50008000	看门狗
SPI	SPI_TypeDef	0x5000B000	串行外设接口
USER_APB		0x5000C300	用户自定义 APB 总线接口基地址
GPIO0	GPIO0_TypeDef	0x40000000	通用输入输出端口
I2C	I2C_TypeDef	0x5000A000	集成电路互联总线
ICACHE_CTRL	ICACHE_TypeDef	0x47000000	ICACHE 控制块基地址
DCACHE_CTRL	DCACHE_TypeDef	0x48000000	DCACHE 控制块基地址
DMAC0	DMAC0_TypeDef	0x49000000	DMAC0
BANK_RAM_BASE0		0x60000000	块 RAM0 基地址
BANK_RAM_BASE1		0x61000000	块 RAM1 基地址
BANK_RAM_BASE2		0x62000000	块 RAM2 基地址
BANK_RAM_BASE3		0x63000000	块 RAM3 基地址
BANK_RAM_BASE4		0x64000000	块 RAM4 基地址
BANK_RAM_BASE5		0x65000000	块 RAM5 基地址
BANK_RAM_BASE6		0x66000000	块 RAM6 基地址
BANK_RAM_BASE7		0x67000000	块 RAM7 基地址
BANK_RAM_BASE8		0x68000000	块 RAM8 基地址
BANK_RAM_BASE9		0x69000000	块 RAM9 基地址
BANK_RAM_BASE10		0x6A000000	块 RAM10 基地址
BANK_RAM_BASE11		0x6B000000	块 RAM11 基地址
BANK_RAM_BASE12		0x6C000000	块 RAM12 基地址
BANK_RAM_BASE13		0x6D000000	块 RAM13 基地址
BANK_RAM_BASE14		0x6E000000	块 RAM14 基地址
BANK_RAM_BASE15		0x6F000000	块 RAM15 基地址
UDP_SPEED_UP_BASE		0x70000000	UDP 硬件加速配置基地址

1.3 内核系统内存映射

PANGO Cortex-M1 内核系统内存映射地址如下表：

表 3 PANGO Cortex-M1 内核系统内存映射

系统控制	类型	地址映射	描述
SysTick	SysTick_Type	0xE000E010	SysTick configuration struct
NVIC	NVIC_BASE	0xE000E100	NVIC configuration struct
SCnSCB	SCnSCB_Type	0xE000E000	System control Register not in SCB
SCB	SCB_Type	0xE000ED00	SCB configuration struct

2 中断处理

PANGO Cortex-M1 提供最多 32 个用户可用的中断处理信号，用户可配置 1 或 8 或 16 或 32 个外部中断处理信号。支持 0-3 级可编程优先级（0 表示最高优先级）。PANGO Cortex-M1 中断控制器如下表所示：

表 4 PANGO Cortex-M1 中断控制器

地址	中断名称	中断号	描述
0x00000000	__initial_sp		Top of Stack
0x00000004	Reset_Handler		Reset Handler
0x00000008	NMI_Handler	-14	NMI Handler
0x0000000C	HardFault_Handler	-13	Hard Fault Handler
0x00000010	0		Reserved
0x00000014	0		Reserved
0x00000018	0		Reserved
0x0000001C	0		Reserved
0x00000020	0		Reserved
0x00000024	0		Reserved
0x00000028	0		Reserved
0x0000002C	SVC_Handler	-5	SVCall Handler
0x00000030	0		Reserved
0x00000034	0		Reserved
0x00000038	PendSV_Handler	-2	PendSV Handler
0x0000003C	SysTick_Handler	-1	SysTick Handler
0x00000040	UART0_Handler	0	UART0 Interrupt handler
0x00000044	UART1_Handler	1	UART1 Interrupt handler
0x00000048	TIMER0_Handler	2	TIMER0 Interrupt handler
0x0000004C	TIMER1_Handler	3	TIMER1 Interrupt handler
0x00000050	GPIO0_Handler	4	GPIO0 Interrupt handler
0x00000054	UARTOVF_Handler	5	UART0,1 Overflow Interrupt handler
0x00000058	ENT_Handler	6	Ethernet Interrupt handler
0x0000005C	I2C_Handler	7	I2C Interrupt handler
0x00000060	CAN_Handler	8	CAN Interrupt handler
0x00000064	RTC_Handler	9	RTC Interrupt handler
0x00000068	Interrupt10_Handler	10	Interrupt 10
0x0000006C	DTimer_Handler	11	DualTimer Interrupt handler
0x00000070	TRNG_Handler	12	TRNG Interrupt handler
0x00000074	Interrupt13_Handler	13	Interrupt 13
0x00000078	Interrupt14_Handler	14	Interrupt 14
0x0000007C	Interrupt15_Handler	15	Interrupt 15
0x00000080	Interrupt16_Handler	16	Interrupt 16
0x00000084	Interrupt17_Handler	17	Interrupt 17
0x00000088	Interrupt18_Handler	18	Interrupt 18
0x0000008C	Interrupt19_Handler	19	Interrupt 19
0x00000090	Interrupt20_Handler	20	Interrupt 20
0x00000094	Interrupt21_Handler	21	Interrupt 21
0x00000098	Interrupt22_Handler	22	Interrupt 22
0x0000009C	Interrupt23_Handler	23	Interrupt 23

0x000000A0	Interrupt24_Handler	24	Interrupt 24
0x000000A4	Interrupt25_Handler	25	Interrupt 25
0x000000A8	Interrupt26_Handler	26	Interrupt 26
0x000000AC	Interrupt27_Handler	27	Interrupt 27
0x000000B0	Interrupt28_Handler	28	Interrupt 28
0x000000B4	Interrupt29_Handler	29	Interrupt 29
0x000000B8	Interrupt30_Handler	30	Interrupt 30
0x000000BC	Interrupt31_Handler	31	Interrupt 31

3 UART

UART 全称为 Universal Asynchronous Receiver-Transmitter（通用异步接收-发射器）。嵌入式中说的串口，一般就是指 UART 口，此说法虽然不是特别严谨，但算是常见的习惯说法。

在传输的过程中，UART 发送端将字节数据以串行的方式逐个比特发送出去，UART 接收端逐个比特接收数据，然后将其重新组织为字节数据。常见的传输格式如下图所示：

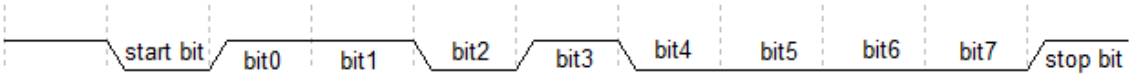

图 1 UART 数据传输格式

- 在空闲时，UART 输出保持高电平。
- 在发送 1 字节之前，应该先发送一个低电平起始位（Start Bit）
- 发送起始位之后，通常以低位先发送的方式逐个比特传输完一整个字节数据位（Data Bit）。当然，也有某些 UART 设备会以高位先发送的方式进行传输。
- 传输完字节以后，可选的会传输一位或者多位奇偶校验位（Parity Bit）
- 最后传输的是以高电平表征的停止位（Stop Bit）

3.1 特征

PANGO Cortex-M1 包含 1 个通过 APB 总线访问的通用异步收发器 UART，功能指标如下：

- 最大波特率 960Kbit/s
- 无奇偶校验位
- 8 位数据位
- 1 位停止位
- 支持两个串口外设：UART0 和 UART1

UART 结构如下图所示：

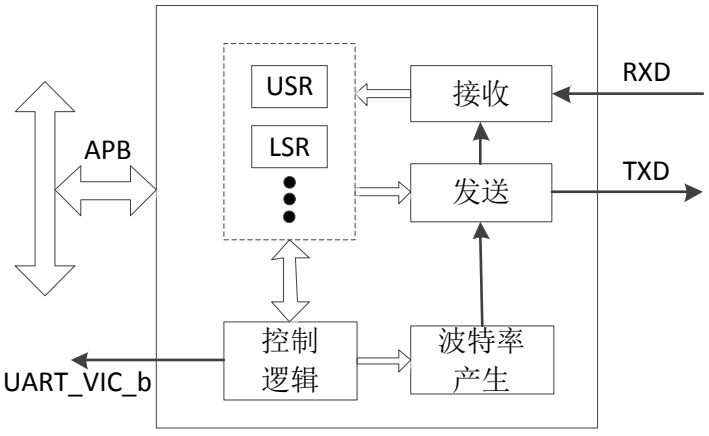

图 2 UART 结构图

3.2 寄存器列表

UART 的可配置寄存器为存储器地址映射寄存器（Memory Address Mapped），寄存器 ENABLE 信号均是高电平有效，寄存器列表如下表所示：

表 5 UART 寄存器列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
DATA	0x000	RW	8	0x--	[7:0] :RX/TX Data Value
STATE	0x004	RW	4	0x0	[3]:RX buffer overrun,write 1 to clear [2]:TX buffer overrun,write 1 to clear [1]:RX buffer full,read-only [0]:TX buffer full,read-only
CTRL	0x008	RW	7	0x00	[6]:High speed test mode for TX only [5]:RX overrun interrupt enable [4]:TX overrun interrupt enable [3]:RX interrupt enable [2]:TX interrupt enable [1]:RX enable [0]:TX enable
INTSTATUS/ INTCLEAR	0x00C	RW	4	0x0	[3]:RX overrun interrupt,write 1 to clear [2]:TX overrun interrupt,write 1 to clear [1]:RX interrupt,write 1 to clear [0]:TX interrupt,write 1 to clear
BAUDDIV	0x010	RW	20	0x00000	[19:0]:Baud rate divider,the minimum number is 16

3.3 初始化定义

UART 寄存器初始化定义如下表：

表 6 UART 寄存器初始化定义

名称	类型	数值	描述
UART_BaudRate	uint32_t	Max 960Kbit/s	Baud rate
UART_Mode	UARTMode_TypeDef	ENABLE/DISABLE	Enable/Disable TX/RX mode
UART_init	UARTInt_TypeDef	ENABLE/DISABLE	Enable/Disable TX/RX interrupt
UART_Ovr	UARTOvr_TypeDef	ENABLE/DISABLE	Enable/Disable TX/RX overrun interrupt
UART_Hstm	FunctionalState	ENABLE/DISABLE	Enable/Disable TX hisgh speed test mode

3.4 库函数使用方法

UART 库函数定义为 PANGO_M1_uart.c，使用方法如下表所示：

表 7 UART 库函数使用方法

函数名	描述
UART_Init	Initializes UARTx
UART_GetRxBufferFull	Returns UARTx RX buffer full status
UART_GetTxBufferFull	Returns UARTx TX buffer full status
UART_GetRxBufferOverrunStatus	Returns UARTx RX buffer overrun status
UART_GetTxBufferOverrunStatus	Returns UARTx TX buffer overrun status
UART_ClearRxBufferOverrunStatus	Clears Rx buffer overrun status
UART_ClearTxBufferOverrunStatus	Clears Tx buffer overrun status
UART_SendChar	Sends a character to UARTx TX buffer
UART_SendString	Sends a string to UARTx TX buffer
UART_ReceiveChar	Receives a character from UARTx RX buffer

UART_GetBaudDivider	Returns UARTx baud rate divider value
UART_GetTxIRQStatus	Returns UARTx TX interrupt status
UART_GetRxIRQStatus	Returns UARTx RX interrupt status
UART_ClearTxIRQ	Clears UARTx TX interrupt status
UART_ClearRxIRQ	Clears UARTx RX interrupt status
UART_GetTxOverrunIRQStatus	Returns UARTx TX overrun interrupt status
UART_GetRxOverrunIRQStatus	Returns UARTx RX overrun interrupt status
UART_ClearTxOverrunIRQ	Clears UARTx TX overrun interrupt request
UART_ClearRxOverrunIRQ	Clears UARTx RX overrun interrupt request
UART_SetHSTM	Sets UARTx TX high speed test mode
UART_ClrHSTM	Clears UARTx TX high speed test mode

3.5 参考设计

设计代码：../software_design/module_design/Cortex-M1_uart0 和../software_design/module_design/Cortex-M1_uart1 以及../software_design/Cortex-M1_uart0_irq 和../software_design/Cortex-M1_uart1_irq。分别有支持中断和不支持中断两种模式。

具体的工程代码实现，请参考工程代码。

4 Timer

TIMER 是一个用户可编程的计时器设备，具有 APB 总线接口。内部包含一个独立的可编程的 32 位计数器，计数器从一个寄存器写入的值开始向下计数，计数到 0 时触发定时器中断。

4.1 特征

PANGO Cortex-M1 包含 1 个通用 APB 总线访问的同步标准定时器：

- 32 位计数器
- 可以产生中断请求信号
- 支持两个定时器中断：Timer0 和 Timer1

Timer 结构如下图所示：

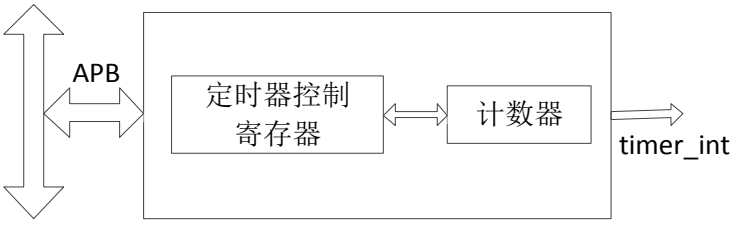

图 3 Timer 结构图

4.2 寄存器列表

Timer 寄存器 ENABLE 信号均是高电平有效，寄存器列表定义如下表：

表 8 Timer 寄存器列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
CTRL	0x00	RW	4	0x0	[3]:Timer interrupt enable [2]:Select external input as clock [1]:Select external input as enable [0]:Enable
VALUE	0x04	RW	32	0x00000000	[31:0]:Current value
RELOAD	0x08	RW	32	0x00000000	[31:0]:Reload value,writing to this register sets the current value
INTSTATUS/ INTCLEAR	0x0C	RW	1	0x0	[0]:Timer interrupt,write 1 to clear

4.3 初始化定义

Timer 寄存器初始化定义如下表所示：

表 9 Timer 寄存器初始化

名称	类型	数值	描述
Reload	uint32_t		Reload value
TIMER_Int	TIMERInt_TypeDef	SET/RESET	Enable/Disable interrupt
Timer_Exit	TIMERExti_TypeDef		External input as enable or clock

4.4 库函数使用方法

Timer 库函数定义为 PANGO_M1_timer.c，使用方法如下表所示：

表 10 Timer 库函数使用方法

函数名	描述
TIMER_Init	Initializes TIMERx
TIMER_StartTimer	Starts TIMERx
TIMER_StopTimer	Stops TIMERx
TIMER_GetIRQStatus	Returns TIMERx interrupt status
TIMER_ClearIRQ	Clears TIMERx interrupt status
TIMER_GetReload	Returns TIMERx reload value
TIMER_SetReload	Sets TIMERx reload value
TIMER_GetValue	Returns TIMERx current value
TIMER_SetValue	Sets TIMERx current value
TIMER_EnableIRQ	Enable TIMERx interrupt request
TIMER_DisableIRQ	Disable TIMERx interrupt request

4.5 参考设计

设计代码位于文件夹：../software_design/module_design/Cortex-M1_timer0 和../software_design/module_design/Cortex-M1_timer1。

具体的工程代码实现，请参考工程代码。

5 WDT

WDT 全称 Watchdog Timer，俗称“看门狗”，是 MCU 中常用的模块。WDT 是一个定时器电路，一般有一个操作俗称“喂狗”。MCU 在正常工作时，每隔一段时间便会对 WDT 进行“喂狗”操作，如果超过规定的时间不“喂狗”（一般程序跑飞），WDT 便会定时超时，从而复位 MCU。简言之，看门狗的常见作用就是防止程序发生死循环，或者程序跑飞等错误情形。

5.1 特征

PANGO Cortex-M1 包含 1 个通用 APB 总线访问的看门狗 Watch Dog：

- 基于由 LOAD 寄存器初始化的 32 位逐减计数器
- 产生中断请求
- 当时钟使能，由 WDOGCLK 信号上升沿触发计数器递减
- 监视中断，当计数器减到 0 时，产生复位请求，计数器停止
- 响应软件崩溃引起的复位，提供软件复位的方法

Watch Dog 结构如下图所示：

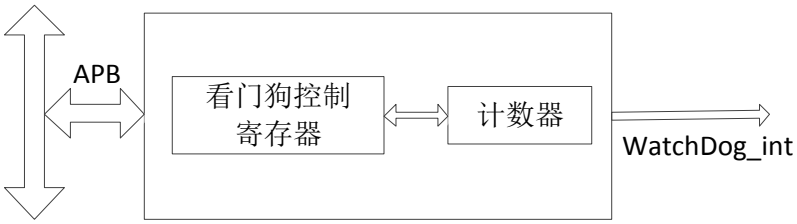

图 4 Watch Dog 结构图

5.2 寄存器列表

Watch Dog 寄存器 ENABLE 信号均是高电平有效，寄存器列表定义如下表：

表 11 Watch Dog 寄存器列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
LOAD	0x00	RW	32	0xFFFFFFFF	The value from which the counter is to decrement
VALUE	0x04	RO	32	0xFFFFFFFF	The current value of the decrementing counter
CTRL	0x08	RW	2	0x00	[1]:Enable reset output [0]:Enable the interrupt
INTCLR	0x0C	WO			Clear the watchdog interrupt and reloads the counter
RIS	0x10	RO	1	0x0	Raw interrupt status from the counter
MIS	0x14	RO	1	0x0	Enable interrupt status from the counter
RESERVED	0xC00-0x014				Reserved
LOCK	0xC00	RW	32	0x00000000	[32:1: Enable register writes [0]:Register write enable status
RESERVED	0xF00-0xC00				Reserved
ITCR	0xF00	RW	1	0x0	Integration test mode enable
ITOP	0xF04	WO	2	0x00	[1]:Integration test WDOGRES value [0]:Integration test WDOGINT value

5.3 初始化定义

Watch Dog 寄存器初始化定义如下表所示：

表 12 Watch Dog 寄存器初始化

名称	类型	数值	描述
WDOG_Reload	uint32_t		Reload value
WDOG_Lock	WDOGLock_TypeDef	SET/RESET	Enable/Disable lock register write access
WDOG_Res	WDOGRes_TypeDef	SET/RESET	Enable/Disable reset flag
WDOG_Int	WDOGInt_TypeDef	SET/RESET	Enable/Disable interrupt flag
WDOG_ITMode	WDOGMode_Typedef	SET/RESET	Enable/Disable integration test mode flag

5.4 库函数使用方法

Watch Dog 库函数定义为 PANGO_M1_wdog.c，使用方法如下表所示：

表 13 Watch Dog 库函数使用方法

函数名	描述
WDOG_Init	Initializes WatchDog
WDOG_RestartCounter	Restart watchdog counter
WDOG_GetCounterValue	Returns counter value
WDOG_SetResetEnable	Sets reset enable
WDOG_GetResStatus	Returns reset status
WDOG_SetIntEnable	Sets interrupt enable
WDOG_GetIntStatus	Returns interrupt enable
WDOG_ClrIntEnable	Clears interrupt enable
WDOG_GetRawIntStatus	Returns raw interrupt status
WDOG_GetMaskIntStatus	Returns masked interrupt status
WDOG_LockWriteAccess	Disable write access all registers
WDOG_UnlockWriteAccess	Enable write access all registers
WDOG_SetITModeEnable	Sets integration test mode enable
WDOG_ClrITModeEnable	Clears integration test mode enable
WDOG_GetITModeStatus	Returns integration test mode status
WDOG_SetITOP	Sets integration test output reset or interrupt
WDOG_GetITOPResStatus	Returns integration test output reset status
WDOG_GetITOPIntStatus	Returns integration test output interrupt status
WDOG_ClrITOP	Clears integration test output reset or interrupt

5.5 参考设计

设计代码位于文件夹：../software_design/module_design/Cortex-M1_watchdog。
具体的工程代码实现，请参考工程代码。

6 GPIO

6.1 特征

PANGO Cortex-M1 包含 16 个通用 AHB 总线访问的通用输入输出接口：

- 提供一组 16 个 I/O 的通用输入输出
- 每个 I/O 均可以直接受软件编程的可配置寄存器控制
- 目前 GPIO_Pin_0-15 支持中断

GPIO 结构如下图所示：

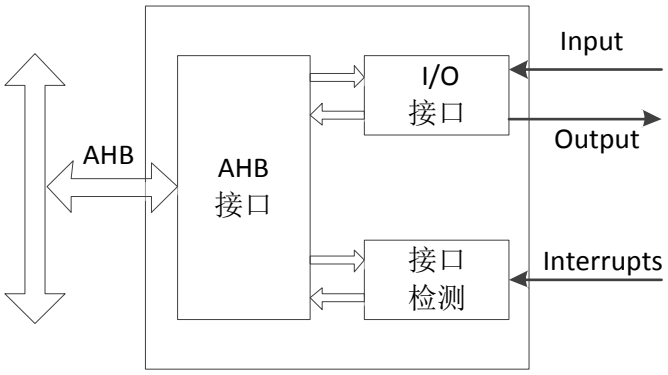

图 5 GPIO 结构图

6.2 寄存器列表

GPIO 寄存器 ENABLE 信号均是高电平有效，寄存器列表定义如下表：

表 14 GPIO 寄存器列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
DATA	0x0000	RW	16	0x--	[15:0]:Data value Read Sampled at pin Write to data output register Read back value goes through double flip-flop synchronization logic with delya of two cycle
DATAOUT	0x0004	RW	16	0x0000	[15:0]:Data output register value Read current value of data output register write to data output register
RESERVED	0x0008-0x000C				Reserved
OUTENSET	0x0010	RW	16	0x0000	[15:0]:Output enable set Write 1 to set the output enable bit Write 0 no effect Read back 0 indicates the signal direction as intput 1 indicates the signal direction as output
OUTENCLR	0x0014	RW	16	0x0000	[15:0]:Output enable clear Write 1 to clear the output enable bit Write 0 no effect Read back 0 indicates the signal direction as intput

					1 indicates the signal direction as output
ALTFUNCSET	0x0018	RW	16	0x0000	[15:0]:Alernative function set Write 1 to set the ALTFUNC bit Write 0 no effect Read back 0 for I/O 1 for an alternate function
ALTFUNCCLR	0x001C	RW	16	0x0000	[15:0]:Alernative function clear Write 1 to clear the ALTFUNC bit Write 0 no effect Read back 0 for I/O 1 for an alternate function
INTENSET	0x0020	RW	16	0x0000	[15:0]:Interrupt enable set Write 1 to set the enable bit Write 0 no effect Read back 0 indicates interrupt disabled 1 indicates interrupt enabled
INTENCLR	0x0024	RW	16	0x0000	[15:0]:Interrupt enable clear Write 1 to clear the enable bit Write 0 no effect Read back 0 indicates interrupt disabled 1 indicates interrupt enabled
INTTYPESET	0x0028	RW	16	0x0000	[15:0]:Interrupt type set Write 1 to set the interrupt type bit Write 0 no effect Read back 0 for LOW/HIGH level 1 for falling edge or rising edge
INTTYPECLR	0x002C	RW	16	0x0000	[15:0]:Interrupt type clear Write 1 to clear the interrupt type bit Write 0 no effect Read back 0 for LOW/HIGH level 1 for falling edge or rising edge
INTPOLSET	0x0030	RW	16	0x0000	[15:0]:Polarity-level,edge IRQ config Write 1 to set the interrupt polarity bit Write 0 no effect Read back 0 for LOW level or falling edge 1 for HIGH level or rising edge
INTPOLCLR	0x0034	RW	16	0x0000	[15:0]:Polarity-level,edge IRQ config Write 1 to clear the interrupt polarity bit Write 0 no effect Read back 0 for LOW level or falling edge 1 for HIGH level or rising edge
INTSTATUS /INTCLEAR	0x0038	RW	16	0x0000	[15:0]:Write IRQ status clear register Write 1 to clear interrupt request Write 0 no effectx Read back IRQ status register
MASKLOWBYTE	0x0400- 0x07FC	RW	16	0x--	Lower 8-bits masked access [9:2]:of the address value are used as enable bit mask for the access

					[15:8]:not used [7:0]:Data for lower byte access,with [9:2] of address value used as enable mask for each bit
MASKHIGHBYTE	0x0800-0x0BFC	RW	16	0x--	Higher 8-bits masked access [9:2]:of the address value are used as enable bit mask for the access [15:8]:Data for higher byte access,with [9:2] of address value used as enable mask for each bit [7:0]:not used
RESERVED	0x0C00-0x0FCF				Reserved

6.3 初始化定义

GPIO 寄存器初始化定义如下表所示：

表 15 GPIO 寄存器初始化

名称	类型	数值	描述
GPIO_Pin	uint32_t	GPIO_Pin_0 GPIO_Pin_1 GPIO_Pin_2 GPIO_Pin_3 GPIO_Pin_4 GPIO_Pin_5 GPIO_Pin_6 GPIO_Pin_7 GPIO_Pin_8 GPIO_Pin_9 GPIO_Pin_10 GPIO_Pin_11 GPIO_Pin_12 GPIO_Pin_13 GPIO_Pin_14 GPIO_Pin_15	16 bits GPIO Pins
GPIO_Mode	GPIOMode_TypeDef	GPIO_Mode_IN GPIO_Mode_OUT GPIO_Mode_AF	16 bits GPIO Pins mode
GPIO_Int	GPIOInt_TypeDef	GPIO_Int_Disable GPIO_Int_Low_Level GPIO_Int_High_Level GPIO_Int_Falling_Edge GPIO_Int_Rising_Edge	16 bits GPIO Pins interrupt

6.4 库函数使用方法

GPIO 库函数定义为 PANGO_M1_gpio.c，使用方法如下表所示：

表 14 GPIO 库函数使用方法

函数名	描述
GPIO_Init	Initializes GPIOx
GPIO_SetOutEnable	Sets GPIOx output enable
GPIO_ClrOutEnable	Clears GPIOx output enable
GPIO_GetOutEnable	Returns GPIOx output enable
GPIO_SetBit	GPIO output 1
GPIO_ResetBit	GPIO output 0
GPIO_WriteBits	GPIO output
GPIO_ReadBits	GPIO input
GPIO_SetAltFunc	Sets GPIOx alternate function enable
GPIO_ClrAltFunc	Clears GPIOx alternate function enable
GPIO_GetAltFunc	Returns GPIOx alternate function enable
GPIO_IntClear	Clears GPIOx interrupt request
GPIO_GetIntStatus	Returns GPIOx interrupt status
GPIO_SetIntEnable	Sets GPIOx interrupt enable Returns GPIOx interrupt status
GPIO_ClrIntEnable	Clears GPIOx interrupt enable Returns GPIOx interrupt enable
GPIO_SetIntHighLevel	Setups GPIOx interrupt as high level
GPIO_SetIntRisingEdge	Setups GPIOx interrupt as rising edge
GPIO_SetIntLowLevel	Setups GPIOx interrupt as low level
GPIO_SetIntFallingEdge	Setups GPIOx interrupt as falling edge
GPIO_MaskedWrite	Setups GPIOx output value using masked access

6.5 参考设计

设计代码位于文件夹：../software_design/module_design/Cortex-M1_led、../software_design/module_design/Cortex-M1_key_irq 以及../software_design/module_design/Cortex-M1_key_scan，其中按键支持中断方式和扫描方式。

具体的工程代码实现，请参考工程代码。

7 SPI

SPI 全称为 Serial Peripheral Interface（串行外设接口），是 MCU 中常用的接口模块。SPI 的通信原理很简单，它以主从方式工作（目前 PANGO Cortex-M1 只支持主模式），通常有一个主设备和一个或多个从设备，需要至少四根线，分别是 SDI（数据输入）、SDO（数据输出）、SCK（时钟）、CS（片选），下面分别予以介绍。

- MOSI：SPI 总线主机输出/从机输入（Master Output/Slave Input）
- MISO：SPI 总线主机输入/从机输出（Master Input/Slave Output）
- SCK：时钟信号，由主设备产生
- CS：从设备是能信号（Chip Select），由主设备控制，有些芯片的此 Pin 脚也称 SS

7.1 特征

PANGO Cortex-M1 包含 1 个通用 APB 总线访问的串行外设接口：

- APB 总线接口
- 作为主 SPI，支持发送和接收数据
- 支持通过寄存器动态配置 SPI 的时钟信号
- 支持同时挂载 8 个 SPI 从设备（CS1 - CS8）

SPI 结构图如下图所示：

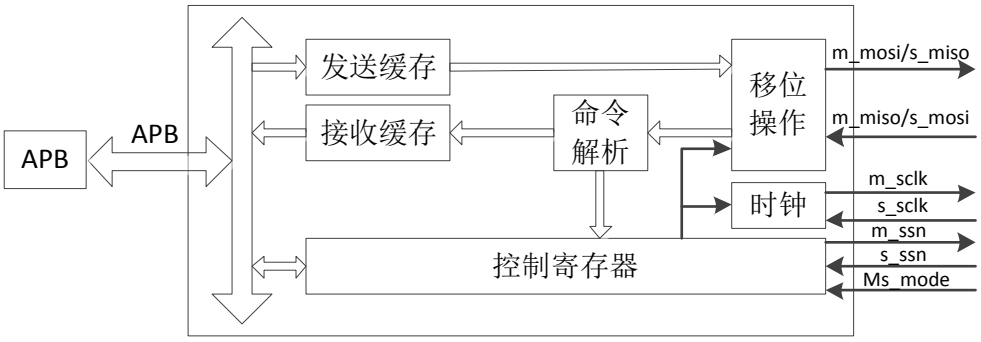

图 6 SPI 结构图

7.2 寄存器列表

SPI 寄存器 ENABLE 信号均是高电平有效，寄存器列表定义如下表：

表 15 SPI 寄存器列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
RDATA	0x00	RO	32	0x00000000	Read data register [31:8]:Reserved [7:0]:Read data
WDATA	0x04	WO	32	0x00000000	Write data register [31:8]:Reserved [7:0]:Write data
STATUS	0x08	RW	32	0x00000000	[31:1]:Reserved [0]:Select and enable slave
SSMASK	0x0C	RW	32	0x00000000	[31:8]:Reserved [7]: Be receiving

					[6]:Receive ready status [5]:Transmit ready status [4]:Be transmitting [3]:Transmit overrun error status [2]:Receive overrun error status [1:0]:Reserved
CTRL	0x10	RW	32	0x00000000	[31:16]:Reserved [15:8]:CS selected,CS1 - CS8 [7:5]:Reserved [4:3]:Clock selected, CLK_4/8/16/32 [2]:Clock polarity [1]:Clock phase [0]:Direction, 1 is MSB first

7.3 初始化定义

SPI 寄存器初始化定义如下表：

表 16 SPI 寄存器初始化定义

名称	类型	数值	描述
DIRECTION	FunctionalState	ENABLE/DISABLE	MSB/LSB first transmission
PHASE	FunctionalState	ENABLE/DISABLE	Posedge/Negedge transmit data
POLARITY	FunctionalState	ENABLE/DISABLE	Initialize ploarity to one/zero
CLKSEL	uint32_t	CLKSEL_CLK_DIV_4 CLKSEL_CLK_DIV_8 CLKSEL_CLK_DIV_16 CLKSEL_CLK_DIV_32	Select clock divided 4/8/16/32
CS_SEL	uint32_t	SPI_CS_SEL_1 SPI_CS_SEL_2 SPI_CS_SEL_3 SPI_CS_SEL_4 SPI_CS_SEL_5 SPI_CS_SEL_6 SPI_CS_SEL_7 SPI_CS_SEL_8	CS Select,CS1 – CS8

7.4 库函数使用方法

SPI 库函数定义为 PANGO_M1_spi.c，使用方法如下表所示：

表 17 SPI 库函数使用方法

函数名	描述
SPI_Init	Initializes SPI
SPI_SetDirection	Sets Direction
SPI_ClrDirection	Clears Direction
SPI_GetDirection	Returns Direction
SPI_SetPhase	Sets Phase
SPI_ClrPhase	Clears Phase

SPI_GetPhase	Returns Phase
SPI_GetPolarity	Returns Polarity
SPI_ClrPolarity	Clears Polarity
SPI_SetPolarity	Sets Polarity
SPI_SetClkSel	Sets ClkSel
SPI_GetClkSel	Returns ClkSel
SPI_GetToeStatus	Reads transmit overrun error status
SPI_GetRoeStatus	Reads receive overrun error status
SPI_GetTmtStatus	Reads transmitting status
SPI_GetTrdyStatus	Reads transmit ready status
SPI_GetRrdyStatus	Reads receive ready status
SPI_GetRcvStatus	Reads recving status
SPI_SetRcvStatus	Write recving status
SPI_ClrToeStatus	Clears transmit overrun error status
SPI_ClrRoeStatus	Clear receive overrun error status
SPI_ClrErrStatus	Clears error status
SPI_WriteData	Writes Data
SPI_ReadData	Reads Data
SPI_Select_Slave	SPI_Slave Select
SPI_CS_Enable	SPI_CS Enable
SPI_CS_Disable	SPI_CS Disable

7.5 参考设计

设计代码位于文件夹：../software_design/module_design/Cortex-M1_spi，其中包含了 W25Qxx 系列 SPI Flash 的驱动代码。

具体的工程代码实现，请参考工程代码。

8 I2C

I2C 全称 Inter-Integrated Circuit（集成电路互联在线），是 MCU 中常用的接口模块。I2C 总线只有两条总线，一条串行数据总线（SDA）和一条串行时钟线（SCL）。SDA（串行数据线）和 SCL（串行时钟线）都是双向 I/O 线。接口电平为开漏输出，需要通过上拉电阻接电源。当总线空闲时，两根线都是高电平。每个连接总线的设备都可以使用唯一的地址来识别。

8.1 特征

PANGO Cortex-M1 包含 1 个通用 APB 总线访问的 I2C 接口：

- APB 总线接口
- 符合业界标准的 I2C 总线协议
- 总线仲裁及仲裁丢失检测
- 总线忙状态检测
- 支持产生发送或接收中断标志（建议中断优先级设置为 0）
- 产生起始、终止、重复起始应答信息
- 支持起始、终止和重复起始检测
- 支持 7 位寻址模式
- 支持 SCL 频率配置，标准速度（最高速率 100KHZ），快速（最高 400KHZ）
- 支持主模式

I2C 结构图如下如所示：

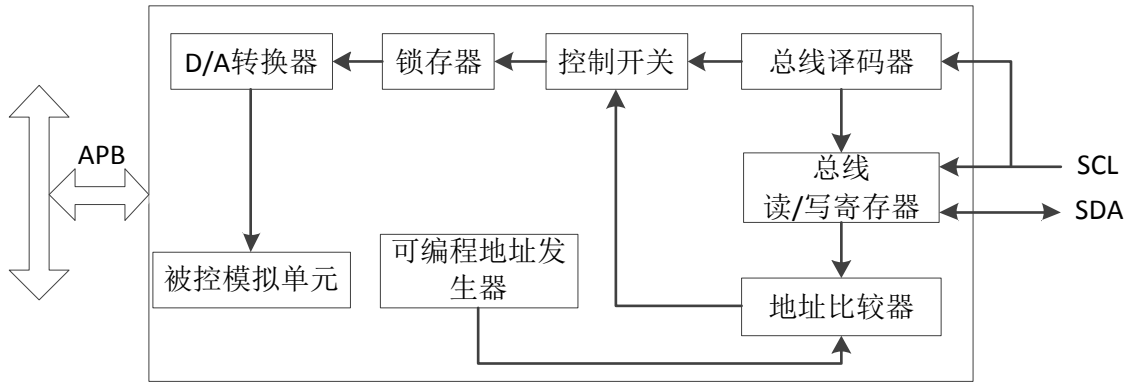

图 7 I2C 结构图

8.2 寄存器列表

I2C 寄存器 ENABLE 信号均是高电平有效，寄存器列表定义如下表：

表 18 I2C 寄存器列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
PRER	0x00	RW	32	0x0000ffff	Clock prescale register [31:16]:Reserved [15:0]: Prescale value = sys_clk/(5*SCL)-1
CTR	0x04	RW	32	0x00000000	[31:8]: Reserved [7]: Enable I2C function [6]: Enable I2C interrupt

					[5:0]: Reserved
TXR	0x08	WO	32	0x00000000	[31:8]: Reserved [7:1]: Next transmission data [0]: Data direction
RXR	0x0c	RO	32	0x00000000	[31:8]: Reserved [7:0]: Last received data
CR	0x10	WO	32	0x00000000	[31:8]: Reserved [7]: STA, Start transmission status [6]: STO, Over transmission status [5]: RD, Read enable, read data from slave [4]: WR, Write enable, write data to slave [3]: Acknowledge [2:1]: Reserved [0]: Interrupt acknowledge
SR	0x14	RO	32	0x00000000	[31:8]: Reserved [7]: Receive acknowledge signal from slave [6]: I2C busy status [5]: Arbitration loss [4:2]: Reserved [1]: Data transmission status flag [0]: Interrupt flag

8.3 库函数使用方法

I2C 库函数定义为 PANGO_M1_i2c.c，使用方法如下表所示：

表 19 I2C 库函数使用方法

函数名	描述
I2C_Init	Initializes IIC
I2C_SendByte	Send a byte to I2C bus
I2C_SendBytes	Send multiple bytes to I2C bus
I2C_ReceiveByte	Read a byte from I2C bus
I2C_ReadBytes	Read multiple bytes from I2C bus
I2C_Rate_Set	Set I2C traffic rate
I2C_Enable	Enable I2C bus
I2C_UnEnable	Disable I2C bus
I2C_InterruptOpen	Open I2C Interrupt
I2C_InterruptClose	Close I2C Interrupt

8.4 参考设计

设计代码位于文件夹：../software_design/module_design/Cortex-M1_i2c，其中包含了 24xx128 系列 EEPROM 的驱动代码。

具体的工程代码实现，请参考工程代码。

9 FreeRTOS

本节介绍如何在PANGO Cortex-M1环境下移植一个简单的FreeROTS示例。用户可以以此为基础进行丰富和完善，开发更多的复杂示例。

9.1 特征

- FreeRTOS 是一个迷你的实时操作系统
- FreeRTOS 作为一个轻量级的操作系统，功能包括：任务管理、时间管理、信号量、消息队列、内存管理、记录功能、软件定时器、协程等，可基本满足较小系统的需要
- FreeRTOS 操作系统是完全免费的操作系统，具有源码公开、可移植、可裁减、调度策略灵活的特点
- FreeRTOS 文档资料齐全，在FreeRTOS官网能下载到内核文件及详细的介绍资料
- FreeRTOS 安全性高，SafeRTOS基于FreeRTOS而来，经过安全认证的RTOS
- FreeRTOS 内核文件简单，内核相关文件仅由3个C文件组成，全部围绕任务调度展开，功能专一，便于理解学习

9.2 操作系统版本

PANGO Cortex-M1参考设计使用的FreeRTOS 版本为V9.0.0。

9.3 操作系统配置

用户可以通过修改include\FreeRTOSConfig.h 来配置FreeRTOS。

在FreeRTOS的FreeRTOSConfig.h文件中，FreeRTOS中的调度算法分为时间片调度算法和抢占式调度，对应的宏定义分别为：configUSE_TIME_SLICING和configUSE_PREEMPTION。即使不配置configUSE_TIME_SLICING为1，FreeRTOS也会默认开启时间片调度。时间片调度算法和抢占式调度简述如下：

- 时间片调度算法：每一个任务给予固定的执行时间，时间结束后进入调度器，由调度器切换到下一个任务，在默认所有任务优先级相同情况下，轮流执行所有任务。
- 抢占式调度需要设置任务优先级，在进入调度器后，调度器选择处于就绪态中优先级最高的任务作为下一个执行的任务。高优先级任务可以抢占低优先级任务，发生抢占时需要有能进入调度器的操作，调度器是任务切换的唯一实体。

移植代码详情，请用户自行参见三个文件名为“port*”的源代码。

9.4 操作系统任务与中断关系

FreeRTOS的任务和中断的优先级关系是移植FreeRTOS的难点，需要被正确的理解，否则程序会运行出错：

- 任务总是可以被中断打断，任务之间具有的优先级，但是与“中断的优先级”没有关系，这两种优先级是相互独立的。
- 不调用任何FreeRTOS API函数的中断，可以设置为任意的“中断优先级”，并且允许嵌套。
- 在FreeRTOSConfig.h中预先定义configMAX_SYSCALL_INTERRUPT_PRIORITY的值，调用API函数的中断优先级只能设置为不大于该值，支持嵌套，但是会被内核延迟。

关于 FreeRTOS 的任务优先级和中断优先级如何设置，以及 FreeRTOS 的更多详细信息，请用户自行查阅相关 FreeRTOS 手册学习。

9.5 参考设计

FreeRTOS 设计代码位于文件夹：../software_design/ rtos/Cortex-M1_FreeRtos_V9.0.0。具体的工程代码实现，请参考工程代码。

内核支持操作系统，需要 RTL 代码做相应的配置，即：cm1_option_defs.v 文件里面的 CM1_OS 宏需要打开，具体操作流程请参考《FPGA Cortex-M1 SoC DEMO 应用指南》。

10 Memory

本节讲解了 PANGO Cortex-M1 软核访问 FPGA 块 RAM，首先需要在 PANGO Cortex-M1 SOC 里面例化块 RAM 的 IP，详细的例化操作请参考《FPGA Cortex-M1 SoC DEMO 应用指南》。

10.1 特征

PANGO Cortex-M1 包含 16 个通用 AHB 总线访问的 RAM 接口：

- AHB 总线接口
- 通过指针直接对 AHB 总线地址上面的块 RAM 进行访问，速度快，效率高
- 数据位宽 32bit
- 单个块 RAM 地址位宽 24bit

11 Systick

Systick 即系统定时器，存放在 NVIC 中，主要目的是为了给操作系统提供一个硬件上的中断（称为滴答中断）。也可以作为定时器单独使用。

11.1 特征

- AHB 总线接口
- 精确延时
- 支持操作系统滴答中断

11.2 寄存器列表

Systick 寄存器 ENABLE 信号均是高电平有效，寄存器定义如下表：

表 20 Systick 寄存器列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
CTRL	0x00	RW	32	0x00000000	[5]: COUNTFLAG Position [2]: CLKSOURCE Position [1]: TICKINT Position [0]: ENABLE Position
LOAD	0x04	RW	32	0x00000000	[31:24]:Reserved [23:0]: RELOAD Position
VAL	0x08	RW	32	0x00000000	[31:24]:Reserved [23:0]: CURRENT Position
CALIB	0x0c	RO	32	0x00000000	[31]: NOREF Position [30]: SKEW Position [29:24]: Reserved [23:0]: TENMS Position

注意：SysTick->LOAD 寄存器值表示 Systick 中断计数值，在使用 ICACHE 或 ICACHE 和 DCACHE 的时候，该寄存器的值建议设置小于 10000，即：SysTick 中断触发周期为 100us。不然会导致程序异常。

11.3 初始化定义

Systick 寄存器初始化定义如下表：

表 21 Systick 寄存器定义

名称	类型	数值	描述
SysTick_CTRL_ENABLE_Msk	uint32_t	0x00000000	ENABLE Mask
SysTick_CTRL_TICKINT_Msk	uint32_t	0x00000002	TICKINT Mask
SysTick_CTRL_CLKSOURCE_Msk	uint32_t	0x00000004	CLKSOURCE Mask
SysTick_CTRL_COUNTFLAG_Msk	uint32_t	0x00000020	COUNTFLAG Mask

11.4 库函数使用方法

Systick 库函数定义为 systick.c，使用方法如下表所示：

表 22 Systick 库函数使用方法

函数名	描述
SystickInit	Initialises the systick
_delay_us	delay us
TimingDelay_Decrement	get systick,add into SysTick_Handler Function

11.5 参考设计

设计代码位于文件夹：../software_design/module_design/Cortex-M1_systick。具体的工程代码实现，请参考工程代码。

内核支持 Systick，需要 RTL 代码做相应的配置，即：cm1_option_defs.v 文件里面的 CM1_OS 宏需要打开，具体操作流程请参考《FPGA Cortex-M1 SoC DEMO 应用指南》。

12 Print

打印函数包含系统打印函数（printf()）以及自定义打印函数（printk()），printf()函数占用资源较多，但是功能丰富，在资源紧张的情况下，并且只需要使用简单的打印，可以选择 printk()函数。

12.1 说明

系统打印函数（printf()）支持的规定符：

- %d：十进制有符号整数
- %u：十进制无符号整数
- %f：浮点数
- %s：字符串
- %c：单个字符
- %p：指针的值
- %e：指数形式的浮点数
- %x，%X：无符号以十六进制表示的整数
- %o：无符号以八进制表示的整数
- %g：把输出的值按照%e 或%f 类型中输出长度较小的方式输出
- %lu：32 位无符号整数
- %llu：64 位无符号整数

并且%和字母之间插进数字表示最大场宽，例如：%3d 表示输出 3 位整型数，不够三位右对齐，这里不一一例举，详细的使用方法用户可自行查阅相关资料。

自定义打印函数（printk()）支持的规定符：

- %d，%D，%i：十进制有符号整数
- %c：单个字符
- %f：浮点数
- %s，%S：字符串
- %x，%X：无符号以十六进制表示的整数

不支持%和字母之间插进数字。

使用系统打印函数（printf()）需要以下三个步骤：

- 重定义 fputc 函数
- 头文件包含 stdio.h 文件
- 编译软件需要选中“User MicroLIB”，在工程属性的“Target”->“Code Generation”选项中勾选“User MicroLIB”

以上三个步骤完成后就可以使用系统打印函数（printf()）。

12.2 参考设计

设计代码位于文件夹：../software_design/module_design/Cortex-M1_uart_printf 和../software_design/module_design/Cortex-M1_uart_printk，

自定义打印函数（printk()）详细设计，请参考 PANGO_M1_printk.c 文件。

13 MyMem

动态内存操作通过自定义函数 mymalloc()和 myfree()完成。先规定好整个动态空间的大小，然后对该空间进行重复的读写操作，操作完成以后，释放申请的空间，合理的利用资源。

13.1 参考设计

设计代码位于文件夹：../software_design/module_design/Cortex-M1_my_mem，具体的工程代码实现，请参考工程代码。

14 DMA

DMA（Direct Memory Access）既可以指内存和外设直接存取数据这种内存访问的计算机技术，又可以指实现该技术的硬件模块（对于通用计算机 PC 而言，DMA 控制逻辑由 CPU 和 DMA 控制接口逻辑芯片共同组成，嵌入式系统的 DMA 控制器内嵌在处理器芯片内部，一般称为 DMA 控制器，DMAC）。

14.1 特征

PANGO Cortex-M1 包含 1 个通用 AHB 总线访问的 DMA 接口：

- AHB 总线接口
- 实现片内 DRM 存放“链结构描述符”
- TX 环为 2 级，RX 环为 32 级，可配置缓冲区大小，最大为 1600B

DMA 结构框图如下：

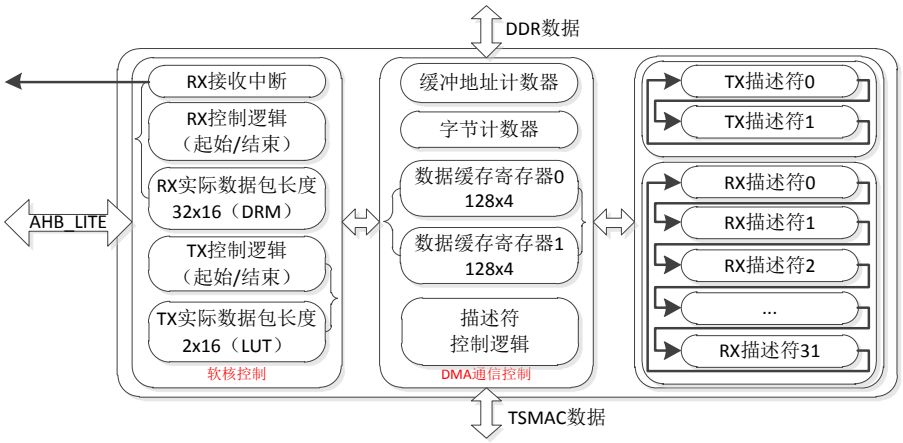

图 8DMAC 结构框图

14.2 寄存器列表

DMA 寄存器 ENABLE 信号均是高电平有效，寄存器定义如下表：

表 23DMA 寄存器列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
DESCRIPTOR_NUM	0x00	WO	32	0x00000000	Descriptor RX/TX Register Num [31:16]:TX Descriptor Num [15:0]: RX Descriptor Num
DESCRIPTOR_CTRL	0x04	WO	32	0x00000000	Descriptor Control Register [31:18]: Reserved [17]:TX Init [16]:TX Link Num [15]:RX Init [14:0]: RX Link Num
DESCRIPTOR_TX_ADDR	0x08	WO	32	0x00000000	Descriptor Tx Address [31:0]: Set Descriptor Tx Address
DESCRIPTOR_RX_ADDR	0x0c	WO	32	0x00000000	Descriptor Rx Address [31:0]: Set Descriptor Rx Address
DESCRIPTOR_LEN	0x10	WO	32	0x00000000	Descriptor Length

					[31:16]:TX Length [15:0]: RX Length
DMAC_CTRL	0x14	WO	32	0x00000000	DMAC Control Register [31:2]: Reserved [1]: DMAC TX Control Register [0]: DMAC RX Control Register
DMAC_TX_LEN	0x18	WO	32	0x00000000	DMAC Tx Data Length [31:0]: Tx Data Length
DMAC_RX_LEN	0x1c	RO	32	0x00000000	DMAC Rx Data Length [31:0]: Rx Data Length
DMAC_TX_NUM	0x20	RO	32	0x00000000	DMAC Tx Descriptor Num [31:0]: Tx Descriptor Num
DMAC_RX_NUM	0x24	RO	32	0x00000000	DMAC Rx Descriptor Num [31:0]: Rx Descriptor Num
INTSTATUS INTCLEAR	0x28	RO WO	32	0x00000000 0x00000000	Interrupt Status Register [31:2]: Reserved [1]: AXI Interrupt Status Register [0]: RX Interrupt Status Register Interrupt Clear Register [31:2]: Reserved [1]: AXI Clear Interrupt Status Register [0]: RX Clear Interrupt Status Register
CTRL	0x2c	WO	32	0x00000000	Interrupt Control Register [31:1]: Reserved [0]: DMA Interrupt Enable Control Register
STATUS	0x30	RW	32	0x00000000	Interrupt Status Register [31:2]: Reserved [1]: DMA Init Status Register [0]: RX Idle Status Register

14.3 初始化定义

DMA 寄存器初始化定义如下表：

表 24DMA 寄存器定义

名称	类型	数值	描述
DESCRIPTOR_RX_INIT	uint32_t	0x00008000	Descriptor Rx Init
DESCRIPTOR_TX_INIT	uint32_t	0x00020000	Descriptor Tx Init
DMAC0_RX_INIT	uint32_t	0x00000001	DMAC Rx Init
DMAC0_TX_INIT	uint32_t	0x00000002	DMAC Tx Init
DMAC0_RX_CTRL	uint32_t	0x00000004	DMAC Rx Control
DMAC0_TX_CTRL	uint32_t	0x00000008	DMAC Tx Control
DMAC0_RXIRQ_ENABLE	uint32_t	0x00000001	DMAC Rx Interrupt Enable
DMAC0_RXIRQ_DISABLE	uint32_t	0x00000000	DMAC Rx Interrupt Disable
DMAC0_STATUS_RX_IDLE	uint32_t	0x00000001	DMAC Rx Idle
DMAC0_STATUS_TX_IDLE	uint32_t	0x00000002	DMAC Tx Idle
DMAC0_STATUS_INIT_DONE	uint32_t	0x00000004	DMAC Init Done

DMAC0_INTSTATUS_AXI_IRQ	uint32_t	0x00000001	DMAC AXI Interrupt Status
DMAC0_INTCLEAR_AXI_IRQ	uint32_t	0x00000001	DMAC Clear AXI Interrupt Status
DMAC0_INTSTATUS_RXIRQ	uint32_t	0x00000002	DMAC RX Data Interrupt Status
DMAC0_INTCLEAR_RXIRQ	uint32_t	0x00000002	DMAC Clear RX Interrupt Status

14.4 库函数使用方法

DMA 库函数定义为 PANGO_M1_eth_dmac.c、PANGO_M1_eth_buffer.c，分别为 DMAC 的库函数使用以及 TX 和 RX 描述符环库函数的使用。使用方法如下表所示：

表 25DMAC 库函数使用方法

函数名	描述
DMAC_Init	Initialises the dmac
SetTxDescriptor	Set Tx Descriptor Info
SetRxDescriptor	Set Rx Descriptor Info
GetCurrentTxDescriptorNum	Get Current Tx Descriptor Num
GetCurrentRxDescriptorNum	Get Current Rx Descriptor Num
SetTxDataLen	Set Tx Data Buffer Size
GetRxDataLen	Set Rx Data Buffer Size
CleanRxDescriptorNum	Clean Rx Descriptor Num Info,Data Process Finished
SetDmacTxCtrl	Set Dmac Tx Ctrl
SetDmacRxCtrl	Set Dmac Rx Ctrl
DMAC_GetRxIdleStatus	Get Rx Idle Status Info
DMAC_GetTxIdleStatus	Get Tx Idle Status Info
DMAC_GetAxiIRQStatus	Get Axi IRQ Status Info
DMAC_ClearAxiIRQ	Clear Axi IRQ Status Info
DMAC_GetRxIRQStatus	Get Rx IRQ Status Info
DMAC_SetInitStatus	Set Descriptor Init Status
DMAC_ClearRxIRQ	Clear Rx IRQ Status Info
TSMAC_Configure	TSMAC Configure

表 26Descriptor Tx Rx 库函数使用方法

函数名	描述
eth_buf_init	Ethernet Buffer Init
eth_tx_buf_init	Ethernet Tx Buffer Init
eth_rx_buf_init	Ethernet Rx Buffer Init
eth_tx_data	Ethernet Tx Data
clear_eth_rx_buf	Clear Ethernet Rx Buffer
eth_tx_buf_flush	Ethernet Tx Buffer Flush
eth_rx_buf_invalidate	Ethernet Rx Buffer Invalidate
eth_rx_buf_flush	Ethernet Rx Buffer Flush
ENT_Handler	Ethernet Rx Handler

14.5 参考设计

设计代码位于文件夹：../software_design/LwIP_V2.1.2。具体的工程代码实现，请参考工程代码。

15 TSMAC 配置

10/100/1000M Ethernet MAC 是深圳市紫光同创电子有限公司FPGA产品中用于实现以太网MAC层协议按照IEEE 802.3-2005标准设计的IP，通过公司Pango Design Suite套件（简称PDS）中IP Compiler工具（简称IPC）例化生成IP模块。

15.1 特征

- 支持 1000Mbps 以太网 MAC
- 支持 RGMII 接口
- 支持 APB 管理控制接口
- 10/100Mbps 模式全、半双工，支持 1000M 模式全双工
- 支持自动填充短帧
- 支持可选的为发送帧添加 FCS
- 支持可选的为接收帧删除FCS
- 支持可选的广播地址、多播地址、单播地址过滤

目前软核仅支持 TSMAC IP 的以上特征配置，详细功能请参考《UG_IP_Sys_002 Titan 系列产品 TSMAC IP 用户指南 v1.6》。

15.2 寄存器列表

TSMAC 寄存器 ENABLE 信号均是高电平有效，寄存器列表定义如下表：

表 27 TSMAC 寄存器列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
TSMAC_TX_RX	C000	WO	32	0x00000000	[31]:Soft Reset [30:9]: Reserved [8]:Loop Back [7:6]: Reserved [5]:Rx Flow [4]:Tx Flow [3]: Reserved [2]:Rx Enable [1]: Reserved [0]:Tx Enable
TSMAC_CONFIG	C010	WO	32	0x00000000	[31:16]: Reserved [15:12]:Preamble Length[3:0] [11:10]: Reserved [9:8]:Interface Mode[1:0] [7:6]: Reserved [5]:Huge Frame [4]:Length Check [3]: Reserved [2]:Pad/CRC [1]:CRC Enable [0]:Full Duplex

TSMAC_ADD_FILTER	C120	WO	32	0x00000000	[31:13]: Reserved [12]:nfcs [11]:fbc [10]:fuc [9]:fmc [8:0]:dahash
TSMAC_ADD0	C130	WO	32	0x00000000	[31:24]:Local MAC Address Low[7:0] [23:16]: Local MAC Address Low[15:8] [15:8]: Local MAC Address Low[23:16] [7:0]: Local MAC Address Low[31:24]
TSMAC_ADD1	C140	WO	32	0x00000000	[31:24]: Local MAC Address High[39:32] [23:16]: Local MAC Address High[47:40] [15:0]:Reserved

注意：地址偏移量相对于 TSMAC_IP 的配置寄存器地址左移了 4 个 bit，即：0xC010 的偏移地址对应 TSMAC_IP 的 0x10 偏移地址。

15.3 初始化定义

TSMAC 寄存器初始化定义如下表：

表 28 TSMAC 寄存器定义

名称	类型	数值	描述
TSMAC_TX_RX_DATA	uint32_t	0x00000035	MAC_Configuration1
TSMAC_CONFIG_DATA	uint32_t	0x00007217	MAC_Configuration2
TSMAC_ADD_FILTER_DATA	uint32_t	0x00000480	Address Filter

15.4 库函数使用方法

TSMAC 库函数定义定义在 PANGO_M1_eth_dmac.c 文件中，使用方法如下表所示：

表 29 TSMAC 库函数使用方法

函数名	描述
TSMAC_Configure	TSMAC Configure

15.5 参考设计

设计代码和 DMA 的驱动代码编写在 PANGO_M1_eth_dmac.c 文件，具体的工程代码实现，请参考工程代码。

16 UDP 硬件加速

为满足网络应用场景中高速通信的需求，设计开发 UDP 硬件加速模块，UDP 硬件加速模块发送数据和软核通过 LwIP 发送数据相互独立，互不影响，但是使用的同一个 TSMAC IP，软核发送数据需要申请 TSMAC IP 使用权，软核发送结束交还 TSMAC IP 使用权给到 UDP 硬件加速模块。

16.1 特征

- AHB 总线接口
- 加速模块根据用户需求可打开或关闭
- UDP 硬件加速模块发送平均速率可达到 985Mbps

16.2 寄存器列表

UDP 硬件加速寄存器 ENABLE 信号均是高电平有效，寄存器定义如下表：

表 30 UDP 硬件加速寄存器列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
UDP_PROTOCOL_DEST_MAC_H	0x00	WO	32	0x00000000	[31:24]:Dest Mac0 [23:16]:Dest Mac1 [15:8]:Dest Mac2 [7:0]:Dest Mac3
UDP_PROTOCOL_DEST_MAC_L	0x04	WO	32	0x00000000	[31:24]:Dest Mac4 [23:16]:Dest Mac5 [15:0]: Reserved
UDP_PROTOCOL_SOURCE_MAC_H	0x08	WO	32	0x00000000	[31:24]:Source Mac0 [23:16]: Source Mac1 [15:8]: Source Mac2 [7:0]: Source Mac3
UDP_PROTOCOL_SOURCE_MAC_L	0x0c	WO	32	0x00000000	[31:24]: Source Mac4 [23:16]: Source Mac5 [15:0]: Reserved
UDP_PROTOCOL_DEST_IP	0x10	WO	32	0x00000000	[31:24]:Dest IP0 [23:16]:Dest IP1 [15:8]:Dest IP2 [7:0]:Dest IP3
UDP_PROTOCOL_SOURCE_IP	0x14	WO	32	0x00000000	[31:24]:Source IP0 [23:16]: Source IP1 [15:8]: Source IP2 [7:0]: Source IP3
UDP_PROTOCOL_PORT	0x18	WO	32	0x00000000	[31:16]:Source Port [15:0]: Dest Port
UDP_PROTOCOL_IP_IDENTICATION	0x1c	RW	32	0x00000000	[31:0]:IP identication
UDP_SPEED_UP_STATUS	0x20	RW	32	0x00000000	[31:3]: Reserved [2]:Soc Connect Status [1]:Soc TX Ask Status [0]:Speed Up Status

16.3 初始化定义

UDP 硬件加速寄存器初始化定义如下表：

表 31 UDP 硬件加速寄存器定义

名称	类型	数值	描述
UDP_SPEED_UP_TX_BUSY	uint32_t	0x00000001	Speed Up Tx Status
UDP_SOC_TX_REQ	uint32_t	0x00000002	SOC Tx Request Status
UDP_SOC_CONNECT	uint32_t	0x00000004	SOC Connect Status

16.4 库函数使用方法

UDP 硬件加速库函数定义为 PANGO_M1_udp_speed_up.c，使用方法如下表所示：

表 32 UDP 硬件加速库函数列表

函数名	描述
SetUdpSpeedUpStatus	Set Speed Up Status
GetUdpSpeedUpTxBusy	Get Speed Up Status
SetTxStatusON	Enable SOC Tx
SetTxStatusOFF	Disable SOC Tx
GetDestEthInfo	Get Dest Ethernet Information
SetSpeedUpInfo	Set Dest And Source Ethernet Information

UDP 硬件加速模块打开或关闭需要用户在 RTL 代码中手动操作，详细配置请参考《FPGA Cortex-M1 SoC DEMO 应用指南》。在软件工程中，使用 PANGO_M1_udp_speed_up.h 文件中的 UDP_SPEED_UP 宏进行配置，和 RTL 代码配置相对应。

在打开 UDP 硬件加速宏定义的基础上，PC 端和嵌入式软核开发板（后文简称开发板）进行一次有效的 UDP 通信后，UDP 硬件加速就会打开并切换到 UDP 加速模式，即：PC 端在 UDP 模式下发送一串用户自定义字符，并且收到开发板的返回信息（一定要接收到发板的返回，PC 端发送的数据很有可能开发板没收到），表示 PC 端和开发板通信连接成功。

16.5 参考设计

设计代码位于文件夹：../software_design/LwIP_V2.1.2。具体的工程代码实现，请参考工程代码。

17 LwIP 移植

本例程使用 LwIP 2.1.2 版本，官方下载链接：http://savannah.nongnu.org/projects/lwip/。

17.1 特征

本例程以LwIP 2. 1.2 为主要对象进行讲解，后续中出现的LwIP 如果没有特殊声明，均指2.1.2 版本。

LwIP 全名：Light weight IP，意思是轻量化的TCP/IP 协议，是瑞典计算机科学院(SICS)的Adam Dunkels 开发的一个小型开源的TCP/IP 协议栈。LwIP 的设计初衷是：用少量的资源消耗实现一个较为完整的TCP/IP 协议栈，其中“完整”主要指的是TCP 协议的完整性，实现的重点是在保持TCP 协议主要功能的基础上减少对RAM 的占用。此外LwIP既可以移植到操作系统上运行，也可以在无操作系统的情况下独立运行。

LwIP 具有主要特性：

- 支持ARP 协议（以太网地址解析协议）
- 支持ICMP 协议（控制报文协议），用于网络的调试与维护
- 支持IGMP 协议（互联网组管理协议），可以实现多播数据的接收
- 支持UDP 协议(用户数据报协议)
- 支持TCP 协议(传输控制协议)，包括阻塞控制、RTT 估算、快速恢复和快速转发
- 支持PPP 协议（点对点通信协议），支持PPPoE
- 支持DNS（域名解析）
- 支持DHCP 协议，动态分配IP 地址
- 支持IP 协议，包括IPv4、IPv6 协议，支持IP 分片与重装功能，多网络接口下的数据包转发
- 支持SNMP 协议（简单网络管理协议）
- 支持AUTOIP，自动IP 地址配置
- 提供专门的内部回调接口(Raw API)，用于提高应用程序性能
- 提供可选择的Berkeley 接口API，即 Socket 套接字 (在多线程情况下使用)

LwIP并没有很完整地实现TCP/IP 协议栈。相比于Linux 和Windows 系统自带的TCP/IP 协议栈，LwIP 的功能不算完整和强大。但对于大多数物联网领域的网络应用程序，LwIP 已经足够了。

17.2 移植注意事项

PANGO Cortex-M1 嵌入式软核移植 LwIP 2.1.2 版本。

需要注意以下几点：

- PANGO Cortex-M1 硬件不具备协议校验功能，需要使用 LwIP 的软件校验功能，即：lwipopts.h 配置文件的 CHECKSUM_BY_HARDWARE 宏赋值为 0
- 替换底层驱动接口函数，主要是替换 ethernetif.c 的函数内容
- 由于描述符缓存区使用的是静态存储空间，将工程的数据段（IRAM1）起始地址修改为：0x30800000

17.3 参考设计

设计代码位于文件夹：../software_design/LwIP_V2.1.2。具体的工程代码实现，请参考工程代码。

18 Bootloader

引导程序 Bootloader 需要配合 ICACHE 或 ICACHE 与 DCACHE 一起使用，Bootloader 的作用是将应用程序从 SPI Flash 里面读出，写入到 ICACHE（DDR 颗粒）里面，并且将应用函数的中断向量表寄存器的值复制到 Bootloader 中断向量表的寄存器里面，然后初始化堆栈，最后跳转到应用程序入口地址。Bootloader 的程序流程图如下图所示：

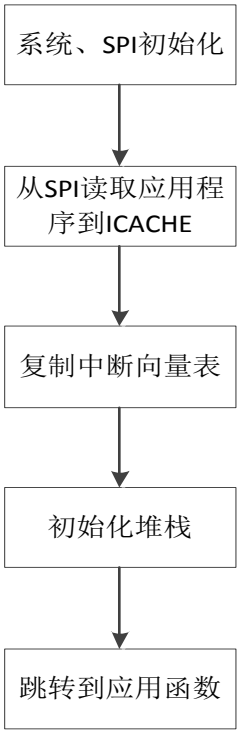

图 9 Bootloader 程序流程图

位流下载到 SPI Flash 里面，首先需要把 FPGA 位流和软核生成的*.bin 文件合成一个*.sfc 文件，然后通过 PANGO cable 下载到 FPGA 的 SPI Flash 里面。详细的操作流程请参考《FPGA Cortex-M1 SoC DEMO 应用指南》。

18.1 ICACHE 和 DCACHE 寄存器列表

ICACHE 寄存器定义列表如下所示：

表 33ICACHE 寄存列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
CTRL	0x00	RW	32	0x00000000	[31:8]: Reserved [7]: Invalidate done [6:1]: Reserved [0]: Invalidate control
STATUS	0x04	RW	32	0x00000000	[31:2]: Reserved [1]: Invalidate done [0]: Write done

DCACHE 寄存器定义列表如下所示：

表 34DCACHE 寄存器列表

寄存器名称	地址偏移	类型	宽度	初始值	描述
CTRL	0x00	RW	32	0x00000000	[31:8]: Reserved [7]: Invalidate/Flush done [6:2]: Reserved [1]: Flush control

					[0]: Invalidate control
STATUS	0x04	RW	32	0x00000000	[31:2]: Reserved [1]: Flush done [0]: Invalidate done
START_ADDR	0x08	RW	32	0x00000000	[31:0]: Invalidate/Flush data address
LEN	0x0c	RW	32	0x00000000	[31:0]: Invalidate/Flush data len

18.2 ICACHE 和 DCACHE 库函数使用方法

ICACHE 函数定义为 PANGO_M1_icache.c，使用方法如下表所示：

表 35ICACHE 库函数使用方法

函数名	描述
ICACHE_Invalidate	ICACHE 全部刷新
ICACHE_GetReadyStatus	读取 ICACHE 写数据完成标志
ICACHE_GetInvalidateStatus	读取 ICACHE 刷新完成标志

DCACHE 函数定义为 PANGO_M1_dcache.c，使用方法如下表所示：

表 36DCACHE 库函数使用方法

函数名	描述
DCACHE_Invalidate	DCACHE 全部刷新
DCACHE_InvalidateRange	指定地址段刷新 DCACHE
DCACHE_Flush	DCACHE 全部回写
DCACHE_FlushRange	指定地址段回写 DCACHE
DCACHE_GetInvalidateStatus	读取 DCACHE 刷新完成标志
DCACHE_GetFlushStatus	读取 DCACHE 写回完成标志

18.3 ICACHE 和 DCACHE 地址说明

受限于 ITCM 的大小，当代码量很大，超出了 ITCM 的最大范围时，就需要使用 Bootloader 配合 ICACHE 或 ICACHE 与 DCACHE 的设计方案。如果用户不想使用太多的 FPGA 块 RAM 来存储指令，也可以使用该方案。该方案 Bootloader 工程支持在线调试，应用工程不支持在线调试。

生成 Bootloader 可执行文件时，需要注意两个地方：

- 第一：软核应用数据位流的 SPI Flash 起始地址与 PDS 合成软核应用数据位流的起始地址相同，默认 oxc0000，对应 Bootloader 软件代码的宏定义为：SPI_FLASH_START_ADD。
- 第二：需要配置应用数据位流的大小，与实际的应用位流大小相等，以使 Bootloader 的启动时间达到最优，默认 4096*4（即：支持软核可执行*.bin 文件大小 16KB），对应 Bootloader 软件代码的宏定义为：APP_BIN_SIZE。

编译 Bootloader 工程生成的 itcm0、itcm1、itcm2、itcm3 文件需要和 RTL 一起编译，详细操作流程请参考《FPGA Cortex-M1 SoC DEMO 应用指南》。

应用程序的软件工程在指令地址的配置时需要做相应的修改，ICACHE 指令起始地址必须配置为：0x10000000，DDR 颗粒映射范围 0x10000000-0x10ffffff（16MB）；DCACHE 数据起始地址必须配置为：0x30000000，DDR 颗粒映射范围 0x30000000-0x3fffffff（256MB），具体的配置如下图所示：

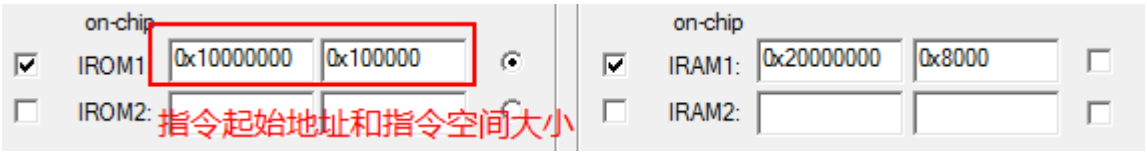

图 10 ICACHE 应用程序指令起始地址设置

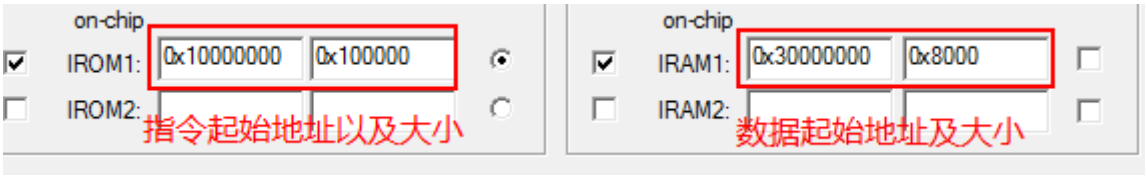

图 11 ICACHE 和 DCACHE 起始地址及大小配置

18.4 ICACHE 和 DCACHE 使用

ICACHE 可以结合 DTCM 使用，也可以结合 DCACHE 使用。

ICACHE 结合 DCACHE 使用方法有两种，分别如下：

- 应用程序指令和数据地址如图 10 所示，堆栈数据空间任然使用 DTCM，但是可以使用应用程序访问 DCACHE 的地址空间（参考代码：../software_design/boot/Cortex-M1_Icache_Dcache_my_mem），如下图所示：

```
#define DCACHE      //（注意：使用Dcache申请缓存时定义该宏定义，IRAM1起始地址需要设置为0x20000000）

#ifdef  DCACHE
    #define mem_base                (uint8_t*)0x30000000
    #define memmap_base             (uint8_t*)(0x30000000 + MEM_MAX_SIZE)
#else
    uint8_t mem_base[MEM_MAX_SIZE];
    uint8_t memmap_base[MEM_ALLOC_TABLE_SIZE];
#endif

const uint32_t memtblsize = MEM_ALLOC_TABLE_SIZE;
const uint32_t memblksize = MEM_BLOCK_SIZE;
const uint32_t memsize = MEM_MAX_SIZE;

m_mallco mallco_dev=
{
    mem_init,
    mem_perused,
    mem_base,
    memmap_base,
    0,
};
m_mallco *p_mallco_dev;
```

使用DCache申请数据缓存

图 12DCACHE 申请数据缓存

上图代码片段位于：../software_design/boot/Cortex-M1_Icache_Dcache_my_mem/USER/PANGO_my_mem.c 文件中。如图 12 所示，需要打开宏定义 DCACHE，并且在编译器里面 IRAM1 的起始地址需要配置为：0x20000000（如图 10 所示），才可以使用 DCACHE 申请数据缓存，但是不能超过 DCACHE 的最大寻址范围（0x30000000-0x3fffffff）。

- 应用程序指令和数据地址如图 11 所示，堆栈数据空间使用 DCACHE（参考代码：../software_design/boot/Cortex-M1_Icache_Dcache_Demo）

18.5 参考设计

设计代码位于文件夹：../software_design/boot/Cortex-M1_Bootloader，里面包含了 Bootloader 代码以及应用工程 Demo 代码，远程升级的代码位于文件夹：../software_design/upgrade，具体的工程代码实现，请参考工程代码。

软核支持 ICACHE 和 DCACHE，需要 RTL 代码做相应的配置，即：cm1_option_defs.v 文件里面的 CORTEXM1_AHB_ICACHE 宏和 CORTEXM1_AHB_DCACHE 宏需要打开，具体操作流程请参考《FPGA Cortex-M1 SoC DEMO 应用指南》。

19 工程固化

通过编译软件生成的可执行文件在相应的工程目录：..\Cortex-M1_xxx\PROJECT 下，文件名称分别是：itcm0、itcm1、itcm2、itcm3。最后要将这四个文件和 RTL 工程一起编译，固化到 FPGA 的外部 Flash 当中，文件具体的存放位置请参考《FPGA Cortex-M1 SoC DEMO 应用指南》。

20 在线调试

在线调试器使用野火(Fire Debugger)的调试器。正确连接调试器连线，点击在线调试按钮，进入调试模式。最多同时支持两个断点调试。使用 ICACHE 和 DCACHE 暂不支持在线调试。