介　绍

本软件著作介绍了 32 位基于ARM微控制器GVM32F030 的固件函数库。GVM32F030 系列是 ARM Cortex™ M0 核的 32 位低成本微控制器。 GVM32F030 系列控制器具备有常用外设和功能，包括高速 12 位的ADC转换器，UART串口，SPI接口，I2C总线接口，看门狗定时器（WDT）， 4 个通用计数器/定时器。

该函数库是一个固件函数包，它由程序、数据结构和宏组成，包括了微控制器所有外设的性能特征。该函数库还包括每一个外设的驱动描述和应用实例。通过使用本固件函数库，无需深入掌握细节，用户也可以轻松应用每一个外设。因此，使用本固态函数库可以大大减少用户的程序编写时间，进而降低开发成本。

因为该固件库是通用的，并且包括了所有外设的功能，所以应用程序代码的大小和执行速度可能不是最优的。对大多数应用程序来说，用户可以直接使用之，对于那些在代码大小和执行速度方面有严格要求的应用程序，该固件库驱动程序可以作为如何设置外设的一份参考资料，根据实际需求对其进行调整。

此份固件库用户手册的整体架构如下：

•定义，文档约定和固态函数库规则。

•固态函数库概述（包的内容，库的架构），安装指南，库使用实例。

•固件库具体描述：设置架构和每个外设的函数。

GVM32 库函数架构

1.GVM32 到底是什么

GVM32 的库函数是 本公司已经封装好一个软件封装库，也就是很多基础的代码，在开发产品的时候只需要直接调用这个库函数的函数接口就可以完成一系列工作；例如，你原来要自己烧饭洗衣服，现在库函数就好像一个保姆，你只需要使唤一声就有饭吃，有干净的衣服了。

2.GVM32 库函数的好处

在 51 单片机的程序开发中，我们直接配置 51 单片机的寄存器，控制芯片的工作方式，如中断，定时器等。配置的时候，我们常常要查阅寄存器表，看用到哪些配置位，为了配置某功能，该置 1 还是置 0。这些都是很琐碎的、机械的工作，因为 51 单片机的软件相对来说较简单，而且资源很有限，所以可以直接配置寄存器的方式来开发。

GVM32 库是由本公司针对 GVM32 提供的函数接口，即 API (Application Program Interface)，开发者可调用这些函数接口来配置 STM32 的寄存器，使开发人员得以脱离最底层的寄存器操作，有开发快速，易于阅读，维护成本低等优点。

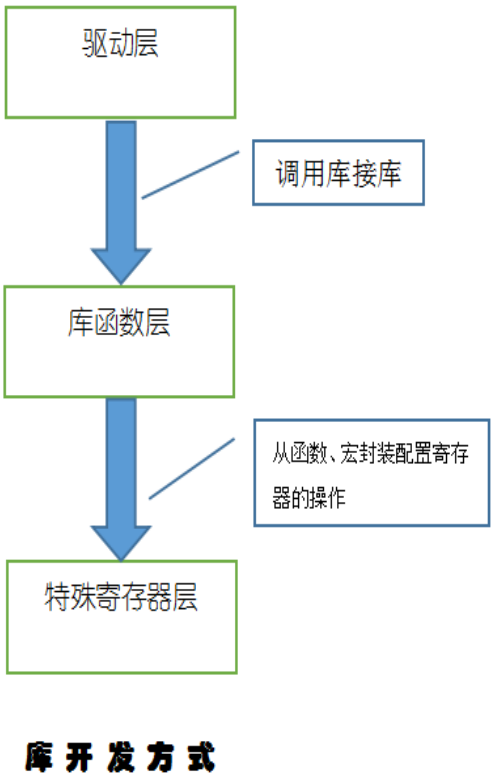


图 2-1 GVM 库开发方式

3. ARM Cortex™ -M0 内核

本公司开发的 GVM32 是基于 ARM Cortex™-M0 内核，包括 1 颗专为嵌入式应用而设计的 ARM 核、紧耦合的可嵌套中断微控制器 NVIC、可选的唤醒中断控制器 WIC，对外提供了基于 AMBA 结构（高级微控制器总线架构）的 AHB-lite 总线和基于 CoreSight 技术的 SWD 或 JTAG 调试接口。ARM Cortex™-M0 内部架构如图 3-1 所示。

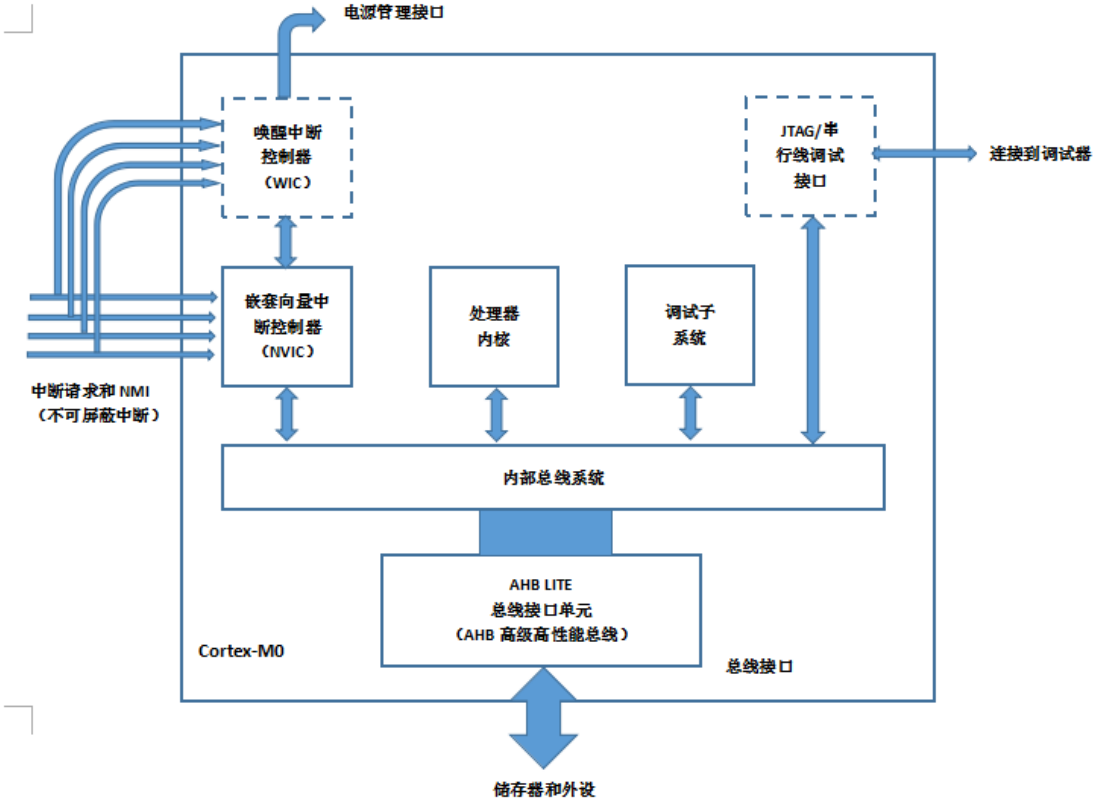


图 3-1 ARM Cortex™-M0 内部架构

4.CMSIS 层与软件架构

CMSIS 层主要由 3 个基本功能层组成，如表 4-1 所示。

CMSIS 层组成	内核外设访问层（由 ARM 负责实现）
	中间件访问层(由 ARM 负责实现，本公司针对设备进行更新)
	设备外设访问层

表 4-1 CMSIS 层主要功能层

一般来说，基于 CMISIS 标准的软件架构主要分为用户应用层、操作系统及中间件接口层、CMSIS 层、硬件外设寄存器层等，如图 4-2 所示。

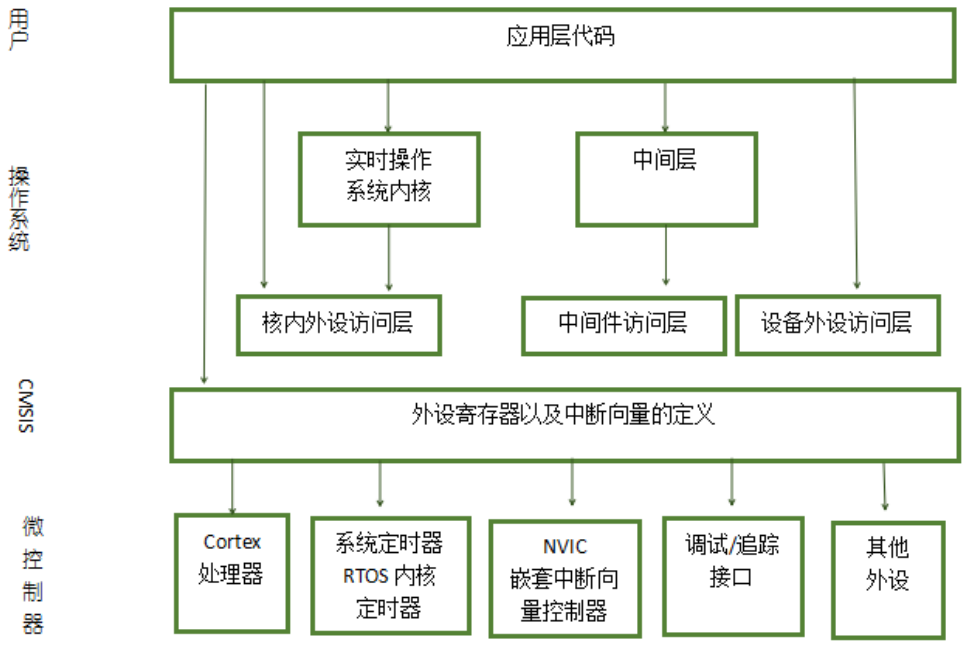


图 4-2 CMSIS 软件架构

1.文档和库规范

本用户手册和固态函数库按照以下章节所描述的规范编写。

1.1 缩写

Table 1 本文档所有缩写定义

缩写	外设/单元
ADC	数模转换器
CRC	循环冗余校验
FLASH	闪存存储器
GPIO	通用输入输出
I2C	I2C 总线
PWM	脉冲宽度调制
SPI	串行外设接口
TIM	通用定时器
UART	增强型串口
WDT	窗口看门狗

1.2 命名规则

固件函数库遵从以下命名规则。

XXX 表示任一外设缩写，例如：***ADC***。系统、源程序文件和头文件命名都以“***gvm 32f0xx _***”作为开头。

常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写。

寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，他们采用与缩写规范与本用户手册一致。

外设函数的命名以该外设的缩写加下划线为开头。每个单词的第一个字母都由英文字母大写书写，例如：***ADC_DeInit***。在函数名中，只允许存在一个下划线，用以分隔外设缩写和函数名的其它部分。

1.名为 ***XXX_Init*** 的函数，其功能是根据 ***XXX_InitTypeDef*** 中指定的参数，初始化外设 ***XXX***，例如 ***GPIO_Init***。

2.名为 ***XXX_DeInit*** 的函数，其功能为复位外设 ***XXX*** 的所有寄存器至缺省值，例如 ***XXX_DeInit***。

3.名为 ***XXX_StructInit*** 的函数，其功能为通过设置 ***XXX_InitTypeDef*** 结构中的各种参数来定义外设的功能。例如：***UART_StructInit***。

4.名为 ***XXX_Cmd*** 的函数，其功能为使能或者失能外设 ***XXX***，例如： ***SPI_Cmd***。

5.名为 ***XXX_ITConfig*** 的函数，其功能为使能或者失能来自外设 ***XXX*** 某中断源，例如：***XXX_ITConfig***。

6.名为 ***_GetFlagStatus*** 的函数，其功能为检查外设某标志位被设置与否，例如：***I2C_GetFlagStatus***。

7.名为 ***_ClearFlag*** 的函数，其功能为清除外设标志位，例如：***I2C_ClearFlag***。

8.名为 ***_GetITStatus*** 的函数，其功能为判断来自外设的中断发生与否，例如：***I2C_GetITStatus***。

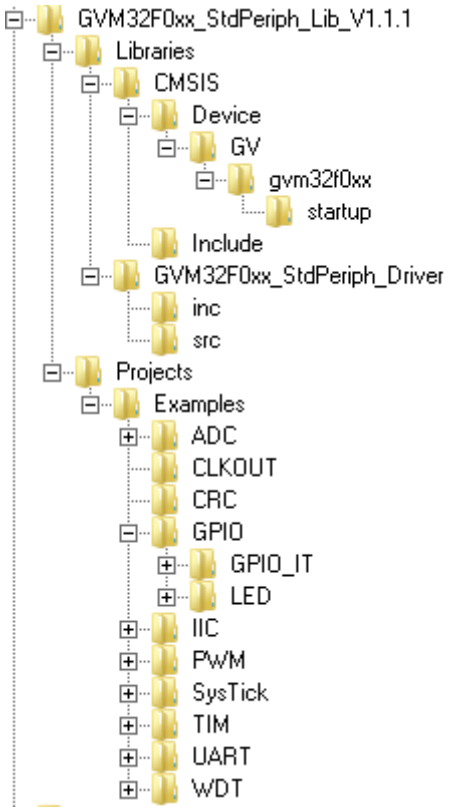
9.名为 ***_ClearITPendingBit*** 的函数，其功能为清除外设中断待处理标志位，例如：***I2C_ClearITPendingBit***。

2.固件函数库

2.1 压缩包描述

gvm32f030 固件函数库被压缩在一个 zip 文件中。解压该文件会产生一个文件夹 ：GVM32F0xx_StdPeriph_Lib_V1.1.1 。

Figure 1：固件函数库文件夹结构



2.2 文件夹 Projects

文件夹 Projects 中包含着一个子文件夹 Example，子文件夹中的 Examples，对应着每一个 GVM32 的外设，都包含着每一个子文件。这些子文件夹包含了整套文件，组成典型的例子，供读者参考学习。

2.3 文件夹 Libraries

文件夹 Libraries 中包含着两个子文件夹：CMSIS 和 GVM32F0xx_StdPeriph。每个子文件夹下又包含着两个子文件夹。

2.3.1 文件夹 CMSIS

文件夹 CMSIS 包含着两子文件：Device 和 Include。Device 的根目录中包含着 gvm32f030 的头文件和子文件，用户无需更改，只需要调用即可；include 文件夹中包含着 gvm32f030 的 Cortex-M0 的内核文件。用户也无需更改，直接调用即可。

2.3.2 文件夹 GVM32F0xx_StdPeriph

文件夹 GVM32F0xx_StdPeriph 中包含着两个子文件夹：inc 和 src。其中 inc 文件夹中包含着 gvm32f030 的外设的头文件，src 文件夹中包含着 gvm32f030 的外设的库函数。用户可以根据自己的需要选择调用。

3.外设固件概述

本节系统描述了每一个外设固件函数库。完整地描述所有相关函数并提供如何使用他们的例子。

函数的描述按如下格式进行：

Table 3.函数描述格式

函数名称	外设函数的名称
函数原形	原形声明
功能描述	简要解释函数是如何执行的
输入参数{x}	输入参数描述
输出参数{x}	输出参数描述
返回值	函数的返回值
先决条件	调用函数前应满足的要求
被调用的函数	其他该函数调用的库函数

4.模拟/数字转换器

GVM32F030 提供 12 位的 ADC 转换器，主要功能如下：

• 1M Hz 转换率， 12 位的 A/D 转换器

•支持 8 个外部 AD 通道采样转换

•模块支持低功耗掉电

• ADC 测量范围 0 ~ VDDA.

•支持突发模式 ADC 转换

•可配置 ADC 转换触发源-输入管脚电平转换或定时器匹配信号.

•每个 ADC 转换器有 8 个寄存器存储转换结果，从而减少中断负担 d.

ADC 寄存器的结构，**ADC_ TypeDef**，在文件“**gvm32f0xx.h**”中定义如下：

```
Typedef strut
{
    _IO uint32t_t CR;
    _IO uint32t_t GDR;
    _IO uint32t_t CHSEL;
    _IO uint32t_t INTEN;
    _IO uint32t_t DR[8];
    _IO uint32t_t STAT;
    _IO uint32t_t HILMT;
    _IO uint32t_t LOLMT;
       uint32t_t RESERVED0;
    _IO uint32t_t SSCR;
}ADC_TypeDef;
```

Section 4.1 ADC 寄存器结构描述了固件函数库所使用的数据结构， Section 4.2 固件库函数介绍了函数库里的所有函数。

4.1 ADC 寄存器结构

ADC 寄存器列表如 Table 4 所展示。

Table4 寄存器列表

寄存器	描述
CR	ADC 控制寄存器
GDR	ADC 全局数据寄存器
INTEN	ADC 中断使能寄存器
DR0	A/D 转换结果寄存器 0
DR1	A/D 转换结果寄存器 1
DR2	A/D 转换结果寄存器 2
DR3	A/D 转换结果寄存器 3
DR4	A/D 转换结果寄存器 4
DR5	A/D 转换结果寄存器 5
DR6	A/D 转换结果寄存器 6
DR7	A/D 转换结果寄存器 7

DR8	A/D 转换结果寄存器 8
INTSTAT	ADC 状态寄存器
HILMT	ADC 上限控制寄存器
LOLMT	ADC 下限控制寄存器
SSCR	软件触发转换控制

4.2 ADC 固件库函数

Table5 为 ADC 固件函数库列表。

Table5 ADC 固件函数库

函数名称	描述
ADC_DeInit	将 ADC 全部的外设寄存器重设为缺省值
ADC_Init	根据 ADC_InitStruct 中指定的参数初始化外设 ADC 的寄存器
ADC_ChannelCmd	使能或者失能 ADC 的通道
ADC_ITCmd	开启 ADC 所选通道的中断
ADC_ClearFlag	清除 ADC 外设的待处理标志位
ADC_HiLimit0Config	配置 ADC 的上限比较器 0
ADC_HiLimit1Config	配置 ADC 的上限比较器 1
ADC_LoLimit0Config	配置 ADC 的下限比较器 0
ADC_LoLimit1Config	配置 ADC 的下限比较器 1

4.2.1 函数 ADC_DeInit

Table6 描述了函数 ADC_DeInit。

Table6 函数 ADC_DeInit

函数名称	ADC_DeInit
函数原形	void ADC_DeInit(void)
功能描述	将外设 ADC 的全部寄存器重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例:

```
/********/
ADC_DeInit();
```

4.2.2 函数 ADC_Init

Table 7 描述了函数 ADC_Init

Table 7 函数 ADC_Init

函数名称	ADC_Init
函数原形	void ADC_Init(ADC_InitTypeDef* ADC_InitStruct)
功能描述	根据 ADC_InitStruct 中指定的参数初始化外设 ADC 的寄存器
输入参数[1]	ADC 用于外设
输入参数[2]	ADC_InitStruct：指向结构 ADC_InitTypeDef 的指针，包含了指定外设 ADC 的配置信息
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

ADC_InitTypeDef structure

ADC_InitTypeDef 定义与文件 ***gvm32f0xx_adc.h*** 内

```
typedef struct
{
    FunctionalState     ADC_BurstMode;
    uint_t              ADC_SampleRate;
    ADC_TriggerSource   ADC_TriggerSource;
    ADC_TriggerEdge     ADC_TriggerEdge;
}ADC_InitTypeDef;
```

ADC_BurstMode

ADC_BurstMode 设置 ADC 的工作模式。

ADC_BurstMode	描述
ADC_BurstMode=ENABLE	使能
ADC_BurstMode=DISABLE	失能

ADC_SampleRate

ADC_SampleRate 设置取样频率，频率控制在 100000Hz 内。

ADC_TriggerSource

ADC_TriggerSource 设置触发源，其触发源形式如下表所示。

ADC_TriggerSource	描述
ADC_TriggerSource_NO	无触发源
ADC_TriggerSource_Software	软件触发
ADC_TriggerSource_TIM2_CAP0	计数器/定时器 TIM2 输入 0 管脚
ADC_TriggerSource_TIM2_CAP1	计数器/定时器 TIM2 输入 1 管脚
ADC_TriggerSource_TIM2_MAT0	计数器/定时器 TIM2 输出 0 管脚
ADC_TriggerSource_TIM2_MAT1	计数器/定时器 TIM2 输出 1 管脚
ADC_TriggerSource_TIM3_MAT0	计数器/定时器 TIM3 输出 0 管脚
ADC_TriggerSource_TIM3_MAT1	计数器/定时器 TIM3 输出 1 管脚

ADC_TriggerEdge

ADC_TriggerEdge 设置触发模式，其触发模式如下表所示。

ADC_TriggerEdge	描述
ADC_TriggerEdge_Rising	上升沿触发
ADC_TriggerEdge_Falling	下降沿触发

4.2.3 函数 ADC_ChannelCmd

Table8 描述了函数 ADC_ChannelCmd

Table8 函数 ADC_ChannelCmd

函数名称	ADC_ChannelCmd
函数原形	void ADC_ChannelCmd(uint8_t ADC_Channel, FunctionalState NewState)
功能描述	使能或失能 ADC 指定的通道
输入参数	uint8_t ADC_Channel：所选择的通道
输出参数	FunctionalState NewState：外设 ADC 的所选通道的新状态 ENABLE：使能 DISABLE：失能
返回值	无
先决条件	无
被调用的函数	无

ADC_Channel

参数 ADC_Channel 设置的 ADC 通道，下表列举了 ADC_Channel 可选的值。

ADC_Channel	描述
ADC_Channel_0	选择 ADC 的通道 0
ADC_Channel_1	选择 ADC 的通道 1
ADC_Channel_2	选择 ADC 的通道 2
ADC_Channel_3	选择 ADC 的通道 3
ADC_Channel_4	选择 ADC 的通道 4
ADC_Channel_5	选择 ADC 的通道 5
ADC_Channel_6	选择 ADC 的通道 6
ADC_Channel_7	选择 ADC 的通道 7
ADC_Channel_AII	选择 ADC 的所有通道

例：

```
/**********/
ADC_ChannelCmd（ADC_CH1,ENABLE）;
```

4.2.4 函数 ADC_ITCmd

Table9 描述了函数 ADC_ITCmd。

Table9 函数 ADC_ITCmd

函数名称	ADC_ITCmd
函数原形	void ADC_ITCmd(uint16_t ADC_IT, FunctionalState NewState)
功能描述	使能或失能所选通道的 ADC 外设的中断

输入参数	uint16_t ADC_IT：开启 ADC 所选通道的中断
输入参数	NewState：ADC 的所选择通道的新状态 这个参数可以取： ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

ADC_IT

参数 ADC_IT 设置了 ADC 中断的通道，下表列举了 ADC_IT 可选值。

ADC_IT	描述
ADC_IT_CH0	中断通道 0
ADC_IT_CH1	中断通道 1
ADC_IT_CH2	中断通道 2
ADC_IT_CH3	中断通道 3
ADC_IT_CH4	中断通道 4
ADC_IT_CH5	中断通道 5
ADC_IT_CH6	中断通道 6
ADC_IT_CH7	中断通道 7
ADC_IT_GCH	全部中断通道

例：

```
/**********/
ADC_ITCmd(ADC_IT_CH4 | ADC_IT_CH5, ENABLE);
```

4.2.5 函数 ADC_ClearFlag

Table10 描述了函数 ADC_ClearFlag。

Table10 函数 ADC_ClearFlag

函数名称	ADC_ClearFlag
函数原形	void ADC_ClearFlag(uint32_t ADC_FLAG)
功能描述	清除 ADC 外设的待处理标志位
输入参数	uint32_t ADC_FLAG：选择清除的外设
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

ADC_FLAG

ADC_FLAG 待处理的标志位下表所示。

ADC_FLAG	描述
ADC_FLAG_CH0_DONE	ADC 通道 0 标志位
ADC_FLAG_CH1_DONE	ADC 通道 1 标志位
ADC_FLAG_CH2_DONE	ADC 通道 2 标志位
ADC_FLAG_CH3_DONE	ADC 通道 3 标志位

ADC_FLAG_CH4_DONE	ADC 通道 4 标志位
ADC_FLAG_CH5_DONE	ADC 通道 5 标志位
ADC_FLAG_CH6_DONE	ADC 通道 6 标志位
ADC_FLAG_CH7_DONE	ADC 通道 7 标志位
ADC_ FLAG_ ADINT	A/D 中断标志位
ADC_ FLAG_HILMTFLAG0	上限比较器 0 标志位
ADC_ FLAG_HILMTFLAG1	上限比较器 1 标志位
ADC_ FLAG_LOLMTELAG0	下限比较器 0 标志位
ADC_ FLAG_LOLMTETAG1	下限比较器 1 标志位
ADC_ FLAG_ADCRDY	ADC 准换器标志位

4.2.6 函数 ADC_HiLimit0Config

Table11 描述了函数 ADC_HiLimit0Config。

Table11 函数 ADC_HiLimit0Config

函数名称	**ADC_HiLimit0Config**
函数原形	void ADC_HiLimit0Config(uint16_t HiLimit0, FunctionalState INT_NEWSTATE)
功能描述	设置 ADC 上限比较器 0
输入参数	uint16_t HiLimit0：设置上限值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

4.2.7 函数 ADC_HiLimit1Config

Table12 描述了函数 ADC_HiLimit1Config。

Table12 函数 ADC_HiLimit1Config

函数名称	**ADC_HiLimit1Config**
函数原形	void ADC_HiLimit1Config(uint16_t HiLimit0, FunctionalState INT_NEWSTATE)
功能描述	设置 ADC 上限比较器 1
输入参数	uint16_t HiLimit1：设置上限值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

4.2.8 函数 ADC_LoLimit0Config

Table13 描述了函数 ADC_LOLimit0Config。

Table13 函数 ADC_LOLimit0Config

函数名称	**ADC_LoLimit0Config**
函数原形	void ADC_LoLimit0Config(uint16_t LoLimit0, FunctionalState INT_NEWSTATE)
功能描述	设置 ADC 下限比较器 0
输入参数	uint16_t LoLimit0：设置下限值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

4.2.9 函数 ADC_LOLimit1Config

Table14 描述了函数 ADC_LOLimit1Config。

Table14 函数 ADC_LOLimit1Config

函数名称	**ADC_LOLimit1Config**
函数原形	void ADC_LoLimit1Config(uint16_t LoLimit1, FunctionalState INT_NEWSTATE)
功能描述	设置 ADC 下限比较器 1
输入参数	uint16_t LoLimit0：设置下限值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

5. CRC 循环冗余

GVM32F030 集成一个微型 CRC 协处理器,它用来处理控制器的复杂计算。软件可以通过调用 CRC 寄存器去完成预设的计算功能。

CRC 系统时钟 CRC_PCLK 由控制器系统时钟提供并由 SYSAHBCLKCTRL 控制。类似控制器其它模块， CRC 可以通过关闭时钟从而节能。 xDSP 可以完成下列任务：

- CRC-CCITT
- CRC-16
- CRC-32

CRC 寄存器的结构，**CRC_ TypeDef**，在文件“**system_gvm32f0xx.h**”中定义如下：

```
Typedef struct
{
 _IO uint32_t MODE;
 _IO uint32_t SEED;
union
```

```
{
    _I uint32_t SUM;
    _O uint32_t WR_DATA_PWORD;
    _O uint32_t WR_DATA_WORD;
    _O uint16_t RESERVED_WORD;
       uint16_t WR_DATA_BYTE;
    _O uint 8_t RESERVED_BYTE[3];
    }
}CRC_TypeDef;
```

Section5.1 CRC 寄存器结构描述了固件函数库所使用的数据结构， Section 5.2 固件库函数介绍了函数库里的所有函数。

5.1CRC 寄存器

CRC 寄存器数据结构如图 Table15 所示。

Table15 CRC 寄存器数据结构

寄存器	描述
CRC_MODE	CRC 模式寄存器
CRC_SEED	CRC 种子寄存器
CRC_SUM	读：CRC 检验寄存器 写：CRC 数据寄存器

5.2 CRC 固件库函数

Table16 为 CRC 固件库函数列表。

Table16 RC 固件库函数列表

函数名称	描述
CRC_CCITT	CCITT 计算(简式：0x1021)
CRC_ModbusCalc	ModbusCalc 计算(简式：0xA001)
CRC_IBMSDLC	IBMSDLC 计算(简式：0x8005)
CRC_CalcCRC32	CalcCRC32 计算(简式：0x04C11DB7)
CRC_STM32F10X	STM32F10X 计算(简式：0x04C11DB7)

5.2.1 函数 CCITT

Table17 描述了函数 CCITT。

Table17 函数 CCITT

函数名称	CRC_CCITT
函数原型	uint16_t CRC_CCITT(uint8_t *pData, uint32_t dataLen)
功能描述	CCITT 计算(简式：0x1021)
输入参数[1]	uint8_t *pData：写入的字符
输入参数[2]	uint32_t dataLen：写入字符的长度

输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例:

```
/********/
result = CRC_CCITT(data1,6);
```

5.2.2 函数 CRC_ModbusClac

Table18 描述了函数 CRC_ModbusClac。

Table18 函数 CRC_ModbudbusClac

函数名称	**CRC_ModbusCalc**
函数原型	uint16_t CRC_ModbusCalc(uint8_t *pData, uint32_t dataLen)
功能描述	ModbusCalc 计算(简式：0xA001)
输入参数	uint8_t *pData：写入的字符
输入参数	uint32_t dataLen：写入字符的长度
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例:

```
/********/
result = CRC_ModbusCalc(data1,6);
```

5.2.3 函数 CRC_IBMSDLC

Table19 描述了函数 CRC_IBMSDLC。

Table19 函数 CRC_IBMSDLC

函数名称	**CRC_IBMSDLC**
函数原型	uint16_t CRC_IBMSDLC(uint8_t *pData, uint32_t dataLen)
功能描述	IBMSDLC 计算(简式：0x8005)
输入参数	uint8_t *pData:写入的字符
输入参数	uint32_t dataLen:写入字符的长度
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

5.2.4 函数 CRC_CalcCRC32

Table20 描述了函数 CRC_CalcCRC32。

Table20 函数 CRC_CalcCRC32

函数名称	CRC_CalcCRC32
函数原型	uint32_t CRC_CalcCRC32(uint8_t *pData, uint32_t dataLen)
功能描述	CalcCRC32 计算(简式：0x04C11DB7)
输入参数	uint8_t *pData：写入的字符
输入参数	uint32_t dataLen：写入字符的长度
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例:

```
/********/
result = CRC_CalcCRC32(data1,6);
```

5.2.5 函数 CRC_STM32F10X

Table21 描述了函数 CRC_STM32F10X。

Table21 函数 CRC_STM32F10X

函数名称	CRC_STM32F10X
函数原型	uint32_t CRC_STM32F10x(uint32_t pBuffer[],uint32_t BufferLength)
功能描述	STM32F10X 计算(简式：0x04C11DB7)
输入参数	uint32_t pBuffer[]：输入的数组
输入参数	uint32_t BufferLength：输入数组的长度
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例:

```
/********/
result = CRC_STM32F10x(buffer,3);
```

6.Flash 储存器

FLASH 寄存器结构， **FLASH_TypeDef** 在文件“**gvm32f0xx.h**”中定义如下：

```
typedef struct
{
  __IO uint32_t CR;
```

```
    __IO uint32_t DR;
    __IO uint32_t AR;
} FLASH_TypeDef;
```

Section 6.1FLASH 寄存器结构描述了固件函数库所使用的数据结构，Section 6.2 固件库函数介绍了函数库里的所有函数。

6.1 FLASH 寄存器结构

Table22 为 Flash 寄存器结构图。

Table22 Flash 寄存器结构图

寄存器	描述
CR	闪存控制寄存器
DR	闪存数据寄存器
AR	FLASH 地址寄存器

6.2 FlASH 固件库函数

Table23 描述了 Flash 固件库函数。

Table23FlASH 固件库库函数

函数名称	描述
FLASH_EraseSector	擦除一个扇区
FLASH_ProgramWord	编写一个字 4Bytes
FLASH_ProgramWordArray	编写一个数组

6.2.1 函数 FlASH_EraseSector

Table24 描述了函数 FlASH_EraseSector。

Table24 函数 FlASH_EraseSector

函数名称	Flash_EraseSector
函数原型	FLASH_Status Flash_EraseSector(uint32_t SectorAddress)
功能描述	擦除一个扇区
输入参数	uint32_t SectorAddress：待擦除的扇区
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

6.2.2 函数 FLASH_ProgramWord

Table25 描述了函数 FLASH_ProgramWord。

Table25 函数 FLASH_ProgramWord

函数名称	FLASH_ProgramWord
函数原型	FLASH_Status FLASH_ProgramWord(uint32_t Address, uint32_t Data)
功能描述	编写一个字(4bytes)
输入参数[1]	uint32_t Address：待编写的地址
输入参数[2]	uint32_t Data:待编写的字
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

6.2.3 函数 FLASH_ProgramWordArray

Table26 描述了函数 FLASH_ProgramWordArray。

Table26 函数 FLASH_ProgramWordArray

函数名称	FLASH_ProgramWordArray
函数原型	FLASH_Status FLASH_ProgramWordArray(uint32_t Address, uint32_t* Array, uint8_t Len)
功能描述	在指定地区编写一组数据
输入参数[1]	uint32_t Address：待编写的地区
输入参数[2]	uint32_t* Array：待编写的数组
输入参数[3]	Len:写入数据的长度
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

7 通用输入输出端口（GPIO）

GPIO 驱动可以用作多个用途，包括管脚设置，单位设置/重置，锁定机制，从端口管脚读入或者向端口管脚写入数据。

GPIO 寄存器结构，**GPIO_TypeDef** 和 **IOCFG_TypeDef**，在文件“**gvm32f0xx.h**”中定义如下：

```
typedef struct
{
    _IO uint32_t MASK;
    _IO  unt32_t PIN;
    _IO uint32_t OUT;
    _IO uint32_t SET;
    _IO uint32_t CLR;
```

```
    _IO uint32_t NOT;
    _IO uint32_t RESERVED0[2];
    _IO uint32_t DIR;
    _IO uint32_t IS;
    _IO uint32_t IBE:
    _IO uint32_t IEV;
    _IO uint32_t IE;
    _IO uint32_t RIS;
    _IO uint32_t MIS;
    _IO uint32_t IC;
}GPIO_TypeDef;

typedef struct
{
    _IO uint32_t PC14;
    _IO uint32_t PC15;
    _IO uint32_t PC7;
    _IO uint32_t PC8;
    _IO uint32_t PC9;
    _IO uint32_t PA0;
    _IO uint32_t PA1;
    _IO uint32_t PA2;
    _IO uint32_t PA3;
    _IO uint32_t PA4;
    _IO uint32_t PA5;
    _IO uint32_t PA6;
    _IO uint32_t PA7;
       uint32_t RESERVED0[8];
    _IO uint32_t PB0;
    _IO uint32_t PB1;
    _IO uint32_t PB2;
    _IO uint32_t PB10;
    _IO uint32_t PB11;
    _IO uint32_t PB12;
    _IO uint32_t PB13;
    _IO uint32_t PB14;
    _IO uint32_t PB15;
       uint32_t RESERVED1[4];
    _IO uint32_t PA8;
    _IO uint32_t PA9;
    _IO uint32_t PA10;
    _IO uint32_t PA11;
    _IO uint32_t PA12;
    _IO uint32_t PA13;
```

```
    _IO uint32_t PC10;
    _IO uint32_t PC11;
    _IO uint32_t PA14;
    _IO uint32_t PA15;
        uint32_t RESERVED2[4];
    _IO uint32_t PB3;
    _IO uint32_t PB4;
    _IO uint32_t PB5;
    _IO uint32_t PB6;
    _IO uint32_t PB7;
    _IO uint32_t PC12;
    _IO uint32_t PB8;
    _IO uint32_t PB9;
}IOCFG_TypeDef;
```

Section 7.1 FLASH 寄存器结构描述了固件函数库所使用的数据结构， Section 7.2 固件库函数介绍了函数库里的所有函数。

7.1GPIO 寄存器结构

Table27 列举了所有 GPIO 寄存器。

Table27 GPIO 寄存器

寄存器名称	描述
MASK	管脚屏蔽寄存器
DR	管脚状态寄存器
OUT	管脚输出值寄存器
SET	管脚输出置位寄存器
CLR	管脚输出清除寄存器
NOT	管脚输出取反寄存器
DIR	数据方向寄存器
IS	中断感应寄存器
IBE	中断边沿触发控制寄存器
IEV	中断事件寄存器
IE	中断屏蔽寄存器
RIS	原始中断状态寄存器
MIS	中断状态寄存器
IC	中断清除寄存器

7.2 GPIO 固件库函数

Table28 列举了 GPIO 的固件库函数。

Table28 GPIO 固件库函数

函数名	描述

GPIO_Init	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
GPIO_Write	向指定 GPIO 数据端口写入数据
GPIO_SetBits	设置指定 GPIO 数据端口位
GPIO_ClearBits	清除指定 GPIO 的端口的数据
GPIO_InvertBits	翻转指定 GPIO 数据端口位
GPIO_ReadOutputData	读取指定的 GPIO 端口输出
GPIO_ReadOutputDataBit	读取指定端口管脚的输出
GPIO_ReadIntputData	写入指定的 GPIO 端口输入
GPIO_ReadInputDataBit	写入指定端口管脚的输入
GPIO_SetMask	把 GPIOx 中的端口屏蔽
GPIO_GetMask	开启 GPIOx 的端口
GPIO_SetMode	选择 GPIOx 的端口引脚作为输入或输出
GPIO_GetMode	GPIOx 管脚状态的复位
GPIO_ITConfig	设置选定 GPIOx 中的端口引脚的中断形式
GPIO_ITCmd	使能或失能 GPIOx 中的端口引脚的中断
GPIO_GetRawITStatus	获取 GPIOx 中的指定端口引脚的原始中断状态
GPIO_GetITStatus	获取 GPIOx 中的指定端口引脚的当前中断状态
GPIO_ClearITpendingBit	清除 GPIOx 中的指定端口引脚的中断

7.2.1 函数 GPIO_Init

Table29 描述了函数 GPIO_Init。

Table29 GPIO_Init 库函数

函数名称	**GPIO_Init**
函数原型	void GPIO_Init(GPIOPortPin_TypeDef PortPin, GPIOMode_TypeDef Mode, uint32_t IoConfig)
功能描述	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
输入参数[1]	GPIOPortPin_TypeDef PortPin:定义端口管脚
输入参数[2]	GPIOMode_TypeDef Mode:定义端口模式
输入参数[3]	uint32_t IOConfig:定义端口配置
输出参数	无
返回值	无
先决条件	无

例:

```
/ ********  /
GPIO_Init(PB6, GPIO_Mode_OUT, IOCFG_DEFAULT);
```

GPIOMode_TypeDef Mode

参数 GPIOMode_TypeDef Mode 设置了 GPIO 的端口模式。

成员	**描述**
GPIO_Mode_IN	GPIO 输入模式
GPIO_Mode_OUT	GPIO 输出模式

7.2.2 函数 GPIO_Write

Table30 描述了函数 GPIO_Write。

Table30 函数 GPIO_Write

函数名称	GPIO_Write
函数原型	void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t Value)
功能描述	将数据写入到指定的 GPIO 端口的数据
输入参数[1]	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输入参数[2]	uint16_t Value：待写入端口数据寄存器的值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/*********/
GPIO_Write(GPIOA, 0x00001);
```

7.2.3 函数 GPIO_ClearBit

Table31 描述了函数 GPIO_ClearBit。

Table31 函数 GPIO_ClearBit

函数名称	GPIO_ClearBit
函数原型	void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
功能描述	将指定的 GPIO 的端口数据清除
输入参数[1]	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输入参数[2]	uint16_t GPIO_Pin：待清除的端口位
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
GPIO_ClearBit(GPIOB, GPIO_Pin_6);
```

GPIO_Pin

该参数选择待设置的 GPIO 管脚，使用操作符“|”可以一次选中多个管脚。可以使用下表中的任意组合。

GPIO_Pin	描述
GPIO_Pin_0	选中管脚 0
GPIO_Pin_1	选中管脚 1

GPIO_Pin_2	选中管脚 2
GPIO_Pin_3	选中管脚 3
GPIO_Pin_4	选中管脚 4
GPIO_Pin_5	选中管脚 5
GPIO_Pin_6	选中管脚 6
GPIO_Pin_7	选中管脚 7
GPIO_Pin_8	选中管脚 8
GPIO_Pin_9	选中管脚 9
GPIO_Pin_10	选中管脚 10
GPIO_Pin_11	选中管脚 11
GPIO_Pin_12	选中管脚 12
GPIO_Pin_13	选中管脚 13
GPIO_Pin_14	选中管脚 14
GPIO_Pin_15	选中管脚 15
GPIO_Pin_All	选中全部管脚

7.2.4 函数 GPIO_InvertBits

Table32 描述了函数 GPIO_InverBits。

Table32 函数 GPIO_InverBits

函数名称	**GPIO_InverBits**
函数原型	void GPIO_InvertBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
功能描述	将指定的 GPIO 的 PIN 的数据翻转
输入参数[1]	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输入参数[2]	uint16_t GPIO_Pin：待翻转的端口位
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/*******/
GPIO_InverBits(GPIOA, GPIO_Pin_1);
```

7.2.5 函数 GPIO_ReadOutputData

Table33 描述了函数 GPIO_ReadOutputData。

Table33 函数 GPIO_ReadOutputData

函数名称	**GPIO_ReadOutputData**
函数原型	uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx)
功能描述	读取指定 GPIO 端口引脚的输出，针对端口输出数据寄存器操作

输入参数	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输出参数	无
返回值	GPIO 端口的输出数据值
先决条件	无
被调用的函数	无

7.2.6 函数 GPIO_ReadOutputDataBit

Table34 描述了函数 GPIO_ReadOutputDataBit。

Table34 函数 GPIO_ReadOutputDataBit

函数名称	**GPIO_ReadOutputDataBit**
函数原型	uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
功能描述	读取指定 GPIO 端口引脚的输出
输入参数[1]	GPIO_TypeDef* GPIOx：选择 GPIO 端口
输入参数[2]	uint16_t GPIO_Pin：待读取的端口引脚位
输出参数	无
返回值	输出端口引脚值
先决条件	无
被调用的函数	无

7.2.7 函数 GPIO_ReadIntputData

Table35 描述了函数 GPIO_ReadIntputData。

Table35 函数 GPIO_ReadIntputData

函数名称	**GPIO_ReadInputData**
函数原型	uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx)
功能描述	读取指定的 GPIO 端口输入数据，针对端口输入数据寄存器操作
输入参数	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输出参数	无
返回值	GPIO 端口的输入数据值
先决条件	无
被调用的函数	无

7.2.8 函数 GPIO_ReadInputDataBit

Table36 描述了函数 GPIO_ReadIntputDataBit。

Table36 函数 GPIO_ReadIntputDataBit

函数名称	**GPIO_ReadInputDataBit**
函数原型	uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t

	GPIO_Pin)
功能描述	读取指定端口引脚的输入
输入参数[1]	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输入参数[2]	uint16_t GPIO_Pin：待写入的端口引脚位
输出参数	无
返回值	输入端口引脚值
先决条件	无
被调用的函数	无

7.2.9 函数 GPIO_SetMask

Table37 描述了函数 GPIO_SetMask。

Table37 函数 GPIO_SetMask

函数名称	**GPIO_SetMask**
函数原型	void GPIO_SetMask(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
功能描述	屏蔽 GPIOx 中的某个端口位
输入参数[1]	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输入参数[2]	uint16_t GPIO_Pin：待屏蔽的端口引脚位
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/*********/
GPIO_SetMask(GPIOA,GPIO_PIN_2);
```

7.2.10 函数 GPIO_GetMask

Table38 描述了函数 GPIO_GetMask。

Table38 函数 GPIO_GetMask

函数名称	**GPIO_GetMask**
函数原型	uint16_t GPIO_GetMask(GPIO_TypeDef* GPIOx)
功能描述	开启 GPIOx 的端口
输入参数	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

7.2.11 函数 GPIO_SetMode

Table39 描述了函数 GPIO_SetMode。

Table39 函数 GPIO_SetMode

函数名称	**GPIO_SetMod**
函数原型	void GPIO_SetMode(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_Mode_TypeDef Mode)
功能描述	选择 GPIOx 的管脚作为输入输出
输入参数[1]	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输入参数[2]	uint16_t GPIO_Pin：待设置的端口位
输入参数[3]	GPIOMode_TypeDef Mode:定义模式
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/**********/
GPIO_SetMode(GPIOA,GPIO_Pin_6,GPIO_Mode_Out);
```

7.2.12 函数 GPIO_GetMode

Table40 描述了函数 GPIO_GetMode。

Table40 函数 GPIO_GetMode

函数名称	**GPIO_GetMod**
函数原型	uint16_t GPIO_GetMode(GPIO_TypeDef* GPIOx)
功能描述	复位 GPIOx 的管脚状态
输入参数	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

7.2.13 函数 GPIO_ITConfig

Table41 描述了函数 GPIO_ITConfig。

Table41 函数 GPIO_ITConfig

函数名称	**GPIO_ITConfig**
函数原型	void GPIO_ITConfig(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIOInterrupt_TypeDef GPIO_Int)
功能描述	选择 GPIOx 中的某个端口位中断触发
输入参数[1]	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输入参数[2]	uint16_t GPIO_Pin：待设置中断的端口位

输入参数[3]	GPIOInterrupt_TypeDef GPIO_Int：定义中断形式
返回参数	无

GPIOInterrupt_TypeDef GPIO_Int

GPIOInterrupt_TypeDef GPIO_Int 用以选择用作事件输出的 GPIO 端口的中断触发形式。

GPIOInterrupt_TypeDef GPIO_Int	描述
GPIO_Interrupt_low	低电平触发
GPIO_Interrupt_High	高电平触发
GPIO_Interrupt_Rising	上升沿触发
GPIO_Interrupt_ Falling	下降沿触发
GPIO_Interrupt_Rising_Falling	延边触发

例：

```
/********/
GPIO_ITConfig（GPIOB，GPIO_PIN_2GPIO_Interrupt_Rising_Falling）；
```

7.2.14 函数 GPIO_ITCmd

Table42 描述了函数 GPIO_ITCmd。

Table42 函数 GPIO_ITCmd

函数名称	GPIO_ITCmd
函数原型	void GPIO_ITCmd(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, FunctionalState NewState)
功能描述	使能或失能 GPIOx 中的端口位的中断
输入参数[1]	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输入参数[2]	uint16_t GPIO_Pin：待设置端口位
输入参数[3]	FunctionalState NewState：定义触发还是禁止中断
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/**********/
GPIO_ITCmd(GPIOA,GPIO_PIN_1,ENABLE);
```

7.2.15 函数 GPIO_GetRawITStatus

Table44 描述了函数 GPIO_GetRawITStatus。

Table44 函数 GPIO_GetRawITStatus

函数名称	GPIO_GetRawITStatus
函数原型	ITStatus GPIO_GetRawITStatus(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
功能描述	获取 GPIOx 中的端口的原始中断状态

输入参数[1]	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输入参数[2]	uint16_t GPIO_Pin：待获取的端口位
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

7.2.16 函数 GPIO_GetITStatus

Table45 描述了函数 GPIO_GetITStatus

Table45 函数 GPIO_GetITStatus

函数名称	**GPIO_GetITStatus**
函数原型	ITStatus GPIO_GetITStatus(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
功能描述	获取 GPIOx 中的端口的当前中断状态
输入参数[1]	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输入参数[2]	uint16_t GPIO_Pin：待获取的端口位
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

7.2.17 函数 GPIO_ClearITPendingBit

Table46 描述了函数 GPIO_ClearITPendingBit。

Table46 函数 GPIO_ClearITPendingBit

函数名称	**GPIO_ClearITPendingBit**
函数原型	void GPIO_ClearITPendingBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
功能描述	清除 GPIOx 中的端口位的中断
输入参数[1]	GPIO_TypeDef* GPIOx：定义 GPIO 外设
输入参数[2]	uint16_t GPIO_Pin：待清除中断的端口位
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/*********/
GPIO_ClearITPendingBit(GPIOB,GPIO_Pin_2);
```

8. I2C

I2C 是两线串行通信接口，它与 I2C 总线兼容。可以支持主、从机两种模式的 I2C 通信方式。

I2C 寄存器的结构，**“I2C_TypeDef”**，在文件“**gvm32f0xx.h**”中定义如下：

```
typedef struct
  {
  _IO  uint32_t  CONSET;
  _I   uint32_t  STAT;
  _IO  uint32_t  DAT;
  _IO  uint32_t  ADR0;
  _IO  uint32_t  SCLH;
  _IO  uint32_t  SCLL;
  _O   uint32_t  CONCLR;
       uint32_t  RESERVED0;
  _IO  uint32_t  ADR1;
  _IO  uint32_t  ADR2;
  _IO  uint32_t  ADR3;
  _I   uint32_t  DATA_BUFFER;
  _IO  uint32_t  MASK;
  }I2C_TypeDef;
```

Section 8.1 I2C 寄存器结构描述了固件函数库所使用的数据结构， Section 8.2 固件库函数介绍了函数库里的所有函数。

8.1 I2C 寄存器结构

Table47 描述了 I2C 寄存器数据结构。

Table47 I2C 寄存器结构

寄存器	描述
CONSET	I2C 控制置位寄存器
STAT	I2C 状态寄存器
DAT	I2C 数据寄存器
ADR0	I2C 从机地址寄存器 0
SCLH	占空比寄存器高半字
SCLL	占空比寄存器低半字
CONCLR	I2C 控制清零寄存器
ADR1	I2C 从机地址寄存器 1
ADR2	I2C 从机地址寄存器 2
ADR3	I2C 从机地址寄存器 3
DATA_BUFFER	数据缓冲寄存器
MASK[4]	I2C 从属地址屏蔽寄存器 0~3

8.2 I2C 固件库函数

Table48 例举了 I2C 固件库函数。

Table48 I2C 固件库函数

函数名称	描述
I2C_DeInit	将外设的所有寄存器设置为缺省值
I2C_SetclockRate	设置 I2C 通信速率
I2C_GetClockRate	获取 I2C 通信速率
I2C_ MasterInit	主机模式初始化
I2C_SlaveInit	从机模式初始化
I2C_MasterWriteOneByte	主机模式下写入一个字节
I2C_MasterReadOneByte	主机模式下读取一个字节
I2C_MasterWriteBytes	主机模式下写入字节
I2C_MasterReadBytes	从机模式下读取字节
I2C_IRQHandler	从机状态处理程序

8.2.1 函数 I2C_DeInit

Table49 描述了函数 I2C_DeInit。

Table49 函数 I2C_DeInit

函数名称	I2C_DeInit
函数原型	void I2C_DeInit(void)
功能描述	将外设的所有寄存器设置为缺省值
输入参数[1]	SYSCON_AHBPeriphResetCmd(SYSCON_AHBRESET_I2C, ENABLE)；通信使能
输入参数[2]	SYSCON_AHBPeriphResetCmd(SYSCON_AHBRESET_I2C, DISABLE)；通信失能
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

8.2.2 函数 I2C_SetClockRate

Table50 描述了函数 I2C_SetClockRate。

Table50 函数 I2C_SetClockRate

函数名称	I2C_SetClockRate
函数原型	void I2C_SetClockRate(uint32_t ClockRate)
功能描述	设置 I2C 通信速率
输入参数[1]	uint32_t ClockRate:定义传输速率
输出参数	无
返回值	无

先决条件	无
被调用的函数	无

例:

```
/********/
I2C_SetClockRate(200000);
```

8.2.3 函数 I2C_GetClockRate

Table51 描述了函数 I2C_GetClockRate。

Table51 函数 I2C_GetClockRate

函数名称	**I2C_GetClockRate**
函数原型	uint32_t I2C_GetClockRate(void)
功能描述	获取 I2C 通信速率
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

8.2.4 函数 I2C_MasterInit

Table52 描述了函数 I2C_MasterInit。

Table52 函数 I2C_MasterInit

函数名称	**I2C_MasterInit**
函数原型	void I2C_MastarInit(void)
功能描述	主机模式初始化
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

8.2.5 函数 I2C_SlaveInit

Table53 描述了函数 I2C_SlaveInit。

Table53 函数 I2C_SlaveInit

函数名称	**I2C_SlaveInit**
函数原型	void I2C_SlaveInit(uint8_t slaveAddr, uint8_t addrMask, I2C_EventHandler_TypeDef eventHandler)
功能描述	从机模式初始化

输入参数[1]	uint8_t slaveAddr：从机地址
输入参数[2]	uint8_t addrMask：地址掩码
输入参数[3]	I2C_EventHandler_TypeDef eventHandler：事件处理程序
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例:

```
/********/
I2C_SlaveInit(0xAE, 0x00, eeprom_eventcb);
```

8.2.6 函数 I2C_MasterWriteOneByte

Table54 描述了函数 I2C_MasterWriteOneByte。

Table54 函数 I2C_MasterWriteOneByte

函数名称	**I2C_MasterWriteOneByte**
函数原型	void I2C_MasterWriteOneByte(uint8_t slvAddr, uint8_t dat)
功能描述	主机模式下写入一个字节
输入参数[1]	uint8_t slvAddr：写入数据的地址
输入参数[2]	uint8_t dat：写入字节
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例:

```
/********/
I2C_MasterWriteOneBytes(0xAE, 2);
```

8.2.7 函数 I2C_MasterReadOneByte

Table55 描述了函数 I2C_MasterReadOneByte。

Table55 函数 I2C_MasterReadOneByte

函数名称	**I2C_MasterReadOneByte**
函数原型	uint8_t I2C_MasterReadOneByte(uint8_t slvAddr)
功能描述	主机模式下读取一个字节
输入参数	uint8_t slvAddr：读取数据的地址
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

8.2.8 函数 I2C_MasterWriteBytes

Table56 描述了函数 I2C_MasterWriteBytes。

Table56 函数 I2C_MasterWriteBytes

函数名称	I2C_MasterWriteBytes
函数原型	void I2C_MasterWriteBytes(uint8_t slvAddr, uint8_t *pData , int len)
功能描述	主机模式下写入字节
输入参数[1]	uint8_t slvAddr：开始写入数据的地址
输入参数[2]	uint8_t *pData：数据数组的首地址
输入参数[3]	nt len：写入数据的长度
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/*********/
 int i;
 buffer[0] = WriteAddr;
 for(i=0; i<NumToWrite; i++)
 {
     buffer[i+1] = *(pBuffer+i);
 }

 I2C_MasterWriteBytes(0xAE, buffer, NumToWrite+1);
```

8.2.9 函数 I2C_MasterReadBytes

Table57 描述了函数 I2C_MasterReadBytes。

Table57 函数 I2C_MasterReadBytes

函数名称	I2C_MasterReadBytes
函数原型	void I2C_MasterReadBytes(uint8_t slvAddr, uint8_t *pBuffer, int rdlen)
功能描述	主机模式下读取字节
输入参数[1]	uint8_t slvAddr：开始读取数据的地址
输入参数[2]	uint8_t *pData：数据数组的首地址
输入参数[3]	int len：读取数据的长度
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
I2C_MasterReadBytes(0xAE, pBuffer, NumToRead);
```

8.2.10 函数 I2C_IRQHandler

Table58 描述了函数 I2C_IRQHandler。

Table58 函数 I2C_IRQHandler

函数名称	I2C_IRQHandler
函数原型	void I2C_IRQHandler(void)
功能描述	从机状态处理程序
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

9. PWM（脉冲宽度调制）

脉冲宽度调制是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术，广泛应用在从测量、通信到功率控制与变换的许多领域中。

I2C 寄存器的结构，**“PWM_TypeDef”**，在文件“**gvm32f0xx.h**”中定义如下：

```
typedef struct
{
_IO  uint32_t  CTRL;
_IO  uint32_t  FCTRC;
_IO  uint32_t  FLTACK;
_IO  uint32_t  OUT;
_I   uint32_t  CNTR;
_IO  uint32_t  CMOD;
_IO  uint32_t  VAL0;
_IO  uint32_t  VAL1;
_IO  uint32_t  VAL2;
_IO  uint32_t  VAL3;
_IO  uint32_t  VAL4;
_IO  uint32_t  VAL5;
_IO  uint32_t  VAL6;
_IO  uint32_t  VAL7;
_IO  uint32_t  DTIM0;
_IO  uint32_t  DTIM1;
_IO  uint32_t  DMAP0;
```

```
_IO  uint32_t  DMAP1;
_IO  uint32_t  CNFG;
_IO  uint32_t  CCTRL;
_IO  uint32_t  FPORTCTRL;
_IO  uint32_t  ICCTRL;
       uint32_t  RESEPVED0[2];
_IO  uint32_t  PSLR:
_IO  uint32_t  CNTRINT;
}PWM_TypeDef;
```

Section 9.1FLASH 寄存器结构描述了固件函数库所使用的数据结构，Section 9.2 固件库函数介绍了函数库里的所有函数。

9.1 PWM 寄存器结构

Table59 描述了 PWM 寄存器数据结构。

Table59 PWM 寄存器结构

寄存器名	描述
CLRL	控制寄存器
FCTRL	故障输入控制寄存器
FLTACk	故障输入状态寄存器
OUT	输出控制寄存器
CNTR	计数器寄存器
CMOD	模数计数器寄存器
VAL0	数值寄存器 0
VAL1	数值寄存器 1
VAL2	数值寄存器 2
VAL3	数值寄存器 3
VAL4	数值寄存器 4
VAL5	数值寄存器 5
DTIM0	死区寄存器 0
DTIM1	死区寄存器 1
DMAP0	映射失效控制寄存器 0
DMAP1	映射失效控制寄存器 1
CNFG	配置寄存器
CCTRL	输出通道控制寄存器
PORT	端口控制寄存器
ICCTRL	内部校正控制寄存器
PSCR	极性反转控制寄存器
CNTRINI	计数器的初始化

9.2 PWM 固件库库函数

Table60 描述了 PWM 固件库函数。

Table60 PWM 固件库函数

库函数	描述
PWM_DeInit	复位外设的所有寄存器为缺省值
PWM_SetPrescaler	设置预分频
PWM_ SetModulo	设置模数计数
PWM_OutputPairConfig	设置互补输出配置
PWM_Init	初始化外设
PWM_Cmd	使 PWM 输出使能或失能
PWM_SetCompare 0	PWM 数据寄存器 0 赋值于比较单元 0
PWM_SetCompare1	PWM 数据寄存器 1 赋值于比较单元 1
PWM_SetCompare2	PWM 数据寄存器 2 赋值于比较单元 2
PWM_SetCompare3	PWM 数据寄存器 3 赋值于比较单元 3
PWM_SetCompare4	PWM 数据寄存器 4 赋值于比较单元 4
PWM_SetCompare5	PWM 数据寄存器 5 赋值于比较单元 5
PWM_SetRiseDeadZone	控制 PWM 死区由低状态到高状态转换
PWM_SetFallDeadZone	控制 PWM 死区由高状态到低状态转换

9.2.1 函数 PWM_DeInit

Table61 描述了函数 PWM_DeInit。

Table61 函数 PWM_DeInit

函数名称	PWM_DeInit
函数原型	void PWM_DeInit(void)
功能描述	将外设的所用寄存器设置为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

9.2.2 函数 PWM_SetPrescaler

Table62 描述了函数 PWM_SetPrescaler。

Table62 函数 PWM_SetPrescaler

函数名称	PWM_SetPrescaler
函数原型	void PWM_SetPrescaler (PWMPrescaler_TypeDef Prescaler)
功能描述	设置外设的预分频
输入参数	PWMPrescaler_TypeDef Prescaler：定义 PWM 预分频频率
输出参数	无
返回值	无

先决条件	无
被调用的函数	无

PWMPrescaler_TypeDef Prescaler

下表描述了 PWM 预分频的分频值

PWM_Prescaler	描述
PWM_Prescaler_Div1	无分频
PWM_Prescaler_Div2	2 分频
PWM_Prescaler_Div4	4 分频
PWM_Prescaler_Div8	8 分频

例:

```
/*********/
PWM_SetPrescaler(10000);
```

9.2.3 函数 PWM_ SetModulo

Table63 描述了函数 PWM_ SetModulo。

Table63 函数 PWM_ SetModulo

函数名称	PWM_ SetModulo
函数原型	void PWM_SetModulo(uint16_t Modulo)
功能描述	设置外设模数计数
输入参数	uint16_t Modulo:模数计数值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

9.2.4 函数 PWM_OutputPairConfig

Table64 描述了函数 PWM_OutputPairConfig。

Table64 函数 PWM_OutputPairConfig

函数名称	PWM_OutputPairConfig
函数原型	void PWM_OutputPairConfig(PWMOutputPair_TypeDef OutputPair, PWMOutputMode_TypeDef OutputMode)
功能描述	设置互补输出配置
输入参数[1]	PWMOutputPair_TypeDef OutputPair：配置输出
输入参数[2]	PWMOutputMode_TypeDef OutputMode：输出模式
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

PWMOutputPair_TypeDef OutputPair

下表描述了配置 PWM 输出的几种形式。

PWM_OutputPair	描述
PWM_OutputPair01	配置通道 0 与通道 1 为互补输出
PWM_OutputPair23	配置通道 2 与通道 3 为互补输出
PWM_OutputPair45	配置通道 4 与通道 5 为互补输出

PWMOutputMode_TypeDef OutputMode

下表描述了 PWM 输出模式。

PWM_OutputMode	描述
PWM_OutputMode_ Independent	独立输出
PWM_OutputMode_Complementary	互补输出

例：

```
/********/
PWM_OutputPairConfig(PWM_OutputPair01,PWM_OutputMode_Complement
ary);
```

9.2.5 函数 PWM_Init

Table65 描述了函数 PWM_Init。

Table65 函数 PWM_Init

函数名称	PWM_Init
函数原型	void PWM_Init(PWMPrescaler_TypeDef Prescaler, uint16_t Modulo, PWMAlign_TypeDef Align)
功能描述	设置 PWM 的外设
输入参数[1]	PWMPrescaler_TypeDef Prescaler： 设置预分频
输入参数[2]	uint16_t Modulo：设置计数模式
输入参数[3]	PWMAlign_TypeDef Align：设置输出波形形式
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

PWMAlign_TypeDef Align

下表描述了设置 PWM 输出的几种形式。

PWM_Align	描述
PWM_Align_Center	中心对齐波形式
PWM_Align_Edge	边缘对齐波形式

例：

```
/********/
PWM_Init(PWM_Prescaler_Div8, 3125, PWM_Align_Edge);
```

9.2.6 函数 PWM_Cmd

Table66 描述了函数 PWM_Cmd。

Table66 函数 PWM_Cmd

函数名称	PWM_Cmd
函数原型	void PWM_Cmd(FunctionalState NewState)
功能描述	使能或失能 PWM
输入参数	FunctionalState NewState：外设 PWM 新状态
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
PWM_Cmd(ENABLE);
```

9.2.7 函数 PWM_SetCompare0

Table67 描述了函数 PWM_SetCompare 0。

Table67 函数 PWM_SetComopare 0

函数名称	PWM_SetCompare 0
函数原型	void PWM_SetCompare0(uint16_t Compare0)
功能描述	PWM 数据寄存器 0 赋值于比较单元 0
输入参数	uint16_t Compare0：比较值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
PWM_SetCompare0(1562);
```

9.2.8 函数 PWM_SetCompare1

Table68 描述了函数 PWM_SetCompare 1。

Table68 函数 PWM_SetCompare 1

函数名称	PWM_SetCompare 1
函数原型	void PWM_SetCompare1(uint16_t Compare1)
功能描述	PWM 数据寄存器 1 赋值于比较单元 1
输入参数	uint16_t Compare1：比较值
输出参数	无
返回值	无

先决条件	无
被调用的函数	无

例：

```
/********/
PWM_SetCompare1(1562);
```

9.2.9 函数 PWM_SetCompare2

Table69 描述了函数 PWM_SetCompare 2。

Table69 函数 PWM_SetCompare 0

函数名称	**PWM_SetCompare 2**
函数原型	void PWM_SetCompare2(uint16_t Compare2)
功能描述	PWM 数据寄存器 2 赋值于比较单元 2
输入参数	uint16_t Compare2：比较值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
PWM_SetCompare2(1562);
```

9.2.10 函数 PWM_SetCompare3

Table70 描述了函数 PWM_SetCompare 3。

Table70 函数 PWM_SetCompare 3

函数名称	**PWM_SetCompare 3**
函数原型	void PWM_SetCompare3(uint16_t Compare3)
功能描述	PWM 数据寄存器 3 赋值于比较单元 3
输入参数	uint16_t Compare3：比较值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
PWM_SetCompare3(1562);
```

9.2.11 函数 PWM_SetCompare4

Table71 描述了函数 PWM_SetCompare 4。

Table71 函数 PWM_SetCompare 4

函数名称	PWM_SetComopare 4
函数原型	void PWM_SetCompare4(uint16_t Compare4)
功能描述	PWM 数据寄存器 4 赋值于比较单元 4
输入参数	uint16_t Compare4：比较值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
PWM_SetCompare4(1562);
```

9.2.12 函数 PWM_SetCompare5

Table72 描述了函数 PWM_SetComopare 5。

Table72 函数 PWM_SetCompare 5

函数名称	PWM_SetCompare 5
函数原型	void PWM_SetCompare5(uint16_t Compare5)
功能描述	PWM 数据寄存器 5 赋值于比较单元 5
输入参数	uint16_t Compare5：比较值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
PWM_SetCompare5(1562);
```

9.2.13 函数 PWM_SetRiseDeadZone

Table73 描述了函数 PWM_SetRiseDeadZone。

Table73 函数 PWM_SetRiseDeadZone

函数名称	PWM_SetRiseDeadZone
函数原型	void PWM_SetRiseDeadZone(uint16_t nCycles)
功能描述	控制 PWM 死区由低状态到高状态转换
输入参数	uint16_t nCycles：由低状态到高状态的转换的时间
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
PWM_SetRiseDeadZone(200);
```

9.2.14 函数 PWM_SetFallDeadZone

Table74 描述了函数 PWM_SetFallDeadZone。

Table74 函数 PWM_SetFallDeadZone

函数名称	PWM_SetFallDeadZone
函数原型	void PWM_SetFallDeadZone (uint16_t nCycles)
功能描述	控制 PWM 死区由高状态到低状态转换
输入参数	uint16_t nCycles： 控制由高状态到低状态的准换时间
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
PWM_FallRiseDeadZone(200);
```

10. SPI(串口外设接口)

SPI，是一种高速的，全双工，同步的通信总线，并且在芯片的管脚上只占用四根线，节约了芯片的管脚，同时为 PCB 的布局上节省空间，提供方便，正是出于这种简单易用的特性，如今越来越多的芯片集成了这种通信协议。

I2C 寄存器的结构，“**SPI_TypeDef**”，在文件“**gvm32f0xx.h**”中定义如下：

```
typedef struct
{
_IO uint32_t CR0;
_IO uint32_t CR1;
_IO uint32_t DR;
_I  uint32_t SR;
_IO uint32_t CPSR;
_IO uint32_t IMSC;
_I  uint32_t RIS;
_I  uint32_t MIS;
_O  uint32_t ICR;
}SPI_TypeDef;
```

Section 10.1 SPI 寄存器结构描述了固件函数库所使用的数据结构， Section 10.2 固件库函数介绍了函数库里的所有函数。

10.1 SPI 寄存器结构

Table75 描述了 SPI 寄存器数据结构。

Table75 SPI 结构

寄存器名称	描述
CR0	控制寄存器 0
CR1	控制寄存器 1
DR	数据寄存器
SR	状态寄存器
CPSP	SPI 时钟分频寄存器
IMSC	中断屏蔽控制寄存器
RIS	原始中断状态寄存器
MiS	中断寄存器
ICR	中断清除寄存器

10.2 SPI 固件库函数

Table76 描述了 SPI 固件库函数。

Table76 SPI 固件库函数

函数名	描述
SPI_DeInit	将外设 SPIx 寄存器重设为缺省值
SPI_Structinit	把 SPI_InitStruct 中的每一个参数按缺省值填入
SPI_Init	根据 SPI_InitStruct 中指定的参数初始化外设 SPIx 寄存器
SPI_Cmd	使能或者失能 SPI 外设
SPI_WriteFIFO	数据写入 FIFO
SPI_ReadFIFO	读取 SPI 接收的 FIFO
SPI_WriteOneData	向 SPI 的写入数据检测 FIFO 的状态
SPI_ReadOneData	向 SPI 的读取数据检测 FIFO 的状态
SPI_ReadWriteData	进行一次 SPI 的数据传输

10.2.1 函数 SPI_DeInit

Table77 描述了函数 SPI_DeInit。

Table77 函数 SPI_DeInit

函数名称	SPI_DeInit
函数原型	void SPI_DeInit(void)
功能描述	将外设 SPI 重设为缺省值
输入参数	无
输出参数	无
返回值	无
先决条件	无

被调用的函数	SYSCON_AHBPeriphClockCmd

10.2.2 函数 SPI_Structinit

Table78 描述了函数 SPI_Structinit。

Table78 函数 SPI_Structinit

函数名称	**SPI_Structinit**
函数原型	void SPI_StructInit(SPI_InitTypeDef* SPI_InitStruct)
功能描述	将外设 SPIx 寄存器重设为缺省值
输入参数	SPI_InitTypeDef* SPI_InitStruct: SPI_InitStruct：指向结构_InitTypeDef 的指针，包含了外设 SPI 的配置信息。
输出参数	无
返回值	无
先决条件	无
被调用的函数	SYSCON_AHBPeriphClockCmd

10.2.3 函数 SPI_Init

Table79 描述了函数 SPI_Init。

Table79 函数 SPI_Init

函数名称	**SPI_Init**
函数原型	void SPI_Init(SPI_InitTypeDef* SPI_InitStruct)
功能描述	根据 SPI_InitStruct 中指定的参数初始化外设 SPIx 寄存器
输入参数	SPI_InitTypeDef* SPI_InitStruct: SPI_InitStruct：指向结构_InitTypeDef 的指针，包含了外设 SPI 的配置信息。
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

10.2.4 函数 SPI_Cmd

Tabl80 描述了函数 SPI_Cmd。

Table80 函数 SPI_Cmd

函数名称	**SPI_Cmd**
函数原型	void SPI_Cmd(FunctionalState NewState)
功能描述	使能或者失能 SPI 外设
输入参数	无
输出参数	无
返回值	无

先决条件	无
被调用的函数	无

10.2.5 函数 SPI_WriteFIF0

Table81 描述了函数 SPI_WriteFIF0。

Table81 函数 SPI_WriteFIF0

函数名称	**SPI_WriteFIF0**
函数原型	void SPI_WriteFIFO(uint16_t* pBuffer, uint8_t nWrite)
功能描述	数据写入 FIFO
输入参数[1]	uint16_t* pBuffer:指针，写入的地址
输入参数[2]	uint8_t nWrite:写入数据
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

10.2.6 SPI_ReadFIFO 函数

Table82 描述了函数 SPI_ReadFIFO。

Table82 函数 SPI_ReadFIFO

函数名称	**SPI_ReadFIFO**
函数原型	uint8_t SPI_ReadFIFO(uint16_t* pBuffer, uint8_t nBuffer)
功能描述	读取 SPI 接收的 FIFO
输入参数[1]	uint16_t* pBuffer:指针读取的地址
输入参数[2]	uint8_t nWrite:读取数据
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

10.2.7 函数 SPI_ReadOneData

Table83 描述了函数 SPI_ReadOneData。

Table83 函数 SPI_ReadOneData

函数名称	**SPI_ReadOneData**
函数原型	uint16_t SPI_ReadOneData(void)
功能描述	向 SPI 的写入数据检测 FIFO 状态
输入参数	无
输出参数	无

返回值	无
先决条件	无
被调用的函数	无

10.2.8 函数 SPI_ReadWriteData

Table84 描述了函数 SPI _ReadWriteData。

Table84 函数 SPI_ReadWriteData

函数名称	**SPI_ReadWriteData**
函数原型	uint16_t SPI_ReadWriteData(uint16_t data)
功能描述	向 SPI 的读取数据检测 FIFO 的状态
输入参数	uint16_t data:数据传送的地址
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

11. 基本定时器 TIM0/TIM1

GVM32F030 内置两个多功能的 16 位定时器/计数器。 定时器/计数器工作时钟由 SYSAHBCLKDIV 寄存器控制。关闭

SYSAHBCLKDIV 寄存器中定时器/计数器的时钟供给可节省系统功耗。主要功能如下：

•可预置分频的 16 位定时器/计数器

•4 个 16 位匹配寄存器:

•可产生中断

•停止定时器

•对定时器复位

Section 11.1 SPI 寄存器结构描述了固件函数库所使用的数据结构， Section 11.2 固件库函数介绍了函数库里的所有函数。

TIM 基本寄存器结构，**TIM_TypeDef**，在文件“**gvm32f0xx.h**”中定义如下：

```
typedef struct
{
    _IO  uint32_t  IR;
    _IO  uint32_t  TCR;
    _IO  uint32_t  TC;
    _IO  uint32_t  PR;
    _IO  uint32_t  PC;
    _IO  uint32_t  MCR;
        _IO  uint32_t  MR0;
    _IO  uint32_t   MR1;
    _IO  uint32_t  MR2;
    _IO  uint32_t  MR3;
```

```
}TIM_Typedef
```

11.1 TIM 基本寄存器

Table85 描述了 TIM 基本寄存器的种类。

Table85 TIM 基本寄存器种类

寄存器名称	描述
IR	中断寄存器
TCR	定时器控制寄存器
TC	定时器计数寄存器
PR	预分频寄存器
PC	预分频计数寄存器
MCR	匹配控制寄存器
MR0	匹配寄存器 0
MR1	匹配寄存器 1
MR2	匹配寄存器 2
MR3	匹配寄存器 3

11.2 TIM 基本定时器固件库函数

Table86 描述了 TIM 基本定时器库函数。

Table86 TIM 基本定时器固件库函数

库函数	描述
TIM_DeInit	将外设 TIMx 寄存器重设为缺省值
TIM_SetPrescaler	设置 TIMx 预分频值
TIM_SetCounter	设置 TIMx 计数器值
TIM_GetPrescaler	获得 TIMx 的预分频值
TIM_GetCounter	获得 TIMx 的计数器值
TIM_CounterReset	复位 TIMx 计数器和预分频器的计数器
TIM_Cmd	使能或失能 TIM 的外设
TIM_Match0Config	设置 TIMx 的匹配寄存器 0
TIM_Match1Config	设置 TIMx 的匹配寄存器 1
TIM_Match2Config	设置 TIMx 的匹配寄存器 2
TIM_Match3Config	设置 TIMx 的匹配寄存器 3
TIM_GetITStatus	获得 TIMx 的中断状态
TIM_ClearITPendingBit	清除 TIMx 的中断

11.2.1 函数 TIM_DeInit

Table87 描述了函数 TIM_DeInit。

Table87 函数 TIM_DeInit

函数名称	TIM_DeInit
函数原型	void TIM_DeInit(TIM_TypeDef* TIMx)
功能描述	将外设 TIMx 寄存器重设为缺省值
输入参数	TIM_TypeDef* TIMx: x 可以选 0，1
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
TIM_DeInit(TIM0);
```

11.2.2 函数 TIM_SetPrescaler

Table88 描述了函数 TIM_SetPrescaler。

Table88 函数 TIM_SetPrescaler

函数名称	TIM_SetPrescaler
函数原型	void TIM_SetPrescaler(TIM_TypeDef* TIMx, uint16_t Prescaler)
功能描述	设置 TIMx 预分频值
输入参数	TIM_TypeDef* TIMx：x 可以选 0，1
输出参数	uint16_t Prescaler：频率值
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
TIM_SetPrescaler(TIM, 98);
```

11.2.3 函数 TIM_SetCounter

Table89 描述了函数 TIM_SetCounter。

Table89 函数 TIM_SetCounter

函数名称	TIM_SetCounter
函数原型	void TIM_SetCounter(TIM_TypeDef* TIMx, uint16_t Counter)
功能描述	设置 TIMx 计数器值
输入参数[1]	TIM_TypeDef* TIMx：x 可以选 0，1
输入参数[2]	uint16_t Prescaler：计数值
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
TIM_SetCounter(TIM, 98);
```

11.2.4 TIM_GetPrescaler 函数

Table90 描述了函数 TIM_GetPrescaler。

Table90 函数 TIM_GetPrescaler

函数名称	TIM_GetPrescaler
函数原型	uint16_t TIM_GetPrescaler(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 预分频值
输入参数	TIM_TypeDef* TIMx：x 可以选 0，1
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

11.2.5 函数 TIM_ GetCounter

Table91 描述了函数 TIM_GetCounter。

Table91 函数 TIM_GetCounter

函数名称	TIM_GetCounter
函数原型	uint16_t TIM_GetCounter(TIM_TypeDef* TIMx)
功能描述	获得 TIMx 计数器值
输入参数	TIM_TypeDef* TIMx：x 可以选 0，1
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

11.2.6 函数 TIM_CounterReset

Table92 描述了函数 TIM_CounterReset。

Table92 函数 TIM_CounterReset

函数名称	TIM_CounterReset
函数原型	void TIM_CounterReset(TIM_TypeDef* TIMx)
功能描述	复位 TIMx 计数器和预分频器的计数器
输入参数	TIM_TypeDef* TIMx：x 可以选 0，1
输出参数	无
返回值	无
先决条件	无

被调用的函数	无

例：,

```
/********/
TIM_CounterReset(TIM0);
```

11.2.7 函数 TIM_Cmd

Table93 描述了函数 TIM_Cmd。

Table93 函数 TIM_Cmd

函数名称	**TIM_Cmd**
函数原型	void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)
功能描述	使能或失能 TIMx 外设
输入参数[1]	TIM_TypeDef* TIMx：x 可以选 0，1
输入参数[2]	FunctionStste NewState:sNewState: TIM 静默模式的新状态 这个参数可以取： ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例:

```
/********/
TIM_Cmd(TIM0,ENABLE);
```

11.2.8 函数 TIM_Match0Config

Table94 描述了函数 TIM_Match0Config。

Table94 函数 TIM_Match0Config

函数名称	**TIM_Match0Config**
函数原型	void TIM_Match0Config(TIM_TypeDef* TIMx, uint16_t Match, uint16_t TIM_MatchMode)
功能描述	设置 TIMx 的匹配寄存器 0
输入参数[1]	TIM_TypeDef* TIMx：x 可以选 0，1
输入参数[2]	uint16_t Match:输出频率值
输入参数[3]	uint16_t TIM_MatchMode：选着模式
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

TIM_MatchMode

下表给出了所可选择的模式。

TIM_MatchMode	描述
TIM_Match_Interrupt	当匹配寄存器的值与计数器的值匹配时，触发中断
TIM_Match_CounterReset	当匹配寄存器的值与计数器的值匹配时，计数器归零
TIM_Match_Stop	当匹配寄存器的值与计数器的值匹配时，定时器停止

例：

```
/********/
TIM_Match0Config(TIM0,9999,TIM_Match_Interrupt|TIM_Match_CounterReset);
```

11.2.9 函数 TIM_Match1Config

Table95 描述了函数 TIM_Match1Config。

Table95 函数 TIM_Match1Config

函数名称	TIM_Match1Config
函数原型	void TIM_Match1Config(TIM_TypeDef* TIMx, uint16_t Match, uint16_t TIM_MatchMode)
功能描述	设置 TIMx 的匹配寄存器 1
输入参数[1]	TIM_TypeDef* TIMx：x 可以选 0，1
输入参数[2]	uint16_t Match:输出频率值
输入参数[3]	uint16_t TIM_MatchMode：选择模式
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

11.2.10 函数 TIM_Match2Config

Table96 描述了函数 TIM_Match2Config。

Table96 函数 TIM_Match2Config

函数名称	TIM_Match2Config
函数原型	void TIM_Match2Config(TIM_TypeDef* TIMx, uint16_t Match, uint16_t TIM_MatchMode)
功能描述	设置 TIMx 匹配寄存器 2
输入参数[1]	TIM_TypeDef* TIMx：x 可以选 0，1
输入参数[2]	uint16_t Match:输出频率值
输入参数[3]	uint16_t TIM_MatchMode：选择模式
输出参数	无
返回值	无
先决条件	无

被调用的函数	无

11.2.11 函数 TIM_Match3Config

Table97 描述了函数 TIM_Match3Config。

Table97 函数 TIM_Match3Config

函数名称	TIM_Match3Config
函数原型	void TIM_Match3Config(TIM_TypeDef* TIMx, uint16_t Match, uint16_t TIM_MatchMode)
功能描述	设置 TIMx 的匹配寄存器 3
输入参数[1]	TIM_TypeDef* TIMx：x 可以选 0，1
输入参数[2]	uint16_t Match:输出频率值
输入参数[3]	uint16_t TIM_MatchMode：选择模式
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

11.2.12 函数 TIM_GetITStatus

Table98 描述了函数 TIM_GetITStatus。

Table98 函数 TIM_GetITStatus

函数名称	TIM_GetITStatus
函数原型	ITStatus TIM_GetITStatus(TIM_TypeDef* TIMx, uint8_t TIM_IT)
功能描述	获得 TIMx 的中断状态
输入参数[1]	TIM_TypeDef* TIMx：x 可以选 0，1
输入参数[2]	uint8_t TIM_IT：中断的地址
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

uint8_t TIM_IT

下表给出了所有的中断列表。

TIM_IT	描述
TIM_IT_MAT0	匹配寄存器 0 中断
TIM_IT_MAT1	匹配寄存器 1 中断
TIM_IT_MAT2	匹配寄存器 2 中断
TIM_IT_MAT3	匹配寄存器 3 中断
TIM_IT_CAP0	捕捉寄存器 0 中断
TIM_IT_CAP1	捕捉寄存器 1 中断
TIM_IT_CAP2	捕捉寄存器 2 中断

TIM_IT_CAP3	捕捉寄存器 3 中断

11.2.13 TIM_ClearITPendingBit 库函数

Table99 描述了函数 TIM_ClearITPendingBit。

Table99 函数 TIM_ClearITPendingBit

函数名称	**TIM_ClearITPendingBit**
函数原型	void TIM_ClearITPendingBit(TIM_TypeDef* TIMx, uint8_t TIM_IT)
功能描述	清除 TIMx 的中断
输入参数[1]	TIM_TypeDef* TIMx：x 可以选 0，1
输入参数[2]	uint8_t TIM_IT：中断的地址
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

12. 16 位定时器/计数器 TIM2/TIM3

GVM32F030 内置两个多功能的 16 位定时器/计数器。定时器/计数器工作时钟由 SYSAHBCLKDIV 寄存器控制。关闭

SYSAHBCLKDIV 寄存器中定时器/计数器的时钟供给可节省系统功耗。主要功能如下：

- 可预置分频的 16 位定时器/计数器
- 增强的定时器/计数器:
  - 沿计数
  - 门控计数
  - 正交计数
  - 触发计数
  - 带符号计数
- 16 位信号捕捉定时器，可触发中断和信号测量
- 4 个 16 位匹配寄存器:
  - 可产生中断。
  - 停止定时器。
  - 对定时器复位。
- 有两个外部输出对应匹配事件发生时:
  - 当匹配发生时，输出低电平。
  - 当匹配发生时，输出高电平。
  - 当匹配发生时，触发事件。

TIM16 位寄存器/计数器结构，**TIM_TypeDef**，在文件“**gvm32f0xx.h**”中定义如下：

```
typedef struct
{
    _IO  uint32_t  IR;
    _IO  uint32_t  TCR;
```

```
    _IO  uint32_t  TC;
    _IO  uint32_t  PR;
    _IO  uint32_t  PC;
    _IO  uint32_t  MCR;
        _IO  uint32_t  MR0;
    _IO  uint32_t  MR1;
    _IO  uint32_t  MR2;
    _IO  uint32_t  MR3;
    _IO  uint32_t  CCR;
    _IO  uint32_t  CR0;
    _IO  uint32_t  CR1;
    _IO  uint32_t  CR2;
    _IO  uint32_t  CR3;
    _IO  uint32_t  EMR;
         uint32_t  RESERVEDO[12];
    _IO  uint32_t  CTCR;
}TIM_Typedef
```

Section 12.1 SPI 寄存器结构描述了固件函数库所使用的数据结构。

12.1 16 位定时器/计数器的寄存器

Table100 描述了 TIM16 位定时器/计数器的寄存器种类。

Table100 TIM16 位定时器/计数器的寄存器种类

寄存器名称	描述
IR	中断寄存器
TCR	定时器控制寄存器
TC	定时器计数寄存器
PR	预分频寄存器
PC	预分频计数寄存器
MCR	匹配控制寄存器
MR0	匹配寄存器 0
MR1	匹配寄存器 1
MR2	匹配寄存器 2
MR3	匹配寄存器 3
CCR	信号捕捉控制寄存器
CR0	捕捉寄存器 0
CR1	捕捉寄存器 1
CR2	捕捉寄存器 2
CR3	捕捉寄存器 3
EMR	外部匹配寄存器
-	-
CTCR	计数器控制寄存器

16 位定时器/计数器固件库函数与基本定时器固件库函数基本一致，详情请请参考基本定时器。在编写程序时只有部分需要运用寄存器进行编写。

13.增强型串口（UART）

GVM32F030 提供 2 个带 16 字节 FIFO 缓存器的 UART 外设： UART0,UART1 。 串行接口都支持红外传输（ IrDA） 协议功能。时钟都受 SYSAHBCLKCTRL 寄存器控制。同时每个 UART 有独立的时钟分频器使之不受系统时钟影响。

I2C 寄存器的结构，**“UART_TypeDef”**，在文件**“gvm32f0xx.h”**中定义如下：

```
typedef struct
{
    _IO uint32_t  DR;
    _IO uint32_t  SR;
    _IO uint32_t  CR;
    _IO uint32_t  ISR;
    _IO uint32_t  BAUDDIV;
    _IO uint32_t  FIFOCLR;
    }
```

Section 13.1 SPI 寄存器结构描述了固件函数库所使用的数据结构， Section 13.2 固件库函数介绍了函数库里的所有函数。

13.1 UART 寄存器

Table101 描述了 UART 寄存器种类。

Table101 UART 寄存器种类

寄存器名称	描述
RDR	接受缓冲寄存器
TDR	发送保持寄存器
STATE	RX 和 TX FIF0 状态
CTRL	UART 中断使能控制寄存器
INTSTATUS	RX 和 TX 中断状态寄存器
BAODDIV	波特率控制寄存器
FIF0CLR	TX/RX FIF0 清除寄存器

13.2 固件库函数 UART

Table102 描述了固件库函数 UART。

Table102 固件库函数 UART

库函数	描述

UART_DeInit	将外设 USART 寄存器重设为缺省值
UART_Init	根据 USART_InitStruct 中指定的参数初始化外设 UARTx 寄存器
UART_WriteFIFO	FIFO 中的数据写入数据缓冲区
UART_ReadFIFO	读取数据缓冲区的 FIFO 数据
UART_ClearFIFO	在 FIFO 中清除寄存器 RX 和 TX
UART_Send	数据发送
UART_ITConfig	使能或者失能指定的 UART 中断
UART_GetFlagStatus	检查指定的 UART 标志位设置与否
UART_ClearFlag	清除 UART 标志位
UART_GetITStatus	使能或者失能指定的 UART
UART_ClearITPendingBit	清除中断的传送位

13.2.1 函数 UART_DeInit

Table103 描述了函数 UART_DeInit。

Table103 函数 UART_DeInit

函数名称	**UART_DeInit**
函数原型	void UART_DeInit(UART_TypeDef* UARTx)
功能描述	将外设 UARTx 寄存器重设为缺省值
输入参数	UART_TypeDef* UARTx：x 可以选 0，1
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

13.2.2 函数 UART_Init

Table104 描述了函数 UART_Init。

Table104 函数 UART_Init

函数名称	**UART_Init**
函数原型	void UART_Init(UART_TypeDef* UARTx, uint32_t Baud, uint32_t Parity, uint32_t Mode)
功能描述	根据 UART_InitStruct 中指定的参数初始化外设 UARTx 寄存器
输入参数[1]	UART_TypeDef* UARTx：x 可以选 0，1
输入参数[2]	uint32_t Baud:波特率
输入参数[3]	uint32_t Parity: 奇偶校验位
输入参数[4]	uint32_t Mode:选择模式
输出参数	无
返回值	无
先决条件	无

被调用的函数	无

uint32_t Parity

该参数设置了奇偶校验。

uint32_t Parity	描述
UART_Parity_No	不进行奇偶校验
UART_Parity_Even	进行偶数校验
UART_Parity_odd	进行奇数校验

uint32_t Mode

该参数设置了通信单元的模式。

Uint32_t Mode	描述
UART_Mode_Rx	通信接受模式
UART_Mode_Tx	通信输出模式

例:

```
/********/
UART_Init(UART0, 9600, UART_Parity_No, UART_Mode_Rx | UART_Mode_Tx);
```

13.2.3 函数 UART_WriteFIFO

Table105 描述了函数 UART_WriteFIFO。

Table105 函数 UART_WriteFIFO

函数名称	UART_WriteFIFO
函数原型	void UART_WriteFIFO(UART_TypeDef* UARTx, uint8_t Data)
功能描述	FIFO 中的数据写入数据缓冲区
输入参数[1]	UART_TypeDef* UARTx：x 可以选 0，1
输入参数[2]	uint32_t Data:写入缓冲区的数据
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

13.2.4 函数 UART_ReadFIFO

Table106 函数描述了 UART_ReadFIFO。

Table106 函数 UART_ReadFIFO

函数名称	UART_ReadFIFO
函数原型	uint8_t UART_ReadFIFO(UART_TypeDef* UARTx)
功能描述	读取数据缓冲区的 FIFO 中的数据
输入参数	UART_TypeDef* UARTx：x 可以选 0，1
输出参数	无
返回值	无
先决条件	无

被调用的函数	无

13.2.5 函数 UART_ClearFIFO

Table107 描述了函数 UART_ClearFIFO。

Table107 函数 UART_ClearFIFO

函数名称	UART_ClearFIFO
函数原型	void UART_ClearFIFO(UART_TypeDef* UARTx, uint32_t UART_FIFO)
功能描述	在 FIFO 中清除寄存器 RX 和 TX
输入参数[1]	UART_TypeDef* UARTx：x 可以选 0，1
输入参数[2]	uint32_t UART_FIFO：所清除的 FIFO 地址
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

uint32_t UART_FIFO

该参数设置了要清除数据缓存区的数据类型。

UART_FIFO	描述
UAST_RX_FIFO	通信发射单元 FIFO 数据缓存
UAST_TX_FIFO	通信接收单元 FIFO 数据缓存

13.2.6 函数 UART_Send

Table108 描述了函数 UART_Send。

Table108 函数 UART_Send

函数名称	UART_Send
函数原型	void UART_Send(UART_TypeDef* UARTx, const uint8_t* txBuf, uint16_t txLen)
功能描述	发送数据
输入参数[1]	UART_TypeDef* UARTx：x 可以选 0，1
输入参数[2]	const uint8_t* txBuf：发送的文本缓存
输入参数[3]	uint16_t txLen:发送的文本长度
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
UART_Send(UART0, text, sizeof(text));
```

13.2.7 函数 UART_ITConfig

Table109 描述了函数 UART_ITConfig。

Table109 函数 UART_ITConfig

函数名称	UART_ITConfig
函数原型	void UART_ITConfig(UART_TypeDef* UARTx, uint8_t UART_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 UART 中断
输入参数[1]	UART_TypeDef* UARTx：x 可以选 0，1
输入参数[2]	uint8_t UART_IT：中断的类型
输入参数[3]	FunctionStste NewState:sNewState: UART 静默模式的新状态 这个参数可以取： ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

uint8_t UART_IT

输入参数 UART_IT 使能或者失能 USART 的中断。可以取下表的一个或者多个取值的组合作为该参数的值。

UART_IT	描述
UART_IT_TXE	发送空状态中断
UART_IT_RXNE	接受非空状态中断
UART_IT_TXF	发送 FIFO 满状态中断
UART_IT_RXF	接受 FIFO 满状态中断
UART_IT_TXHLF	发送 FIFO 半满状态中断
UART_IT_RXHIF	接受 FIFO 半满状态中断
UART_IT_PARIERR	奇偶校验错误状态中断
UART_IT_OVERRUN	重载错误中断

例：

```
/********/
UART_ITConfig(UART0, UART_IT_RXNE, ENABLE);
```

13.2.8 函数 UART_GetFlagStatus

Table110 函数描述了 UART_GetFlagStatus。

Table110 函数 UART_GetFlagStatus

函数名称	UART_GetFlagStatus
函数原型	FlagStatus UART_GetFlagStatus(UART_TypeDef* UARTx, uint8_t UART_FLAG)
功能描述	使能或者失能指定的标志位的中断
输入参数[1]	UART_TypeDef* UARTx：x 可以选 0，1
输入参数[2]	uint8_t UART_IT：中断的类型
输入参数[3]	FunctionStste NewState:sNewState: UART 静默模式的新状态

	这个参数可以取： ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

uint8_t UART_FLAG

下表给出了所有标志位列表。

UART_FLAG	描述
UART_FLAG_TXNE	发送 FIFO 空状态标志位
UART_FLAG_RXNE	接收 FIFO 非空状态标志位
UART_FLAG_TXF	发送 FIFO 满状态标志位
UART_FLAG_RXF	接受 FIFO 满状态标志位
UART_FLAG_TXHLF	发送 FIFO 半满状态标志位
UART_FLAG_RXHLF	接受 FIFO 半满状态标志位
UART_FLAG_PARIERR	奇偶校验状态标志位
UART_FLAG_OVERRUN	接受缓存器溢出状态标志位

13.2.9 函数 UART_ClearFlag

Table111 描述了函数 UART_ClearFlag。

Table111 函数 UART_ClearFlag

函数名称	UART_ClearFlag
函数原型	VoidUART_ClearFlag(UART_TypeDef* UARTx, uint8_t UART_FLAG)
功能描述	清除 UART 标志位
输入参数[1]	UART_TypeDef* UARTx：x 可以选 0，1
输入参数[2]	uint8_t UART_FLAG：标志位的类型
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

13.2.10 函数 UART_GetITStatus

Table112 描述了函数 UART_GetITStatus。

Table112 函数 UART_GetITStatus

函数名称	UART_GetITStatus
函数原型	ITStatus UART_GetITStatus(UART_TypeDef* UARTx, uint8_t UART_IT)
功能描述	使能或者失能指定的 UART
输入参数[1]	UART_TypeDef* UARTx：x 可以选 0，1
输入参数[2]	uint8_t UART_IT：中断的类型

输出参数	无
返回值	无
先决条件	无
被调用的函数	无

13.2.11 函数 UART_ClearITPendingBit

Table113 描述了函数 UART_ClearITPendingBit。

Table113 函数 UART_ClearITPendingBit

函数名称	**UART_ClearITPendingBit**
函数原型	void UART_ClearITPendingBit(UART_TypeDef* UARTx, uint8_t UART_IT)
功能描述	清除中断的传送位
输入参数[1]	UART_TypeDef* UARTx：x 可以选 0，1
输入参数[2]	uint8_t UART_IT：中断的类型
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

14.WDT 窗口看门狗

看门狗定时器用于在用户程序出错并无法喂狗后对系统进行中断和复位处理。使用可编程的看门狗定时器，用户可改变定时器时间去应对不同的应用程序。该看门狗定时器有如下主要功能：

- 独立的频率可以设定的看门狗时钟振荡器
- 看门狗定时器可触发中断或复位
- 支持低功耗模式

WDT 窗口看门狗寄存器结构，**WDT_TypeDef**，在文件“**gvm32f0xx.h**”中定义如下：

```
typedef struct
{
_IO uint32_t MOOD;
_IO uint32_t TC;
_IO uint32_t FEED;
_I  uint32_t TV;
_IO uint32_t CLKSEL;
_IO uint32_t WARNINT;
_IO uint32_t WINDOW;
}WDT_TypeDef
```

Section 14.1 WDT 寄存器结构描述了固件函数库所使用的数据结构， Section 14.2 固件库函数介绍了函数库里的所有函数。

14.1 WDT 看门狗寄存器

Table114 描述了 WDT 看门狗寄存器。

Table114 WDT 看门狗寄存器

寄存器	描述
MOD	看门狗模式寄存器
TC	看门狗定时器常数寄存器
FEED	看门狗喂狗命令寄存器
TV	看门狗定时器寄存器
CLKSEL	看门狗时钟源选择寄存器
WARNINT	看门狗警告中断比较寄存器
WINDOW	看门狗窗口寄存器

14.2 WDT 看门狗固件库函数

Table115 描述了 WDT 看门狗固件库函数。

Table115 WDT 看门狗固件库函数

库函数	描述
WDT_DeInit	将外设 WDT 寄存器重设为缺省值
WDT_ClockConfig	配置看门狗计数器的时钟
WDT_SetMode	配置看门狗的模式
WDT_SetReload	配置看门狗的定时常数
WDT_ Cmd	使能或失能外设 WDT 寄存器
WDT_Feed	喂狗
WDT_SetWarningValue	配置看门狗警告中断比较值寄存器
WDT_SetWindowValue	配置看门狗窗口寄存器
WDT_GetITStatus	获得中断状态
WDT_ClearITPendingBit	清除中断标志位

14.2.1 函数 WDT_DeInit

Table116 描述了函数 WDT_DeInit。

Table116 函数 WDT_DeInit。

函数名称	WDT_DeInit
函数原型	void WDT_DeInit(void)
功能描述	将外设 WDT 寄存器重设为缺省值
输入参数	无
输出参数	无
返回值	无

先决条件	无
被调用的函数	无

14.2.2 函数 WDT_ClockConfig

Table117 描述了函数 WDT_ClockConfig。

Table117 函数 WDT_ClockConfig

函数名称	**WDT_ClockConfig**
函数原型	void WDT_ClockConfig(WDTClockSource_TypeDef ClockSource, uint16_t DivSel)
功能描述	配置看门狗计数器的时钟
输入参数[1]	WDTClockSource_TypeDef ClockSource：设置频率
输入参数[2]	uint16_t DivSel：设置周期
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

WDTClockSource_TypeDef ClockSource

下表给出了所取 的频率值。

WDT_ClockSource	**描述**
WDT_ClockSource_IRC	内部 IRC 晶振
WDT_ClockSource_32KHz	外部 32KHz

例：

```
/********/
WDT_ClockConfig(WDT_ClockSource_32KHz, 1);
```

14.2.3 函数 WDT_SetMode

Table118 描述了函数 WDT_SetMode。

Table118 函数 WDT_SetMode

函数名称	**WDT_SetMode**
函数原型	void WDT_SetMode(WDTMode _TypeDef Mode)
功能描述	配置看门狗的模式
输入参数	WDTMode_TypeDef Mode：看门狗的模式
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

WDTMode

下表描述了看门狗的模式。

WDT_Mode	**描述**

WDT_Mode_Interrupt	中断模式
WDT_Mode_ ChipReset	芯片复位模式

14.2.4 函数 WDT_SetReload

Table119 描述了函数 WDT_SetReload。

Table119 函数 WDT_SetReload

函数名称	**WDT_SetReload**
函数原型	void WDT_SetReload(uint32_t Reload)
功能描述	配置看门狗的定时常数
输入参数	WDTMode_TypeDef Mode：定时常数
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
WDT_SetReload(4000);
```

14.2.5 函数 WDT_ Cmd

Table120 描述了函数 WDT_ Cmd。

Table120 函数 WDT_ Cmd

函数名称	**WDT_ Cmd**
函数原型	void WDT_Cmd(FunctionalState NewState)
功能描述	使能或失能外设 WDT 寄存器
输入参数	FunctionalState NewState：WDT 静默模式的新状态 这个参数可以取： ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

例：

```
/********/
WDT_Cmd(ENABLE);
```

14.2.6 函数 WDT_Feed

Table121 描述了函数 WDT_Feed。

Table121 函数 WDT_Feed

函数名称	WDT_Feed
函数原型	void WDT_Feed(void)
功能描述	喂狗
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

14.2.7 函数 WDT_SetWarningValue

Table 122 描述了函数 WDT_SetWarningValue。

Table 122 函数 WDT_SetWarningValue

函数名称	WDT_SetWarningValue
函数原型	void WDT_SetWarningValue(uint16_t Value)
功能描述	配置看门狗警告中断比较值寄存器
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

14.2.8 函数 WDT_SetWindowValue

Table123 描述了函数 WDT_SetWindowValue。

Table123 函数 WDT_SetWindowValue

函数名称	WDT_SetWindowValue
函数原型	void WDT_SetWindowValue(uint32_t WindowValue)
功能描述	配置看门狗窗口寄存器
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

14.2.9 WDT_GetITStatus 函数

Table124 描述了函数 WDT_GetITStatus。

Table124 函数 WDT_GetITStatus

函数名称	WDT_GetITStatus

函数原型	ITStatus WDT_GetITStatus(uint8_t WDT_IT)
功能描述	获得中断状态
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

14.2.10 函数 WDT_ClearITPendingBit

Table125 描述了函数 WDT_ClearITPendingBit。

Table125 函数 WDT_ClearITPendingBit

函数名称	WDT_ClearITPendingBit
函数原型	void WDT_ClearITPendingBit(uint8_t WDT_IT)
功能描述	清除中断标志位
输入参数	无
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

15. Cortex 系统定时器（SysTick）

SysTick 提供 1 个 24 位、降序、零约束、写清除的计数器，具有灵活的控制机制。

15.1 Cortex 系统定时器（SysTick）寄存器

Table126 描述了 SysTick 的寄存器结构。

Table126 SysTick 寄存器结构

寄存器	描述
CTRL	控制寄存器
LOAD	重载寄存器
VAL	当前值寄存器
CALIB	校准值寄存器

15.2 Cortex 系统定时器（ SysTick）固件库函数

Table127 描述了 SysTick 固件库函数。

Table127 SysTick 固件库函数

库函数	描述
SysTick_CLKSoureConfig	设置 SysTick 时钟源
SysTick_Delay_us	微妙延时
SysTick_Delay_ms	毫秒延时

15.2.1 函数 SysTick_CLKSoureConfig

Table128 描述了函数 SysTick_CLKSoureConfig。

Table128 函数 SysTick_CLKSoureConfig

函数名称	SysTick_CLKSoureConfig
函数原型	void SysTick_CLKSourceConfig(uint32_t SysTick_CLKSource)
功能描述	设置 SysTick 时钟源
输入参数	uint32_t SysTick_CLKSource：AHB 时钟作为 SysTick 时钟源
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

15.2.2 函数 SysTick_Delay_us

Table129 描述了函数 SysTick_Delay_us。

Table129 函数 SysTick_Delay_us

函数名称	SysTick_Delay_us
函数原型	void SysTick_Delay_us(uint32_t nus)
功能描述	微妙延时
输入参数	uint32_t nus：延时时间
输出参数	无
返回值	无
先决条件	无
被调用的函数	无

15.2.3 函数 SysTick_Delay_ms

Table130 描述了函数 SysTick_Delay_ms。

Table130 函数 SysTick_Delay_ms

函数名称	SysTick_Delay_ms
函数原型	void SysTick_Delay_ms (uint32_t nms)
功能描述	毫秒延时
输入参数	uint32_t nms：延时时间
输出参数	无

返回值	无
先决条件	无
被调用的函数	无

例:

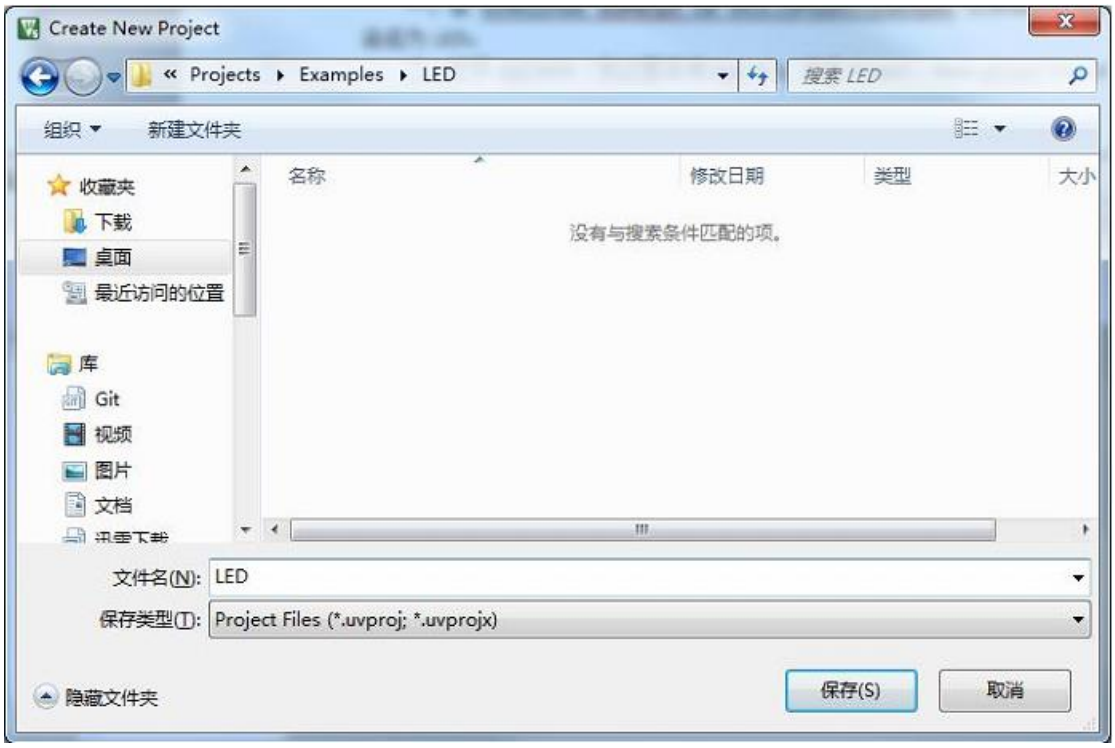
```
/********/
int mian(void)
{
    SystemCoreClockUpdate();
    GPIO_Init(PB6,GPIO_Mode_Out,IOCFG_DEFAULT);
    GPIO_SetBits(GPIOB,GPIO_Pin_6);
    while(1)
    {
        SysTick_Delay_us(100);
        GPIO_InvertBits(GPIOB,GPIO_Pin_6);
    }
}
void SysTick_Delay_ms(uint32_t nms)
{
    uint32_t temp;
    SysTick->LOAD=num*12500;
    SysTick->VAL=0x00;
    SysTick->CTRL=SysTick_CTRL_ENABLE_Mak;
    do
    {
        temp=SysTick->CTRL;
    }while((temp&0x01)&&!(temp&SysTick_CTRL_COUNTELAG_Mak));
    SysTick->CTRL&=~SysTick_CTRL_ENABLE_Mak;
    SysTick->VAL=0x00;
}
```

16.GVM32F030 工程文件建立向导

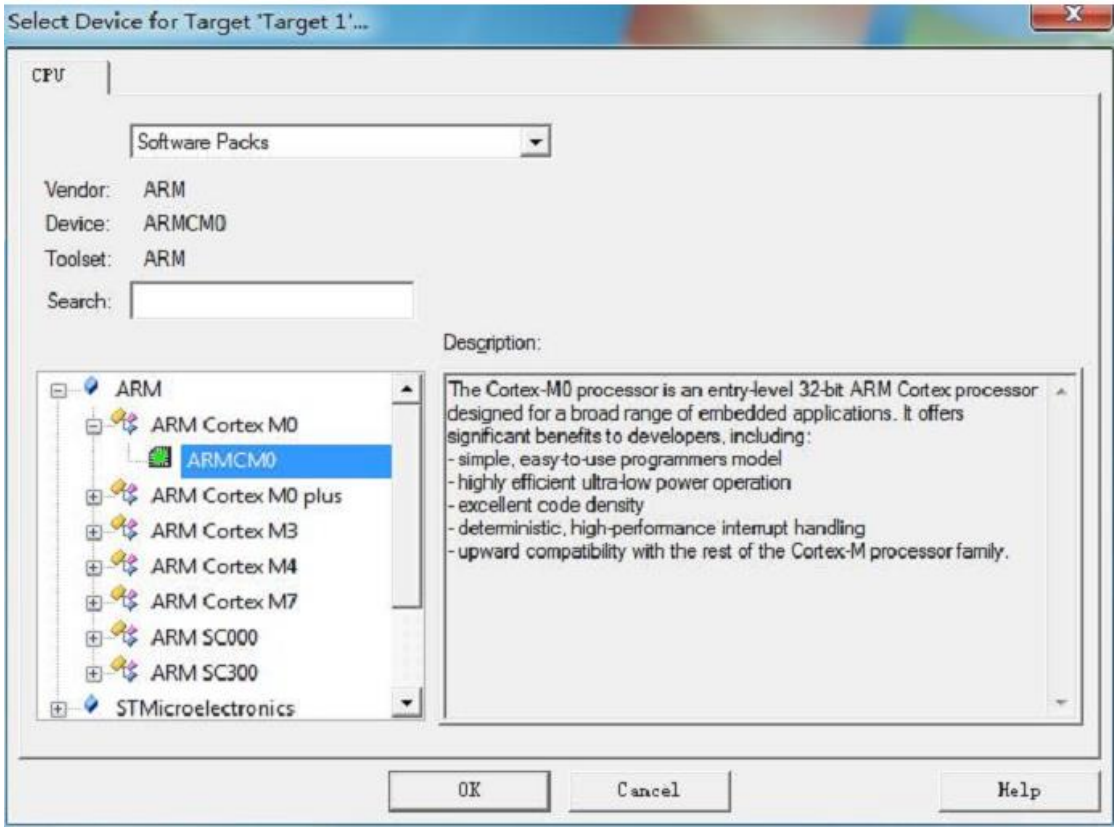
本文以一个 LED 例程的建立来讲解 GVM32F030 工程的建立工程。

一、在 ***GVM32F0xx_StdPeriph_Lib_V0.0.1\Projects\Examples*** 文件夹中新建一个文件夹，命名为 LED。

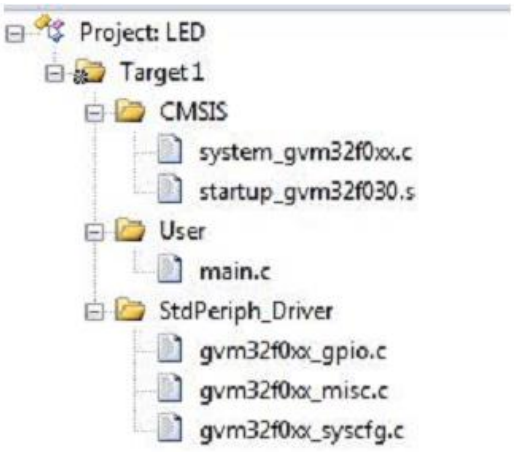
二、 打开 Keil MDK（我这里采用 Keil v5）； 点击 ***Project -- New uVision Project；*** 选择工程文件的路径（刚才新建的文件夹下）， 命名工程为 LED； 点击保存。



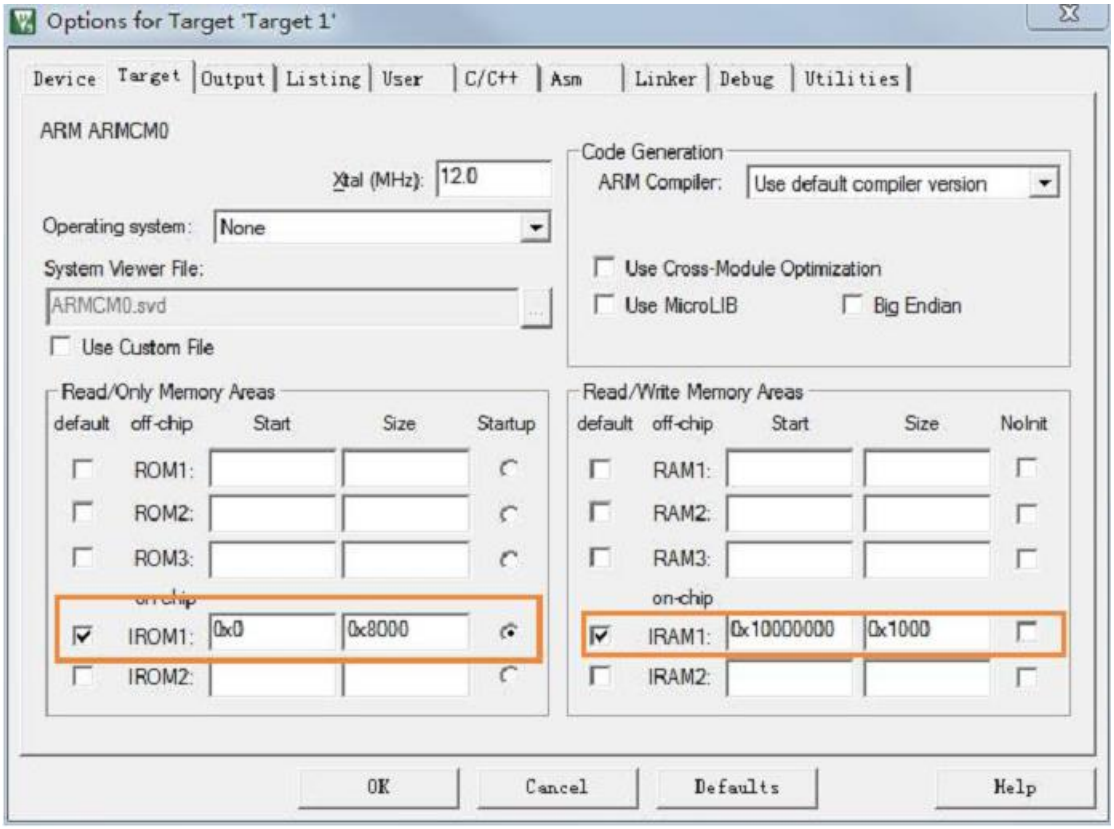
三、 在 ***Select Device for Target ‘Target 1’*** 对话框中， 选择 ***ARM -- Cortex M0*** 内核，然后点击 OK。



四、对工程添加固件库文件。固件库文件在 ***GVM32F0xx_StdPeriph_Lib_V0.0.1\Libraries*** 文件夹下， 新建 main.c（可从 Examples 中拷贝） 最终的工程文件结构如下：



五、 配置项目 的 Flash 和 RAM 的地址范围， 打开项目 配置， 按如下图所示配置：

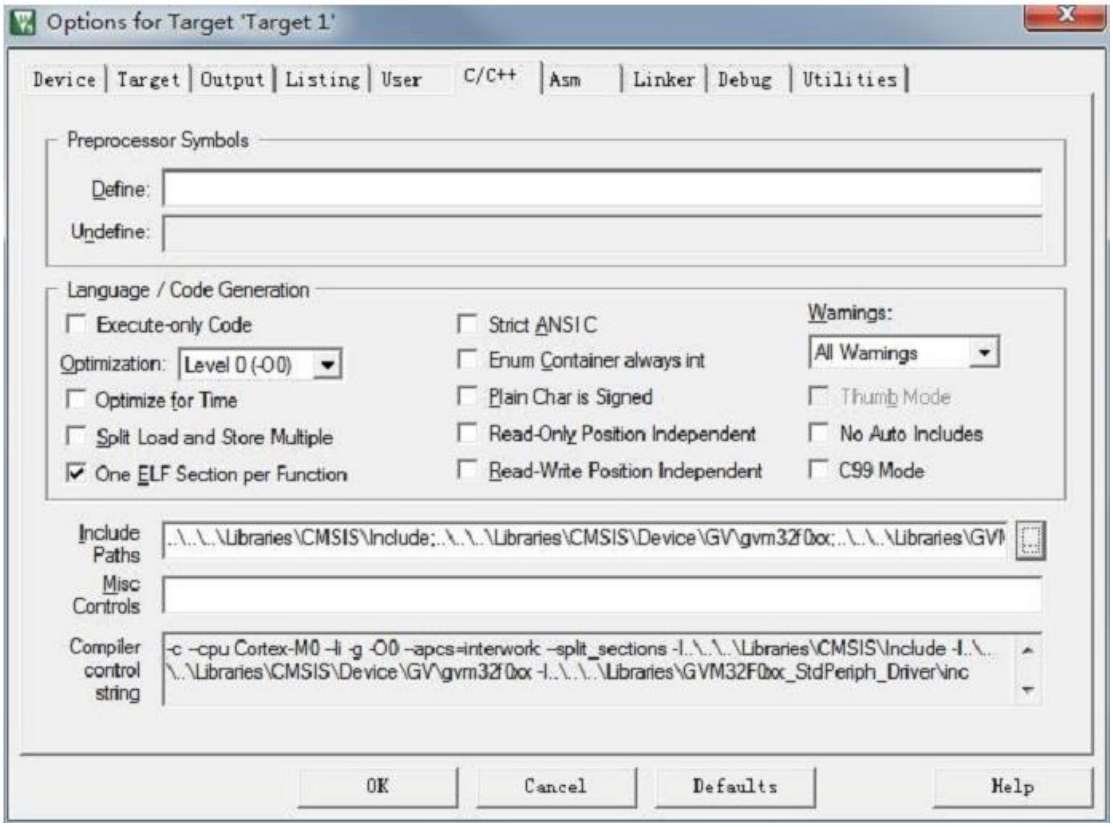


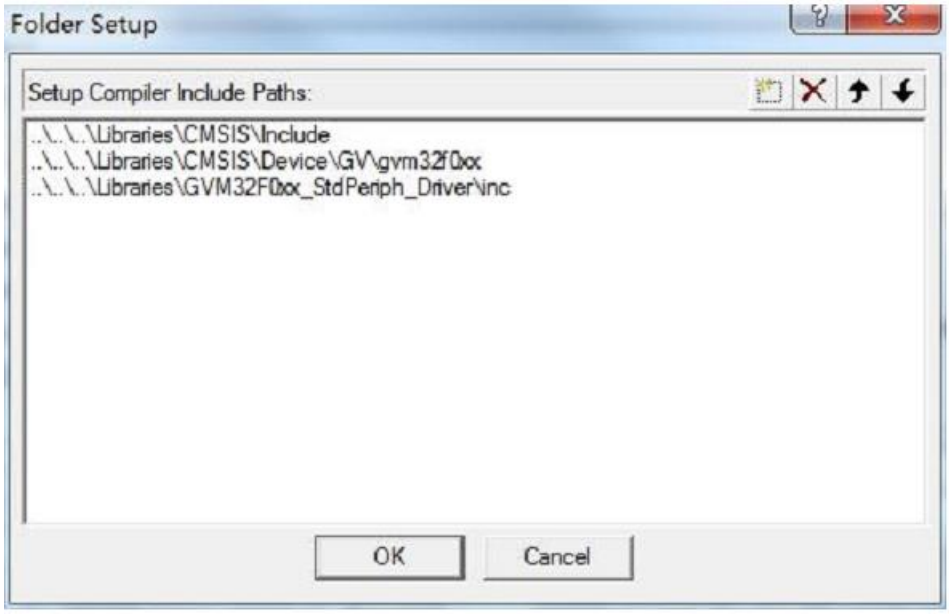
六、添加头文件包含路径。 需要添加的路径有：

GVM32F0xx_StdPeriph_Lib_V0.0.1\Libraries\CMSIS\Include

GVM32F0xx_StdPeriph_Lib_V0.0.1\Libraries\CMSIS\Device\GV\gvm32f0xx

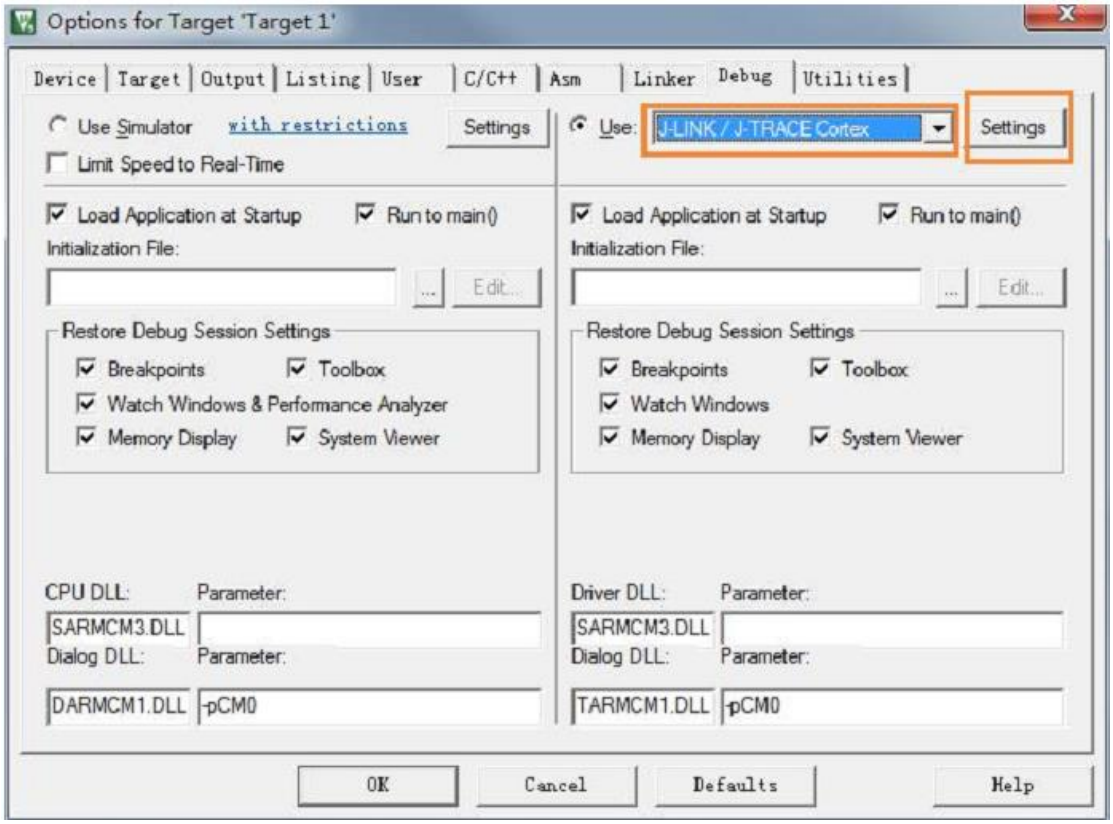
GVM32F0xx_StdPeriph_Lib_V0.0.1\Libraries\GVM32F0xx_StdPeriph_Driver\inc





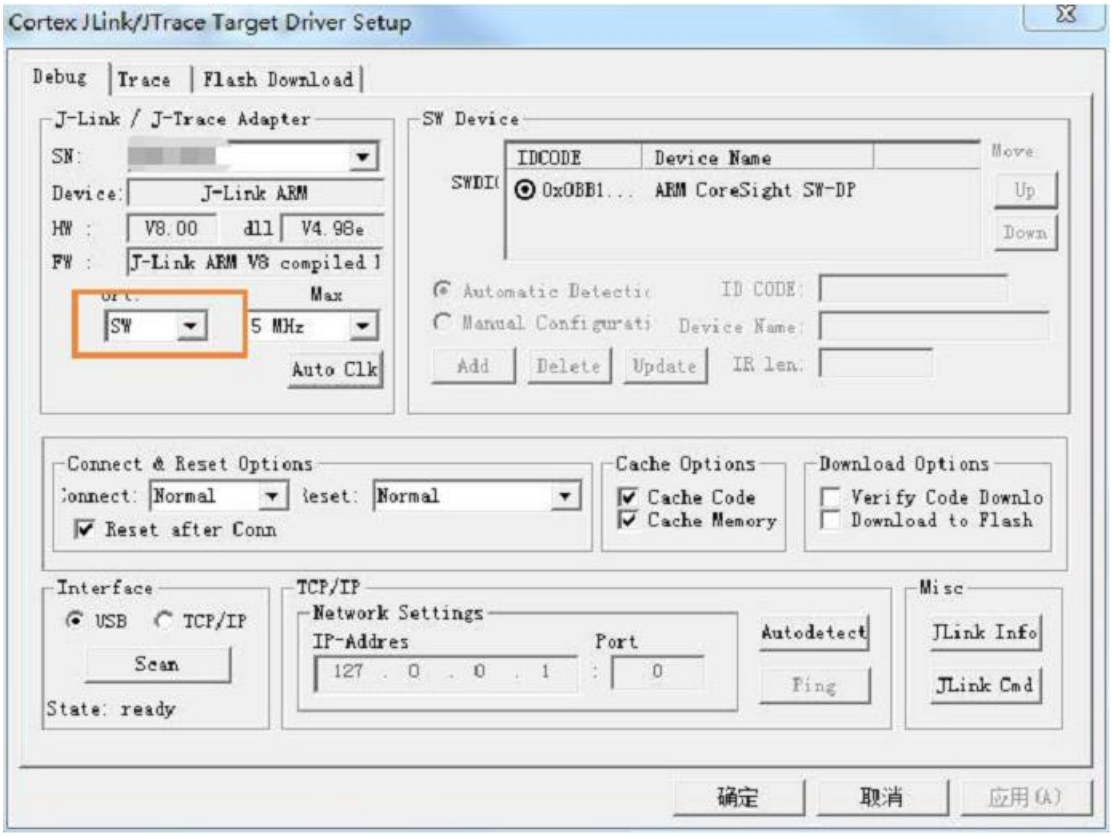
七、将 GVM32F030 的 Flash 烧录算法文件 ***GVM32F030.FLM*** 拷贝到 Keil 的 安装目录 ***C:\Program Files (x86)\Keil_v5\ARM\Flash*** 下。

八、配置调试工具， 这里以 Jlink 为例。



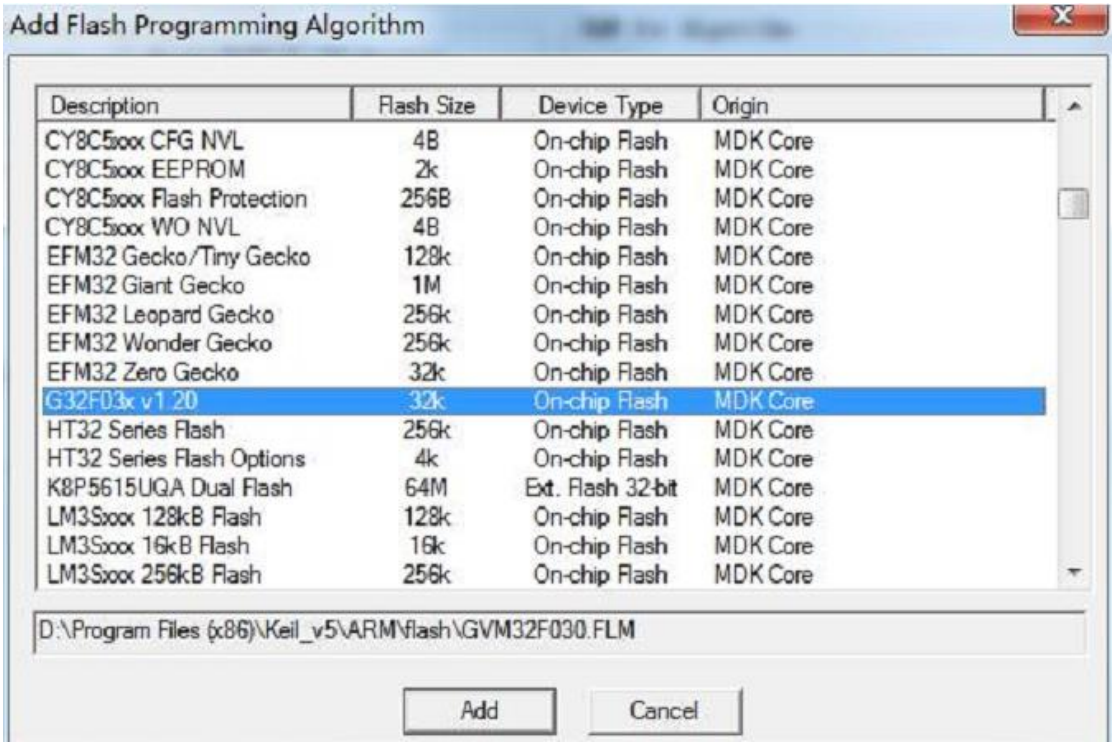
点击 Settings，若弹出对话框选择 No。

选择 ***SW*** 模式。



在 Flash Download 标签页中，将 ***New Device 256kB Flash*** 选中， Remove 掉。RAM for Algorithm 配置为 ***Start: 0x10000000 Size: 0x1000***。

点击 Add， 在列表中选中 ***G32F03x v1.20*** ； 点击 Add。



最后点击确定。

至此配置完成。