文档版本	V1.20
发布日期	20210825

APT32F102x LIB 用户手册

目录

1 前言

本手册主要介绍了如何使用 APT32F102x 系列芯片的通用 LIB 库文件。

2 CDK 使用介绍

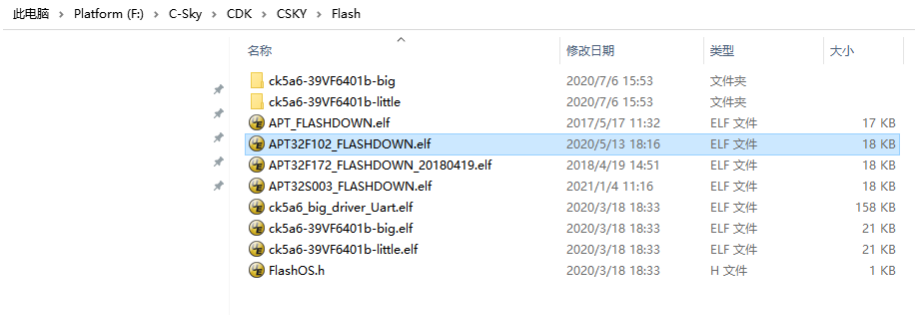
开发 APT32F102x 系列芯片时，使用者需要在电脑上安装 CDK 编译器，CDK 请使用至少 V2.8 或以上版本。在更新到 V2.8.3 后，CDK 版本可进行在线升级更新。若您的 APTLink 为旧版本驱动，在安装新版本 CDK 后，会在您连接 APTLink 和电脑时自动更新驱动，在更新期间请勿拔掉电源。您需要注意的是，更新 APTLink 驱动后旧版本 CDK 将无法与之连接。

2.1 CDK 配置

用户第一次使用 CDK 及 APT 库文件工程时，需在电脑上操作如下步骤:

2.1.1 添加算法文件

将 APT32F102_FLASHDOWN.elf 文件，复制添加到到 CSKY 安装目录: \C-SKY\CDK\CSKY\Flash 文件夹。

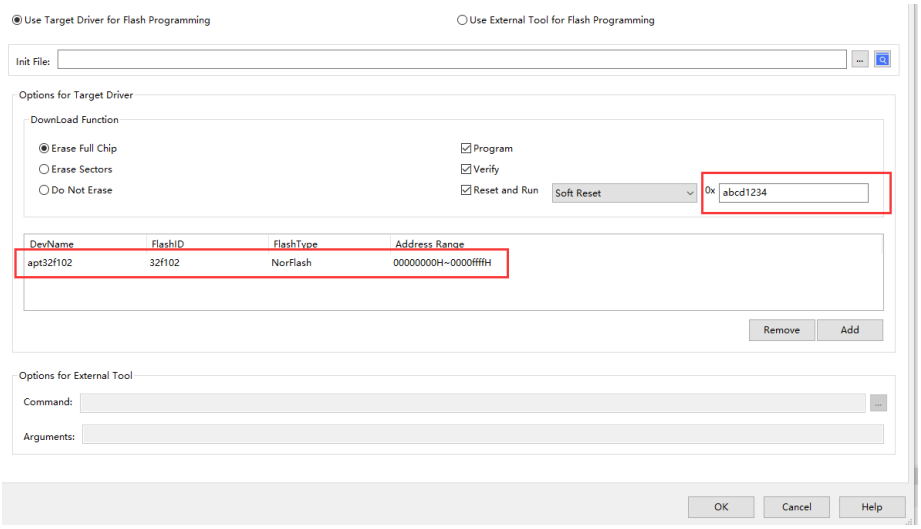


说明：在 Lib 库文件中的 Flashdown 文件夹包含有*.elf 文件，请务必使用此算法文件。若*.elf 不正确，可能会出现无法调试或在下载程序时校验错误的问题

2.1.2 在 CDK 中配置

在 CDK 中菜单栏选择

Flash->Configure Flash Tools->Flash->Add “APT32F102_FLASHDOWN.elf”文件，
同时在 Soft Reset 栏添加“abcd1234”,添加完成后 Projec settings 界面会显示以下信息：



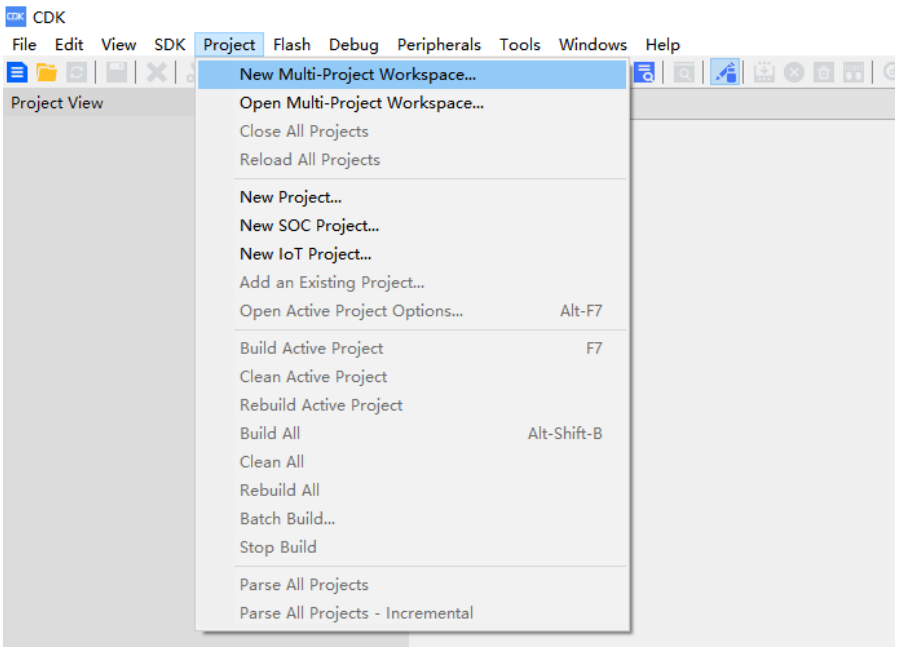
2.2 Workspace 工程项目

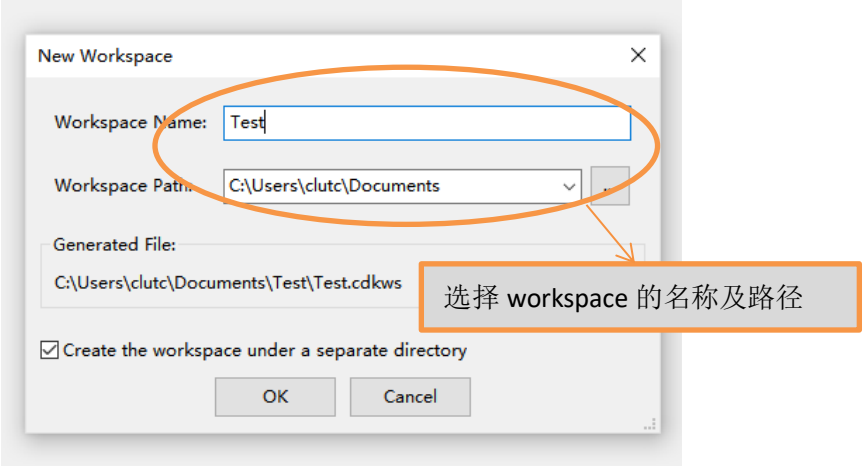
2.2.1 打开 workspace

直接双击库文件 workspace 文件夹的*.cdkws 开启工程，亦可通过 CDK 中的 Open Multi-Project Workspace 打开工程项目

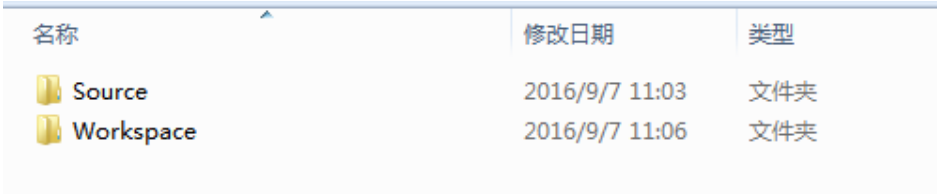
2.2.2 新建 workspace

使用新的工程项目时，请按以下步骤（新建工程文件夹名字不能是中文）
Project->New Multi-Project Workspace

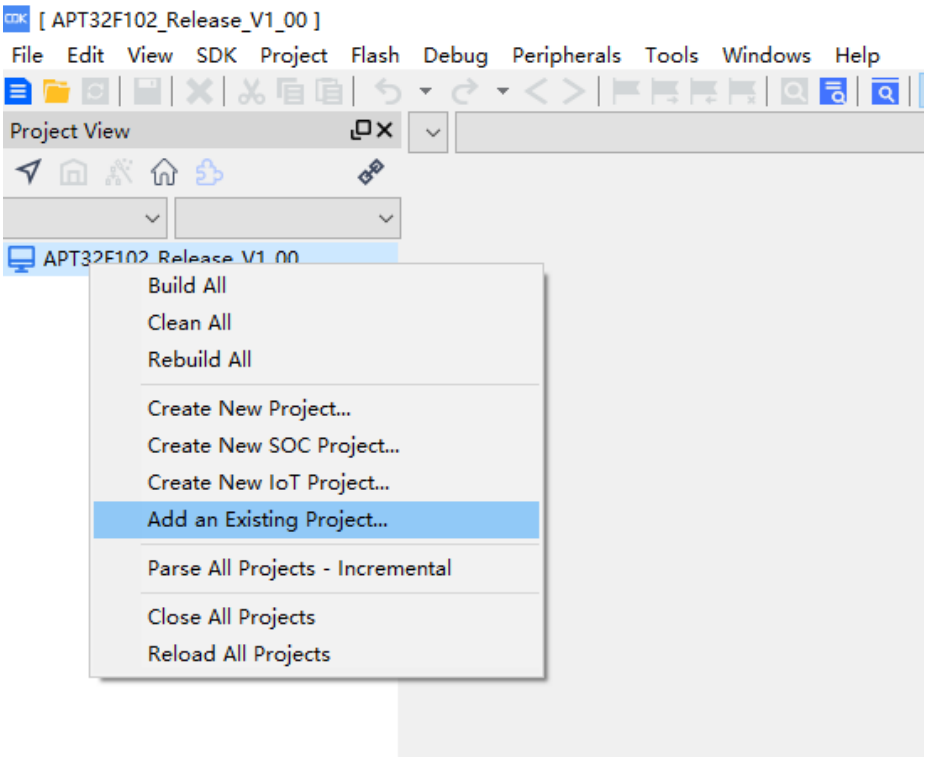


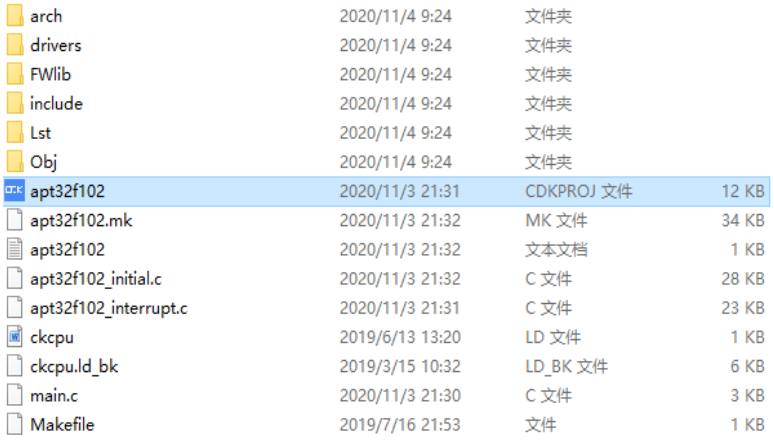


选择好路径后，将库文件中的 Source 文件夹拷贝到新的工程目录下



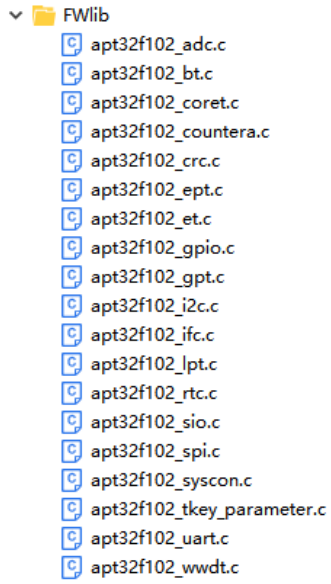
最后在 CDK 中将 Source 中的 apt32f102.cdkproj 添加进来即可。


[APT32F102_Release_V1_00]
File Edit View SDK Project Flash Debug Peripherals Tools Windows Help
Project View
APT32F102_Release_V1_00
Build All
Clean All
Rebuild All
Create New Project...
Create New SOC Project...
Create New IoT Project...
Add an Existing Project...
Parse All Projects - Incremental
Close All Projects
Reload All Projects

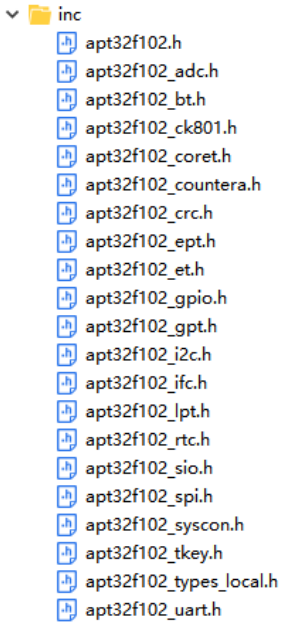

arch 2020/11/4 9:24 文件夹
drivers 2020/11/4 9:24 文件夹
FWlib 2020/11/4 9:24 文件夹
include 2020/11/4 9:24 文件夹
Lst 2020/11/4 9:24 文件夹
Obj 2020/11/4 9:24 文件夹
apt32f102 2020/11/3 21:31 CDKPROJ 文件 12 KB
apt32f102.mk 2020/11/3 21:32 MK 文件 34 KB
apt32f102 2020/11/3 21:32 文本文档 1 KB
apt32f102_initial.c 2020/11/3 21:32 C 文件 28 KB
apt32f102_interrupt.c 2020/11/3 21:31 C 文件 23 KB
ckcpu 2019/6/13 13:20 LD 文件 1 KB
ckcpu.ld_bk 2019/3/15 10:32 LD_BK 文件 6 KB
main.c 2020/11/3 21:30 C 文件 3 KB
Makefile 2019/7/16 21:53 文件 1 KB

2.3 库文件结构介绍

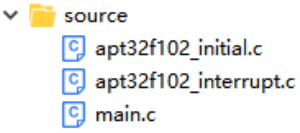
FWlib—库函数目录



inc—头文件目录



Source----初始化、中断、main 文件



若要对某个模块进行初始化，可在“apt32f102_intial.c”文档中对每个模块进行配置：

```
        void APT32F102_init(void)
{
//-------------------------------------------------------------/
//Peripheral clock enable and disable
//EntryParameter:NONE
//ReturnValue:NONE
//-------------------------------------------------------------/
        SYSCON->PCER0=0xFFFFFFF;                    //PCLK Enable
        SYSCON->PCER1=0xFFFFFFF;                    //PCLK Enable
        while(!(SYSCON->PCSR0&0x1));                //Wait PCLK enabled
//-------------------------------------------------------------/
//ISOSC/IMOSC/EMOSC/SYSCLK/IWDT/LVD/EM_CMFAIL/EM_CMRCV/CMD_ERR OSC stable interrupt
//EntryParameter:NONE
//ReturnValue:NONE
//-------------------------------------------------------------/
        SYSCON_CONFIG();                            //syscon   initial
        CK_CPU_EnAllNormalIrq();                    //enable all IRQ
//-------------------------------------------------------------/
//Other IP config
//-------------------------------------------------------------/
          //GPIO_CONFIG();                          //GPIO initial
          //EPT0_CONFIG();                          //EPT0 initial
          //GPT0_CONFIG();                          //GPT0 initial
          //BT_CONFIG();                            //BT initial
          //COUNTA_CONFIG();                        //CountA initial
          //RTC_CONFIG();                           //RCT initial
          //ET_CONFIG();                            //ETCB initial
         //I2C_MASTER_CONFIG();                     //I2C harware master initial
          //I2C_SLAVE_CONFIG();                     //I2C harware slave initial
          //SPI_MASTER_CONFIG();                    //SPI Master initial
          //SPI_SLAVE_CONFIG();                     //SPI Slaver initial
          //SIO_CONFIG();                           //SIO initial
         //UART0_CONFIG();                          //UART0 initial
          //UART1_CONFIG();                         //UART1 initial
          //UART2_CONFIG();                         //UART2 initial
          //ADC12_CONFIG();                         //ADC initial
          //TK_CONFIG();                            //Touch Key initial
          //SYSCON_INT_Priority();                  //interrupt priority initial
}
```

主函数编写在“main.c”文档中。具体位置如下：

```
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();     //clear wdt
        //...
    }
}
```

中断函数在“apt32f102_interrupt.c”文档中，每个函数都是以模块名+IntHandler 命名，eg:下面为外部中断 2~3 的中断函数

```
void EXI2to3IntHandler(void)
{
    // ISR content ...
    if ((SYSCON->EXIRS&EXI_PIN2)==EXI_PIN2)
    {
        SYSCON->EXICR = EXI_PIN2;
    }
    else if ((SYSCON->EXIRS&EXI_PIN3)==EXI_PIN3)
    {
        SYSCON->EXICR = EXI_PIN3;
    }
}
```

***注意：以上为完整的库文件工程，若使用时有不需要用到的模块，可以手动将程序文件移除出工程。移除文件时会有是否删除源文件的提示，请使用者注意。**

在 apt32f102_interrupt.c 中默认列出了所有的中断标志，使用时可以将未使用到的中断标志删除，以节省程序空间

请使用最新的库文件，若您使用旧版本库文件，本文中所描述的某些功能可能暂未包含

3 开发工具使用介绍

3.1 仿真连接

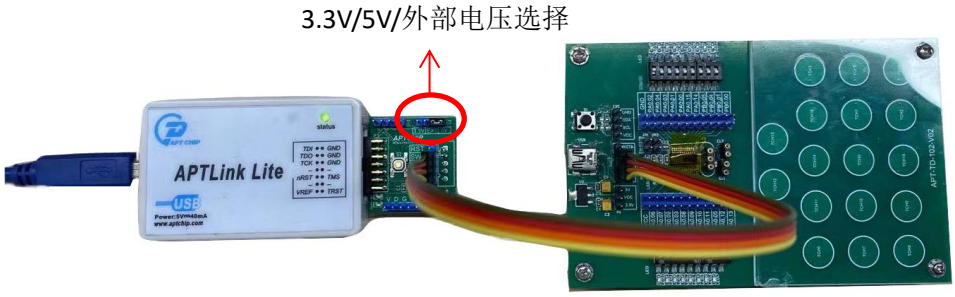


图 3-1 APTLink 连接目标板

APTLink 支持在线仿真和下载程序镜像使用。将 APTLink 与目标板的 VDD/SWDO/SCLK/GND 对应连接，如图（3-1）。前提条件是 SWCLK 和 SWDIO 没有用作其他功能。如果程序中将 SWCLK 和 SWDIO 设为其他功能，CDK 将无法与目标板连接，需要使用烧录器擦除芯片才能再次连接；建议您在调试阶段使用到 SWCLK 和 SWDIO 时，在芯片初始化前加入延时程序，在此延时时间内您仍可以通过 CDK 下载程序。若调试口已用做其他 IO 功能，则可使用带擦除功能的转接板(CT01 转接板)擦除 IC，或使用烧录器擦除芯片程序后才能正常调试。

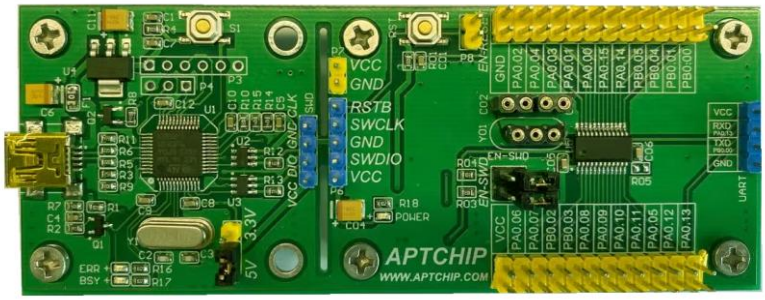


图 3-2 APT32F102 学习板

客户也可以使用 APT32F102 学习板，自带仿真器和目标板，如图 3-2

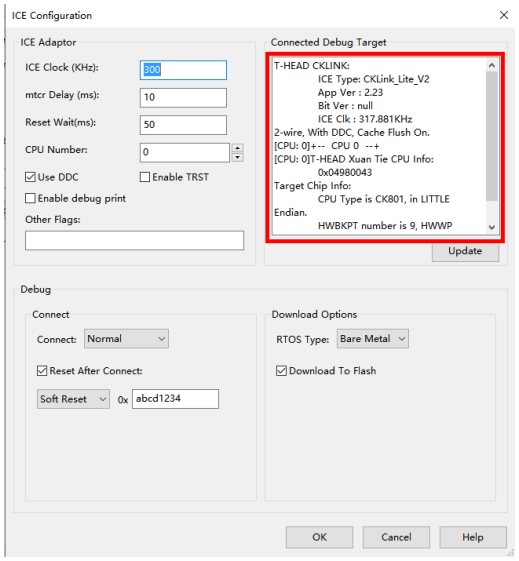


图 3-4 ICE 连接状态图

开发工具连接好后，可通过 Project Setting->Debug->ICE Settings 查看连接状态，如图 3-4

3.2 烧录连接

图 3-5 APT 烧录器 WD01

烧录脚位：VDD、SWCLK(PA0.6)、SWDIO(PA0.7)、RESET(PA0.2)、GND

注：烧录器 APT-WD01 操作步骤详细操作见“APTISP 使用手册简介”

4 APT32F102x 库文件使用介绍

本库文件包含 APT32F102X 系列完整的功能模块。系列芯片中 APT32F102/ APT32F1021/ APT32F1022/ APT32F1023 对应产品定义分类如下

资源		102	1021	1022	1023
异	HSIO	-	4	-	4
	TTL	-	-	-	√
	UART	2	2	2	3
	SIO	1	-	-	1
	SPI	1/-	1	1	1
	HWD	-	-	1	1
	TOUCH	-	17/15/13	-	17/15/13
	ADC	16/14/6	16/14	16	16
	PFLASH	64K/32K	32K	64K/32K	64K/32K
同	DFLASH	2K			
	SRAM	4K			
	CRC	1			
	IWDT	1			
	WWDT	1			
	BT	2			
	CNTA	1			
	RTC	1			
	LPT	1			
	EPT	1			
	GPT	1			
	IIC	1			

4.1 SYSCON 系统模块

4.1.1SYSCON 时钟配置

SYSCON_General_CMD(ENABLE,ENDIS_ISOSC);

系统时钟使能/禁止函数

SYSCON_HFOSC_SELECTE(HFOSC_SELECTE_48M);

HFOSC 选择频率函数

SYSCON_IMOSC_SELECTE(IMOSC_SELECTE_5556K);

IMOCS 选择频率函数

EMOSC_OSTR_Config(0XAD,0X1f,EM_LFSEL_DIS,EM_FLEN_EN,EM_FLSEL_10ns);

外部时钟配置函数

0XAD 参数:此位参数为外部晶振时钟稳定计数器

0x1f 参数:外部晶振增益控制。频率越高，此值应越大

EM_LFSEL_DIS 参数：低速模式使能/禁止位

EM_FLEN_EN 参数：外部滤波使能/禁止位

EM_FLSEL_10ns 参数：外部振荡器滤波范围

SystemCLK_HCLKDIV_PCLKDIV_Config(SYSCLK_HFOSC,HCLK_DIV_1,PCLK_DIV_1,HFOSC_48M);

系统时钟源选择，系统分频选择

4.1.2 内部主频 HFOSC 做系统时钟

- **功能实例:**

  开启内部主频 HFOSC,并作为系统时钟。HFOSC 有 48M/24M/12M/6M 可选, HCLK 分频设置为 1，PCLK 分频设置为 1。

  WDT 使能，溢出时间 1s。

- **操作步骤:**

  1. 使能所有 IP 模块
  2. Syscon 函数配置
  3. 打开全局中断向量

- **程序范例:**

```
void SYSCON_CONFIG(void)
    {
        //------SYSTEM CLK AND PCLK FUNTION--------------------------/
        SYSCON_RST_VALUE();                                    //SYSCON all register clr
        SYSCON_General_CMD(ENABLE,ENDIS_ISOSC);
        SYSCON_HFOSC_SELECTE(HFOSC_SELECTE_48M);        //HFOSC selected 48MHz
        SystemCLK_HCLKDIV_PCLKDIV_Config(SYSCLK_HFOSC,HCLK_DIV_1,PCLK_DIV_1,HFOSC_48M);
        //system clock set, Hclk div ,Pclk div   set system clock=SystemCLK/Hclk div/Pclk div
        //------------   WDT FUNTION   -------------------------------/
        SYSCON_IWDCNT_Config(IWDT_TIME_1S,IWDT_INTW_DIV_7); //WDT TIME 1s,WDT alarm interrupt
```

```
    time=1s-1s*1/8=0.875S
        SYSCON_WDT_CMD(ENABLE);                                  //enable WDT
        SYSCON_IWDCNT_Reload();                                  //reload WDT
        //------------   LVD FUNTION   --------------------------------/
        SYSCON_LVD_Config(DISABLE_LVDEN,INTDET_LVL_3_3V,RSTDET_LVL_1_9V,DISABLE_LVD_INT,I
    NTDET_POL_fall);
    }
    /*****************************************************************************/
    //APT32F102_init
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*****************************************************************************/
    void APT32F102_init(void)
    {
        SYSCON->PCER0=0xFFFFFFF;                                 //PCLK Enable
        SYSCON->PCER1=0xFFFFFFF;                                 //PCLK Enable
        while(!(SYSCON->PCSR0&0x1));                             //Wait PCLK enabled

        SYSCON_CONFIG();                                         //syscon 参数   初始化
        CK_CPU_EnAllNormalIrq();                                 //enable all IRQ
    }
```

4.1.3 内部主频 IMOSC 做系统时钟

- **功能实例:**

  开启内部主频 IMOSC,并作为系统时钟。IMOSC 有 5.556M/4.194M/2.097M/131K 可选 HCLK 分频设置为 1，PCLK 分频设置为 1。

- **操作步骤:**

  1. 使能所有 IP 模块
  2. Syscon 函数配置
  3. 打开全局中断向量

```
void SYSCON_CONFIG(void)
    {
        //------SYSTEM CLK AND PCLK FUNTION--------------------------/
        SYSCON_RST_VALUE();                                      //SYSCON all register clr
        SYSCON_General_CMD(ENABLE,ENDIS_ISOSC);
        SYSCON_HFOSC_SELECTE(HFOSC_SELECTE_12M);          //HFOSC selected 48MHz
        SystemCLK_HCLKDIV_PCLKDIV_Config(SYSCLK_HFOSC,HCLK_DIV_1,PCLK_DIV_1,HFOSC_12M);
        //system clock set, Hclk div ,Pclk div    set system clock=SystemCLK/Hclk div/Pclk div
        SYSCON_IMOSC_SELECTE(IMOSC_SELECTE_5556K);
        SystemCLK_HCLKDIV_PCLKDIV_Config(SYSCLK_IMOSC,HCLK_DIV_1,PCLK_DIV_1,IMOSC);
        SYSCON_General_CMD(DISABLE,ENDIS_HFOSC);
        //------------   WDT FUNTION   ------------------------------/
```

```
    SYSCON_IWDCNT_Config(IWDT_TIME_1S,IWDT_INTW_DIV_7);  //WDT TIME 1s,WDT alarm interrupt time=1s-1s*1/8=0.875S
    SYSCON_WDT_CMD(ENABLE);                              //enable WDT
    SYSCON_IWDCNT_Reload();                              //reload WDT
    //------------ LVD FUNTION ---------------------------------/
    SYSCON_LVD_Config(DISABLE_LVDEN,INTDET_LVL_3_3V,RSTDET_LVL_1_9V,DISABLE_LVD_INT,INTDET_POL_fall);
}
/*******************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*******************************************************************************/
void APT32F102_init(void)
{
    SYSCON->PCER0=0xFFFFFFF;                             //PCLK Enable
    SYSCON->PCER1=0xFFFFFFF;                             //PCLK Enable
    while(!(SYSCON->PCSR0&0x1));                         //Wait PCLK enabled

    SYSCON_CONFIG();                                     //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                             //enable all IRQ
}
```

4.1.4 内部主频 EMOSC 做系统时钟

- **功能实例:**

  开启外部晶振 EMOSC,并作为系统时钟

  WDT 使能，溢出时间 1s

  HCLK 分频设置为 1，PCLK 分频设置为 1

- **操作步骤:**

  1. 使能所有 IP 模块
  2. Syscon 函数配置
  3. 打开全局中断向量

```
void SYSCON_CONFIG(void)
{
    //------SYSTEM CLK AND PCLK FUNTION--------------------------/
    SYSCON_RST_VALUE();                                  //SYSCON all register clr
    SYSCON_General_CMD(ENABLE,ENDIS_ISOSC);
    EMOSC_OSTR_Config(0XAD,0X1f, EM_LFSEL_DIS,EM_FLEN_EN,EM_FLSEL_10ns);
    //0xad,LFSEL Disable, EM filter Enable,10ns filter
    SYSCON_General_CMD(ENABLE,ENDIS_EMOSC);
    SystemCLK_HCLKDIV_PCLKDIV_Config(SYSCLK_EMOSC,HCLK_DIV_1,PCLK_DIV_1,EMOSC_24M);
    //------------ WDT FUNTION ---------------------------------/
```

```
    SYSCON_IWDCNT_Config(IWDT_TIME_1S,IWDT_INTW_DIV_7);  //WDT TIME 1s,WDT alarm interrupt time=1s-1s*1/8=0.875S
    SYSCON_WDT_CMD(ENABLE);                              //enable WDT
    SYSCON_IWDCNT_Reload();                              //reload WDT
    //------------ LVD FUNTION ---------------------------------/
    SYSCON_LVD_Config(DISABLE_LVDEN,INTDET_LVL_3_3V,RSTDET_LVL_1_9V,DISABLE_LVD_INT,INTDET_POL_fall);
}
/*******************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*******************************************************************************/
void APT32F102_init(void)
{
    SYSCON->PCER0=0xFFFFFFF;                             //PCLK Enable
    SYSCON->PCER1=0xFFFFFFF;                             //PCLK Enable
    while(!(SYSCON->PCSR0&0x1));                         //Wait PCLK enabled

    SYSCON_CONFIG();                                     //syscon 参数  初始化
    CK_CPU_EnAllNormalIrq();                             //enable all IRQ
}
```

4.1.5 系统时钟切换

系统时钟可以根据不用应用要求，支持在多个时钟源间进行切换。系统上电时，缺省选择 IMOSC 的 5.556MHz 作为工作时钟。在切换时钟源时，建议先执行 SystemCLK_Clear(); 操作将时钟源寄存器恢复为默认以保证时钟源切换稳定

- **时钟源：**

IMOSC（Internal Main OSC)	5.556MHz/4.194MHz/2.097MHz/131.072KHz
HFOSC（High Frequency OSC)	48MHz/24MHZ/12MHZ/6MHZ
EMOSC（External Main OSC)	32.768KHz / 36KHz ~ 24MHz
ISOSC（Internal Sub OSC)	27KHz

例：系统完成上电复位和硬件初始化后,使能 EMOSC 时钟，再切换到内部 HFOSC，并把 HFOSC 作为系统时钟

```
    /************************************************************/
    //syscon Functions
    //EntryParameter:NONE
    //ReturnValue:NONE
    /************************************************************/
    void SYSCON_CONFIG(void)
    {
    //------SYSTEM CLK AND PCLK FUNTION--------------------------/
     SYSCON_RST_VALUE();                                          //SYSCON all register clr
     SYSCON_General_CMD(ENABLE,ENDIS_ISOSC);                      //SYSCON enable/disable clock source

     SYSCON_General_CMD(ENABLE,ENDIS_EMOSC);
     EMOSC_OSTR_Config(0XAD,0X1f,EM_LFSEL_DIS,EM_FLEN_EN,EM_FLSEL_10ns);
    //system clock set, Hclk div ,Pclk div   set system clock=SystemCLK/Hclk div/Pclk div
     SystemCLK_HCLKDIV_PCLKDIV_Config(SYSCLK_EMOSC,HCLK_DIV_1,PCLK_DIV_1,EMOSC_24M);
    //
    SystemCLK_Clear();
    SYSCON_HFOSC_SELECTE(HFOSC_SELECTE_48M);
    SystemCLK_HCLKDIV_PCLKDIV_Config(SYSCLK_HFOSC,HCLK_DIV_1,PCLK_DIV_1,HFOSC_48M);
}
```

4.1.6 LVD/LVR 配置

```
    SYSCON_LVD_Config(ENABLE_LVDEN,INTDET_LVL_3_3V,RSTDET_LVL_1_9V,DISABLE_LVD_INT,INTDET_POL_fall);    //LVD LVR Enable
      LVD_Int_Enable();
```

ENABLE_LVDEN　**LVD 使能** /　DISABLE_LVDEN **LVD 禁止**
ENABLE_LVD_INT　**LVD 中断使能** /　DISABLE_LVD_INT **LVD 中断禁止**
INTDET_POL_fall　LVD 低电压检测中断触发极性为下降沿
INTDET_POL_X_rise　LVD 低电压检测中断触发极性为上升沿沿
INTDET_POL_X_riseORfall　LVD 低电压检测中断触发极性为下降沿和上升沿

LVD 电压等级为 2.1/2.4/2.7/3.0/3.3/3.6/3.9V
LVR 电压等级为 1.9/2.2/2.5/2.8/3.1/3.4/3.7/4.0V

LVD 的中断在 SYSCONIntHandler 中的 LVD_INT_ST 位置

4.1.7 IWDT 配置

```
SYSCON_IWDCNT_Config(IWDT_TIME_1S,IWDT_INTW_DIV_7);
//WDT TIME 1s,WDT alarm interrupt time=1s*7/8=0.875S
SYSCON_WDT_CMD(ENABLE);                                    //enable WDT
SYSCON_IWDCNT_Reload();                                    //reload WDT
//IWDT_Int_Enable();
```

SYSCON_WDT_CMD(ENABLE);

IWDT 使能禁止函数

SYSCON_IWDCNT_Reload();

IWDT 喂狗函数

IWDT_Int_Enable();

IWDT 中断使能

如上例中，WDT 设置为 1s，分频为 7，则报警中断时间为 1s*7/8=0.875S。用户需要在 IWDT 设置时间内进行清狗操作。若达到报警时间未清狗，在 IWDT 中断开启时会产生 IWDT 报警中断，报警中断产生后程序将进入中断程序 SYSCONIntHandler 的 IWDT_INT_ST 位置，报警中断后需要进行喂狗操作，否则会产生 IWDT 复位。

4.1.8 系统睡眠及唤醒

在缺省上电复位后，系统工作于运行模式（RUN MODE）。在某些特殊应用下，CPU 不需要再继续工作，出于节省功耗考虑，用户可以选择将系统切换到低功耗模式。在需要 CPU 再次处理任务时，通过预先设置的触发条件对系统进行唤醒。

- 睡眠模式 （SLEEP MODE）：CPU 时钟被关闭，所有外设的时钟可以通过进行预先设置为关闭或者使能，当有任何外设中断发生时，都可以唤醒 CPU，并退出 SLEEP 模式。

  PCLK_goto_idle_mode();　　//进入睡眠函数

- 深睡眠模式（DEEP SLEEP MODE）：CPU 时钟被关闭，所有外设时钟被关闭。由于某些外设可以独立于 PCLK 工作（例如 RTC，LPT 和 TOUCH），可以通过配置选择时钟源是否关闭。可以由深度睡眠唤醒的源有 LVD，EXI，WDT，RTC，TOUCH，LPT

  PCLK_goto_deepsleep_mode();　　//深度睡眠函数

4.2 中断优先级配置

APT32F102 的中断优先级共 4 级，设定数值越小，代表的优先级越高，所以设置为‘0’时代表最高优先级。如果优先级号相同，则根据中断源号来决定优先的顺序，号码越小，优先级越高。 例如， IRQ0 和 IRQ1 的优先级号设置为相同，当 IRQ0 和 IRQ1 同时提交中断，由于 IRQ0 的中断源号小于 IRQ1，因此 IRQ0 先得到 CPU 的响应。

所有中断分 8 组，分别是 IQR0-3、IQR4-7、IQR8-11、IQR12-15、IQR16-19、IQR20-23、IQR24-27、IQR28-31。每一个寄存器对应 4 个中断向量高两位赋值，分别有 00/40/80/C0 四种。以下是中断向量表

Coret_INT	IRQ0	UART2_INT	IRQ15
SYSCON_INT	IRQ1	I2C_INT	IRQ17
IFC_INT	IRQ2	SPI_INT	IRQ19
ADC_INT	IRQ3	SIO_INT	IRQ20
EPT0_INT	IRQ4	EXI2_INT	IRQ21
WWDT_INT	IRQ6	EXI3_INT	IRQ22
EXI0_INT	IRQ7	EXI4_INT	IRQ23
EXI1_INT	IRQ8	CA_INT	IRQ24
GPT0_INT	IRQ9	TKEY_INT	IRQ25
RTC_INT	IRQ12	LPT_INT	IRQ26
UART0_INT	IRQ13	BT0_INT	IRQ28
UART1_INT	IRQ14	BT1_INT	IRQ29

在 syscon.c 中 void SYSCON_INT_Priority(void)函数可以统一配置中断优先级，使用是需要按需求设定对应的优先级配置，再调用次函数即可。

以下是将 UART0，UART1 设为中断优先级 0，其余中断设为优先级 1 的配置

```
void SYSCON_INT_Priority(void)
{
    INTC_IPR0_WRITE(0X40404040);    //IQR0-3
    INTC_IPR1_WRITE(0X40404040);    //IQR4-7
    INTC_IPR2_WRITE(0X40404040);    //IQR8-11
    INTC_IPR3_WRITE(0X40000040);    //IQR12-15
    INTC_IPR4_WRITE(0X40404040);    //IQR16-19
    INTC_IPR5_WRITE(0X40404040);    //IQR20-23
    INTC_IPR6_WRITE(0X40404040);    //IQR24-27
    INTC_IPR7_WRITE(0X40404040);    //IQR28-31
}
```

4.3 GPIO 模块

4.3.1 GPIO 配置

- GPIO 输入与输出状态

```
GPIO_Init(GPIOA0,0,0);          //表示将 GPIOA0.0 设为输出
GPIO_Init(GPIOA0,0,1);          //表示将 GPIOA0.0 设为输入
GPIO_Init(GPIOB0,0,0);          //表示将 GPIOB0.0 设为输出
GPIO_Init(GPIOB0,2,0);          //表示将 GPIOB0.2 设为输出
```

- GPIO 输入输出禁止

```
GPIO_InPutOutPut_Disable(GPIOA0,0);    //表示将 GPIOA0.0 输入输出禁止(高阻态)
GPIO_InPutOutPut_Disable(GPIOA0,1);    //表示将 GPIOA0.1 输入输出禁止(高阻态)
```

- GPIO 输出与输出控制

```
GPIO_Set_Value(GPIOA0,0,1);       //表示 GPIOA0.0 输出高电平
GPIO_Set_Value(GPIOA0,0,0);       //表示 GPIOA0.0 输出低电平
GPIO_Write_High(GPIOA0,0);        //表示 GPIOA0.0 输出高电平
GPIO_Write_Low(GPIOA0,0);         //表示 GPIOA0.0 输出低电平
```

- GPIO 输出翻转

```
GPIO_Reverse(GPIOA0,0);         //表示 GPIOA0.0 输出翻转
```

- GPIO 模式设置

```
GPIO_PullHigh_Init(GPIOA0,1);              //配置 PA0.1 上拉
GPIO_PullLow_Init (GPIOA0,1);              //配置 PA0.1 下拉
GPIO_PullHighLow_DIS(GPIOA0,1);       //配置 PA0.1 上下拉关闭
GPIO_OpenDrain_EN(GPIOA0,1);          //配置 PA0.1 为漏极开路
GPIO_OpenDrain_DIS(GPIOA0,1);         //配置 PA0.1 漏极开路关闭
*GPIO_DriveStrength_EN(GPIOA0,8);     //配置 PA0.8 为大电流输出脚
*GPIO_DriveStrength_DIS (GPIOA0,8);   //配置 PA0.8 大电流输出关闭
*GPIO_TTL_COSM_Selecte(GPIOA0,0,INPUT_MODE_SETECTED_TTL1);    //配置 PA0.0 为 TTL1 模式
```

*： 大电流输出功能仅 APT32F1021 及 APT32F1023 系列支持
且仅 PA0.8/9,PB0.2/3 四个 IO 支持
TTL 模式仅 APT32F1023 系列支持，芯片默认为 CMOS 模式

- 读取 IO 状态设置

```
GPIO_Read_Status (GPIOA0,11);          //读取 PA0.11 的状态
```

4.3.2 GPIO 输入输出及外部中断唤醒配置

- **功能实例:**

  PA0.0 作为外部中断上升沿唤醒口，使能 PA0.0 内部上拉。

  PA0.12 默认输出高,唤醒后 PA0.12 输出低。

  PA0.1 漏极输出，PB0.0 强驱动使能

- **操作步骤:**

  1. SYSCON_CONFIG();函数配置
  2. GPIO_CONFIG();函数配置
  3. 主循环代码

- **程序范例:**

```
/************************************************************/
//GPIO Functions
//EntryParameter:NONE
//ReturnValue:NONE
/************************************************************/
void GPIO_CONFIG(void)
{
    GPIO_Init(GPIOA0,12,0);                 //PA0.12  输出模式
    GPIO_Set_Value(GPIOA0,12,1);            //PA0.12  输出高
    GPIO_DriveStrength_EN(GPIOB0,2);        //PB0.2 强驱动使能
    GPIO_OpenDrain_EN(GPIOA0,1);            //漏极开路输出使能
    GPIO_PullHigh_Init(GPIOA0,0);
    //------------   EXI FUNTION   --------------------------------/
    //EXI0_INT= EXI0/EXI16,EXI1_INT= EXI1/EXI17, EXI2_INT=EXI2~EXI3/EXI18/EXI19,
//EXI3_INT=EXI4~EXI9, EXI4_INT=EXI10~EXI15
    GPIO_IntGroup_Set(PA0,0,Selete_EXI_PIN0);       //EXI0 set PBA.0
    GPIOA0_EXI_Init(EXI0);                          //PA0.0 as input
    EXTI_trigger_CMD(ENABLE,EXI_PIN0,_EXIFT);       //ENABLE falling edge
    EXTI_interrupt_CMD(ENABLE,EXI_PIN0);            //enable EXI
    GPIO_EXTI_interrupt(GPIOA0,0b00000000000001);   //enable GPIOA00 as EXI
    EXI0_Int_Enable();                              //EXI0 INT Vector

    EXI0_WakeUp_Enable();                           //EXI0 interrupt wake up enable
}
/************************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/************************************************************************************/
void APT32F102_init(void)
{
```

```
        SYSCON->PCER0=0xFFFFFFFF;                    //使能 IP
        SYSCON->PCER1=0xFFFFFFFF;                    //使能 IP
        while(!(SYSCON->PCSR0&0x1));                 //判断 IP 是否使能
        SYSCON_CONFIG();                             //syscon 参数 初始化
        CK_CPU_EnAllNormalIrq();                     //打开全局中断
        GPIO_CONFIG ()；
}
/*************************************************************/
//main
/*************************************************************/
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                      //清狗
        GPIO_Write_High(GPIOA0,12);                  //GPIOA0.12 输出高
        PCLK_goto_deepsleep_mode();                  //进入深度睡眠
        GPIO_Write_Low(GPIOA0,12);                   //GPIOA0.12 输出低
    }
}
```

中断注意事项：

- 在开启多个 GPIO 同时中断时，需要在 *GPIO_EXTI_interrupt();*函数中同时使能，如同时将 GPIOA0.0 和 GPIOA0.1 做中断时，在分别配置完中断状态后，需要同时在此函数中使能这两个 IO:*GPIO_EXTI_interrupt(GPIOA0,0b00000000000011);*

4.3.3 GPIO 中断扩展配置

在 APT32F102x 的 GPIO 中断配置中，EXI GROUP0~15 是以管脚名的后缀分组的。例如，GROUP 0 只可能是 PA0.0 ,PB0.0 中的一个。除此以外，也提供了 EXI GROUP16~19 依据管脚的前缀来分组，例如，GROUP16,17 可以是 PA0.0~7 中任意一个，而 GROUP18,19 则可以是 PB0.x 中任意一个。

因此，使用者可以使用中断扩展功能，将管脚后缀相同的 IO 扩展到 GROUP16~19 组中，以实现相同管脚后缀的 IO 同时具有中断功能。

扩展后，中断入口对应列表：

GROUP	Int Handler
EXI16	EXI0IntHandler
EXI17	EXI1IntHandler
EXI18	EXI2to3IntHandler
EXI19	EXI2to3IntHandler

如下，是同时将 PA0.0 和 PB0.0 同时配置为外部下降沿中断的实例。

```
GPIO_IntGroup_Set(PA0,0,Selete_EXI_PIN0);                    //EXI0 set PA0.0
GPIOA0_EXI_Init(EXI0);                                       //PA0.0 as input
EXTI_trigger_CMD(ENABLE,EXI_PIN0,_EXIFT);                    //ENABLE falling edge
EXTI_interrupt_CMD(ENABLE,EXI_PIN0);                         //enable EXI
GPIO_EXTI_interrupt(GPIOA0,0b0000000000000001);              //enable GPIOA00 as EXI
GPIO_IntGroup_Set(PB0,0,Selete_EXI_PIN18);                   //EXI18 set PB0.
GPIOB0->CONLR = (GPIOB0->CONLR&0XFFFFFFF0) | 0X00000001;     //PB0.0 as input
EXTI_trigger_CMD(ENABLE,EXI_PIN18,_EXIFT);                   //ENABLE falling edge
EXTI_interrupt_CMD(ENABLE,EXI_PIN18);                        //enable EXI
GPIO_EXTI_interrupt(GPIOB0,0b0000000000000001);              //enable GPIOA00 as EXI

EXI0_Int_Enable();                                           //EXI0 INT Vector
EXI2_Int_Enable();                                           //EXI2~EXI3 INT Vector
```

4.3.4 GPIO Re-Map 功能配置

为提供更灵活的 IO 功能配置，系统提供了自定义 GPIO 复用的功能。芯片提供两个预设的 GPIO GROUP，分别为 GROUP0 和 GROUP1，两个 GROUP 分别对应 8 个预设的可选择的复用功能。在每个 GROUP 内，每个 GPIO 可以被指定为这 8 个预设功能中的任意一个作为该 GPIO 的 AF7 功能。在使用时，调用 GPIO_Remap 函数，确定需要配置的 IO 及 Re-map 的功能，如下是将 GPIOA0.0 重定义为 GPT CHA 的配置。

```
GPIO_Remap(GPIOA0,0,PIN_GPT_CHA);    //将 GPIOA0.0 定义为 GPTCHA
```

配置的对应表如下：

IOGROUP0:

GPIO	Re-map Function
PA0.0	I2C_SCL
PA0.1	I2C_SDA
PA0.2	GPT_CHA
PA0.3	GPT_CHB
PA0.4	SPI_MOSI
PA0.5	SPI_MISO
PA0.6	SPI_SCK
PA0.7	SPI_NSS

IOGROUP1:

GPIO	Re-map Function
PB0.2	UART0_RX
PB0.3	UART0_TX
PA0.8	EPT_CHAX
PA0.9	EPT_CHBX
PA0.10	EPT_CHCX
PA0.11	EPT_CHAY
PA0.12	EPT_CHBY
PA0.13	EPT_CHCY

4.4 CORET 系统定时器模块

4.4.1 CORET 说明

系统定时器提供了一个简单易用的 24 位循环递减的计数器，当系统计时器使能时，计数器开始工作。当计数器递减到 0 时，会向中断控制器发起中断请求。定时器的计算公式为：*Time=(CORET_RVR+1)*(8/48M)*

4.4.2 CORET 定时功能

- **功能实例:**

  开启内部主频 48MHz,并作为系统时钟。

  利用 Coret 定时产生中断，PA0.12 输出占空比为 50%,周期为 20ms 方波。

- **操作步骤:**

  1.SYSCON_CONFIG();函数配置

  2.GPIO_CONFIG();函数配置

  3.CORET_CONFIG();函数配置

  4.PA0.12 输出翻转写在“CORETHandler();”中断函数中

- **程序范例:**

```
volatile U8_T f_io_toggle;
/*************************************************************/
//CORET Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void CORET_CONFIG(void)
{
    CORET_DeInit();                                  //Coret 所有寄存器复位赋值
    SYSCON_General_CMD(ENABLE,ENDIS_SYSTICK);  //使能 STCLK 时钟
    CK801->CORET_RVR=59999;                          //coret 计数值
    //CORETCLK=sysclock/8=48M/8=6Mhz
    //e.g:10ms=(CORET_RVR+1)*(8/48M);CORET_RVR=60000-1=59999
    CORET_reload();                                  // Coret CVR 清除
    CORET_CLKSOURCE_EX();                            //使用时钟源为sysclk/8
    CORET_TICKINT_Enable();                          //使能计数器清零中断
    CORET_start();                                   //Coret 计时开始
    CORET_Int_Enable();                              //使能计数器清零中断向量
}
/*************************************************************/
//GPIO Functions
//EntryParameter:NONE
//ReturnValue:NONE
```

```
/*************************************************************/
void GPIO_CONFIG(void)
{
    GPIO_Init(GPIOA0,12,0);                 //PA0.12 输出模式

    GPIO_Set_Value(GPIOA0,12,1);            //PA0.12 输出高
}
/*****************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*****************************************************************************/
void APT32F102_init(void)
{

    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                    //打开全局中断

    CORET_CONFIG();
    GPIO_CONFIG ()；
}
/*************************************************************/
//CORET Interrupt
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void CORETHandler(void)
{
    CK801->CORET_CVR = 0;               // Coret CVR 清除
    if(!f_GPIO_Toggle)
    {
        GPIO_Write_Low(GPIOA0,12);      //PA0.12 输出低
        f_GPIO_Toggle=1;
    }
    else
    {
        GPIO_Write_High(GPIOA0,12);     //PA0.12 输出高
        f_GPIO_Toggle=0;
    }
}
```

4.5 ADC 数模转换器模块

4.5.1 ADC 说明

12 位模数转换器(ADC)模块使用一个逐次逼近电路将模拟电平转换为一个 12 位的数字值。输入的模拟电平值必须在 AVREF 和 AVSS 的值之间。

- 带逐次逼近逻辑的模拟比较器
- 参考电压(AVREF)支持选择内部或者外部
- 自带固定电压参考源(INTVREF)
- 支持多路外部模拟输入 AIN[15:0]，内部固定电压参考源输入，以及 1/4VDD 输入
- 支持多序列转换模式，可灵活配置转换通道，转换顺序，转换次数
- 每个转换序列都有一个 20 位转换结果寄存器(ADC_DR)
- 支持多个外部触发源，可以触发转换序列
- 最大转换速度: 1MSPS
- 模拟输入范围：AVSS 到 AVREF

4.5.2 ADC 配置

4.5.3 ADC 转换速率

ADC 在使用时，PRLVAL 配置的值必须保证采样速度不超过手册规定的最大值(1MSPS)。如果 PCLK 频率是 20MHz，并且 PCLK/2 被选择位转换时钟，那么一个时钟周期就是 100ns。转换速度计算如下(假设 S/H 时间为默认值 6 个周期)：

(6 个 S/H 时钟周期) + (每位 1 个时钟转换周期 x 13 位) + (3 个同步和结果处理时钟周期) = 22 个周期：22 x 100ns = 2.2us (476ksps)

采样和保持时间的长度可以由下面公式计算：

S/H 时间 = (6 + (ADC_SHR – 3)) * (1/F_ANA) *

*：如果 PRLVAL 是 0，那么 F_ANA = PCLK

否则 PRLVAL 是其它任何值的话，F_ANA = PCLK / (2*PRLVAL)

4.5.4 ADC 单次转换模式

- **功能实例:**

  开启内部主频 48MHz,并作为系统时钟。

  使能 ADCIN10、ADCIN11 通道,12BIT ADC,参考电压选择内部 2.048V,单次转换模式，PRLVAL=2，ADC_SHR =6,ADC 采样周期=(6+(6-3))=9。

  ADC 转换周 F_ANA=PCLK/(2*PRLVAL)->F_ANA=48M/(2*2)=0.08us

  ADC 转换时间：ADC 采样周期+1 转换周期*12bit（或 10bit）+3 个处理结果周期=24 个转换周期=24*0.08us=520KSPS

  注意：

  1.ADC 转换速率不能超过 1MSPS。

  2.选择内部参考电压 Vref 需要接 104 电容到 GND。

  3.选择 INTVREF 1V 作为参考电压，Vref 不用外接 104 电容，且该脚可以作为其他功能使用。（B 版或以上版本 IC）

- **操作步骤:**

  1.SYSCON_CONFIG();函数配置

  2.GPIO_CONFIG();函数配置

  3.ADC_CONFIG();函数配置

  4.主函数读 ADC 数据

- **程序范例:**

```
/************************************************************/
//ADC Functions
//EntryParameter:NONE
//ReturnValue:NONE
/************************************************************/
void ADC_CONFIG(void)
{
    ADC12_RESET_VALUE();                          //ADC 所有寄存器复位赋值
    ADC12_CLK_CMD(ADC_CLK_CR , ENABLE);           //使能 ADC CLK
    ADC12_Software_Reset();                       //ADC 软件复位
    ADC12_Configure_Mode(ADC12_12BIT , One_shot_mode ,0, 6 ,2 , 2);

    //选择 12BIT ADC; 单次模式; 转换优先序列寄存器为 0; ADC_CLK=PCLK/2*2=0.08us; 转换序列个数为 2
    ADC12_Configure_VREF_Selecte(ADC12_VREFP_FVR2048_VREFN_VSS);
    //使用内部为参考电压 2.048mV
    ADC12_ConversionChannel_Config(ADC12_ADCIN10,ADC12_CV_RepeatNum1,ADC12_AVGDIS
    ,0);
    //转换序列 0,选择 ADCIN10 通道,连续重复采样次数为 1,平均值计算禁止
    ADC12_ConversionChannel_Config(ADC12_ADCIN11,ADC12_CV_RepeatNum1,ADC12_AVGDIS
```

```
        ,1);
        //转换序列 1,选择 ADCIN11 通道,连续重复采样次数为 1,平均值计算禁止
        ADC12_CMD(ENABLE);                                  //使能 ADC 模块
        ADC12_ready_wait();                                 //等待 ADC 模块配置完成
        ADC12_Control(ADC12_START);                         //ADC 模块启动
}
/*******************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*******************************************************************************/
void APT32F102_init(void)
{

        SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
        SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
        while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

        SYSCON_CONFIG();                            //syscon 参数  初始化
        CK_CPU_EnAllNormalIrq();                    //打开全局中断

        ADC_CONFIG()；
}

/*****************************************************************/
//main
/*****************************************************************/
volatile U32_T R_ADC_Buf1, R_ADC_Buf2;
int main(void)
{
        APT32F102_init();
        while(1)
        {
                SYSCON_IWDCNT_Reload();             //清狗
                ADC12_SEQEND_wait(0);               //等待转换序列 0 转换完成
                        R_ADC_Buf1= ADC0->DR[0];            //转换结果保存

                ADC12_SEQEND_wait(1);               //读取转换序列 1 数据
                R_ADC_Buf2= ADC0->DR[1];            //转换结果保存
                ADC12_Control(ADC12_START);         //ADC 模块启动
        }
}
```

4.5.5 ADC 单通道切换

初始化时，将转换序列设置为 1 个通道

```
void ADC12_CONFIG(void)
{
    ADC12_RESET_VALUE();      //ADC 所有寄存器复位赋值
    ADC12_CLK_CMD(ADC_CLK_CR , ENABLE);          //使能 ADC CLK
  ADC12_Software_Reset();            //ADC 软件复位
    ADC12_Configure_Mode(ADC12_12BIT , One_shot_mode ,0, 6 ,2 , 1);
//选择 12BIT ADC; 连续模式; 转换优先序列寄存器为 0; 采样保持时间=6 ;ADC_CLK=PCLK/2*2=0.2us; 转换序列个数为 2
    ADC12_Configure_VREF_Selecte(ADC12_VREFP_VDD_VREFN_VSS);  //选择 VDD 做参考电压
    ADC12_ConversionChannel_Config(ADC12_ADCIN0,ADC12_CV_RepeatNum1,ADC12_AVGDIS,0);
//转换序列 0,选择 ADCIN0 通道, 6 个转换周期, 连续重复采样次数为 1 平均值计算禁止
    ADC12_CMD(ENABLE);            //使能 ADC 模块
    ADC12_ready_wait();           //等待 ADC 模块配置完成
}
```

切换扫描两个 ADC 通道

```
void adc_test1(void)
{
    ADC12_ConversionChannel_Config(ADC12_ADCIN0,ADC12_CV_RepeatNum1,ADC12_AVGDIS,0);
    //channel switch
    ADC12_Control(ADC12_START);              //Start conversion
  ADC12_SEQEND_wait(0);                 //End of conversion wait
  R_ADC_Buf2[0]=ADC0->DR[0];            //ADC data0
  ADC12_Control(ADC12_STOP);
    ADC12_ConversionChannel_Config(ADC12_ADCIN1,ADC12_CV_RepeatNum1,ADC12_AVGDIS,0);
//channel switch
    ADC12_Control(ADC12_START);           //Start conversion
  ADC12_SEQEND_wait(0);                 //End of conversion wait
  R_ADC_Buf2[1]=ADC0->DR[0];            //ADC data1
  ADC12_Control(ADC12_STOP);
}
```

4.5.6 ADC 连续转换模式

- **功能实例:**

  开启内部主频 48MHz,并作为系统时钟。

  使能 ADCIN10、ADCIN11 通道,12BIT ADC,参考电压选择内部 2.048V,单次转换模式，PRLVAL=2，ADC_SHR =6,ADC 采样周期=(6+(6-3))=9。

  ADC 转换周期：F_ANA=PCLK/(2*PRLVAL)-F_ANA=48M/(2*2)=0.08us

ADC 转换时间：ADC 采样周期+1 转换周期*12bit（或 10bit）+3 个处理结果周期=24 个转换周期=24*0.08us=520KSPS

注意：

1.ADC 转换速率不能超过 1MSPS。

2.选择内部参考电压 Vref 需要接 104 电容到 GND。

3.选择 INTVREF 1V 作为参考电压，Vref 不用外接 104 电容，且该脚可以作为其他功能使用。（B 版或以上版本 IC）

- **操作步骤:**

1.SYSCON_CONFIG();函数配置

2.GPIO_CONFIG();函数配置

3.ADC_CONFIG();函数配置

4.主函数读 ADC 数据

- **程序范例:**

```
/*************************************************************/
//ADC Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void ADC_CONFIG(void)
{
    ADC12_RESET_VALUE();                                    //ADC 所有寄存器复位赋值
    ADC12_CLK_CMD(ADC_CLK_CR , ENABLE);                     //使能 ADC CLK
    ADC12_Software_Reset();                                 //ADC 软件复位
    ADC12_Configure_Mode(ADC12_12BIT , Continuous_mode ,0, 6 ,2 , 2);

    //选择 12BIT ADC; 单次模式; 转换优先序列寄存器为 0; ADC_CLK=PCLK/2*2=0.08us; 转换序列个数为 2
    ADC12_Configure_VREF_Selecte(ADC12_VREFP_FVR2048_VREFN_VSS);
    //使用内部为参考电压 2.048mV
    ADC12_ConversionChannel_Config(ADC12_ADCIN10,ADC12_CV_RepeatNum1,ADC12_AVGDIS,0);
    //转换序列 0,选择 ADCIN10 通道,连续重复采样次数为 1,平均值计算禁止
    ADC12_ConversionChannel_Config(ADC12_ADCIN11,ADC12_CV_RepeatNum1,ADC12_AVGDIS,1);
    //转换序列 1,选择 ADCIN11 通道,连续重复采样次数为 1,平均值计算禁止
    ADC12_CMD(ENABLE);                                      //使能 ADC 模块
    ADC12_ready_wait();                                     //等待 ADC 模块配置完成
    ADC12_Control(ADC12_START);                             //ADC 模块启动
}
/*************************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************************************/
```

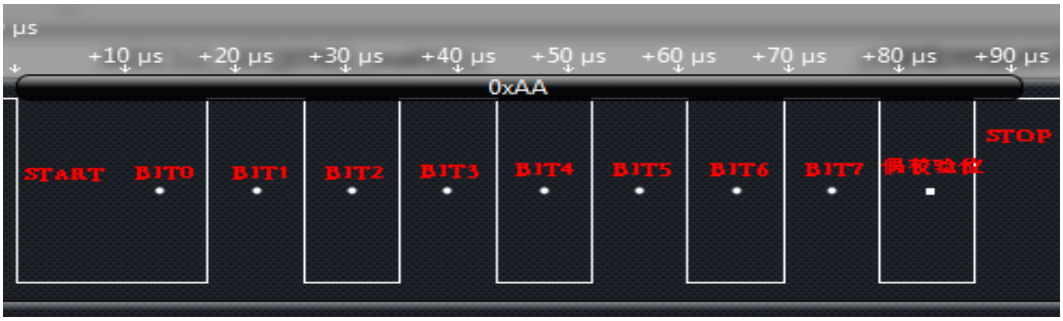
```
void APT32F102_init(void)
{
    SYSCON->PCER0=0xFFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                 //判断 IP 是否使能

    SYSCON_CONFIG();                             //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                     //打开全局中断

    ADC_CONFIG()；
}
/*****************************************************************/
//main
/*****************************************************************/
volatile U32_T R_ADC_Buf1, R_ADC_Buf2;
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                   //清狗
        ADC12_SEQEND_wait(0);                    //等待转换序列 0 转换完成
        R_ADC_Buf1= ADC0->DR[0];                 //转换结果保存

        ADC12_SEQEND_wait(1);                    //读取转换序列 1 数据
        R_ADC_Buf2= ADC0->DR[1];                 //转换结果保存
    }
}
```

4.6 UART 通用异步收发器模块

4.6.1 UART 发送接收

- **功能实例:**

开启内部主频 48MHz,并作为系统时钟。
波特率：PCLK/DIV=48M/416=115200
禁止奇偶数校验位。
发送顺序:bit0->bit7
UART 脚位选择：PA0.1->RXD0, PA0.0->TXD0
将 RXD 于 TXD 短接，发送 Uart 数据 0XAA，接收数据同样为 0XAA。
波形图如下：



- **操作步骤:**

1.SYSCON_CONFIG();函数配置
2.GPIO_CONFIG();函数配置
3.UART_CONFIG();函数
4.主函数发送 UART 数据
5.中断函数中接收 UART 数据

- **程序范例:**

```
/*************************************************************/
//UART Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void UART_CONFIG(void)
{
    UART_DeInit();                                  //UART 所有寄存器复位赋值
    UART_IO_Init(IO_UART0,0);                       //UART0 脚位选择 PA0.1->RXD0,
PA0.0->TXD0
    UARTInitRxTxIntEn(UART0,416, UART_PAR_NONE);
    //使能 Uart 中断, baudrate=48000000/416=115200, 禁止奇偶校验位
    UART0_Int_Enable();                             //uart0 中断向量使能
}
/*************************************************************/
```

```
//main
/*************************************************************/
volatile U8_T R_Uart_RDBUF;
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                 //清狗
        UARTTxByte (UART0,0XAA);                //发送 0xaa
        while(R_Uart_RDBUF!=0xAA);              //判断是否接收到 0XAA
    }
}
/*****************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*****************************************************************************/
void APT32F102_init(void)
{

    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                    //打开全局中断

    UART_CONFIG()；
}
/*************************************************************/
//UART0 Interrupt
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
extern volatile U8_T R_Uart_RDBUF;
void UART0IntHandler(void)
{
    if ((UART0->ISR&UART_RX_INT_S)==UART_RX_INT_S)
    {
        UART0->ISR=UART_RX_INT_S;
        R_Uart_RDBUF=UART_ReturnRxByte(UART0);
    }
}
```

4.7 EPT 增强型通用定时器模块

4.7.1 EPT 说明

- 16 位可复位计数器
- 可编程计数器计数方式
  - 递增计数（Up-counting）
  - 递减计数（Down-counting）
  - 递增递减计数（Up-down-counting）
- 7 路 PWM 输出，包括 4 路波形产生控制单元，支持 4 路独立输出或者 3 组互补输出：
  - 4 路独立的 PWM 输出，单边沿工作
  - 4 路独立的 PWM 输出，双边沿对称工作
  - 3 组独立的 PWM 互补输出 + 1 路独立的 PWM 输出
- 可编程的死区控制单元
- 通过软件异步重置 PWM 的波形输出
- 支持可编程的相位控制
- 异常情况处理控制单元
  - 异常事件发生时，自动触发预设波形输出
  - 多种触发方式，包括外部管脚和模拟比较器（如含有）
- 支持片间多设备同步
  - 支持多个 TIMER 间的同步触发
  - 触发源包括 GPIO 输入，其他外设触发，软件设置和事件触发
  - 支持单次触发和连续触发模式
- 支持单脉冲输出模式
- 支持突发计数模式
- 支持通过外部时钟计数
- 支持事件计数器，可通过配置事件计数器（最大 15）触发相应中断
- 支持 PWM 对更高载波频率进行斩波输出
- 支持捕获模式，最多支持 4 个捕获值存储

4.7.2 EPT 六路 PWM 互补输出死区可调

➢ **功能实例:**

开启内部主频 48MHz,并作为系统时钟。

计数器单周期时间：CLKS= MCLK /1=48MHz

输出周期为 100us，占空比为 50us,死区为 0.5us

波形如下：

PA0.10->CHAX　周期:CLKS*4800=100us,占空比:CLKS*2400=50us

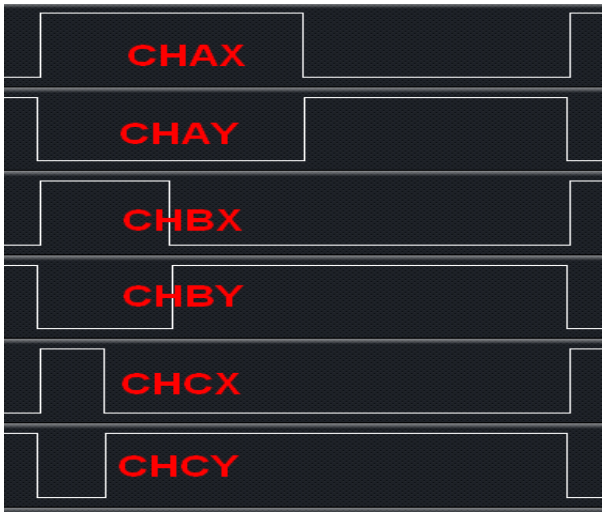
PB0.3->CHAY

PB0.2->CHBX　周期:CLKS*4800=100us,占空比:CLKS*1200=25us

PB0.4->CHBY

PB0.5->CHCX　周期:CLKS*4800=100us,占空比:CLKS*600=12.5us

PA0.4->CHCY



- **操作步骤:**

  1.SYSCON_CONFIG();函数配置

  2.EPT0_CONFIG();函数配置

- **程序范例:**

```
/*************************************************************/
    //EPT Functions
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*************************************************************/
    void EPT0_CONFIG(void)
    {
        EPT_Software_Prg();
        EPT_IO_SET(EPT_IO_CHAX,IO_NUM_PA10);
        EPT_IO_SET(EPT_IO_CHAY,IO_NUM_PB03);
        EPT_IO_SET(EPT_IO_CHBX,IO_NUM_PB02);
        EPT_IO_SET(EPT_IO_CHBY,IO_NUM_PB04);
        EPT_IO_SET(EPT_IO_CHCX,IO_NUM_PB05);
        EPT_IO_SET(EPT_IO_CHCY,IO_NUM_PA04);
        EPT_PWM_Config(EPT_Selecte_PCLK,EPT_CNTMD_increase,EPT_OPM_Continue,0);
        //PCLK 作为 TCLK 时钟，递增模式，连续模式，TCLK=PCLK/(0+1)
        EPT_PWMX_Output_Control(EPT_PWMA,EPT_CA_Selecte_CMPA,EPT_CB_Selecte_CMPA,EPT_PWM_ZR
        O_Event_OutHigh,EPT_PWM_PRD_Event_Nochange,EPT_PWM_CAU_Event_OutLow,EPT_PWM_CAD_Eve
        nt_OutLow,EPT_PWM_CBU_Event_Nochange,EPT_PWM_CBD_Event_Nochange,EPT_PWM_T1U_Event_No
        change,EPT_PWM_T1D_Event_Nochange,EPT_PWM_T2U_Event_Nochange,EPT_PWM_T2D_Event_Nocha
        nge);
        EPT_PWMX_Output_Control(EPT_PWMB,EPT_CA_Selecte_CMPB,EPT_CB_Selecte_CMPB,EPT_PWM_ZR
        O_Event_OutHigh,EPT_PWM_PRD_Event_Nochange,EPT_PWM_CAU_Event_OutLow,EPT_PWM_CAD_Eve
        nt_OutLow,EPT_PWM_CBU_Event_Nochange,EPT_PWM_CBD_Event_Nochange,EPT_PWM_T1U_Event_No
```

```
        change,EPT_PWM_T1D_Event_Nochange,EPT_PWM_T2U_Event_Nochange,EPT_PWM_T2D_Event_Nocha
        nge);
        EPT_PWMX_Output_Control(EPT_PWMC,EPT_CA_Selecte_CMPC,EPT_CB_Selecte_CMPC,EPT_PWM_ZR
        O_Event_OutHigh,EPT_PWM_PRD_Event_Nochange,EPT_PWM_CAU_Event_OutLow,EPT_PWM_CAD_Eve
        nt_OutLow,EPT_PWM_CBU_Event_Nochange,EPT_PWM_CBD_Event_Nochange,EPT_PWM_T1U_Event_No
        change,EPT_PWM_T1D_Event_Nochange,EPT_PWM_T2U_Event_Nochange,EPT_PWM_T2D_Event_Nocha
        nge);
        EPT_PRDR_CMPA_CMPB_CMPC_CMPD_Config(4800,2400,1200,600,0);//PRDR=2400,CMPA=1200,CMPB=
        600,CMPC=2400,CMPD=0
        EPT_DB_CLK_Config(0,24,24);//Fdbclk=Fhclk/(0+1)，DTR=24clk，DTF=24clk
        EPT_DBCR_Config(EPT_CHA_Selecte,EPT_CHAINSEL_PWMA_RISE_FALL,EPT_CHA_OUTSEL_EnRise_E
        nFall,EPT_PB_OUT_Reverse,EPT_PAtoCHX_PBtoCHY);//PWMA 作为互补输入源,CHX 上升沿，CHY 下降沿

        EPT_DBCR_Config(EPT_CHB_Selecte,EPT_CHBINSEL_PWMB_RISE_FALL,EPT_CHB_OUTSEL_EnRise_E
        nFall,EPT_PB_OUT_Reverse,EPT_PAtoCHX_PBtoCHY);//PWMB 作为互补输入源,CHX 上升沿，CHY 下降沿

        EPT_DBCR_Config(EPT_CHC_Selecte,EPT_CHCINSEL_PWMC_RISE_FALL,EPT_CHC_OUTSEL_EnRise_E
        nFall,EPT_PB_OUT_Reverse,EPT_PAtoCHX_PBtoCHY);//PWMC 作为互补输入源,CHX 上升沿，CHY 下降沿

        EPT_Start();
}
/*******************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*******************************************************************************/
void APT32F102_init(void)
{
        SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
        SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
        while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

        SYSCON_CONFIG();                            //syscon 参数 初始化
        CK_CPU_EnAllNormalIrq();                    //打开全局中断

        EPT0_CONFIG();                          //EPT0 initial
}

/*************************************************************/
//main
/*************************************************************/
int main(void)
{
        APT32F102_init();
```

```
        while(1)
        {
            SYSCON_IWDCNT_Reload();                    //清狗
        }
    }
```

- 代码说明:

EPT_IO_SET(EPT_IO_Mode_Type ***EPT_IO_X*** **,** EPT_IO_NUM_Type ***IO_Num_X*****)**

***EPT_IO_X*:**表示选择需要设置的 PWM 通道

***IO_Num_X*:**表示选择 GPIO 管脚 作为 PWM 功能

EPT_IO_CHAX->PA0.7/PA0.10/PA0.15

EPT_IO_CHAY->PB0.3/PB0.5/PA0.12

EPT_IO_CHBX->PB0.2/PA0.11/PA0.14

EPT_IO_CHBY->PB0.4/PA0.5/PA0.8

EPT_IO_CHCX->PB0.5/PA0.3/PB0.3/PB0.0

EPT_IO_CHCY->PB0.4/PA0.4/PA0.9/PA0.13

EPT_IO_CHD->PB0.3/PA0.8

EPT_PWM_Config(EPT_TCLK_Selecte_Type ***EPT_TCLK_Selecte_X*** **,**

EPT_CNTMD_SELECTE_Type ***EPT_CNTMD_SELECTE_X*** **,**

EPT_OPM_SELECTE_Type ***EPT_OPM_SELECTE_X*****,**

U16_T ***EPT_PSCR*****)**

***EPT_TCLK_Selecte_X*:**

表示选择 EPT 输入时钟源，可配置:

EPT_Selecte_PCLK/

EPT_Selecte_SYNCUSR3(EPT 同步触发源 3)。

EPT_CNTMD_SELECTE_X:

表示计数器模式选择，可配置:

EPT_CNTMD_increase（递增）

EPT_CNTMD_decrease（递减）

EPT_CNTMD_increaseTOdecrease（递增递减）

***EPT_OPM_SELECTE_X*:**

表示计数触发工作模式选择可配置:

EPT_OPM_Once(单次触发)/

EPT_OPM_Continue(连续触发)。

***EPT_PSCR*:**

表示 TCLK 分频配置，若选择 PCLK 作为输入时钟源

公式=PCLK/(**EPT_PSCR** +1)，可配置 0~0xffff。

EPT_PWMX_Output_Control(

EPT_PWMX_Selecte_Type ***EPT_PWMX_Selecte*** **,**

EPT_CA_Selecte_Type ***EPT_CA_Selecte_X,***

EPT_CB_Selecte_Type ***EPT_CB_Selecte_X ,***

EPT_PWM_ZRO_Output_Type ***EPT_PWM_ZRO_Event_Output,***

EPT_PWM_PRD_Output_Type ***EPT_PWM_PRD_Event_Output,***

EPT_PWM_CAU_Output_Type ***EPT_PWM_CAU_Event_Output,***

EPT_PWM_CAD_Output_Type ***EPT_PWM_CAD_Event_Output ,***

EPT_PWM_CBU_Output_Type ***EPT_PWM_CBU_Event_Output,***

EPT_PWM_CBD_Output_Type ***EPT_PWM_CBD_Event_Output ,***

EPT_PWM_T1U_Output_Type ***EPT_PWM_T1U_Event_Output,***

EPT_PWM_T1D_Output_Type ***EPT_PWM_T1D_Event_Output ,***

EPT_PWM_T2U_Output_Type ***EPT_PWM_T2U_Event_Output,***

EPT_PWM_T2D_Output_Type ***EPT_PWM_T2D_Event_Output*)**

***EPT_PWMX_Selecte*:**

PWMX 输出配置，可配置:

EPT_PWMA

EPT_PWMB

EPT_PWMC

EPT_PWMD

***EPT_CA_Selecte_X*:**

CA 比较值选择，可配置:

EPT_CA_Selecte_CMPA

EPT_CA_Selecte_CMPB

EPT_CA_Selecte_CMPC

EPT_CA_Selecte_CMPD

***EPT_CB_Selecte_X*:**

CB 比较值选择，可配置:

EPT_CB_Selecte_CMPA

EPT_CB_Selecte_CMPB

EPT_CB_Selecte_CMPC

EPT_CB_Selecte_CMPD

***EPT_PWM_ZRO_Event_Output*:**

当计数器事件为 ZRO 时，PWMX 管脚输出配置，可配置:

EPT_PWM_ZRO_Event_OutHigh(输出高状态) (下同)

EPT_PWM_ZRO_Event_OutLow(输出低状态) (下同)

EPT_PWM_ZRO_Event_Nochange(输出无变化) (下同)

EPT_PWM_ZRO_Event_Negate(输出反向)(下同)

***EPT_PWM_PRD_Event_Output*:**

当计数器事件为 PRD 时,PWMX 管脚输出配置，可配置:

EPT_PWM_PRD_Event_OutHigh

EPT_PWM_PRD_Event_OutLow

EPT_PWM_PRD_Event_Nochange

EPT_PWM_PRD_Event_Negate。

***EPT_PWM_CAU_Event_Output*:**

当计数器事件为 CAU 时,PWMX 管脚输出配置，可配置:

EPT_PWM_CAU_Event_OutHigh

EPT_PWM_CAU_Event_OutLow

EPT_PWM_CAU_Event_Nochange

EPT_PWM_CAU_Event_Negate

***EPT_PWM_CAD_Event_Output*:**

当计数器事件为 CAD 时,PWMX 管脚输出配置，可配置:

EPT_PWM_CAD_Event_OutHigh

EPT_PWM_CAD_Event_OutLow

EPT_PWM_CAD_Event_Nochange

EPT_PWM_CAD_Event_Negate

***EPT_PWM_CBU_Event_Output*:**

当计数器事件为 CBU 时,PWMX 管脚输出配置，可配置:

EPT_PWM_CBU_Event_OutHigh

EPT_PWM_CBU_Event_OutLow

EPT_PWM_CBU_Event_Nochange

EPT_PWM_CBU_Event_Negate

***EPT_PWM_CBD_Event_Output*:**

当计数器事件为 CBD 时,PWMX 管脚输出配置，可配置:

EPT_PWM_CBD_Event_OutHigh

EPT_PWM_CBD_Event_OutLow

EPT_PWM_CBD_Event_Nochange

EPT_PWM_CBD_Event_Negate

***EPT_PWM_T1U_Event_Output*:**

当计数器事件为 T1U 时,PWMX 管脚输出配置，可配置:

EPT_PWM_T1U_Event_OutHigh

EPT_PWM_T1U_Event_OutLow

EPT_PWM_T1U_Event_Nochange

EPT_PWM_T1U_Event_Negate

***EPT_PWM_T1D_Event_Output*:**

当计数器事件为 T1D 时,PWMX 管脚输出配置，可配置:

EPT_PWM_T1D_Event_OutHigh

EPT_PWM_T1D_Event_OutLow

EPT_PWM_T1D_Event_Nochange

EPT_PWM_T1D_Event_Negate

***EPT_PWM_T2U_Event_Output*:**

当计数器事件为 T2U 时,PWMX 管脚输出配置，可配置:

EPT_PWM_T2U_Event_OutHigh

EPT_PWM_T2U_Event_OutLow

EPT_PWM_T2U_Event_Nochange

EPT_PWM_T2U_Event_Negate

***EPT_PWM_T2D_Event_Output*:**

当计数器事件为 T2D 时,PWMX 管脚输出配置，可配置:

EPT_PWM_T2D_Event_OutHigh

EPT_PWM_T2D_Event_OutLow

EPT_PWM_T2D_Event_Nochange

EPT_PWM_T2D_Event_Negate

EPT_PRDR_CMPA_CMPB_CMPC_CMPD_Config

(U16_T ***EPT_PRDR_Value*** **,**U16_T ***EPT_CMPA_Value*** **,** U16_T ***EPT_CMPB_Value*** **,** U16_T ***EPT_CMPC_Value*** **,** U16_T ***EPT_CMPD_Value*****)**

EPT_PRDR_Value**:**当前周期设定，可配置 0~0Xffff

EPT_CMPA_Value**:**当前周期设定，可配置 0~0Xffff

EPT_CMPB_Value**:**当前周期设定，可配置 0~0Xffff

EPT_CMPC_Value**:**当前周期设定，可配置 0~0Xffff

EPT_CMPD_Value**:**当前周期设定，可配置 0~0Xffff

EPT_DB_CLK_Config(U16_T ***DPSC*** **,** U16_T ***DTR*** **,** U16_T ***DTF*****)**

DPSC**:**死区时钟分频设定参数，公式=FHclk/(DPSC+1),可配置:0~0xffff

EPT_DBCR_Config(

EPT_CHX_Selecte_Type ***EPT_CHX_Selecte,***

EPT_INSEL_Type ***EPT_INSEL_X,***

EPT_OUTSEL_Type ***EPT_OUTSEL_X,***

EPT_OUT_POLARITY_Type ***EPT_OUT_POLARITY_X,***

EPT_OUT_SWAP_Type ***EPT_OUT_SWAP_X*****)**

EPT_CHX_Selecte**:**

CHA 通道选择可配置:

EPT_CHA_Selecte

EPT_CHB_Selecte

EPT_CHC_Selecte

EPT_INSEL_X**:**

CHA/ CHB/CHC 输入源 PWMX 选择

EPT_CHA_Selecte 可配置:

EPT_CHAINSEL_PWMA_RISE_FALL

EPT_CHAINSEL_PWMB_RISE_PWMA_FALL

EPT_CHAINSEL_PWMA_RISE_PWMB_FALL

EPT_CHAINSEL_PWMB_RISE_FALL

EPT_CHB_Selecte 可配置:

EPT_CHBINSEL_PWMB_RISE_FALL

EPT_CHBINSEL_PWMC_RISE_PWMB_FALL

EPT_CHBINSEL_PWMB_RISE_PWMC_FALL

EPT_CHBINSEL_PWMC_RISE_FALL

EPT_CHC_Selecte 可配置:

EPT_CHCINSEL_PWMC_RISE_FALL

EPT_CHCINSEL_PWMD_RISE_PWMC_FALL

EPT_CHCINSEL_PWMC_RISE_PWMD_FALL

EPT_CHCINSEL_PWMD_RISE_FALL

EPT_OUTSEL_X**:**

CHA/CHB/CHC 输出死区配置使能

EPT_CHA_Selecte 可配置:

EPT_CHA_OUTSEL_PWMA_PWMB_Bypass

EPT_CHA_OUTSEL_DisRise_EnFall

EPT_CHA_OUTSEL_EnRise_DisFall

EPT_CHA_OUTSEL_EnRise_EnFall

EPT_CHB_Selecte 可配置:

EPT_CHB_OUTSEL_PWMB_PWMC_Bypass

EPT_CHB_OUTSEL_DisRise_EnFall

EPT_CHB_OUTSEL_EnRise_DisFall

EPT_CHB_OUTSEL_EnRise_EnFall

EPT_CHC_Selecte 可配置:

EPT_CHC_OUTSEL_PWMC_PWMD_Bypass

EPT_CHC_OUTSEL_DisRise_EnFall

EPT_CHC_OUTSEL_EnRise_DisFall

EPT_CHC_OUTSEL_EnRise_EnFall

***EPT_OUT_POLARITY_X*:**

CHAX/CHBX/CHCX CHAY/CHBY/CHCY 输出状态翻转设定

EPT_CHA_Selecte 可配置:

EPT_PA_PB_OUT_Direct(CHAX、CHAY 直接输出)

EPT_PA_OUT_Reverse（CHAX 输出反向，CHAY 保持不变）

EPT_PB_OUT_Reverse（CHAY 输出反向，CHAX 保持不变）

EPT_PA_PB_OUT_Reverse (CHAX、CHAY 输出全反向)

EPT_CHB_Selecte 可配置:

EPT_PA_PB_OUT_Direct(CHBX、CHBY 直接输出)

EPT_PA_OUT_Reverse（CHBX 输出反向，CHBY 保持不变）

EPT_PB_OUT_Reverse（CHBY 输出反向，CHBX 保持不变）

EPT_PA_PB_OUT_Reverse (CHBX、CHBY 输出全反向)

EPT_CHC_Selecte 可配置:

EPT_PA_PB_OUT_Direct(CHCX、CHCY 直接输出)

EPT_PA_OUT_Reverse（CHCX 输出反向，CHCY 保持不变）

EPT_PB_OUT_Reverse（CHCY 输出反向，CHCX 保持不变）

EPT_PA_PB_OUT_Reverse (CHCX、CHCY 输出全反向)

***EPT_OUT_SWAP_X*:**

CHAX/CHBX/CHCX CHAY/CHBY/CHCY 输出状态互换设定

EPT_CHA_Selecte 可配置:

EPT_PAtoCHX_PBtoCHY(CHAX、CHAY 直接输出)

EPT_PBtoCHX_PBtoCHY(CHAX、CHAY 同时输出 CHAY)

EPT_PAtoCHX_PAtoCHY(CHAX、CHAY 同时输出 CHAX)

EPT_PBtoCHX_PAtoCHY(CHAX、CHAY 输出互换)

EPT_CHB_Selecte 可配置:

EPT_PAtoCHX_PBtoCHY(CHBX、CHBY 直接输出)

EPT_PBtoCHX_PBtoCHY(CHBX、CHBY 同时输出 CHBY)

EPT_PAtoCHX_PAtoCHY(CHBX、CHBY 同时输出 CHBX)

EPT_PBtoCHX_PAtoCHY(CHBX、CHBY 输出互换)

EPT_CHC_Selecte 可配置:

EPT_PAtoCHX_PBtoCHY(CHCX、CHCY 直接输出)
EPT_PBtoCHX_PBtoCHY(CHCX、CHCY 同时输出 CHCY)
EPT_PAtoCHX_PAtoCHY(CHCX、CHCY 同时输出 CHCX)
EPT_PBtoCHX_PAtoCHY(CHCX、CHCY 输出互换)

4.7.3 EPT 输入捕捉

- **功能实例:**
  开启内部主频 48MHz,并作为系统时钟。
  计数器单周期时间：CLKS= MCLK /1=48MHz
  PA0.0 作为捕捉输入口
  捕捉波形周期=100uS，占空比=79.2us
  R_CMPA_BUF 存储低电平计数值，R_CMPB_BUF 存储周期计数值
- **操作步骤:**
  1. SYSCON_CONFIG();函数配置
  2. GPIO_CONFIG();函数配置
  3. EPT0_CONFIG();函数配置
  4. ET_CONFIG();函数配置
- **程序范例:**

```
/*************************************************************/
    //GPIO Initial
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*************************************************************/
    void GPIO_CONFIG(void)
    {
        //------------  EXI FUNTION  -------------------------------/
        //EXI0_INT=  EXI0/EXI16,EXI1_INT=  EXI1/EXI17,  EXI2_INT=EXI2~EXI3/EXI18/EXI19,  EXI3_INT=EXI4~EXI9,
    EXI4_INT=EXI10~EXI15
        GPIO_IntGroup_Set(PA0,0,Selete_EXI_PIN0);                    //EXI0 set PBA.0
        GPIOA0_EXI_Init(EXI0);                                       //PA0.0 as input
        EXTI_trigger_CMD(ENABLE,EXI_PIN0,_EXIRT);                    //ENABLE rising edge
        EXTI_interrupt_CMD(ENABLE,EXI_PIN0);                         //enable EXI
        GPIO_EXTI_interrupt(GPIOA0,0b00000000000001);                    //enable GPIOA00 as EXI
    }
    /*************************************************************/
    //ETP0 Functions
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*************************************************************/
    void EPT0_CONFIG(void)
    {
        EPT_Software_Prg();
```

```
    EPT_Capture_Config(EPT_Selecte_PCLK,EPT_CNTMD_increase,EPT_CAPMD_Continue,EPT_CAP_EN,EPT
    _LDARST_EN,EPT_LDBRST_DIS,EPT_LDCRST_DIS,EPT_LDDRST_DIS,1,0);//TCLK=pclk/(1+0),CMPAload
    CMPBload
    EPT_SYNCR_Config(EPT_Triggle_Continue,EPT_SYNCUSR0_REARMTrig_DIS,EPT_TRGSRC0_ExtSync_S
    YNCUSR0,EPT_TRGSRC1_ExtSync_SYNCUSR4,0x04);//使能 SYNCUSR2 ,连续触发
    EPT_PRDR_CMPA_CMPB_CMPC_CMPD_Config(0XFFFF,0,0,0,0);
    EPT_Int_Enable(EPT_CAP_LD0);//CMPA 载入中断
    EPT_Int_Enable(EPT_CBP_LD1);//CMPB 载入中断
    EPT_Vector_Int_Enable();
    EPT_Start();
}
/*****************************************************************/
//ET Initial
//EntryParameter:NONE
//ReturnValue:NONE
/*****************************************************************/
void ET_CONFIG(void)
{
    ET_DeInit()
    ET_CH0_SRCSEL(ET_SRC0,ENABLE,ET_EXI_SYNC0);
    ET_CH0_CONTROL(ENABLE,TRG_HW,ET_EPT0_TRGSRC2);
    ET_ENABLE();
}
/*****************************************************************/
//syscon Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*****************************************************************/
void SYSCON_CONFIG(void)
{
    SYSCON_RST_VALUE();
    SYSCON_General_CMD(ENABLE,ENDIS_ISOSC);
    SYSCON_HFOSC_SELECTE(HFOSC_SELECTE_48M);
    SystemCLK_HCLKDIV_PCLKDIV_Config(SYSCLK_HFOSC,HCLK_DIV_1,PCLK_DIV_1,HFOSC_48M);
    SYSCON_IWDCNT_Config(IWDT_TIME_1S,IWDT_INTW_DIV_7);
    SYSCON_WDT_CMD(ENABLE);
    SYSCON_IWDCNT_Reload();
    SYSCON->EVTRG=0X00|0x01<<20;
    SYSCON->EVPS=0X00;
}
/*****************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
```

```
/*******************************************************************************/
void APT32F102_init(void)
{

    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                    //打开全局中断

    GPIO_CONFIG();                          //GPIO initial
    EPT0_CONFIG();                          //EPT0 initial
    ET_CONFIG();
}
volatile int R_CMPA_BUF,R_CMPB_BUF;
/*************************************************************/
//EPT0 Interrupt
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void EPT0IntHandler(void)
{
    if((EPT0->MISR&EPT_CAP_LD0)==EPT_CAP_LD0)
    {
        EPT0->ICR=EPT_CAP_LD0;
        EXTI_trigger_CMD(DISABLE,EXI_PIN0,_EXIRT);
        EXTI_trigger_CMD(ENABLE,EXI_PIN0,_EXIFT);
        R_CMPA_BUF=EPT0->CMPA;              //低电平计数
    }
    else if((EPT0->MISR&EPT_CBP_LD1)==EPT_CBP_LD1)
    {
        EPT0->ICR=EPT_CBP_LD1;
        EXTI_trigger_CMD(ENABLE,EXI_PIN0,_EXIRT);
        EXTI_trigger_CMD(DISABLE,EXI_PIN0,_EXIFT);
        R_CMPB_BUF=EPT0->CMPB;              //周期计数值
    }
}
/*************************************************************/
//main
/*************************************************************/
int main(void)
{
    APT32F102_init();
```

```
        while(1)
        {
            SYSCON_IWDCNT_Reload();                    //清狗
        }
    }
```

- 代码说明:

EPT_Capture_Config(

EPT_TCLK_Selecte_Type ***EPT_TCLK_Selecte_X,***

EPT_CNTMD_SELECTE_Type ***EPT_CNTMD_SELECTE_X,***

EPT_CAPMD_SELECTE_Type ***EPT_CAPMD_SELECTE_X,***

EPT_CAPLDEN_CMD_Type ***CAP_CMD,***

EPT_LOAD_CMPA_RST_CMD_Type ***EPT_LOAD_CMPA_RST_CMD,***

EPT_LOAD_CMPB_RST_CMD_Type ***EPT_LOAD_CMPB_RST_CMD,***

EPT_LOAD_CMPC_RST_CMD_Type ***EPT_LOAD_CMPC_RST_CMD,***

EPT_LOAD_CMPD_RST_CMD_Type ***EPT_LOAD_CMPD_RST_CMD,***

U8_T ***EPT_STOP_WRAP,***

U16_T ***EPT_PSCR*)**

***EPT_TCLK_Selecte_X*:**

表示选择 EPT 输入时钟源，可配置:

EPT_Selecte_PCLK/

EPT_Selecte_SYNCUSR3(EPT 同步触发源 3)

EPT_CNTMD_SELECTE_X:

表示计数器模式选择，可配置:

EPT_CNTMD_increase（递增）

EPT_CNTMD_decrease（递减）

EPT_CNTMD_increaseTOdecrease（递增递减）。

EPT_CNTMD_SELECTE_X:

表示计数器模式选择，可配置:

EPT_CNTMD_increase（递增）

EPT_CNTMD_decrease（递减）

EPT_CNTMD_increaseTOdecrease（递增递减）。

***CAP_CMD*:**

捕捉使能位，可配置:

EPT_CAP_EN（使能）

EPT_CAP_DIS（禁止）

***EPT_LOAD_CMPA_RST_CMD*:**

CMPA 载入计数器重新计数，可配置:

EPT_LDARST_EN（使能）

EPT_LDARST_DIS（禁止）

***EPT_LOAD_CMPB_RST_CMD*:**

CMPB 载入计数器重新计数，可配置:

EPT_LDBRST_EN（使能）

EPT_LDBRST_DIS（禁止）

***EPT_LOAD_CMPC_RST_CMD*:**

CMPC 载入计数器重新计数，可配置:

EPT_LDCRST_EN（使能）

EPT_LDCRST_DIS（禁止）

***EPT_LOAD_CMPD_RST_CMD*:**

CMPD 载入计数器重新计数，可配置:

EPT_LDDRST_EN（使能）

EPT_LDDRST_DIS（禁止）

EPT_SYNCR_Config(

EPT_Triggle_Mode_Type ***EPT_Triggle_X* ,**

EPT_SYNCUSR0_REARMTrig_Selecte_Type

***EPT_SYNCUSR0_REARMTrig_Selecte* ,**

EPT_TRGSRC0_ExtSync_Selected_Type ***EPT_TRGSRC0_ExtSync_Selected* ,**

EPT_TRGSRC1_ExtSync_Selected_Type ***EPT_TRGSRC1_ExtSync_Selected* ,**

U8_T ***EPT_SYNCR_EN***

)

***EPT_Triggle_X*:**

同步触发单次或连续触发选择，可配置:

EPT_Triggle_Continue（连续触发）

EPT_Triggle_Once（单次触发）

***EPT_SYNCUSR0_REARMTrig_Selecte*:**

SYNCUSR0 单次触发时 rearm 清除事件选择，可配置:

EPT_SYNCUSR0_REARMTrig_DIS（禁止）

EPT_SYNCUSR0_REARMTrig_T1(T1 事件触发时 rearm)

EPT_SYNCUSR0_REARMTrig_T2(T2 事件触发时 rearm)

EPT_SYNCUSR0_REARMTrig_T1T2(T1 或 T2 事件触发时 rearm)

***EPT_TRGSRC0_ExtSync_Selected*:**

选择同步事件作为 TRGSRC0 触发事件 ExtSyn，可配置:0~5

EPT_TRGSRC0_ExtSync_SYNCUSR0

……

EPT_TRGSRC0_ExtSync_SYNCUSR5

***EPT_TRGSRC1_ExtSync_Selected*:**

选择同步事件作为 TRGSRC1 触发事件 ExtSync 可配置:0~5

EPT_TRGSRC1_ExtSync_SYNCUSR0

……

EPT_TRGSRC1_ExtSync_SYNCUSR5

EPT_PRDR_CMPA_CMPB_CMPC_CMPD_Config

(U16_T ***EPT_PRDR_Value*** **,** U16_T ***EPT_CMPA_Value*** **,** U16_T ***EPT_CMPB_Value*** **,** U16_T ***EPT_CMPC_Value*** **,** U16_T ***EPT_CMPD_Value*****)**

***EPT_PRDR_Value*:**当前周期设定，可配置:0~0Xffff。

EPT_CMPA_Value:当前周期设定，可配置:0~0Xffff。
EPT_CMPB_Value:当前周期设定，可配置:0~0Xffff。
EPT_CMPC_Value:当前周期设定，可配置:0~0Xffff。
EPT_CMPD_Value:当前周期设定，可配置:0~0Xffff。

4.7.4 EPT 四路 PWM 独立输出

- **功能实例:**

开启内部主频 48MHz,并作为系统时钟。
计数器单周期时间：CLKS= MCLK /1=48MHz
输出周期为 100us
波形如下：
PA0.10->CHAX　周期:CLKS*4800=100us,占空比:CLKS*2400=50us
PB0.2->CHBX　周期:CLKS*4800=100us,占空比:CLKS*1200=25us
PB0.5->CHCX　周期:CLKS*4800=100us,占空比:CLKS*600=12.5us
PA0.4->CHD　周期:CLKS*4800=100us,占空比:CLKS*300=6.25us

- **操作步骤:**

1.SYSCON_CONFIG();函数配置
2.EPT0_CONFIG();函数配置

- **程序范例:**

```
/************************************************************/
//ETP0 Functions
//EntryParameter:NONE
//ReturnValue:NONE
/************************************************************/
void EPT0_CONFIG(void)
{
    EPT_Software_Prg();
    EPT_IO_SET(EPT_IO_CHAX,IO_NUM_PA10);
    EPT_IO_SET(EPT_IO_CHBX,IO_NUM_PB02);
    EPT_IO_SET(EPT_IO_CHCX,IO_NUM_PB05);
    EPT_IO_SET(EPT_IO_CHD,IO_NUM_PA08);
```

```
    EPT_PWM_Config(EPT_Selecte_PCLK,EPT_CNTMD_increase,EPT_OPM_Continue,0);//PCLK 作为TCLK 时
    钟，递增模式，连续模式，TCLK=PCLK/(0+1)
    EPT_PWMX_Output_Control(EPT_PWMA,EPT_CA_Selecte_CMPA,EPT_CB_Selecte_CMPA,EPT_PWM_ZR
    O_Event_OutHigh,EPT_PWM_PRD_Event_Nochange,EPT_PWM_CAU_Event_OutLow,EPT_PWM_CAD_Eve
    nt_OutLow,EPT_PWM_CBU_Event_Nochange,EPT_PWM_CBD_Event_Nochange,EPT_PWM_T1U_Event_No
    change,EPT_PWM_T1D_Event_Nochange,EPT_PWM_T2U_Event_Nochange,EPT_PWM_T2D_Event_Nocha
    nge);
    EPT_PWMX_Output_Control(EPT_PWMB,EPT_CA_Selecte_CMPB,EPT_CB_Selecte_CMPB,EPT_PWM_ZR
    O_Event_OutHigh,EPT_PWM_PRD_Event_Nochange,EPT_PWM_CAU_Event_OutLow,EPT_PWM_CAD_Eve
    nt_OutLow,EPT_PWM_CBU_Event_Nochange,EPT_PWM_CBD_Event_Nochange,EPT_PWM_T1U_Event_No
    change,EPT_PWM_T1D_Event_Nochange,EPT_PWM_T2U_Event_Nochange,EPT_PWM_T2D_Event_Nocha
    nge);
    EPT_PWMX_Output_Control(EPT_PWMC,EPT_CA_Selecte_CMPC,EPT_CB_Selecte_CMPC,EPT_PWM_ZR
    O_Event_OutHigh,EPT_PWM_PRD_Event_Nochange,EPT_PWM_CAU_Event_OutLow,EPT_PWM_CAD_Eve
    nt_OutLow,EPT_PWM_CBU_Event_Nochange,EPT_PWM_CBD_Event_Nochange,EPT_PWM_T1U_Event_No
    change,EPT_PWM_T1D_Event_Nochange,EPT_PWM_T2U_Event_Nochange,EPT_PWM_T2D_Event_Nocha
    nge);
    EPT_PWMX_Output_Control(EPT_PWMD,EPT_CA_Selecte_CMPD,EPT_CB_Selecte_CMPD,EPT_PWM_ZR
    O_Event_OutHigh,EPT_PWM_PRD_Event_Nochange,EPT_PWM_CAU_Event_OutLow,EPT_PWM_CAD_Eve
    nt_OutLow,EPT_PWM_CBU_Event_Nochange,EPT_PWM_CBD_Event_Nochange,EPT_PWM_T1U_Event_No
    change,EPT_PWM_T1D_Event_Nochange,EPT_PWM_T2U_Event_Nochange,EPT_PWM_T2D_Event_Nocha
    nge);
    EPT_PRDR_CMPA_CMPB_CMPC_CMPD_Config(4800,2400,1200,600,300);//PRDR=4800,CMPA=2400,CMP
    B=1200,CMPC=600,CMPD=300
    EPT_DBCR_Config(EPT_CHA_Selecte,EPT_CHAINSEL_PWMA_RISE_FALL,EPT_CHA_OUTSEL_PWMA_P
    WMB_Bypass,EPT_PA_PB_OUT_Direct,EPT_PAtoCHX_PBtoCHY);//PWMA 输作为CHAX 输出源
    EPT_DBCR_Config(EPT_CHB_Selecte,EPT_CHBINSEL_PWMB_RISE_FALL,EPT_CHB_OUTSEL_PWMB_P
    WMC_Bypass,EPT_PA_PB_OUT_Direct,EPT_PAtoCHX_PBtoCHY);//PWMB 输作为CHBX 输出源
    EPT_DBCR_Config(EPT_CHC_Selecte,EPT_CHCINSEL_PWMC_RISE_FALL,EPT_CHC_OUTSEL_PWMC_P
    WMD_Bypass,EPT_PA_PB_OUT_Direct,EPT_PAtoCHX_PBtoCHY);//PWMC 输作为CHCX 输出源
    EPT_Start();
}
/*****************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*****************************************************************************/
void APT32F102_init(void)
{
    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数 初始化
```

```
    CK_CPU_EnAllNormalIrq();                    //打开全局中断


    EPT0_CONFIG();                              //EPT0 initial
}
/*************************************************************/
//main
/*************************************************************/
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                 //清狗
    }
}
```

- 代码说明:

EPT_IO_SET(EPT_IO_Mode_Type ***EPT_IO_X*** **,** EPT_IO_NUM_Type ***IO_Num_X*****)**

***EPT_IO_X*:**

表示选择需要设置的 PWM 通道

***IO_Num_X*:**

表示选择 GPIO 管脚 作为 PWM 功能

EPT_IO_CHAX->PA0.7/PA0.10/PA0.15

EPT_IO_CHAY->PB0.3/PB0.5/PA0.12

EPT_IO_CHBX->PB0.2/PA0.11/PA0.14

EPT_IO_CHBY->PB0.4/PA0.5/PA0.8

EPT_IO_CHCX->PB0.5/PA0.3/PB0.3/PB0.0

EPT_IO_CHCY->PB0.4/PA0.4/PA0.9/PA0.13

EPT_IO_CHD->PB0.3/PA0.8

EPT_PWM_Config(EPT_TCLK_Selecte_Type ***EPT_TCLK_Selecte_X*** **,**

EPT_CNTMD_SELECTE_Type ***EPT_CNTMD_SELECTE_X*** **,**

EPT_OPM_SELECTE_Type ***EPT_OPM_SELECTE_X*****,**

U16_T ***EPT_PSCR*****)**

***EPT_TCLK_Selecte_X*:**

表示选择 EPT 输入时钟源,可配置:

EPT_Selecte_PCLK/

EPT_Selecte_SYNCUSR3(EPT 同步触发源 3)

EPT_CNTMD_SELECTE_X:

表示计数器模式选择,可以配置:

EPT_CNTMD_increase（递增）

EPT_CNTMD_decrease（递减）

EPT_CNTMD_increaseTOdecrease（递增递减）。

***EPT_OPM_SELECTE_X*:**

表示计数触发工作模式选择,可配置:

EPT_OPM_Once(单次触发)

EPT_OPM_Continue(连续触发)

***EPT_PSCR*:**

表示 TCLK 分频配置，若选择 PCLK 作为输入时钟源，公式=PCLK/(**EPT_PSCR** +1)

可配置 0~0xffff。

EPT_PWMX_Output_Control(

EPT_PWMX_Selecte_Type ***EPT_PWMX_Selecte*** **,**

EPT_CA_Selecte_Type ***EPT_CA_Selecte_X,***

EPT_CB_Selecte_Type ***EPT_CB_Selecte_X*** **,**

EPT_PWM_ZRO_Output_Type ***EPT_PWM_ZRO_Event_Output,***

EPT_PWM_PRD_Output_Type ***EPT_PWM_PRD_Event_Output,***

EPT_PWM_CAU_Output_Type ***EPT_PWM_CAU_Event_Output,***

EPT_PWM_CAD_Output_Type ***EPT_PWM_CAD_Event_Output*** **,**

EPT_PWM_CBU_Output_Type ***EPT_PWM_CBU_Event_Output,***

EPT_PWM_CBD_Output_Type ***EPT_PWM_CBD_Event_Output*** **,**

EPT_PWM_T1U_Output_Type ***EPT_PWM_T1U_Event_Output,***

EPT_PWM_T1D_Output_Type ***EPT_PWM_T1D_Event_Output*** **,**

EPT_PWM_T2U_Output_Type ***EPT_PWM_T2U_Event_Output,***

EPT_PWM_T2D_Output_Type ***EPT_PWM_T2D_Event_Output*** **)**

***EPT_PWMX_Selecte*:**

PWMX 输出配置,可配置:

EPT_PWMA

EPT_PWMB

EPT_PWMC

EPT_PWMD。

***EPT_CA_Selecte_X*:**

CA 比较值选择,可配置:

EPT_CA_Selecte_CMPA

EPT_CA_Selecte_CMPB

EPT_CA_Selecte_CMPC /

EPT_CA_Selecte_CMPD

***EPT_CB_Selecte_X*:**

CB 比较值选择,可配置:

EPT_CB_Selecte_CMPA

EPT_CB_Selecte_CMPB

EPT_CB_Selecte_CMPC

EPT_CB_Selecte_CMPD。

***EPT_PWM_ZRO_Event_Output*:**

当计数器事件为 ZRO 时，PWMX 管脚输出配置,可配置:

EPT_PWM_ZRO_Event_OutHigh

EPT_PWM_ZRO_Event_OutLow

EPT_PWM_ZRO_Event_Nochange

EPT_PWM_ZRO_Event_Negate

***EPT_PWM_PRD_Event_Output*:**

当计数器事件为 PRD 时,PWMX 管脚输出配置,可配置:

EPT_PWM_PRD_Event_OutHigh

EPT_PWM_PRD_Event_OutLow

EPT_PWM_PRD_Event_Nochange

EPT_PWM_PRD_Event_Negate。

***EPT_PWM_CAU_Event_Output*:**

当计数器事件为 CAU 时,PWMX 管脚输出配置,可配置:

EPT_PWM_CAU_Event_OutHigh

EPT_PWM_CAU_Event_OutLow

EPT_PWM_CAU_Event_Nochange

EPT_PWM_CAU_Event_Negate

***EPT_PWM_CAD_Event_Output*:**

当计数器事件为 CAD 时,PWMX 管脚输出配置,可配置:

EPT_PWM_CAD_Event_OutHigh

EPT_PWM_CAD_Event_OutLow/

EPT_PWM_CAD_Event_Nochange

EPT_PWM_CAD_Event_Negate。

***EPT_PWM_CBU_Event_Output*:**

当计数器事件为 CBU 时,PWMX 管脚输出配置,可配置:

EPT_PWM_CBU_Event_OutHigh

EPT_PWM_CBU_Event_OutLow

EPT_PWM_CBU_Event_Nochange

EPT_PWM_CBU_Event_Negate。

***EPT_PWM_CBD_Event_Output*:**

当计数器事件为 CBD 时,PWMX 管脚输出配置,可配置:

EPT_PWM_CBD_Event_OutHigh

EPT_PWM_CBD_Event_OutLow

EPT_PWM_CBD_Event_Nochange

EPT_PWM_CBD_Event_Negate。

***EPT_PWM_T1U_Event_Output*:**

当计数器事件为 T1U 时,PWMX 管脚输出配置,可配置:

EPT_PWM_T1U_Event_OutHigh

EPT_PWM_T1U_Event_OutLow

EPT_PWM_T1U_Event_Nochange

EPT_PWM_T1U_Event_Negate

***EPT_PWM_T1D_Event_Output*:**

当计数器事件为 T1D 时,PWMX 管脚输出配置,可配置:

EPT_PWM_T1D_Event_OutHigh

EPT_PWM_T1D_Event_OutLow

EPT_PWM_T1D_Event_Nochange

EPT_PWM_T1D_Event_Negate。

***EPT_PWM_T2U_Event_Output*:**

当计数器事件为 T2U 时,PWMX 管脚输出配置,可配置:

EPT_PWM_T2U_Event_OutHigh

EPT_PWM_T2U_Event_OutLow

EPT_PWM_T2U_Event_Nochange

EPT_PWM_T2U_Event_Negate。

***EPT_PWM_T2D_Event_Output*:**

当计数器事件为 T2D 时,PWMX 管脚输出配置,可配置:

EPT_PWM_T2D_Event_OutHigh

EPT_PWM_T2D_Event_OutLow

EPT_PWM_T2D_Event_Nochange

EPT_PWM_T2D_Event_Negate

EPT_PRDR_CMPA_CMPB_CMPC_CMPD_Config

(U16_T ***EPT_PRDR_Value,*** U16_T ***EPT_CMPA_Value,*** U16_T ***EPT_CMPB_Value,*** U16_T ***EPT_CMPC_Value ,*** U16_T ***EPT_CMPD_Value*)**

***EPT_PRDR_Value*:**当前周期设定，可配置:0~0Xffff。

***EPT_CMPA_Value*:**当前周期设定，可配置:0~0Xffff。

***EPT_CMPB_Value*:**当前周期设定，可配置:0~0Xffff。

***EPT_CMPC_Value*:**当前周期设定，可配置:0~0Xffff。

***EPT_CMPD_Value*:**当前周期设定，可配置:0~0Xffff。

EPT_DBCR_Config(

EPT_CHX_Selecte_Type ***EPT_CHX_Selecte,***

EPT_INSEL_Type ***EPT_INSEL_X,***

EPT_OUTSEL_Type ***EPT_OUTSEL_X,***

EPT_OUT_POLARITY_Type ***EPT_OUT_POLARITY_X,***

EPT_OUT_SWAP_Type ***EPT_OUT_SWAP_X*)**

***EPT_CHX_Selecte*:**

CHA 通道选择,可配置:

EPT_CHA_Selecte

EPT_CHB_Selecte

EPT_CHC_Selecte

***EPT_INSEL_X*:**

CHA/ CHB/CHC 输入源 PWMX 选择

EPT_CHA_Selecte 可配置:

EPT_CHAINSEL_PWMA_RISE_FALL

EPT_CHAINSEL_PWMB_RISE_PWMA_FALL

EPT_CHAINSEL_PWMA_RISE_PWMB_FALL

EPT_CHAINSEL_PWMB_RISE_FALL

EPT_CHB_Selecte 可配置:

EPT_CHBINSEL_PWMB_RISE_FALL

EPT_CHBINSEL_PWMC_RISE_PWMB_FALL

EPT_CHBINSEL_PWMB_RISE_PWMC_FALL

EPT_CHBINSEL_PWMC_RISE_FALL

EPT_CHC_Selecte 可配置:

EPT_CHCINSEL_PWMC_RISE_FALL

EPT_CHCINSEL_PWMD_RISE_PWMC_FALL

EPT_CHCINSEL_PWMC_RISE_PWMD_FALL

EPT_CHCINSEL_PWMD_RISE_FALL

***EPT_OUTSEL_X*:**

CHA/CHB/CHC 输出死区配置使能

EPT_CHA_Selecte 可配置:

EPT_CHA_OUTSEL_PWMA_PWMB_Bypass

EPT_CHA_OUTSEL_DisRise_EnFall

EPT_CHA_OUTSEL_EnRise_DisFall

EPT_CHA_OUTSEL_EnRise_EnFall

EPT_CHB_Selecte 可配置:

EPT_CHB_OUTSEL_PWMB_PWMC_Bypass/

EPT_CHB_OUTSEL_DisRise_EnFall

EPT_CHB_OUTSEL_EnRise_DisFall

EPT_CHB_OUTSEL_EnRise_EnFall

EPT_CHC_Selecte 可配置:

EPT_CHC_OUTSEL_PWMC_PWMD_Bypass

EPT_CHC_OUTSEL_DisRise_EnFall

EPT_CHC_OUTSEL_EnRise_DisFall

EPT_CHC_OUTSEL_EnRise_EnFall

***EPT_OUT_POLARITY_X*:**

CHAX/CHBX/CHCX CHAY/CHBY/CHCY 输出状态翻转设定

EPT_CHA_Selecte 可配置:

EPT_PA_PB_OUT_Direct(CHAX、CHAY 直接输出)

EPT_PA_OUT_Reverse（CHAX 输出反向，CHAY 保持不变）

EPT_PB_OUT_Reverse（CHAY 输出反向，CHAX 保持不变）

EPT_PA_PB_OUT_Reverse (CHAX、CHAY 输出全反向)

EPT_CHB_Selecte 可配置:

EPT_PA_PB_OUT_Direct(CHBX、CHBY 直接输出)

EPT_PA_OUT_Reverse（CHBX 输出反向，CHBY 保持不变）

EPT_PB_OUT_Reverse（CHBY 输出反向，CHBX 保持不变）

EPT_PA_PB_OUT_Reverse (CHBX、CHBY 输出全反向)

EPT_CHC_Selecte 可配置:

EPT_PA_PB_OUT_Direct(CHCX、CHCY 直接输出)

EPT_PA_OUT_Reverse（CHCX 输出反向，CHCY 保持不变）

EPT_PB_OUT_Reverse（CHCY 输出反向，CHCX 保持不变）

EPT_PA_PB_OUT_Reverse (CHCX、CHCY 输出全反向)

EPT_OUT_SWAP_X:

CHAX/CHBX/CHCX CHAY/CHBY/CHCY 输出状态互换设定

EPT_CHA_Selecte 可配置:

EPT_PAtoCHX_PBtoCHY(CHAX、CHAY 直接输出)

EPT_PBtoCHX_PBtoCHY(CHAX、CHAY 同时输出 CHAY)/

EPT_PAtoCHX_PAtoCHY(CHAX、CHAY 同时输出 CHAX)
EPT_PBtoCHX_PAtoCHY(CHAX、CHAY 输出互换)
EPT_CHB_Selecte 可配置:
EPT_PAtoCHX_PBtoCHY(CHBX、CHBY 直接输出)
EPT_PBtoCHX_PBtoCHY(CHBX、CHBY 同时输出 CHBY)
EPT_PAtoCHX_PAtoCHY(CHBX、CHBY 同时输出 CHBX)
EPT_PBtoCHX_PAtoCHY(CHBX、CHBY 输出互换)
EPT_CHC_Selecte 可配置:
EPT_PAtoCHX_PBtoCHY(CHCX、CHCY 直接输出)
EPT_PBtoCHX_PBtoCHY(CHCX、CHCY 同时输出 CHCY)
EPT_PAtoCHX_PAtoCHY(CHCX、CHCY 同时输出 CHCX)
EPT_PBtoCHX_PAtoCHY(CHCX、CHCY 输出互换)

4.7.5 EPT 不同计数模式的波形输出

- 递增模式：可以配置产生非对称的 PWM 波形。波形更新的触发点为 CNT=Zero。当 CMP 值由 0 到 PRDR+1 进行调整时，可以获得 0 到 100%的 PWM 占空比输出（如果需要获得 100%占空比，需要设置 CMP 值>PRDR，在此设置下，将不会发生 CAU 或者 CBU 触发事件）。
- 递减模式：可以配置产生非对称的 PWM 波形。波形更新的触发点为 CNT=Period。当 CMP 值由 PRDR 到 0 进行调整时，可以获得 0 到几乎 100%的 PWM 占空比输出，由于在递减模式下，CMP 比较值不能设置为比 0 更小的数值。但是当比较值设置为 0 时，PWM 在周期结束前，会输出一个 CLK 宽度的脉冲。**在需要完全 100%PWM 占空比输出的情下，需要选择递增计数模式，或者递增递减计数模式**。
- 递增递减模式：可以配置产生非对称或者对称的 PWM 波形。通常情况下，在递增和递减阶段使用同一个比较值对输出进行处理，可以输出一个对称的 PWM 波形。在对称输出时，当比较值配置为零时，可以得到 100%占空比输出的 PWM 对称波形。随着比较值的增大，输出波形的占空比逐渐缩小。当比较值等于 PRDR-1 时，可以获得最小非零占空比输出的波形。当比较值设为等于或者大于 PRDR 时，输出的 PWM 波形占空比为零。在计数器递增递减模式下，配置为非对称 PWM 波形输出时，可以通过设置递增阶段的 CA 和递减阶段的 CB 两个比较点产生非对称的 PWM 波形输出

4.8 I2C 通讯模块

4.8.1 I2C 说明

- 主机或者从机工作模式，支持多主机总线
- 串行，8 位的双向数据传输
- 两种速度：
  - 标准模式 (0 到 100Kbits/s)
  - 快速模式 (<=400Kbit/s) 或者超快速模式 (<=1000Kbit/s)
- 7 位或者 10 位寻址方式
- 7 位或者 10 位地址组合传输
- 从机发送模式下支持大量传输模式
- 支持发送和接收缓冲 (FIFO)
- 可编程的 SDA 保持时间
- 支持总线清除功能

4.8.2 I2C 主机通讯配置

➢ **功能实例:**

开启内部主频 48MHz,并作为系统时钟。

设置 SDA(PA0.0)，SCL(PA0.1)且使能漏极开路输出，使能外部上拉 4.7k。

I2C 做主机模式

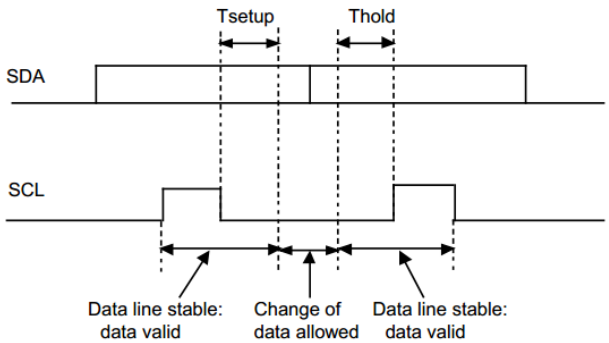
SCL_H=48M*160=3.3uS ， SCL_L=48M*160=3.3uS

e.g: I2C 速度=1/6.6us=151K

I2C setup time=0X40/48M=1.3uS

I2C THold/RHold time=0X40/48M=1.3uS

hold/setup time 如下：



I2C 器件地址设置为 0X56

0X01 地址中写 0XAA

0X01 地址中回读 0XAA

➢ **操作步骤:**

1.SYSCON_CONFIG();函数配置

2.GPIO_CONFIG();函数配置
3.I2C_MASTER_CONFIG();函数配置
4.主函数中对应 0X01 地址进行读写操作。

- **程序范例:**

```
/*************************************************************/
//GPIO Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void GPIO_CONFIG(void)
{
    GPIO_OpenDrain_EN(GPIOA0,1);        //PA0.1 漏极开路输出
    GPIO_OpenDrain_EN(GPIOA0,0);        //PA0.0 漏极开路输出
}

/*************************************************************/
//I2C MASTER Initial
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void I2C_MASTER_CONFIG(void)
{
   I2C_DeInit();                                    //所有寄存器复位赋值
   I2C_Master_CONFIG(I2C_SDA_PA00,I2C_SCL_PA01,FAST_MODE,I2C_MASTRER_7BIT,0X57,0Xa0,0Xa0);
  //管脚配置 SDA(PA0.9)，SCL(PA0.8)，通讯速度配置
   I2C_SDA_TSETUP_THOLD_CONFIG(0x40,0x40,0x40);
            //TSETUP=0X40*PCLK,RX_THOLD=0X40*PCLK,TX_THOLD=0X20*PCLK
   I2C_Enable();
}
/*************************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************************************/
void APT32F102_init(void)
{

    SYSCON->PCER0=0xFFFFFFF;                //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                //使能 IP
    while(!(SYSCON->PCSR0&0x1));            //判断 IP 是否使能

    SYSCON_CONFIG();                        //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                //打开全局中断
```

```
        I2C_MASTER_CONFIG ();
}

U8_T R_i2c_read_data;
/*************************************************************/
//main
/*************************************************************/
int main(void)
{
        APT32F102_init();
        while(1)
        {
                SYSCON_IWDCNT_Reload();                    //清狗
                I2C_WRITE_Byte(0X01,0xaa);
                delay_nms(40);                              //延时 1ms
                R_i2c_read_data= I2C_READ_Byte(0X01);
      }
}
```

4.8.3 I2C 从机通讯配置

- **功能实例:**

  开启内部主频 48MHz,并作为系统时钟。

  设置 SDA(PA0.0)，SCL(PA0.1)且使能漏极开路输出，使能外部上拉 4.7k。

  I2C 做主机模式

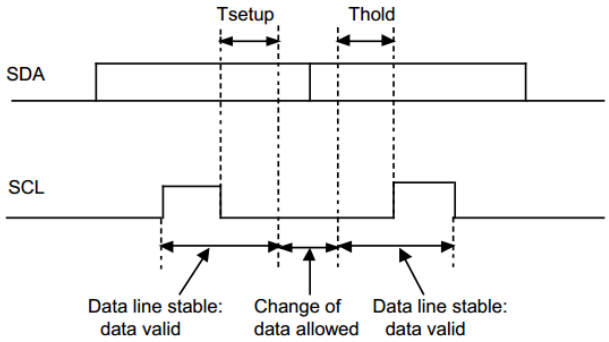
  SCL_H=48M*160=3.3uS ， SCL_L=48M*160=3.3uS

  e.g: I2C 速度=1/6.6us=151K

  I2C setup time=0X40/48M=1.3uS

  I2C THold/RHold time=0X40/48M=1.3uS

  hold/setup time 如下：



  I2C 器件地址设置为 0X56

0X01 地址中写 0XAA
0X01 地址中回读 0XAA

- **操作步骤:**

1.SYSCON_CONFIG();函数配置
2.GPIO_CONFIG();函数配置
3.I2C_SLAVE_CONFIG ();函数配置
4.I2C 中断调用 I2C_Slave_Receive();函数。
5.在主循环中对读取到的数据进行处理。

a) 由主机写入的地址及数据存储在 I2CRdBuffer[BUFSIZE]中，传送给主机的数据存储在 I2CWrBuffer[BUFSIZE]中；
b) 主机写入数据时，所要操作的寄存器值存储在 I2CRdBuffer[BUFSIZE]
eg. 主机向从机寄存器地址 0x1 写入数据 0x55，操作完成后 I2CRdBuffer[1]=0x55；
c) 主机读取值时，读取值寄存器值与 I2CWrBuffer[BUFSIZE]一一对应；
eg.读取数据地址 0x01 的值，及代表读取 I2CWrBuffer[1]的值。

- **程序范例:**

```
/*************************************************************/
//GPIO Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void GPIO_CONFIG(void)
{
    GPIO_OpenDrain_EN(GPIOA0,1);                    //PA0.1 漏极开路输出
    GPIO_OpenDrain_EN(GPIOA0,0);                    //PA0.0 漏极开路输出
}
/*************************************************************/
//I2C SLAVE Initial
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void I2C_SLAVE_CONFIG(void)
{
    I2C_DeInit();
    I2C_Slave_CONFIG(I2C_SDA_PA00,I2C_SCL_PA01,FAST_MODE,I2C_SLAVE_7BIT,0X57,0X50,0X50);
    //从机地址不能设置为 0x00~0x07,0x78~0x7f
    I2C_SDA_TSETUP_THOLD_CONFIG(0x40,0x40,0x40);
    I2C_FIFO_TriggerData(0,0);                      //发送 FIFO 设置为 7
    I2C_ConfigInterrupt_CMD(ENABLE,I2C_SCL_SLOW);
    I2C_ConfigInterrupt_CMD(ENABLE,I2C_STOP_DET);
    I2C_ConfigInterrupt_CMD(ENABLE,I2C_RD_REQ);
    I2C_ConfigInterrupt_CMD(ENABLE,I2C_RX_FULL);
    I2C_ConfigInterrupt_CMD(ENABLE,I2C_TX_ABRT);
```

```
        I2C_Enable();
        I2C_Int_Enable();
 }
/*************************************************************/
//I2C Interrupt
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void I2CIntHandler(void)
{
        I2C_Slave_Receive();
}
/*****************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*****************************************************************************/
void APT32F102_init(void)
{

        SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
        SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
        while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

        SYSCON_CONFIG();                            //syscon 参数 初始化
        CK_CPU_EnAllNormalIrq();                    //打开全局中断

        I2C_SLAVE_CONFIG ();
}
/*************************************************************/
//main
/*************************************************************/
int main(void)
{
        APT32F102_init();
       while(1)
        {
                SYSCON_IWDCNT_Reload();             //清狗
                //I2CWrBuffer[1] 从机写数据
                //I2CRdBuffer[1] 从机读数据
      }
}
```

4.9 SPI 串行外设模块

4.9.1 SPI 主机通讯配置

- **功能实例:**

  开启内部主频 48MHz,并作为系统时钟。

  功能管脚配置：

  SPI_NSS=PB0.5;

  SPI_SCK=PA0.9;

  SPI_MISO=PA0.11;

  SPI_MOSI=PA0.10

  通讯速度：

  FSSPCLKOUT=48M/48=1M

  发送接收模式：

  发送数据大小为 8BIT；SCK 工作时为高电平；SCK 第一个时钟沿捕捉；串行正常输出

- **操作步骤:**

  1.SYSCON_CONFIG();函数配置

  2.SPI_MASTER_CONFIG();函数配置

  3.主循环调用 SPI_WRITE_BYTE();发送数据

- **程序范例:**

```
/*************************************************************/
    //SPI MASTER Initial
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*************************************************************/
    void SPI_MASTER_CONFIG(void)
    {
        SPI_DeInit();
        SPI_NSS_IO_Init(1);
        SPI_Master_Init(SPI_G1,SPI_DATA_SIZE_8BIT,SPI_SPO_0,SPI_SPH_0,SPI_LBM_0,SPI_RXIFLSEL_1_8,0,1
    0);
        //选择 SPI IO group1; 发送数据大小为 8BIT; SCK 工作时为高电平; SCK 第一个时钟沿捕捉; 串行正常输出; 接收
        占用 1/8 FIFO 中断触发断点; FSSPCLKOUT=48M/48=1M
        SPI_ConfigInterrupt_CMD(ENABLE,SPI_RXIM);                    //使能 FIFO 接收中断
        SPI_Int_Enable();                                            //使能 SPI 中断向量
    }

    U32_T R_i2c_read_data;
    /*************************************************************/
    //main
    /*************************************************************/
    int main(void)
    {
```

```
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                    //清狗
        SPI_WRITE_BYTE (0x01);                     //发送 0x01
    }
}
```

4.9.2 SPI 从机通讯配置

- **功能实例:**

  开启内部主频 48MHz,并作为系统时钟。

  功能管脚配置：

  SPI_NSS=PB0.5;

  SPI_SCK=PA0.9;

  SPI_MISO=PA0.11;

  SPI_MOSI=PA0.10

  通讯速度：

  FSSPCLKOUT=48M/12=4M

  使能接收中断

- **操作步骤:**

  1.SYSCON_CONFIG();函数配置

  2.SPI_SLAVE_CONFIG ();函数配置

  3.在 SPI 中断接收程序中接收数据。

- **程序范例:**

```
/*************************************************************/
    //SPI SLAVE Initial
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*************************************************************/
    void SPI_SLAVE_CONFIG(void)
    {
        SPI_DeInit();
        SPI_NSS_IO_Init(1);
        SPI_Slave_Init(SPI_G1,SPI_DATA_SIZE_8BIT,SPI_SPH_0,SPI_RXIFLSEL_1_8,0,12);
        //选择 SPI IO group1; 发送数据大小为 8BIT;接收占用 1/8 FIFO 中断触发断点 ;FSSPCLKOUT=48M/12=4M
        SPI_ConfigInterrupt_CMD(ENABLE,SPI_RXIM);          //使能 FIFO 接收中断
        SPI_Int_Enable();                                  //使能 SPI 中断向量
    }
```

```
volatile unsigned int   SPI_DATA[8];
/*************************************************************/
//SPI Interrupt
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void SPIIntHandler(void)
{
    if((SPI0->MISR&SPI_RXIM)==SPI_RXIM)                    //接收 FIFO 中断,FIFO 占用 1/8,1/4,1/2 中断
    {
        SPI0->ICR = SPI_RXIM;
        /*SPI_DATA[0]=SPI0->DR;
        SPI_DATA[1]=SPI0->DR;
        SPI_DATA[2]=SPI0->DR;
        SPI_DATA[3]=SPI0->DR;
        SPI_DATA[4]=SPI0->DR;
        SPI_DATA[5]=SPI0->DR;
        SPI_DATA[6]=SPI0->DR;
        SPI_DATA[7]=SPI0->DR;
        nop;*/
    }
}
/*************************************************************/
//main
/*************************************************************/
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                    //清狗
    }
}
```

4.10 COUNTA 计数器模块

4.10.1 COUNTA 定时功能

- **功能实例:**
  开启内部主频 48MHz,并作为系统时钟。
  PA0.12 输出占空比为 50%,周期为 2ms 方波。
- **操作步骤:**
  1.SYSCON_CONFIG();函数配置
  2.GPIO_CONFIG();函数配置
  3.COUNTA_CONFIG ();函数配置
  4.PA0.12 输出翻转写在“CNTAIntHandler ();”中断函数中
- **程序范例:**

```
/*************************************************************/
//GPIO Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void GPIO_CONFIG(void)
{
    GPIO_Init(GPIOA0,12,0);                 //PA0.12  输出模式

    GPIO_Set_Value(GPIOA0,12,1);            //PA0.12  输出高
}
/*************************************************************/
//COUNTA Initial
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void COUNTA_CONFIG(void)
{
  COUNT_DeInit();                           //clear all countA Register
  COUNTA_Init(24000,0,Period_H,DIV1,REPEAT_MODE,CARRIER_ON,OSP_LOW); //Data_H=Time/(1/sysclock)
  COUNTA_Config(SW_STROBE,PENDREM_OFF,MATCHREM_OFF,REMSTAT_0,ENVELOPE_0);
    //countA mode set
  COUNTA_Start();                           //countA start
  COUNTA_Int_Enable();                      //countA INT enable
}
/*************************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************************************/
```

```
void APT32F102_init(void)
{

    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                    //打开全局中断

    GPIO_CONFIG ();
    COUNTA_CONFIG();
}
/*************************************************************/
//CONTA Interrupt
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void CNTAIntHandler(void)
{
  // ISR content ...
    if(!f_io_toggle)
    {
        f_io_toggle=1;
        GPIO_Set_Value(GPIOA0,12,1);
    }
    else
    {
        f_io_toggle=0;
        GPIO_Set_Value(GPIOA0,12,0);
    }
}
/*************************************************************/
//main
/*************************************************************/
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                 //清狗
    }
}
```

4.10.2 COUNTA PWM 输出

- **功能实例:**
  开启内部主频 48MHz,并作为系统时钟。
  PB0.1 输出占空比为 50%,周期为 1ms 方波。
- **操作步骤:**
  1.SYSCON_CONFIG();函数配置
  2.COUNTA_CONFIG ();函数配置
- **程序范例:**

```
/*************************************************************/
    //COUNTA Initial
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*************************************************************/
    void COUNTA_CONFIG(void)
    {
        COUNT_DeInit();                                          //clear all countA Register
        COUNTA_IO_Init(COUNTA_PB01);                             //set PA0.07 as counter IO
        COUNTA_Init(24000,12000,Period_H,DIV1,REPEAT_MODE,CARRIER_ON,OSP_LOW);
        //Data_H=Time/(1/sysclock)
        COUNTA_Config(SW_STROBE,PENDREM_OFF,MATCHREM_OFF,REMSTAT_0,ENVELOPE_0);
        //countA mode set
        COUNTA_Start();                                           //countA start
        COUNTA_Int_Enable();                                     //countA INT enable
    }
    /*****************************************************************************/
    //APT32F102_init
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*****************************************************************************/
    void APT32F102_init(void)
    {
        SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
        SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
        while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

        SYSCON_CONFIG();                            //syscon 参数  初始化
        CK_CPU_EnAllNormalIrq();                    //打开全局中断

        GPIO_CONFIG ();
        COUNTA_CONFIG();
    }
    /*************************************************************/
```

```
//main
/************************************************************/
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                    //清狗
    }
}
```

4.11 GPT 增强型通用定时器模块

4.11.1 GPT 说明

- 16 位可复位计数器
- 可编程计数器计数方式
  - 递增计数（Up-counting）
  - 递减计数（Down-counting）
  - 递增递减计数（Up-down-counting）
- 两路波形产生控制单元，支持双路独立输出：
  - 两路独立的 PWM 输出，单边沿工作
  - 两路独立的 PWM 输出，双边沿对称工作
- 通过软件异步重置 PWM 的波形输出
- 支持片间多设备同步
  - 支持多个 TIMER 间的同步触发
  - 触发源包括 GPIO 输入，其他外设触发，软件设置和事件触发
  - 支持单次触发和连续触发模式
- 支持单脉冲输出模式
- 支持突发计数模式
- 支持通过外部时钟计数
- 支持事件计数器，可通过配置事件计数器（最大 15）触发相应中断
- 支持捕获模式，最多支持 2 个捕获值存储

4.11.2 GPT 两路独立 PWM 输出

➢ **功能实例:**

开启内部主频 48MHz,并作为系统时钟。

计数器单周期时间：CLKS= MCLK /1=48MHz

PA0.9->CHA:

输出周期为 100us，占空比为 50us

PA0.10->CHB:

输出周期为 100us，占空比为 25us

波形如下：

- **操作步骤:**

  1.SYSCON_CONFIG();函数配置

  2.GPT0_CONFIG ();函数配置

- **程序范例:**

```
/*************************************************************/
//GPT0 Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void GPT0_CONFIG(void)
{
    GPT_IO_Init(GPT_CHA_PA09);
    GPT_IO_Init(GPT_CHB_PA010);
    GPT_Configure(GPTCLK_EN,GPT_PCLK,GPT_SHADOW,0);
    GPT_WaveCtrl_Configure(GPT_INCREASE,GPT_SWSYNDIS,GPT_IDLE_LOW,GPT_PRDLD_PEND,G
    PT_OPM_CONTINUOUS,GPT_BURST_DIS,GPT_CKS_PCLK,GPT_CG_CHAX,GPT_CGFLT_00,GPT_
    PRDLD_ZERO);
    GPT_Period_CMP_Write(48000,24000,12000);
    GPT_WaveLoad_Configure(GPT_WAVEA_SHADOW,GPT_WAVEB_SHADOW,GPT_AQLDA_ZERO,GP
    T_AQLDB_ZERO);
    GPT_WaveOut_Configure(GPT_CHA,GPT_CASEL_CMPA,GPT_CBSEL_CMPA,2,0,1,0,0,0,0,0,0);
    GPT_WaveOut_Configure(GPT_CHB,GPT_CASEL_CMPA,GPT_CBSEL_CMPB,2,0,0,0,1,0,0,0,0);
    GPT_Start();
}
/*************************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************************************/
void APT32F102_init(void)
{

    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                    //打开全局中断

    GPT0_CONFIG ();
}
/*************************************************************/
//main
/*************************************************************/
```

```
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                    //清狗
    }
}
```

波形输出配置

```
GPT_WaveOut_Configure(GPT_CHA,GPT_CASEL_CMPA,GPT_CBSEL_CMPA,2,0,1,0,0,0,0,0,0,0);
GPT_WaveOut_Configure(GPT_CHB,GPT_CASEL_CMPA,GPT_CBSEL_CMPB,2,0,0,0,1,0,0,0,0,0);
```

GPT_CHA/ GPT_CHB:选择需要配置的输出为通道 A 或通道 B

GPT_CASEL_CMPA/ GPT_CBSEL_CMPB：选择 CMPA 或 CMPB 寄存器作为比较值的数据源

```
(2,0,1,0,0,0,0,0,0,0)
a,b,c,d,e,f,g,h,i,j
```

函数中从左至右10个数字代表的状态为：

a) 当CNT值等于零时，在通道上做出的波形输出

b) 当CNT值等于PRDR时，在通道上做出的波形输出

c) 当CNT值等于CMPA，且此时计数方向为递增时，在通道上做出的波形输出

d) 当CNT值等于CMPA，且此时计数方向为递减时，在通道上做出的波形输出

e) 当CNT值等于CMPB，且此时计数方向为递增时，在通道上做出的波形输出

f) 当CNT值等于CMPB，且此时计数方向为递增时，在通道上做出的波形输出

g) 当T1事件发生，且此时计数方向为递增时，在通道上做出的波形输出

h) 当T1事件发生，且此时计数方向为递减时，在通道上做出的波形输出

i) 当T2事件发生，且此时计数方向为递增时，在通道上做出的波形输出

j) 当T2事件发生，且此时计数方向为递减时，在通道上做出的波形输出

函数中的 10 个数字数值大小为 0~3 所代表的输出状态如下：

0：保持原来的输出（不动作）

1：清除输出（低电平）

2：置位输出（高电平）

3：反向（翻转）

4.11.3 GPT 输入捕捉

- **功能实例:**

开启内部主频 48MHz,并作为系统时钟。

计数器单周期时间：CLKS= MCLK /1=48MHz

PA0.0 作为捕捉输入口
捕捉波形周期=100uS，占空比=79.2us
R_CMPA_BUF 存储低电平计数值，R_CMPB_BUF 存储周期计数值

- **操作步骤:**
  1. SYSCON_CONFIG();函数配置
  2. GPIO_CONFIG();函数配置
  3. GPT0_CONFIG ();函数配置
  4. ET_CONFIG();函数配置
  5. GPT0IntHandler();读取 R_CMPA_BUF，R_CMPB_BUF 值
- **程序范例:**

```
/*****************************************************************/
    //GPIO Initial
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*****************************************************************/
    void GPIO_CONFIG(void)
    {
        //------------  EXI FUNTION  ---------------------------------/
        //EXI0_INT=  EXI0/EXI16,EXI1_INT=  EXI1/EXI17,  EXI2_INT=EXI2~EXI3/EXI18/EXI19,
    EXI3_INT=EXI4~EXI9, EXI4_INT=EXI10~EXI15
        GPIO_IntGroup_Set(PA0,0,Selete_EXI_PIN0);                    //EXI0 set PBA.0
        GPIOA0_EXI_Init(EXI0);                                       //PA0.0 as input
        EXTI_trigger_CMD(ENABLE,EXI_PIN0,_EXIRT);                    //ENABLE rising edge
        EXTI_interrupt_CMD(ENABLE,EXI_PIN0);                         //enable EXI
        GPIO_EXTI_interrupt(GPIOA0,0b00000000000001);                    //enable GPIOA00 as EXI
    }

    /*****************************************************************/
    //GPT0 Functions
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*****************************************************************/
    void GPT0_CONFIG(void)
    {
        GPT_Configure(GPTCLK_EN,GPT_PCLK,GPT_SHADOW,0);
        GPT_Capture_Config(EPT_CNTMD_increase,GPT_CAPMD_Continue,GPT_CAP_EN,GPT_LDARST_E
        N,GPT_LDBRST_DIS,GPT_LDCRST_DIS,GPT_LDDRST_DIS,1);
        GPT0->SYNCR=0X04;
        GPT_Period_CMP_Write(0xffff,0,0);
        GPT_ConfigInterrupt_CMD(ENABLE,GPT_INT_CAPLD0);// CMPA 载入中断
        GPT_ConfigInterrupt_CMD(ENABLE,GPT_INT_CAPLD1);// CMPB 载入中断
        GPT_INT_ENABLE();
        GPT_Start();
```

```
}
/*************************************************************/
//syscon Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void SYSCON_CONFIG(void)
{
    SYSCON_RST_VALUE();
    SYSCON_General_CMD(ENABLE,ENDIS_ISOSC);
    SYSCON_HFOSC_SELECTE(HFOSC_SELECTE_48M);
    SystemCLK_HCLKDIV_PCLKDIV_Config(SYSCLK_HFOSC,HCLK_DIV_1,PCLK_DIV_1,HFOSC_48M);
    SYSCON_IWDCNT_Config(IWDT_TIME_1S,IWDT_INTW_DIV_7);
    SYSCON_WDT_CMD(ENABLE);
    SYSCON_IWDCNT_Reload();
    SYSCON->EVTRG=0X00|0x01<<20;
    SYSCON->EVPS=0X00;
}
/*************************************************************/
//ET Initial
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void ET_CONFIG(void)
{
    ET_DeInit()
    ET_CH0_SRCSEL(ET_SRC0,ENABLE,ET_EXI_SYNC0);
    ET_CH0_CONTROL(ENABLE,TRG_HW, ET_GPT0_TRGSRC2);
    ET_ENABLE();
}
/*****************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*****************************************************************************/
void APT32F102_init(void)
{
    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                    //打开全局中断
```

```
    GPIO_CONFIG();                                  //GPIO initial
    GPT0_CONFIG ();                                 //GPT0 initial
    ET_CONFIG();
}

/*************************************************************/
//main
/*************************************************************/
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                     //清狗
    }
}
/*************************************************************/
//GPT0 Interrupt
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void GPT0IntHandler(void)
{
    if((GPT0->MISR&GPT_INT_CAPLD0)==GPT_INT_CAPLD0)
    {
        GPT0->ICR = GPT_INT_CAPLD0;
        EXTI_trigger_CMD(DISABLE,EXI_PIN0,_EXIRT);
        EXTI_trigger_CMD(ENABLE,EXI_PIN0,_EXIFT);
        R_CMPA_BUF=GPT0->CMPA;                      //低电平
    }
    else if((GPT0->MISR&GPT_INT_CAPLD1)==GPT_INT_CAPLD1)
    {
        GPT0->ICR = GPT_INT_CAPLD1;
        EXTI_trigger_CMD(ENABLE,EXI_PIN0,_EXIRT);
        EXTI_trigger_CMD(DISABLE,EXI_PIN0,_EXIFT);
        R_CMPB_BUF=GPT0->CMPB;                      //周期计数值
    }
}
```

4.12 LPT 低功耗定时器模块

4.12.1 LPT 说明

- 16 位递增计数器
- 4 Bit 预分频控制，支持（1，2，4，8，16，32，64，128，256，512，1024，2048，4096 分频）
- 支持多种计数时钟：
  - 内部时钟：ISCLK, IMCLK, EMCLK 或 PCLK
  - 外部时钟：LPT_IN（当没有内部时钟时，可以作为脉冲计数）
  - 一路独立的 PWM 输出
- 一个比较值寄存器
- 支持连续或单次计数模式
- 支持通过 ETCB 触发
- 支持脉冲和 PWM 输出模式

4.12.2 LPT 定时

- **功能实例:**
  开启内部主频 48MHz,并作为系统时钟。
  PA0.12 输出占空比为 50%,周期为 2ms 方波。
- **操作步骤:**
  1. SYSCON_CONFIG();函数配置
  2. LPT_CONFIG ();函数配置
  3. IO 翻转放在中断函数 LPTIntHandler();中
- **程序范例:**

```
/*************************************************************/
//GPIO Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void GPIO_CONFIG(void)
{
    GPIO_Init(GPIOA0,12,0);                 //PA0.12  输出模式

    GPIO_Set_Value(GPIOA0,12,1);            //PA0.12  输出高
}
/*************************************************************/
//LPT Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
```

```
void LPT_CONFIG(void)
{
    LPT_DeInit();                                          //LPT DeInit
    LPT_Configure(LPTCLK_EN,LPT_PCLK_DIV4,LPT_IMMEDIATE,LPT_PSC_DIV0,0,LPT_OPM_CONTINUOU);
    LPT_Period_CMP_Write(12000,0);
    LPT_ConfigInterrupt_CMD(ENABLE,LPT_PEND);
    LPT_Start();
    LPT_INT_ENABLE();
}
/*******************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*******************************************************************************/
void APT32F102_init(void)
{

    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数  初始化
    CK_CPU_EnAllNormalIrq();                    //打开全局中断

    GPIO_CONFIG();
    LPT_CONFIG( ();
}
/*************************************************************/
//LPT Interrupt
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void LPTIntHandler(void)
{
    // ISR content ...
    //Interupt
    if((LPT->MISR&LPT_PEND)==LPT_PEND)
    {
        LPT->ICR = LPT_PEND;
        if(!f_io_toggle)
        {
            f_io_toggle=1;
            GPIO_Set_Value(GPIOA0,12,1);
        }
```

```
            else
            {
                f_io_toggle=0;
                GPIO_Set_Value(GPIOA0,12,0);
            }
        }
    }
    /*************************************************************/
    //main
    /*************************************************************/
    int main(void)
    {
        APT32F102_init();
        while(1)
        {
            SYSCON_IWDCNT_Reload();                    //清狗
        }
    }
```

4.12.3 LPT PWM 输出

- **功能实例:**
  开启内部主频 48MHz,并作为系统时钟。
  PB0.1 输出占空比为 50%,周期为 1ms 方波。
- **操作步骤:**
  1. SYSCON_CONFIG();函数配置
  2. LPT_CONFIG ();函数配置

- **程序范例:**

```
    /*************************************************************/
    //LPT Functions
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*************************************************************/
    void LPT_CONFIG(void)
    {
        LPT_DeInit();                                  //LPT DeInit
        LPT_IO_Init(LPT_OUT_PB01);                     //PA0.9 as LPT out
        //LPT_IO_Init(LPT_IN_PA10);                    //PA0.10 as LPT in
        LPT_Configure(LPTCLK_EN,LPT_PCLK_DIV4,LPT_IMMEDIATE,LPT_PSC_DIV0,0,LPT_OPM_CONTIN
    UOU);
        //LPT_Debug_Mode(ENABLE);
```

```
    LPT_ControlSet_Configure(LPT_SWSYNDIS,LPT_IDLE_Z,LPT_PRDLD_DUTY_END,LPT_POL_HIGH,L
    PT_FLTDEB_00,LPT_PSCLD_0,LPT_CMPLD_IMMEDIATELY);
    //LPT_SyncSet_Configure(LPT_TRGEN_EN,LPT_OSTMD_ONCE,LPT_AREARM_DIS);
    LPT_Period_CMP_Write(12000,6000);
    LPT_Start();
}
/*****************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*****************************************************************************/
void APT32F102_init(void)
{

    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数  初始化
    CK_CPU_EnAllNormalIrq();                    //打开全局中断

    LPT_CONFIG( ();
}
/*****************************************************************/
//main
/*****************************************************************/
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                 //清狗
    }
}
```

4.13 BT 基本计数器模块

4.13.1 BT 说明

- 16 位可编程递增计数器。
- 16 位预设计数器时钟分频器 (支持 On-the-fly 修改配置)。
- 一个比较值寄存器，支持 PWM 波形输出。
- 支持通过 ETCB 进行硬件自动同步触发和外部计数。

4.13.2 BT 定时

- **功能实例:**
  开启内部主频 48MHz,并作为系统时钟。
  PA0.12 输出占空比为 50%,周期为 2ms 方波。
- **操作步骤:**
  1. SYSCON_CONFIG();函数配置
  2. BT_CONFIG ();函数配置
  3. IO 翻转放在中断函数 BTIntHandler();中
- **程序范例:**

```
/************************************************************/
//GPIO Functions
//EntryParameter:NONE
//ReturnValue:NONE
/************************************************************/
void GPIO_CONFIG(void)
{
    GPIO_Init(GPIOA0,12,0);                 //PA0.12  输出模式

    GPIO_Set_Value(GPIOA0,12,1);            //PA0.12  输出高
}
/************************************************************/
//BT Initial
//EntryParameter:NONE
//ReturnValue:NONE
/************************************************************/
void BT_CONFIG(void)
{
    BT_DeInit(BT0);
    BT_Configure(BT0,BTCLK_EN,0,BT_IMMEDIATE,BT_CONTINUOUS,BT_PCLKDIV);//TCLK=PCLK/(0+1)

    BT_Period_CMP_Write(BT0,48000,0);
    BT_Start(BT0);
```

```
    BT_ConfigInterrupt_CMD(BT0,ENABLE,BT_PEND);
    BT0_INT_ENABLE();
}
/*******************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*******************************************************************************/
void APT32F102_init(void)
{

    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                    //打开全局中断

    GPIO_CONFIG();
    BT_CONFIG ( );
}

/*************************************************************/
//main
/*************************************************************/
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                 //清狗
    }
}
/*************************************************************/
//BT0 Interrupt
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void BT0IntHandler(void)
{
  // ISR content ...
    if((BT0->MISR&BT_PEND)==BT_PEND)
    {
        BT0->ICR = BT_PEND;
```

```
            if(!f_io_toggle)
            {
                f_io_toggle=1;
                GPIO_Set_Value(GPIOA0,12,1);
            }
            else
            {
                f_io_toggle=0;
                GPIO_Set_Value(GPIOA0,12,0);
            }
        }
    }
```

4.13.3 BT PWM 输出

- **功能实例:**
  开启内部主频 48MHz,并作为系统时钟。
  PB0.2 输出占空比为 50%,周期为 1ms 方波。
- **操作步骤:**
  1. SYSCON_CONFIG();函数配置
  2. BT_CONFIG ();函数配置
- **程序范例:**

```
/*************************************************************/
//BT Initial
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void BT_CONFIG(void)
{
    BT_DeInit(BT0);
    BT_IO_Init(BT0_PB02);
    BT_Configure(BT0,BTCLK_EN,0,BT_IMMEDIATE,BT_CONTINUOUS,BT_PCLKDIV);//TCLK=PCLK/(0+1)
    BT_ControlSet_Configure(BT0,BT_START_HIGH,BT_IDLE_LOW,BT_SYNC_DIS,BT_SYNCMD_DIS,BT_OST
    MDX_ONCE,BT_AREARM_DIS,BT_CNTRLD_EN);
    BT_Period_CMP_Write(BT0,48000,24000);
    BT_Start(BT0);
}
/*************************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
```

```
/*******************************************************************************/
void APT32F102_init(void)
{

    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                    //打开全局中断

    BT_CONFIG ( );
}
/*****************************************************************/
//main
/*****************************************************************/
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                 //清狗
    }
}
```

4.14 CRC 模块

4.14.1 CRC 多项式

CRC-CCITT: x16 + x12 + x5 + 1
CRC-16: x16 + x15 + x2 + 1
CRC-32: x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1

4.14.2 CRC 参数

XORIN：输入异或使能与禁止
XOROUT：输出异或使能与禁止
REFIN：输入数据按位反转使能与禁止
REFOUT：输出数据按位反转使能与禁止
SEED：CRC 种子寄存器，每次 CRC 计算前都需操作一次

4.14.3 CR 程序范例

- **CRC CCITT 32bit Input**

```
//CRC 输入数据
U32_T twords_ccitt[6] = {0x01020304,0X05060708,0X3409F0CE};
//CRC 标准计算结果
U32_T expect_value_ccitt=0x1DDC;

CRC_Soft_Reset();
CRC_CMD(ENABLE);
CRC_Configure(XORIN_DIS,XOROUT_DIS,REFIN_EN,REFOUT_EN,POLY_CCITT);
CRC_Seed_Write(0x00000000);
calc_result_ccitt= Chip_CRC_CRC32(twords_ccitt, 3); //输入数据个数为3 个

//运行结果
calc_result_ccitt=0x1DDC;
```

- **CRC Poly16 32bit Input**

```
//CRC 输入数据
U32_T twords_poly16[6] = {0x01020304,0X05060708,0X3409F0CE};
```

```
//CRC 标准计算结果
U32_T expect_value_poly16=0x5B69;

CRC_Soft_Reset();
CRC_CMD(ENABLE);
CRC_Configure(XORIN_DIS,XOROUT_DIS,REFIN_EN,REFOUT_EN,POLY_16);
CRC_Seed_Write(0x00000000);
calc_result_poly16= Chip_CRC_CRC32(twords_poly16, 3); //输入数据个数为3 个

//运行结果
calc_result_poly16=0x5B69;
```

- **CRC Poly32 32bit Input**

```
//CRC 输入数据
U32_T twords_poly32[6] =
{0x01020304,0X05060708,0X3409F0CE,0X2767A0BB,0X1FC87200,0XEC78900A};
//CRC 标准计算结果
U32_T expect_value_poly32=0xB5AEAED3;

CRC_Soft_Reset();
CRC_CMD(ENABLE);
CRC_Configure(XORIN_DIS,XOROUT_EN,REFIN_EN,REFOUT_EN,POLY_32);
CRC_Seed_Write(0xFFFFFFFF);
calc_result_poly32= Chip_CRC_CRC32(twords_poly32, 6); //输入数据个数为6 个

//运行结果
calc_result_poly32= 0xB5AEAED3;
```

- **CRC CCITT 8bit Input**

```
//CRC 输入数据
U8_T twords_ccitt8[6] = {0x01,0x02,0x03,0x04,0X05};
//CRC 标准计算结果
U32_T expect_value_ccitt8=0xED9B;


U8_T CRC_TEST_CCITT8(void)
{
    CRC_Soft_Reset();
    CRC_CMD(ENABLE);
    CRC_Configure(XORIN_DIS,XOROUT_DIS,REFIN_EN,REFOUT_EN,POLY_CCITT);
```

```
        CRC_Seed_Write(0x00000000);
        calc_result_ccitt8= Chip_CRC_CRC8(twords_ccitt8, 5); //输入数据个数为5 个
        if(calc_result_ccitt8==expect_value_ccitt8)
        {
            nop;
            return TRUE;
        }
        else return FALSE;
    }

    //运行结果
    calc_result_ccitt8= 0XED9B;
```

- **CRC CCITT 16bit Input**

```
    //CRC 输入数据
    U16_T twords_ccitt16[6] = {0x0102,0x0203,0x0304,0x0405,0X0506};
    //CRC 标准计算结果
    U32_T expect_value_ccitt16=0xD5FF;

    U8_T CRC_TEST_CCITT16(void)
    {
        CRC_Soft_Reset();
        CRC_CMD(ENABLE);
        CRC_Configure(XORIN_DIS,XOROUT_DIS,REFIN_EN,REFOUT_EN,POLY_CCITT);
        CRC_Seed_Write(0x00000000);
        calc_result_ccitt16= Chip_CRC_CRC16(twords_ccitt16, 4); //输入数据个数为4 个
        if(calc_result_ccitt16==expect_value_ccitt16)
        {
            nop;
            return TRUE;
        }
        else return FALSE;
    }
    //运行结果
   calc_result_ccitt8= 0XD5FF;
```

➢ **CRC 运行结果显示**

Expression	Value
expect_value_poly32	0xb5aeaed3
calc_result_poly32	0xb5aeaed3
expect_value_ccitt	0x00001ddc
calc_result_ccitt	0x00001ddc
expect_value_poly16	0x00005b69
calc_result_poly16	0x00005b69
expect_value_ccitt8	0x0000ed9b
calc_result_ccitt8	0x0000ed9b
expect_value_ccitt16	0x0000d5ff
calc_result_ccitt16	0x0000d5ff

4.14.3 CRC 说明

1. 输入数据的格式需要与函数匹配，如 8Bit 的数据需要按 8Bit 函数格式输入；
2. CRC 计算结果，可以参考网站 http://www.ip33.com/crc.html 上标准计算结果，计算时需选择与程序相同的多项式格式

4.15 SIO 串行输入输出模块

4.15.1 SIO 初始化

```
SIO_IO_Init(SIO_PA012);// 选择 PA0.12 所为 SIO 输出
SIO_TX_Init(SIOCLK_EN,39);//使能 SIO 时钟，FTXCLK = FPCLK / (TCKPRS +1) 5.556M/(39+1)=138.9K
SIO_TX_Configure(SIO_IDLE_HIGH,SIO_TX_MSB,15,8,0,0,SIO_OBH_3BIT,SIO_OBL_3BIT,3,1);
//Idle 状态输出 High，发送方向为 MSB→LSB，发送 buffer 长度为 15+1，发送序列长度为 8+1，D0，D1 长度为 1
个 T，DH，DL 长度为 3bit，DH 对象序列为 3，DL 序列对象为 1
```

4.15.2 SIO 发送配置

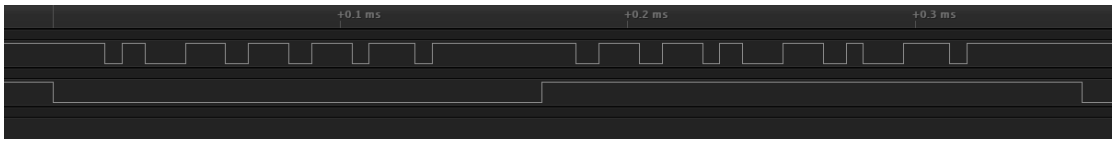
```
if(SIO0->RISR&0X01)    //RISR bit1==1 时表示发送完成
        {
        SIO0->ICR=0X01;
        SIO_TXBUF_Set(TX_D1,TX_D0,TX_DL,TX_D1,TX_D1,TX_D0,TX_DH,TX_DH,TX_DH,TX_D0,TX_D1,T
X_D0,TX_D1,TX_D0,TX_D1,TX_D0);
        GPIO_Set_Value(GPIOA0,4,0);
        }

        if(SIO0->RISR&0X01) //RISR bit1==1 时表示发送完成
        {
        SIO0->ICR=0X01;
        SIO_TXBUF_Set(TX_D0,TX_D1,TX_D1,TX_D0,TX_DH,TX_DL,TX_DH,TX_DL,TX_DH,TX_D0,TX_D1,T
X_D0,TX_D1,TX_D0,TX_D1,TX_D0);
        GPIO_Set_Value(GPIOA0,4,1);
        }
```

4.15.4 SIO 输出配置方式

D0：全 0 序列
D1：全 1 序列
DL：由序列长度和序列数据控制，如序列长度设置为 SIO_OBL_3BIT，序列数据设置为 1，表示将序列数据 0b00000001，由 bit0~bit7 方向发送共 3bit。第 1 个 bit 为高，其余两个 bit 为低.即 4.11.4 图中 DL 波形
DH：由序列长度和序列数据控制，如序列长度设置为 SIO_OBH_3BIT，序列数据设置为 3，表示将序列数据 0b00000011，由 bit0~bit7 方向发送共 3bit。第 1,2 个 bit 为高，第三个 bit 为低. 即 4.11.4 图中 DH 波形

4.15.4 SIO 波形输出

连续波形



测试口 PA0.4 输出低时的波形

测试口 PA0.4 输出高时的波形

4.16 ETCB 事件触发控制器模块

4.16.1 ETCB 说明

使用 ETCB 可以利用事件触发控制器收到某一 IP 的某个事件后，触发另一 IP 的相应动作。使用者在使用时，需要配置事件源和目标事件，同时事件源和目标事件也需要分别配置成事件触发和同步触发输入使能。

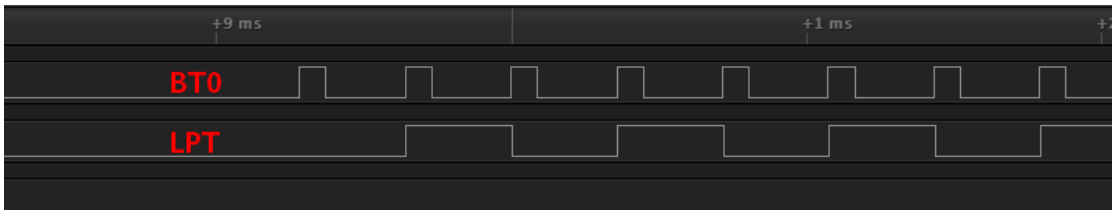
4.16.1 ETCB 范例

- **BT0 触发 LPT**

```
//以下程序实现 BT0 输出 PEND 状态时触发 LPT 输出波形
BT_DeInit(BT0);
BT_IO_Init(BT_PB02);
BT_Configure(BT0,BTCLK_EN,0,BT_IMMEDIATE,BT_CONTINUOUS,BT_PCLKDIV);//TCLK=PCLK/(0+1)
BT_ControlSet_Configure(BT0,BT_START_HIGH,BT_IDLE_LOW,BT_SYNC_DIS,BT_SYNCMD_DIS,BT_OST
MDX_ONCE,BT_AREARM_DIS,BT_CNTRLD_EN);
BT_Trigger_Configure(BT0,BT_TRGSRC_PEND,BT_TRGOE_EN);
BT_Period_CMP_Write(BT0,2000,500);
BT_Start(BT0);
//BT_ConfigInterrupt_CMD(BT0,ENABLE,BT_PEND);
//BT0_INT_ENABLE();

LPT_DeInit();                                   //LPT DeInit
LPT_IO_Init(LPT_OUT_PB01);                      //PA0.9 as LPT out
//LPT_IO_Init(LPT_IN_PA10);                     //PA0.10 as LPT in
LPT_Configure(LPTCLK_EN,LPT_PCLK_DIV4,LPT_IMMEDIATE,LPT_PSC_DIV0,0,LPT_OPM_CONTINUOUS
);          //
//LPT_Debug_Mode(ENABLE);
LPT_ControlSet_Configure(LPT_SWSYNDIS,LPT_IDLE_Z,LPT_PRDLD_DUTY_END,LPT_POL_HIGH,LPT_FL
TDEB_00,LPT_PSCLD_0,LPT_CMPLD_IMMEDIATELY);
LPT_SyncSet_Configure(LPT_TRGEN_EN,LPT_OSTMD_ONCE,LPT_AREARM_DIS);
//LPT_Trigger_Configure(LPT_SRCSEL_EN,LPT_BLKINV_DIS,LPT_CROSSMD_DIS,LPT_TRGSRC0_CMP,L
PT_ESYN0OE_EN,5,5,0X0F);
LPT_Period_CMP_Write(1000,500);
LPT_ConfigInterrupt_CMD(ENABLE,LPT_PEND);
//LPT_Start();
//LPT_INT_ENABLE();

ET_DeInit();
```

```
    //BT0 触发 LPT
    ET_CH0_SRCSEL(ET_SRC0,ENABLE,ET_BT_SYNC0);
    ET_CH0_CONTROL(ENABLE,TRG_HW,ET_LPT_TRGSRC);
    ET_ENABLE();
```

输出波形



➢ LPT 触发 BT0

```
    //以下程序实现 LPT 输出在满足 15 次 PEND 状态后触发 BT0 输出波形
    BT_DeInit(BT0);
    BT_IO_Init(BT_PB02);
    BT_Configure(BT0,BTCLK_EN,0,BT_IMMEDIATE,BT_CONTINUOUS,BT_PCLKDIV);//TCLK=PCLK/(0+1)
    BT_ControlSet_Configure(BT0,BT_START_HIGH,BT_IDLE_LOW,BT_SYNC_EN,BT_SYNCMD_EN,BT_OSTM
DX_ONCE,BT_AREARM_DIS,BT_CNTRLD_EN);
    BT_Trigger_Configure(BT0,BT_TRGSRC_PEND,BT_TRGOE_DIS);
    BT_Period_CMP_Write(BT0,2000,500);
    //BT_Start(BT0);
    //BT_ConfigInterrupt_CMD(BT0,ENABLE,BT_PEND);
    //BT0_INT_ENABLE();

    LPT_DeInit();                                   //LPT DeInit
    LPT_IO_Init(LPT_OUT_PB01);                      //PA0.9 as LPT out
    //LPT_IO_Init(LPT_IN_PA10);                     //PA0.10 as LPT in
    LPT_Configure(LPTCLK_EN,LPT_PCLK_DIV4,LPT_IMMEDIATE,LPT_PSC_DIV0,0,LPT_OPM_CONTINUOUS
);          //
    //LPT_Debug_Mode(ENABLE);
    LPT_ControlSet_Configure(LPT_SWSYNDIS,LPT_IDLE_Z,LPT_PRDLD_DUTY_END,LPT_POL_HIGH,LPT_FL
TDEB_00,LPT_PSCLD_0,LPT_CMPLD_IMMEDIATELY);
    //LPT_SyncSet_Configure(LPT_TRGEN_EN,LPT_OSTMD_ONCE,LPT_AREARM_DIS);
    LPT_Trigger_Configure(LPT_SRCSEL_EN,LPT_BLKINV_DIS,LPT_CROSSMD_DIS,LPT_TRGSRC0_CMP,LP
T_ESYN0OE_EN,5,5,0X0F);
    LPT_Period_CMP_Write(1000,500);
    LPT_ConfigInterrupt_CMD(ENABLE,LPT_PEND);
    LPT_Start();
    //LPT_INT_ENABLE();

    //LPT 触发 BT0
    ET_CH0_SRCSEL(ET_SRC0,ENABLE,ET_LPT_SYNC);
    ET_CH0_CONTROL(ENABLE,TRG_HW,ET_BT0_TRGSRC0);
```

```
    ET_ENABLE();
```

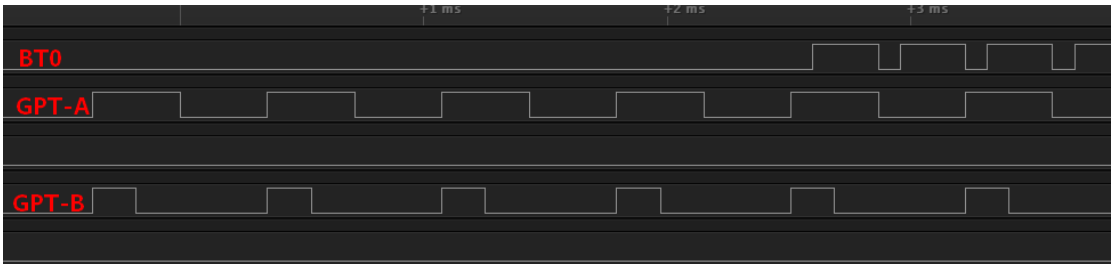
输出波形

➢ GPT0 触发 BT0

```
    //以下程序实现 GPT0 输出在满足 3 次 PRD 状态后触发 BT0 输出波形
     BT_DeInit(BT0);
    BT_IO_Init(BT_PB02);
    BT_Configure(BT0,BTCLK_EN,0,BT_IMMEDIATE,BT_CONTINUOUS,BT_PCLKDIV);//TCLK=PCLK/(0+1)
    BT_ControlSet_Configure(BT0,BT_START_HIGH,BT_IDLE_LOW,BT_SYNC_EN,BT_SYNCMD_EN,BT_OSTMDX_ONCE,BT_AREARM_DIS,BT_CNTRLD_EN);
    BT_Trigger_Configure(BT0,BT_TRGSRC_PEND,BT_TRGOE_DIS);
    BT_Period_CMP_Write(BT0,2000,500);
    //BT_Start(BT0);
    //BT_ConfigInterrupt_CMD(BT0,ENABLE,BT_PEND);
    //BT0_INT_ENABLE();

    GPT_IO_Init(GPT_CHA_PA09);
    GPT_IO_Init(GPT_CHB_PA010);
    GPT_Configure(GPTCLK_EN,GPT_PCLK,GPT_IMMEDIATE,1);
    GPT_WaveCtrl_Configure(GPT_INCREASE,GPT_SWSYNDIS,GPT_IDLE_LOW,GPT_PRDLD_PEND,GPT_OPM_CONTINUOUS,GPT_BURST_DIS,GPT_CKS_PCLK,GPT_CG_CHAX,GPT_CGFLT_00,GPT_PRDLD_ZERO);
    GPT_Period_CMP_Write(2000,1000,500);
    GPT_WaveLoad_Configure(GPT_WAVEA_IMMEDIATE,GPT_WAVEB_SHADOW,GPT_AQLDA_ZERO,GPT_AQLDB_ZERO);
    GPT_WaveOut_Configure(GPT_CHA,GPT_CASEL_CMPA,GPT_CBSEL_CMPA,2,0,1,1,0,0,0,0,0,0);
    GPT_WaveOut_Configure(GPT_CHB,GPT_CASEL_CMPA,GPT_CBSEL_CMPB,2,0,0,0,1,1,0,0,0,0);
    //GPT_SyncSet_Configure(GPT_SYNCUSR0_EN,GPT_OST_CONTINUOUS,GPT_TXREARM_DIS,GPT_TRGO0SEL_SR0,GPT_TRG10SEL_SR0,GPT_AREARM_DIS);
    GPT_Trigger_Configure(GPT_SRCSEL_TRGUSR0EN,GPT_BLKINV_DIS,GPT_ALIGNMD_PRD,GPT_CROSSMD_DIS,5,5);
    GPT_EVTRG_Configure(GPT_TRGSRC0_PRD,GPT_TRGSRC1_PRD,GPT_ESYN0OE_EN,GPT_ESYN1OE_EN,GPT_CNT0INIT_EN,GPT_CNT1INIT_EN,3,3,3,3);
    GPT_Start();
    //GPT_ConfigInterrupt_CMD(ENABLE,GPT_INT_PEND);
    //GPT_INT_ENABLE();
    //INTC_ISER_WRITE(GPT0_INT);
    //INTC_IWER_WRITE(GPT0_INT);
```

```
//GTP0 SYNC0  触发 LPT
ET_CH0_SRCSEL(ET_SRC0,ENABLE,ET_GPT0_SYNC0);
ET_CH0_CONTROL(ENABLE,TRG_HW,ET_BT0_TRGSRC0);
ET_ENABLE();
```

输出波形

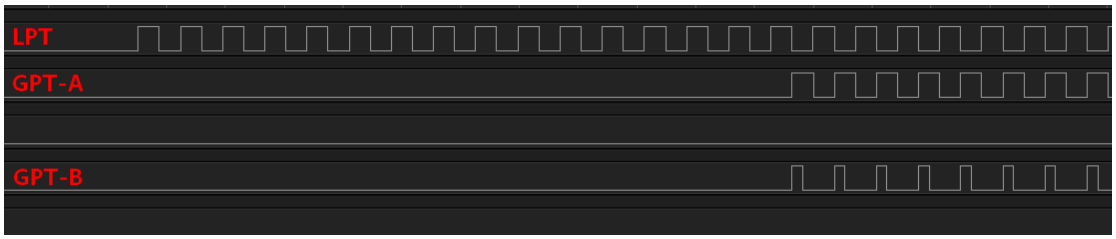


➢ LPT 触发 GPT0

```
//以下程序实现 LPT 输出在满足 15 次 PEND 状态后触发 GPT0 输出波形
        LPT_DeInit();                                   //LPT DeInit
        LPT_IO_Init(LPT_OUT_PB01);                      //PA0.9 as LPT out
        //LPT_IO_Init(LPT_IN_PA10);                     //PA0.10 as LPT in
        LPT_Configure(LPTCLK_EN,LPT_PCLK_DIV4,LPT_IMMEDIATE,LPT_PSC_DIV0,0,LPT_OPM_CONTINUOUS);        //
        //LPT_Debug_Mode(ENABLE);
        LPT_ControlSet_Configure(LPT_SWSYNDIS,LPT_IDLE_Z,LPT_PRDLD_DUTY_END,LPT_POL_HIGH,LPT_FLTDEB_00,LPT_PSCLD_0,LPT_CMPLD_IMMEDIATELY);
        //LPT_SyncSet_Configure(LPT_TRGEN_EN,LPT_OSTMD_ONCE,LPT_AREARM_DIS);
        LPT_Trigger_Configure(LPT_SRCSEL_EN,LPT_BLKINV_DIS,LPT_CROSSMD_DIS,LPT_TRGSRC0_CMP,LPT_ESYN0OE_EN,5,5,0X0F);
        LPT_Period_CMP_Write(1000,500);
        LPT_ConfigInterrupt_CMD(ENABLE,LPT_PEND);
        LPT_Start();
        //LPT_INT_ENABLE();

        GPT_IO_Init(GPT_CHA_PA09);
        GPT_IO_Init(GPT_CHB_PA010);
        GPT_Configure(GPTCLK_EN,GPT_PCLK,GPT_IMMEDIATE,1);
        GPT_WaveCtrl_Configure(GPT_INCREASE,GPT_SWSYNDIS,GPT_IDLE_LOW,GPT_PRDLD_PEND,GPT_OPM_CONTINUOUS,GPT_BURST_DIS,GPT_CKS_PCLK,GPT_CG_CHAX,GPT_CGFLT_00,GPT_PRDLD_ZERO);
        GPT_Period_CMP_Write(2000,1000,500);
        GPT_WaveLoad_Configure(GPT_WAVEA_IMMEDIATE,GPT_WAVEB_SHADOW,GPT_AQLDA_ZERO,GPT_AQLDB_ZERO);
        GPT_WaveOut_Configure(GPT_CHA,GPT_CASEL_CMPA,GPT_CBSEL_CMPA,2,0,1,1,0,0,0,0,0,0);
        GPT_WaveOut_Configure(GPT_CHB,GPT_CASEL_CMPA,GPT_CBSEL_CMPB,2,0,0,0,1,1,0,0,0,0);
```

```
GPT_SyncSet_Configure(GPT_SYNCUSR0_EN,GPT_OST_CONTINUOUS,GPT_TXREARM_DIS,GPT_TRGO0SEL_SR0,GPT_TRG10SEL_SR0,GPT_AREARM_DIS);
//GPT_Trigger_Configure(GPT_SRCSEL_TRGUSR0EN,GPT_BLKINV_DIS,GPT_ALIGNMD_PRD,GPT_CROSSMD_DIS,5,5);
//GPT_EVTRG_Configure(GPT_TRGSRC0_PRD,GPT_TRGSRC1_PRD,GPT_ESYN0OE_EN,GPT_ESYN1OE_EN,GPT_CNT0INIT_EN,GPT_CNT1INIT_EN,3,3,3,3);
//GPT_Start();
//GPT_ConfigInterrupt_CMD(ENABLE,GPT_INT_PEND);
//GPT_INT_ENABLE();
//INTC_ISER_WRITE(GPT0_INT);
//INTC_IWER_WRITE(GPT0_INT);

//LPT 触发 GTP0 SYNC0
ET_CH0_SRCSEL(ET_SRC0,ENABLE,ET_LPT_SYNC);
ET_CH0_CONTROL(ENABLE,TRG_HW,ET_GPT0_TRGSRC0);
ET_ENABLE();
```

输出波形

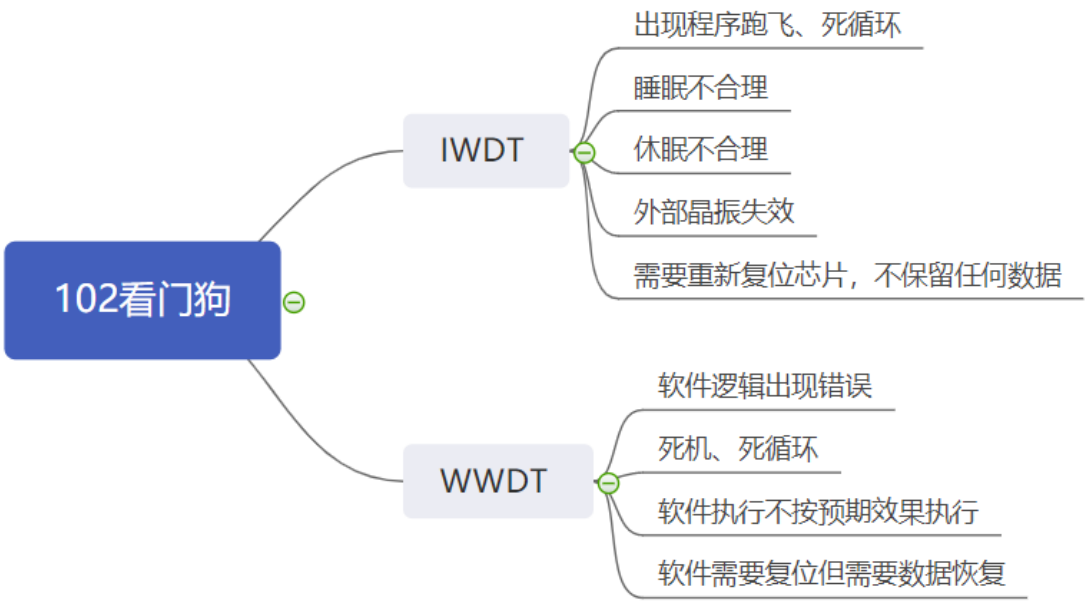


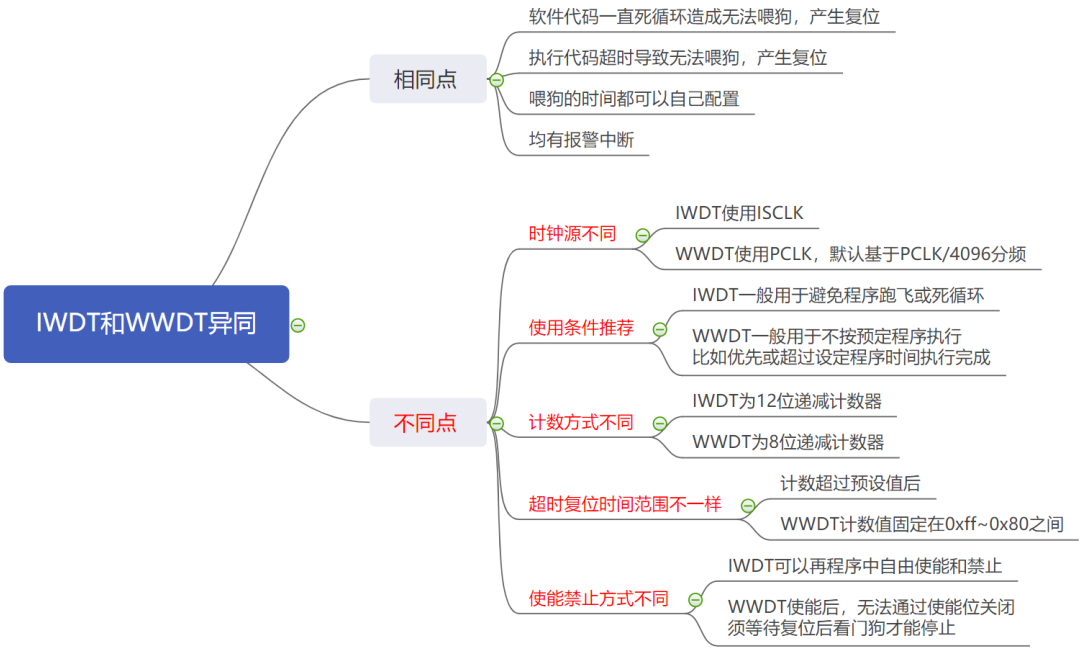
4.17 WWDT 窗口型看门狗模块

4.17.1 WWDT 说明

窗口型看门狗（WWDT）作为可靠性保护逻辑，用于监测当前程序运行状况。当外部干扰或不可预见的逻辑错误发生时，造成当前程序运行错误，看门狗逻辑可以在预设时间周期结束时产生系统复位信号。看门狗计数器可以通过软件刷新以防止计数器溢出而产生复位，如果刷新事件发生在计数器值大于预设窗口计数值时，也将会触发复位信号。也就是刷新必须在预设的时间窗口内进行才有效。

4.17.2 WWDT 与 IWDT 异同





4.17.3 WWDT 配置

```
WWDT_CNT_Load(0xFF);
WWDT_CONFIG(PCLK_4096_DIV0,0xFF,WWDT_DBGDIS);
WWDT_Int_Config(ENABLE);
WWDT_CMD(ENABLE);
```

溢出时间计算（@24MHz）：

公式：$T_{WWDG} = \frac{4096 \times 2^{PSC} \times (CNT[6:0] + 1)}{PCLK}$

实际计算：$T_{WWDG} = \frac{4096 \times 2^{0} \times (0x7F + 1)}{24000000} = 0.021845s \approx 21.85ms$

WWDT 清狗函数：*WWDT_CNT_Load(0xff);*

4.18 RTC 实时时钟计数器模块

4.18.1 RTC 说明

- 支持万年历功能，自动闰年判定。
- 计时器包括小时、分钟、秒和微秒
- BCD 格式计数。
- 二十四小时或者十二小时制可选，支持星期判断。
- 支持多个时钟源，包括外部晶振、内部低速振荡器和内部主振荡器。
- 支持低功耗唤醒功能
- 一个可编程闹钟 。
- 支持周期事件触发

4.18.2 RTC 定时

- **功能实例:**
  开启内部主频 48MHz,并作为系统时钟。
  PA0.12 输出占空比为 50%,周期为 2s 方波。
- **操作步骤:**
  1. SYSCON_CONFIG();函数配置
  2. RTC_CONFIG();函数配置
  3. IO 翻转放在中断函数 RTCIntHandler();中
- **程序范例:**

```
/*************************************************************/
//GPIO Functions
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void GPIO_CONFIG(void)
{
    GPIO_Init(GPIOA0,12,0);              //PA0.12  输出模式
    GPIO_Set_Value(GPIOA0,12,1);         //PA0.12  输出高
}
/*************************************************************/
//RTC Initial
//EntryParameter:NONE
//ReturnValue:NONE
/*************************************************************/
void RTC_CONFIG(void)
{
    RTC_RST_VALUE();
    RTC_Stop();
```

```
    RTCCLK_CONFIG(2777,124,CLKSRC_IMOSC_4div); //5.556M/4/4/(2777+1)/(124+1)=1S
    RTC_Function_Config(RTC_24H,CPRD_1S,COSEL_NoCali_1hz);//Enalbe AlarmA , Enalbe AlarmB , RTC
        Select 24h , CPRD Select1s
    RTC_Int_Enable(CPRD_INT);
    RTC_Vector_Int_Enable();
    RTC_Start();
}
/*******************************************************************************/
//APT32F102_init
//EntryParameter:NONE
//ReturnValue:NONE
/*******************************************************************************/
void APT32F102_init(void)
{
    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能

    SYSCON_CONFIG();                            //syscon 参数 初始化
    CK_CPU_EnAllNormalIrq();                    //打开全局中断

    GPIO_CONFIG();
    RTC_CONFIG ( );
}
/*************************************************************/
//main
/*************************************************************/
int main(void)
{
    APT32F102_init();
    while(1)
    {
        SYSCON_IWDCNT_Reload();                 //清狗
    }
}
```

- **代码说明:**

RTC 可以选择外部晶振/内部副振/内部主振作为输入时钟源，计算公式如下：

RTCCLK= (CLKSRC_EMOSC/4)/(DIVS+1)/(DIVA+1)/4

(ISCLK)/(DIVS+1)/(DIVA+1)/4

(IMCLK/4)/(DIVS+1)/(DIVA+1)/4

(IMCLK)/(DIVS+1)/(DIVA+1)/4

RTCCLK_CONFIG(U16_T ***DIVS*** **,** **U**16_T ***DIVA*** **,** RTC_CLKSRC_TypeDef ***CLKSRC_X)***

DIVS:表示分频参数

可配置 0~0x7fff

DIVA:表示分频参数

可配置 0~0x7f

CLKSRC_X：时钟源选择

可配置 CLKSRC_ISOSC/ CLKSRC_IMOSC_4div/ CLKSRC_EMOSC/ CLKSRC_EMOSC_4div

RTC_Function_Config(RTC_FMT_MODE_TypeDef ***RTC_FMT_MODE*** **,** RTC_CPRD_TypeDef ***RTC_CPRD_x*** **,** Rtc_ClockOutput_Mode_TypeDef ***Rtc_ClockOutput_x*****)**

RTC_FMT_MODE:时间制式配置

可配置 RTC_24H/ RTC_12H

RTC_CPRD_x:CPD 周期配置

可配置 CPRD_05S（0.5s）/ CPRD_1S/ CPRD_1MIN/ CPRD_1HOUR/ CPRD_1DAY/ CPRD_1MONTH

4.18.3 RTC 计时&闹钟报警

- **功能实例:**

开启内部主频 48MHz,并作为系统时钟。
通过串口 UART0 打印 RTC 时钟数据
初始时间设定：20 年 5 月 29 日 11 时 09 分 00 秒
闹钟报警设定：20 时 01 分 00 秒
串口波特率:115200
PA0.1->RXD0
PA0.0->TXD0

- **操作步骤:**

1. SYSCON_CONFIG();函数配置
2. RTC_CONFIG();函数配置
3. 串口打印数据
4. 设置 RTC ALRA 中断 **(必须增加 ALRA 中断和中断里面的配置程序，解决 24H 制小时异常问题)**

- **程序范例:**

```
/************************************************************/
//RTC Initial
//EntryParameter:NONE
//ReturnValue:NONE
/************************************************************/
void RTC_CONFIG(void)
{
    RTC_RST_VALUE();
    RTC_Stop();
    RTCCLK_CONFIG(2777,124,CLKSRC_IMOSC_4div);
//5.556M/4/4/(2777+1)/(124+1)=1S    RTC_Function_Config(RTC_24H,CPRD_1S,COSEL_NoCali_1hz);//Enalbe AlarmA , Enalbe AlarmB , RTC Select 24h , CPRD Select 1s
    RTC_TimeDate_buf.u8Second=0x00;
    RTC_TimeDate_buf.u8Minute=0x09;
```

```
        RTC_TimeDate_buf.u8Hour=0X11;                                   //24 制
        RTC_TimeDate_buf.u8Day=0X29;
        RTC_TimeDate_buf.u8Month=0X05;
        RTC_TimeDate_buf.u8Year=0X20;
        RTC_TimeDate_buf.u8DayOfWeek=0x04;
        RTC_TIMR_DATR_SET(&RTC_TimeDate_buf);   //20 年 5 月 29 日 11 时 09 分 00 秒
        RTC_AlarmA_buf.u8Second=0x59;           //必须打开 ALARMA 设置该时间用于调整小时进位错误问题
        RTC_AlarmA_buf.u8Minute=0x59;
        RTC_AlarmA_buf.u8Hour=0X09;
    RTC_Alarm_TIMR_DATR_SET(Alarm_A,&RTC_AlarmA_buf,Alarm_Second_Compare_EN,Alarm_Minute_Compare_
EN,Alarm_Hour_Compare_EN,Alarm_DataOrWeek_Compare_DIS,Alarm_data_selecte);
    RTC_AlarmB_buf.u8Second=0x00;
    RTC_AlarmB_buf.u8Minute=0x01;
    RTC_AlarmB_buf.u8Hour=0X20;
    RTC_AlarmB_buf.u8WeekOrData=0X22;
    RTC_Alarm_TIMR_DATR_SET(Alarm_B,&RTC_AlarmB_buf,Alarm_Second_Compare_EN,Alarm_Minute_Compare_
EN,Alarm_Hour_Compare_EN,Alarm_DataOrWeek_Compare_DIS,Alarm_data_selecte);//闹钟时间 20 时 01 分 00 秒,每月 22
号
        RTC_Int_Enable(ALRA_INT);
        RTC_Int_Enable(ALRB_INT);
        RTC_Vector_Int_Enable();
        RTC_Start();
    }
    /*************************************************************/
    //UART0   CONFIG
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*************************************************************/
    void UART0_CONFIG(void)
    {
        UART0_DeInit();                                         //clear all UART Register
        UART_IO_Init(IO_UART0,0);                               //use PA0.1->RXD0, PA0.0->TXD0
        UARTInitRxTxIntEn(UART0,416,UART_PAR_NONE);             //baudrate=sysclock/416=115200
    }
    /*****************************************************************************/
    //APT32F102_init
    //EntryParameter:NONE
    //ReturnValue:NONE
    /*****************************************************************************/
    void APT32F102_init(void)
    {
    SYSCON->PCER0=0xFFFFFFF;                    //使能 IP
    SYSCON->PCER1=0xFFFFFFF;                    //使能 IP
    while(!(SYSCON->PCSR0&0x1));                //判断 IP 是否使能
```

```
SYSCON_CONFIG();                                //syscon 参数 初始化
CK_CPU_EnAllNormalIrq();                        //打开全局中断

RTC_CONFIG ( );
UART0_CONFIG();
}
/*************************************************************/
//main
/*************************************************************/
int main(void)
{
APT32F102_init();
while(1)
{
SYSCON_IWDCNT_Reload();                         //清狗
RTC_TIMR_DATR_Read(&RTC_TimeDate_buf);
if(RTC_TimeDate_buf.u8Second!=R_RTC_BUF[0])
{
    R_RTC_BUF[0]=RTC_TimeDate_buf.u8Second;
    UARTTxByte(UART0,0X0a);
    UARTTxByte(UART0,RTC_TimeDate_buf.u8Hour);
    UARTTxByte(UART0,RTC_TimeDate_buf.u8Minute);
    UARTTxByte(UART0,RTC_TimeDate_buf.u8Second);
}
}
```

- **代码说明:**

RTC 可以选择外部晶振/内部副振/内部主振作为输入时钟源，计算公式如下：

RTCCLK= (CLKSRC_EMOSC/4)/(DIVS+1)/(DIVA+1)/4

(ISCLK)/(DIVS+1)/(DIVA+1)/4

(IMCLK/4)/(DIVS+1)/(DIVA+1)/4

(IMCLK)/(DIVS+1)/(DIVA+1)/4

RTCCLK_CONFIG(U16_T ***DIVS*** **,** **U**16_T ***DIVA*** **,** RTC_CLKSRC_TypeDef ***CLKSRC_X*****)**

DIVS:表示分频参数

可配置 0~0x7fff

DIVA:表示分频参数

可配置 0~0x7f

CLKSRC_X:时钟源选择

可配置 CLKSRC_ISOSC/ CLKSRC_IMOSC_4div/ CLKSRC_EMOSC/ CLKSRC_EMOSC_4div

RTC_Function_Config(RTC_FMT_MODE_TypeDef ***RTC_FMT_MODE*** **,** RTC_CPRD_TypeDef ***RTC_CPRD_x*** **,** Rtc_ClockOutput_Mode_Type Def ***Rtc_ClockOutput_x)***

RTC_FMT_MODE:时间制式配置

可配置 RTC_24H/ RTC_12H

RTC_CPRD_x:CPD 周期配置

可配置 CPRD_05S（0.5s）/ CPRD_1S/ CPRD_1MIN/ CPRD_1HOUR/ CPRD_1DAY/ CPRD_1MONTH

RTC_TIMR_DATR_SET(RTC_time_t *RTC_TimeDate)

***RTC_TimeDate:**表示 RTC 所有数据参数的结构体指针，

内部成员包括:u8Hour/u8Minute/u8Second/u8DayOfWeek/u8Year/u8Month/u8Day

RTC_Alarm_TIMR_DATR_SET(

RTC_Alarm_Register_select_Type Def ***Alarm_x,***

RTC_Alarmset_T ****RTC_AlarmA,***

RTC_Alarm_Second_mask_TypeDef ***RTC_Alarm_Second_x,***

RTC_Alarm_Minute_mask_TypeDef ***RTC_Alarm_Minute_x,***

RTC_Alarm_Hour_mask_TypeDef ***RTC_Alarm_Hour_x,***

RTC_Alarm_DataOrWeek_mask_TypeDef ***RTC_Alarm_DataOrWeek_x,***

RTC_Alarm_WeekData_select_TypeDef ***Alarm_x_selecte***

)

Alarm_x:AlarmA 或 B 选择

可以配置 Alarm_A/Alarm_B

***RTC_AlarmA:**RTC 所有数据的结构体指针

内部成员包括 u8Hour/u8Minute/u8Second/u8DayOfWeek/u8Year/u8Month/u8Day

RTC_Alarm_Second_x: Alarm Second 是否作为比较使能

可配置 Alarm_Second_EN/Alarm_Second_DIS

RTC_Alarm_Minute_x: Alarm Minute 是否作为比较使能

可配置 Alarm_Minute_EN/Alarm_Minute_DIS

RTC_Alarm_Hour_x: Alarm Hour 是否作为比较使能

可配置 Alarm_Hour_EN/Alarm_Hour_DIS

RTC_Alarm_DataOrWeek_x: Alarm DataOrWeek 是否作为比较使能

可配置 Alarm_DataOrWeek_EN/Alarm_DataOrWeek_DIS

Alarm_x_selecte:配置 data 或者 week 选择

可配置 Alarm_data_selecte/Alarm_week_selecte

4.19 IFC 闪存控制器模块

4.19.1 IFC DROM 地址对应表

DROM 大小一共 2K Bytes，可靠性支持 100000 次擦写。地址由 0x10000000~0x100007FF

DROM Page	起始地址
DROM_PageAdd0	0x10000000
DROM_PageAdd1	0x10000040
DROM_PageAdd2	0x10000080
DROM_PageAdd3	0x100000C0
DROM_PageAdd4	0x10000100
DROM_PageAdd5	0x10000140
DROM_PageAdd6	0x10000180
DROM_PageAdd7	0x100001C0
DROM_PageAdd8	0x10000200
DROM_PageAdd9	0x10000240
DROM_PageAdd10	0x10000280
DROM_PageAdd11	0x100002C0
DROM_PageAdd12	0x10000300
DROM_PageAdd13	0x10000340
DROM_PageAdd14	0x10000380
DROM_PageAdd15	0x100003C0
DROM_PageAdd16	0x10000400
DROM_PageAdd17	0x10000440
DROM_PageAdd18	0x10000480
DROM_PageAdd19	0x100004C0
DROM_PageAdd20	0x10000500
DROM_PageAdd21	0x10000540
DROM_PageAdd22	0x10000580
DROM_PageAdd23	0x100005C0
DROM_PageAdd24	0x10000600
DROM_PageAdd25	0x10000640
DROM_PageAdd26	0x10000680
DROM_PageAdd27	0x100006C0
DROM_PageAdd28	0x10000700
DROM_PageAdd29	0x10000740
DROM_PageAdd30	0x10000780
DROM_PageAdd32	0x100007C0

4.19.2 IFC 写数据

写入数据，目标地址的闪存必须先进行页擦除

擦除要写的 Page→写数据

```
PageErase(DROM_PageAdd0);                                    //擦除 page0
PageErase(DROM_PageAdd1);                                    //擦除 page1
Page_ProgramData(0x10000000,8,&FlashDataArry0[0]);//写数据 DROM 一次最多写 64BYTE
Page_ProgramData(0x10000040,8,&FlashDataArry1[0]);    //写数据 DROM 一次最多写 64BYTE
```

注意事项：

- 写数据时，起始地址必须是 4 的倍数
- 在同一页中写数据时，若起始地址并非该页的开始地址，则该地址前面的数据会被擦除掉

4.19.3 IFC 读数据

```
ReadDataArry_U8(0x10000000,8,&DataBuffer0[0]);          //读 ifc page0 内数据
ReadDataArry_U8(0x10000040,8,&DataBuffer1[0]);          //读 ifc page1 内数据
```

4.20 TOUCH KEY 电容式触摸模块

4.20.1 TOUCH KEY 说明

- 最大支持 17 通道按键检测
- 单个按键支持灵敏度单独修改
- 支持 Single key, Multi key 两种按键模式，Multi key 模式支持限制最大按键个数
- 支持低功耗睡眠唤醒模式
- 支持参考电源 VDD 和 FVR 两种模式
- 内置触摸算法库，有中断扫描和主循环扫描两种版本
- 滤波倍数可灵活设置
- 按键更新速度可调
- 长按强制校准时间可配置为 16s 的倍数

4.20.2 TOUCH KEY 配置

使用触摸功能时，首先需要在 apt32f102_initial.c 打开 *TK_CONFIG();*函数，
开启触摸功能后，默认的库文件会占用 Coret 定时模块，用户不能在程序中配置 Coret。
然后在 apt32f102_tkey_parameter.c 文件中对 Touch Key 的参数进行配置。
以下为常用参数的说明：

➢ 常用配置参数：

- **Tkey IO 使能**

  bit=1 表示使能对应的 TCHx 做 touch key 功能，低位至高位的顺序对应 TCH0~TCH16

  TK_IO_ENABLE=0B00011111111000000;

  或者

  TK_IO_ENABLE=TCH_EN(4)|TCH_EN(5)|TCH_EN(6)|TCH_EN(7);

- **Tkey 通道扫描周期配置。**

  值越大灵敏度越高，但不能超过理论值否则按键无法扫描通过，常用值不大于 150*

  TK_senprd [i]=50；　　//i=0~16

- **Tkey 通道触发门槛值配置。**

  值越大门槛值越高，取值范围为按键差值的 50%~60%，未使用的通道设置成 0xFF

  TK_Triggerlevel[i]=60;　　//i=0~16

- **Tkey 触发去抖配置**

  Press_debounce_data=5;　　//按下去抖 1~10，默认配置为 5

- **Tkey 释放去抖配置**

  Release_debounce_data=5;　　//释放去抖 1~10，默认配置为 5

- **Tkey 按键模式**

  0 表示单键模式，1 表示多键模式

  Key_mode=1;　　//0=single key 1=multi key

- **OFFSET 滤波倍数**

  大于等于 4 时，表示开启相应的倍数滤波；小于 4 时表示倍数滤波关闭；默认配置关闭

MultiTimes_Filter=0; *//倍数滤波>=4*

- **最多有效按键个数**

  此配置表示允许同时按下按键时最多有效个数。默认为 4

  Valid_Key_Num=4; *//最大按下有效个数*

- **Baseline 更新速度**

  数值越小，baseline 更新速度越快；数值越大，baseline 更新速度越慢；默认为 10 约 100ms

  Base_Speed=10;

- **按键长按强制更新时间设置**

  长按键强制更新配置。时间= *TK_longpress_time*1s*；默认 16 秒

  TK_longpress_time=16;

- **按键扫描基准时间配置**

  若系统时钟修改时需要修改此参数，保证基准时间为 10ms；计算公式

  TK_BaseCnt=10ms*PCLK/8-1，默认 59999 数值基于 48MHz

  TK_BaseCnt=59999;

：某些特殊应用时，此数值可能会大于此值

➢ 低功耗配置参数：

- **低功耗模式使能**

  当低功耗模式配置为 ENABLE 时，系统在进入睡眠后，Touch 模块才能实现唤醒

  TK_Lowpower_mode=ENABLE;

- **低功耗模式扫描间隔**

  此处为配置在低功耗模式下，Touch 的扫描间隔，间隔越小，则扫描的频次越大，唤醒的速度也越快；0=20ms/1=50ms /2=100ms /3=150ms /4=200ms,

  TK_Lowpower_level=2;

- **低功耗模式唤醒阈值**

  在低功耗模式下，所有的 Touch 通道拥有相同的唤醒阈值；此值可设置为 TK_Triggerlevel[17] 数组中最小值的 80%左右

  TK_Wakeup_level=50;

  注意：Touch 模块在低功耗下的扫描启动信号由 LPT 提供，所以开启低功耗模式后，将占用 LPT 模块，用户不能再配置 LPT 模块。

➢ 特殊配置参数:

- **触摸参考电压源配置**

  TK_PSEL_AVDD 表示选择 VDD；

  TK_PSEL_FVR 表示选择 FVR(PIN23 需要接 104 电容)

  默认选择 VDD 做参考；选择 FVR 时拥有更好的抗干扰能力

  TK_PSEL_MODE=TK_PSEL_AVDD;

- **FVR 参考电压**

  选择FVR做参考时，需根据实际VDD选择4.096V或2048V两个参考电压点，默认选择为4.096V。

  当选择 TK_PSEL_AVDD 时，此位无效。选择的 FVR 参考电压必须比 VDD 电压小 500mV 以上

  TK_FVR_LEVEL=TK_FVR_4096V;

- **C0 充放电电压选择**

  C0 充放电电压选择，默认配置 3V，可配置 1V/2v/3/v/3.6v，当 FVR 为 2.048V 时强制选择 1V。

选择的 C0 充放电电压必须比 VDD 电压小 500mV 以上

TK_EC_LEVEL=TK_EC_3V;

- **充电电流配置参数。**

  若上电 PAD 寄生电容太大，采样值超出上限，那么触摸扫描出现异常，此时应调高对应通道 ICON 值。ICON 的取值范围为 0~7

  TK_icon [i]=4; *//i=0~16*

4.20.3 TOUCH KEY 低功耗配置

1. 设置 TK_Lowpower_mode 为 ENABLE。使用 Touch Key 睡眠唤醒功能必须使能此参数
2. 配置 TK_Lowpower_level 和 TK_Wakeup_level.
   TK_Lowpower_level 数值越大，睡眠状态下触摸扫描间隔越大，唤醒速度也越慢；
   TK_Wakeup_level 数值越小，唤醒门槛越低，越容易唤醒
3. 在睡眠前和唤醒后需对 Touch 进行配置

```
TK_setup_sleep();                 //touch key 准备进入深度睡眠
PCLK_goto_deepsleep_mode();  //进入深度睡眠
TK_quit_sleep();                  //touch key 退出睡眠
```

4.20.4 TOUCH KEY 数据读取

- **Tkey 按键值**

  有按键触发时，在 single key 模式 Key_Map 寄存器对应的 bit 位置 1。当使能 Multi key 模式时同时按下按键对应的 bit 位会同时置 1.

序号	Key_Map	对应按键
1	00000001b	TCH0 触发
2	00000010b	TCH1 触发
3	00000100b	TCH2 触发
4	00001000b	TCH3 触发
5	00010000b	TCH4 触发
6	00100000b	TCH5 触发
7	01000000b	TCH6 触发
8	10000000b	TCH7 触发
9	100000000b	TCH8 触发
10	1000000000b	TCH9 触发
11	10000000000b	TCH10 触发
12	100000000000b	TCH11 触发
13	1000000000000b	TCH12 触发
14	10000000000000b	TCH13 触发
15	100000000000000b	TCH14 触发
16	1000000000000000b	TCH15 触发
17	10000000000000000b	TCH16 触发

- **Tkey 按键差值**

  offset_data0_abs[17];数组对应 TCH0~TCH16 的按键差值

- **Tkey 按键实时采样值**

  sampling_data0 [17];数组对应 TCH0~TCH16 的按键实时采样值

- **Tkey 按键基准值**

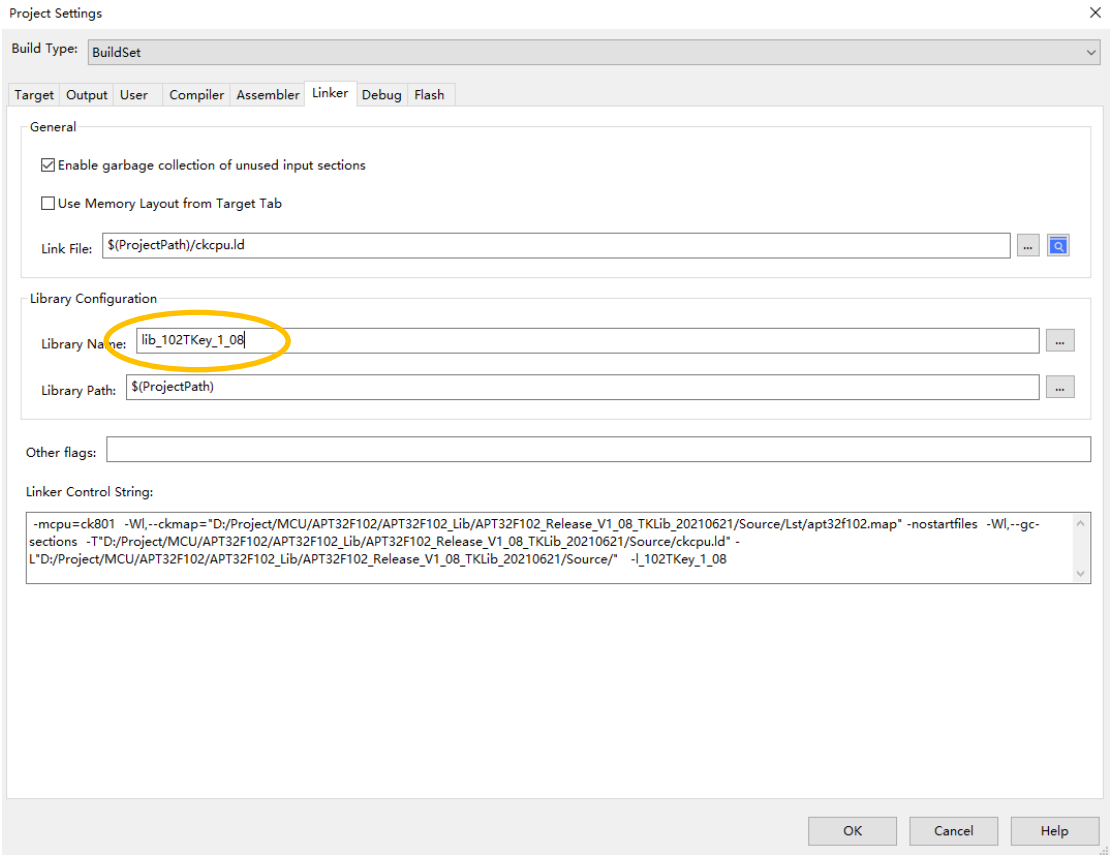
  baseline_data0 [17];数组对应 TCH0~TCH16 的按键基准值

- **Tkey 按键强制更新**

  将 base_update_f=1；可以将触摸按键数据强制更新

4.20.5 TOUCH KEY 库文件版本说明

从 1.08 版本开始，触摸库文件根据客户不同的使用场景，提供不同的版本方便客户使用。客户需要在 Linker 中更改您使用的库文件名称，如下图：



- **Touch Key 中断扫描版本**

lib_102TKey_1_08.a　　触摸库文件完整版(默认库文件)

lib_102TKey_1_08C.a　　触摸库文件精简版，程序占用空间更小，扫描速度更快，抗干扰性能降低，睡眠功耗更低

- **Touch Key 主循环扫描版本**

lib_102TKey_1_08M.a　　触摸库文件主循环扫描完整版，不支持睡眠唤醒

lib_102TKey_1_08M_C.a　　触摸库文件主循环扫描精简版，程序占用空间更小，扫描速度更快，抗干扰性能降低，去除 coret 模块占用，没有长按强制更新功能，不支持睡眠唤醒

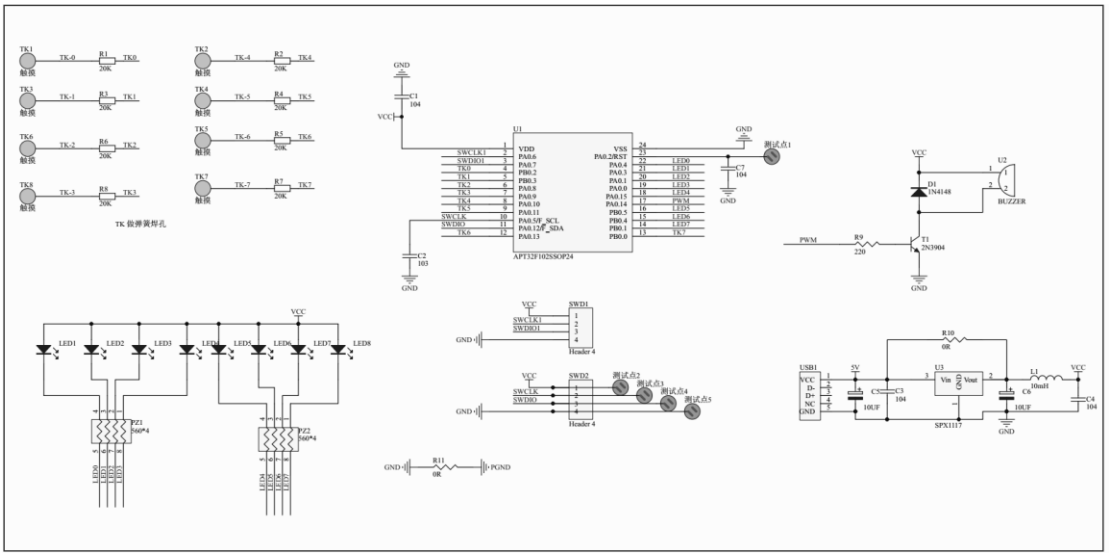
注意：

1. 使用 Touch Key 主循环扫描版本，需要在主循环中添加 tk_prgm();函数，每次执行时间在 1~1.8ms 之间
2. 未使用 coret 功能的版本，需要在 apt32f102_interrupt.c 中重新打开 CORETHandler()入口
3. 中断扫描版本：每一轮的按键扫描时间可控，触摸体验良好；会占用中断资源，如果有高时序要求的中断，没有配置中断好中断优先级的话会影响高时序要求的中断
4. 主循环版本：不会占用中断资源，对别的中断不会有影响；每一轮的按键扫描时间不可控，如果主循环一次循环里有函数占用大量时间，会影响按键的触摸体验

4.20.6 TOUCH KEY 灵敏度调整

用户可以使用发送 Touchkey 按键差值和 Touchkey 按键采样基准值的调试函数，透过 UART TX 发送实时的触摸数据，再利用串口转 USB 工具在 PC 上打印出实时的波形数据。详情参考《AN1511 Touch Key 使用串口工具波形分析使用指南》。

4.20.7 TOUCH KEY 参考原理图



TouchKey C0 电容选择：

使用触摸功能 C0(PA0.5)必须接一个 103 电容。

TouchKey 参考电压选择：

1. **若选择 FVR 做参考 PIN23(PA0.2)需要预留出来接 104 电容。**
2. **若选择 VDD 做参考 PIN23 可以做其他功能。**

4.20.8 TOUCH KEY 使用注意事项

- 使用触摸功能时 **C0(PA0.5)**必须接一个 **103** 电容到地
- 参考电源选择 **FVR** 模式时，**PIN23(PA0.2)**需要预留出来接 **104** 电容到地
- 触摸脚位上串的电阻建议在 **1K~20K**，若需要过高抗干扰测试如 **CS** 等，必须串 **20K** 电阻
- 使用 **FVR** 模式比 **AVDD** 模式拥有更好的抗干扰能力，若需要过高抗干扰测试如 **CS** 等，必须选择 **FVR** 模式，同时 **TK_EC_LEVEL** 选择 **3.6V**
- 低功耗功能只支持 **AVDD** 模式
- 触摸库使用了 **Coret** 模块，客户在使用时不允许对 **Coret** 模块重新进行配置
- 触摸库开启低功耗模式时使用了 **LPT** 模块，客户在开启低功耗功能后不允许对 **LPT** 模块进行配置
- 在使用触摸功能时，配置为触摸功能的脚位不允许在外部拉高或拉低，也不能在程序中再配置为其他功能
- **SENPRD** 是触摸按键的扫描周期，数值越大扫描时间越长，采样值也越大，灵敏度也越高。不建议该值超过 **255**
- **ICON** 是触摸按键的补偿电流，该寄存器数值越小采样值越大，数值越大采样值越小；需要注意的是，当补偿电流值过小时，采样值容易超过上限，可能会造成触摸通道的扫描卡住。当某些应用中，触摸的走线过长时，可能需要将 **ICON** 值调大。
- 当使用者直接按压弹簧等触摸介质时，可能会造成触摸按键的采样值超过扫描上限；可以通过调大 **ICON** 减小采样值或直接按压弹簧时使用轻触的方式
- 相关触摸 **layout** 的指南可参考《**AN1601 APT 触摸按键 Layout Guide**》

5 改版历史

版本	修改日期	修改概要
V1.0	2020-11-04	初版
V1.01	2020-11-18	1. 修改EPT0部分 2. 增加RTC、CRC、SIO、ETCB、WWDT、TOUCH模块
V1.02	2020-11-20	1. 增加了常见错误，修正了一些格式和字体
V1.03	2020-11-24	1. 增加TK模块
V1.04	2020-11-30	1. 修改了部分格式 2. 修改了TK内容
V1.05	2021-01-10	1. 增加了Touch部分低功耗内容 2. 增加了中断优先级配置说明
V1.10	2021-03-26	1. 修改部分内容，增加部分模块描述
V1.11	20210416	1. 修改了触摸低功耗的部分内容
V1.15	20210423	1. 修改范例格式、修改logo 2. 修改了ETC章节的内容 3. 修改了ADC章节的内容 4. 修改部分错误描述 5. 补充了部分函数说明
V1.16	20210601	1. 修改了Touch Key章节电路图 2. 增加了Touch Key章节使用注意事项
V1.18	20210702	1. 增加了Touch Key章节内容 2. 修改了部分模块的信息缺失
V1.20	20210831	1. 增加了部分章节，模块的说明 2. 增加了部分新的功能，如IO-remap 3. 修改了部分章节内容的描述 4. 新增中断扩展，睡眠唤醒描述 5. 删除了原本的常见错误描述章节