

版本	V1.1
日期	202310



# Quick Start

## APT32F173x 系列

版权所有©深圳市爱普特微电子有限公司

本资料内容为深圳市爱普特微电子有限公司在现有数据资料基础上慎重且力求准确无误编制而成，本资料中所记载的实例以正确的使用方法和标准操作为前提，使用方在应用该等实例时请充分考虑外部诸条件，深圳市爱普特微电子有限公司不担保或确认该等实例在使用方的适用性、适当性或完整性，深圳市爱普特微电子有限公司亦不对使用方因使用本资料所有内容而可能或已经带来的风险或后果承担任何法律责任。基于使本资料的内容更加完善等原因，公司保留未经预告的修改权。

## Revision History

版本	日期	描述	作者
V1.0	2023-5-15	新建使用说明	YYM
V1.1	2023-10-27	1.根据最新的代码更新demo 2.增加手动创建工程章节	GQQ

## 目录

1. 文档说明 .....	5
2. 资源列表 .....	5
3. 硬件环境 .....	5
3.1 调试模块（蓝色部分） .....	5
3.2 芯片（红色部分） .....	6
4. 软件环境 .....	6
4.1 CDK 开发环境 .....	6
4.2 代码结构 .....	7
5. 例程运行 .....	8
5.1 例程从原厂获取 .....	8
5.2 例程从 CDK 中获取 .....	8
5.3 手动搭建工程 .....	11
5.4 工程配置 .....	15
5.5 导入 flash 算法 .....	17
5.6 适配代码 .....	18
5.7 外设使用 .....	24
5.8 编译工程 .....	24
5.9 下载调试 .....	25
6. APPENDIX .....	26
6.1 apt32f173x_demo .....	26
6.2 apt32f173x_demo_callback .....	77



## 1. 文档说明

该文档基于爱普特CSI驱动代码架构，适用于APT32F173x系列芯片

## 2. 资源列表

为了使用APT32F173x系列芯片，您将可能需要下列资源

- 系列开发板。详见 [硬件环境](#) 说明。
- miniUSB线。通常随开发板发出。
- CDK。详见 [软件环境](#) 说明。
- 相关文档
  - 本文档。
  - 系列使用手册。面向应用开发人员，旨在给予173x系列内核、存储及全部外围的完整信息。
  - 系列内产品的数据手册。包含芯片管脚分布、封装信息、电器参数等型号专属信息。
- 驱动和demo工程。详见[例程运行](#)。 内含
  - 芯片CSI驱动库及模块示例代码。

## 3. 硬件环境

您收到的开发板大致如下图所示，分左右两部分。蓝色部分是调试模块；红色部分是 APT32F173x 系列模块。左右部分通过 SWD 接口和供电相互连接，所以在某些时候这两部分可以分开使用。

开发板实际布局不同时期可能会有一点差别，下图仅为参考。

### 3.1 调试模块（蓝色部分）

- 1、miniUSB 口，用以连接 PC。
- 2、选择系列模块的供电电压。系列模块的工作电压来自调试模块，5V/3V3 可选。
- 3、调试主芯片。
- 4、调试主芯片复位按键。
- 5、SWD 调试接口。

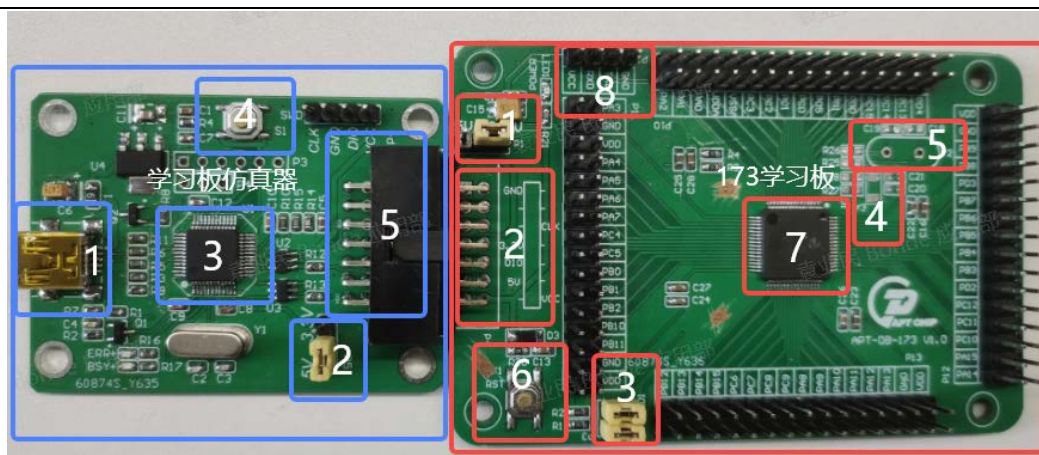


Figure 1 学习板仿真器及 173 学习板

### 3.2 芯片（红色部分）

- 1、反馈电压，5V/3V3 可选。
- 2、SWD 调试接口。与调试模块连接的接口。
- 3、SWD 跳线。APT32F173x 系列只有 1 组 SWD。SWD 口为 PA14（SWCLK）和 PA13（SWDIO）。这组 SWD 跳线用来控制来自调试模块的 SWD 信号要不要连接芯片的 PA13 和 PA14。使用时需用跳线连接
- 4、副晶振电路和跳线。APT32F173x 系列支持外部晶振。但因为内部也有时钟，所以外部晶振不是必须的。需要外部晶振时，这里预留了晶振和电容接插口，可以直接插入对应晶振和电容。
- 5、主晶振电路和跳线。APT32F173x 系列支持外部晶振。但因为内部也有时钟，所以外部晶振不是必须的。需要外部晶振时，这里预留了晶振和电容接插口，可以直接插入对应晶振和电容。
- 6、复位电路和跳线。APT32F173x 系列支持外部复位脚复位。但因为同时支持 POR，所以外部复位不是必须的。当复位脚对应的管脚用作它途时，需要将这个跳线断开。
- 7、APT32F173x 系列芯片。
- 8、调试串口，使用的为 UART1,,其中 PA2 为 TX,PA3 为 RX。

## 4. 软件环境

### 4.1 CDK开发环境

#### 4.1.1 CDK下载

APT32F173x系列使用平头哥的剑池CDK作为软件开发平台。

CDK下载地址：<https://occ.t-head.cn/community/download?id=4176310636235526144>

### 4.1.2 CDK开发环境注意事项

- CDK 不支持中文路径，安装路径及工程路径上不可以有空格或中文字符。
- 使用CDK2.20以后的版本，以确保debug server版本5.16.8+。
- 如果您的电脑使用了如360之类的杀毒软件，除了在安装过程中允许CDK的操作之外，安装之后，必须将整个CDK安装目录加入到杀毒软件的白名单区。
- 如果需要增减工程文件，必须在CDK工程视图下完成。如果在windows文件目录中操作（复制粘贴）后编译会出现错误。
- 软复位只复位 CPU，如果要复位整个芯片，请使用硬复位功能
- 如果要调试 IWDT，WWDT 等可以触发复位的功能，需要在 ICE Configuration/ICE/Other Flags 下添加 - setresethaltreq-always-on 。

## 4.2 代码结构

下图为工程视图。这里面包含了6个组件。

apt32f173x是工程组件，客户的业务逻辑代码一般放在Packages这个组件下面。

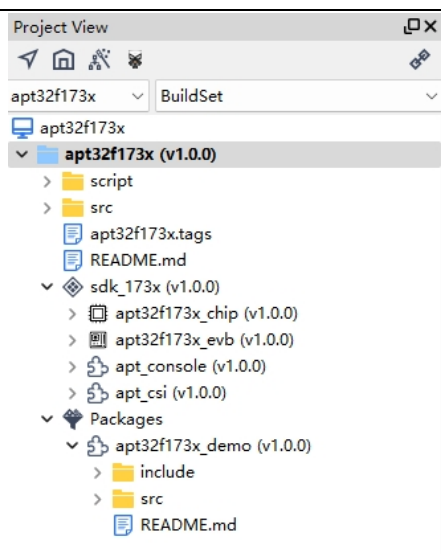


Figure 2 代码结构

## 5. 例程运行

### 5.1 例程从原厂获取

从原厂FAE处，获取最新的例程。

### 5.2 例程从CDK中获取

#### 5.2.1 SDK包获取

1、打开CDK，更新demo工程（CDK版本为V2.19.5）

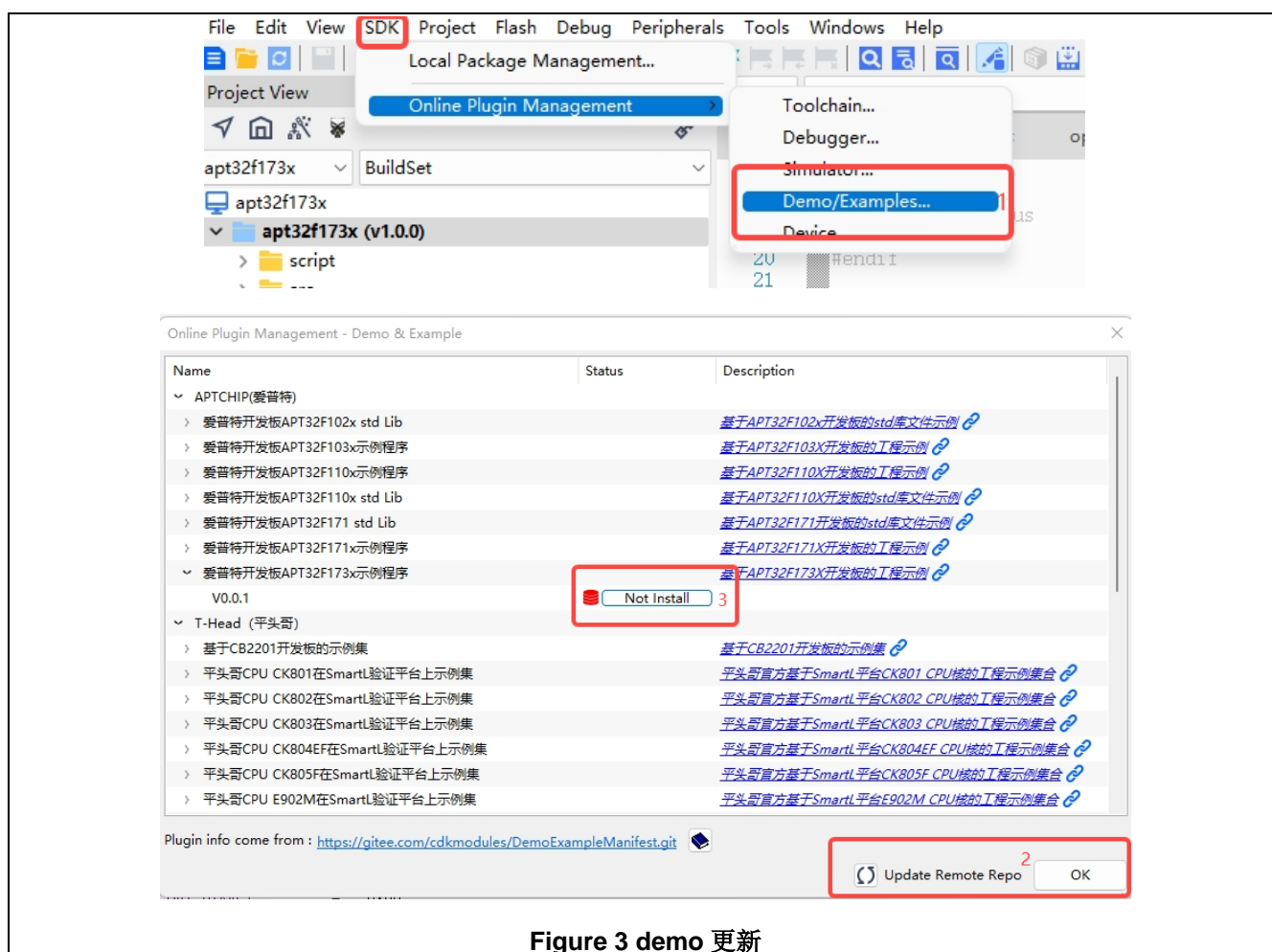


Figure 3 demo 更新

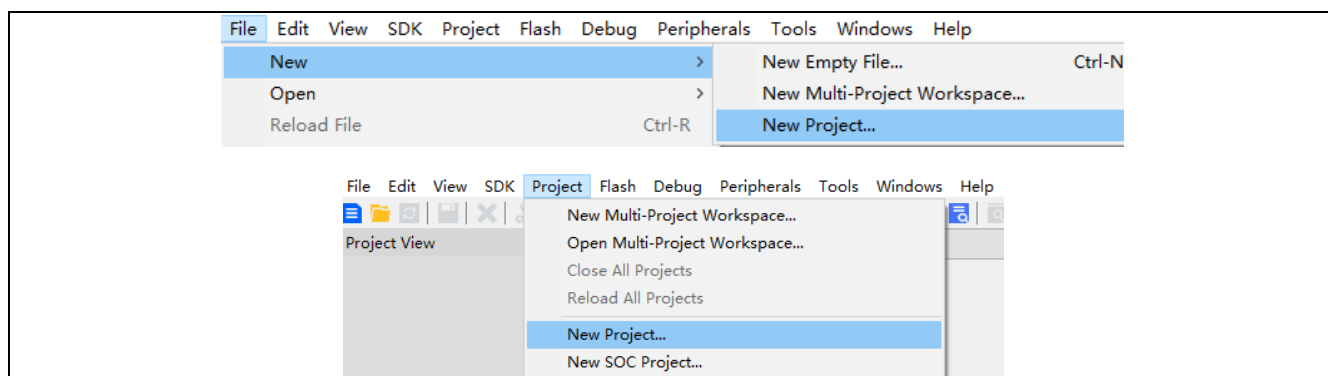
2、Demo工程更新完成会显示Installed，更新完成的Demo & Example页面如下图

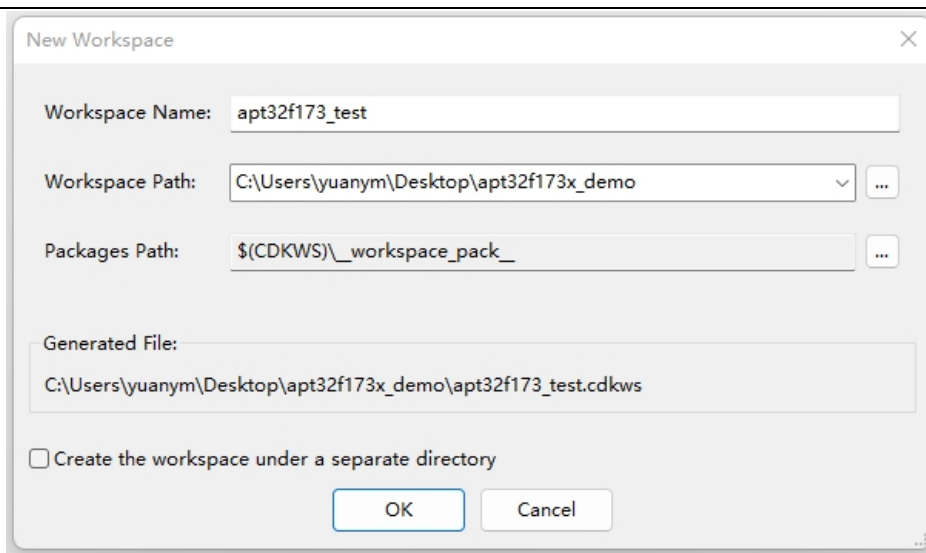




## 5.2.2 创建工程

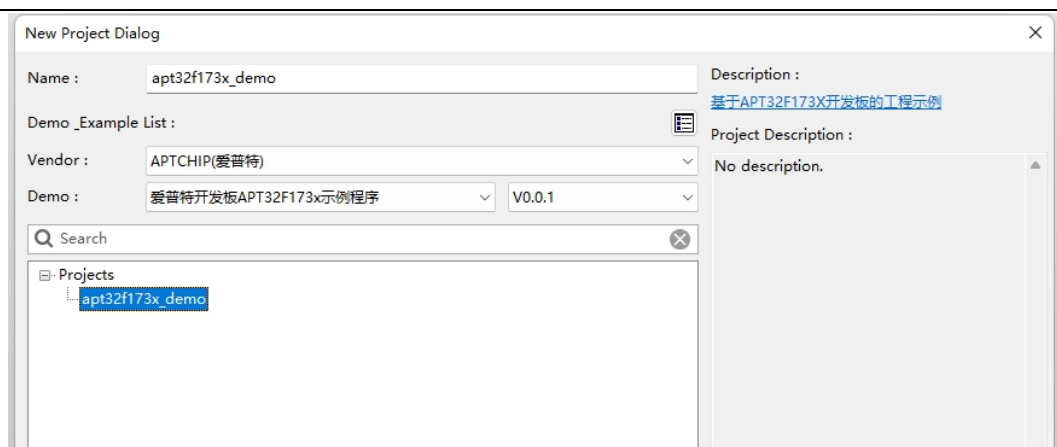
- 1、打开CDK，创建一个新的Project到一个目录，可以通过“File->New->New Project”或者“Project->New Project”两种方式创建一个新的Project。



**Figure 5 创建新的 Project**

2、在接下来弹出的界面中配置好工程参数，参考下图进行配置，点击OK结束配置界面。

注意：在输入工程名之前，要先选中下方的“apt32f173x\_demo”。

**Figure 6 配置工程参数**

3、到这里，我们就完成了APT32F173x工程的创建，此时在CDK左边栏我们就可以看到工程的目录树。

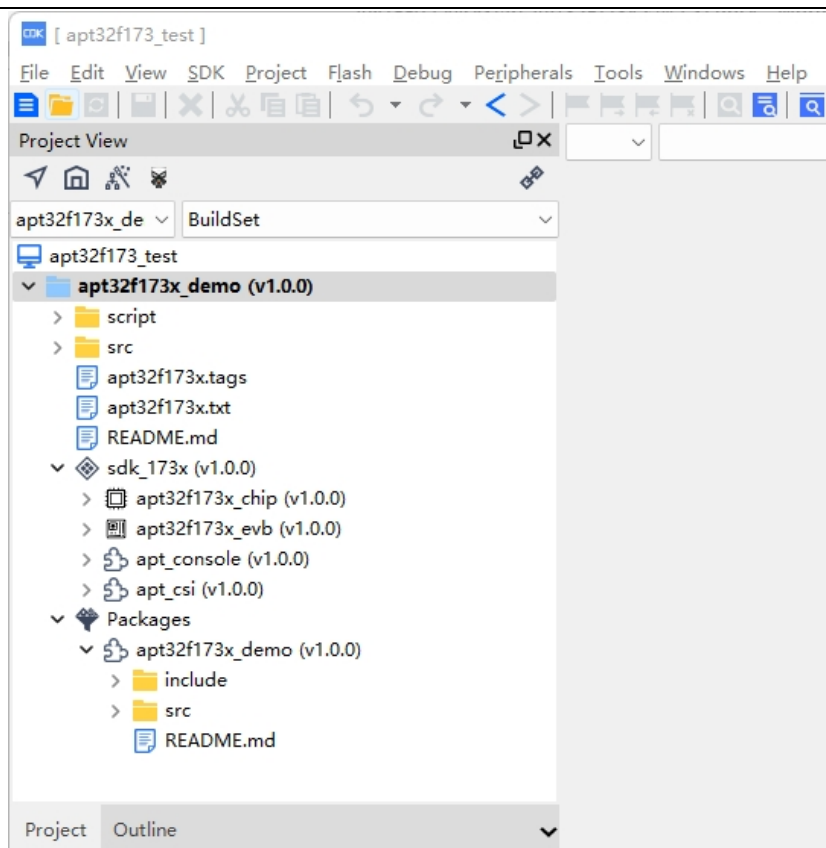
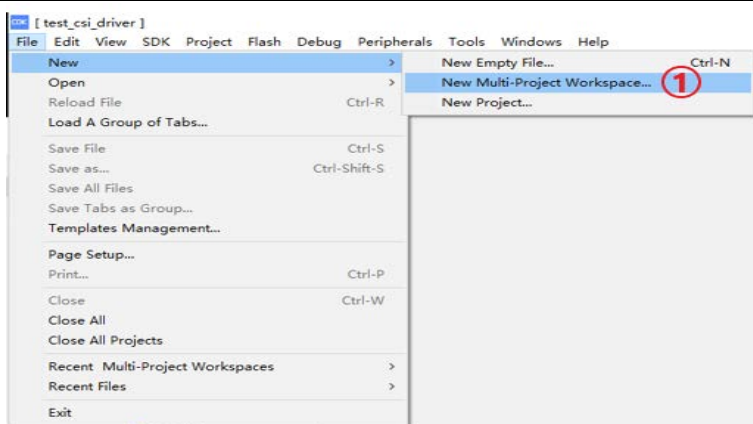
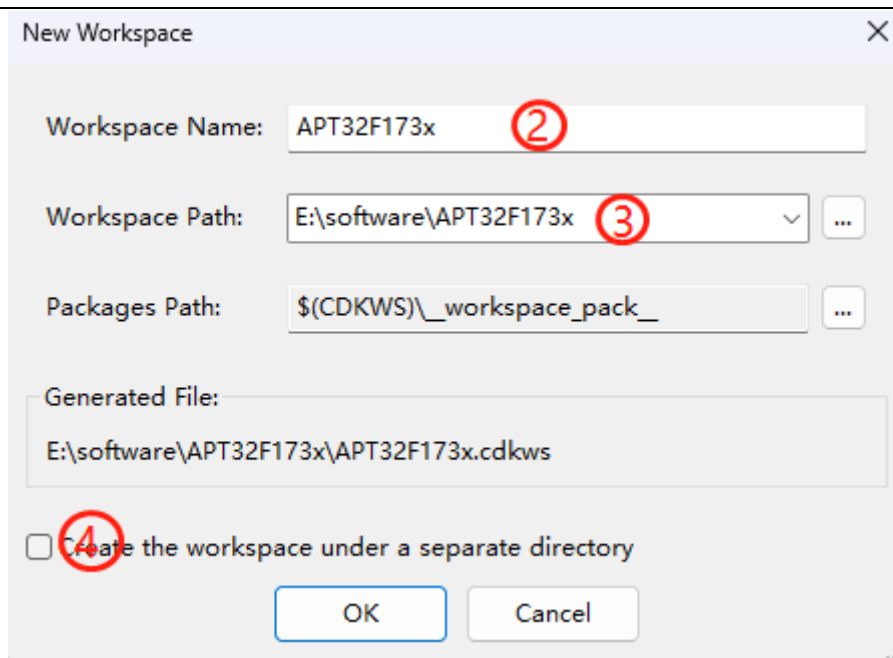


Figure 7 创建好的工程目录树

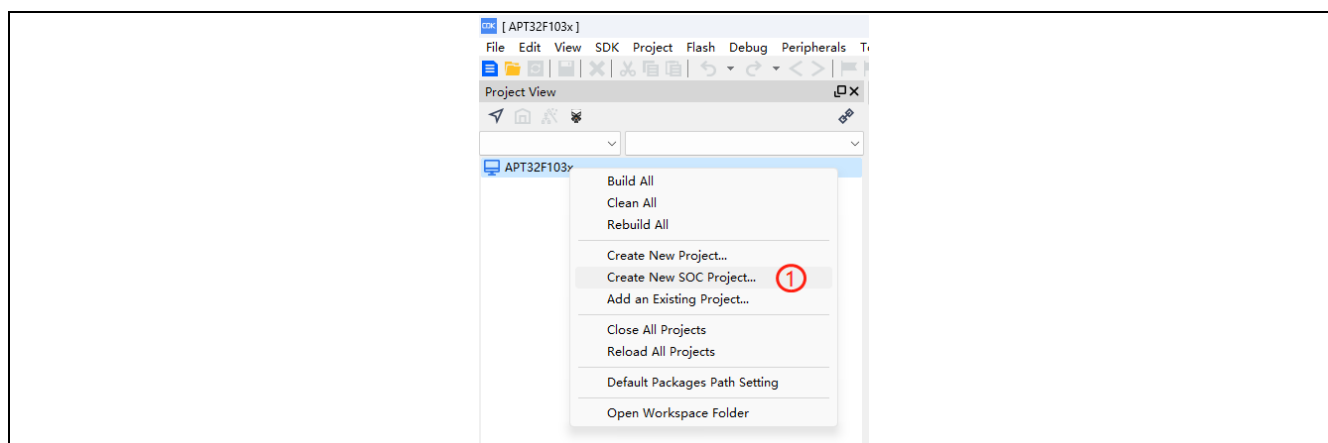
### 5.3 手动搭建工程

- 1、打开CDK，创建一个新的workspace到某一目录（假设E:\software\APT32F173x），将173SDK包放到此文件夹下。



**Figure 7 创建新的 workspace**

2、在workspace下创建新的soc project，按照图中的进行配置。



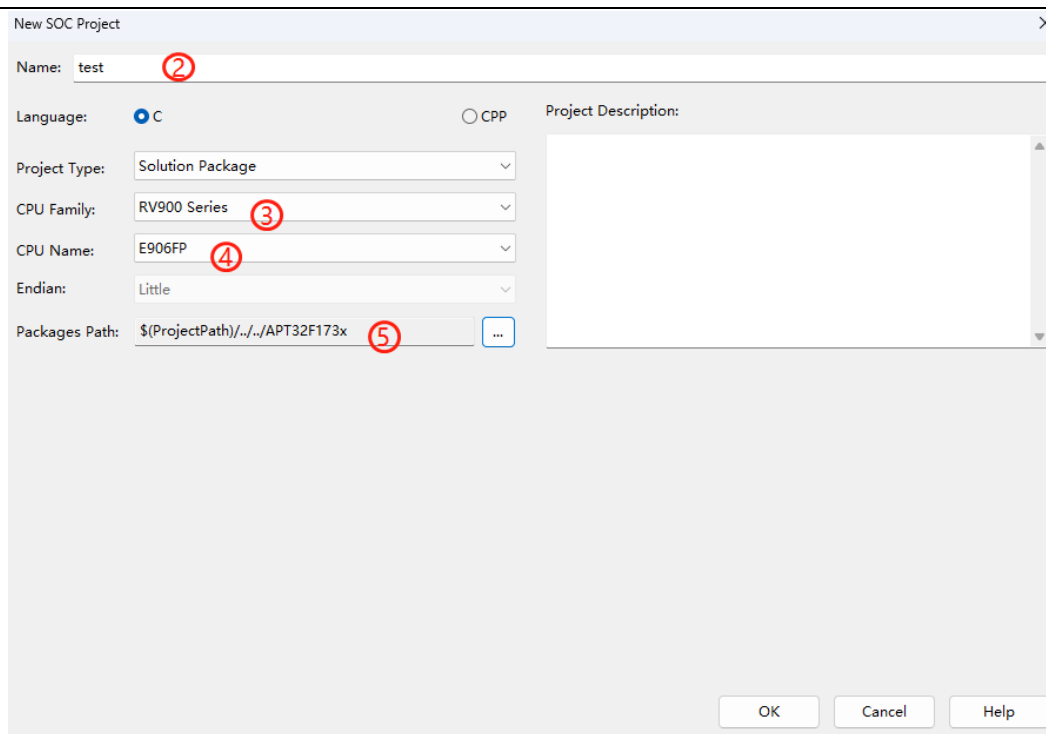
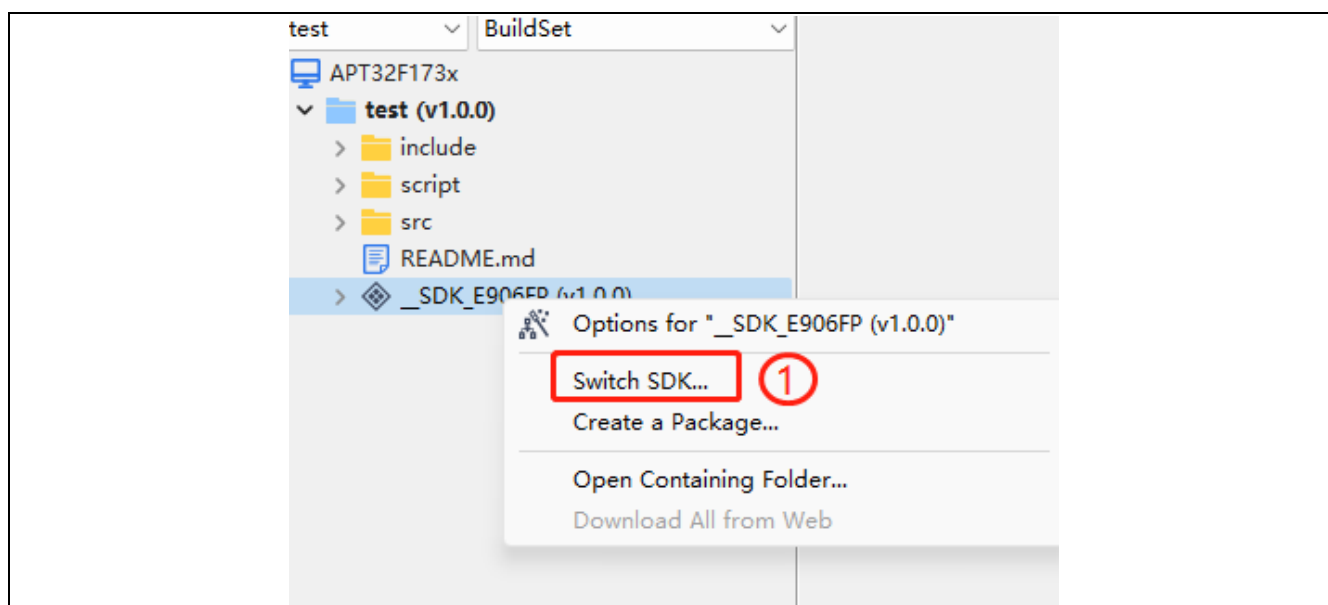
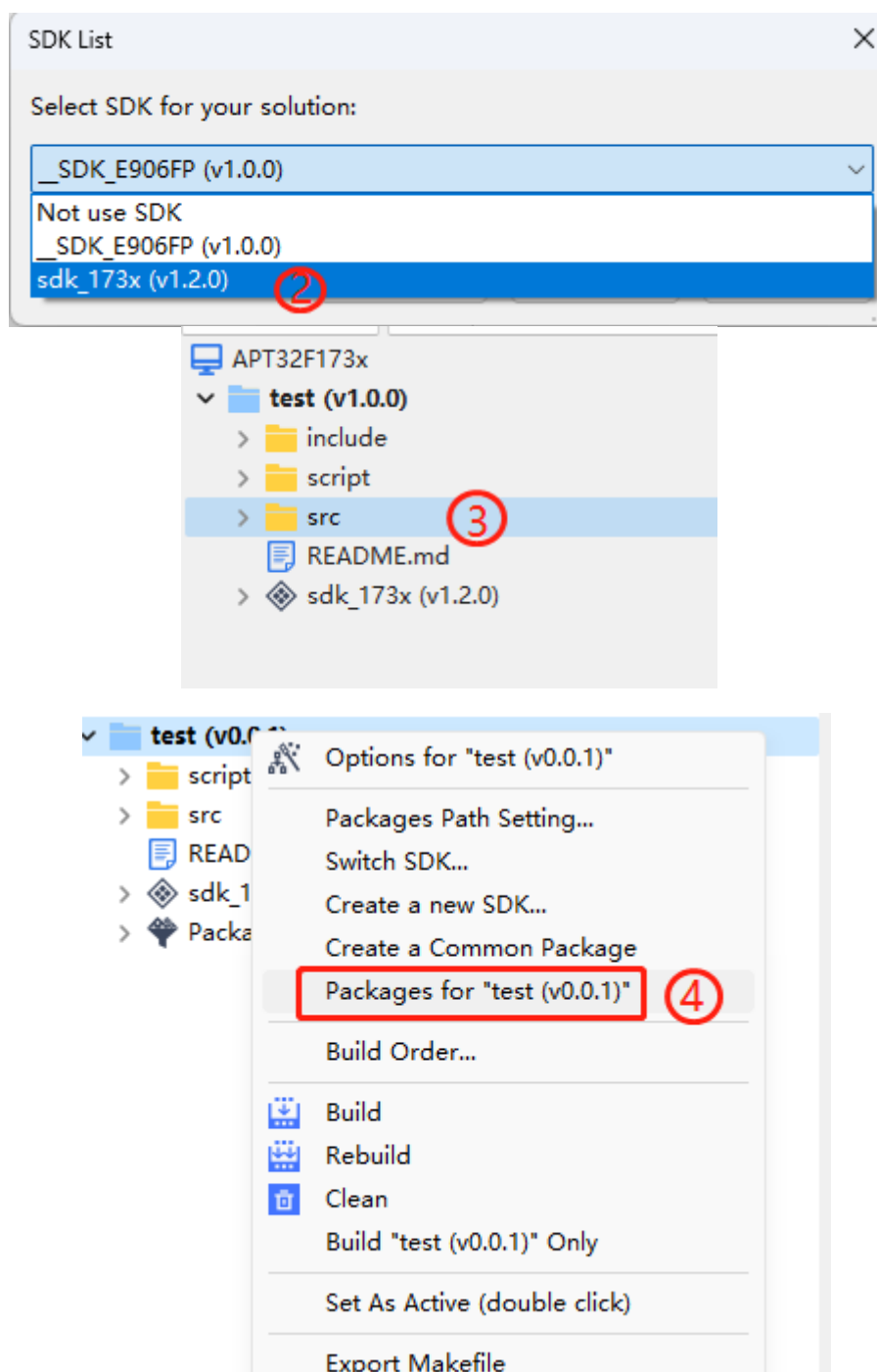


Figure 8 创建新的 soc project

### 3、一键切换SDK。

1. 按照图中红色标注1右键点击test工程，选择切换SDK。切换完成后，建议在工程目录下删除\_\_SDK\_E906和\_\_CHIP\_E906的目录。
2. 按照图中红色标注4右键点击test工程选择创建Package。
3. 在弹出的对话框中按照红色标志5、6、7、8选择添加需要的组件，test工程建立完毕。





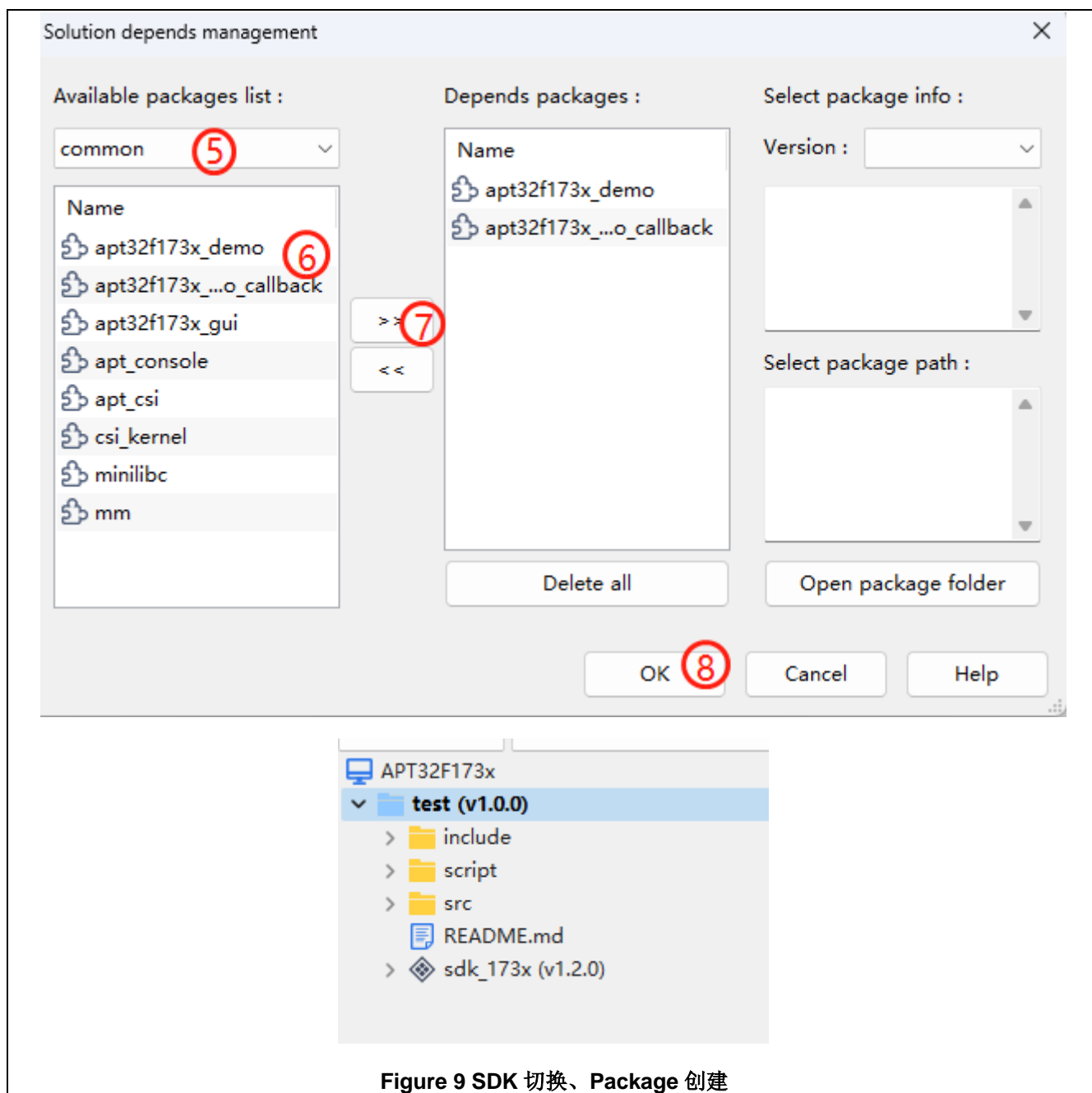


Figure 9 SDK 切换、Package 创建

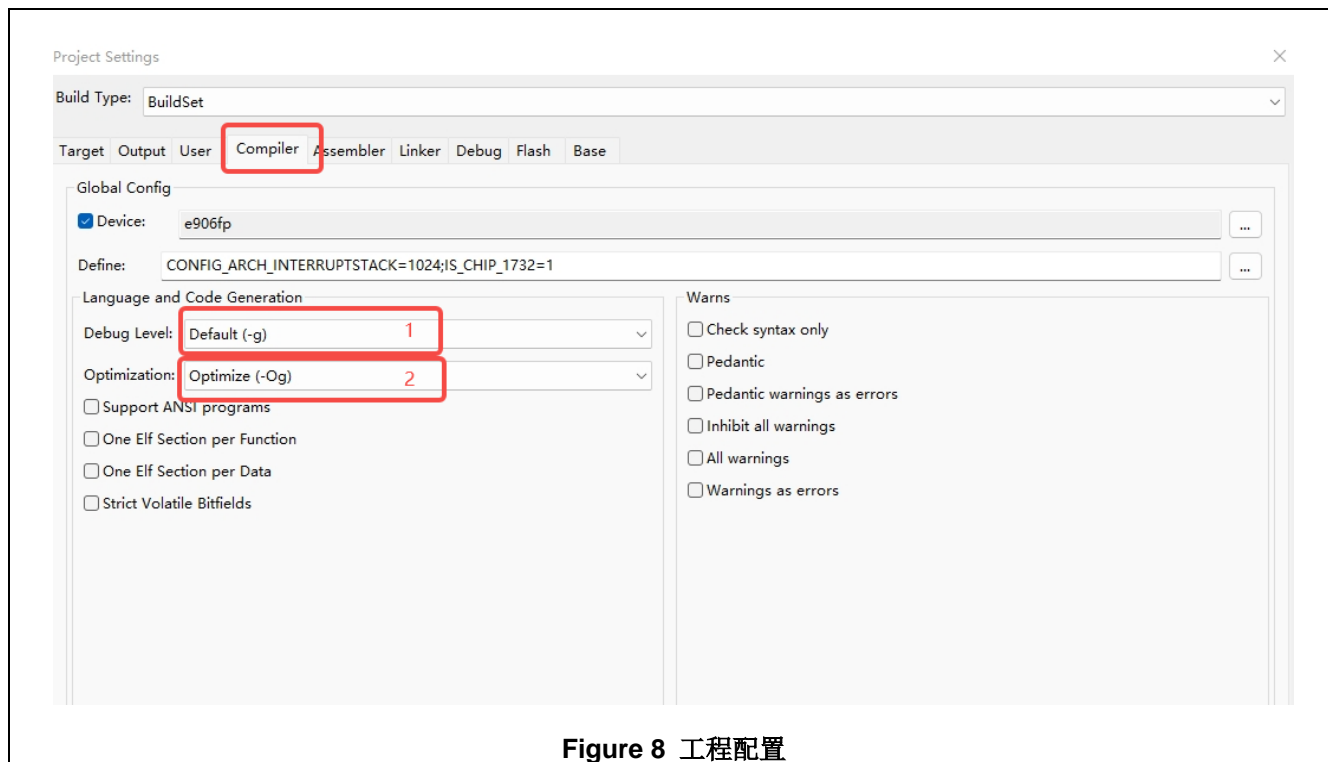
注意：

1. 工程中必须添加的组件有apt32f173x\_chip, apt32f173x\_evb, apt\_console, apt\_csi。
2. 如果需要查看IP例程，可以添加apt32f173x\_demo组件或者apt32f173x\_demo\_callback组件。如果需要图形化配置，可以添加apt32f173x\_gui组件。如果需要使用实时操作系统freeRTOS，可以添加csi\_kernel组件。

## 5.4 工程配置

## 1. 配置选项Compiler

- 标注1, **Debug Level**: 编译的调试选项; 包含None -g -g1 -g3, None表示不包含调试信息, -g3包含最全面的调试信息; 不同的调试选项会影响生成elf的文件大小, 但是不影响最终下载到目标板的image大小, 所以建议开最大即可。若不开调试信息选项, 将导致无法进行源码调试, 只能进行汇编调试;
- 标注2, **Optimization**: 编译的优化选项; 分为-O0, -Og, -O1, -O2, -O3, -Os六种类型:  
例如: -Os, 该优化会以代码密度为优先, 尽可能优化生成的代码。在此选项下, 生成的代码较小, 比较适合空间敏感的MCU程序编译。



## 2. 配置选项Output

- 标注3, 表示工程编译后, 会输出hex文件;
- 标注4, 表示工程编译后, 会输出bin文件;
- 标注5, 表示工程编译后, 会输出.asm汇编文件, 可查看函数的汇编运行指令;
- 标注6, 表示工程编译后, 会输出.map汇编文件, 可通过此文件, 可查看变量和函数空间分配等信息。





Figure 9 工程配置

### 3. 配置选项Linker

- 标注7选择Linker文件,一般在mcu厂商提供的SDK包的board目录下, 名称叫gcc\_flash\_1732.ld。点击标注8按钮, 在标注9窗口, 可根据需求, 选择合适的ld文件。

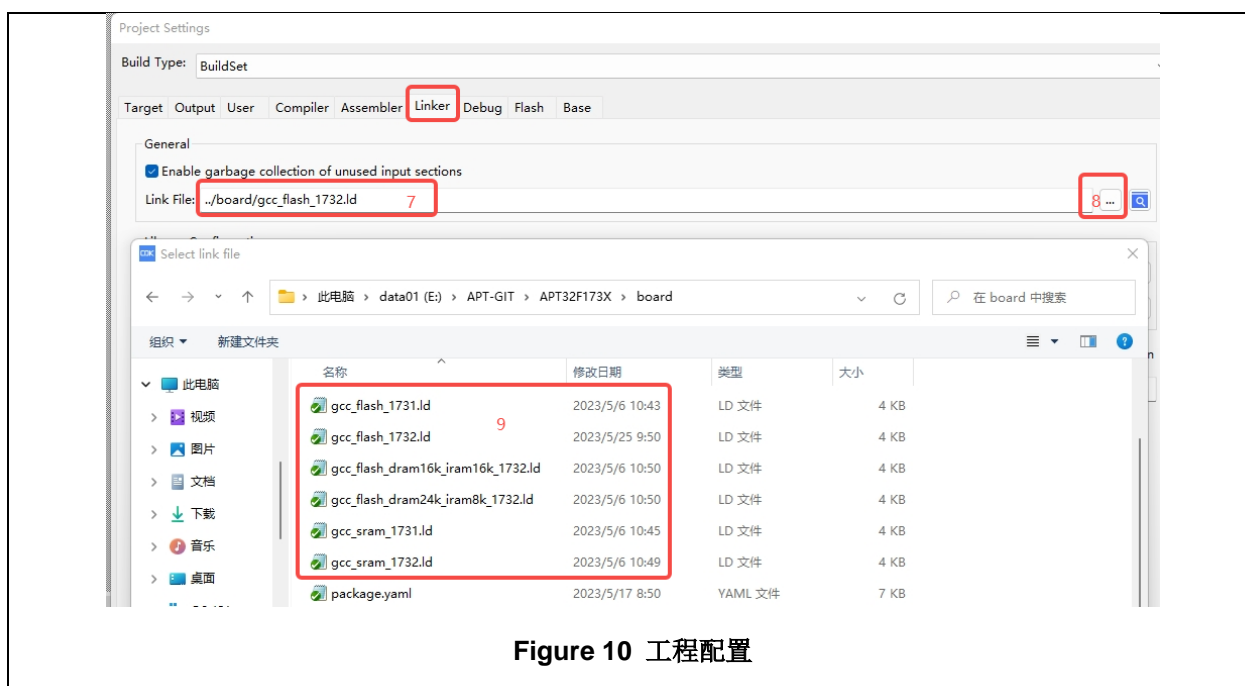


Figure 10 工程配置

## 5.5 导入flash算法

CSI驱动代码使用了SOC工程结构, SDK中已包含工程所需要的算法文件。所以无需额外导入。

## 5.6 适配代码

这部分说明涉及最基础的嵌入式编程相关的内容，包括系统初始化和中断实现。

`main()` 函数需要调用两个函数，`system_init()`和`board_init()`，如下图所示。分别完成时钟配置和串口配置的工作。

```
10  */
11  /* Includes -----*/
12  #include <string.h>
13  #include <csi_drv.h>
14  #include <iostream.h>
15
16  #include "demo.h"
17  /* Private macro -----*/
18
19  /* externs function -----*/
20  extern void system_init(void);
21  extern void board_init(void);
22  extern void user_demo(void);
23
24  /* externs variables -----*/
25  /* Private variables -----*/
26
27  int main()
28  {
29      system_init();
30      board_init();
31      csi_pin_set_mux(PA6, PA6_OUTPUT);    //PA6 output
32      csi_pin_set_high(PA6);               //PA6 output high;
33  }
```

Figure 11 main 函数

### 5.6.1 时钟配置

示例工程中提供了时钟配置，可以根据应用增减，但下图中黄色背景的为必须有的步骤。

函数	说明	位置
system_init()	<pre> __attribute__((weak)) void system_init(void) {     uint32_t i;     csi_icache_enable();  #ifdef CODE_REMAP_TO_IRAM     csi_iram_init(); //需要与 gcc_flash_dram16k_iram16k 或 gcc_flash_dram24k_iram8k.ld 配套使用 #endif      __disable_excp_irq();     uint32_t mstatus = __get_MSTATUS();     mstatus  = (1 &lt;&lt; 13); //使能 mstatus FS 功能（必须有）     __set_MSTATUS(mstatus);     /* 从信息中获取中断级别*/     CLIC-&gt;CLICCFG = (((CLIC-&gt;CLICINFO &amp; CLIC_INFO_CLICINTCTLBITS_Msk) &gt;&gt;     CLIC_INFO_CLICINTCTLBITS_Pos) &lt;&lt; CLIC_CLICCFG_NLBIT_Pos);     for (i = 0; i &lt; 64; i++) {         CLIC-&gt;CLICINT[i].IP = 0;         CLIC-&gt;CLICINT[i].ATTR = 1; // 使用矢量中断（必须有）     }  #ifdef CONFIG_IRQ_LOOKUP //Table lookup method for interrupt processing     irq_vectors_init(); #endif      csi_iwdt_close(); //关看门狗 iwdt（调试时用）     csi_sysclk_config(g_tClkConfig); //配置系统时钟（必须有）     csi_tick_init(); //初始化 systick     __enable_excp_irq(); //开启全局中断 } </pre>	system.c

其中csi\_sysclk\_config()会去读取一个时钟配置结构体变量g\_tClkConfig（位于board\_config.c）。可以通过修改g\_tClkConfig来实现系统和外设主时钟的配置。下图是默认配置。

```

/* system clock configuration parameters to define source, source freq(if selectable), sdiv and pdiv
 *
 * such as: g_tClkConfig.eSdiv = SCLK_DIV1, g_tClkConfig.wSclk = g_tClkConfig.wFreq / 1
 *           g_tClkConfig.ePdiv = SCLK_DIV4, g_tClkConfig.wPclk = g_tClkConfig.wSclk / 4
 *
 */
csi_clk_config_t g_tClkConfig =
{
    // {SRC_HFOSC, HFOSC_24M_VALUE, SCLK_DIV1, PCLK_DIV1, HFOSC_24M_VALUE, HFOSC_24M_VALUE};
    // {SRC_HFOSC, HFOSC_12M_VALUE, SCLK_DIV1, PCLK_DIV1, HFOSC_12M_VALUE, HFOSC_12M_VALUE};
    // {SRC_HFOSC, HFOSC_6M_VALUE, SCLK_DIV1, PCLK_DIV1, HFOSC_6M_VALUE, HFOSC_6M_VALUE};
    // {SRC_HFOSC, HFOSC_3M_VALUE, SCLK_DIV1, PCLK_DIV1, HFOSC_3M_VALUE, HFOSC_3M_VALUE};
    {SRC_AUTO_HF_PLL, PLL_VALUE, SCLK_DIV1, PCLK_DIV1, PLL_VALUE, PLL_VALUE};
    // {SRC_AUTO_EM_PLL, PLL_VALUE, SCLK_DIV1, PCLK_DIV1, PLL_VALUE, PLL_VALUE};
    // {SRC_MANUAL_PLL, PLL_VALUE, SCLK_DIV1, PCLK_DIV1, PLL_VALUE, PLL_VALUE};
    // {SRC_EMOSC, EMOSC_VALUE, SCLK_DIV2, PCLK_DIV2, EMOSC_VALUE/2, EMOSC_VALUE/4};
    // {SRC_IMOSC, IMOSC_5M_VALUE, SCLK_DIV1, PCLK_DIV1, IMOSC_5M_VALUE, IMOSC_5M_VALUE};
    // {SRC_IMOSC, IMOSC_4M_VALUE, SCLK_DIV1, PCLK_DIV1, IMOSC_4M_VALUE, IMOSC_4M_VALUE};
}

```

Figure 12 时钟配置结构体

### 5.6.2 串口配置

仿真环境下，支持两种串口信息的输出。

1. 虚拟串口，使用时需要勾选semihost配置，同时需要在使用打印之前，添加框图所示的代码，并且打印的时候使用printf，例如printf("Hello APT32F173x\n")，Semihost打印信息在Output窗口查看。

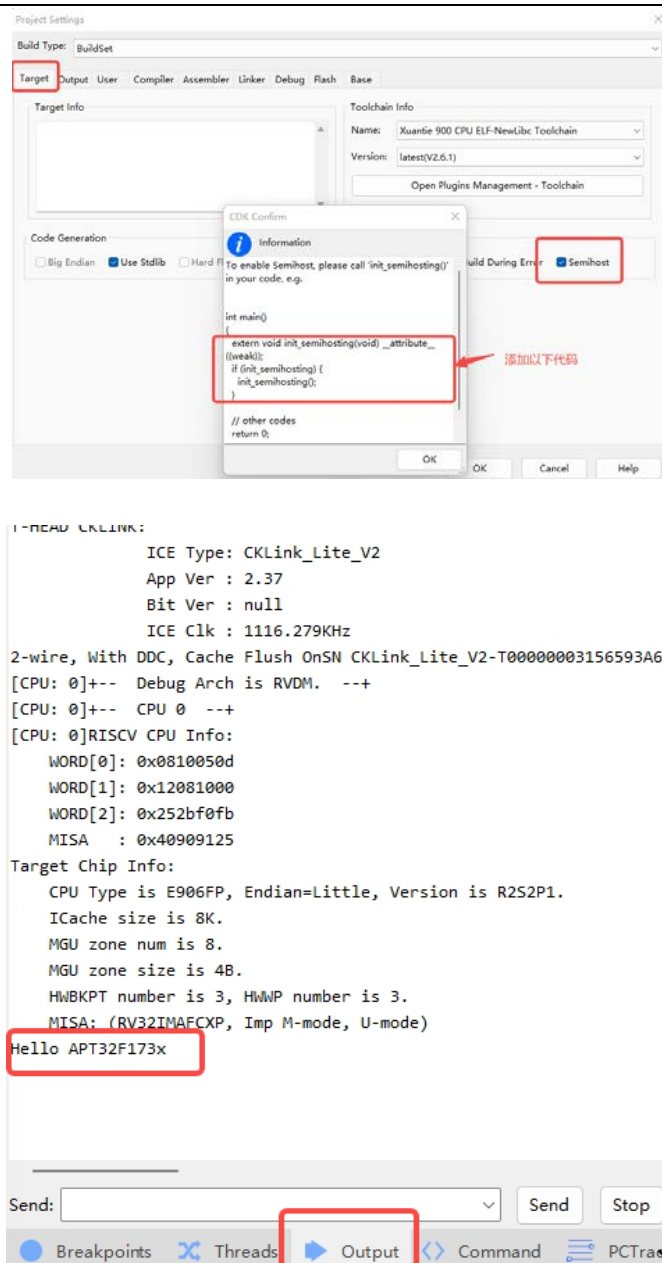
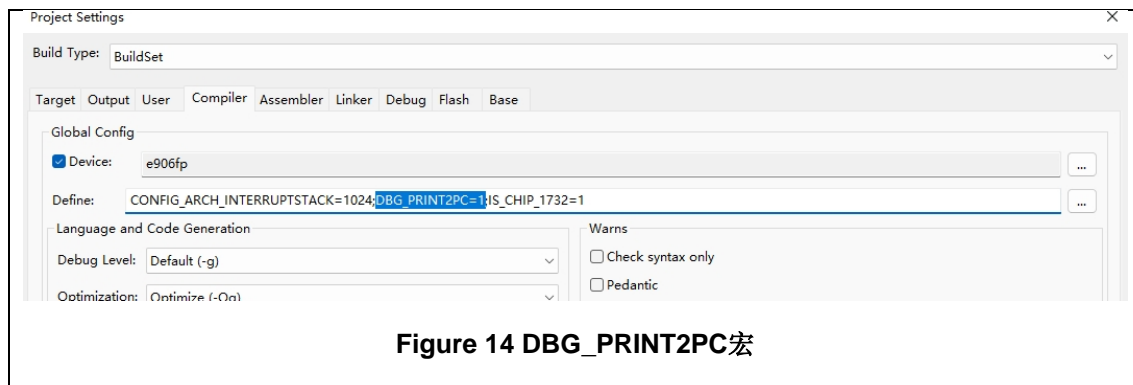


Figure 13 勾选 Semihost

2. 真实的串口输出，使用真实的硬件串口。使用时需要确保全局宏DBG\_PRINT2PC=1存在。调试串口数据格式固定为数据位8位，停止位1位，不校验。默认使用UART1，PA2（TXD）和PA3（RXD）。注意查看MCU 型号是否有UART1外设。UART口及对应的管脚资源可改。如果要调整串口资源，需要在board\_config.h中更改相应的宏，并调整硬件连接。

打印的时候使用my\_printf，例如my\_printf("hello world!\n");



函数	说明	位置
board_init()	<p>实现 UART 硬件配置，包括 UART id、波特率、管脚等。 所有的宏在 board_config.h 中定义。</p> <pre>__attribute__((weak)) void board_init(void) {     //console config for print     g_tConsole.uart_id = (uint32_t)CONSOLE_IDX;     g_tConsole.baudrate = 115200U;     g_tConsole.tx.pin = CONSOLE_TXD;     g_tConsole.tx.func = CONSOLE_TXD_FUNC;     g_tConsole.rx.pin = CONSOLE_RXD;     g_tConsole.rx.func = CONSOLE_RXD_FUNC;     g_tConsole.uart = (csp_uart_t *) (APB_UART0_BASE + CONSOLE_IDX * 0x1000);     console_init(&amp;g_tConsole); }</pre>	board_config.c

### 5.6.3 中断函数

中断处理函数位于board/src/interrupt.c中。这个文件搭建了中断处理函数的架子，特别是某些功能实现和中断强相关的外设，如ADC0。

```
ATTRIBUTE_ISR void adc0_int_handler(void)
{
    #if ADC0_INT_HANDLE_EN
        // ISR content ...
        adc_irqhandler(ADC0);
    #endif
}
```

Figure 15 中断处理函数举例

在上述例子中，adc\_irqhandler()函数在驱动代码（adc\_demo.c）中定义，但具有weak属性。我们推荐用户使用驱动中已经定义的中断处理函数。但在一些特殊的场合，用户仍然可以以同样的名字对这个函数重新定义，而不需要修改adc0\_int\_handler内部的代码。此时，编译会忽略weak属性的预定义函数，将用户新写的同名函数加入编译。

另外需要注意的是，中断函数的进出会比普通的函数调用有更多的步骤，会消耗更多的代码资源。所以,如果

- 应用对代码大小敏感，可以删掉无用的中断处理函数。以usart0\_int\_handler ()为例，如果应用中没有用到USART相关的中断，那么删除这部分代码的步骤如下：

- 1、interrupt.c中删除void usart0\_int\_handler (void)

- 2、srtartup.S中将原来usart0\_int\_handler替换为Default\_Handler

- 应用对运行时间敏感，除了尽量减少中断函数内部的处理外，还可以考虑减少函数调用层级。

## 5.7 外设使用

外设使用的示例代码在APT32F173x\_demo这个组件中。main()函数罗列了所有的示例函数，可以通过“打开、关闭”注释的方式调用相应的示例函数。

## 5.8 编译工程

点击“Build Project”既可实现工程的编译和连接。

按钮1：工程构建按钮；对工作空间中的活动工程进行构建；

按钮2：工程重新构建按钮；对工作空间中的活动工程进行重新构建。

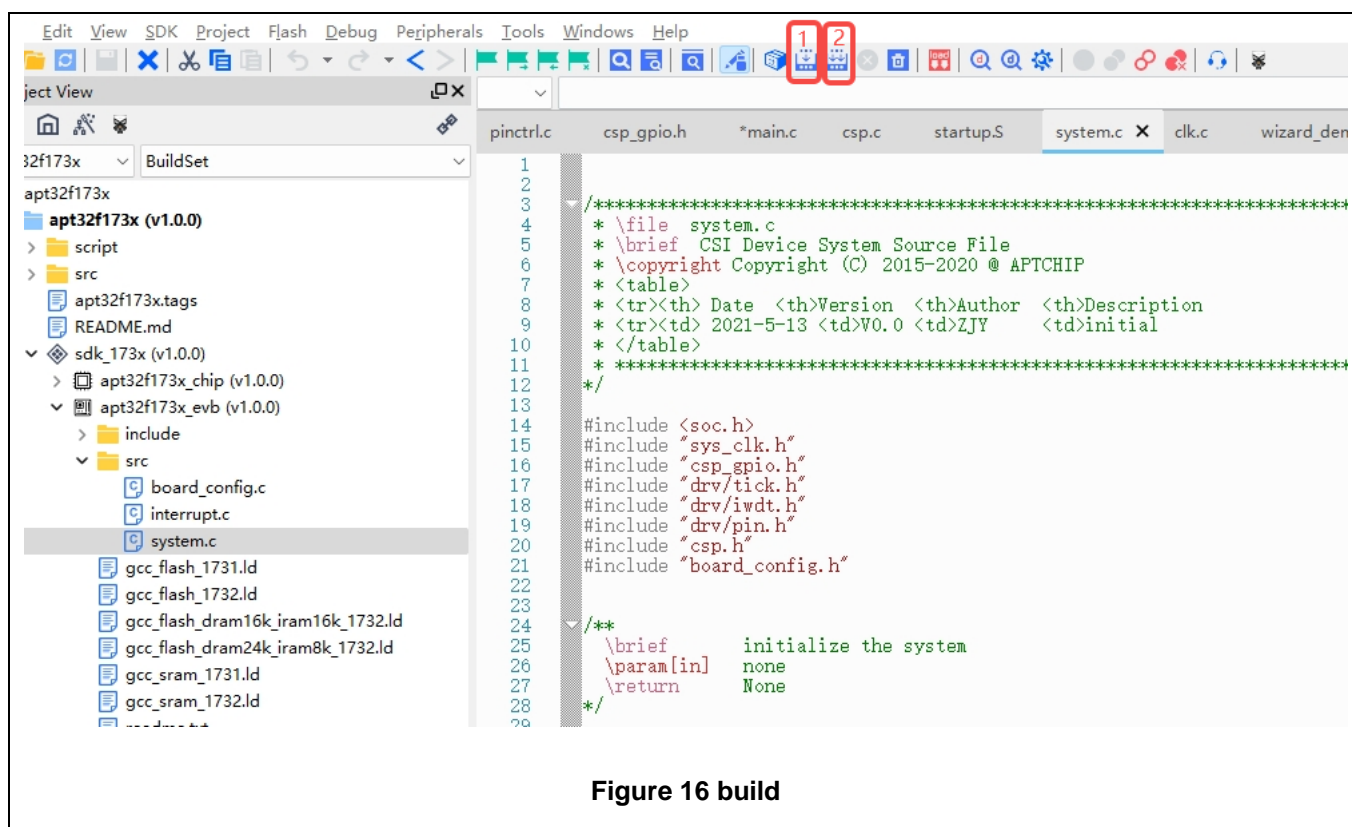


Figure 16 build



## 5.9 下载调试

视应用情况，可以选择三种下载方式：

1. 将代码下载到flash区，不进入debug模式
2. 将代码下载到flash区，随后进入debug模式
3. Attach操作，不重新下载代码，不对芯片做任何复位操作，直接进入调试模式，查看芯片当前运行状态下图中的1，2，3分别对应上面三种下载方式的操作菜单。

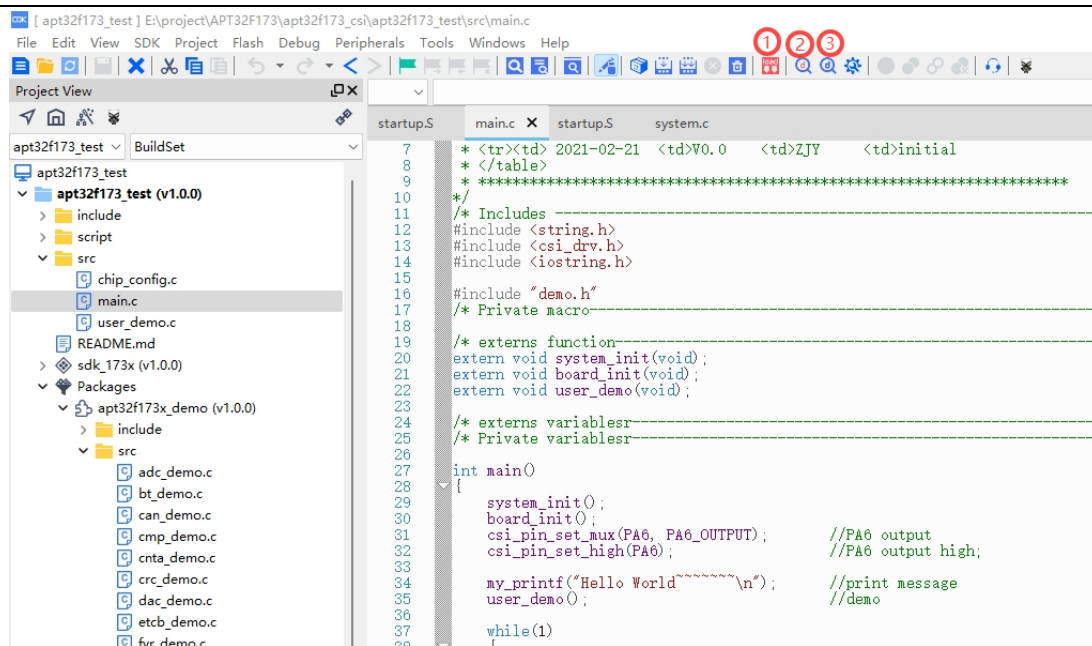


Figure 17 下载调试的三种选择

## 6. Appendix

这部分内容用于说明芯片外设示例代码的使用方法，示例代码位于`apt32f173x_demo`和`apt32f173x_demo_callback`组件下。其中`apt32f173x_demo`中的代码主要演示外设的简单使用，`apt32f173x_demo_callback`中的代码主要演示中断回调的使用方法。在`main`函数中调用了`user_demo()`函数，该函数罗列了所有的示例代码，可以通过“打开、关闭”注释的方式调用相应的示例代码。

### 6.1 apt32f173x\_demo

#### 6.1.1 GPIO

##### 6.1.1.1 gpio\_output\_demo

###### 1、函数位置

`gpio_demo.c`

###### 2、功能描述

配置GPIO为输出功能，并输出高电平或低电平。

###### 3、参考用法

```
csi_gpio_set_mux(GPIOA, PA2, PA2_OUTPUT);           //PA5配置为输出
csi_gpio_set_high(GPIOA, PA2);                      //输出高
csi_gpio_set_low(GPIOA, PA2);                       //输出低
```

详见`gpio_output_demo`函数。

##### 6.1.1.2 gpio\_input\_demo

###### 1、函数位置

`gpio_demo.c`

###### 2、功能描述

配置GPIO为输入功能，并使能上拉或者下拉。

###### 3、参考用法

```
csi_gpio_set_mux(GPIOA, PA0, PA0_INPUT);           //PA5配置为输入
csi_gpio_pull_mode(GPIOA, PA0, GPIO_PULLNONE);    //无上下拉
csi_gpio_pull_mode(GPIOA, PA0, GPIO_PULLUP);      //上拉使能
csi_gpio_pull_mode(GPIOA, PA0, GPIO_PULLDOWN);    //下拉使能
```

详见`gpio_input_demo`函数。

##### 6.1.1.3 gpio\_irq\_demo

###### 1、函数位置

gpio\_demo.c

## 2、功能描述

配置GPIO为外部中断功能，并配置中断边沿。

## 3、参考用法

```
csi_gpio_set_mux(GPIOA, PA0, PA0_INPUT);           //PA0配置为输入
csi_gpio_pull_mode(GPIOA, PA0, GPIO_PULLUP);       //上拉
csi_gpio_int_enable(GPIOA,PA0);                     //使能PA0端口对应外部中断
csi_gpio_irq_mode(GPIOA, PA0, EXI_GRP0,GPIO_IRQ_FALLING_EDGE); //下降沿
csi_gpio_vic_irq_enable(EXI_GRP0, ENABLE);          //GPIOA端口对应VIC中断使能
```

详见pin\_irq\_demo函数。

## 6.1.2 Syscon

### 6.1.2.1 lvd\_demo

#### 1、函数位置

syscon\_demo.c

#### 2、功能描述

掉电监测功能示例。用于监控外部电源的电压值，在外部供电电压低于或高于设置值时，产生中断信号。

#### 3、参考用法

```
csi_set_lvd(LVD_INTF,LVD_30);                      //配置LVD 为： VDD掉电到3.0V即触发LVD中断
csi_lvd_lvr_enable();                               //使能LVD
```

详见lvd\_demo函数。

### 6.1.2.2 lvr\_demo

#### 1、函数位置

syscon\_demo.c

#### 2、功能描述

掉电复位功能示例。用于监控外部电源的电压值，在外部供电电压低于设置值时，产生芯片复位信号。

#### 3、参考用法

```
csi_set_lvr(LVR_31);                               //配置LVR为： VDD掉电到3.1V，芯片复位
csi_lvd_lvr_enable();                               //使能LVR
```

详见lvr\_demo函数。

### 6.1.2.3 memorycheck\_demo

#### 1、函数位置

syscon\_demo.c

## 2、功能描述

内存可靠性监测功能示例。通过硬件校验电路对存储单元的数据写入或读取进行校验，校验合格后，数据才会被写入系统总线，若校验失败，控制器会根据设置进行重试，当重试计数达到预设值时，系统将会强制复位。

## 3、参考用法

```
csi_flashcheck_set_time(10);      //开启flash check功能，检查错误次数上限10
csi_flashcheck_rst();             //错误到达上限，芯片复位
csi_sramcheck_set_time(8);        //开启sram check功能，检查错误次数上限8
csi_sramcheck_rst();              //错误到达上限，芯片复位
```

详见memorycheck\_demo函数。

### 6.1.2.4 emcm\_demo

#### 1、函数位置

syscon\_demo.c

#### 2、功能描述

外部时钟可靠性监测功能示例。当外部时钟失效时，可复位芯片或切换到内部时钟运行。

#### 3、参考用法

```
csi_gpio_set_mux(XIN_PORT,XIN_PIN, XIN_PIN_FUNC);
csi_gpio_set_mux(XOUT_PORT,XOUT_PIN, XOUT_PIN_FUNC);
csi_emosc_enable(8000000);        //使能外部晶振驱动电路，输入频率参数，以调整内部增益
csi_emcm_switch_imosc_int ();     //一旦检测到外部晶振失常，系统时钟切到IMOSC，并触发中断
csi_emcm_rst();                   //一旦检测到外部晶振失常，系统复位
```

详见emcm\_demo函数。

### 6.1.2.5 escm\_demo

#### 1、函数位置

syscon\_demo.c

#### 2、功能描述

外部时钟可靠性监测功能示例。当外部时钟失效时，可复位芯片或切换到内部时钟运行。

#### 3、参考用法

```
csi_gpio_set_mux(SXIN_PORT,SXIN_PIN, SXIN_PIN_FUNC);
csi_gpio_set_mux(SXOUT_PORT,SXOUT_PIN, SXOUT_PIN_FUNC);
csi_esosc_enable();               //使能外部晶振驱动电路,输入频率参数，以调整内部增益
csi_escm_2_imosc_int ();          //一旦检测到外部晶振失常，系统时钟切到IMOSC，并触发中断
csi_escm_rst();                   //一旦检测到外部晶振失常，系统复位。
csi_escm_disable();              //取消对外部晶振的检测
```

详见escm\_demo函数。

### 6.1.2.6 syscon\_cqcr\_demo

#### 1、函数位置

syscon\_demo.c

#### 2、功能描述

时钟源频率验证功能

#### 3、参考用法

```
uint32_t wCqsrValue = 0;
csi_set_cqcr(CQCR_REFSEL_EM,CQCR_SRCSEL_IM,0x3ff); //参考时钟选择 EM, 源时钟选择控 IM
wCqsrValue = csi_get_cqsr(); //读取参考源时钟计数值
my_printf("cqsr value =%d",wCqsrValue);
```

详见syscon\_cqcr\_demo函数。

### 6.1.2.7 pmp\_demo

#### 1、函数位置

syscon\_demo.c

#### 2、功能描述

示例前8K地址区域只能执行，不可读写。

#### 3、参考用法

```
volatile uint32_t wTestData

mpu_region_attr_t tPmpConfig;
tPmpConfig.r = 0;
tPmpConfig.w = 0;
tPmpConfig.x = 1;
tPmpConfig.a = ADDRESS_MATCHING_NAPOT;
tPmpConfig.l = 1;
//设置[0x0 - 0x7FF]为保护区间：加锁，只能执行，不能读写。此时memory区显示正常值
csi_mpu_config_region(0, 0x0, REGION_SIZE_8KB, tPmpConfig, ENABLE);
//保护区间memory区不再显示正常值;
```

详见pmp\_demo函数。

## 6.1.3 IWDT

### 6.1.3.1 iwdt\_normal\_demo

#### 1、函数位置

iwdt\_demo.c

## 2、功能描述

独立看门狗定时喂狗功能示例。如果超时没有喂狗，产生系统复位信号。

## 3、参考用法

```
csi_iwdt_init(IWDT_TO_1024);           //初始化看门狗，溢出时间为1024ms(系统复位时间)
csi_iwdt_open();                       //打开看门狗
csi_iwdt_feed();                       //喂狗
```

详见iwdt\_normal\_demo函数。

### 6.1.3.2 iwdt\_int\_demo

#### 1、函数位置

iwdt\_demo.c

#### 2、功能描述

独立看门狗报警功能示例。当计数值达到报警设置值时，会产生一个报警中断信号，提醒用户及时喂狗，防止芯片复位。

#### 3、参考用法

```
csi_iwdt_init(IWDT_TO_1024);           //初始化看门狗，溢出时间为1024ms(系统复位时间)
csi_iwdt_int_enable(IWDT_ALARMTO_2_8, ENABLE); //使能看门狗报警中断，报警时间为2/8溢出时间
csi_iwdt_open();                       //打开看门狗
```

详见iwdt\_int\_demo函数。

### 6.1.4 WWDT

#### 6.1.4.1 wwdt\_demo

#### 1、函数位置

wwdt\_demo.c

#### 2、功能描述

窗口看门狗功能示例。用于监测当前程序运行状况。

#### 1、参考用法

```
csi_wwdt_init(80);                     //设置timeout时间为80ms 时间设置过大会返回错误
csi_wwdt_debug_enable(ENABLE);         //可以配置在debug模式下，wwdt是否继续计时
csi_wwdt_set_window_time(40);          //设置窗口值为40ms
csi_wwdt_int_enable();                 //中断使能
csi_wwdt_open();                       //WWDT一旦使能，软件将不能停止
```

详见wwdt\_demo函数。

## 6.1.5 IFC

### 6.1.5.1 ifc\_read\_demo

#### 1、函数位置

ifc\_demo.c

#### 2、功能描述

flash读操作功能示例。操作单位为word。

#### 3、参考用法

```
csi_ifc_read(IFC, 0x00000000, s_wReadBuf, 2); //从0x0地址读取2个word数据
```

详见ifc\_read\_demo函数。

### 6.1.5.2 ifc\_dflash\_page\_program\_demo

#### 1、函数位置

ifc\_demo.c

#### 2、功能描述

dflash页编程功能示例。起始地址必须word对齐，数据类型为word，数据必须在一个page内，不允许跨页。

#### 3、参考用法

```
csi_ifc_dflash_page_program(IFC, 0x10000000, s_wWriteData, 5); //从0x10000000地址写入5个word数据
```

详见ifc\_dflash\_page\_program\_demo函数。

### 6.1.5.3 ifc\_dflash\_page\_parallel\_program\_demo

#### 1、函数位置

ifc\_demo.c

#### 2、功能描述

dflash并行模式页编程功能示例。只有dflash支持并行模式，在并行模式下，dflash擦写的同时，CPU仍然可以从pflash取址运行。起始地址必须word对齐，数据类型为word，数据必须在一个page内，不允许跨页。

#### 3、参考用法

```
csi_ifc_dflash_paramode_enable(IFC, ENABLE); //使能dflash并行模式  
csi_ifc_dflash_page_program(IFC, 0x10000000, s_wWriteData, 5); //从0x10000000地址写入5个word数据
```

详见ifc\_dflash\_page\_parallel\_program\_demo函数。

#### 6.1.5.4 ifc\_pflash\_page\_program\_demo

##### 1、函数位置

ifc\_demo.c

##### 2、功能描述

pflash页编程功能示例。起始地址必须word对齐，数据类型为word，数据必须在一个page内，不允许跨页。

##### 3、参考用法

```
csi_ifc_pflash_page_program(IFC, 0x0000F000, s_wWriteData, 5); //从0x0000F000地址写入5个word数据
```

详见ifc\_pflash\_page\_program\_demo函数。

#### 6.1.5.5 ifc\_page\_erase\_demo

##### 1、函数位置

ifc\_demo.c

##### 2、功能描述

flash页擦除功能示例。无论dfflash还是pflash，编程操作都自带erase步骤，不需要额外调用该函数，否则会影响flash寿命。

##### 3、参考用法

```
csi_ifc_page_erase(IFC, 0x10000000); //擦除dfflash第一个page
```

详见ifc\_page\_erase\_demo函数。

### 6.1.6 PM

#### 6.1.6.1 lp\_lpt\_wakeup\_demo

##### 1、函数位置

lowpower\_demo.c

##### 2、功能描述

LPT唤醒低功耗功能示例。系统进入低功耗模式（SLEEP或者DEEP\_SLEEP），通过LPT定时500ms唤醒唤醒。

##### 2、参考用法

```
csi_pm_set_wakeup_src(SRC_WKUP_LPT, ENABLE); //配置唤醒源
//lpt初始化配置
tTimConfig.wTimeVal = 500; //LPT定时值 = 500ms
tTimConfig.eRunMode = LPT_CNT_CONT; //LPT计数器工作模式，连续
```



```

tTimConfig.eClksrc=LPT_CLK_ISCLK;           //LPT时钟源
csi_lpt_timer_init(LPT,&tTimConfig);          //初始化lpt,默认采用PEND中断
csi_lpt_int_enable(LPT,LPT_INTSRC_PEND);      //使能LPT PEND中断
csi_lpt_start(LPT);
....
csi_pm_enter_sleep(_LOW_POWER_MODE_);        //进入低功耗模式

```

详见lp\_lpt\_wakeup\_demo函数

### 6.1.6.2 lp\_exi\_wakeup\_demo

#### 1、函数位置

lowpower\_demo.c

#### 2、功能描述

exi唤醒低功耗功能示例。系统进入低功耗模式（SLEEP或者DEEP\_SLEEP），通过外部管脚下降沿唤醒唤醒。

#### 3、参考用法

```

csi_gpio_set_mux(GPIOB,PB1,PB1_INPUT);      //PB1 输入,作为低功耗唤醒源
csi_gpio_pull_mode(GPIOB,PB1, GPIO_PULLUP); //PB1 上拉
csi_gpio_int_enable(GPIOB,PB1);              //PB1中断使能
csi_gpio_irq_mode(GPIOB,PB1,EXI_GRP1, GPIO_IRQ_FALLING_EDGE); //PB1 下降沿产生中断
csi_gpio_vic_irq_enable(EXI_GRP1, ENABLE);    //PB1 中断使能, 选择中断组1
....
csi_pm_enter_sleep(_LOW_POWER_MODE_);        //进入低功耗模式

```

详见lp\_exi\_wakeup\_demo函数

### 6.1.6.3 lp\_lvd\_wakeup\_demo

#### 1、函数位置

lowpower\_demo.c

#### 2、功能描述

LVD唤醒低功耗功能示例。系统进入低功耗模式（SLEEP或者DEEP\_SLEEP），通过LVD掉电到3V以下唤醒唤醒。

#### 4、参考用法

```

csi_pm_set_wakeup_src(SRC_WKUP_LVD, ENABLE); //配置唤醒源

//lvd配置
csi_set_lvd(LVD_INTF,LVD_30);                //VDD掉电到3.0V以下即触发LVD中断
csi_lvd_lvr_enable();                         //LVD使能
....
csi_pm_enter_sleep(_LOW_POWER_MODE_);        //进入低功耗模式

```

详见lp\_lvd\_wakeup\_demo函数

#### 6.1.6.4 lp\_rtc\_wakeup\_demo

##### 1、函数位置

lowpower\_demo.c

##### 2、功能描述

RTC唤醒低功耗功能示例。系统进入低功耗模式（SLEEP或者DEEP\_SLEEP），通过RTC定时500ms唤醒唤醒。

##### 5、参考用法

```
csi_pm_set_wakeup_src(SRC_WKUP_RTC_ENABLE);           //配置唤醒源

//RTC 配置
tRtcConfig.eClkSrc = RTC_CLKSRC_ISOSC;                //选择时钟源
tRtcConfig.eFmt = RTC_MODE_24FMT;                    //选择时间模式
csi_rtc_init(RTC, &tRtcConfig);                        //初始化RTC
csi_rtc_set_timer(RTC, RTC_TIMER_0_5S);               //每0.5s进行一次中断
csi_rtc_start(RTC);                                   //启动RTC

csi_pm_enter_sleep(_LOW_POWER_MODE_);                 //进入低功耗模式
```

详见lp\_rtc\_wakeup\_demo函数

#### 6.1.7 BT

##### 6.1.7.1 bt\_timer\_demo

##### 1、函数位置

bt\_demo.c

##### 2、功能描述

BT1基本定时功能示例。

##### 3、参考用法

```
csi_bt_time_config_t tTimConfig;                      //BT timer 结构体

tTimConfig.wTimeVal = 1000;                           //BT定时值 = 1000us
tTimConfig.eRunMode = BT_RUN_CONT;                    //BT计数器工作模式
csi_bt_timer_init(BT1, &tTimConfig);                  //初始化BT

csi_bt_int_enable(BT1,BT_INTSRC_PEND);                //使能BT PEND中断
csi_bt_start(BT1);                                    //启动BT定时器
```

详见bt\_timer\_demo函数。

### 6.1.7.2 bt\_pwm\_demo

#### 1、函数位置

bt\_demo.c

#### 2、功能描述

BT1 PWM输出功能示例。可配置PWM周期和占空比，占空比只能为整数。

#### 3、参考用法

```
csi_bt_pwm_config_t tPwmConfig;           //BT PWM输出参数初始化配置结构体

csi_gpio_set_mux(GPIOC, PC11, PC11_BT1_OUT); //PC11 作为BT1 PWM输出引脚

tPwmConfig.eIdleLevel = BT_PWM_IDLE_HIGH;    //PWM 输出空闲电平
tPwmConfig.eStartLevel = BT_PWM_START_HIGH;  //PWM 输出起始电平
tPwmConfig.eRunMode    = BT_RUN_CONT;        //BT计数器工作模式：连续/单次
tPwmConfig.byDutyCycle = 90;                 //PWM 输出占空比(0 < DutyCycle < 100)
tPwmConfig.wFreq       = 50;                 //PWM 输出频率
csi_bt_pwm_init(BT1, &tPwmConfig);           //初始化BT1 PWM输出

csi_bt_start(BT1);                           //启动BT1
```

详见bt\_pwm\_demo函数。

### 6.1.7.3 bt\_sync\_trg\_count\_demo

#### 1、函数位置

bt\_demo.c

#### 2、功能描述

BT1触发计数值加1功能示例。可用外部中断EXI事件通过ETCB模块触发BT0 Sync Port2，BT1计数加1。

#### 3、参考用法

```
//PB1配置为外部中断功能，选择中断组1
.....
//BT1 初始化
tTimConfig.wTimeVal = 100;           //BT定时值 = 100us
tTimConfig.eRunMode = BT_RUN_CONT;    //BT计数器工作模式
csi_bt_timer_init(BT1,&tTimConfig);    //BT1 定时
csi_bt_set_sync(BT1, BT_SYNCIN2, BT_SYNC_CONT, BT_TRG_AUTOAREARM); //外部触发BT1计数(SYNCIN2)
csi_bt_sync_enable(BT1, BT_SYNCIN2);  //BT1 同步输入2使能
csi_bt_start(BT1);                   //启动定时器
.....
//ETCB配置，设置目标事件(EXI_TRGOUT1)和源事件(BT1 SYNCIN2)： EXI_TRGOUT1 -> ETCB -> BT1 SYNCIN2
.....
csi_etb_init(); //ETCB初始化
```

详见bt\_sync\_trg\_count\_demo函数。

#### 6.1.7.4 bt\_trg\_out\_demo

##### 1、函数位置

bt\_demo.c

##### 2、功能描述

BT0触发输出功能示例。可用BT0 PEND事件通过ETCB模块触发BT1 Sync Port0，即BT1启动。

##### 3、参考用法

```
//BT0初始化
tTimConfig.wTimeVal = 10000;           //BT定时值 = 1000us
tTimConfig.eRunMode = BT_RUN_CONT;     //BT计数器工作模式
csi_bt_timer_init(BT0,&tTimConfig);    //BT0 定时
csi_bt_set_evtrg(BT0, BT_TRGSRG_PEND); //BT0 PEND事件触发输出
csi_bt_evtrg_enable(BT0);              //BT0 事件触发输出使能器

.....

csi_bt_pwm_init(BT1, &tPwmCfg);         //初始化BT1 PWM输出
//外部触发bt启动(SYNCIN0)
csi_bt_set_sync(BT1, BT_SYNCIN0, BT_SYNC_ONCE, BT_TRG_AUTOAREARM);
csi_bt_sync_enable(BT1, BT_SYNCIN0);    //BT1 同步输入2使能

.....

//ETCB配置，设置目标事件(BT0 PEND)和源事件(BT1 SYNCIN0)：BT0 PEND -> ETCB -> BT1 SYNCIN0
.....

csi_etb_init(); //ETCB初始化
```

详见bt\_trg\_out\_demo函数。

#### 6.1.8 CNTA

##### 6.1.8.1 cnta\_timer\_demo

##### 1、函数位置

cnta\_demo.c

##### 2、功能描述

CA0基本定时功能示例。

##### 3、参考用法

```
csi_cnta_timer_config_t tTimerCfg;

tTimerCfg.eClkDiv = CNTA_CK_DIV8;      //时钟8分频
tTimerCfg.wTime = 1000;                 //周期1000us
tTimerCfg.eRunMode = CNTA_RUN_CONT;     //重复模式
csi_cnta_timer_init(CA0, &tTimerCfg);   //初始化CountA
```

```
csi_cnta_int_enable(CA0, CNTA_INTSRC_PENDL);    //使能cnta PENDL中断
csi_cnta_start(CA0);                            //启动CountA
```

详见cnta\_timer\_demo函数。

### 6.1.8.2 cnta\_pwm\_demo

#### 1、函数位置

cnta\_demo.c

#### 2、功能描述

CA0 PWM输出功能示例。可配置PWM周期和占空比，占空比只能为整数。

#### 3、参考用法

```
csi_gpio_set_mux(GPIOA, PA10,PA10_CNTA_BUZ);    //PA10作为CA0 PWM输出引脚
.....
//初始化tPwmCfg, 设置PWM电平参数, 周期和占空比
.....
csi_cnta_pwm_init(CA0,&tPwmCfg);                  //初始化CountA
csi_cnta_start(CA0);                             //启动CountA
```

详见cnta\_pwm\_demo函数。

### 6.1.8.3 cnta\_envelope\_demo

#### 1、函数位置

cnta\_demo.c

#### 2、功能描述

CounterA 和BT0搭配包络输出PWM示例

#### 3、参考用法

```
//PB1配置为外部中断功能, 选择中断组1
csi_gpio_set_mux(GPIOA, PA0, PA0_BT0_OUT);      //PA0 作为BT0 PWM输出引脚
.....
csi_bt_pwm_init(BT0, &tBTPwmCfg);               //初始化BT0 PWM输出
csi_bt_start(BT0);                              //启动BT0

csi_gpio_set_mux(GPIOA, PA10,PA10_CNTA_BUZ);    //PA10作为CA0 PWM输出引脚
.....
csi_cnta_pwm_init(CA0,&tPwmCfg);                 //初始化CountA
csi_cnta_start(CA0);                             //启动CountA
```

详见cnta\_envelope\_demo函数。

## 6.1.9 GPTA

### 6.1.9.1 gpta\_timer\_demo

#### 1、函数位置

gpta\_demo.c

#### 2、功能描述

GPTA基本定时功能示例。

#### 3、参考用法

```
csi_gpta_time_config_t tTimConfig;

tTimConfig.wTimeVal = 10000;           //GPTA定时值 = 1000us
tTimConfig.eRunMode = GPTA_RUN_CONT;   //GPTA计数器工作模式
csi_gpta_timer_init(GPTA0, &tTimConfig); //初始化GPTA0, 定时10000us; GPTA定时, 默认采用向上计数
csp_gpta_int_enable(GPTA0, GPTA_INT_PEND); //使能GPTA0 PEND中断
csi_gpta_start(GPTA0);
```

详见gpta\_timer\_demo函数。

### 6.1.9.2 gpta\_capture\_demo

#### 1、函数位置

gpta\_demo.c

#### 2、功能描述

GPTA捕获功能示例。sync2 sync3区分, 实现2次捕获, 由外部中断EXI事件通过ETCB模块触发GPTA SYNCIN3, 产生捕获。

#### 3、参考用法

```
//PA3上升沿和下降沿配置为外部中断功能, 选择中断组3和中断组16
csi_exi_set_evtrg(EXI_TRGOUT0, EXI_TRGSRC_GRP3, 1); //EXI3触发EXI_TRGOUT0
csi_exi_evtrg_enable(EXI_TRGOUT0);
.....
csi_exi_set_evtrg(EXI_TRGOUT5, EXI_TRGSRC_GRP16, 1); //EXI16触发EXI_TRGOUT5
csi_exi_evtrg_enable(EXI_TRGOUT5);
//ETCB配置, 设置目标事件(GPTA0_SYNCIN3)和源事件(EXI_TRGOUT5): EXI_TRGOUT5 -> ETCB ->
GPTA0_SYNCIN3
.....
csi_etb_init(); //初始化两个ETCB
.....
//初始化tCapConfig, 设置GPTA0的捕获参数
.....
csi_gpta_capture_init(GPTA0, &tCapConfig);

//使能SYNCIN2外部触发
csi_gpta_set_sync(GPTA0, GPTA_SYNCIN2, GPTA_SYNC_CONT, GPTA_AUTO_REARM_ZRO);
```

```
csi_gpta_sync_enable(GPTA0, GPTA_SYNCIN2);

//使能SYNCIN3外部触发
csi_gpta_set_sync(GPTA0, GPTA_SYNCIN3, GPTA_SYNC_CONT, GPTA_AUTO_REARM_ZRO);
csi_gpta_sync_enable(GPTA0, GPTA_SYNCIN3);

csi_gpta_start(GPTA0);           //启动GPTA0
```

详见gpta\_capture\_demo函数。

### 6.1.9.3 gpta\_pwm\_demo

#### 1、函数位置

gpta\_demo.c

#### 2、功能描述

GPTA波形输出功能示例。可配置PWM周期和占空比，占空比只能为整数，如果有更细致的配置要求可以通过csi\_gpta\_change\_ch\_duty()实现。

#### 3、参考用法

```
csi_gpio_set_mux(GPIOA, PA0, PA0_GPTA0_CHA);    //PA0作为PWM输出引脚
.....
//初始化tPwmCfg, 设置PWM周期和占空比
.....
csi_gpta_wave_init(GPTA0, &tPwmCfg);
.....
//初始化tGptachannelCfg, 设置PWM波形输出
.....
csi_gpta_set_channel (GPTA0,& tGptachannelConfig,GPTA_CHANNEL_1);
csi_gpta_start(GPTA0);
csi_gpta_update_ch_duty (GPTA0,GPTA_COMPA, 20);    //修改PWM1占空比为20%
```

详见gpta\_pwm\_demo函数。

### 6.1.9.4 gpta\_pwm\_waveform\_demo

#### 1、函数位置

gpta\_demo.c

#### 2、功能描述

GPTA波形强制输出demo，包含一次性强制性输出和连续强制。

#### 3、参考用法

```
csi_gpio_set_mux(GPIOA, PA0, PA0_GPTA0_CHA);    //PB00作为PWM输出引脚
.....
//初始化tPwmConfig, 设置PWM周期和占空比
```

```

.....
csi_gpta_pwm_init (GPTA0, &tPwmConfig);
.....
//初始化tGptachannelConfig, 设置PWM波形输出
.....
csi_gpta_set_channel(GPTA0, &tGptachannelConfig, GPTA_CHANNEL_1);
csi_gpta_start(GPTA0);
//一次强制输出
csi_gpta_set_aqosf(GPTA0,GPTA_CHANNEL_1,GPTA_ACT_HI);
//连续强制输出
csi_gpta_set_aqcsf_loadmode(GPTA0, GPTA_AQCSF_LOAD_NOW);
csi_gpta_set_aqcsf(GPTA0, GPTA_CHANNEL_1, GPTA_AQCSF_L);
csi_gpta_set_aqcsf(GPTA0, GPTA_CHANNEL_1, GPTA_AQCSF_NONE);

```

详见gpta\_pwm\_waveform\_demo函数。

### 6.1.9.5 gpta\_reglk\_demo

#### 1、函数位置

gpta\_demo.c

#### 2、功能描述

GPTA链接代码实例。通过GPTA1链接GPTA0,实现波形的输出。

#### 3、参考用法

```

csi_gpio_set_mux(GPIOA, PA0, PA0_GPTA0_CHA); //GPTA0 CHA 输出
csi_gpio_set_mux(GPIOA, PA2, PA2_GPTA1_CHA); //GPTA1 CHA 输出
.....
//初始化tPwmConfig, 设置PWM周期和占空比
.....
csi_gpta_pwm_init(GPTA0, &tPwmConfig);
.....
//初始化tGptachannelCf, 设置PWM波形输出
.....
csi_gpta_set_channel(GPTA0, &tGptachannelConfig, GPTA_CHANNEL_1);
.....
//初始化tPwmConfig, 设置PWM周期和占空比
.....
csi_gpta_pwm_init(GPTA1, &tPwmConfig);
.....
//初始化tGptachannelCf, 设置PWM波形输出
.....
csi_gpta_set_channel(GPTA1, &tGptachannelConfig, GPTA_CHANNEL_1);
.....
//链接配置, GPTA1链接GPTA0
csi_gpta_set_reglk(GPTA0, &tReglkConfig);

```



```
csi_gpta_start(GPTA1);
//链接,改变GPTA1的波形, GPTA0也会随着改变
csi_gpta_update_ch_duty(GPTA1,GPTA_COMPA, 50);
```

详见gpta\_reglk\_demo函数。

## 6.1.10 GPTB

### 6.1.10.1 gptb\_timer\_demo

#### 1、函数位置

gptb\_demo.c

#### 2、功能描述

GPTB基本定时功能示例。

#### 3、参考用法

```
csi_gptb_time_config_t tTimConfig;           //GPTB timer参数配置结构体

tTimConfig.wTimeVal = 1000;                  //GPTB定时值 = 1000us
tTimConfig.eRunMode = GPTB_RUN_CONT;         //GPTB计数器工作模式
csi_gptb_timer_init(GPTB0, &tTimConfig);     //初始化GPTB

csi_gptb_int_enable(GPTB0,GPTB_INTSRC_PEND);
csi_gptb_start(GPTB0);                       //启动定时器
```

详见gptb\_timer\_demo函数。

### 6.1.10.2 gptb\_capture\_demo

#### 1、函数位置

gptb\_demo.c

#### 2、功能描述

GPTB捕获功能示例。sync2 sync3区分, 实现2次捕获, 由外部中断EXI事件通过ETCB模块触发GPTB SYNCIN3, 产生捕获。

#### 3、参考用法

```
//PA1上升沿和下降沿配置为外部中断功能, 选择中断组3和中断组16
csi_exi_set_evtrg(EXI_TRGOUT1, EXI_TRGSRC_GRP3, 1);           //EXI1触发EXI_TRGOUT1
csi_exi_evtrg_enable(EXI_TRGOUT1);

.....
csi_exi_set_evtrg(EXI_TRGOUT5, EXI_TRGSRC_GRP16, 1);          //EXI16触发EXI_TRGOUT5
csi_exi_evtrg_enable(EXI_TRGOUT5);
//ETCB配置, 设置目标事件(GPTB0_SYNCIN3)和源事件(EXI_TRGOUT5): EXI_TRGOUT5 -> ETCB ->
GPTA0_SYNCIN3
.....
```

```

csi_etb_init(); //初始化两个ETCB
.....
//初始化tCapConfig, 设置GPTB0的捕获参数
.....
csi_gptb_capture_init(GPTB0, &tCapConfig);

//使能SYNCIN2外部触发
csi_gptb_set_sync(GPTB0, GPTB_SYNCIN2, GPTB_SYNC_CONT, GPTB_AUTO_REARM_ZRO);
csi_gptb_sync_enable(GPTB0, GPTB_SYNCIN2);

//使能SYNCIN3外部触发
csi_gptb_set_sync(GPTB0, GPTB_SYNCIN3, GPTB_SYNC_CONT, GPTB_AUTO_REARM_ZRO);
csi_gptb_sync_enable(GPTB0, GPTB_SYNCIN3);

csi_gptb_start(GPTB0); //启动GPTB0

```

详见gptb\_capture\_demo函数。

### 6.1.10.3 gptb\_pwm\_demo

#### 1、函数位置

gptb\_demo.c

#### 2、功能描述

GPTB波形输出功能示例。可配置PWM周期和占空比，占空比只能为整数，如果有更细致的配置要求可以通过csi\_gptb\_prdr\_cmp\_update()实现。

#### 3、参考用法

```

csi_gpio_set_mux(GPIOC, PC13, PC13_GPTB0_CHAX) //PC13作为PWM输出引脚
.....
//初始化tPwmCfg, 设置PWM周期和占空比
.....
csi_gptb_pwm_init(GPTB0, &tPwmCfg);
.....
//初始化channel, 设置PWM波形输出
.....
csi_gptb_set_channel(GPTB0, &tPwmChCfg, GPTB_CHANNEL_1);
csi_gptb_start(GPTB0)
csi_gptb_pwm_update(GPTB0, 5000, 20, GPTB_COMPA); //修改PWM参数为5KHz/20%
csi_gptb_cmp_update(GPTB0, GPTB_COMPA, 1000); //修改CMPA比较值为1000
csi_gptb_prdr_update(GPTB0, 2000); //修改PWM周期为2000

```

详见gptb\_pwm\_demo函数。

### 6.1.10.4 gptb\_pwm\_dz\_demo

#### 1、函数位置

gptb\_demo.c

## 2、功能描述

GPTB波形输出+死区控制功能示例。可配置PWM波形和死区时间。

## 3、参考用法

```
csi_gpio_set_mux(GPIOC, PC13, PC13_GPTB0_CHAX);           //PC13作为PWM输出引脚
.....
//初始化tPwmCfg, 设置PWM周期和占空比
.....
csi_gptb_pwm_init(GPTB0, &tPwmCfg);
.....
//初始化channel, 设置PWM波形输出
.....
csi_gptb_set_channel(GPTB0, &tPwmChCfg, GPTB_CHANNEL_1);
.....
//初始化tGptbDeadZoneTime, 设置死区时间与边沿
.....
csi_gptb_deadzone_init(GPTB0, &tDeadZoneCfg, GPTB_CHANNEL_1)
csi_gptb_start(GPTB0);
```

详见gptb\_pwm\_dz\_demo函数。

### 6.1.10.5 gptb\_pwm\_dz\_em\_demo

## 1、函数位置

gptb\_demo.c

## 2、功能描述

EPT波形输出+死区控制+紧急模式功能示例。可配置PWM波形和死区时间，以及紧急模式的处理。

## 3、参考用法

```
csi_gpio_set_mux(GPIOC, PC13, PC13_GPTB0_CHAX);           //PC13作为PWM输出引脚
.....
//初始化tPwmCfg, 设置PWM周期和占空比
.....
csi_gptb_pwm_init(GPTB0, &tPwmCfg);
.....
//初始化channel, 设置PWM波形输出
.....
csi_gptb_set_channel(GPTB0, &tPwmChCfg, GPTB_CHANNEL_1);
.....
//初始化tGptbDeadZoneTime, 设置死区时间与边沿
.....
csi_gptb_set_channel(GPTB0, &tPwmChCfg, GPTB_CHANNEL_1);
.....
//初始化tGptbEmergencyCfg, 设置紧急模式的输入源与处理方式
```

```

.....
csi_gptb_emergency_init(GPTB0,&tEmCfg);
.....
csi_gptb_start(GPTB0);

```

详见gpt\_pwm\_dz\_em\_demo函数。

### 6.1.10.6 gptb\_reglk\_demo

#### 2、函数位置

gptb\_demo.c

#### 2、功能描述

GPTB链接代码实例,通过GPTB0链接GPTB1,实现波形的输出。GPTB1启动会出发GPTB0同事启动,且更新GPTB1 PWM参数时,GPTB0的PWM参数也会随之更新。

#### 3、参考用法

```

csi_gpio_set_mux(GPIOC, PC13, PC13_GPTB0_CHAX);           //初始化PC13为CHAX
csi_gpio_set_mux(GPIOC, PC14, PC14_GPTB1_CHAX);           //初始化PC14为CHAY
.....
//初始化tPwmCfg, 设置PWM周期和占空比
.....
csi_gptb_pwm_init(GPTB0, &tPwmCfg);
csi_gptb_pwm_init(GPTB1, &tPwmCfg);
.....
//初始化channel, 设置PWM波形输出
.....
csi_gptb_set_channel(GPTB0, &tPwmChCfg, GPTB_CHANNEL_1);
csi_gptb_set_channel(GPTB1, &tPwmChCfg, GPTB_CHANNEL_1);
.....
//配置GPTB0链接到GPTB1
.....
csi_gptb_set_reglk(GPTB0, &tReglkCfg);
.....
csi_gptb_start(GPTB1);

```

详见gptb\_reglk\_demo函数。

### 6.1.11 RTC

#### 6.1.11.1 rtc\_set\_time\_demo

#### 1、函数位置

rtc\_demo.c

#### 2、功能描述

设置rtc时间的示例代码: 包括时钟源、时间模式、时间设置, 当前时间回读函数。

### 3、参考用法

```

.....
//初始化tRtcConfig, 选择时钟源, 选择时间模式
.....
csi_rtc_init(RTC, &tRtcConfig);      //初始化RTC参数配置
.....
csi_rtc_set_current_time (RTC, &tRtcTime);    //设置时间
csi_rtc_start(RTC);                    //RTC开始计时
csi_rtc_get_time(RTC, &tRtcTimeRdbk); //回读当前时间

```

详见rtc\_set\_time\_demo函数。

#### 6.1.11.2 rtc\_alarm\_demo

##### 1、函数位置

rtc\_demo.c

##### 2、功能描述

设置rtc时间的示例代码：包括时钟源、时间模式、时间设置，当前时间回读函数。

### 3、参考用法

```

.....
//初始化tRtcConfig, 选择时钟源, 选择时间模式
.....
csi_rtc_init(RTC, &tRtcConfig);      //初始化RTC参数配置
.....
csi_rtc_set_current_time (RTC, &tRtcTime);    //设置时间
csi_rtc_start(RTC);                    //RTC开始计时
csi_rtc_set_alarm(RTC, RTC_ALMA, tAlmA.byAlmMode, &tAlmTime); //设置闹钟A
.....
//如果闹钟时间没有到, 每秒打印一次当前时间和距离闹钟的时间
.....
csi_rtc_cancel_alarm(RTC, RTC_ALMA); //取消闹钟, 保持原闹钟时间

```

详见rtc\_alarm\_demo函数。

#### 6.1.11.3 rtc\_timer\_demo

##### 1、函数位置

rtc\_demo.c

##### 2、功能描述

如何将RTC当做一个简单timer来使用

### 3、参考用法

```

.....

```

```
//初始化tRtcConfig, 选择时钟源, 选择时间模式
csi_rtc_init(RTC, &tRtcConfig);           //初始化RTC参数配置
ccsi_rtc_set_timer(RTC, RTC_TIMER_1S);    //每1s进行一次中断
csi_rtc_start(RTC);                       //RTC开始计时
```

详见rtc\_timer\_demo函数。

#### 6.1.11.4 rtc\_etcb\_bt\_pwm\_demo

##### 1、函数位置

rtc\_demo.c

##### 2、功能描述

RTC通过ETCB 触发 BT0 输出PWM。

##### 3、参考用法

```
.....
//初始化tRtcConfig, 选择时钟源, 选择时间模式
.....
csi_rtc_init(RTC, &tRtcConfig);           //初始化RTC参数配置
.....
csi_rtc_set_current_time(RTC, &tRtcTime); //设置时间
csi_rtc_start(RTC);                       //RTC开始计时
.....
csi_rtc_set_alarm(RTC, RTC_ALMB, tAlmA.byAlmMode, &tAlmTime); //设置闹钟B
csi_rtc_set_evtrg(RTC, RTC_TRGOUT1, RTC_TRGOUT_ALRMB, 0);    //RTC TRGEV1 闹铃B到时产生trigger event
csi_rtc_evtrg_enable(RTC, RTC_TRGOUT1); //RTC触发使能
.....
//配置ETCB。ETCB_RTC_TRGOUT1作为触发源, ETCB_BT0_SYNCIN0 同步输入作为目标事件
csi_etcb_ch_init(ch,&tEtbConfig1);
.....
//配置BT0 作为触发目标事件
csi_bt_pwm_init(BT0, &tPwmCfg);           //初始化BT0 PWM输出
csi_bt_set_sync(BT0, BT_SYNCIN0, BT_SYNC_ONCE, BT_TRG_AUTOAREARM); //外部触发bt启动(SYNCIN0)
csi_bt_sync_enable(BT0, BT_SYNCIN0); //BT0 同步使能
```

详见rtc\_etcb\_bt\_pwm\_demo函数。

#### 6.1.12 ADC

##### 6.1.12.1 adc\_samp\_one-shot\_demo

##### 1、函数位置

adc\_demo.c

##### 2、功能描述

ADC单次采样功能示例。启动后进行整个序列的采样，采样完成后停止。

##### 3、参考用法

```
csi_gpio_set_mux(GPIOC,PC13, PC13_ADC_INA0);           // PC13作为ADC输入引脚
.....
//初始化tAdcConfig, 配置ADC参数, 单次采样
.....
csi_adc_init(ADC0, &tAdcConfig);           //初始化ADC参数配置
csi_adc_set_seq_num(ADC0,iChnlNum); //配置ADC总采样通道个数
csi_adc_set_seqx(ADC0,0,ADC_INA0,ADC_CV_COUNT_1,ADC_AVG_COF_1,ADCSYNC_NONE); //配置ADC采样通道0
csi_adc_start(ADC0);
```

详见adc\_samp\_oneshot\_demo函数。

### 6.1.12.2 adc\_samp\_continuous\_demo

#### 1、函数位置

adc\_demo.c

#### 3、功能描述

ADC连续采样功能示例。启动后进行整个序列的采样，采样完成后继续从序列第一个通道开始，如此循环。

#### 3、参考用法

```
csi_pin_set_mux(PC13, PC13_ADC_INA0); // PC13作为ADC输入引脚
.....
//初始化tAdcConfig, 配置ADC参数, 连续采样
.....
csi_adc_init(ADC0, &tAdcConfig);           //初始化ADC参数配置
csi_adc_set_seq_num(ADC0,iChnlNum); //配置ADC总采样通道个数
csi_adc_set_seqx(ADC0,0,ADC_INA0,ADC_CV_COUNT_1,ADC_AVG_COF_1,ADCSYNC_NONE); //配置ADC采样通道0
csi_adc_start(ADC0);
```

详见adc\_samp\_continuous\_demo函数。

### 6.1.12.3 adc\_samp\_oneshot\_int\_demo

#### 1、函数位置

adc\_demo.c

#### 2、功能描述

ADC单次采样中断模式功能示例。启动后进行整个序列的采样，采样完成后停止。

#### 3、参考用法

```
csi_pin_set_mux(PC13, PC13_ADC_INA0); // PC13作为ADC输入引脚
.....
//初始化tAdcConfig, 配置ADC参数, 单次采样, 中断模式
```

```

.....
csi_adc_init(ADC0, &tAdcConfig);           //初始化ADC参数配置
csi_adc_set_seq_num(ADC0,iChnlNum);        //配置ADC总采样通道个数
csi_adc_set_seqx(ADC0,0,ADC_INA0,ADC_CV_COUNT_1,ADC_AVG_COF_1,ADCSYNC_NONE); //配置ADC采样通道0
csi_adc_int_enable(ADC0, ADC_INTSRC_SEQ0); //设置采样通道0的转换完成中断
csi_adc_start(ADC0);

```

详见adc\_samp\_oneshot\_int\_demo函数。

#### 6.1.12.4 adc\_samp\_continuous\_int\_demo

##### 1、函数位置

adc\_demo.c

##### 2、功能描述

ADC连续采样中断模式功能示例。启动后进行整个序列的采样，采样完成后继续从序列第一个通道开始，如此循环。

##### 3、参考用法

```

csi_pin_set_mux(PC13, PC13_ADC_INA0); // PC13作为ADC输入引脚
.....
//初始化tAdcConfig, 配置ADC参数, 连续采样, 中断模式
.....
csi_adc_init(ADC0, &tAdcConfig);           //初始化ADC参数配置
csi_adc_set_seq_num(ADC0,iChnlNum);        //配置ADC总采样通道个数
csi_adc_set_seqx(ADC0,0,ADC_INA0,ADC_CV_COUNT_1,ADC_AVG_COF_1,ADCSYNC_NONE); //配置ADC采样通道0
csi_adc_int_enable(ADC0, ADC_INTSRC_SEQ0); //设置采样通道0的转换完成中断
csi_adc_start(ADC0);

```

详见adc\_samp\_continuous\_int\_demo函数。

#### 6.1.13 FVR

##### 6.1.13.1 fvr\_output\_demo

##### 1、函数位置

fvr\_demo.c

##### 2、功能描述

配置FVR电平，并通过管脚输出

##### 3、参考用法

```

csi_gpio_set_mux(GPIOB,PB7, PB7_FVROUT); // 设置PB7为FVR输出
//配置FVR时钟
.....

```



```
csi_fvr_init(FVR,&tFvrConfig);
csi_fvr_enable(FVR);           //使能FVR;
```

详见fvr\_output\_demo函数。

### 6.1.13.2 buf\_output\_demo

#### 1、函数位置

fvr\_demo.c

#### 2、功能描述

配置BUF(intervref)电平，并通过管脚输出。

#### 3、参考用法

```
csi_gpio_set_mux(GPIOD,PD4, PD4_INPUT);           //设置BUF输入管脚PD4
csi_gpio_set_mux(GPIOB,PB8, PB8_BUF);             //设置BUF输出管脚PB8
csi_set_buf(FVR,FVR_BUF_LEVEL_EXTERNAL);          //设置buf的输出为外部管脚输入
csi_fvr_enable(FVR);                               //使能fvr模块
```

详见buf\_output\_demo函数。

## 6.1.14 DAC

### 6.1.14.1 dac\_demo

#### 1、函数位置

dac\_demo.c

#### 2、功能描述

配置DAC的初始电平码值，设置时钟分频

#### 3、参考用法

```
.....
csi_dac_init(DAC0, &tDacConfig); // 配置DAC 分频、参考电平选择等
csi_dac_enbale(DAC0);
csi_dac_start(DAC0);           // 开启DAC
```

详见dac\_demo函数。

### 6.1.14.2 dac\_sync\_demo

#### 3、函数位置

dac\_demo.c

#### 4、功能描述

配置DAC的中断及触发模式

#### 3、参考用法

```

.....
csi_dac_init(DAC0, &tDacConfig);           // 配置DAC 分频、参考电平选择等
csi_dac_enbale(DAC0);

.....
csi_dac_int_enable(DAC0, DAC_INTSRC_EOC);   //使能EOC中断
.....
csi_dac_sync_enable(DAC0, DAC_SYNCIN0);    //开启DAC_SYNCIN0触发
.....
csi_dac_set_step(DAC0, 409);               //设置触发增减值
csi_dac_start(DAC0);                      // 开启DAC

```

详见dac\_sync\_demo函数。

## 6.1.15 CRC

### 6.1.15.1 crc32\_demo

#### 1、函数位置

crc\_demo.c

#### 2、功能描述

CRC-32模式的CRC计算。

#### 3、参考用法

```

csi_crc_init(); //CRC模块初始化
//进行CRC-32模式计算，种子值0xffffffff，byTransData数组前3个数据，数据长度3个字节
csi_crc32_be(0xffffffff, byTransData, 3);

```

详见crc32\_demo函数。

### 6.1.15.2 crc16\_ccitt\_demo

#### 3、函数位置

crc\_demo.c

#### 4、功能描述

CRC-16/CCITT模式的CRC计算。

#### 3、参考用法

```

csi_crc_init(); //CRC模块初始化
//进行CRC-16/CCITT模式计算，种子值0x00，byTransData数组前16个数据，数据长度16个字节
csi_crc16_ccitt(0x00, byTransData, 16);

```

详见crc16\_ccitt\_demo函数。

### 6.1.15.3 crc16\_demo

#### 5、函数位置

crc\_demo.c

## 6、功能描述

CRC-16模式的CRC计算。

## 3、参考用法

```
csi_crc_init();    //CRC模块初始化
//进行CRC-16模式计算，种子值0x00,byTransData数组前5个数据，数据长度5个字节
csi_crc16(0x00, byTransData, 5);
```

详见crc16\_demo函数。

### 6.1.15.4 crc16\_xmodem\_demo

## 7、函数位置

crc\_demo.c

## 8、功能描述

CRC-16 XMODEM模式的CRC计算。

## 3、参考用法

```
csi_crc_init();    //CRC模块初始化
//进行CRC-16 XMODEM模式计算，种子值0x00, byTransData数组前3个数据，数据长度3个字节
csi_crc16_itu(0x00, byTransData, 3);
```

详见crc16\_xmodem\_demo函数。

### 6.1.16 ETCB

#### 6.1.16.1 exi\_etcb\_bt\_start\_demo

## 1、函数位置

etcb\_demo.c

## 2、功能描述

EXI通过ETCB触发BT启动的demo。

## 3、参考用法

```
//PB1配置为外部中断功能，选择中断组1
.....
csi_exi_set_evtrg(EXI_TRGOUT1, EXI_TRGSRC_GRP1, 0);    //EXI 触发输出配置
csi_exi_evtrg_enable(EXI_TRGOUT1);                    //使能EXI_TRGOUT1触发输出
//初始化BT0，外部触发BT0启动(SYNCIN0),使用PEND事件REARM
.....
csi_bt_timer_init(BT0,&tTimConfig);                    //BT0 定时
```

```
csi_bt_set_sync(BT0, BT_SYNCIN0, BT_SYNC_CONT, BT_TRG_SYCAREARM);    //外部触发bt0启动(SYNCIN0)
csi_bt_sync_enable(BT0, BT_SYNCIN0);    //BT0同步输入2使能
//ETCB配置，设置目标事件(BT0 SYNCIN0)和源事件(EXI TRGOUT1): EXI TRGOUT1 -> ETCB -> BT0 SYNCIN0
.....
csi_etb_init();
.....
```

详见exi\_etcb\_bt\_start\_demo函数。

### 6.1.16.2 exi\_etcb\_bt\_stop\_demo

#### 3、函数位置

etcb\_demo.c

#### 4、功能描述

EXI通过ETCB触发BT启动的demo。

#### 3、参考用法

```
//PB1配置为外部中断功能，选择中断组1
.....
csi_exi_set_evtrg(EXI_TRGOUT1, EXI_TRGSRC_GRP1, 0);    //EXI 触发输出配置
csi_exi_evtrg_enable(EXI_TRGOUT1);    //使能EXI_TRGOUT1触发输出
//初始化BT0，外部触发BT0停止(SYNCIN1),使用PEND事件REARM
.....
csi_bt_timer_init(BT0,&TimConfig);    //BT0 定时
csi_bt_set_sync(BT0, BT_SYNCIN1, BT_SYNC_CONT, BT_TRG_SYCAREARM);    //外部触发bt0启动(SYNCIN1)
csi_bt_sync_enable(BT0, BT_SYNCIN1);    //BT0同步输入2使能
csi_bt_start(BT0);    //启动BT0
//ETCB配置，设置目标事件(BT0 SYNCIN0)和源事件(EXI TRGOUT1): EXI TRGOUT1 -> ETCB -> BT0 SYNCIN0
.....
csi_etb_init();
.....
```

详见exi\_etcb\_bt\_stop\_demo函数。

### 6.1.16.3 exi\_etcb\_adc\_samp\_demo

#### 5、函数位置

etcb\_demo.c

#### 6、功能描述

EXI通过ETCB触发ADC采样的demo。

#### 3、参考用法

```

//PB1配置为外部中断功能，选择中断组1
.....
csi_ex_i_set_evtrg(EXI_TRGOUT1, EXI_TRGSRC_GRP1, 0);           //EXI 触发输出配置
csi_ex_i_evtrg_enable(EXI_TRGOUT1);                           //使能EXI_TRGOUT1触发输出
//初始化ADC0，外部触发ADC采样(SYNCIN0)
.....
csi_adc_init(ADC0, &tAdcConfig);                               //初始化ADC参数配置
csi_adc_set_seq_num(ADC0, iChnlNum);                           //配置ADC总采样通道个数
csi_adc_set_seqx(ADC0, 0, ADC_INA0, ADC_CV_COUNT_1, ADC_AVG_COF_1, ADCSYNC_IN0); //配置ADC采样通道0，触发信号为ADCSYNC_IN0
csi_adc_set_sync(ADC0, ADC_SYNCIN0, ADC_SYNC_CONT, 0);         //选择ADC_SYNCIN0同步事件，同步事件发生后延迟 0us @pclk=40Mhz启动ADC
//ETCB配置，设置目标事件(EXI TRGOUT1)和源事件(ADC SYNCIN0): EXI TRGOUT1 -> ETCB -> ADC SYNCIN0
.....
csi_etb_init();
.....

```

详见exi\_etcb\_adc\_samp\_demo函数。

#### 6.1.16.4 exi\_etcb\_gpta\_capture\_demo

##### 1、函数位置

etcb\_demo.c

##### 2、功能描述

GPTA sync2 sync3区分捕获示例。

##### 3、参考用法

```

// PA3上升沿和下降沿配置为外部中断功能，选择中断组3和中断组16
csi_ex_i_set_evtrg(EXI_TRGOUT0, EXI_TRGSRC_GRP3, 1);           //EXI3触发EXI_TRGOUT0
csi_ex_i_evtrg_enable(EXI_TRGOUT0);
.....
csi_ex_i_set_evtrg(EXI_TRGOUT5, EXI_TRGSRC_GRP16, 1);          //EXI16触发EXI_TRGOUT5
csi_ex_i_evtrg_enable(EXI_TRGOUT5);
//ETCB配置，设置目标事件(GPTA0_SYNCIN3)和源事件(EXI_TRGOUT5): EXI_TRGOUT5 -> ETCB -> GPTA0_SYNCIN3
.....
csi_etb_init(); //初始化两个ETCB
.....
//初始化tCapConfig，设置GPTA0的捕获参数
.....
csi_gpta_capture_init(GPTA0, &tCapConfig);

//使能SYNCIN2外部触发
csi_gpta_set_sync(GPTA0, GPTA_SYNCIN2, GPTA_SYNC_CONT, GPTA_AUTO_REARM_ZRO);
csi_gpta_sync_enable(GPTA0, GPTA_SYNCIN2);

```

```
//使能SYNCIN3外部触发
csi_gpta_set_sync(GPTA0, GPTA_SYNCIN3, GPTA_SYNC_CONT, GPTA_AUTO_REARM_ZRO);
csi_gpta_sync_enable(GPTA0, GPTA_SYNCIN3);

csi_gpta_start(GPTA0);           //启动GPTA0
```

详见exi\_etcb\_gpta\_capture\_demo函数。

#### 6.1.16.5 exi\_etcb\_gptb\_start\_demo

##### 4、函数位置

gptb\_demo.c

##### 5、功能描述

EXI通过ETCB触发GPTB启动输出波形（10KHz/50%）的demo。

##### 6、参考用法

```
//PA1上升沿和下降沿配置为外部中断功能，选择中断组1
csi_exi_set_evtrg(EXI_TRGOUT1, EXI_TRGSRC_GRP1, 1);           //EXI1触发EXI_TRGOUT1
csi_exi_evtrg_enable(EXI_TRGOUT1);

.....
//ETCB配置，设置目标事件(GPTB0_SYNCIN0)和源事件(EXI_TRGOUT1): EXI_TRGOUT1 -> ETCB ->
GPTA0_SYNCIN0
.....
csi_etb_init(); //初始化ETCB
.....
//初始化tPwmCfg，设置PWM波形
.....
csi_gptb_pwm_init(GPTB0, &tPwmCfg);
//初始化tPwmChCfg
.....
csi_gptb_set_channel(GPTB0, &tPwmChCfg, GPTB_CHANNEL_1)
//使能SYNCIN0外部触发
csi_gptb_set_sync(GPTB0, GPTB_SYNCIN0, GPTB_SYNC_CONT, GPTB_AUTO_REARM_ZRO);
csi_gptb_sync_enable(GPTB0, GPTB_SYNCIN0);
```

详见exi\_etcb\_gptb\_start\_demo函数。

#### 6.1.16.6 bt\_etcb\_gptb\_start\_demo

##### 7、函数位置

gptb\_demo.c

##### 8、功能描述

BT0的周期事件通过ETCB触发GPTB启动输出波形（10KHz/50%）的demo。

##### 9、参考用法

```

//BT0初始化，并设置触发源
.....
csi_bt_timer_init(BT0,&tTimConfig);           //BT0 定时
csi_bt_set_evtrg(BT0, BT_TRGSRC_PEND);       //BT0 PEND事件触发输出
csi_bt_evtrg_enable(BT0);                   //BT0 事件触发输出使能
csi_bt_start(BT0);

.....
//ETCB配置，设置目标事件(GPTB0_SYNCIN0)和源事件(BT0_TRGOUT): BT0_TRGOUT -> ETCB ->
GPTB0_SYNCIN0
.....
csi_etb_init(); //初始化ETCB
.....
//初始化tPwmCfg，设置PWM波形
.....
csi_gptb_pwm_init(GPTB0, &tPwmCfg);
//初始化tPwmChCfg
.....
csi_gptb_set_channel(GPTB0, &tPwmChCfg, GPTB_CHANNEL_1)
//使能SYNCIN0外部触发
csi_gptb_set_sync(GPTB0, GPTB_SYNCIN0, GPTB_SYNC_CONT, GPTB_AUTO_REARM_ZRO);
csi_gptb_sync_enable(GPTB0, GPTB_SYNCIN0);

```

详见exi\_etcb\_gptb\_start\_demo函数。

## 6.1.17 UART

### 6.1.17.1 uart\_send\_demo

#### 1、函数位置

uart\_demo.c

#### 2、功能描述

UART发送一串数据功能示例。使用轮询方式。

#### 3、参考用法

```

//初始化tUartConfig，设置UART基本参数：波特率115200，奇校验，发送和接收都是轮询模式，不开启中断
.....
csi_uart_init(UART0, &tUartConfig);           //初始化串口
csi_uart_start(UART0, UART_FUNC_RX_TX);       //开启UART的RX和TX功能，也可单独开启
csi_uart_send(UART0,(void *)bySendData,18);  //发送18个字节的数据

```

详见uart\_send\_demo函数。

### 6.1.17.2 uart\_send\_int\_demo

#### 1、函数位置

uart\_demo.c

## 2、功能描述

UART发送一串数据功能示例。使用中断方式。

## 9、参考用法

```
//初始化tUartConfig, 设置UART基本参数: 波特率115200, 奇校验, 发送中断模式, 接收轮询模式, 使用TX中断
.....
csi_uart_init(UART0, &tUartConfig);           //初始化串口
csi_uart_int_enable(UART0, UART_INTSRC_TXFIFO); //开启TXFIFO中断
csi_uart_start(UART0, UART_FUNC_RX_TX);        //开启UART的RX和TX功能, 也可单独开启
csi_uart_send(UART0, (void *)bySendData, 28); //采用中断方式。调用该函数时, UART发送中断使能
```

详见uart\_send\_int\_demo函数。

### 6.1.17.3 uart\_receive\_int\_demo

## 1、函数位置

uart\_demo.c

## 2、功能描述

UART接收功能示例。使用FIFO中断方式, 接收数据。

## 3、参考用法

```
//初始化tUartConfig, 设置UART基本参数: 波特率115200, 奇校验, 发送和接收都是轮询模式, 不开启中断
.....
csi_uart_init(UART0, &tUartConfig);           //初始化串口
csi_uart_int_enable(UART0, UART_INTSRC_RXFIFO); //开启RXFIFO中断, 字节接收超时中断不开启(用户可根据具体处理需要是否开启)
csi_uart_start(UART0, UART_FUNC_RX_TX);        //开启UART的RX和TX功能, 也可单独开启
```

详见uart\_receive\_int\_demo函数

### 6.1.17.4 uart\_send\_dma\_demo

## 1、函数位置

uart\_demo.c

## 2、功能描述

UART通过DMA发送数据。

## 2、参考用法

```
.....
```



```
csi_uart_init(UART0, &tUartConfig);           //初始化串口
csi_uart_set_send_dma(UART0, UART_DMA_TXFIFO_NFULL); //设置send1 DMA工作模式并使能
csi_uart_start(UART0, UART_FUNC_RX_TX); //开启UART的RX和TX功能，也可单独开启
.....
csi_dma_ch_init(DMA0, DMA_CH1, &tDmaConfig);           //初始化DMA0，选择CH1
csi_dma_int_enable(DMA0, DMA_CH0,DMA_INTSRC_TCIT);     //使用TCIT中断
.....
csi_etcb_ch_init(ETCB_CH20, &tEtbConfig);           //初始化ETCB，DMA ETCB CHANNEL > ETCB_CH19
csi_uart_send_dma(UART0, DMA0, DMA_CH1, (void *)s_bySendBuf, 26); //发送数据
```

详见uart\_send\_dma\_demo函数。

#### 6.1.17.5 uart\_receive\_dma\_demo

##### 1、函数位置

uart\_demo.c

##### 2、功能描述

UART通过DMA接收数据。

##### 3、参考用法

```
.....
csi_uart_init(UART0, &tUartConfig);           //初始化串口
csi_uart_set_receive_dma(UART0, UART_DMA_RXFIFO_NSPACE); //设置receive DMA工作模式并使能
csi_uart_start(UART0, UART_FUNC_RX_TX); //开启UART的RX和TX功能，也可单独开启
.....
csi_dma_ch_init(DMA0, DMA_CH1, &tDmaConfig);           //初始化DMA0，选择CH1
csi_dma_int_enable(DMA0, DMA_CH0,DMA_INTSRC_TCIT);     //使用TCIT中断
.....
csi_etcb_ch_init(ETCB_CH20, &tEtbConfig);           //初始化ETCB，DMA ETCB CHANNEL > ETCB_CH19
csi_uart_rcv_dma(UART0, DMA0, DMA_CH0, (void*)s_byRecvBuf,22); //接收数据
```

详见uart\_receive\_dma\_demo函数。

#### 6.1.18 USART

##### 6.1.18.1 usart\_send\_demo

##### 1、函数位置

usart\_demo.c

##### 2、功能描述

USART发送一串数据功能示例。使用轮询方式。

##### 4、参考用法

```
//初始化tUsartConfig，设置USART基本参数：波特率115200，偶校验，发送和接收都是轮询模式，不开启中断
```

```

.....
csi_usart_init(USART0, &tUsartConfig);           //初始化串口
csi_usart_start(USART0, USART_FUNC_RX_TX); //开启USART的RX和TX功能，也可单独开启
csi_usart_send(USART0, (void *)bySendData, 18); //发送18个字节的数据

```

详见usart\_send\_demo函数。

### 6.1.18.2 usart\_send\_int\_demo

#### 1、函数位置

usart\_demo.c

#### 2、功能描述

USART发送一串数据功能示例。使用中断方式。

#### 10、 参考用法

```

//初始化tUsartConfig，设置USART基本参数：波特率115200，偶校验，发送中断模式，接收轮询模式，使用TX中断
.....
csi_usart_init(USART0, &tUsartConfig);           //初始化串口
csi_usart_int_enable(USART0, USART_INTSRC_TXFIFO); //开启TXFIFO中断
csi_usart_start(USART0, USART_FUNC_RX_TX); //开启USART的RX和TX功能，也可单独开启
csi_usart_send(USART0, (void *)bySendData, 28); //采用中断方式。调用该函数时，USART发送中断使能

```

详见usart\_send\_int\_demo函数。

### 6.1.18.3 usart\_receive\_int\_demo

#### 1、函数位置

usart\_demo.c

#### 2、功能描述

USART接收功能示例。使用FIFO中断方式，接收数据。

#### 3、参考用法

```

//初始化tUsartConfig，设置USART基本参数：波特率115200，奇校验，发送和接收都是轮询模式，不开启中断
.....
csi_usart_init(USART0, &tUsartConfig);           //初始化串口
csi_usart_int_enable(USART0, USART_INTSRC_RXFIFO); //开启RXFIFO中断，字节接收超时中断不开启(用户可根据具体处理需要是否开启)
csi_usart_start(USART0, USART_FUNC_RX_TX); //开启USART的RX和TX功能，也可单独开启

```

详见usart\_receive\_int\_demo函数

#### 6.1.18.4 usart\_send\_dma\_demo

##### 1、函数位置

usart\_demo.c

##### 2、功能描述

USART通过DMA发送数据。

##### 4、参考用法

```
.....
csi_usart_init(USART0, &tUsartConfig);           //初始化串口
csi_usart_set_send_dma(USART1, USART_DMA_TXFIFO_NFULL); //设置send1 DMA工作模式并使能
csi_usart_start(USART0, USART_FUNC_RX_TX); //开启USART的RX和TX功能，也可单独开启
.....
csi_dma_ch_init(DMA0, DMA_CH1, &tDmaConfig);           //初始化DMA0，选择CH1
csi_dma_int_enable(DMA0, DMA_CH0,DMA_INTSRC_TCIT);     //使用TCIT中断
.....
csi_etcb_ch_init(ETCB_CH20, &tEtbConfig); //初始化ETCB，DMA ETCB CHANNEL > ETCB_CH19
csi_usart_send_dma(USART0, DMA0, DMA_CH1, (void *)s_bySendBuf, 26); //发送数据
```

详见usart\_send\_dma\_demo函数。

#### 6.1.18.5 usart\_receive\_dma\_demo

##### 1、函数位置

usart\_demo.c

##### 2、功能描述

USART通过DMA接收数据。

##### 5、参考用法

```
.....
csi_usart_init(USART0, &tUsartConfig);           //初始化串口
csi_usart_set_receive_dma(USART1, USART_DMA_RXFIFO_NSPACE); //设置receive DMA工作模式并使能
csi_usart_start(USART0, USART_FUNC_RX_TX); //开启USART的RX和TX功能，也可单独开启
.....
csi_dma_ch_init(DMA0, DMA_CH1, &tDmaConfig);           //初始化DMA0，选择CH1
csi_dma_int_enable(DMA0, DMA_CH0,DMA_INTSRC_TCIT);     //使用TCIT中断
.....
csi_etcb_ch_init(ETCB_CH20, &tEtbConfig); //初始化ETCB，DMA ETCB CHANNEL > ETCB_CH19
csi_usart_receive_dma(USART0, DMA0, DMA_CH0, (void*)s_byRecvBuf,22); //接收数据
```

详见usart\_receive\_dma\_demo函数。。

## 6.1.19 CAN

### 6.1.19.1 can\_send\_demo

#### 1、函数位置

can\_demo.c

#### 2、功能描述

can 发送demo，不开启中断，可通过TXOK状态中断判断发送是否完成。

#### 3、参考用法

```
//初始化tCanConfig，设置can基本参数：波特率500k，时钟选择，自动重传配置
.....
csi_can_init(CAN0, &tCanConfig);           //初始化can
//CAN发送报文配置，每个报文配置在一个报文通道中(通道：CAN_CH1~CAN_CH32)
//报文通道1配置(CAN_CH1)
//id,识别符配置；tx mcr 发送报文控制配置；data a; data b
.....
csi_can_set_msg_tx(CAN0, CAN_CH1, &tCanTxConfig); //配置发送报文通道
csi_can_start(CAN0);                             //使能(打开)CAN模块
csi_can_msg_send(CAN0, CAN_CH1, 7);              //报文通道1中的报文，发送7字节数据
//更新报文中数据(DAR/DBR)
.....
csi_can_set_adata(CAN0, CAN_CH1, byData);        //更新报文通道1中的低4字节数据
//更新报文通道1中报文ID模式为扩展模式
.....
csi_can_set_id(CAN0, CAN_CH1, &tCanTxConfig.tId); //配置报文识别符
//更新后的的报文发送
csi_can_msg_send(CAN0, CAN_CH1, 8);              //报文通道1中的报文，发送8字节数据
.....
```

详见can\_send\_demo函数。

### 6.1.19.2 can\_send\_int\_demo

#### 1、函数位置

can\_demo.c

#### 2、功能描述

can 发送demo，开启状态中断和报文通道中断，可通过具体的报文通道发送中断判断发送完成。

#### 3、参考用法

```
//初始化tCanConfig，设置can基本参数：波特率500k，时钟选择，自动重传配置
.....
csi_can_init(CAN0, &tCanConfig);           //初始化can
//CAN发送报文配置，每个报文配置在一个报文通道中(通道：CAN_CH1~CAN_CH32)
//报文通道1配置(CAN_CH1)
//id,识别符配置；tx mcr 发送报文控制配置；data a; data b
```

```

.....
csi_can_set_msg_tx(CAN0, CAN_CH1, &tCanTxConfig); //配置发送报文通道
csi_can_int_enable(CAN0, CAN_INTSRC_ERR_ALL); //使能所有异常状态中断
csi_can_ch_int_enable(CAN0, CAN_CH1); //使能发送报文对应的报文通道中断
csi_can_start(CAN0); //使能(打开)CAN模块

```

详见can\_send\_int\_demo函数。

### 6.1.19.3 can\_receive\_int\_demo

#### 1、函数位置

can\_demo.c

#### 2、功能描述

CAN报文接收使用中断(开启所有状态中断和对应发送报文通道中断)，在中断函数中处理报文接收。

#### 3、参考用法

```

//初始化tCanConfig, 设置can基本参数: 波特率500k, 时钟选择, 自动重传配置
.....
csi_can_init(CAN0, &tCanConfig); //初始化can
//报文接收到CH3~CH4, ID为全匹配, 匹配上的ID数据接收到对应的报文接收通道中
//接收报文通道3配置(CAN_CH3)
//id,识别符配置; rx mcr 发送报文控制配置
.....
csi_can_set_msg_rx(CAN0, CAN_CH3, &tCanRxConfig); //配置发送报文通道
csi_can_ch_int_enable(CAN0, CAN_CH3); //使能发送报文对应的报文通道中断
csi_can_int_enable(CAN0, CAN_INTSRC_ERR_ALL); //使能所有异常状态中断
csi_can_start(CAN0); //使能(打开)CAN模块

```

详见can\_receive\_int\_demo函数。

### 6.1.19.4 can\_receive\_fifo\_int\_demo

#### 1、函数位置

can\_demo.c

#### 2、功能描述

can 接收demo, 接收报文通道配置为FIFO缓存的报文接收, 接收过滤掩码不使能(ID不过滤全匹配); 接收报文根据报文匹配的ID接收到对应的报文通道中。CAN报文接收FIFO缓存的配置必须和ID滤波配合使用, 即UMASK位使能时(bMskEn=ENABLE), 报文接收FIFO功能才能实现, 若ID不使用过滤匹配, 则报文接收FIFO不可用。

#### 3、参考用法

```

//初始化tCanConfig, 设置can基本参数: 波特率500k, 时钟选择, 自动重传配置
.....
csi_can_init(CAN0, &tCanConfig); //初始化can
//报文接收到CH3~CH5, ID为滤波匹配, 在一个范围内ID匹配(0x7F0~0x7FF), 数据接收到对应的报文接收通道中
//接收报文通道3配置(CAN_CH3)

```

```
//id,识别符配置; rx mcr 发送报文控制配置
.....
csi_can_set_msg_rx(CAN0, CAN_CH3, &tCanRxConfig); //配置发送报文通道
csi_can_ch_int_enable(CAN0, CAN_CH3);             //使能发送报文对应的报文通道中断
.....
csi_can_int_enable(CAN0, CAN_INTSRC_ERR_ALL);      //使能所有异常状态中断
csi_can_start(CAN0);                               //使能(打开)CAN模块
```

详见can\_receive\_fifo\_int\_demo函数。

### 6.1.19.5 can\_remote\_frames\_send\_demo

#### 1、函数位置

can\_demo.c

#### 2、功能描述

CAN远程帧发送demo，使能报文接收通道中断和异常状态中断

#### 3、参考用法

```
//初始化tCanConfig, 设置can基本参数: 波特率500k, 时钟选择, 自动重传配置
.....
csi_can_init(CAN0, &tCanConfig);                 //初始化can
//报文接收到CH3~CH4, ID为全匹配, 匹配上的ID数据接收到对应的报文接收通道中
//CH3~CH4可以作为远程帧的发送, 即发送远程帧来请求数据
//接收报文通道3配置(CAN_CH3/CH4)
//id,识别符配置; rx mcr 发送报文控制配置
.....
csi_can_set_msg_rx(CAN0, CAN_CH3, &tCanRxConfig); //配置发送报文通道
csi_can_ch_int_enable(CAN0, CAN_CH3);             //使能发送报文对应的报文通道中断
.....
csi_can_set_msg_rx(CAN0, CAN_CH4, &tCanRxConfig); //配置发送报文通道
csi_can_ch_int_enable(CAN0, CAN_CH4);             //使能发送报文对应的报文通道中断
.....
csi_can_int_enable(CAN0, CAN_INTSRC_ERR_ALL);      //使能所有异常状态中断
csi_can_start(CAN0);                               //使能(打开)CAN模块
//远程帧(数据请求), 即发送远程帧; 配置为接收报文通道的报文, 通过csi_can_msg_send发送(TXRQST置位), 是为发送远程帧
csi_can_msg_send(CAN0, CAN_CH3, 8);
.....
csi_can_msg_send(CAN0, CAN_CH4, 8);
.....
```

详见can\_remote\_frames\_send\_demo函数。

### 6.1.19.6 can\_remote\_frames\_response\_demo

#### 1、函数位置

can\_demo.c

## 2、功能描述

CAN远程帧应答demo，不使用中断

## 3、参考用法

```
//初始化tCanConfig，设置can基本参数：波特率500k，时钟选择，自动重传配置
.....
csi_can_init(CAN0, &tCanConfig);           //初始化can
//CAN发送报文配置，每个报文配置在一个报文通道中(通道：CAN_CH1~CAN_CH32)
//发送报文对象配置时，使能了远程帧功能，即接收到匹配ID的远程数据请求，自动发送数据
//报文通道1配置(CAN_CH1/CH2)
//id,识别符配置；tx mcr 发送报文控制配置;data a;data b
.....
csi_can_set_msg_tx(CAN0, CAN_CH1, &tCanTxConfig); //配置发送报文通道
.....
csi_can_set_msg_tx(CAN0, CAN_CH2, &tCanTxConfig); //配置发送报文通道
.....
csi_can_start(CAN0);                          //使能(打开)CAN模块
```

详见can\_remote\_frames\_response\_demo函数

## 6.1.20 SPI

### 6.1.20.1 spi\_master\_send\_demo

#### 1、函数位置

spi\_demo.c

#### 2、功能描述

Spi 主机发送一串数据,TX使用轮询。

#### 3、参考用法

```
//端口配置
.....
//初始化tSpiConfig，主机模式、clk空闲电平、帧数据长度、通讯速率1Mbps
.....
csi_spi_init(SPI0,&tSpiConfig);           //初始化spi
csi_spi_start(SPI0);                     //打开SPI0
csi_spi_send(SPI0,byData,8);             //发送数据
```

详见spi\_master\_send\_demo函数。

### 6.1.20.2 spi\_master\_send\_int\_demo

#### 1、函数位置

spi\_demo.c

## 2、功能描述

spi 主机发送数据,使用TXFIFO中断。

## 3、参考用法

```
//端口配置
.....
//初始化tSpiConfig, 主机模式、clk空闲电平、帧数据长度、通讯速率1Mbps
.....
csi_spi_init(SPI0,&tSpiConfig);           //初始化spi
csi_spi_int_enable(SPI0, SPI_INTSRC_TXFIFO); //使能TXFIFO中断
csi_spi_start(SPI0);                      //打开SPI0
.....                                   //发送数据在中断中完成
```

详见spi\_master\_send\_int\_demo函数。

### 6.1.20.3 spi\_slave\_receive\_int\_demo

## 1、函数位置

spi\_demo.c

## 2、功能描述

spi 从机接收数据,使用RXFIFO中断。

## 3、参考用法

```
//端口配置
.....
//初始化tSpiConfig, 主机模式、clk空闲电平、帧数据长度、通讯速率1Mbps
.....
csi_spi_init(SPI0,&tSpiConfig);           //初始化spi
csi_spi_int_enable(SPI0, SPI_INTSRC_RXFIFO); //使能RXFIFO中断
csi_spi_start(SPI0);                      //打开SPI0
.....                                   //接收数据在中断中完成
```

详见spi\_slave\_receive\_int\_demo函数。

### 6.1.20.4 spi\_master\_send\_receive\_demo

## 1、函数位置

spi\_demo.c

## 2、功能描述

spi 主机收发一串数据,收发使用轮询。

## 3、参考用法

```
//端口配置
.....
```



```
//初始化tSpiConfig, 主机模式、clk空闲电平、帧数据长度、通讯速率1Mbps、初始配置无中断、接收轮询模式
.....
csi_spi_init(SPI0,&tSpiConfig);           //初始化spi
csi_spi_start(SPI0);                     //打开SPI0
csi_spi_send_receive(SPI0, bySendData, byRecvData, 16); //接收数据
```

详见spi\_master\_send\_receive\_demo函数。

#### 6.1.20.5 spi\_send\_dma\_demo

##### 1、函数位置

spi\_demo.c

##### 2、功能描述

spi DMA发送示例代码，使用DMA循环发送16字节数据。

##### 3、参考用法

```
//初始化tSpiConfig, 主机模式、clk空闲电平、帧数据长度、通讯速率1Mbps
.....
csi_spi_init(SPI0,&tSpiConfig);           //初始化spi
csi_spi_set_send_dma(SPI0, SPI_DMA_TXFIFO_NFULL); //设置send DMA工作模式并使能
csi_spi_start(SPI0);                     //打开SPI0
//初始化tDmaConfig,
.....
csi_dma_ch_init(DMA0, DMA_CH1, &tDmaSendCfg); //初始化DMA
csi_dma_int_enable(DMA0, DMA_CH0,DMA_INTSRC_TCIT); //使用TCIT中断
//初始化tEtbConfig,
.....
csi_etcb_ch_init(ETCB_CH20, &tEtcSendCfg); //初始化ETCB, DMA ETCB CHANNEL 大于 ETCB_CH19_ID
//spi端口配置
.....
csi_spi_send_dma(SPI0, DMA0, DMA_CH1, (void *)s_bySendBuf, sizeof(s_bySendBuf)); //DMA发送数据
.....
//等待直到dma传输完成
```

详见spi\_send\_dma\_demo函数。

#### 6.1.20.6 spi\_receive\_dma\_demo

##### 1、函数位置

spi\_demo.c

##### 2、功能描述

spi DMA接收示例代码，使用DMA接收16字节数据。

##### 3、参考用法

```
//初始化tSpiConfig, 主机模式、clk空闲电平、帧数据长度
.....
csi_spi_init(SPI0,&tSpiConfig);           //初始化spi
```

```

csi_spi_set_receive_dma(SPI0, SPI_DMA_RXFIFO_NSPACE);    //设置receive DMA工作模式并使能
csi_spi_start(SPI0);                                    //打开SPI0
//初始化tDmaConfig,
.....
csi_dma_ch_init(DMA0, DMA_CH1, &tDmaSendCfg);            //初始化DMA
csi_dma_int_enable(DMA0, DMA_CH0,DMA_INTSRC_TCIT);      //使用TCIT中断
//初始化tEtbConfig,
.....
csi_etcb_ch_init(ETCB_CH21, &tEtcRecvCfg);    //初始化ETCB, DMA ETCB CHANNEL 大于 ETCB_CH19_ID
//spi端口配置
.....
csi_spi_recv_dma(SPI0, DMA0, DMA_CH0, (void*)s_byRecvBuf, 16);    //DMA接收数据
.....
//等待直到dma传输完成

```

详见spi\_receive\_dma\_demo函数。

### 6.1.20.7 spi\_flash\_read\_demo

#### 1、函数位置

spi\_demo.c

#### 2、功能描述

spi读flash示例代码。

#### 3、参考用法

```

//初始化tSpiConfig, 主机模式、clk空闲电平、帧数据长度, 通讯速率1Mbps
.....
csi_spi_init(SPI0,&tSpiConfig);                                //初始化spi
csi_spi_start(SPI0);                                          //打开SPI0
.....
spi_flash_read_id()                                           //读取Flash ID
.....
spi_flash_read_bytes((uint8_t *)s_byRecvBuf, 0x1000, 16);    //在0x1000地址读16字节数据
.....

```

详见spi\_flash\_read\_demo函数。

### 6.1.20.8 spi\_flash\_write\_demo

#### 1、函数位置

spi\_demo.c

#### 2、功能描述

spi写flash示例代码。

#### 3、参考用法

```

//初始化tSpiConfig, 主机模式、clk空闲电平、帧数据长度, 通讯速率1Mbps
.....

```

```

csi_spi_init(SPI0,&tSpiConfig);           //初始化spi
csi_spi_start(SPI0);                     //打开SPI0
.....
spi_flash_read_id()                      //读取Flash ID
.....
spi_flash_sector_erase(0x1000);          //擦除0x1000地址
spi_flash_write_bytes(s_bySendBuf, 0x1000, sizeof(s_bySendBuf)); //往0x1000地址写16字节数据
.....

```

详见spi\_flash\_write\_demo函数。

## 6.1.21 IIC

### 6.1.21.1 iic\_master\_eeprom\_demo

#### 1、函数位置

iic\_demo.c

#### 2、功能描述

iic主机读写EEPROM功能。

#### 3、参考用法

```

//iic端口配置
.....
//初始化g_tlicMasterCfg，设置主机地址模式 7/10 bit、使能重复起始位、设置主机速度模式等
.....
csi_iic_master_init(I2C0,&g_tlicMasterCfg);           //初始化iic主机
csi_iic_int_enable(IIC0, IIC_INTSRC_TX_ABRT);         //使能TX_ABRT中断
csi_iic_write_nbyte(I2C0,0xa0,0x0001,2,&data[0],9);   //写数据
csi_iic_read_nbyte(I2C0,0xa0,0x0001,2,&data1[0],9);   //读数据

```

详见iic\_master\_eeprom\_demo函数。

### 6.1.21.2 iic\_master\_demo

#### 1、函数位置

iic\_demo.c

#### 2、功能描述

iic主机读功能。

#### 3、参考用法

```

//iic端口配置
.....
//初始化g_tlicMasterCfg，设置主机地址模式 7/10 bit、使能重复起始位、设置主机速度模式等
.....
csi_iic_master_init(I2C0,&g_tlicMasterCfg);           //初始化iic主机
csi_iic_int_enable(IIC0, IIC_INTSRC_TX_ABRT);         //使能TX_ABRT中断

```

```
csi_iic_master_read_nbyte (I2C0,0xa0,0x01,1,&data1[0],19); //读数据
```

详见iic\_master\_demo函数。

### 6.1.21.3 iic\_slave\_demo

#### 1、函数位置

iic\_demo.c

#### 2、功能描述

iic从机功能。作为从机时需要在IIC中断里调用 i2c\_irqhandler(I2C0) 函数

#### 3、参考用法

```
//iic端口配置
.....
//初始化g_tlicSlaveCfg, 设置从机地址模式 7/10 bit、使能重复起始位、设置主机速度模式、使能中断等
.....
csi_iic_set_slave_buffer(g_byWriteBuffer,32,g_bySendBuffer,32) //从机就是数组和发送数组设置
csi_iic_slave_init(I2C0,&g_tlicSlaveCfg); //初始化从机
csi_iic_int_enable(IIC0, IIC_INTSRC_SCL_SLOW | IIC_INTSRC_STOP_DET | IIC_INTSRC_RD_REQ |
IIC_INTSRC_RX_FULL | IIC_INTSRC_TX_ABRT|IIC_INTSRC_TX_OVER); //使能需要的中断
```

详见iic\_slave\_demo函数。

### 6.1.21.4 iic\_dma\_send\_demo

#### 1、函数位置

iic\_demo.c

#### 2、功能描述

iic 主机 DMA发送功能。

#### 3、参考用法

```
//iic端口配置
.....
//初始化tEtbConfig
.....
csi_etcb_ch_init(ETCB_CH21, &tEtbConfig); //初始化ETCB, DMA ETCB CHANNEL > ETCB_CH19_ID
.....
//初始化tDmaConfig
csi_dma_ch_init(DMA0, DMA_CH0, &tDmaConfig); //初始化DMA
csi_dma_ch_start(DMA0, DMA_CH0, (void *)g_wTxBuff, (void *)0x400A0010,9,1)
//初始化g_tlicMasterCfg, 设置主机地址模式 7/10 bit、使能重复起始位、设置主机速度模式、使能中断等
.....
csi_iic_master_init(I2C0,&g_tlicMasterCfg); //主机初始化
csi_iic_set_send_dma(IIC0, IIC_TDMA_EN, IIC_TDMA_FIFO_TX_FLSEL);
.....
```

```
csi_iic_enable(I2C0);
```

详见iic\_dma\_send\_demo函数。

### 6.1.21.5 iic\_dma\_receive\_demo

#### 1、函数位置

iic\_demo.c

#### 2、功能描述

iic 主机 DMA接收功能。

#### 3、参考用法

```
//iic端口配置
.....
csi_etb_init();           //初始化从机
.....
//初始化tEtbConfig
csi_etb_ch_config(ETB_CH21, &tEtbConfig);      //初始化ETB, DMA ETB CHANNEL > ETB_CH19_ID
.....
//初始化tDmaConfig
csi_dma_ch_init(DMA0, DMA_CH0, &tDmaConfig); //初始化DMA
csi_dma_ch_start(DMA0, DMA_CH0, (void *)g_wTxBuff, (void *)0x400A0010,9,1)
//初始化g_tlicMasterCfg, 设置主机地址模式 7/10 bit、使能重复起始位、设置主机速度模式、使能中断等
.....
csi_iic_master_init(I2C0,&g_tlicMasterCfg);      //主机初始化
csi_iic_set_receive_dma(IIC0, IIC_RDMA_EN, IIC_RDMA_FIFO_NSPACE);
csi_iic_int_enable(IIC0, IIC_INTSRC_NONE);      //使能需要的中断
.....
csi_iic_read_nbyte_dma(I2C0,0xa0,0x0001,2,&g_bRxBuff[0],9);
```

详见iic\_dma\_receive\_demo函数。

### 6.1.22 LED

#### 6.1.22.1 led\_demo

#### 1、函数位置

led\_demo.c

#### 2、功能描述

LED示例代码。

#### 3、参考用法

```
//led端口配置
.....
//初始化ptLedCfg, COM0~3打开、LED显示亮度50%、显示周期时间、Non-Overlap时间
.....
```

```
csi_led_init(LED, &ptLedCfg);           //初始化led
//根据byDisplayStatus进行led控制
//display status 0:正常控制, 依次显示1, 2, 3, 4
//display status 1:闪烁控制, 关闭COM0, 依次显示2, 3, 4
//display status 2:闪烁控制, 关闭COM0/1, 依次显示3, 4
//display status 3:闪烁控制, 关闭COM0/1/2, 显示4
```

详见led\_demo函数。

## 6.1.23 SIO

### 6.1.23.1 sio\_led\_rgb\_demo

#### 1、函数位置

sio\_demo.c

#### 2、功能描述

sio 驱动RGB LED(ws2812), RGB DATA = 24bit; 驱动数据输出排列方式:GRB。

#### 3、参考用法

```
//sio端口配置
.....
//初始化tSioTxCfg, Dx 对象序列长度、发送数据缓存长度、一次发送总的的数据长度、SIO空闲时刻IO管脚输出电平
.....
csi_sio_tx_init(SIO0, &tSioTxCfg);       //初始化sio
led_rgb_display(byDipData, 8);          //驱动led显示
```

详见sio\_led\_rgb\_demo函数。

### 6.1.23.2 sio\_led\_rgb\_txbufempt\_int\_demo

#### 1、函数位置

sio\_demo.c

#### 2、功能描述

SIO驱动RGB三色LED的demo, 使用TXBUFEMPT中断。

#### 3、参考用法

```
//sio端口配置
.....
//初始化tSioTxCfg, Dx 对象序列长度、发送数据缓存长度、一次发送总的的数据长度、SIO空闲时刻IO管脚输出电平
.....
csi_sio_tx_init(SIO0, &tSioTxCfg);       //初始化sio
//数据转换, RGB数据转换为SIO发送数据
.....
set_led_rgb_store(s_wSioTxbuf, s_byDipData, byCount);
.....
```

```
csi_sio_int_enable(SIO0, SIO_INTSRC_TXBUFEMPT); //使能TXBUFEMPT中断，中断函数中发送数据
```

详见sio\_led\_rgb\_txbufempt\_int\_demo函数。

### 6.1.23.3 sio\_led\_rgb\_txdne\_int\_demo

#### 1、函数位置

sio\_demo.c

#### 2、功能描述

SIO驱动RGB三色LED的demo，使用TXBUFEMPT中断。

#### 3、参考用法

```
//sio端口配置
.....
//初始化tSioTxCfg, Dx 对象序列长度、发送数据缓存长度、一次发送总的的数据长度、SIO空闲时刻IO管脚输出电平
.....
csi_sio_tx_init(SIO0, &tSioTxCfg); //初始化sio
//数据转换，RGB数据转换为SIO发送数据
.....
set_led_rgb_store(s_wSioTxbuf, s_byDipData, byCount);
.....
csi_sio_int_enable(SIO0, SIO_INTSRC_TXDNE); //使能TXBUFEMPT中断，中断函数中发送数据
csi_sio_send(SIO0, s_wSioTxbuf, 3); //发送24个bit，产生TXDNE中断
```

详见sio\_led\_rgb\_txdne\_int\_demo函数。

### 6.1.23.4 sio\_led\_rgb\_send\_dma\_demo

#### 1、函数位置

sio\_demo.c

#### 2、功能描述

DMA方式SIO驱动RGB三色LED的demo。

#### 3、参考用法

```
//sio端口配置
.....
//初始化tSioTxCfg, Dx 对象序列长度、发送数据缓存长度、一次发送总的的数据长度、SIO空闲时刻IO管脚输出电平
.....
csi_sio_tx_init(SIO0, &tSioTxCfg); //初始化sio
csi_sio_send_dma_enable(SIO0); //SIO发送DMA使能
.....
set_led_rgb_store(byDipDataEnd, byCount);
.....
//初始化tEtbConfig、初始化tDmaConfig
.....
```

```
csi_dma_ch_init(DMA0, DMA_CH1, &tDmaConfig);           //初始化DMA
csi_sio_send_dma(SIO0, DMA0, DMA_CH1, hwDmaSendData, 24); //dma发送
.....
csi_etb_init();           //初始化etcb
csi_etb_ch_config(ETB_CH21, &tEtbConfig);           //配置etcb
```

详见sio\_led\_rgb\_send\_dma\_demo函数。

### 6.1.23.5 sio\_hdq\_master\_wrcmd\_demo

#### 1、函数位置

sio\_demo.c

#### 2、功能描述

HDQ协议主机写数据命令demo，非中断方式。

#### 3、参考用法

```
//sio端口配置
.....
//初始化tSioTxCfg, Dx 对象序列长度、发送数据缓存长度、一次发送总的的数据长度、SIO空闲时刻IO管脚输出电平等
.....
csi_sio_tx_init(SIO0, &tHdqTxCfg);           //初始化sio
wSendBuf[0] = sio_hdq_addr_conver(byHdqData[0] | HDQ_WR_CMD) //第一个字节：地址+命令
wSendBuf[1] = sio_data_conver(byHdqData[1]           // 第二个字节：数据字节
csi_sio_send(SIO0, wSendBuf, 2);           //发送数据
```

详见sio\_hdq\_master\_wrcmd\_demo函数。

### 6.1.23.6 sio\_hdq\_master\_rdcmd\_demo

#### 1、函数位置

sio\_demo.c

#### 2、功能描述

HDQ协议主机读取数据命令demo，接收使用RXDNE中断(也可用RXBUFFULL)。

#### 3、参考用法

```
//sio端口配置
.....
//初始化tHdqRxCfg, Dx 对象序列长度、发送数据缓存长度、一次发送总的的数据长度、SIO空闲时刻IO管脚输出电平等
.....
csi_sio_rx_init(SIO0, &tHdqRxCfg);           //初始化sio
csi_sio_break_rst(SIO0, SIO_BKLEV_LOW, 19, ENABLE); //检测接收break
csi_sio_int_enable(SIO0, SIO_INTSRC_RXDNE); //接收RXDNE中断, byRxCnt <=32bit; 接收byRxCnt(8)个bit,
产生中断, 读取数据到接收buf, 每次读取byRxCnt(8)bit
//发送读命令
.....
```



```
wSendBuf[0] = sio_hdq_addr_conversion(byHdqData[0]); //第一个字节: 地址+命令(低7位是地址, 第8位是读写(0/1)控制位);
csi_sio_send(SIO0, wSendBuf, 1); //发送读数据命令
csi_sio_set_wkmode(SIO0, SIO_RECV); //设置SIO为接收模式,即进入接收模式
```

详见sio\_hdq\_master\_rdcmd\_demo函数。

### 6.1.23.7 sio\_hdq\_slave\_demo

#### 1、函数位置

sio\_demo.c

#### 2、功能描述

sio 实现TI HDQ单线通讯协议, 主机读取数据; 数据传输方式LSB, 低7位是地址, 最高位是R/W(0/1)控制位; 一次传输两个字节。

#### 3、参考用法

```
//sio端口配置
.....
//初始化tHdqRxCfg, Dx 对象序列长度、发送数据缓存长度、一次发送总的长度、SIO空闲时刻IO管脚输出电平等
.....
csi_sio_tx_init(SIO0, &tHdqTxCfg); //初始化SIO发送参数
s_wTxData = sio_data_conversion(byTxChar); //发送数据转换, 数据发送在中断函数里执行
.....
csi_sio_rx_init(SIO0, &tHdqRxCfg); //初始化sio
csi_sio_set_break(SIO0, SIO_BKLEV_LOW, 19, ENABLE); //接收检测break, HDQ协议起始需要break总线
csi_sio_int_enable(SIO0, SIO_INTSRC_BREAK); //使能BREAK中断
csi_sio_int_enable(SIO0, SIO_INTSRC_RXBUFFULL); //使能RXBUFFULL中断, 接收byRxBufLen(8)个bit, 产生中断, 读取数据到接收buf, 每次读取byRxBufLen(8)bit
.....
```

详见sio\_hdq\_slave\_demo函数。

### 6.1.24 CMP

#### 6.1.24.1 cmp\_base\_demo

#### 1、函数位置

cmp\_demo.c

#### 2、功能描述

CMP基本功能示例。比较器的P极和N极输入不同电平值, 如果P>N, 输出高电平; 如果P<N, 输出低电平。

#### 3、参考用法

```
csi_gpio_set_mux(GPIOA, PA8,PA8_CPIN1P); //设置PA9为比较器的N级输入
csi_gpio_set_mux(GPIOA, PA9,PA9_CPIN1N); //设置PA8为比较器的P级输入
csi_gpio_set_mux(GPIOB, PB2,PB2_CP0_OUT); //设置PB3为比较器的输出
//初始化tCmpCfg, 设置CMP基本参数: 输入端口, 参考电压, 输出极性等
.....
csi_cmp_init(CMP0,&tCmpCfg);
csi_cmp_int_enable(CMP0, CMP_INTSRC_EDGEDET); //若需使用中断, 请调该接口使能对应中断, 这里使用
CMP_INTSRC_EDGEDET中断
csi_cmp_start(CMP0);
```

详见cmp\_base\_demo函数。

#### 6.1.24.2 cmp\_dfcf\_demo

##### 1、函数位置

cmp\_demo.c

##### 2、功能描述

比较器滤波功能demo。

##### 4、参考用法

```
csi_gpio_set_mux(GPIOA, PA9,PA9_CPIN1N); //设置PA9为比较器的N级输入
csi_gpio_set_mux(GPIOA, PA8,PA8_CPIN1P); //设置PA8为比较器的P级输入
csi_gpio_set_mux(GPIOB, PB2,PB2_CP0_OUT); //设置PB3为比较器的输出
//初始化tCmpCfg, 设置CMP基本参数: 输入端口, 参考电压, 输出极性等
.....
csi_cmp_init(CMP0,&tCmpCfg);
//初始化tCmpFltcrCfg, 设置滤波器的时钟及滤波周期
.....
csi_cmp_set_dflt1(CMP0,&tCmpDflt1Cfg,ENABLE); //数字滤波1
.....
csi_cmp_set_dflt2(CMP0,&tCmpDflt2Cfg,DISABLE); //数字滤波2
csi_cmp_start(CMP0);
```

详见cmp\_dfcf\_demo函数。

#### 6.1.24.3 cmp\_wfcf\_demo

##### 1、函数位置

cmp\_demo.c

##### 2、功能描述

CMP窗口滤波功能示例。BT0通过ETCB模块触发比较器窗口滤波功能。在特定的触发窗口内，比较器的输出将通过一个采样寄存器输出（该寄存器的采样时钟为PCLK）。一旦该窗口关闭，寄存器将保持该输出状态或者以指定逻辑输出，直到下次的窗口有效。捕获窗口可以设置相应的延时时间，在触发信号有效后，延时特定

的时间，再使能捕获窗口。该功能适用于在某些强干扰环境中，对比较器输出在特定时间内进行分析的应用。

#### 4、参考用法

```
csi_pin_set_mux(PA9,PA9_CPIN1N);    //设置PA9为比较器的N级输入
csi_pin_set_mux(PA8,PA8_CPIN1P);    //设置PA8为比较器的P级输入
csi_pin_set_mux(PB2,PB2_CP0_OUT);    //设置PB3为比较器的输出
//初始化tCmpCfg, 设置CMP基本参数: 输入端口, 参考电压, 输出极性等
.....
csi_cmp_init(CMP0,&tCmpCfg);
//初始化tCmpWfcrCfg, 设置窗口滤波器的时钟分频及延迟参数等
.....
csi_cmp_wfcr_config(CMP0,&tCmpWfcrCfg);
csi_cmp_start(CMP0);
//BT配置
.....
csi_bt_timer_init(BT0, &tTimConfig);           //初始化BT0, 定时40000us
csi_bt_int_enable(BT1,BT_INTSRC_PEND);         //使能BT PEND中断
csi_bt_start(BT0);                             //启动定时器
csi_bt_set_evtrg(BT0, BT_TRGSRC_PEND);
csi_bt_evtrg_enable(BT0);
//ETCB配置, 设置目标事件(BT0 PEND)和源事件(ETB_CMP0_SYNCIN): BT0 PEND -> ETCB -> CMP0 SYNCIN
.....
csi_etcb_ch_init(ch, &tEtbConfig);
```

详见cmp\_wfcr\_demo函数。

### 6.1.25 OPA

#### 6.1.25.1 opa\_internal\_gain\_mode\_demo

##### 1、函数位置

opa\_demo.c

##### 2、功能描述

OPA内部增益模式功能示例。使用内部增益，使用ADC采集输入，设置需要放大的倍数。

##### 3、参考用法

```
//配置OPA0/1/2/3的输入输出引脚
.....
//初始化ptOpaConfig_t, 使用内部增益, 选择增益倍数
csi_opa_init(OPA0,&tOpaConfig);
csi_opa_enable(OPA0);
```

详见opa\_internal\_gain\_mode\_demo函数。

### 6.1.25.2 opa\_external\_gain\_mode\_demo

#### 1、函数位置

opa\_demo.c

#### 2、功能描述

OPA外部增益模式功能示例。使用外部增益，放大倍数由外部反馈电阻确定。

#### 4、参考用法

```
//配置OPA0/1/2/3的输入输出引脚
.....
//初始化ptOpaConfig_t, 使用外部增益
csi_opa_init(OPA0,&tOpaConfig);
csi_opa_enable(OPA0);
```

详见opa\_external\_gain\_mode\_demo函数。

### 6.1.26 WIZARD

#### 6.1.26.1 lvr\_wizard\_demo

#### 1、函数位置

wizard\_demo.c

#### 2、功能描述

Wizard图形化配置LVR功能的方法,及复位源查询方式

#### 3、参考用法

```
// 用户配置格式化控制串启动文本，只有输入该文本之后，图形配置功能才会启动
// <<< Use Configuration Wizard in Context Menu >>>
// 符号图形化配置控制串
// <q> LVR Level (in Bytes)<2>[LVR_19//LVR_22//LVR_25//LVR_28//LVR_31//LVR_34//LVR_37//LVR_40]
// <i> Config LVR Level for the application
#define          LVR_LEVEL          LVR_31

csi_set_lvr(LVR_LEVEL);          //VDD掉电到设定LVR电压，芯片复位
csi_lvd_lvr_enable();
//VDD掉电到设定LVR电压，芯片复位用户配置格式化控制串启动文本，只有输入该文本之后，图形配置功能才会启动
.....
```

详见lvr\_wizard\_demo函数。

#### 6.1.26.2 cqcr\_wizard\_demo

#### 1、函数位置

wizard\_demo.c

## 2、功能描述

Wizard图形化配置CQCR功能的方法

## 3、参考用法

```
// 用户配置格式化控制串启动文本，只有输入该文本之后，图形配置功能才会启动
// <<< Use Configuration Wizard in Context Menu >>>
// 符号图形化配置控制串
// <q> SRC Freq (in Bytes)<2>[CQCR_SRCSEL_IM//CQCR_SRCSEL_ES//CQCR_SRCSEL_IS//CQCR_SRCSEL_HF]
// <i> Config SRC Freq for the application
#define SRC_FREQ CQCR_SRCSEL_IM
// <q> REF Freq(in Bytes)<2>[CQCR_REFSEL_EM//CQCR_REFSEL_ES]
// <i> Config REF Freq for the application
#define REF_FREQ CQCR_REFSEL_ES
// 数字图形化配置控制串
// <o> REF Value (in Bytes) <0x0-0x3FFFF:2>
// <i> Config REF Value for the application
#define REF_VALUE 0x3f2

csi_set_cqcr(REF_FREQ,SRC_FREQ,REF_VALUE); //参考时钟选择 EM， 源时钟选择控 IM
.....
```

详见cqcr\_wizard\_demo函数。

## 6.2 apt32f173x\_demo\_callback

该部分示例代码，主要用于演示中断使用回调的情况。注意每个使用回调的IP都有一个宏用于打开和关闭中断回调。以BT为例，该宏为USE\_BT\_CALLBACK，在board\_config.h中定义。当USE\_BT\_CALLBACK=1时，使用BT中断回调；当USE\_BT\_CALLBACK=0时，不使用BT中断回调。默认值为0，即不使用中断回调。

以下给出部分回调函数示例。详细内容可在csi代码中查看。

### 6.2.1 ADC

#### 6.2.1.1 adc\_continuous\_callback\_demo

## 3、函数位置

adc\_demo\_callback.c

## 4、功能描述

连续转换模式：ADC启动后进行整个序列采样，序列采样完成后，继续从序列第一个通道开始，如此循环。采样期间可以停止。

### 3、参考用法

```
csi_gpio_set_mux(GPIOC,PC13, PC13_ADC_INA0);           // PC13作为ADC输入引脚
.....
//初始化tAdcConfig, 配置ADC参数, 单次采样
.....
csi_adc_init(ADC0, &tAdcConfig);           //初始化ADC参数配置
csi_adc_set_seq_num(ADC0,iChnlNum);        //配置ADC总采样通道个数
csi_adc_set_seqx(ADC0,0,ADC_INA0,ADC_CV_COUNT_1,ADC_AVG_COF_1,ADCSYNC_NONE); //配置ADC采样通道0
csi_adc_int_enable(ADC0, ADC_INTSRC_SEQ0|ADC_INTSRC_SEQ1|ADC_INTSRC_SEQ2); //设置采样通道0, 1, 2的转换完成中断
csi_adc_register_callback(ADC0, user_adc0_callback);
csi_adc_start(ADC0);
```

详见 adc\_continuous\_callback\_demo 函数。

## 6.2.2 BT

### 6.2.2.1 bt\_timer\_callback\_demo

#### 1、函数位置

bt\_demo\_callback.c

#### 2、功能描述

BT1基本定时功能示例。

#### 3、参考用法

```
csi_bt_time_config_t tTimConfig;           //BT timer 结构体

tTimConfig.wTimeVal = 1000;                //BT定时值 = 1000us
tTimConfig.eRunMode = BT_RUN_CONT;         //BT计数器工作模式
csi_bt_timer_init(BT1, &tTimConfig);       //初始化BT

csi_bt_int_enable(BT1,BT_INTSRC_PEND);     //使能BT PEND中断
csi_bt_register_callback(BT0, user_bt0_callback); //注册中断回调函数
csi_bt_start(BT1);                         //启动BT定时器
```

详见bt\_timer\_callback\_demo函数。

## 6.2.3 CAN

### 6.2.3.1 can\_send\_int\_callback\_demo

#### 1、函数位置

can\_demo\_callback.c

#### 2、功能描述

CAN报文发送使用中断(开启所有状态中断和对应发送报文通道中断), 使用标准csi库函数, 库函数中断支持回调函数, 用户可注册报文通道发送完毕中断的回调函数, 用户可在回调函数里做自己的处理, 不必关注底层的中断。

### 3、参考用法

```
//初始化tCanConfig, 设置can基本参数: 波特率500k, 时钟选择, 自动重传配置
.....
csi_can_init(CAN0, &tCanConfig);           //初始化can
//CAN发送报文配置, 每个报文配置在一个报文通道中(通道: CAN_CH1~CAN_CH32)
//报文通道1配置(CAN_CH1)
//id,识别符配置; tx mcr 发送报文控制配置; data a; data b
.....
csi_can_set_msg_tx(CAN0, CAN_CH1, &tCanTxConfig); //配置发送报文通道
//注册发送完成中断用户回调函数
csi_can_register_callback(CAN0, CAN_CALLBACK_SEND, user_send_callback);
//注册传输异常中断用户回调函数
csi_can_register_callback(CAN0, CAN_CALLBACK_ERR, user_err_callback);
csi_can_start(CAN0);                       //使能(打开)CAN模块
csi_can_msg_send_int(CAN0, CAN_CH1, 8);    //发送报文通道1中的报文, 开启所有异常状态中断和对应报文通道中断
```

详见can\_send\_int\_callback\_demo函数。

## 6.2.4 CMP

### 6.2.4.1 cmp\_base\_int\_callback\_demo

#### 1、函数位置

cmp\_demo\_callback.c

#### 2、功能描述

CMP基本功能示例。比较器的P极和N极输入不同电平值, 如果 $P > N$ , 输出高电平;如果 $P < N$ , 输出低电平。

### 5、参考用法

```
csi_gpio_set_mux(GPIOA, PA8, PA8_CPIN1P); //设置PA9为比较器的N级输入
csi_gpio_set_mux(GPIOA, PA9, PA9_CPIN1N); //设置PA8为比较器的P级输入
csi_gpio_set_mux(GPIOB, PB2, PB2_CP0_OUT); //设置PB3为比较器的输出
//初始化tCmpCfg, 设置CMP基本参数: 输入端口, 参考电压, 输出极性等
.....
csi_cmp_init(CMP0, &tCmpCfg);
csi_cmp_int_enable(CMP0, CMP_INTSRC_EDGEDET); //若需使用中断, 请调该接口使能对应中断, 这里使用
CMP_INTSRC_EDGEDET中断
csi_cmp_register_callback(CMP0, cmp0_callback); //注册中断回调函数
csi_cmp_start(CMP0);
```

详见cmp\_base\_int\_callback\_demo函数。

## 6.2.5 CNTA

### 6.2.5.1 cnta\_timer\_int\_callback\_demo

#### 1、函数位置

cnta\_demo\_callback.c

#### 2、功能描述

CA0基本定时功能示例。

#### 3、参考用法

```
csi_cnta_timer_config_t  tTimerCfg;

tTimerCfg.eClkDiv = CNTA_CK_DIV8;           //时钟8分频
tTimerCfg.wTime = 1000;                     //周期1000us
tTimerCfg.eRunMode = CNTA_RUN_CONT;         //重复模式
csi_cnta_timer_init(CA0, &tTimerCfg);        //初始化CountA

csi_cnta_int_enable(CA0, CNTA_INTSRC_PENDL); //使能cnta PENDL中断
csi_cnta_register_callback(CA0, cnta_callback); //注册中断回调函数
csi_cnta_start(CA0);                        //启动CountA
```

详见cnta\_timer\_int\_callback\_demo函数。

## 6.2.6 DAC

### 6.2.6.1 dac\_callback\_demo

#### 5、函数位置

dac\_demo\_callback.c

#### 6、功能描述

配置DAC的初始电平码值，设置时钟分频

#### 3、参考用法

```
.....
csi_dac_init(DAC0, &tDacConfig);           // 配置DAC 分频、参考电平选择等
csi_dac_enbale(DAC0);
csi_dac_int_enable(DAC0, DAC_INTSRC_EOC);  //使能EOC中断
csi_dac_register_callback(DAC0, dac_callback); //注册中断回调函数
csi_dac_start(DAC0);                      // 开启DAC
```

详见dac\_callback\_demo函数。



## 6.2.7 GPTA

### 6.2.7.1 gpta\_timer\_int\_callback\_demo

#### 1、函数位置

gpta\_demo\_callback.c

#### 2、功能描述

GPTA基本定时功能示例。

#### 3、参考用法

```
csi_gpta_time_config_t tTimConfig;

tTimConfig.wTimeVal = 10000;           //GPTA定时值 = 1000us
tTimConfig.eRunMode = GPTA_RUN_CONT;   //GPTA计数器工作模式
csi_gpta_timer_init(GPTA0, &tTimConfig); //初始化GPTA0, 定时10000us; GPTA定时, 默认采用向上计数
csp_gpta_int_enable(GPTA0, GPTA_INT_PEND); //使能GPTA0 PEND中断
csi_gpta_register_callback(GPTA0, gpta0_timer_callback); //注册中断回调函数
csi_gpta_start(GPTA0);
```

详见gpta\_timer\_int\_callback\_demo函数。

## 6.2.8 GPTB

### 6.2.8.1 gptb\_pwm\_int\_callback\_demo

#### 2、函数位置

gptb\_demo.c

#### 2、功能描述

GPTB 波形输出示例代码。

#### 3、参考用法

```
csi_gpio_set_mux(GPIOC, PC13, PC13_GPTB0_CHAX) //PC13作为PWM输出引脚
.....
//初始化tPwmCfg, 设置PWM周期和占空比
.....
csi_gptb_pwm_init(GPTB0, &tPwmCfg);
.....
//初始化channel, 设置PWM波形输出
.....
csi_gptb_set_channel(GPTB0, &tPwmChCfg, GPTB_CHANNEL_1);
csi_gptb_int_enable(GPTB0, GPTB_INTSRC_PEND); //使能PEND中断
csi_gptb_register_callback(GPTB0, user_pwm_callback); //注册回调函数, 当发生中断时在该回调函数中处理
csi_gptb_start(GPTB0)
```

详见gptb\_pwm\_int\_callback\_demo函数。

## 6.2.9 IIC

### 6.2.9.1 iic\_slave\_int\_callback\_demo

#### 1、函数位置

iic\_demo\_callback.c

#### 2、功能描述

IIC从机功能。接收数据在user\_iic\_receive\_callback函数中处理。

#### 3、参考用法

```
//IIC端口配置
.....
//初始化g_tlicSlaveCfg, 设置从机地址模式 7/10 bit、使能重复起始位、设置主机速度模式、使能中断等
.....
csi_iic_set_slave_buffer(g_byWriteBuffer,32,g_bySendBuffer,32) //从机就是数组和发送数组设置
csi_iic_slave_init(I2C0,&g_tlicSlaveCfg); //初始化从机
csi_iic_register_callback(IIC0,IIC_CALLBACK_RECV, user_iic_receive_callback); //注册中断回调函数
csi_iic_int_enable(IIC0, IIC_INTSRC_SCL_SLOW | IIC_INTSRC_STOP_DET | IIC_INTSRC_RD_REQ |
IIC_INTSRC_RX_FULL | IIC_INTSRC_TX_ABRT|IIC_INTSRC_TX_OVER); //使能需要的中断
```

详见iic\_slave\_int\_callback\_demo函数。

## 6.2.10 IWDG

### 6.2.10.1 iwdt\_int\_callback\_demo

#### 1、函数位置

iwdt\_demo\_callback.c

#### 2、功能描述

IWDG中断回调例程。

#### 3、参考用法

```
csi_iwdt_init(IWDG_TO_1024); //初始化看门狗，溢出时间为1000ms(系统复位时间)
csi_iwdt_int_enable(IWDG_ALARMTO_2_8); //使能看门狗报警中断，报警时间为4/8溢出时间
csi_iwdt_register_callback(SYSCON, user_iwdt_callback); //注册中断回调函数
csi_iwdt_open(); //打开看门狗
```

详见iwdt\_int\_callback\_demo函数。

## 6.2.11 LED

### 6.2.11.1 led\_callback\_demo

#### 1、函数位置

led\_demo\_callback.c

## 2、功能描述

LED中断回调示例代码。

## 3、参考用法

```
//led端口配置
.....
//初始化ptLedCfg, COM0~3打开、LED显示亮度50%、显示周期时间、Non-Overlap时间
.....
csi_led_init(LED, &ptLedCfg);           //初始化led
csi_led_int_enable(LED, LED_INTSRC_ICEND); //若需使用中断, 请调该接口使能对应中断, 这里使用LED_
INTSRC_ICEND中断
csi_led_register_callback(LED, user_led_callback); //注册中断回调函数
//根据byDisplayStatus进行led控制
//display status 0:正常控制, 依次显示1, 2, 3, 4
//display status 1:闪烁控制, 关闭COM0, 依次显示2, 3, 4
//display status 2:闪烁控制, 关闭COM0/1, 依次显示3, 4
//display status 3:闪烁控制, 关闭COM0/1/2, 显示4
```

详见led\_callback\_demo函数。

## 6.2.12 RTC

### 6.2.12.1 rtc\_alarm\_callback\_demo

## 3、函数位置

rtc\_demo\_callback.c

## 4、功能描述

设置rtc时间的示例代码：包括时钟源、时间模式、时间设置，当前时间回读函数。

## 3、参考用法

```
.....
//初始化tRtcConfig, 选择时钟源, 选择时间模式
.....
csi_rtc_init(RTC, &tRtcConfig);           //初始化RTC参数配置
.....
csi_rtc_set_current_time (RTC, &tRtcTime); //设置时间
csi_rtc_register_callback(RTC, user_rtc_callback); //注册中断回调函数
csi_rtc_start(RTC);                       //RTC开始计时
csi_rtc_set_alarm(RTC, RTC_ALMA, tAlmA.byAlmMode, &tAlmTime); //设置闹钟A
.....
//如果闹钟时间没有到, 每秒打印一次当前时间和距离闹钟的时间
.....
csi_rtc_cancel_alarm(RTC, RTC_ALMA); //取消闹钟, 保持原闹钟时间
```

详见rtc\_alarm\_callback\_demo函数。

## 6.2.13 SIO

### 6.2.13.1 sio\_led\_rgb\_int\_callback\_demo

#### 1、函数位置

sio\_demo\_callback.c

#### 2、功能描述

sio 驱动RGB LED(ws2812), RGB DATA = 24bit; 驱动数据输出排列方式:GRB。

#### 3、参考用法

```
//sio端口配置
.....
//初始化tSioTxCfg, Dx 对象序列长度、发送数据缓存长度、一次发送总的的数据长度、SIO空闲时刻IO管脚输出电平
.....
csi_sio_tx_init(SIO0, &tSioTxCfg);           //初始化sio
//注册SIO0发送中断回调函数
csi_sio_register_callback(SIO0, SIO_CALLBACK_SEND, user_send_callback);
csi_sio_send_int(SIO0, wDipDataEnd, 24);     //数据发送
```

详见sio\_led\_rgb\_int\_callback\_demo函数

## 6.2.14 SPI

### 6.2.14.1 spi\_master\_send\_int\_callback\_demo

#### 1、函数位置

spi\_demo\_callback.c

#### 2、功能描述

Spi 主机发送一串数据,TX使用轮询。

#### 3、参考用法

```
//端口配置
.....
//初始化tSpiConfig, 主机模式、clk空闲电平、帧数据长度、通讯速率1Mbps
.....
csi_spi_init(SPI0,&tSpiConfig);             //初始化spi
//注册SPI0发送中断回调函数
csi_spi_register_callback(SPI0, SPI_CALLBACK_SEND, user_send_callback);
csi_spi_start(SPI0);                       //打开SPI0
csi_spi_send(SPI0,byData,8);               //发送数据
```

详见spi\_master\_send\_int\_callback\_demo函数。

## 6.2.15 UART

### 6.2.15.1 uart\_send\_int\_callback\_demo

#### 1、函数位置

uart\_demo\_callback.c

#### 2、功能描述

UART发送一串数据功能示例。使用中断回调方式。

#### 11、 参考用法

```
//初始化tUartConfig, 设置UART基本参数: 波特率115200, 奇校验, 发送中断模式, 接收轮询模式, 使用TX中断
.....
csi_uart_init(UART0, &tUartConfig);           //初始化串口
csi_uart_register_callback(UART1, UART_CALLBACK_SEND, user_send_callback); //注册回调函数
csi_uart_start(UART0, UART_FUNC_RX_TX);       //开启UART的RX和TX功能, 也可单独开启
csi_uart_send_int(UART0, (void *)bySendData, 28); //采用中断方式。调用该函数时, UART发送中断使能
```

详见uart\_send\_int\_callback\_demo函数。

## 6.2.16 USART

### 6.2.16.1 usart\_send\_int\_callback\_demo

#### 1、函数位置

usart\_demo\_callback.c

#### 2、功能描述

USART发送一串数据功能示例。使用中断回调方式。

#### 12、 参考用法

```
//初始化tUsartConfig, 设置USART基本参数: 波特率115200, 偶校验, 发送中断模式, 接收轮询模式, 使用TX中断
.....
csi_usart_init(USART0, &tUsartConfig);           //初始化串口
csi_usart_register_callback(USART0, USART_CALLBACK_ID_SEND, user_send_callback); //注册回调函数
csi_usart_start(USART0, USART_FUNC_RX_TX);       //开启USART的RX和TX功能, 也可单独开启
csi_usart_send_int(USART0, (void *)bySendData, 28); //采用中断方式。调用该函数时, USART发送中断使能
```

详见usart\_send\_int\_callback\_demo函数。

## 6.2.17 WWDT

### 6.2.17.1 wwdt\_int\_callback\_demo

#### 1、函数位置

wwdt\_demo\_callback.c

## 2、功能描述

窗口看门狗功能示例。用于监测当前程序运行状况。

## 6、参考用法

```
csi_wwdt_init(80);           //设置timeout时间为80ms 时间设置过大会返回错误
csi_wwdt_debug_enable(ENABLE); //可以配置在debug模式下，wwdt是否继续计时
csi_wwdt_set_window_time(40); //设置窗口值为40ms
csi_wwdt_int_enable( );      //中断使能
csi_wwdt_register_callback(WWDT, user_wwdt_callback); //注册中断回调函数
csi_wwdt_open();             //WWDT一旦使能，软件将不能停止
```

详见wwdt\_int\_callback\_demo函数。