

版本	V1.0
日期	202204



Quick Start

APT32F171x 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	2022-4-15	新建使用说明	lq

目录

1. 文档说明.....	3
2. 资源列表.....	3
3. 硬件环境.....	3
3.1 调试模（蓝色部分）	4
3.2 芯片（红色部分）	4
4. 软件环境.....	5
4.1 CDK 开发环境.....	5
4.2 代码结构	5
5. 例程运行	6
5.1 例程获取方式.....	6
5.2 导入 flash 算法.....	14
5.3 适配代码	15
5.4 GPIO 可视化配置.....	20
5.5 外设使用	25
5.6 编译工程	25
5.7 下载调试	26

1. 文档说明

该文档基于爱普特CSI驱动代码架构，适用于APT32F171x系列芯片

2. 资源列表

为了使用APT32F171x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171x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。
 - APT32F171x系列CSI_API说明手册.pdf。
 - APTCHIP_FAQ.chm。集合了一些使用爱普特芯片时的常见问题和解决方法。
- **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 APT32F171x 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。

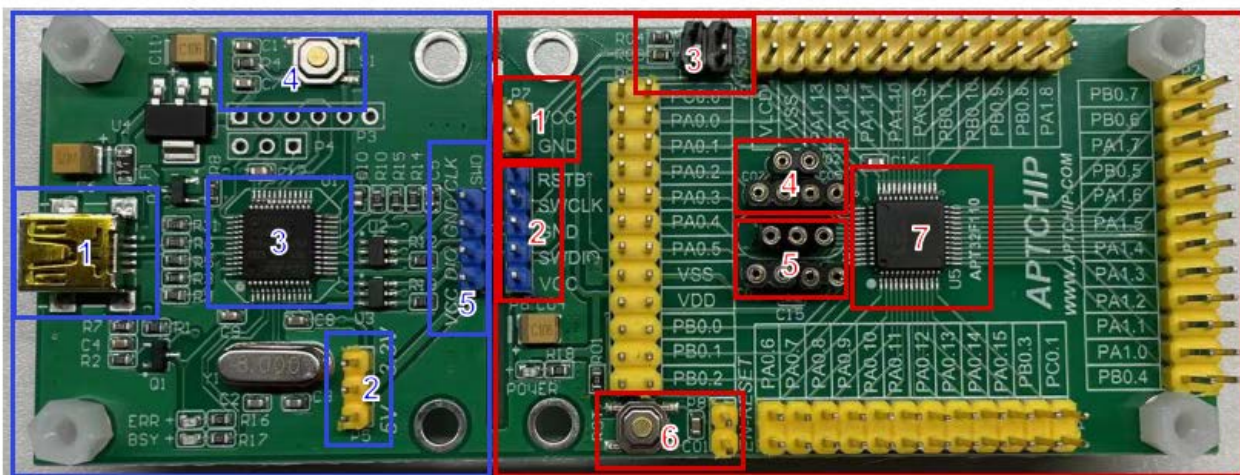


Figure 1 开发板实物图

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。

3.2 芯片（红色部分）

- 1、电源。可用于调试时飞线，电压取决于调试模块 2 的选择
- 2、SWD 调试接口。与调试模块连接的接口。
- 3、SWD 跳线。APT32F171x 系列只有一组 SWD 接口。SWD 口为 PA0.4（SWCLK）和 PA0.3（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 PA0.3 和 PA0.4。使用时需用跳线连接
- 4、晶振电路和跳线。APT32F171x 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，这里是副晶振。
- 5、晶振电路和跳线。APT32F171x 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，这里是主晶振。
- 6、复位电路和跳线。APT32F171x 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、芯片。APT32F171x 系列芯片。

4. 软件环境

4.1 CDK开发环境

APT32F171x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=575997419775328256>

按照提示安装即可。有几个注意事项:

- CDK不支持中文路径。
- 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。

4.2 代码结构

下图为工程视图。这里面包含了6个组件。

apt32f171x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。

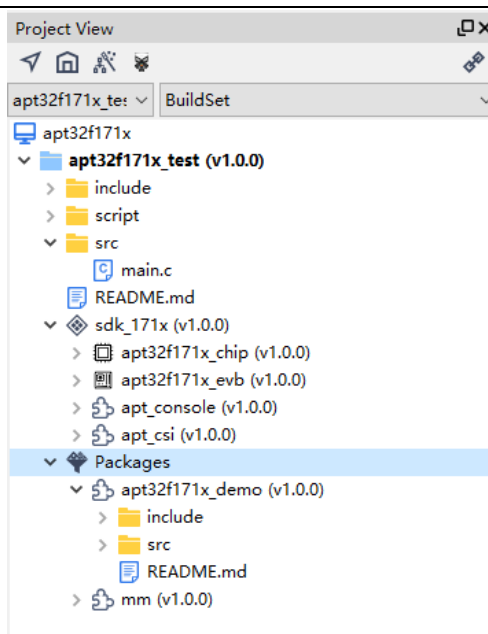


Figure 2 代码结构

5. 例程运行

5.1 例程获取方式

5.1.1 SDK包获取

SDK包可从MCU厂家获取

5.1.2 工程创建方式

- 1、打开CDK，创建一个新的workspace到某一目录（假设D:\lin_qiang\apt32f171\program\apt32f171x_csi），将171SDK包放到此文件夹下。

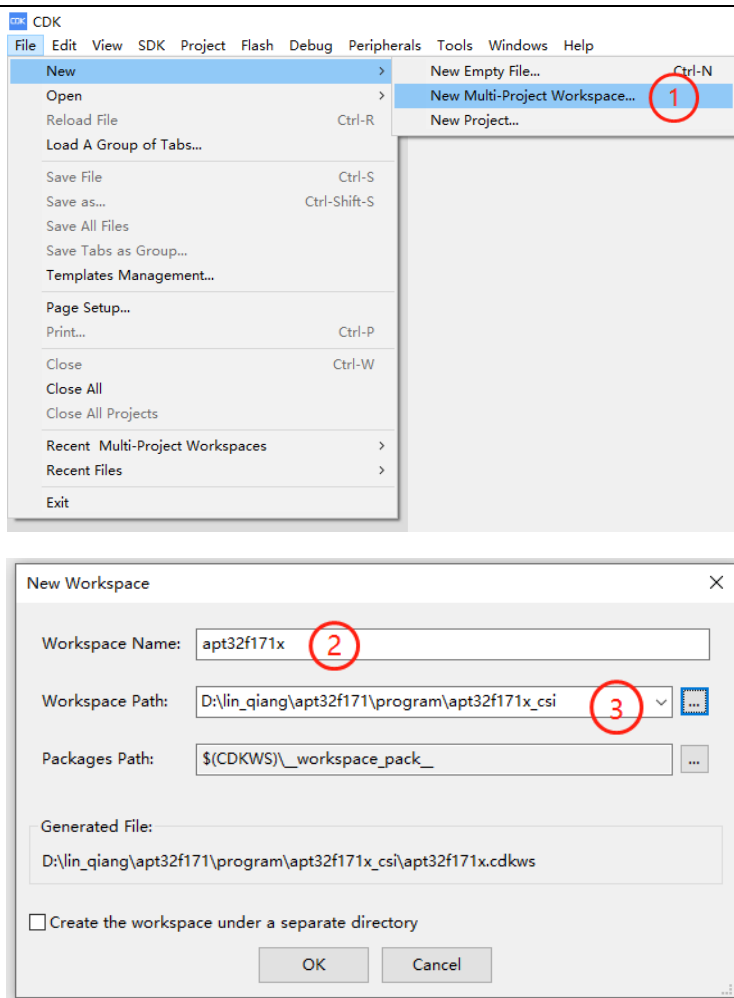


Figure 3 创建新的 workspace

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

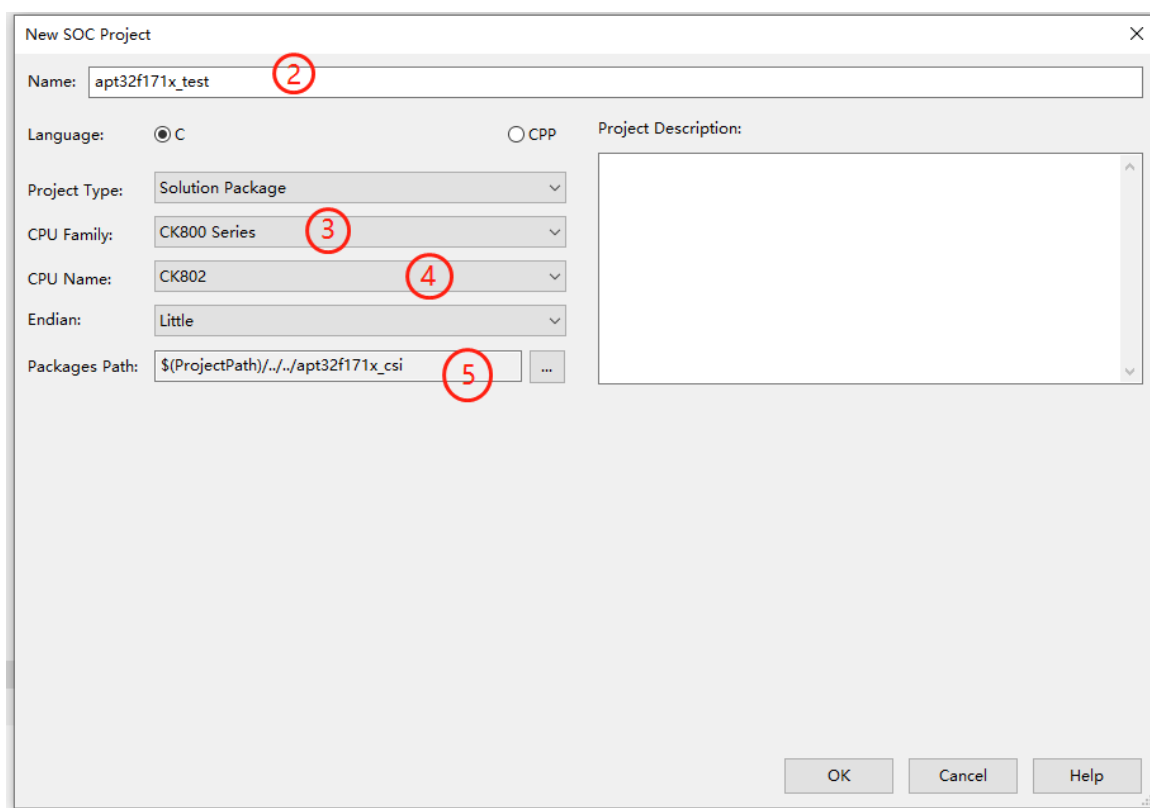
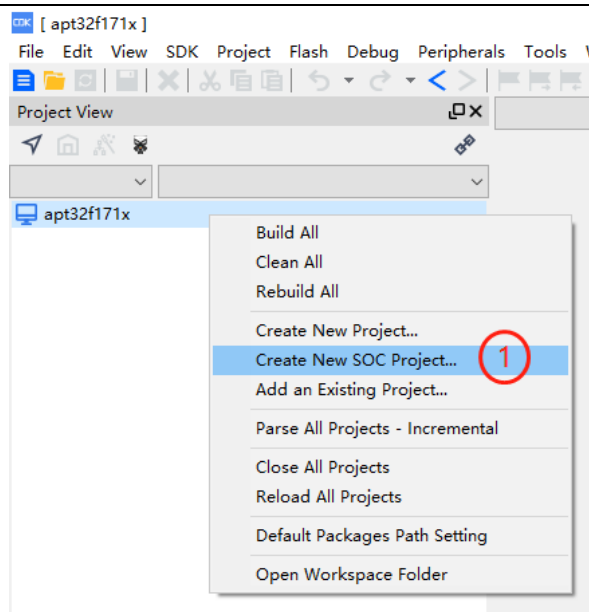
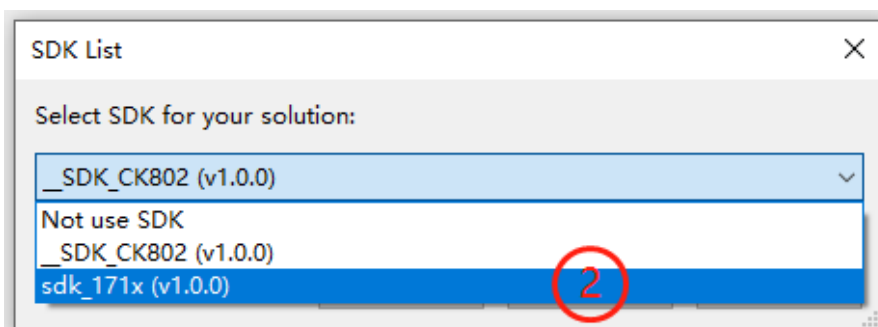
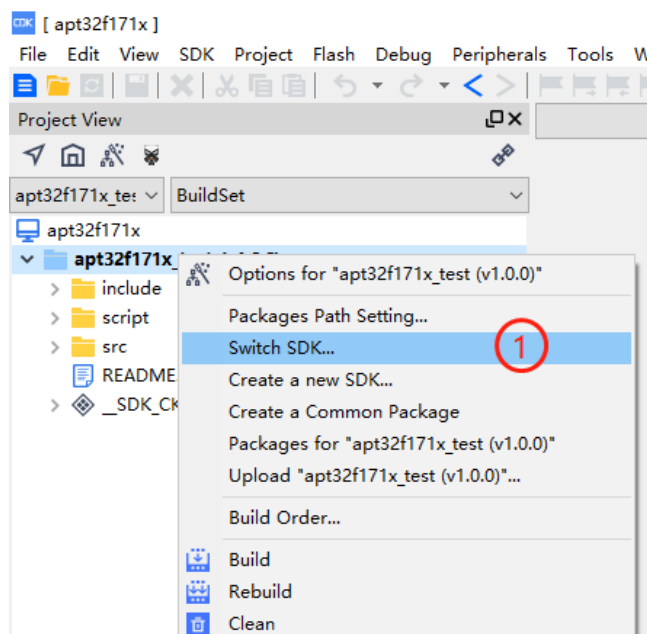


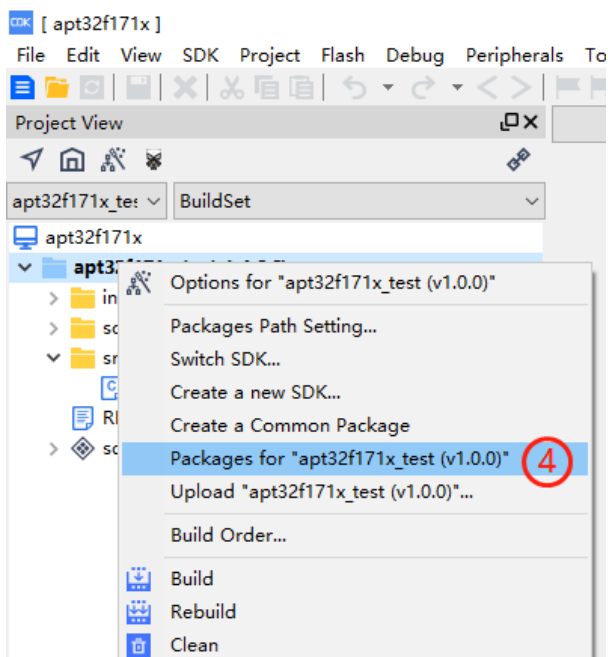
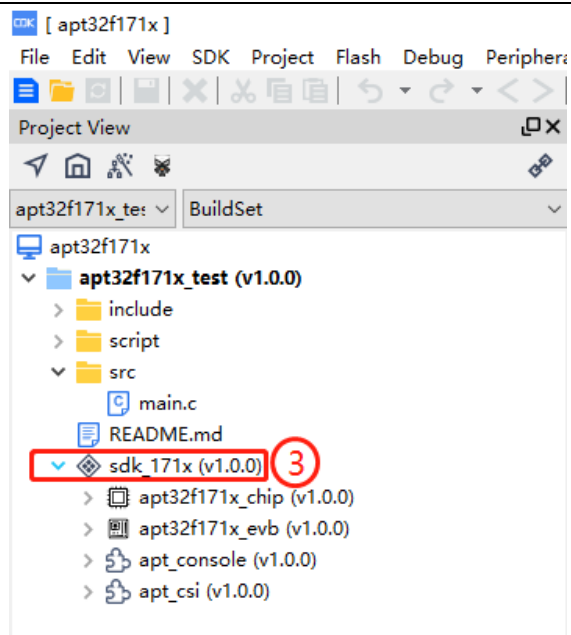
Figure 4 创建新的 soc project

3、一键切换SDK。

1. 按照图中红色标注1右键点击apt32f171x_test工程，选择切换SDK。切换完成后，建议在工程目录下删除__SDK_CK802和__CHIP_CK802的目录。
2. 按照图中红色标注4右键点击apt32f171x_test工程选择创建Package。

3. 在弹出的对话框中按照红色标5选择添加，选择点击OK，apt32f171x_test工程建立完毕。





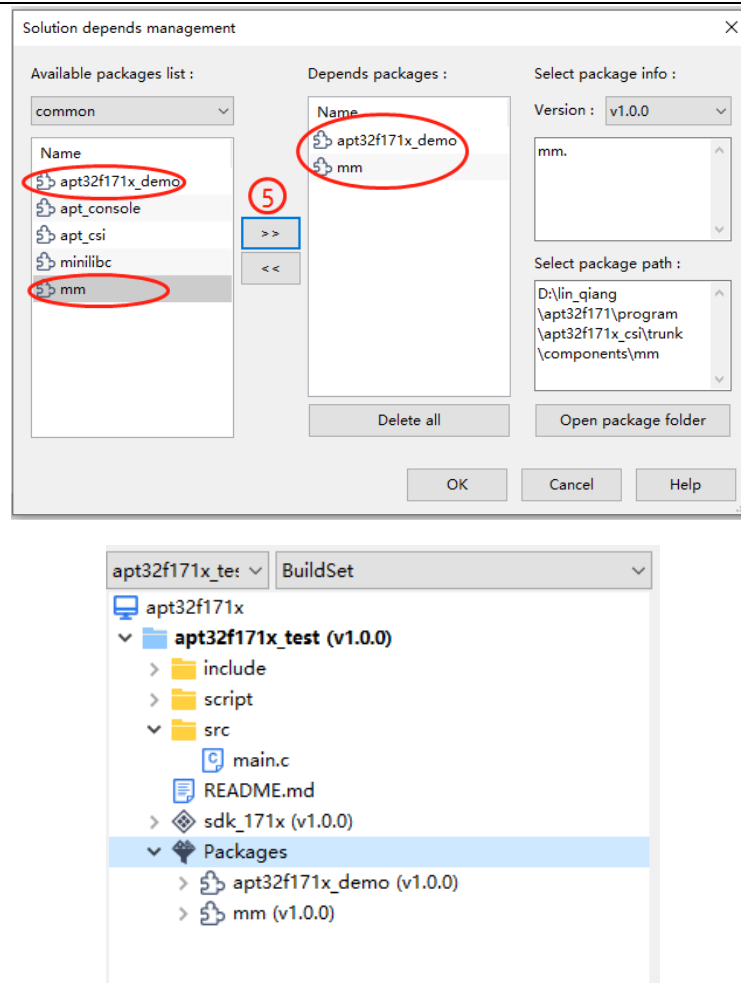


Figure 5 SDK 切换、Package 创建

5.1.3 工程配置

1. 配置选项Compiler

- 点击图中标注1，弹出配置对话框
- 标注2、3按照图中配置所选择
- 标志4中的配置选项在SDK包中的board文件夹下global config文件中，将里面的配置信息copy到4中的配置选项里。标注5建议勾选。

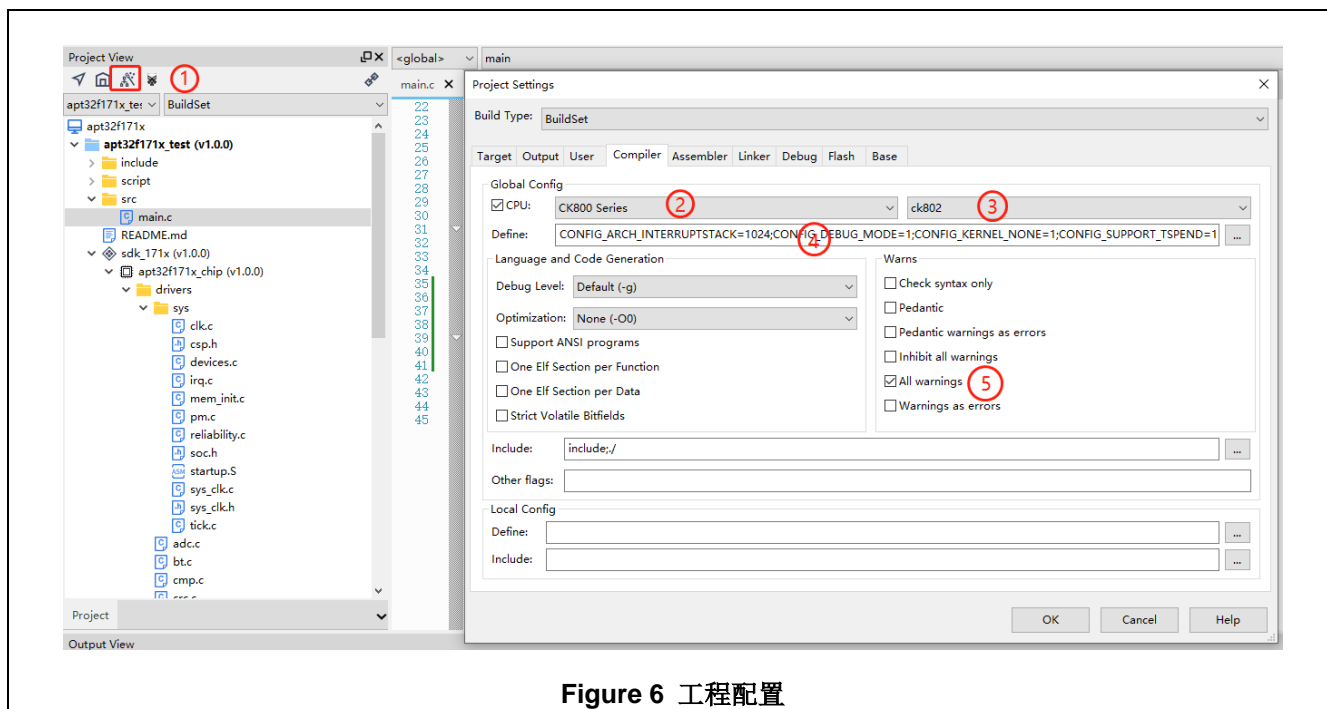


Figure 6 工程配置

2. 配置选项Output

- 标注6, 7选择需要的输出文件

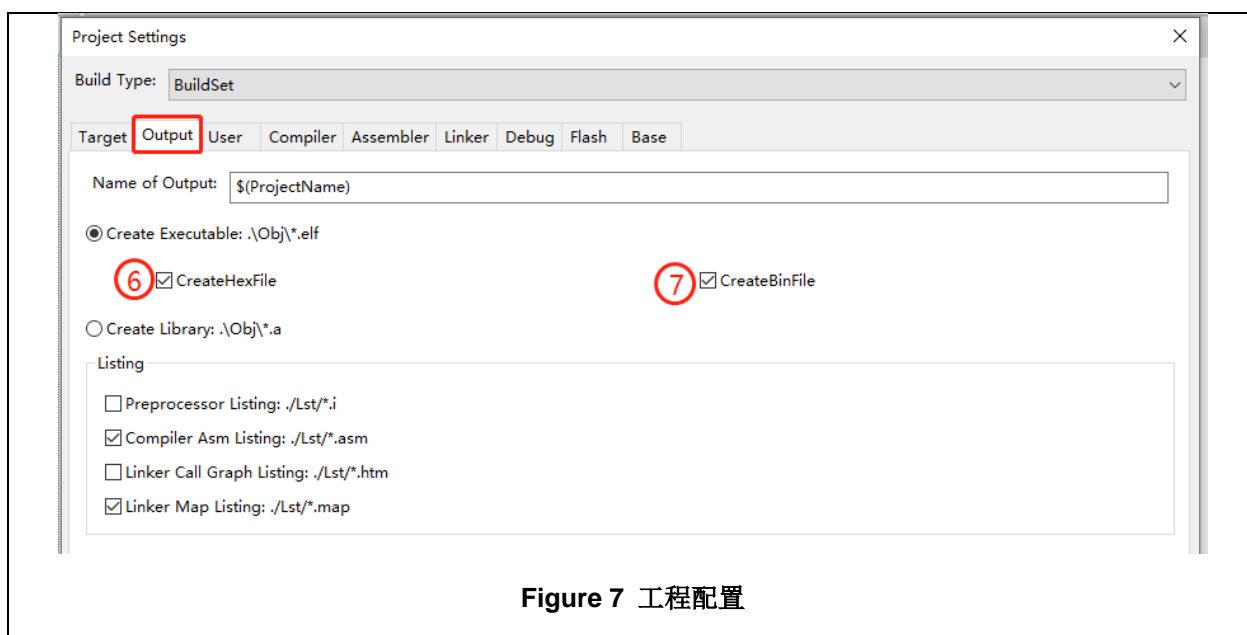


Figure 7 工程配置

3. 配置选项Linker

- 标注8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.ld。结合自己的工程存放位置选择

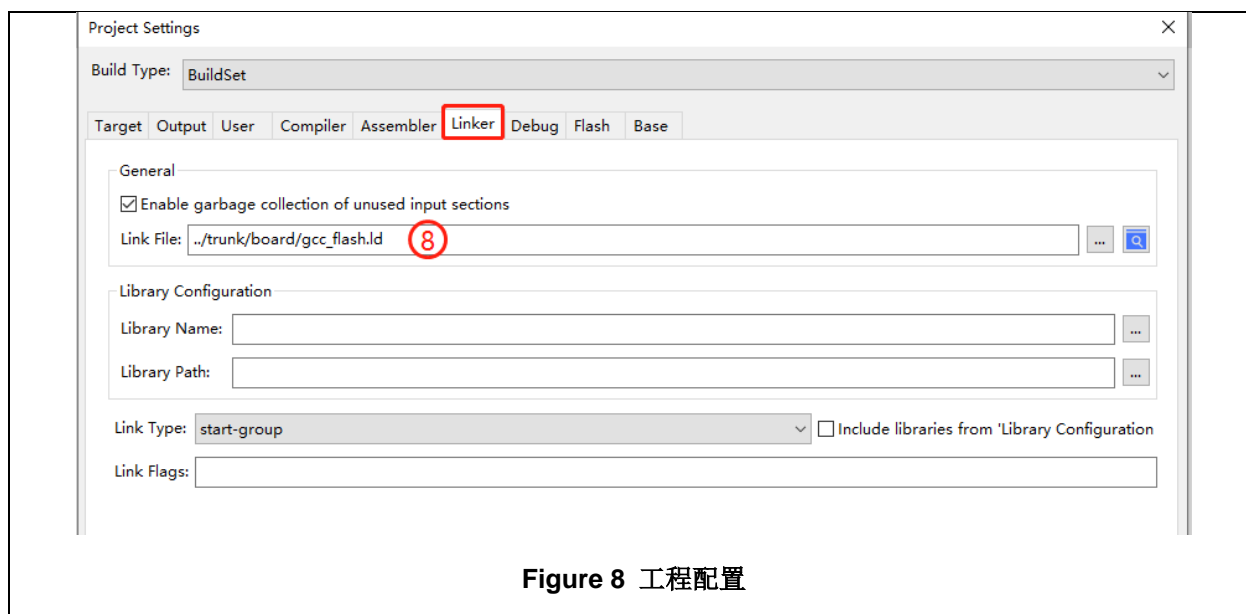


Figure 8 工程配置

4. 配置选项Debug

- 标注9,10,11按图所示配置
- 点击标注12, 标注13, 14按图所示配置

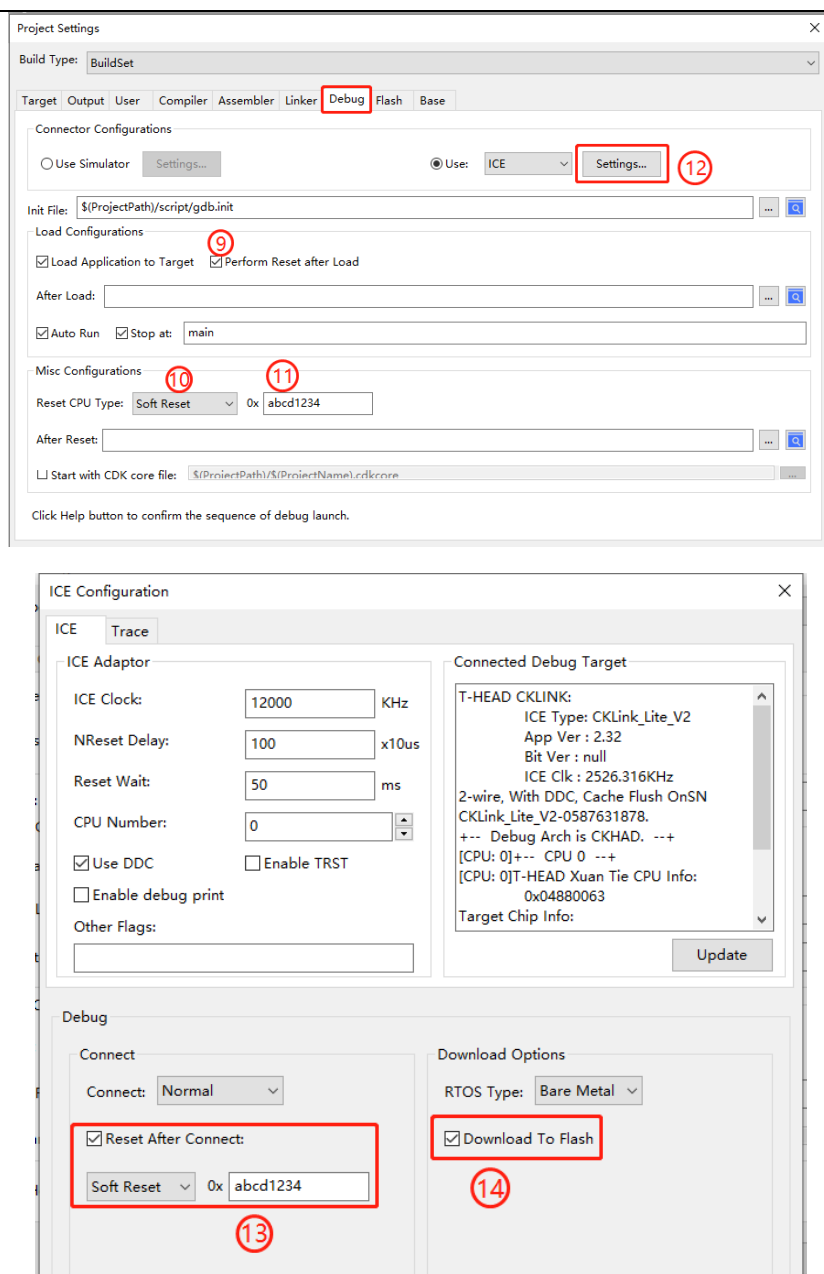


Figure 9 工程配置

5. 配置选项Flash

- 标注15主要是和下载相关的配置，按图配置即可

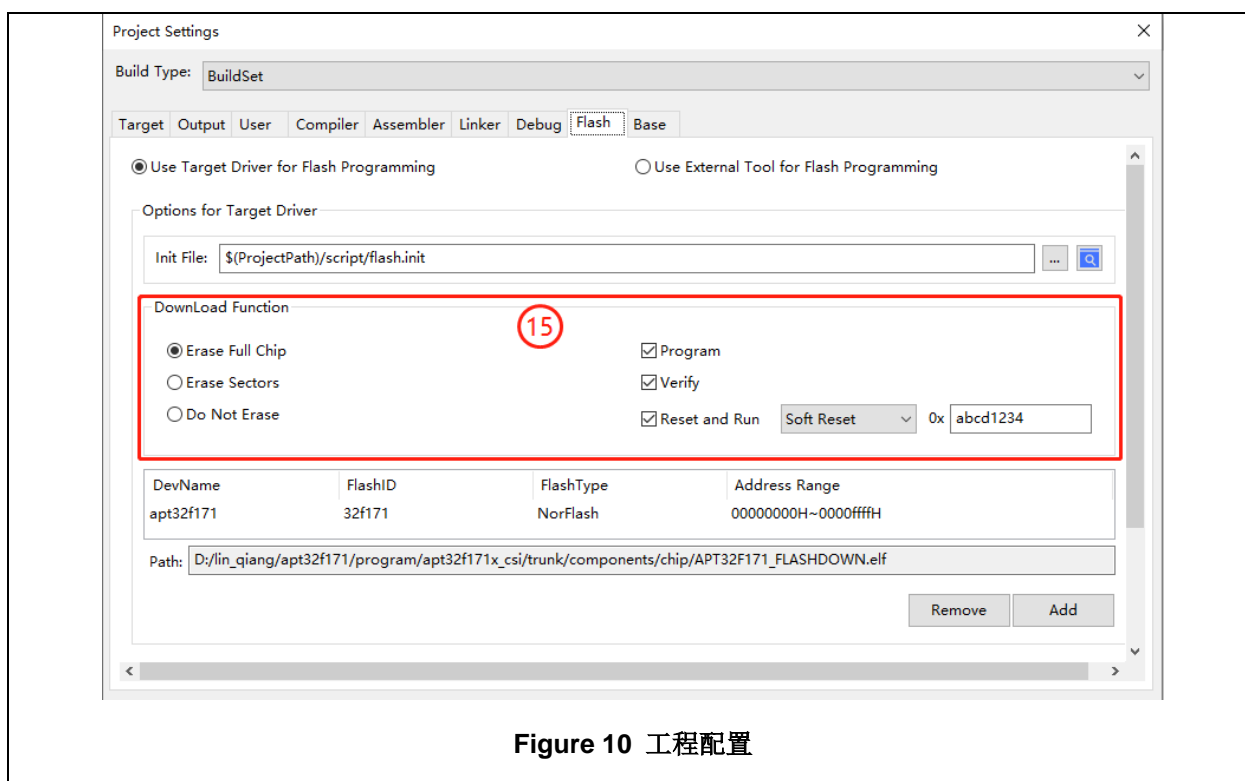


Figure 10 工程配置

5.2 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。

5.3 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。

```
#include "iostring.h"
#include "csi_drv.h"
#include "board_config.h"
#include "demo.h"

//-----
extern void system_init(void);
extern void __ChipInitHandler(void);
//-----

int main()
{
    // TODO
    system_init();
    board_init();
    my_printf("hello world!\n");
}
```

Figure 11 main 函数

5.3.1 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) { CK_CPU_DISALLNORMALIRQ; //关闭全局中断 CSI_iwdt_close(); //关看门狗 iwdt（调试时用） CSI_sysclk_config(); //配置系统时钟（必须有） CSI_get_sclk_freq(); //更新系统时钟 SCLK 的值（必须有） CSI_get_pclk_freq(); //更新外设时钟 PCLK 的值（必须有） CSI_tick_init(); //初始化 systick CK_CPU_ENALLNORMALIRQ; //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tClkConfig（位于board_config.c）。可以通过修改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
csi_clk_config_t tClkConfig =
{
    {SRC_HFOSC, HFOSC_48M_VALUE, SCLK_DIV1, PCLK_DIV1, 5556000, 5556000};
    //{SRC_EMOSC, 20000000, SCLK_DIV1, PCLK_DIV2, 5556000, 5556000};
    //{SRC_IMOSC, IMOSC_5M_VALUE, SCLK_DIV1, PCLK_DIV2, 5556000, 5556000};
    //{SRC_IMOSC, IMOSC_4M_VALUE, SCLK_DIV1, PCLK_DIV1, 5556000, 5556000};
};
```

Figure 12 时钟配置结构体

5.3.2 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，即debug print。使用时需要勾选semi host配置，并且打印的时候使用printf，例如printf("hello world!\n");

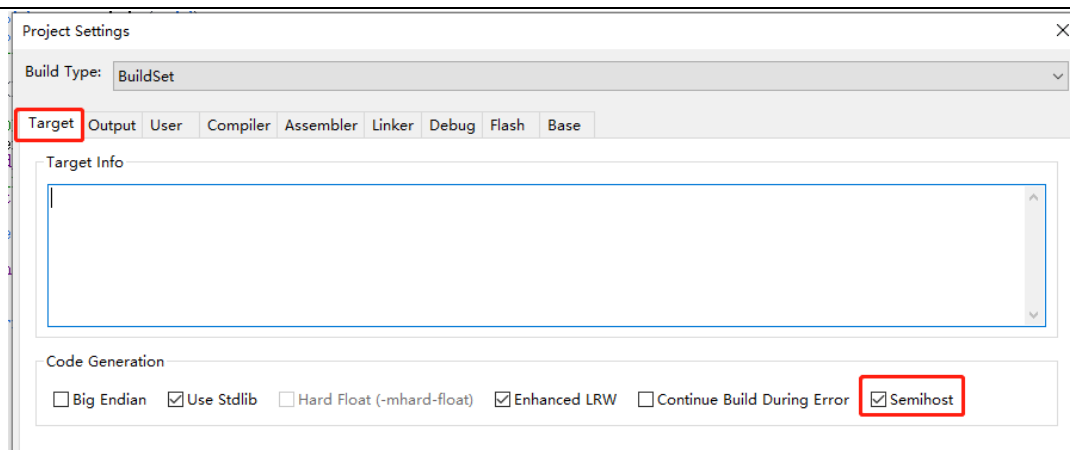


Figure 13 勾选 Semihost

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（见Figure11）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UART0，PA11（TXD）和PA12（RXD）。注意查看MCU 型号是否有UART0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");

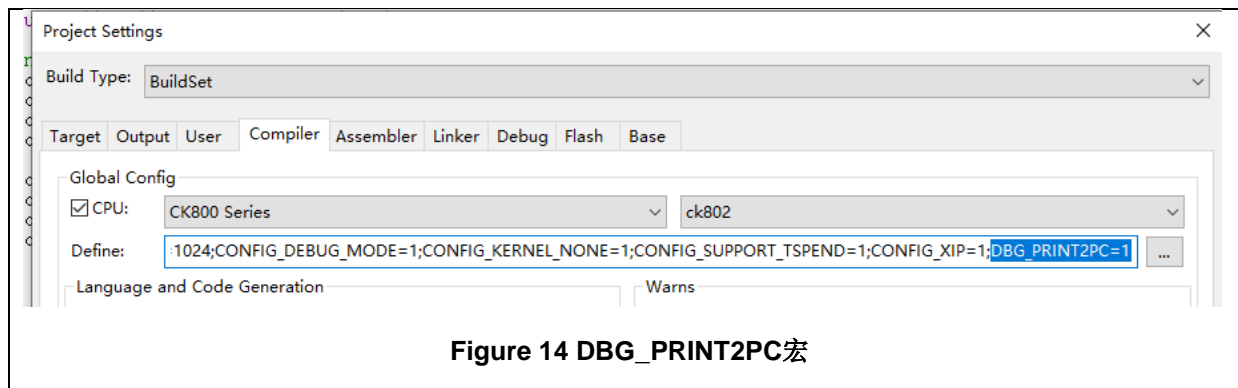


Figure 14 DBG_PRINT2PC宏

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。</p> <p>所有的宏在 board_config.h 中定义。</p> <pre>__attribute__((weak)) void board_init(void) { //console config for print console.uart_id = (uint32_t)CONSOLE_IDX; console.baudrate = 115200U; console.tx.pin = CONSOLE_TXD; console.tx.func = CONSOLE_TXD_FUNC; console.rx.pin = CONSOLE_RXD; console.rx.func = CONSOLE_RXD_FUNC; console.uart = (csp_uart_t*)(APB_UART0_BASE + CONSOLE_IDX * 0x1000); console_init(&console); }</pre>	board_config.c

5.3.3 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BT。

```
void BT0IntHandler(void)
{
    // ISR content ...
    bt_irqhandler(BT0);
}
```

Figure 35 中断处理函数举例

在上述例子中，bt_irqhandler()函数在驱动代码（bt.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BT0IntHandler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ifc_irqhandler ()为例，如果应用中没有用到IFC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 IFCIntHandler(void)

2、srartup.S中将原来IFCIntHandler替换为DummyHandler

- 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。

5.4 GPIO可视化配置

5.4.1 简介

171x支持GPIO初始化的可视化配置，可视化配置在工程sdk的evb组件的svc文件夹下，即下图标识2指示，里面包含171x系列所有封装信息；若用户用SDK包新建工程，工程建立后，需要将svc目录下chip_config_dll.dll文件拷贝到工程目录下，即下图标识1所指示的文件夹下面。例如示例工程，就将chip_config_dll.dll拷贝到D:\lin_qiang\apt32f171\program\apt32f171x_csi\apt32f171x_test文件夹下

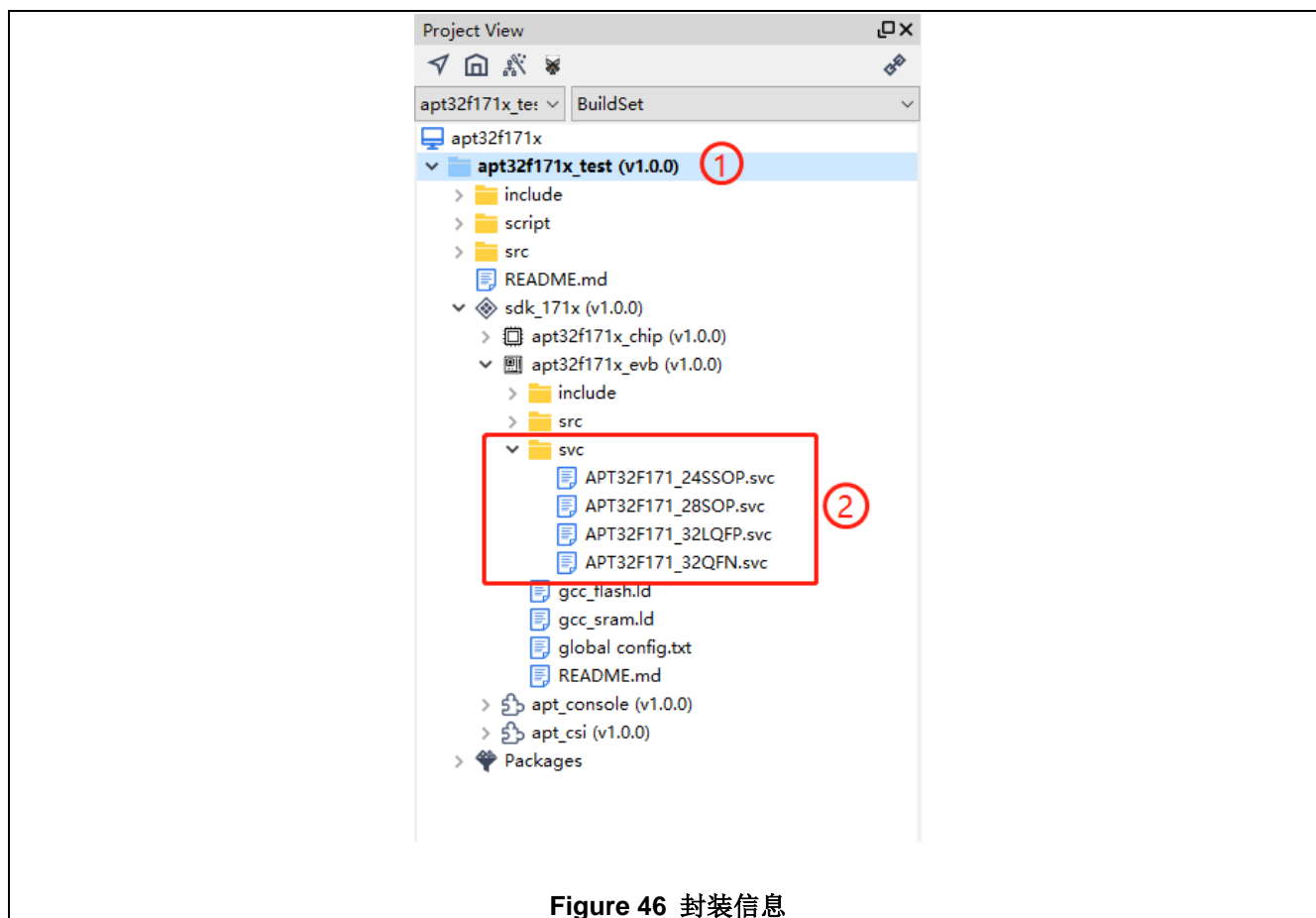


Figure 46 封装信息

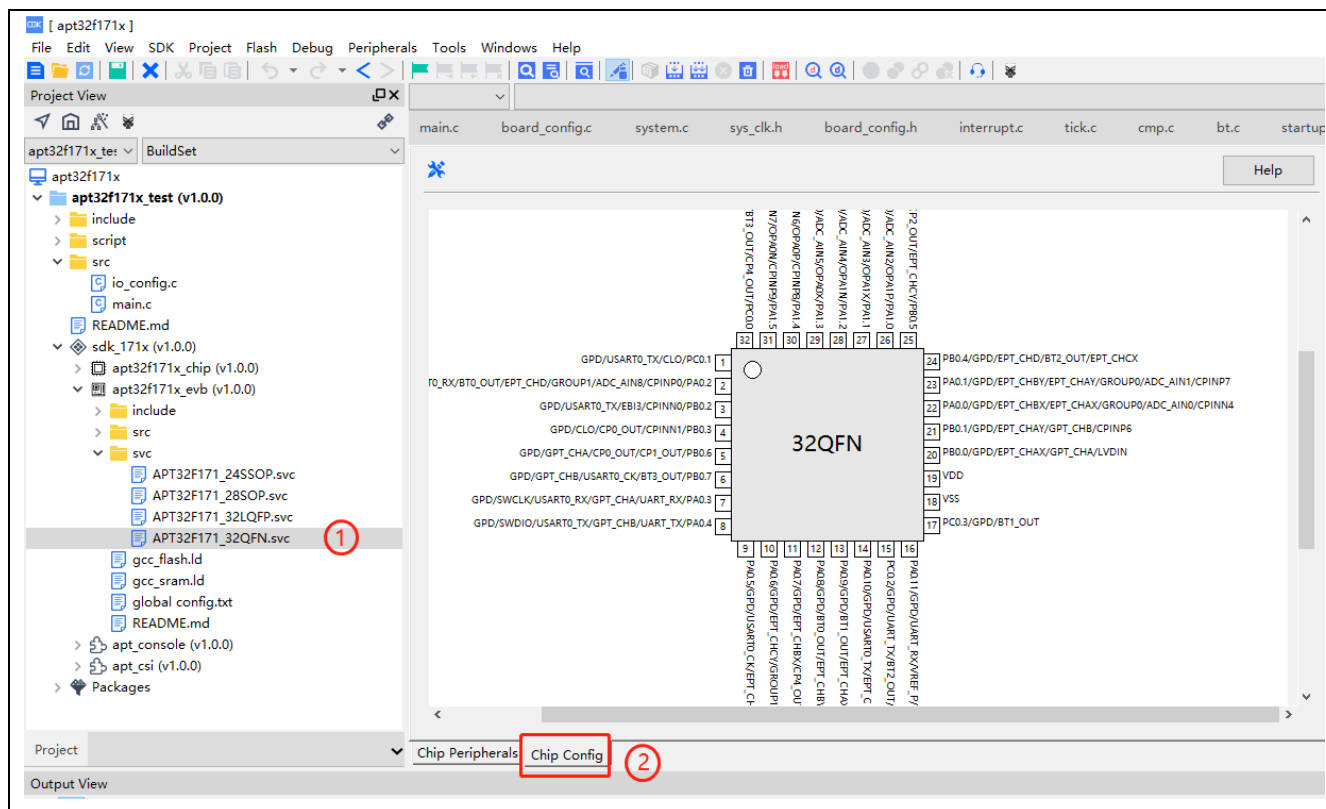
5.4.2 GPIO配置成复用功能

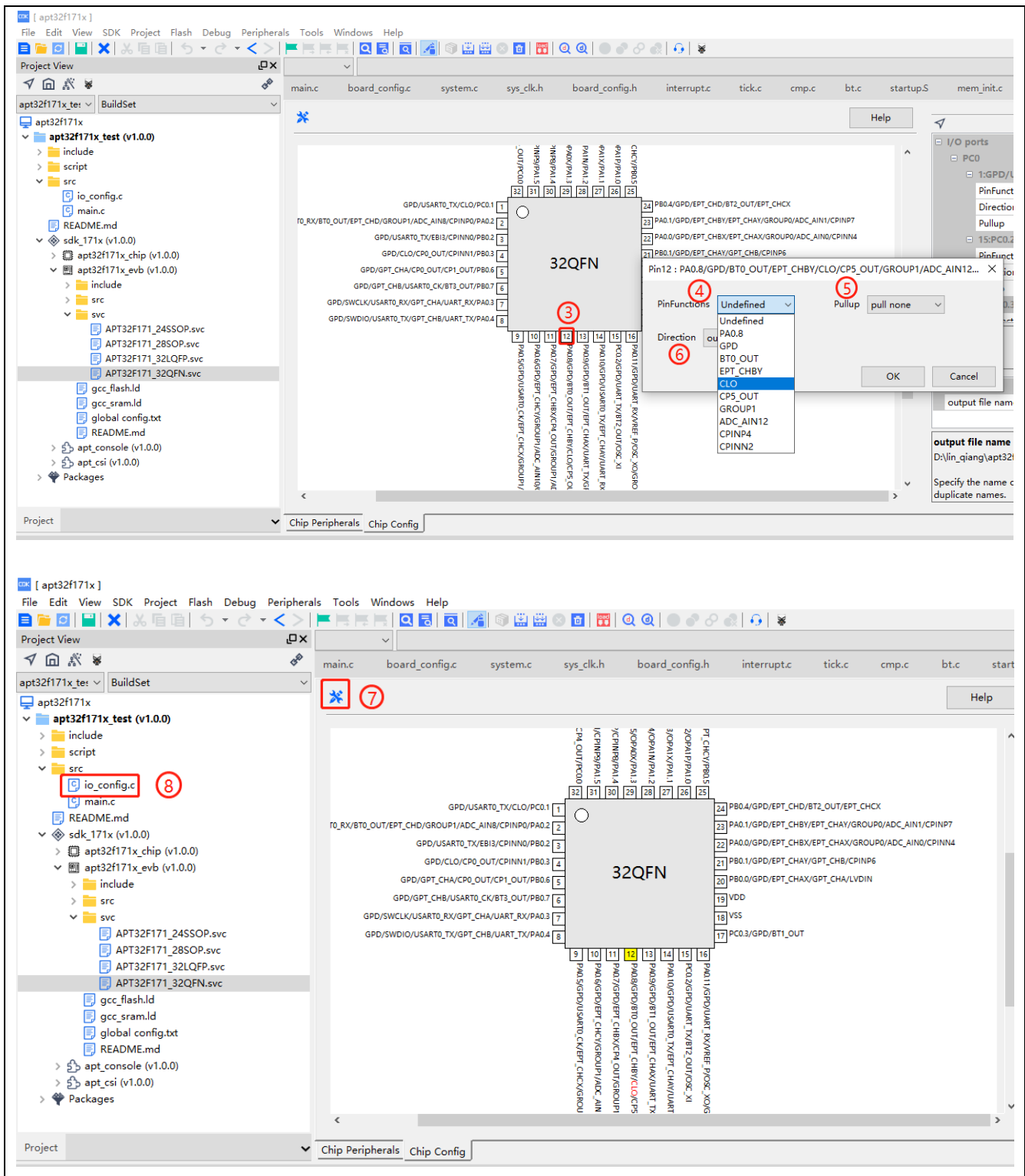
以下配置以APT32F171的QFN32封装为例（最大管脚资源），配置12脚（PA0.8）PinFunctions:CLO，

Pullup : pull none, Direction : output。

1. 点击图中红色1标注地方：apt32f171x_evb/svc下的APT32F171_QFN32；再点击红色2标注的地方Chip Config。
2. 双击图中红色3标注的管脚12，在弹出的对话框中红色标注4选择需要配置的功能，在红色标注5选择管脚的上拉/下拉/禁止上下拉功能
3. 点击红色标注6 设置输入还是输出，最后点击OK，完成配置。

- 再点击图中红色标注7的蓝色图标，在弹出对话框中点yes选项，将在工程根目录src文件夹下生成io_config.c文件。配置完成后，对应管脚的颜色将会变成黄色，对应选择功能会变成红色。
- io_config.c中生成两个函数PinConfigInit和__ChipInitHandler函数。配置语句为图中红色标注9，配置语句在PinConfigInit函数中，用户初始化时直接调用即可。
- 编译时候如果提示找不到对应参考函数，则需要添加工程目录下的io_config.c文件(添加一次即可)，本示例中，该文件在D:\lin_qiang\apt32f171\program\apt32f171x_cs\apt32f171x_test\src,详细操作参考标注10, 11, 12





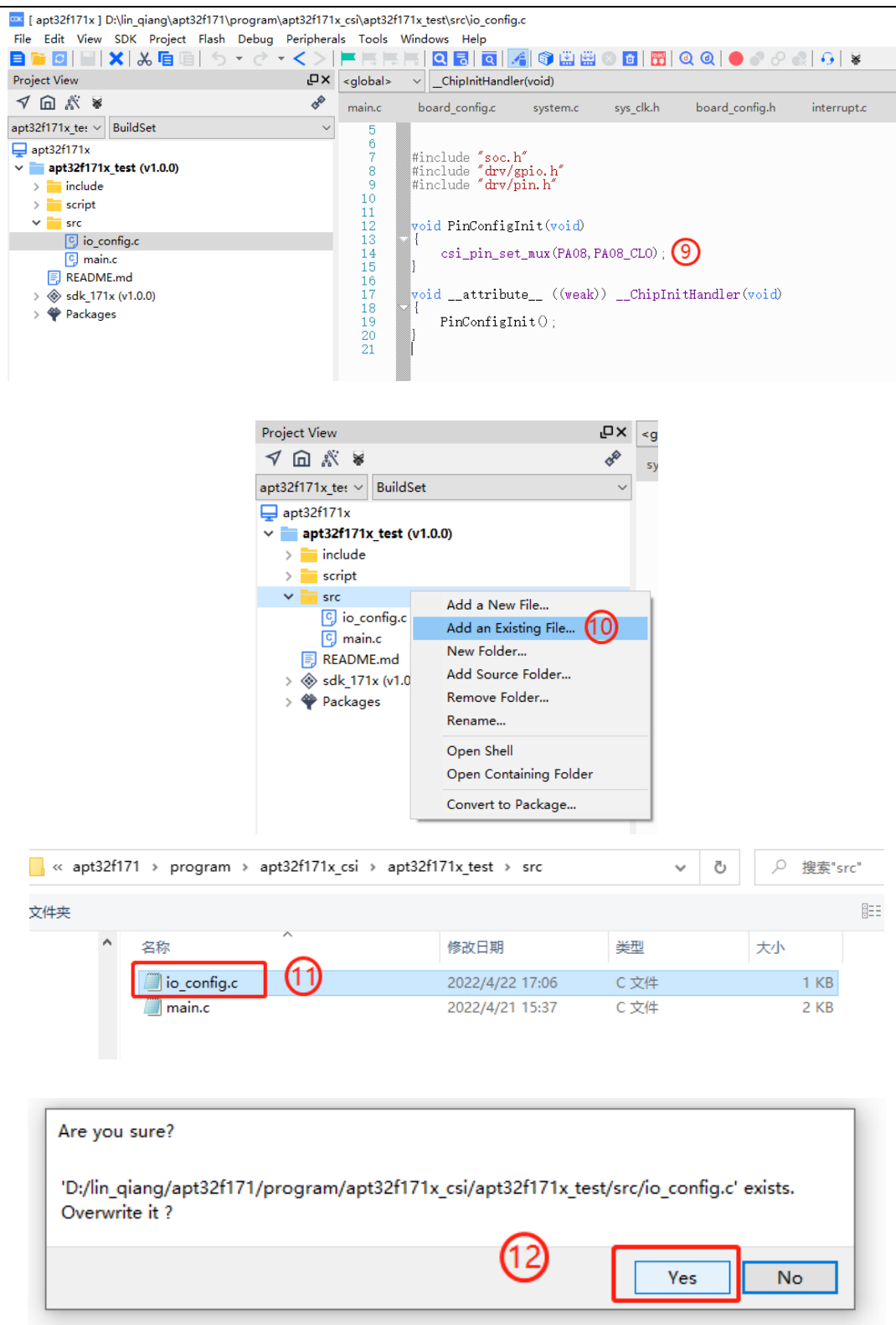


Figure 57 可视化配置步骤

5.4.3 GPIO配置为输入/输出

GPIO配置为输入/输出功能时，具体步骤和复用功能基本相同，步骤中不同的是配置步骤图片中的第二幅图，具体实例以配置32脚（PC0.1）为OUTPUT，下拉使能。

配置完成后，对应管脚的颜色将会变成黄色，对应选择功能会变成红色。io_config.c中函数PinConfigInit中将生成对应的配置语句。

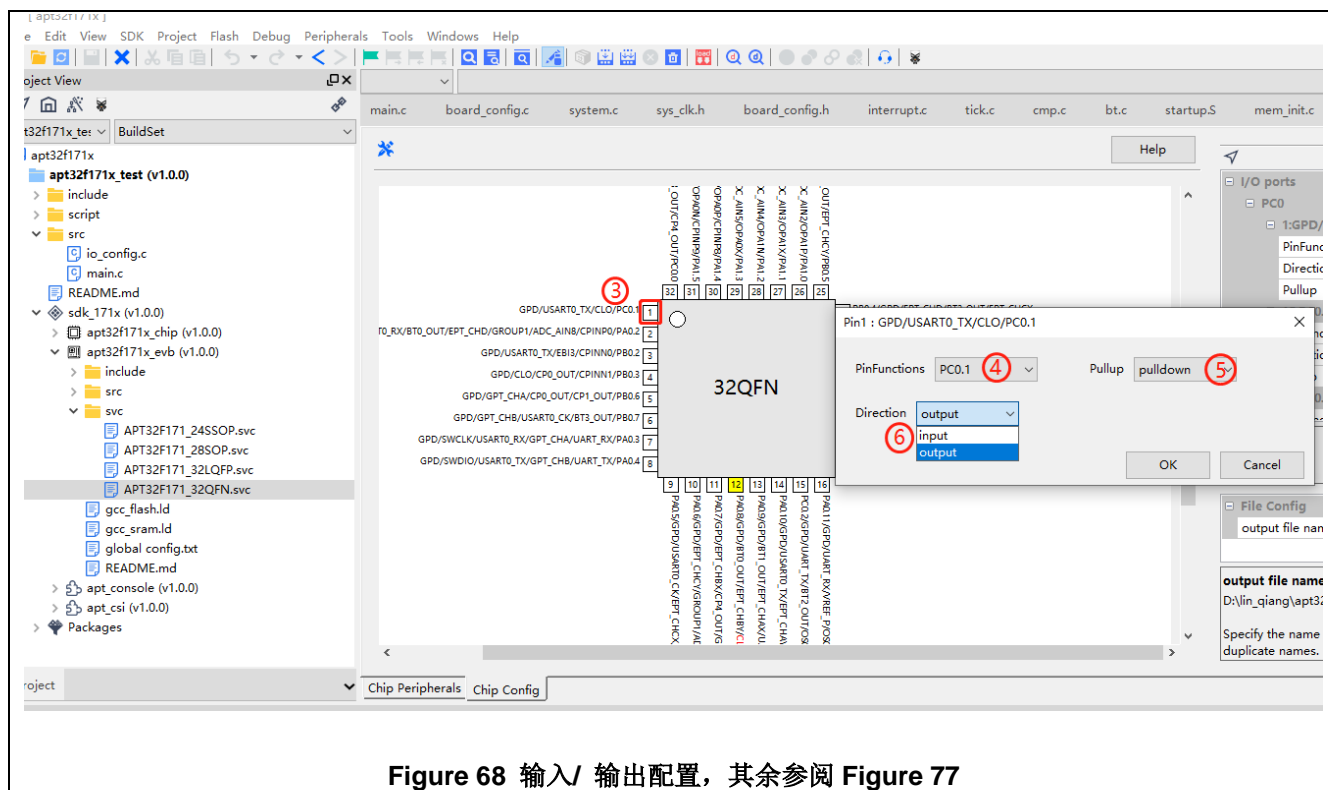


Figure 68 输入/输出配置，其余参阅 Figure 77

5.5 外设使用

外设使用的示例代码在APT32F171x_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。

5.6 编译工程

点击“Build Project”既可实现工程的编译和连接。

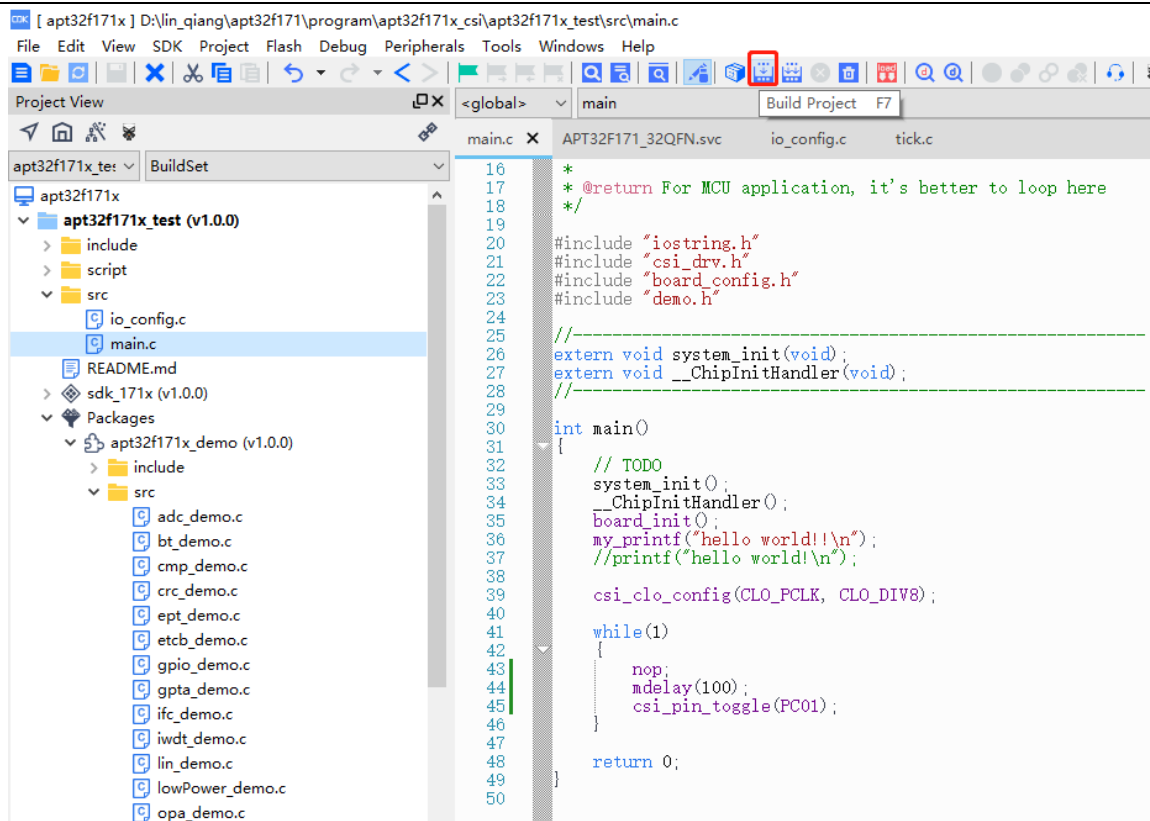


Figure 89 build

5.7 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. 仍然使用芯片内flash数据，直接进入debug模式（这个方式可用来回读芯片内flash内容）

下图中的1，2，3分别对应上面三种下载方式的操作菜单。

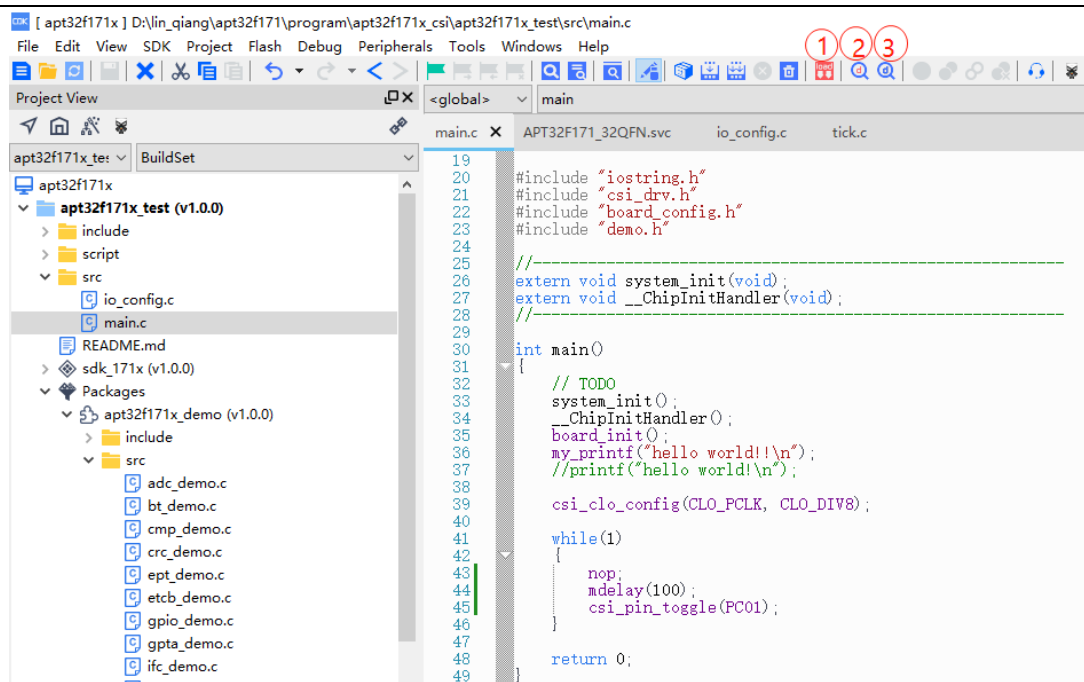


Figure 20 下载调试的三种选择