版本	V1.0
日期	202201

Quick Start

APT32F110x 系列

Revision History

版本	日期	描述	作者
V1.0	2022-1-14	新建使用说明	ZJY
V1.1	2022-2-24	更新GPIO可视化配置部分	ZJY

目录

1. 文档说明

该文档基于爱普特CSI驱动代码架构，适用于APT32F110x系列内所有的产品型号。包含：APT32F1101、APT32F1102、APT32F11103和APT32F1104。

2. 资源列表

为了使用APT32F110x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102x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。
    - APT32F110x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（包含touch静态库*）及模块示例代码。

* 如果使用touch功能，请确保使用最新版本的库文件，在工程设置->linker下手动添加或替换老版本静态库。下载地址：https://occ.t-head.cn/vendor/detail/download?id=393797265421239500 8&vendorId=371293494720092160 0&module=3#sticky

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 APT32F110x 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。

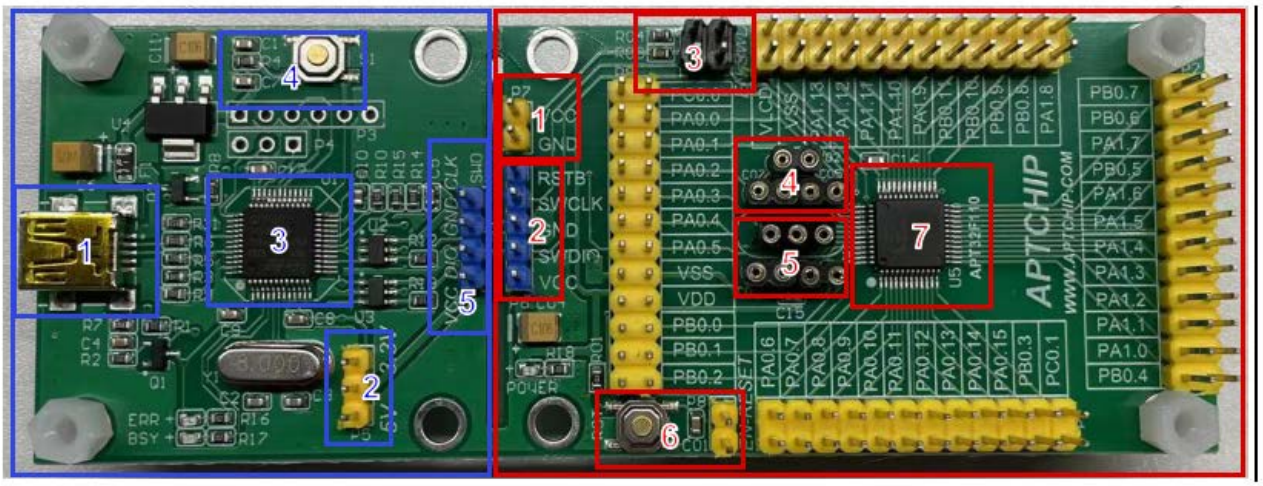


Figure 1 开发板实物图

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。APT32F110x 系列只有一组 SWD 接口。SWD 口为 PA1.8（SWCLK）和 PA1.7（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 PA1.7 和 PA1.8。使用时需用跳线连接

4、 晶振电路和跳线。APT32F110x 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。APT32F110x 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。APT32F110x 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。APT32F110x 系列芯片。

4. 软件环境

4.1 CDK开发环境

APT32F110x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7个组件。

apt32f110x是工程组件，客户的业务逻辑代码一般放在这个组件下面。

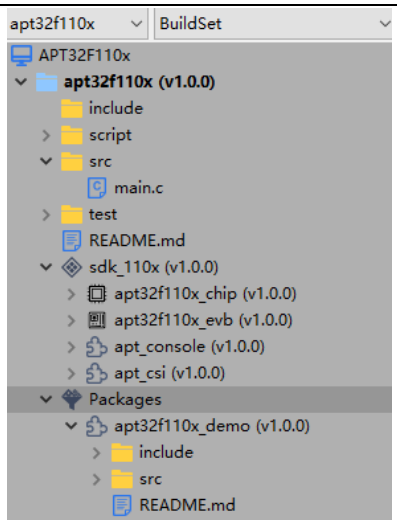


Figure 2 代码结构

5. 例程运行

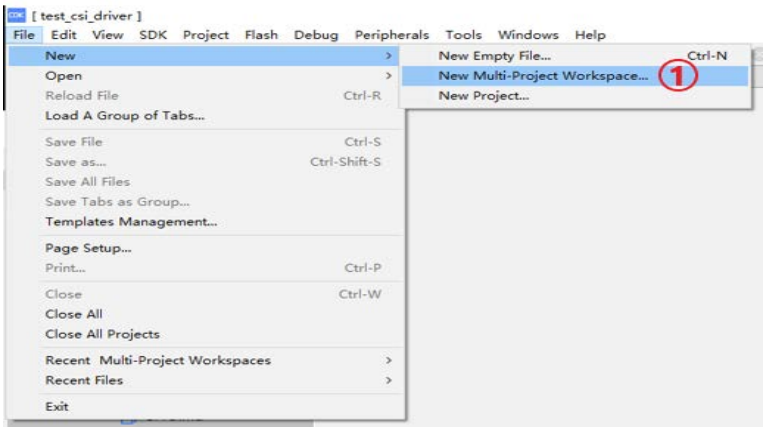
5.1 例程获取方式

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:\03_MysoftAreas\APT32F110x），将110SDK包放到此文件夹下。



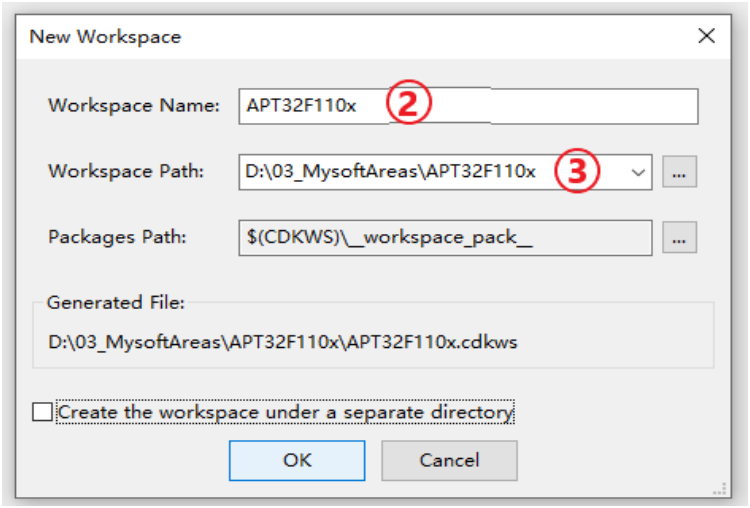
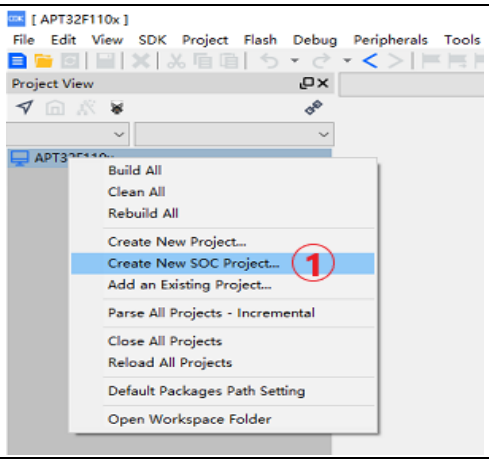


Figure 3 创建新的 workspace

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



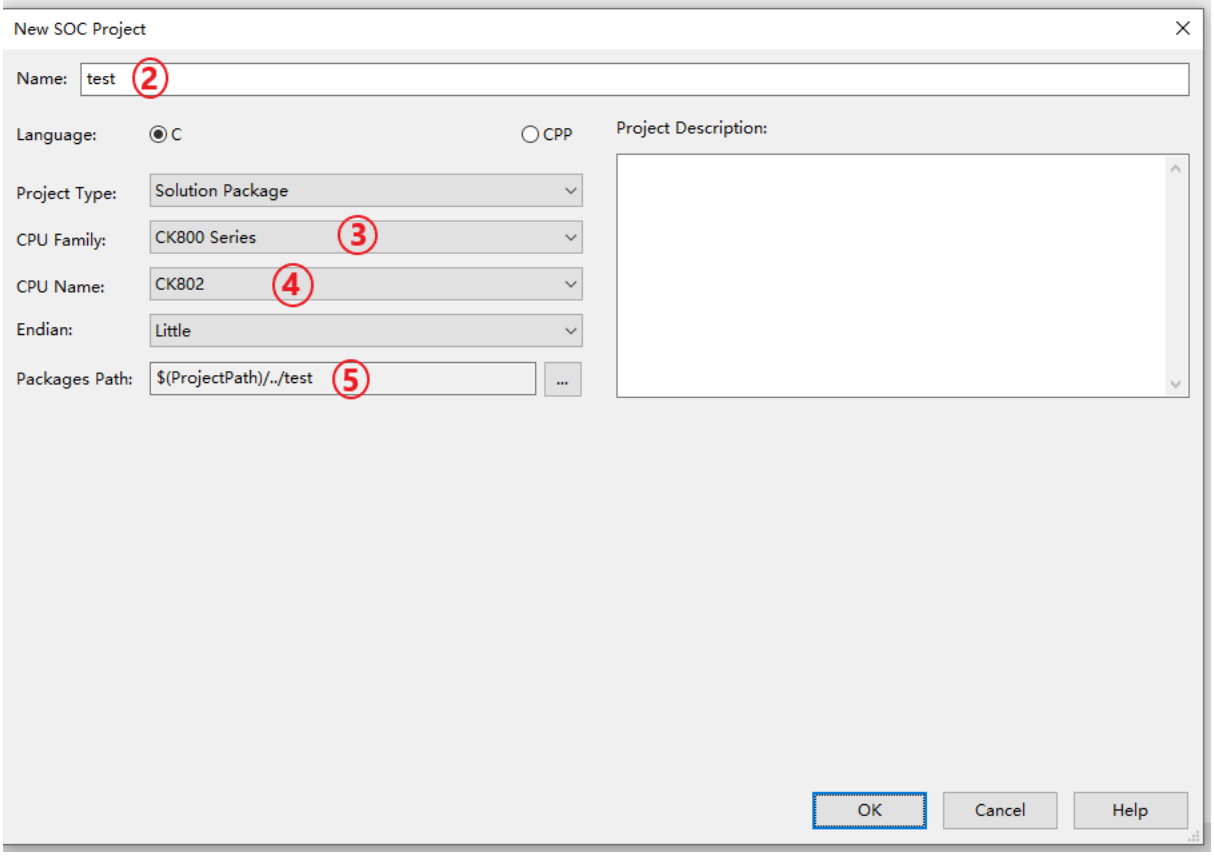
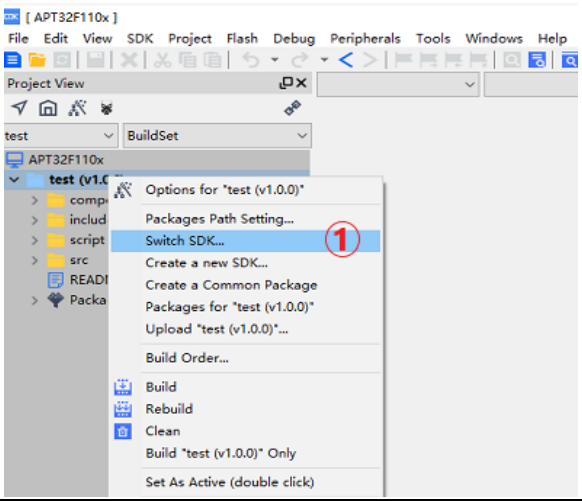
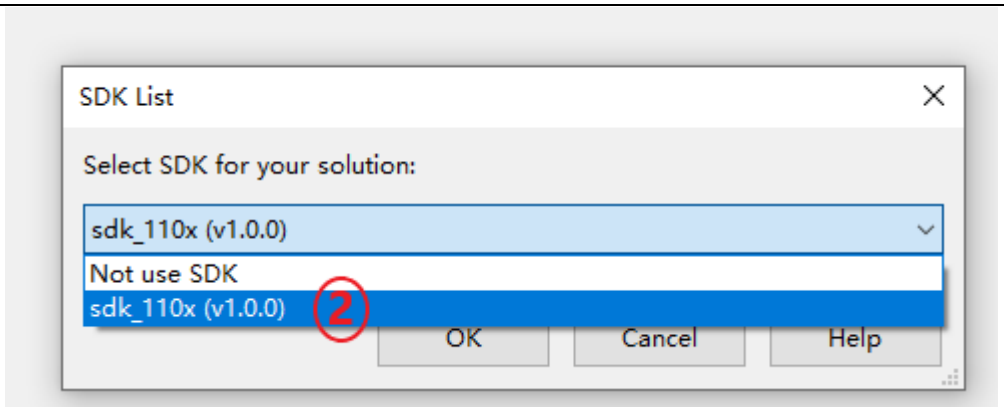


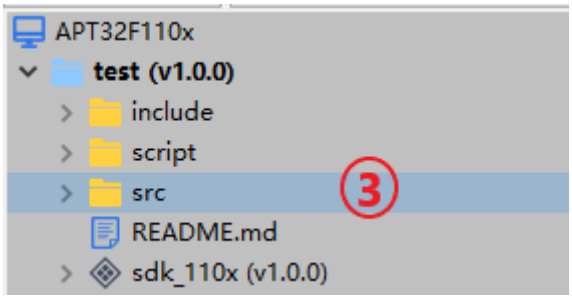
Figure 4 创建新的 soc project

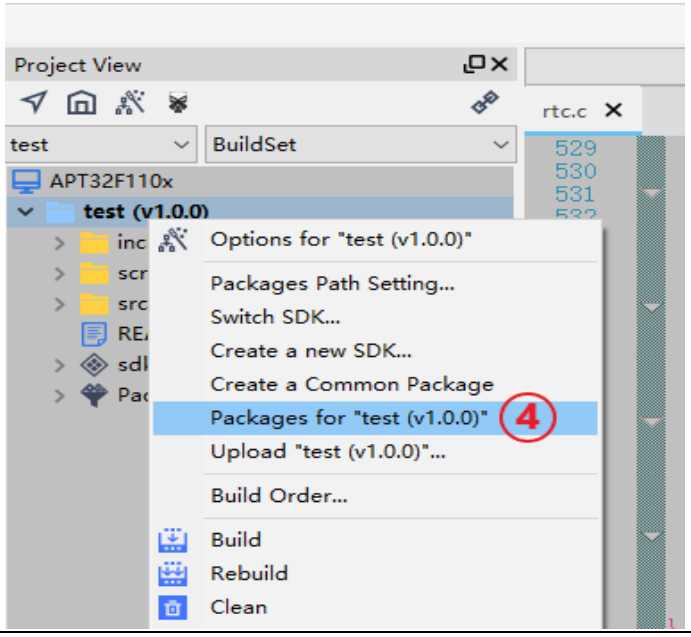
3、 一键切换SDK。

1. 按照图中红色标注1右键点击test工程，选择切换SDK。切换完成后，建议在工程目录下删除__SDK_CK801和__CHIP_CK801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7、8、9选择添加，选择点击OK，test工程建立完毕。




SDK List
Select SDK for your solution:
sdk_110x (v1.0.0)
Not use SDK
sdk_110x (v1.0.0)
2
OK
Cancel
Help


APT32F110x
test (v1.0.0)
include
script
src
3
README.md
sdk_110x (v1.0.0)


Project View
rtc.c
test
BuildSet
APT32F110x
test (v1.0.0)
Options for "test (v1.0.0)"
Packages Path Setting...
Switch SDK...
Create a new SDK...
Create a Common Package
Packages for "test (v1.0.0)"
4
Upload "test (v1.0.0)"...
Build Order...
Build
Rebuild
Clean

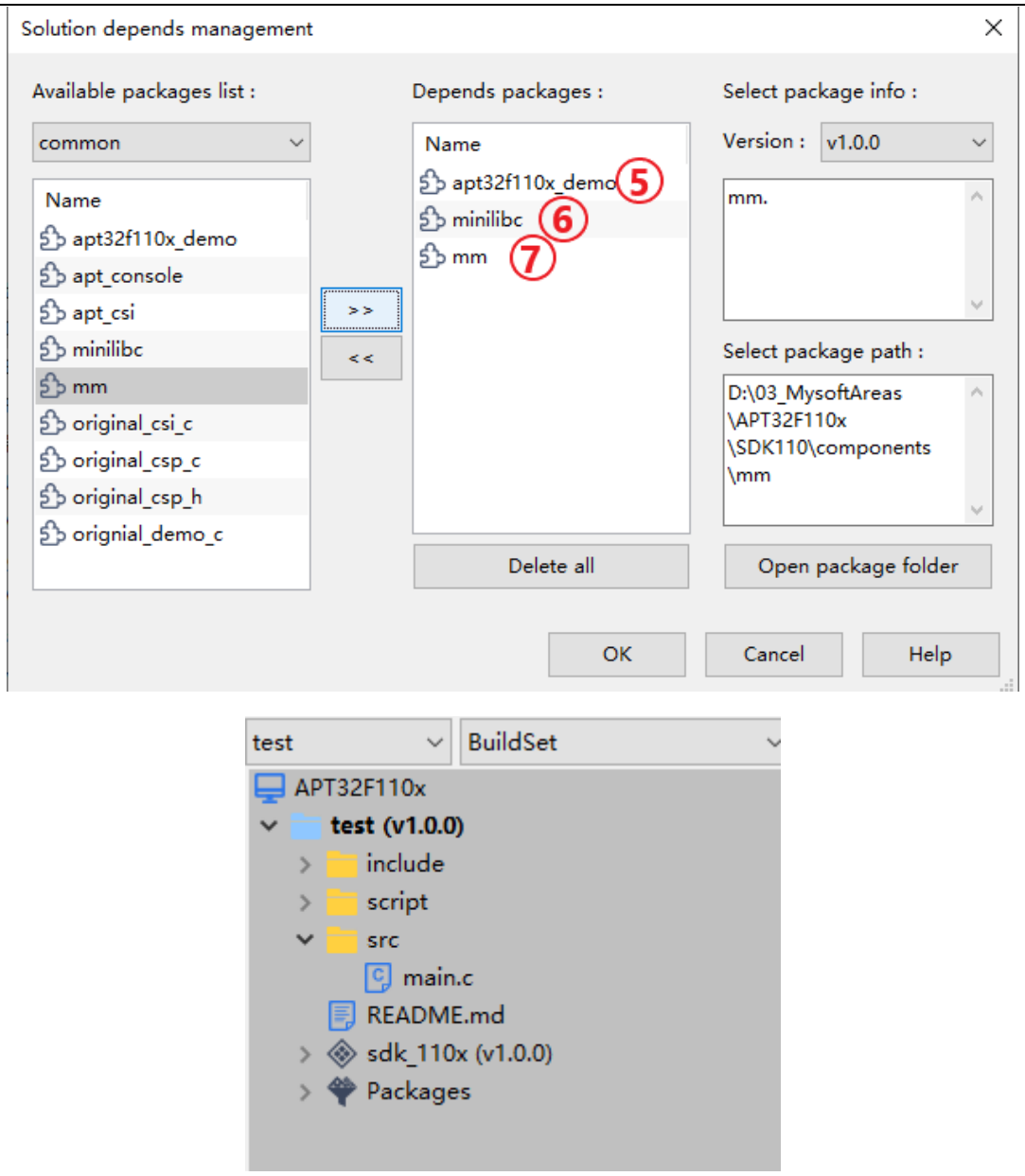


Figure 5 SDK 切换、Package 创建

5.1.3 工程配置

1. 点击图中1，弹出配置对话框
2. 2、3按照图中配置所选择
3. 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。配置完成，编译即可

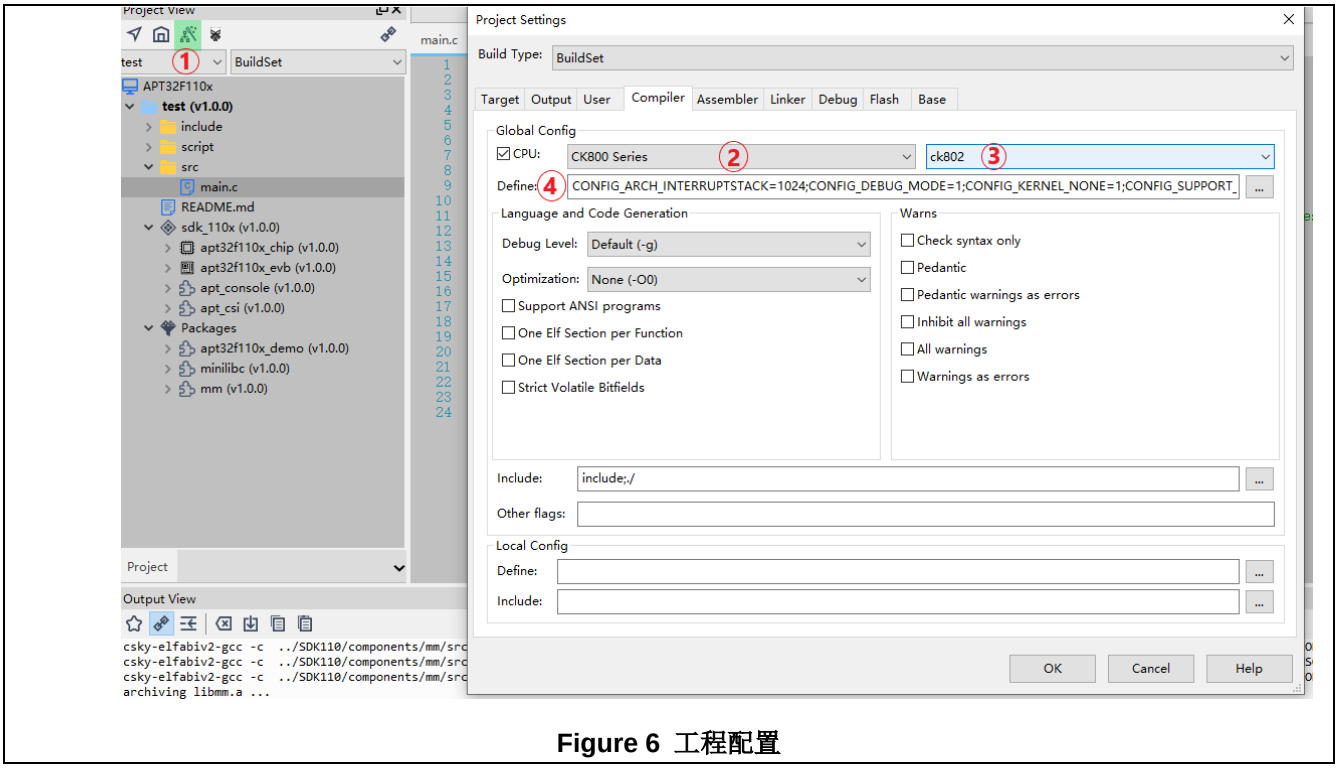


Figure 6 工程配置

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 选择芯片型号

APT32F110x是一个系列，内部还有很多型号。请根据您所使用的芯片型号，定义不同的全局宏。

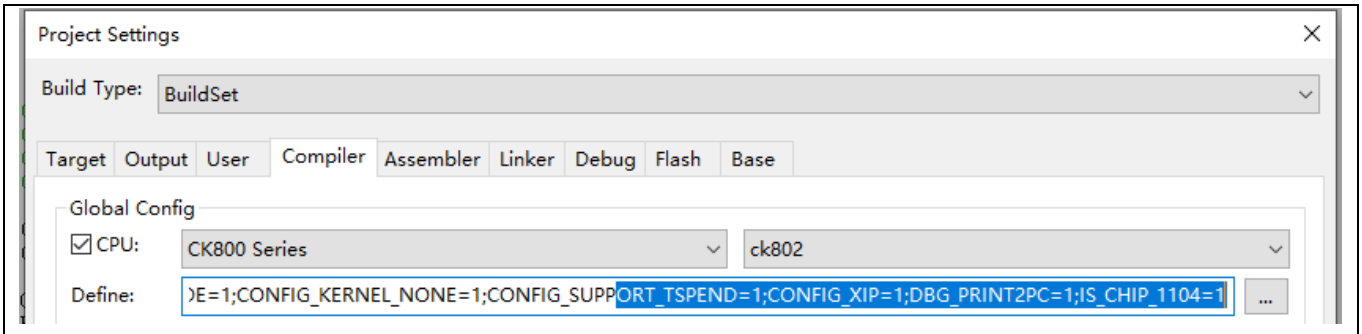


Figure 7 选择芯片型号

- APT32F1101：IS_CHIP_1101=1
- APT32F1102：IS_CHIP_1102=1
- APT32F1103：IS_CHIP_1103=1
- APT32F1104：IS_CHIP_1104=1

5.4 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。

```
int main()
{
    tClkConfig.eClkSrc = SRC_HFOSC;
    tClkConfig.wFreq = HFOSC_48M_VALUE;
    tClkConfig.eSdiv = 2;
    tClkConfig.ePdiv = 1;
    system_init();

    board_init();
    my_printf("hello~~~\n");
```

Figure 8 main 函数

5.4.1 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()	__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
	}
		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_HFOSC, HFOSC_48M_VALUE, SCLK_DIV2, PCLK_DIV1,5556000, 5556000};
    //{SRC_IMOSC, IMOSC_4M_VALUE, SCLK_DIV1, PCLK_DIV1,5556000, 5556000};

    //{SRC_HFOSC, HFOSC_48M_VALUE, SCLK_DIV2, PCLK_DIV1,5556000, 5556000};
    //{SRC_IMOSC, IMOSC_4M_VALUE, SCLK_DIV1, PCLK_DIV1,5556000, 5556000};
```

Figure 3 时钟配置结构体

5.4.2 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。使用时需要删除全局宏DBG_PRINT2PC=1。

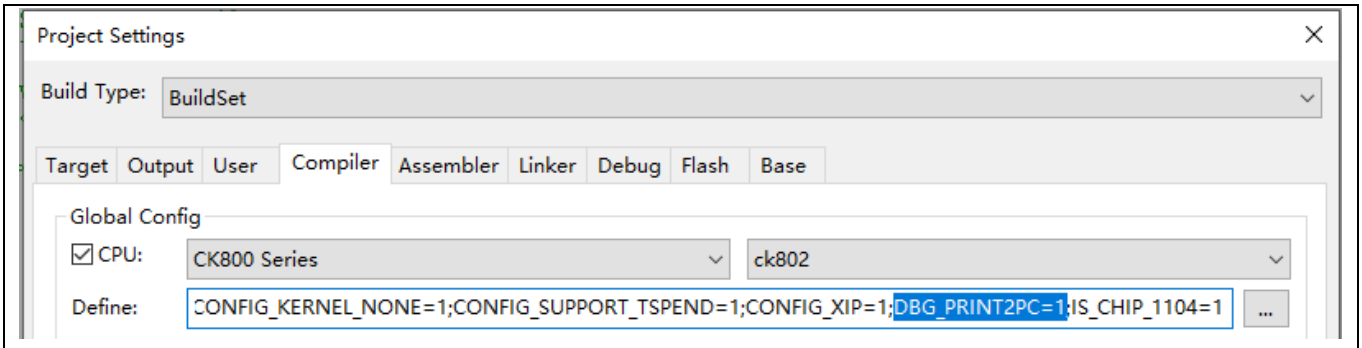


Figure 4 DBG_PRINT2PC 宏

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UART2，PA111（TXD）和PA112（RXD）。注意查看MCU 型号是否有UART2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。

函数	说明	位置

board_init()	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。
	__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);
	}
		board_config.c

5.4.3 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SPI。

```
void spi0_int_andler(void)
{
    // ISR content ...
    spi_irqhandler(SPI0);
}
```

Figure 5 中断处理函数举例

在上述例子中，spi_irqhandler()函数在驱动代码（spi.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SPI0IntHandler（）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 ifc_irqhandler (void)

2、 srartup.S中将原来ifc_irqhandler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。

5.5 GPIO可视化配置

5.5.1 简介

110x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110x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。

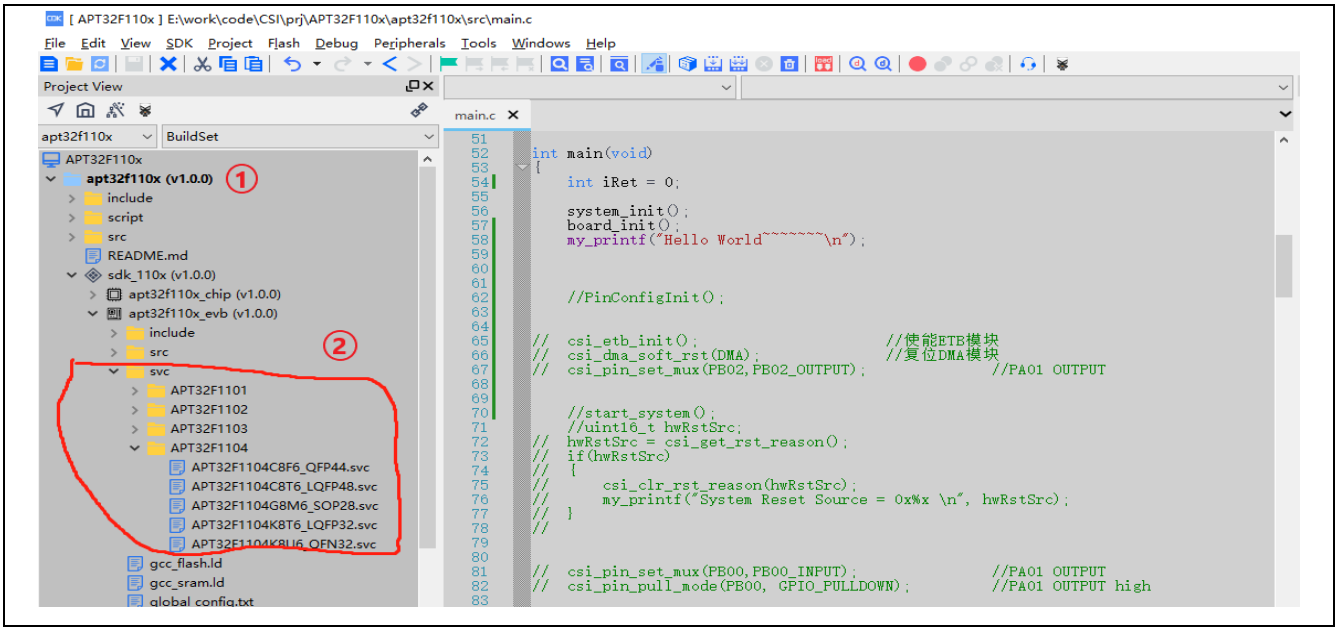
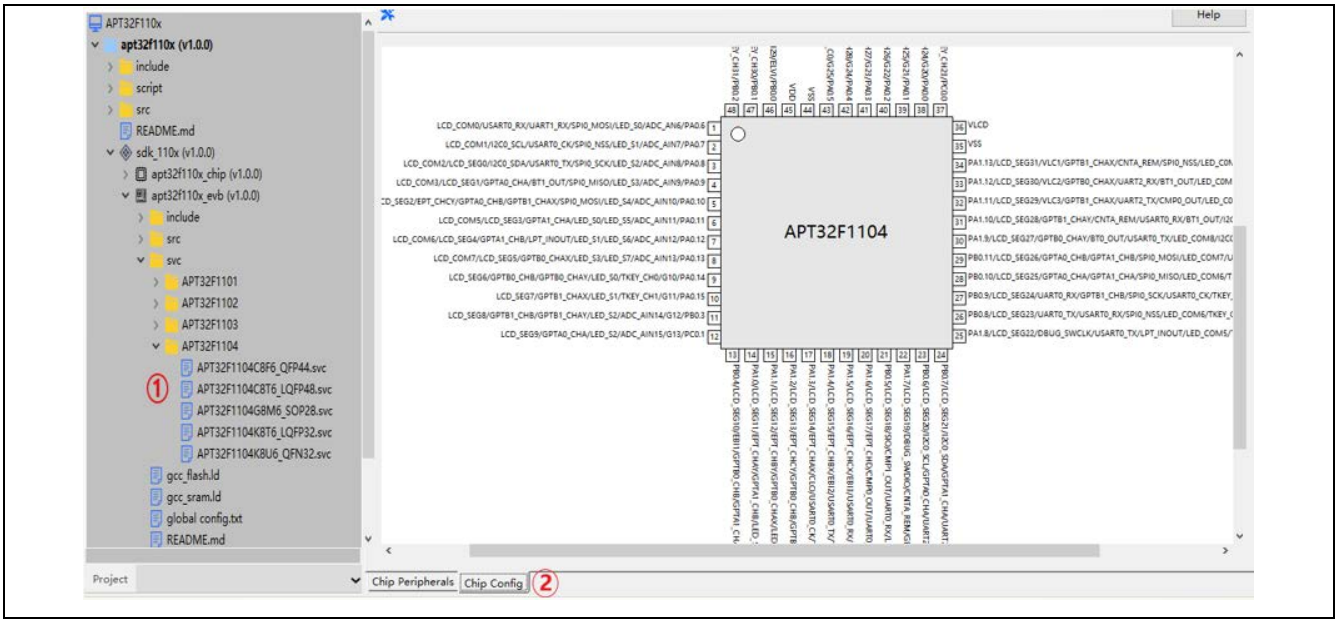


Figure 6 封装信息

5.5.2 GPIO配置成复用功能

以下配置以APT32F1104的LQFP48封装为例（最大管脚资源），配置1脚（PA0.6）为UART1_RX，上拉使能。

1. 点击图中红色1标注地方：svc/APT32F1104下的APT32F1104C8T6_LQFP48；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1，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 OK，完成配置。
4. 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。
5. 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和10，配置语句在PinConfigInit函数中，用户初始化时直接调用即可。



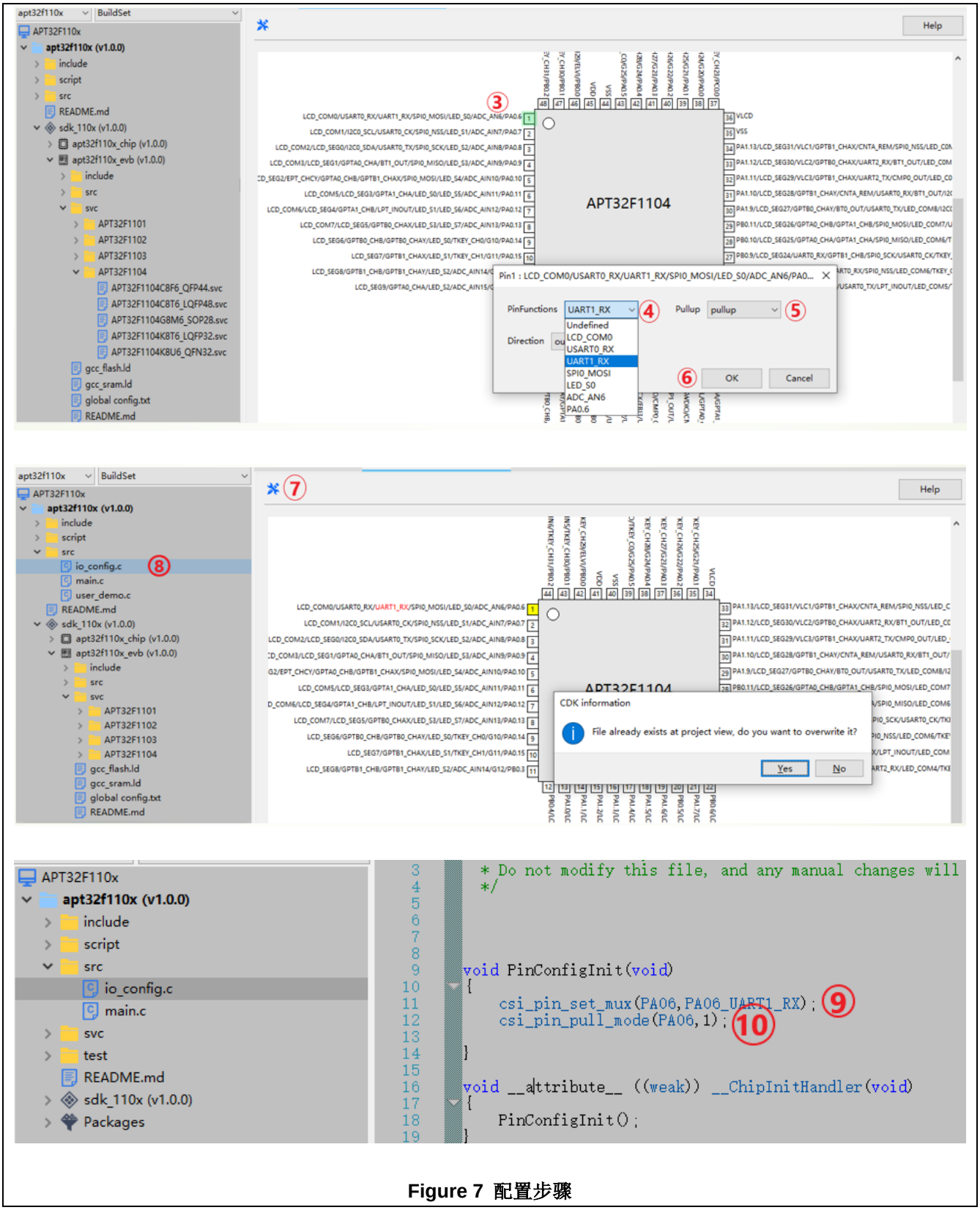


Figure 7 配置步骤

5.5.3 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48脚（PB0.2）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。

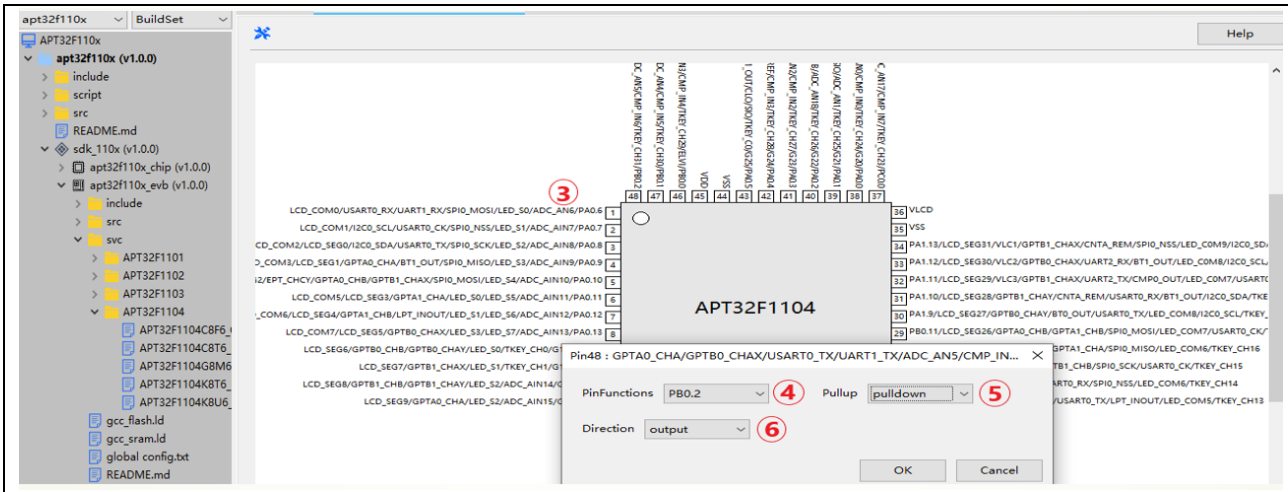


Figure 8 输入/ 输出配置中不同地方，其余参阅 Figure 9

5.6 外设使用

外设使用的示例代码在APT32F110x_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。

5.7 编译工程

点击“Build Project”既可实现工程的编译和连接。

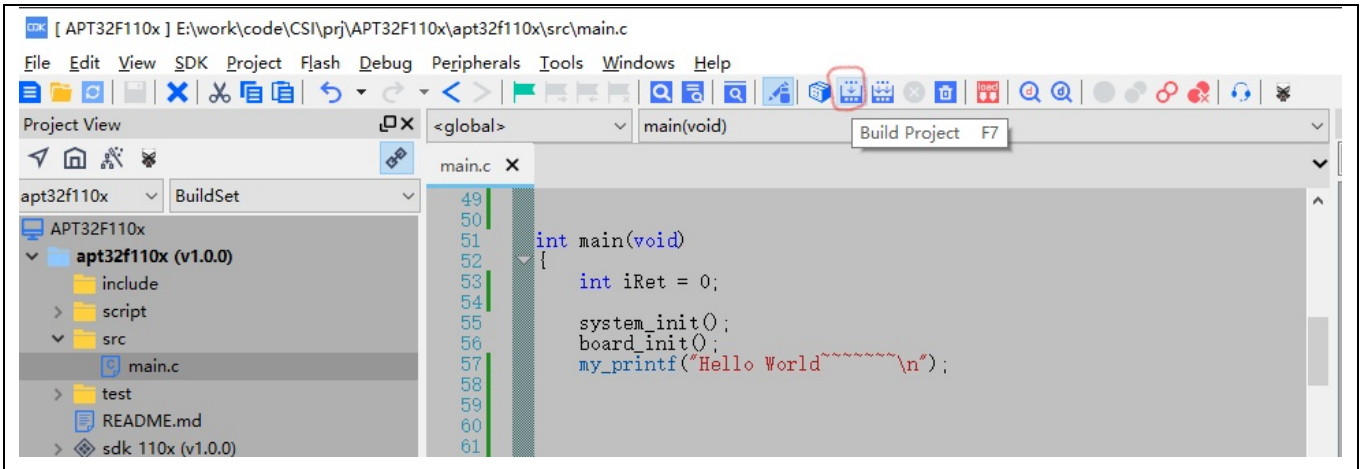


Figure 10 build

5.8 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，随后进入debug模式
2. 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）
3. 将代码下载到flash区，不进入debug模式

下图中的1，2，3分别对应上面三种下载方式的操作菜单。

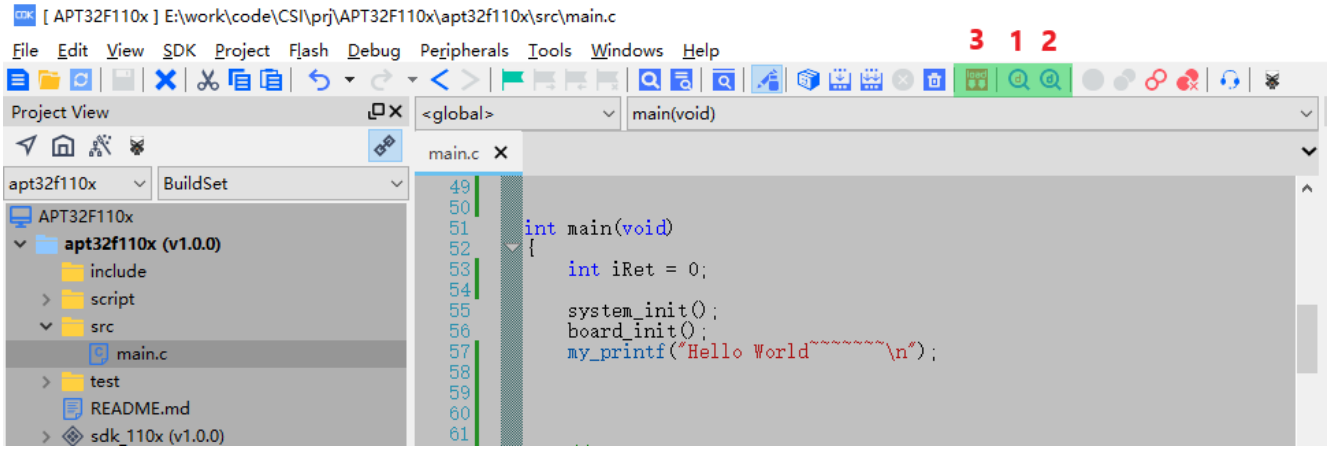


Figure 11 下载调试的三种选择