版本	V2.0
日期	202110

APT32F110x 系列

CSI 代码结构和使用说明

相关文档：

APT32F1101/APT32F1102/APT32F1103/APT32F1104 数据手册

APT32F110x系列使用手册

QuickStart_APT32F110x系列_CSI

APT32F110x系列CSI_API说明手册

剑池CDK V2.6版本特性

Revision History

版本	日期	描述	作者
V1.0	2022-1-14	新建	ZJY
V1.1	2022-1-24	修改部分错误	ZJY

目录

1. 概述

本文主要说明于 APT32F110x 系列 CSI（Chip Standard Interface）代码结构和使用说明。CSI 充分考虑应用，将 csp 层驱动代码合理组织打包。

2. 代码结构

在CDK中，APT提供的开发例程由7个组件组成，分别是工程组件（demo）、sdk组件集合（内含chip组件、board组件、console组件、csi组件）和demo组件。当然用户可以自定义自己的组件，对代码进行合理打包，方便管理。

组件名后面的数字（如v1.0.0）代表版本号，版本release信息可以在对应组件的README.md中查看。

Figure 1 CDK 工程视图（全部）

其中工程组件对应用户的工程，可以根据需要调整目录结构。建议在 CDK 视图下新建、组织或删除文件/文件夹，否则编译可能出错，windows 文件目录会同步更新。

Board组件（apt32f110x_evb）

Board 集合了所有用户工程可能需要更改的代码。

Figure 2 board 组件说明

2.1 Chip组件（apt32f10x_chip）

chip组件是驱动代码的主体。

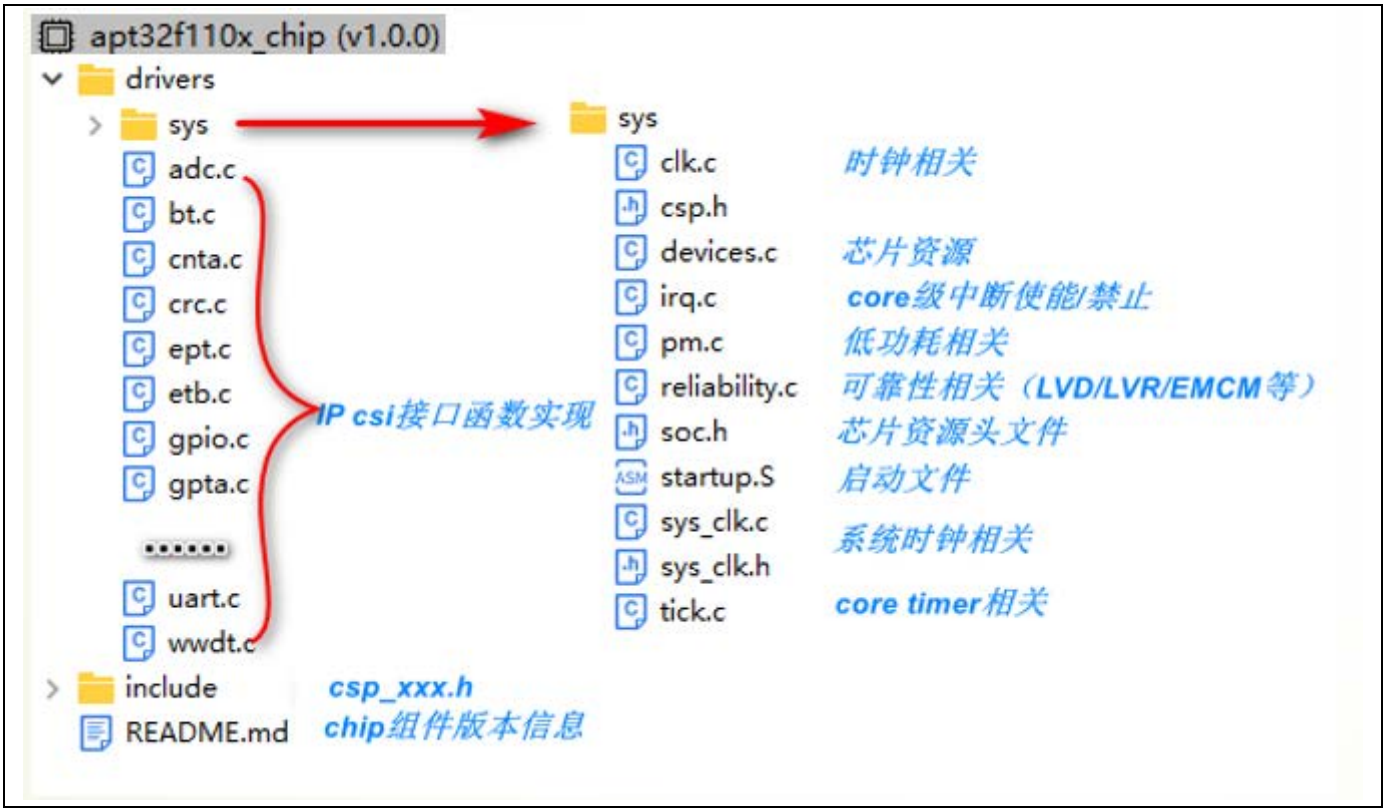

Figure 3 chip 组件说明

举几个例子说明

2.1.1 ADC

和 adc 相关的代码有三个 csp_adc.h, adc.c，adc.h,。前两个在 chip 组件下，第三个在 csi 组件下。用户代码中如果使用 ADC，只需要#include “adc.h”即可。

- adc.h 声明了所有adc.c 里实现的csi接口函数。
- adc.c 实现了所有adc 相关的csi接口函数，除内部调用的函数外，函数名全部以csi_adc_xxx开头。
- csp_adc.h 定义了寄存器结构体、底层的宏定义、枚举和一些以内联函数来实现的寄存器操作函数。如果csi接口函数可以满足应用需求，可以忽略这个文件的存在。

2.1.2 系统可靠性

系统可靠性包含关于LVD，LVR，RSR，EMCM，memory check等内容。
和系统可靠性（reliability）相关的代码有csp_syscon.h，reliability.c，reliability.h。前两个在chip组件下，第三个在csi组件下。用户代码中如果使用reliability相关函数，只需要#include “reliability.h”即可。

- reliability.h 声明了所有reliability.c 里实现的csi接口函数。
- reliability.c实现了所有和系统可靠性相关的csi接口函数，除内部调用的函数外，函数名以 csi_lvd/lvr/emcm/sramcheck/flashcheck_xxx 开头。
- csp_syscon.h 定义了寄存器结构体、底层的宏定义、有效设置值的枚举和一些以内联函数来实现的寄存器操作函数。

2.2 Demo组件（apt32f110x_demo）

Demo组件用来给用户示例芯片内集成IP的使用方法。用户可以参考demo组件中的示例，自己在工程组件中组织代码。最终的工程中可以将demo组件整个移除。移除方法如下：

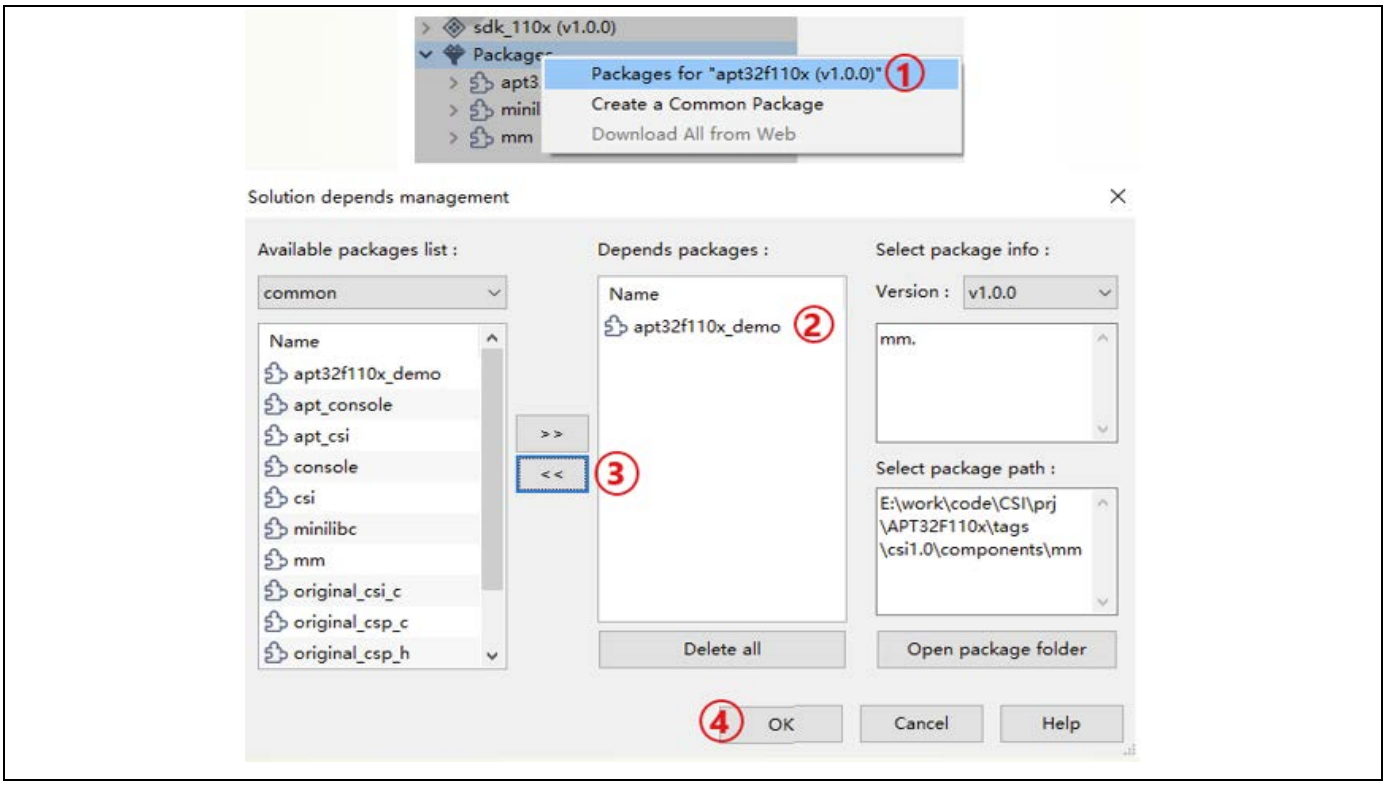

Figure 4 移除 demo 组件的方法

3. 代码移植

如Board组件（apt32f110x_evb）所述，代码移植时需要修改的文件一般都位于board组件中。

3.1 管脚资源分配

工程视图中，需要修改的文件位于：sdk_110x -> apt32f1110x_evb-> include -> board_config.h

实际文件路径为：工程目录__workspace_pack__\apt32f110x_evb\v1.0.0\include

这个文件中，要特别注意调试口console的部分必须正确。否则调用my_printf()将不会有正常输出。

```
/* example pin manager */

#define CONSOLE_IDX                     2
#define CONSOLE_TXD                     PA111
#define CONSOLE_RXD                     PA112
#define CONSOLE_TXD_FUNC                PA111_UART2_TX
#define CONSOLE_RXD_FUNC                PA112_UART2_RX

#define XIN_PIN                         PA03
#define XOUT_PIN                        PA04
#define XIN_PIN_FUNC                    PA03_OSC_XI
#define XOUT_PIN_FUNC                   PA04_OSC_XO

#define SXIN_PIN                        PA01
#define SXOUT_PIN                       PA02
#define SXIN_PIN_FUNC                   PA01_OSC_SXI
#define SXOUT_PIN_FUNC                  PA02_OSC_SXO
```

Figure 5 管脚资源配置示例

3.2 时钟设置

工程视图中，需要修改的文件位于：sdk_110x -> apt32f110x_evb -> src -> board_config.c

实际文件路径为：工程目录\board\src

```
/// 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};
```

Figure 6 芯片时钟配置

结构体tClkConfig包含4个成员，分别是时钟源、时钟源频率，SCLK分频值和PCLK分频值。可以在sys_clk.h（sdk_110x -> apt32f110x_chip -> sys）中查询枚举值。

当然，也可以在初始化前给变量tClkConfig重新赋值，再进行初始化。

3.3 初始化

3.3.1 上电初始化

芯片上电后，启动文件（startup.s）会调用__main()完成SRAM数据初始化，然后跳转main（）进行系统和调试口初始化。

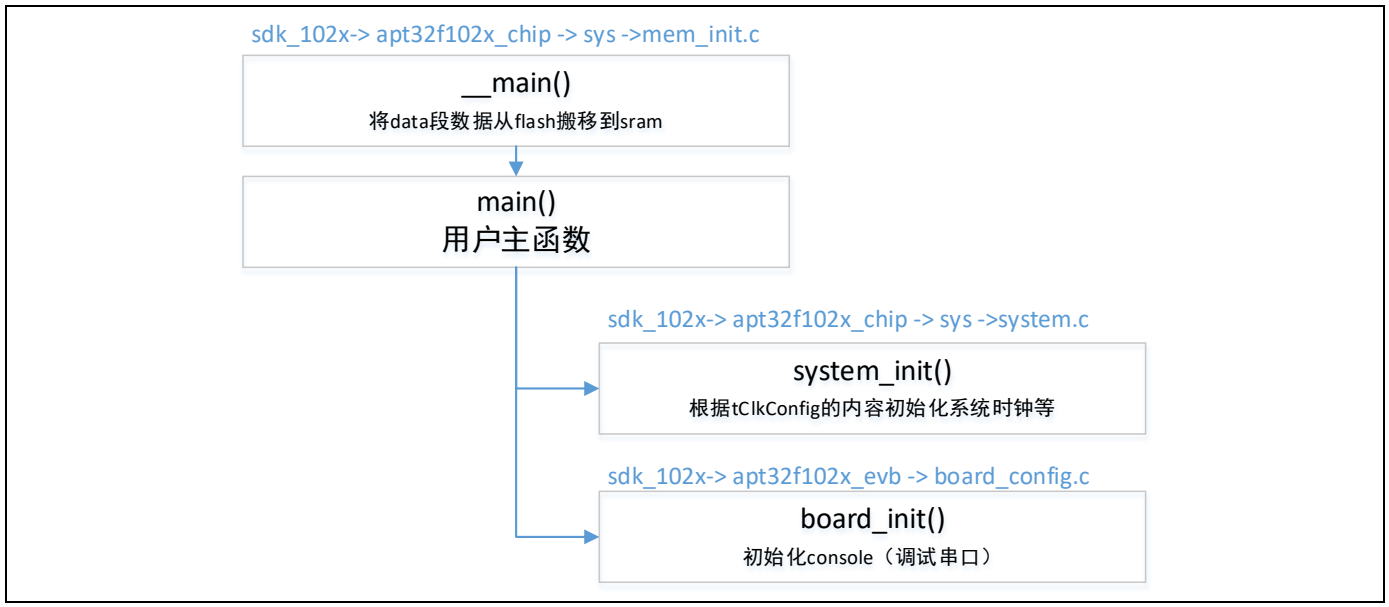

Figure 7 上电启动流程

main() 开始的时候调用system_init() 实现程序正式运行前的准备工作，需要包含但不限于下述内容。

```
__attribute__((weak)) void system_init(void)
{
    CK_CPU_DISALLNORMALIRQ();

    csi_iwdt_close();                   //close iwdt
    csi_sysclk_config();                //sysclk config
    csi_tick_init();

    CK_CPU_ENALLNORMALIRQ();
}
```

weak函数，一旦工程其他地方有重新定义system_init，此段代码会被编译器忽略
关闭全局中断
关闭IWDT，视应用保留或删除
系统时钟配置，需要事先完成对结构体变量tClkConfig的初始化（时钟参数）
初始化systick（CORET），以实现mdelay等函数，视应用保留或删除
使能全局中断，视应用保留 或删除

Figure 8 system_init()函数

3.3.2 管脚初始化

所有模块的csi 接口函数都不包括管脚功能设定的部分，需要在模块使用前用以下函数指定管脚的AF功能。可以用资源分配章节里提到的board_config.h中的宏定义。

```
csi_pin_set_mux(ADC_PIN, ADC_PIN_FUNC);
```

Figure 9 管脚初始化示例

4. Q & A

4.1 Q1：如果CSI的代码架构不能满足系统对实时性的要求怎么办？

A1：有好几种方法可以提高系统实时性。

1) 使用ETCB设备

尽可能使用 ETCB 设备，设置好输入和输出，实现不同设备间的硬件互动。

2) 把 csp.h 加到对执行速度敏感的源文件中，就可以在这个文件内调动 csp 接口函数或者直接进行寄存器操作。

4.2 Q2：如果CSI现有代码（API）不能满足特定的应用场合，怎么办？

A2：原则上，用户只调用 CSI 的 API 接口。但因为标准代码的覆盖率有限，所以可能会出现现有 API 不满足特定应用场合的情况。解决方法：把 csp.h 加到源文件中，就可以直接调用 csp 的接口，或者进行寄存器操作。

4.3 Q3：CSI代码可以在老版本CDK上运行吗？

A3：CSI代码结构和CDK版本并没有直接的关系。只是CDK2.6（含）以上的版本支持组件的概念，所以工程形式上存在差异。