

版本	V1.0
日期	202308



APT32F103x 系列

CSI 代码结构和使用说明

相关文档:

APT32F1031 数据手册

APT32F103x系列使用手册

QuickStart_APT32F103x系列_CSI

APT32F103x系列CSI_API说明手册

剑池CDK V2.6版本特性

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	2023-08-04	新建	GQQ

目录

1. 概述 1

2. 代码结构 1

 2.1 Board 组件（APT32F103x_evb） 1

 2.2 Chip 组件（APT32F103x_chip） 2

 2.2.1 ADC..... 3

 2.2.2 系统可靠性 3

 2.3 Demo 组件（APT32F103x_demo） 3

3. 代码移植 4

 3.1 管脚资源分配..... 4

 3.2 时钟设置 5

 3.3 初始化..... 5

 3.3.1 上电初始化 5

 3.3.2 管脚初始化 6

4. Q & A 6

 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？ 6

 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？ 6

 4.3 Q3：CSI 代码可以在老版本 CDK 上运行吗？ 6

1. 概述

本文主要说明于 APT32F103x 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（APT32F103x_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 (apt32f103x_chip)

chip组件是驱动代码的主体。

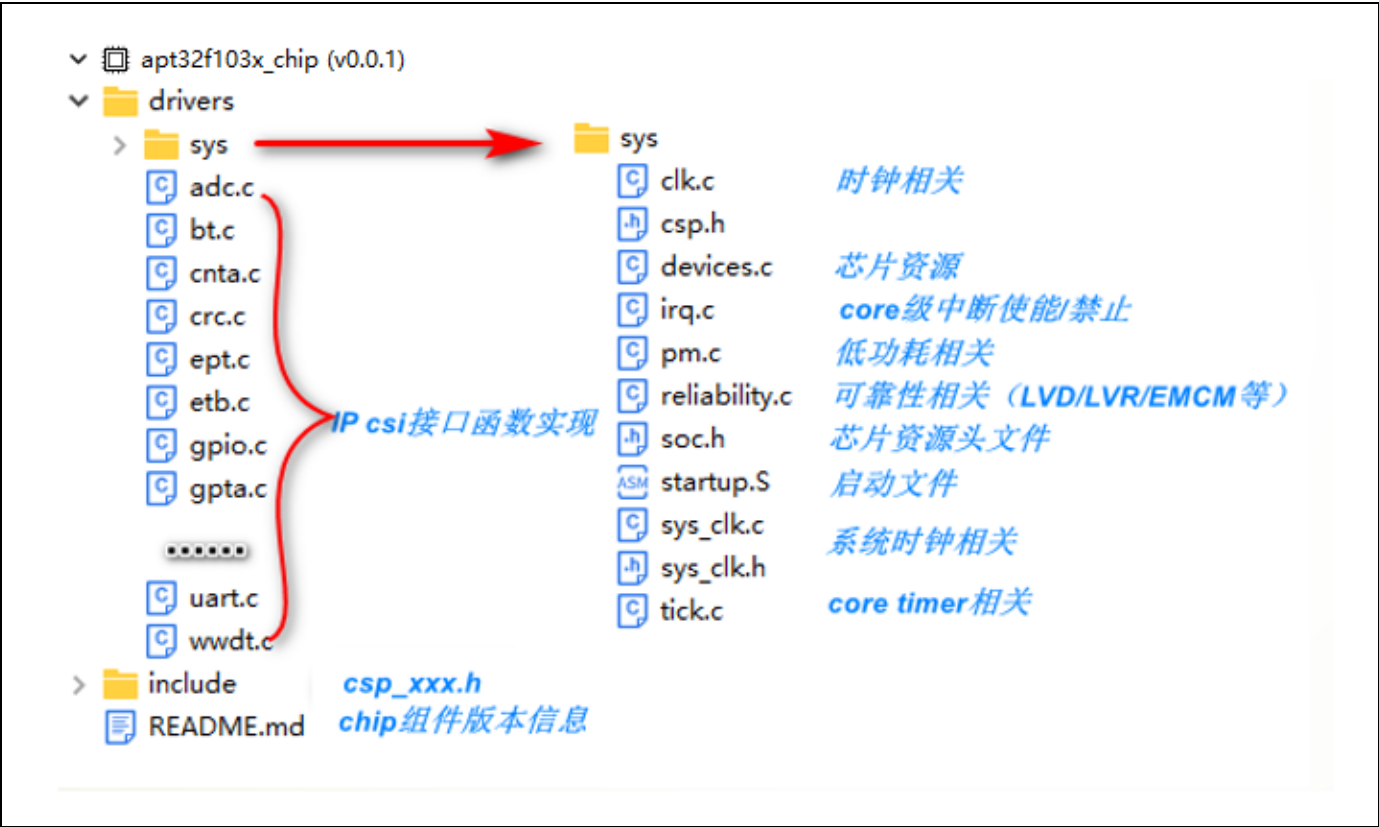


Figure 3 chip 组件说明

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（APT32F103x_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：

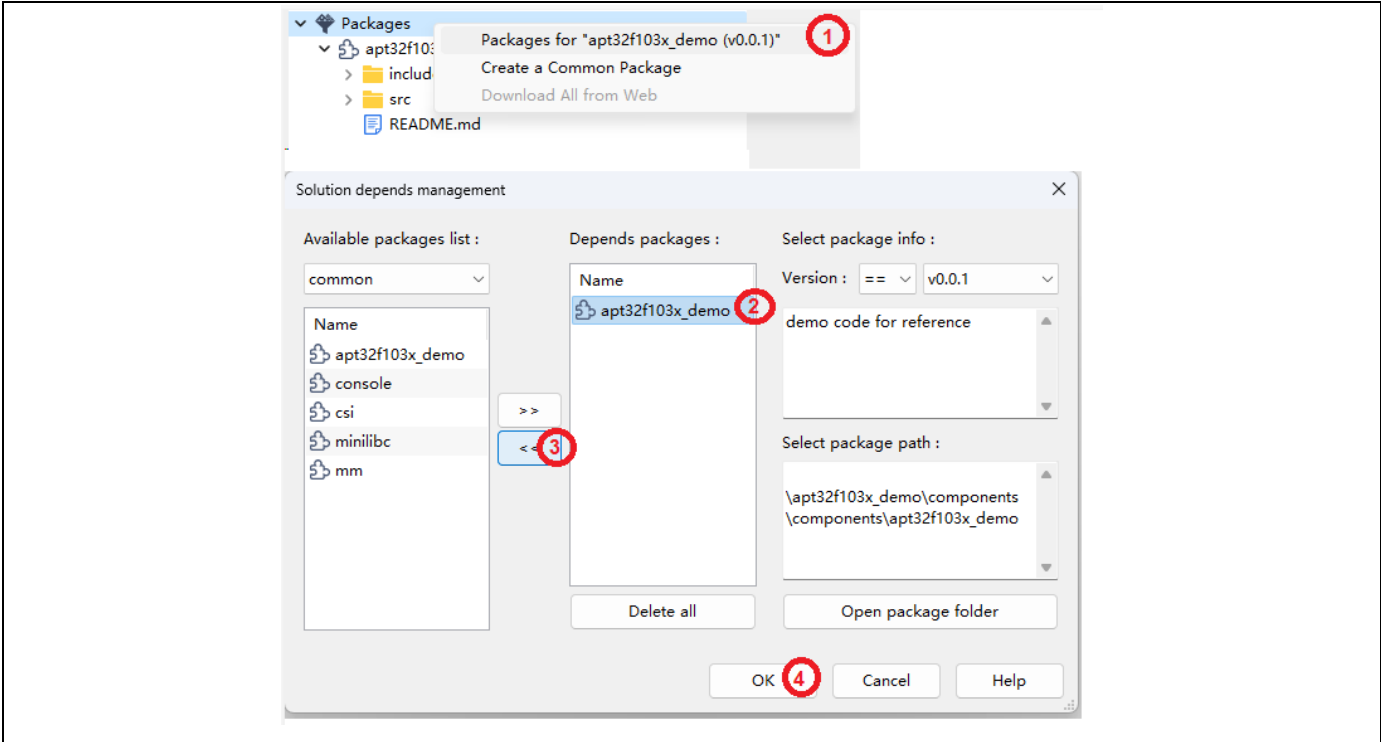


Figure 4 移除 demo 组件的方法

3. 代码移植

如[Board组件（APT32F103x_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_103x -> apt32f103x_evb-> include -> board_config.h

实际文件路径为： 工程目录__workspace_pack__\APT32F103x_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。

```
#define  CONSOLE_IDX                2
#define  CONSOLE_TXD                PA01
#define  CONSOLE_RXD                PA00
#define  CONSOLE_TXD_FUNC           PA01_UART2_TX
#define  CONSOLE_RXD_FUNC           PA00_UART2_RX

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

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_103x -> APT32F103x_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, EMOSC_VALUE, SCLK_DIV1, PCLK_DIV2, 5556000, 5556000}; //Note: Config external OSC
    // {SRC_IMOSC, IMOSC_5M_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};
    // {SRC_IMOSC, IMOSC_2M_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_103x -> APT32F103x_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。

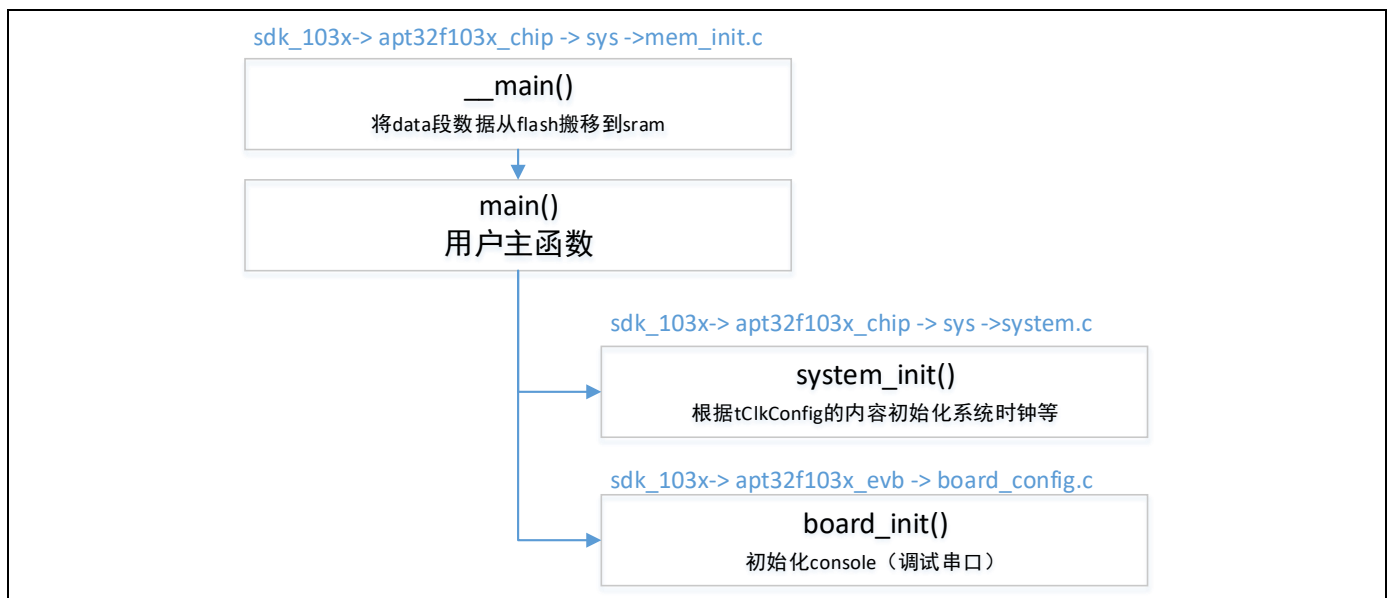


Figure 7 上电启动流程

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();
    csi_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。