

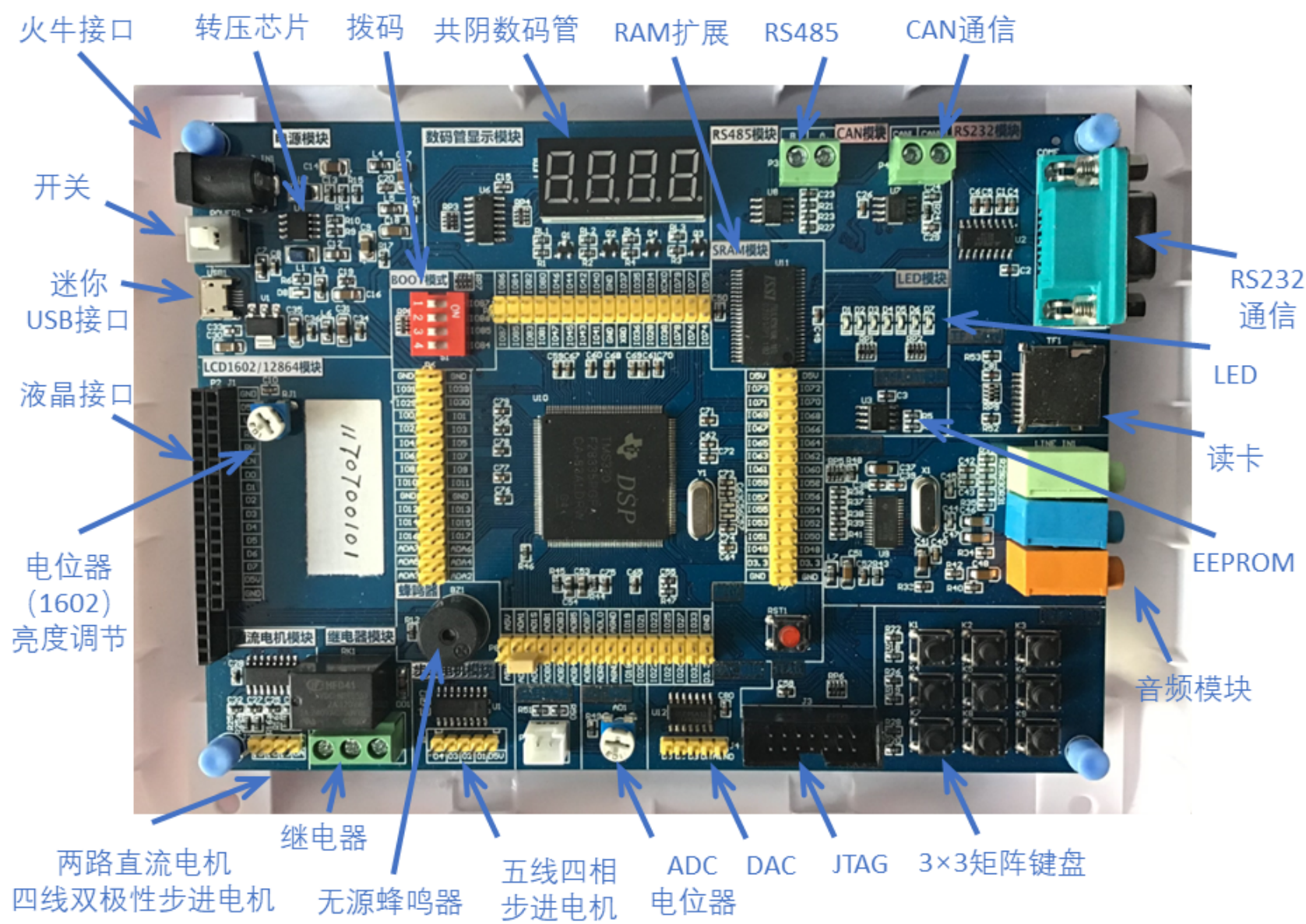
目录

- 1. 开发板功能及使用介绍
- 2. DSP TMS320F28335介绍
 - 2.1 命名
 - 2.2 F28335内核的主要特点
 - 2.3 F28335处理器的主要资源
 - 2.4 引脚分布
 - 2.5 F28335能做什么
- 3. 最小系统介绍
 - 3.1 最小系统组成
 - 3.2 F28335启动模式
- 4. CCS6工程导入
- 5. 存储器与寄存器
 - 5.1 存储器映射
 - 5.2 寄存器和寄存器映射
 - 5.3 如何访问寄存器内容
- 6. 创建工程模板
 - 6.1 基础文件获取
 - 6.2 新建工程
 - 6.3 添加文件

学习资源：普中科技DSP TMS320F28335、CCS6.0.0

[回到顶部](#)

1. 开发板功能及使用介绍



主要功能：电机控制，增强型PWM。

使用方法：安装CCS软件，安装DSP仿真器驱动，给开发板供电并通过仿真器连接电脑。

2. DSP TMS320F28335介绍

2.1 命名

TMS：前缀（合格设备），320：系列号（DSP Family），F：工艺（Flash EEPROM），28335：设备类型，PGF：封装形式（176引脚QFP封装），A：温度范围（40~85℃）。

2.2 F28335内核的主要特点

- 集成了DSP和微控制器的长处，F28335能够在一个周期内完成32×32位的乘法累加运算，或者两个16×16位的乘法累加运算；
- 快速的中断响应；
- 可在任何内存位置进行单周期读、修改、写操作；
- 可采用C/C++编程，效率非常高。

2.3 F28335处理器的主要资源

- 32位浮点DSP，主频是150MHz，方便电机控制，电力设备控制及工业控制等；
- 片上存储器：Flash——256K×16位，SRAM——34K×16位，BOOT ROM——8K×16位，OPT ROM——2K×16位；
- 片上外设：PWM——18路，HRPWM——6路，CAP——6路，QEP——2通道，ADC——2×8通道，12位，80ns转换时间，**0~3V量程**（采集高电压可以在外部分压），SCI——3通道，MCBPS——2通道，CAN——2通道，SPI——1通道，I2C——1通道，外部存储器扩展接口——XINTF，通用输入/输出IO——88个，看门狗电路。

与2812对比：

性 能	F28335	F2812
CPU	32 位定点+单精度浮点单元 FPU	32 位定点 CPU
系统频率	150 MHz	150 MHz
片内 FLASH	256K×16 位	128K×16 位
Boot ROM	8K×16 位	8K×16 位
OTP	1K×16 位	1K×16 位
32 位 CPU 定时器	3 个	3 个
SRAM	34K×16 位	18K×16 位
128 位密码保护	有	有
系统外部接口(XNTF)	有	有
通用 I/O 口(GPIO)	88 个(可配置 4 种工作模式)	56 个(可配置 2 种工作模式)
ADC	12 位、16 通道、12.5MSPS	12 位、16 通道、12.5MSPS

电机控制外设	ePWM (最多 18 路,包括 6 路 HRPWM) 6 路 32 位 eCAP 输入 (可配置为 PWM) 2 个 32 位 eQEP	EVA EVB
SPI	1 个	1 个
SCI	3 个	2 个
Ecan	2 个	1 个
I²C	1 个	无
McBSP/SPI	2 个	1 个
外部中断	8 个	3 个
PIE	支持 58 个外设中断	支持 45 个外设中断
DMA	6 个	无

2.4 引脚分布

- JTAG引脚：烧录程序，调试时烧录到RAM，最终烧录到Flash；
- 时钟引脚：X1、X2；
- 复位引脚：XRS；
- 电源引脚：VDD、VSS...；
- ADC信号引脚：16路；
- GPIO和外设信号引脚。

2.5 F28335能做什么

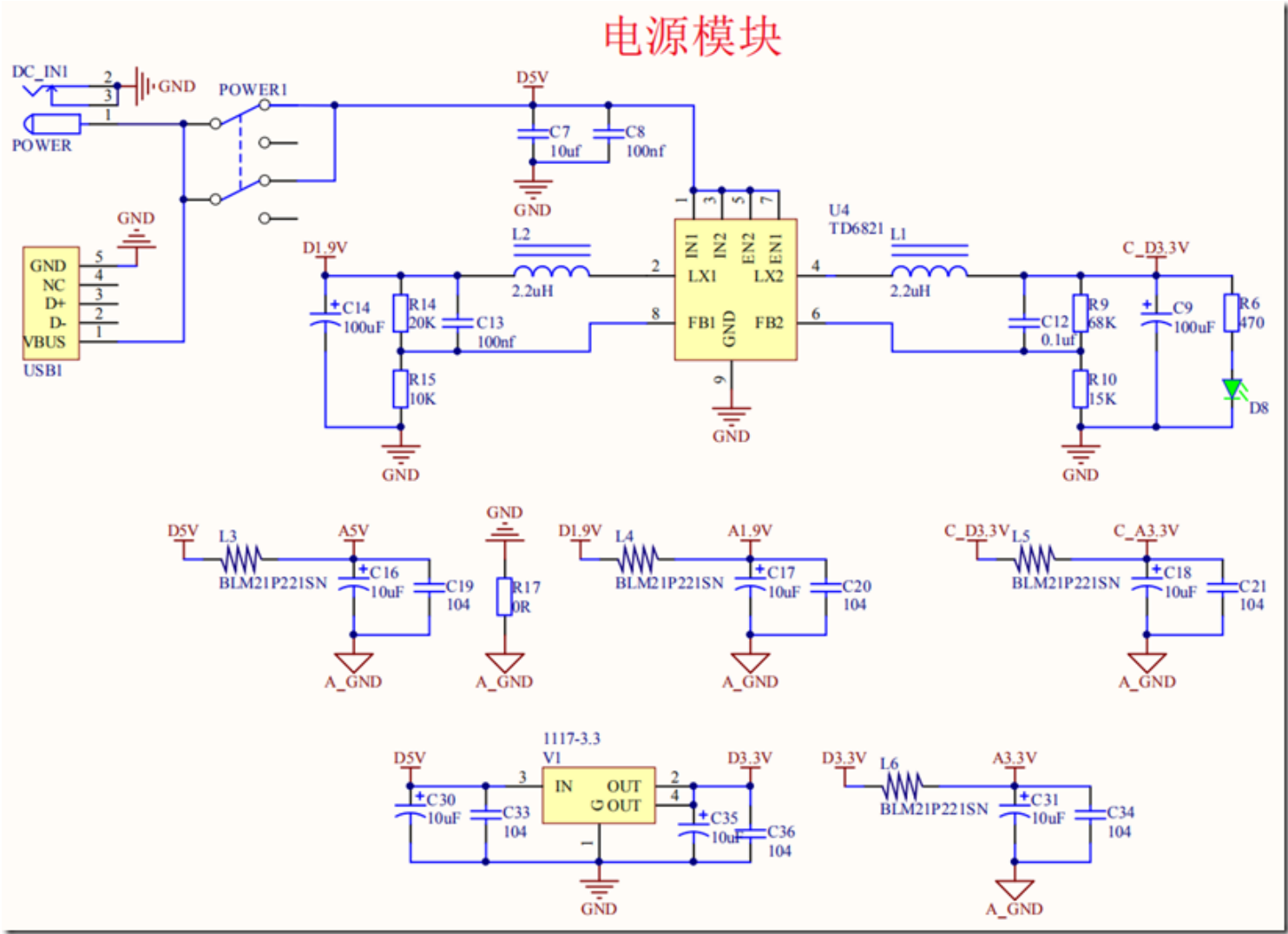
- 1. 自动化技术；
- 2. 智能测量仪器；
- 3. 家用、商用电子产品；
- 4. 通信；
- 5. 军事领域。

[回到顶部](#)

3. 最小系统介绍

3.1 最小系统组成

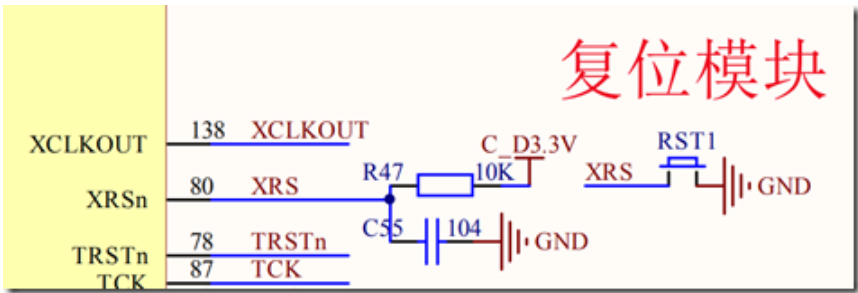
1. 电源电路



两个供电端→开关→转压。D8为指示灯。

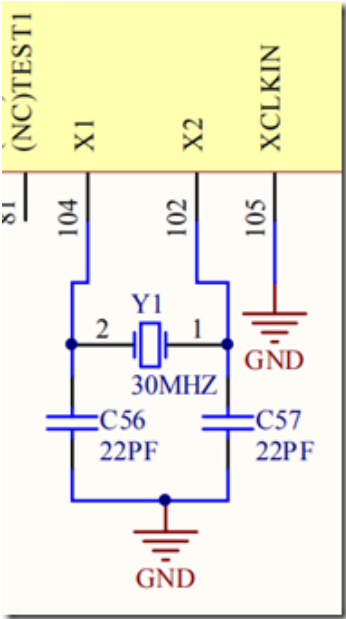
模拟、数字隔离。

2. 复位电路



RST1手动复位，XRS直接接地。电容充电电路：开机复位。

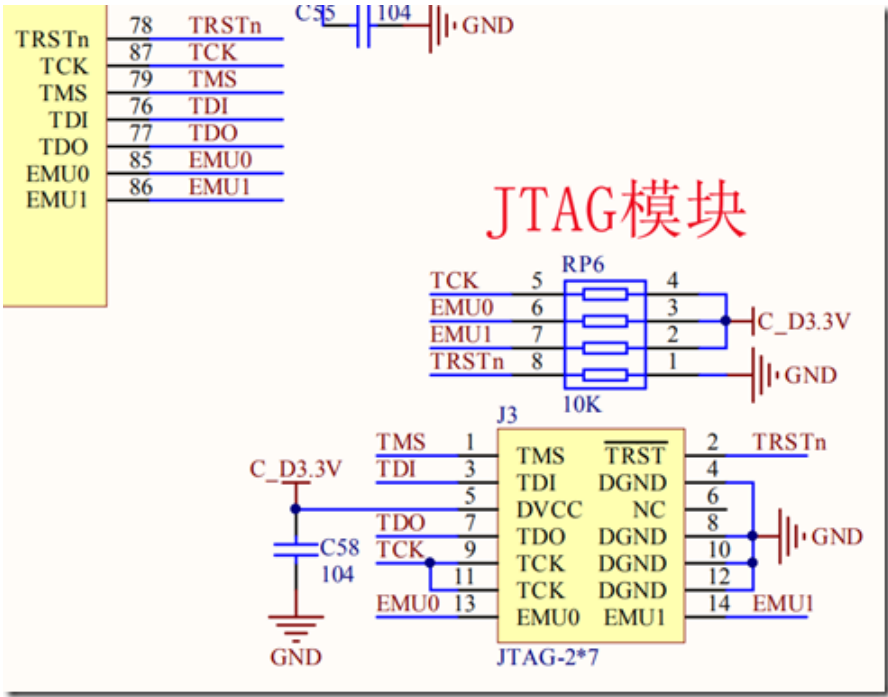
3. 晶振电路



提供系统时钟。

30MHz，X1和X2接晶振，XCLKIN接地。

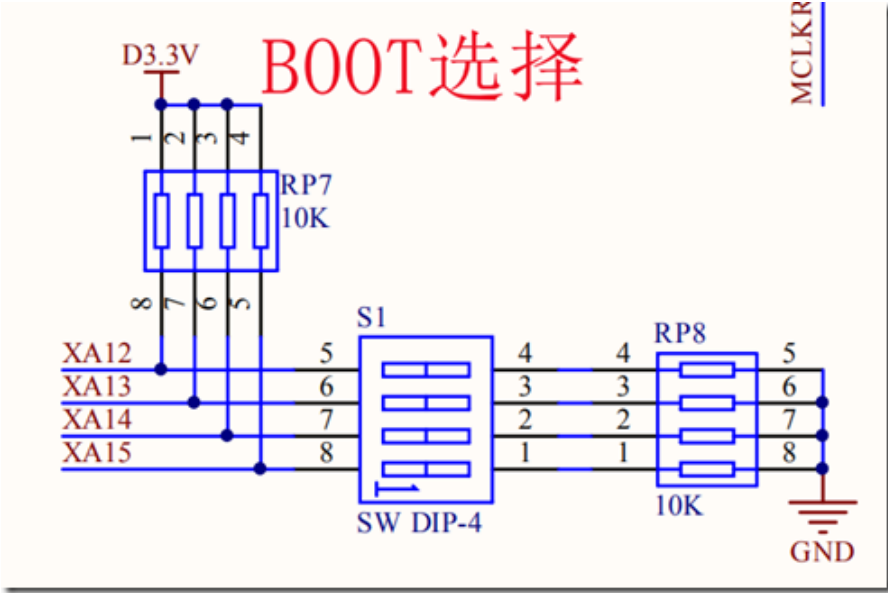
4. 下载电路



配合JTAG下载器。
C_D3.3V上拉，GND下拉。

3.2 F28335启动模式

TMS320F28335支持多种启动模式。



通过4个拨码选择16种启动方式：

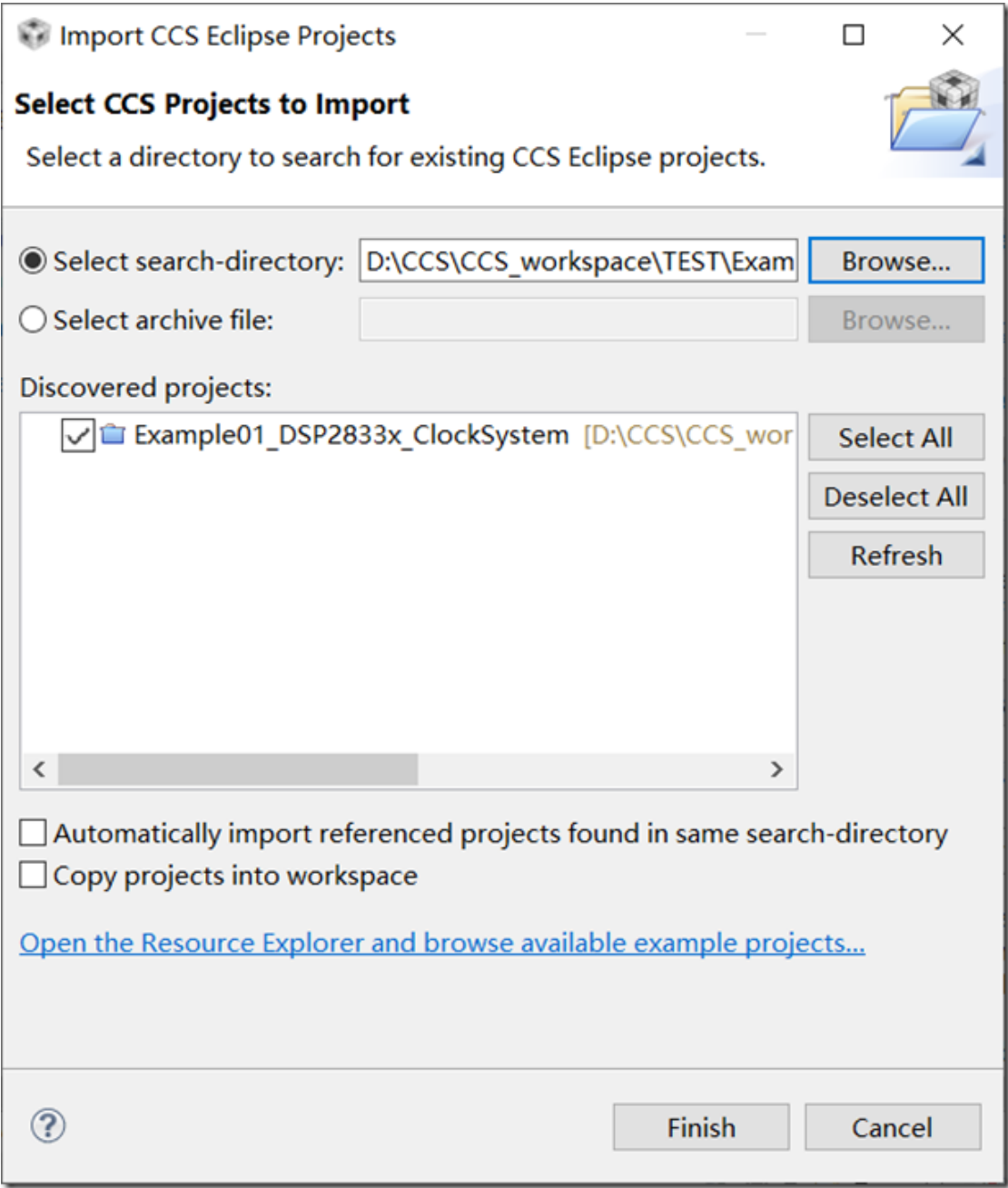
GPI087	GPI086	GPI085	GPI084	
XA15	XA14	XA13	XA12	
PU	PU	PU	PU	

1	1	1	1	Jump to Flash <- "boot to Flash"
1	1	1	0	SCI-A boot
1	1	0	1	SPI-A boot
1	1	0	0	I2C-A boot
1	0	1	1	eCAN-A boot
1	0	1	0	McBSP-A boot
1	0	0	1	Jump to XINTF x16
1	0	0	0	Jump to XINTF x32
0	1	1	1	Jump to OTP
0	1	1	0	Parallel GPIO I/O boot
0	1	0	1	Parallel XINTF boot
0	1	0	0	Jump to SARAM
0	0	1	1	Branch to check boot mode
0	0	1	0	Boot to flash, bypass ADC cal
0	0	0	1	Boot to SARAM, bypass ADC cal
0	0	0	0	Boot to SCI-A, bypass ADC cal
Boot_Table_End\$				

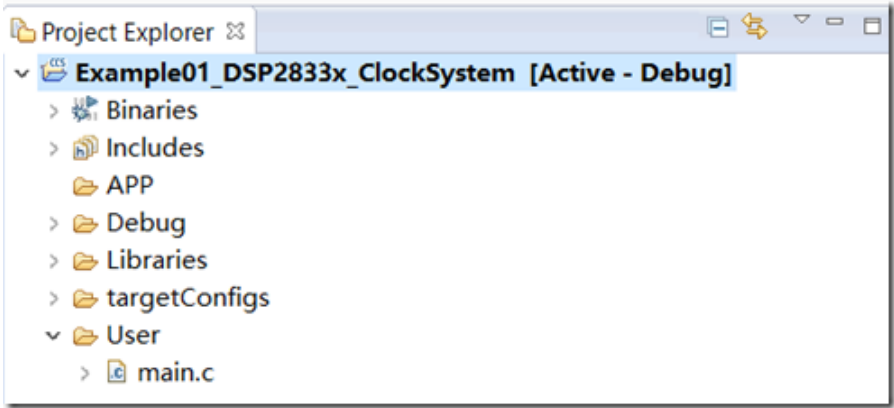
[回到顶部](#)

4. CCS6工程导入

Project→Import CCS Projects...



View→Project Explorer

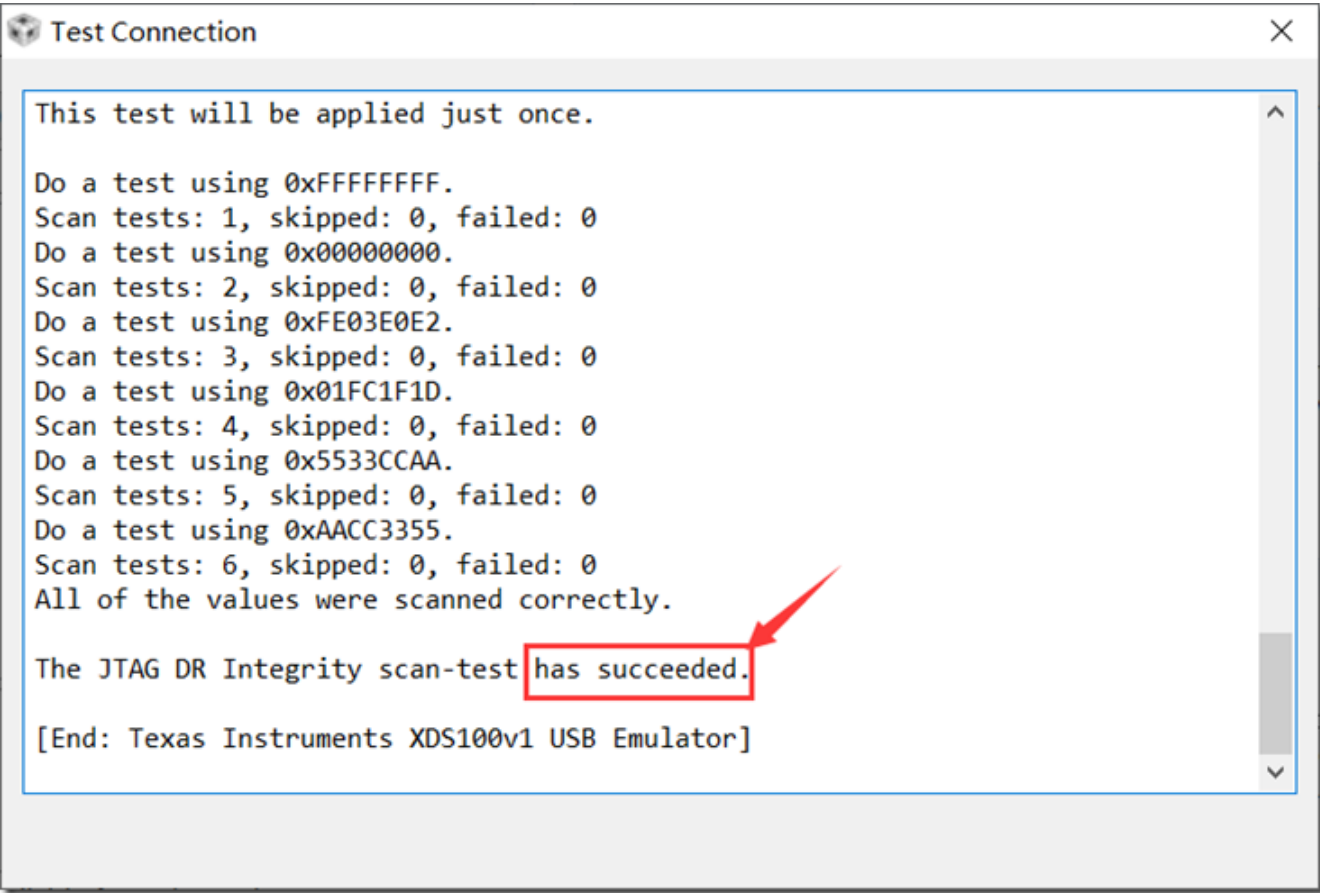


View→Target Configuration

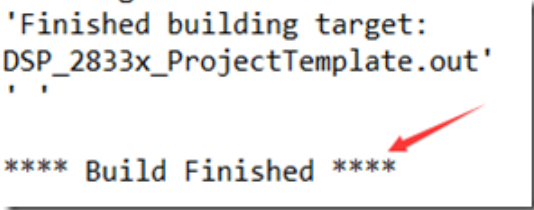


Test Connection: 测试是否连接成功。





编译、调试。



[回到顶部](#)

5. 存储器与寄存器

5.1 存储器映射

给存储器分配地址的过程就是存储器映射。

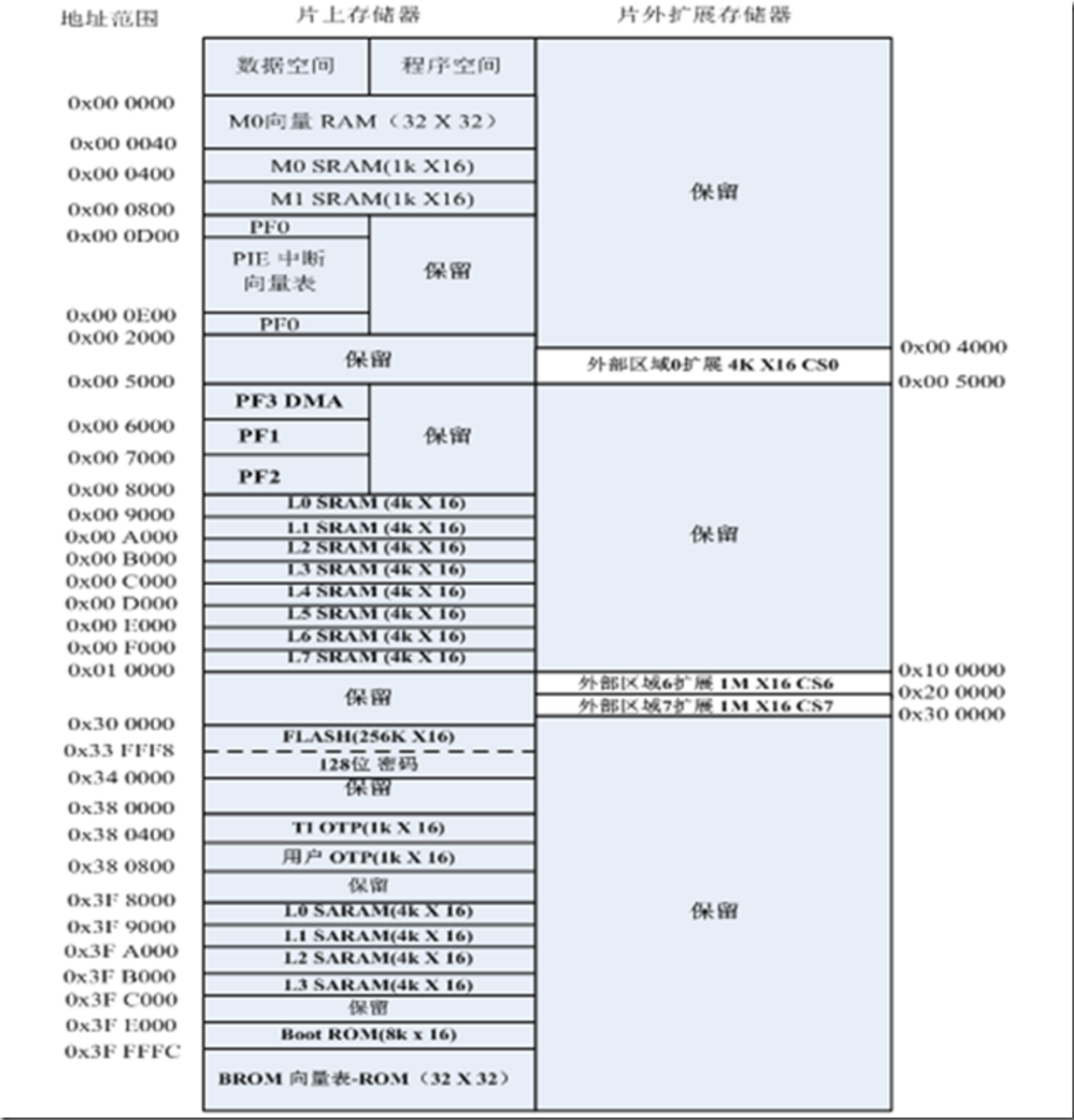
1. 存储器分配

F28335片上有256K×16位的FLASH，34K×16位的SARAM，8K×16位的BOOT ROM，2K×16位的OPT ROM，采用统一寻址方式（程序、数据和I/O统一寻址），从而提高了存储空间的利用率，方便程序的开发。除此之外，F28335还提供了外部并行扩展接口XINTF，可进一步外扩存储空间。

2. F28335存储器特点

F28335是采用多级流水线的增强的哈佛总线结构，能够并行访问程序和数据存储空间。

- (1) 片上SARAM
- (2) BOOT ROM
- (3) 片上FLASH和OTP



flash分区：（共256K×16）

地址范围	程序和数据空间
0x30 0000-0x30 7FFF	扇区 H (32K x 16)
0x30 8000-0x30 FFFF	扇区 G (32K x 16)
0x31 0000-0x31 7FFF	扇区 F (32K x 16)
0x31 8000-0x31 FFFF	扇区 E (32K x 16)
0x32 0000-0x32 7FFF	扇区 D (32K x 16)
0x32 8000-0x32 FFFF	扇区 C (32K x 16)
0x33 0000-0x33 7FFF	扇区 B (32K x 16)
0x33 8000-0x33 FF7F	扇区 A (32K x 16)
0x33 FF80-0x33 FFF5	当使用 代码安全模块时，编程至 0x0000
0x33 FFF6-0x33 FFF7	引导至闪存进入点 (程序分支指令所在的位置)
0x33 FFF8-0x33 FFFF	安全密码 (128 位) (不要设定为零)

3. 代码安全模块

通过一个128位的密码（相当于8个16位的字）来对安全区来进行加密或解密。这段密码保存在FLASH的最后8个字中(0X33FFF8-0X33FFFF)，也就是密码区中（PWL），通过密码匹配（PMF），可以解锁器件。

全1表示未加密，全0则不能解锁。

4. 外部存储器接口XINTF

5.2 寄存器和寄存器映射

通过#pragma预处理命令和DATA_SECTION将定义的寄存器指定到相应的存储单元内，然后即可通过C语言来操作这些寄存器。

比方说我们找到0x007010这个单元地址，那么可以通过查阅芯片数据手册了解到此单元是系统控制寄存器功能。因此为了更好区分此单元的功能和方便后续的程序开发，可以给这个单元取一个别名SysCtrlRegs，那么这个SysCtrlRegs就是寄存器，并且这个寄存器地址就是0x007010。这个过程就是寄存器映射。

5.3 如何访问寄存器内容

对于GPIO控制寄存器，通过查询数据手册可知其首地址是0x006F80，然后使用#pragma和DATA_SECTION将定义的寄存器与实际的存储单元对应起来。

```
#pragma DATA_SECTION(GpioCtrlRegs,"GpioCtrlRegsFile");

volatile struct GPIO_CTRL_REGS GpioCtrlRegs;
```


该定义可在DSP2833x_GlobalVariableDefs.c文件中查找到。

GpioCtrlRegsFile是SECTIONS内定义的，该定义可在DSP2833x_Headers_nonBIOS.cmd文件中查找到。

GPIO数据寄存器GPIO_DATA_REGS:

```

struct GPIO_DATA_REGS
{
    union  GPADAT_REG      GPADAT;      // GPIO Data Register (GPIO0 to 31)
    union  GPADAT_REG      GPASET;      // GPIO Data Set Register (GPIO0 to 31)
    union  GPADAT_REG      GPACLEAR;    // GPIO Data Clear Register (GPIO0 to 31)
    union  GPADAT_REG      GPATOGGLE;    // GPIO Data Toggle Register (GPIO0 to 31)
    union  GPBDAT_REG      GPBDAT;      // GPIO Data Register (GPIO32 to 63)
    union  GPBDAT_REG      GPBSET;      // GPIO Data Set Register (GPIO32 to 63)
    union  GPBDAT_REG      GPBCLEAR;    // GPIO Data Clear Register (GPIO32 to 63)
    union  GPBDAT_REG      GPBTOGGLE;    // GPIO Data Toggle Register (GPIO32 to 63)
    union  GPCDAT_REG      GPCDAT;      // GPIO Data Register (GPIO64 to 95)
    union  GPCDAT_REG      GPCSET;      // GPIO Data Set Register (GPIO64 to 95)
    union  GPCDAT_REG      GPCCLEAR;    // GPIO Data Clear Register (GPIO64 to 95)
    union  GPCDAT_REG      GPCTOGGLE;    // GPIO Data Toggle Register (GPIO64 to 95)
    Uint16                                rsvdl[8];
};

union GPCDAT_REG {
    Uint32                all;
    struct GPCDAT_BITS    bit;
};
```

all可对32位整体操作，一般使用bit进行位操作。

```

struct GPCDAT_BITS {
    // bits    description
    Uint16 GPIO64:1;    // 0      GPIO64
    Uint16 GPIO65:1;    // 1      GPIO65
    Uint16 GPIO66:1;    // 2      GPIO66
    Uint16 GPIO67:1;    // 3      GPIO67
    Uint16 GPIO68:1;    // 4      GPIO68
    ...
    Uint16 GPIO87:1;    // 23     GPIO87
    Uint16 rsvdl:8;     // 31:24  reserved
};
```

所以要让GPIO68输出一个低电平可使用C语言调用结构体内成员，如下：

```
GpioDataRegs.GPCCLEAR.bit.GPIO68=1; //设置GPIO输出低电平信号
```

[回到顶部](#)

6. 创建工程模板

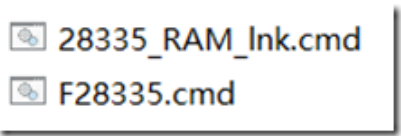
6.1 基础文件获取

TI头文件下载地址：<http://www.ti.com/tool/CONTROLSUITE?keyMatch=controlsuite&tisearch=Search-EN>

目录：\device_support\f2833x\v141

名称	修改日期	类型
doc	2019/8/29 16:19	文件夹
DSP2823x_examples_ccsv5	2019/8/29 16:19	文件夹
DSP2833x_common	2019/8/29 16:19	文件夹
DSP2833x_examples_ccsv5	2019/8/29 16:19	文件夹
DSP2833x_headers	2019/8/29 16:19	文件夹



\DSP2833x_common\cmd:



28335_RAM_Ink：仿真调试，F28335：烧录到Flash。

\DSP2833x_common\include：头文件；\DSP2833x_common\lib：运算库文件；\DSP2833x_common\source：源文件。

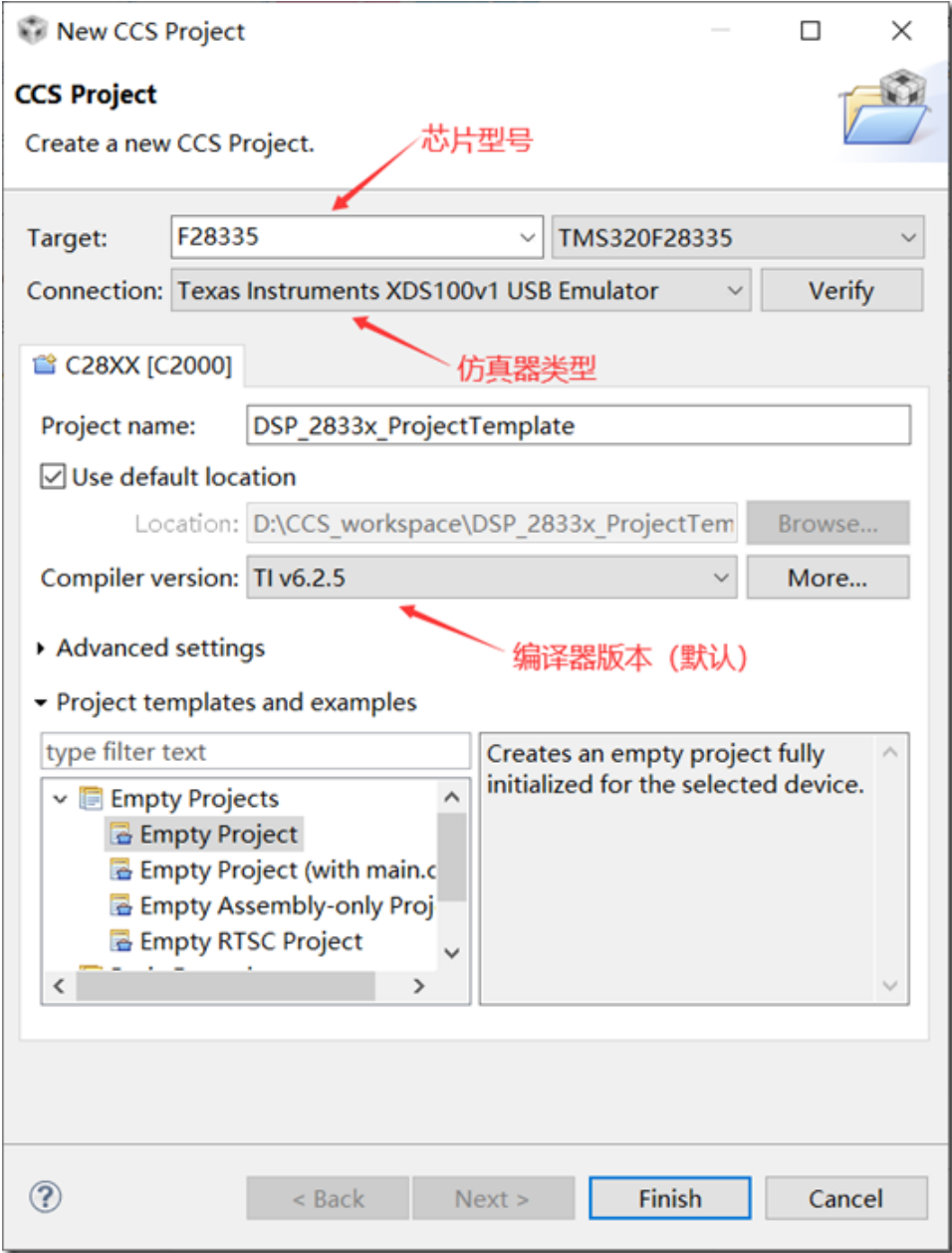
\DSP2833x_headers\cmd：是否使用操作系统。

-  DSP2833x_Headers_BIOS.cmd
-  DSP2833x_Headers_nonBIOS.cmd

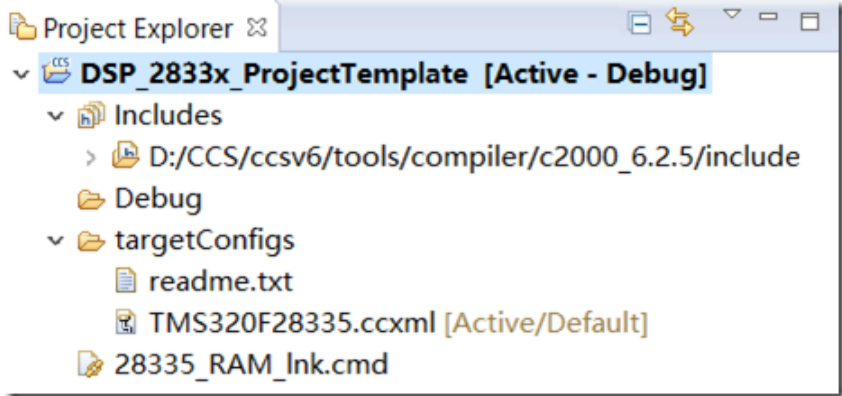
\\DSP2833x_headers\\include：头文件；\\DSP2833x_headers\\source：全局定义。

6.2 新建工程

Project→New CCS Project...



自动生成的文件：




一个完整的基础工程的构成文件：

1. 首先需要仿真调试或者flash烧写所需的.cmd文件和DSP的BIOS或nonBIOS .cmd文件；
2. 其次需要我们使用的芯片的.ccxml目标配置文件；
3. 芯片内核及外设.c源文件，比如DSP2833x_Gpio.c、 DSP2833x_PieCtrl.c等；
4. 芯片内核及外设.h头文件，比如DSP2833x_Gpio.h、 DSP2833x_PieCtrl.h等；
5. DSP .lib库文件，常用的如IQmath.lib等。

6.3 添加文件

右击User→New→Source File→main.c



```
#include "DSP2833x_Device.h"    // DSP2833x Headerfile Include File
#include "DSP2833x_Examples.h"  // DSP2833x Examples Include File

void main()
{

}
```



右击工程→Properties→Include Options: 添加对应头文件路径。

工程结构:

```
| .ccsproject
| .cproject
| .project
| 28335_RAM_Ink.cmd
| F28335.cmd
|
├─.launches
| DSP_2833x_ProjectTemplate.launch
|
├─.settings
| org.eclipse.cdt.codan.core.prefs
| org.eclipse.cdt.debug.core.prefs
| org.eclipse.core.resources.prefs
|
├─APP
├─Debug
| | ...
|
├─include
| ...
|
├─Libraries
| DSP2833x_ADC_cal.asm
| DSP2833x_CodeStartBranch.asm
| DSP2833x_DefaultIsr.c
| DSP2833x_GlobalVariableDefs.c
| DSP2833x_Gpio.c
| DSP2833x-Headers_nonBIOS.cmd
| DSP2833x_PieCtrl.c
| DSP2833x_PieVect.c
| DSP2833x_SysCtrl.c
| DSP2833x_usDelay.asm
| IQmath.lib
|
├─targetConfigs
| readme.txt
| TMS320F28335.ccxml
|
└─User
main.c
```