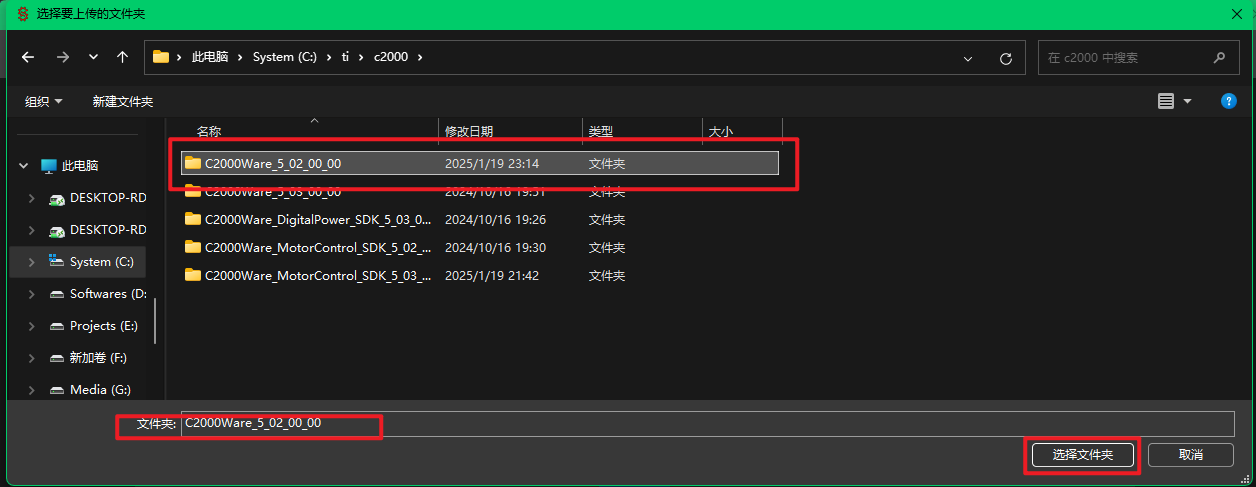
x使用SysConfig建立电机控制软件的要点：

零. 新建工程

截图里有图片

描述已自动生成

选择右边导入软件包，选择安装C2000ware的目录。

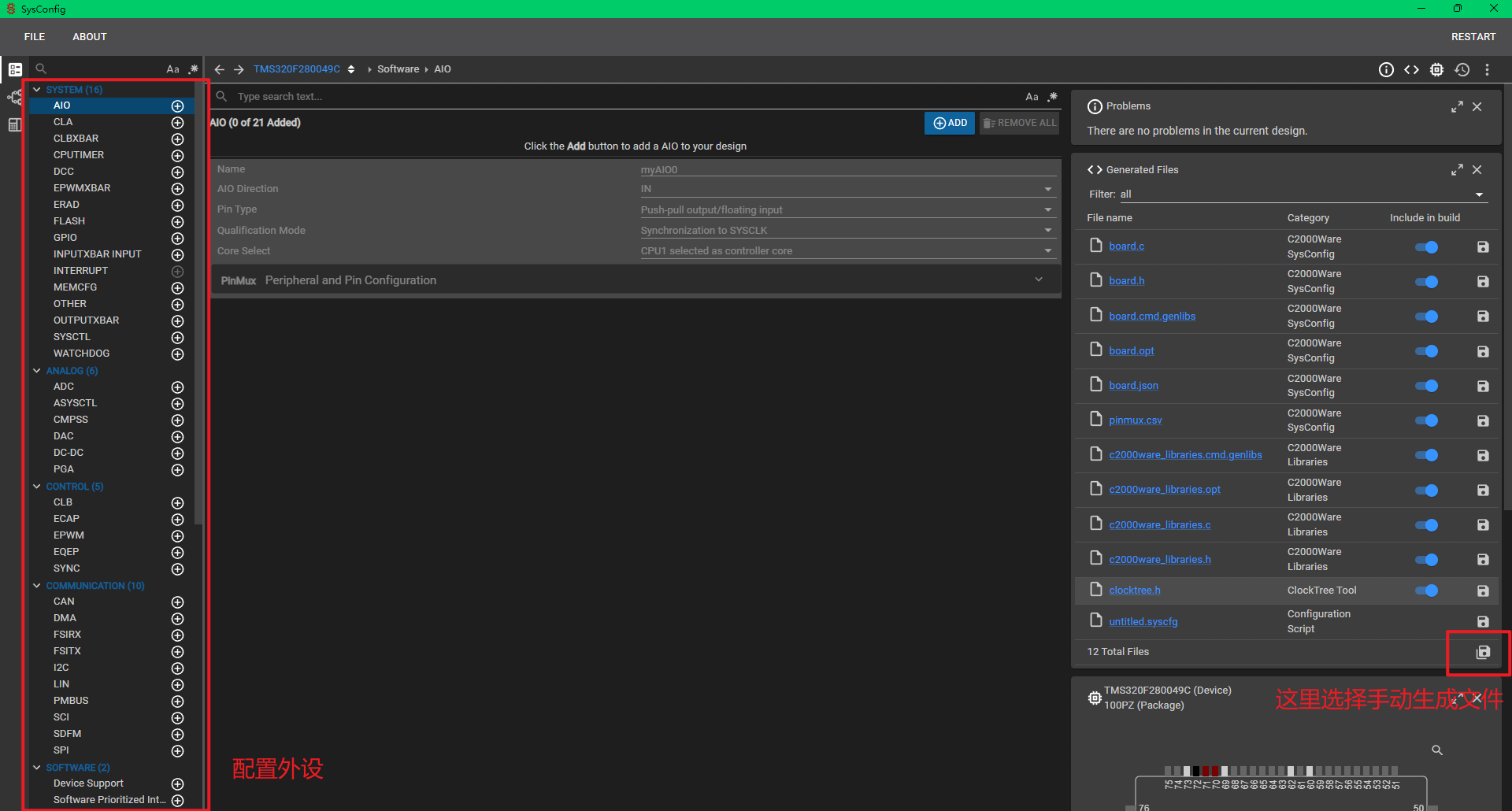


之后选择目标芯片

图形用户界面, 文本, 应用程序

描述已自动生成

就可以进入下图所示的配置界面：

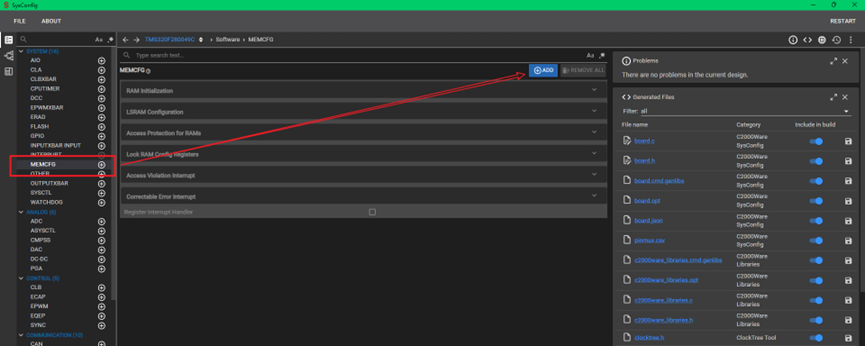


一. 基本的设备支持：

1. 作为一个基于Syscfg的工程，推荐打开以下几个选项，之后再进行需要的外设的配置。

|  |  |
| --- | --- |
| 必要基础设置 | 功能概述 |
| System - MEMCFG | 进行内存区功能设置，使用CLA时一定需要用到，方便DSP和CLA通信。  如果不使用CLA可以只添加，并不修改。 |
| System - SYSCTL | 用于设置核心、外设、非屏蔽中断等的映射。 |
| Software – Device Support | 将会依据System – SYSCTL的设置生成设备支持，正确的引入头文件和基础文件。 |
| Linker Command File Config | 如果需要在连接命令文件中附加连接命令，可以使用这个选项进行配置，比如需要配置CLA的工作内存。  如果不使用CLA可以只添加，不做修改。 |
| Analog - ASYSCTL | 启用模拟外设控制器，启用后默认选择模拟参考源为外部参考源。 |

添加的方法，参考下图：



对于控制器的设计，下面这些内容通常也是必要的项目:

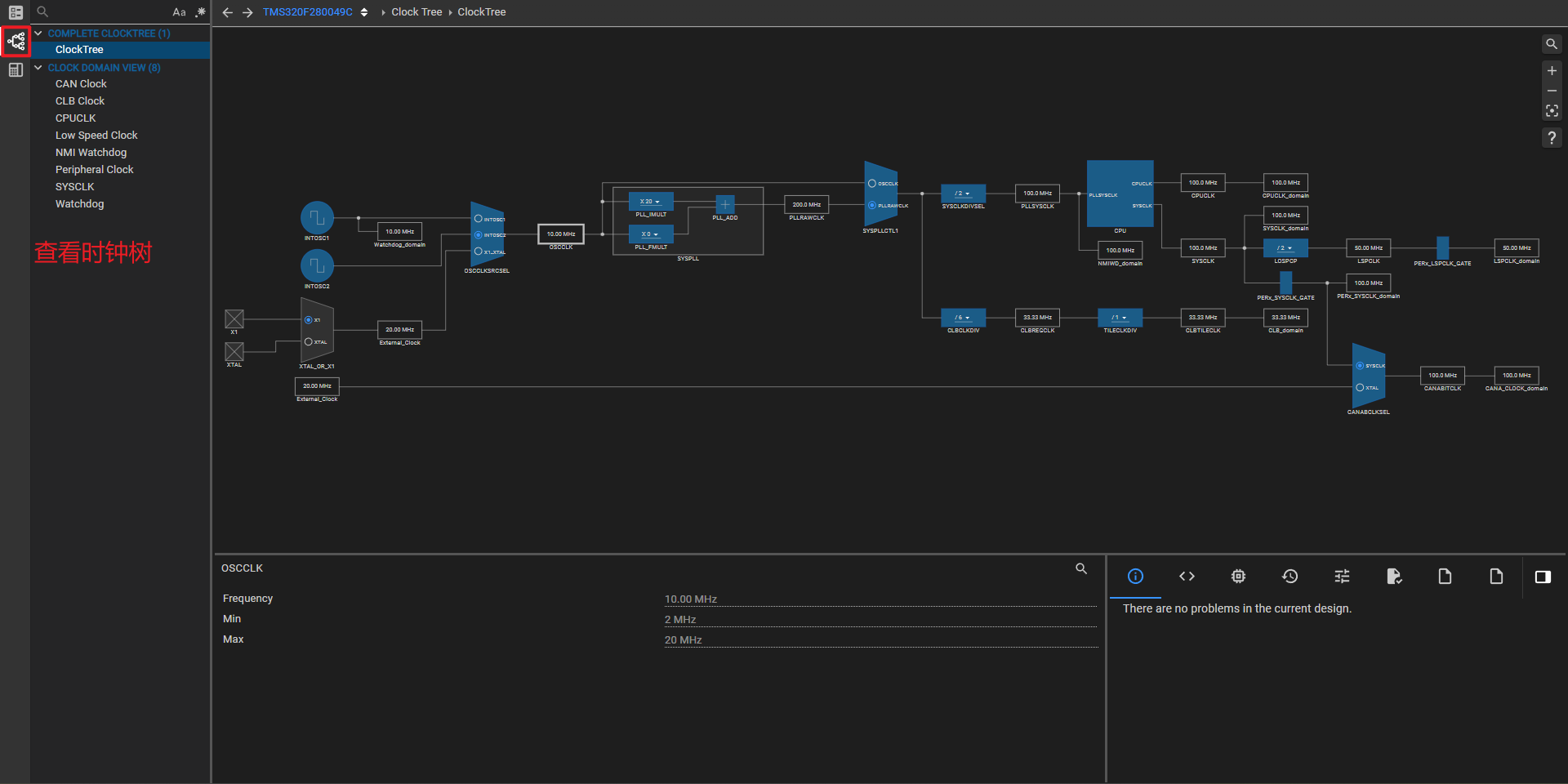
可以在C2000WARE LIBRARY选下卡下根据需要添加：

|  |  |
| --- | --- |
| C2000Ware 库 | 功能概述 |
| C2000ware Library - TCM | 瞬态捕捉模块 |
| C2000ware Library – IQmath | 数学库 |
| C2000ware Library – REFGEN | 参考发生器 |

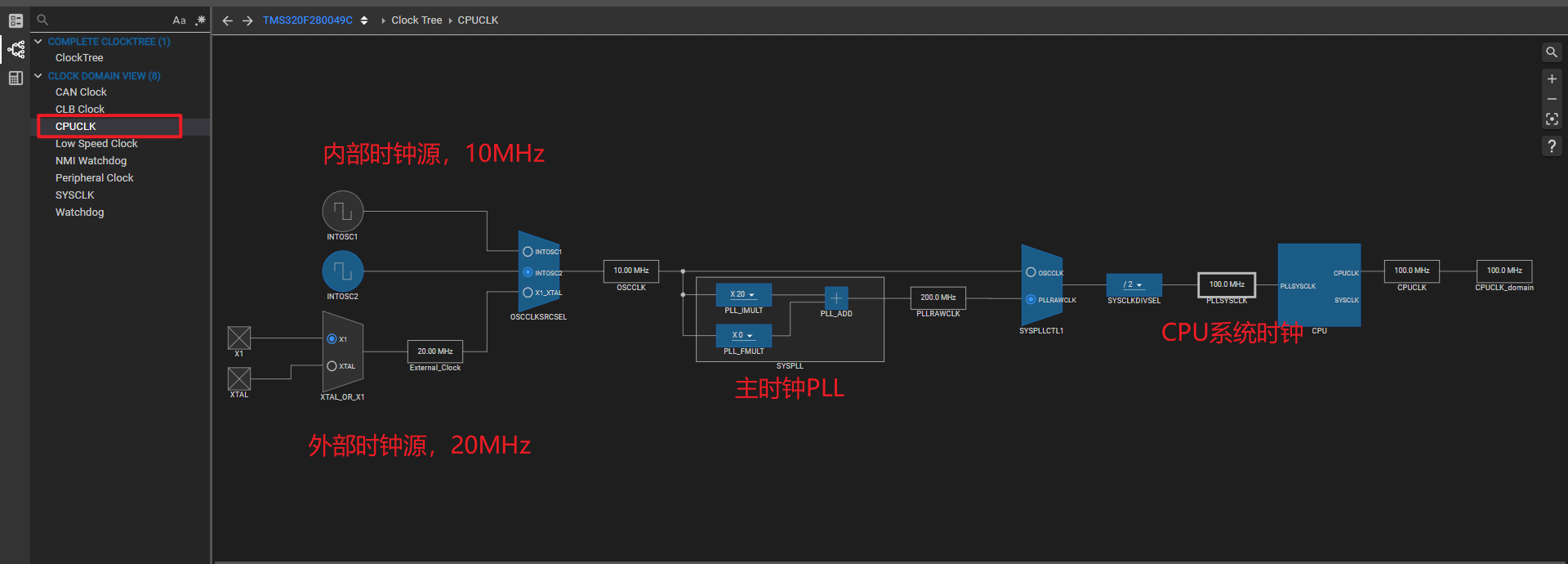
特别注意，使用SysCfg配置的工程并不会主动启动全局中断，需要用户手动选择启动全局中断，否则中断响应函数不会执行！

二. 设置系统时钟

接下来涉及外设的时钟设置，所以建议先完成时钟设置，再进行外设的功能设置。



选择CPUCLK，可以查看CPU的时钟设置。默认情况下，内部时钟源是10MHz，外部时钟源是20MHz。以280049C为例，这款芯片的最大主频为100MHz，默认情况下，芯片将工作在最高主频。如果需要使用外部时钟源输入则需要进行相应调整。



选择外设低速时钟LSP，可以进行外设时钟设置，默认情况下低速外设工作频率是50MHz，高速外设工作频率是100MHz。

电脑萤幕画面

描述已自动生成

电脑萤幕的截图

描述已自动生成

此外还涉及CLB模块、CAN模块、NMI WD模块、WD模块的时钟按照需要进行设计。

三. ADC模拟外设设置参考

下面可以根据需要启动ADC外设，不需要配置AIO功能。

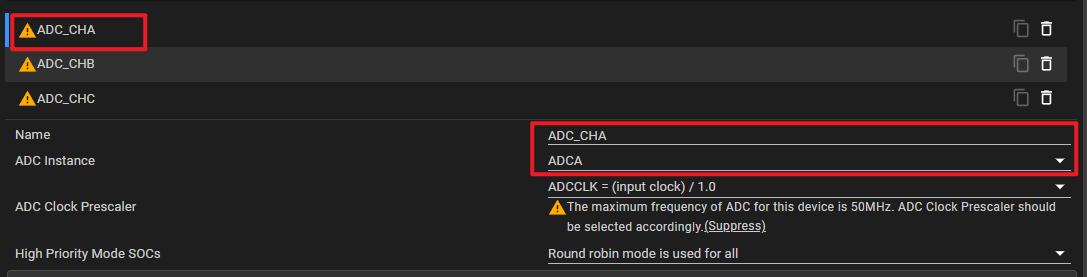
相关资料可以参考：

https://dev.ti.com/tirex/explore/node?node=A\_\_ASJNfY4V8S0ZYiiDJ-kGaQ\_\_C28X-ACADEMY-CN\_\_hMpsP7d\_\_LATEST

电脑萤幕的截图

描述已自动生成

通常情况下，我们会同时启用全部ADC外设，以提升ADC测量过程的实时性。启用ADC，可以先为每一个ADC外设分配一个物理ADC。



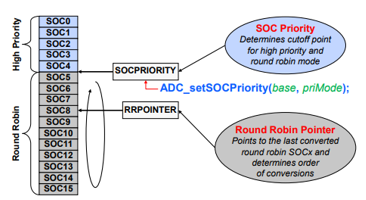
由于TMS320F280049的ADC外设最大速度为50 MHz，所以需要进行合理的ADC时钟分频设计。

图形用户界面, 文本, 网站

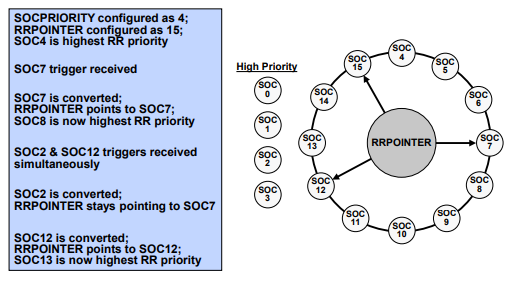
描述已自动生成

在控制器设计领域，需要将控制器相关的ADC通道设置为高优先级。在TI的ADC外设中，ADC有三个有限级，分别为轮询（正常模式，Round Mode，默认值），高优先级（插入模式，Hi-pri，需要指定连续的数个SOC），轮询突发模式（突发模式，Burst，突发访问非高优先级ADC通道）。

高优先级可以分配给一组SOC，形成高优先级序列。高优先级触发信号到达后，会中断轮询，然后立即开始执行高优先级序列。在高优先级序列中数字小的先执行（转换）。在高优先级序列执行完后将会退回轮询模式。



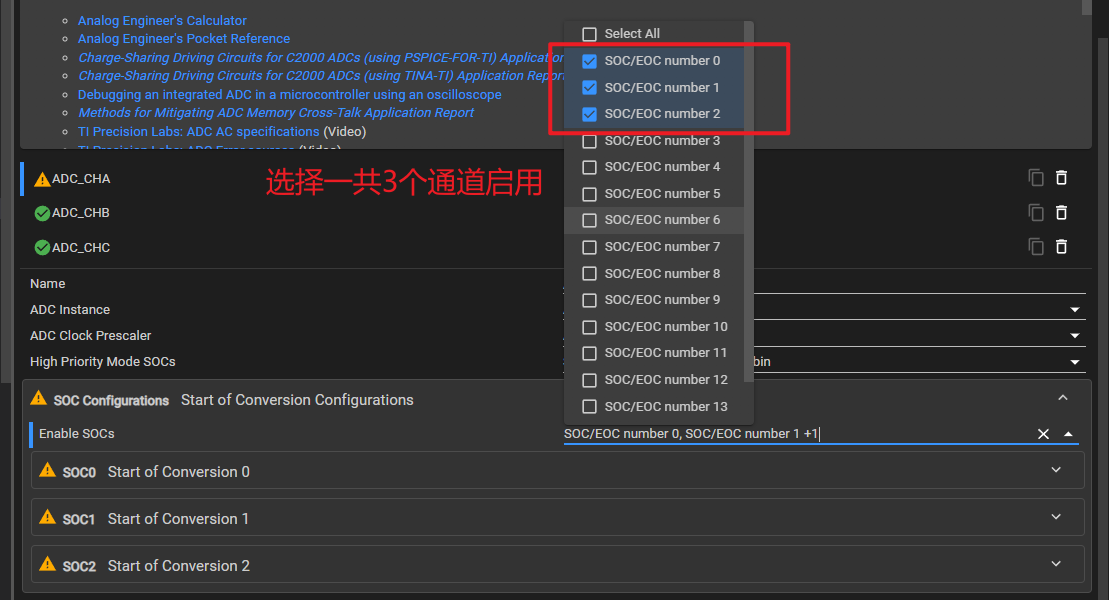
在ADS配置中，需要指定SOC的使能数量，和高优先级的SOC数量，高优先级SOC数量决定有多少个ADC通道是高优先级，在所有使能的SOC通道中除了高优先级的SOC构成Round Robin序列。参考下图：

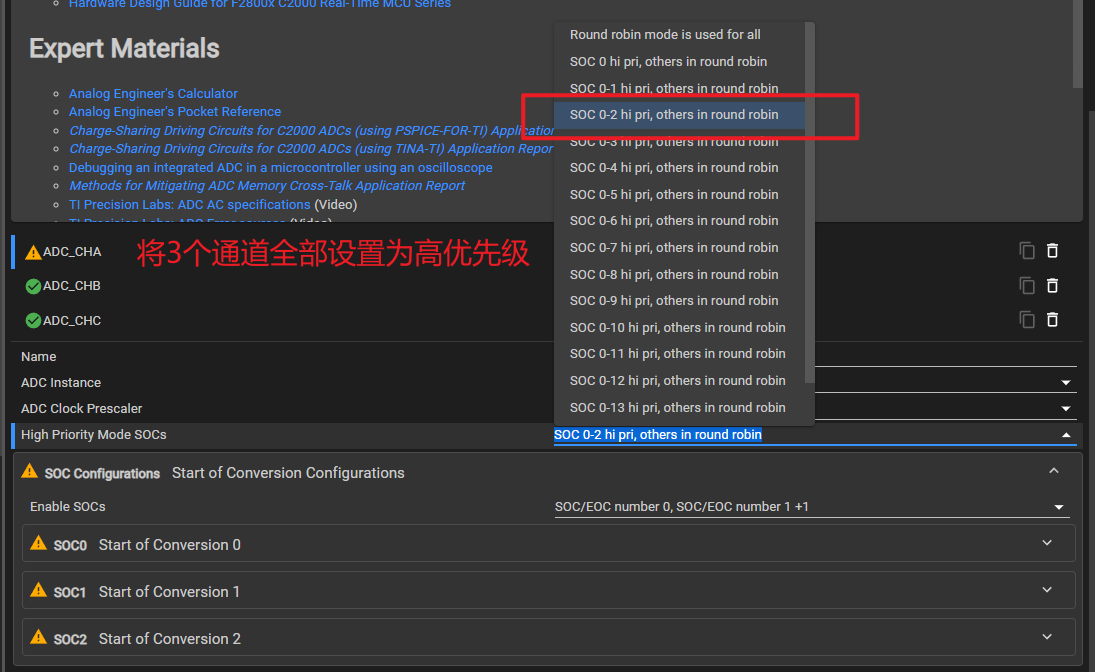


在电机控制实际场景中，考虑共有UDC, IA, IB, IC, UA, UB, UC七个信号需要实时采集并参与控制，有一个温度传感信号Tc（控制器温度传感）作为普通优先级信号。

那么此时可以将共计个信号划分为3组3（ChA，IC，VC，Udc）+3（ChB，IA，VA，Tc）+2（ChC，IB，UB）。

对于A通道，设置3个通道启用，3个通道全部为高优先级。





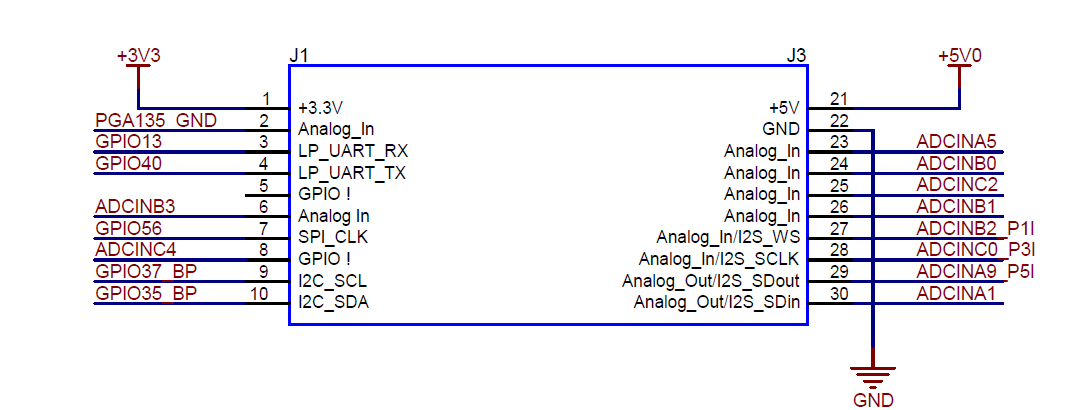
下面设置三个通道的转换目标：

根据280049C的原理图和3phGanInv的原理图，如下图所示，得到一个ADC通道的对应表。

图示

中度可信度描述已自动生成

3phGanInv板的模拟接口通道

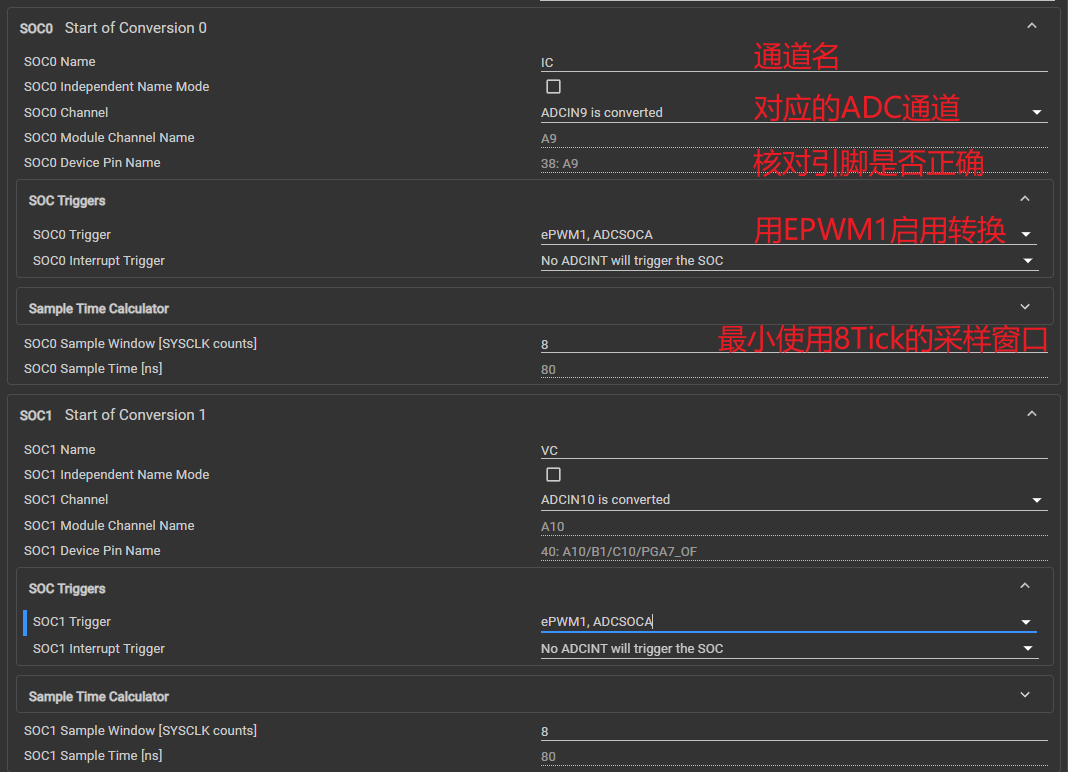


280049C板的模拟通道接口

|  |  |  |
| --- | --- | --- |
| 驱动板信号 | 核心板信号 | ADC通道名 |
| VDC | ADC IN A5 | A5 |
| VA | ADC IN B0 | B0 |
| VB | ADC IN C2 | C2 |
| VC | ADC IN B1 | A10 |
| IA | ADC IN B2 | B2 |
| IB | ADC IN C0 | C0 |
| IC | ADC IN A9 | A9 |
| Tc | ADC IN B3 | B3 |

注意，此处根据同步性原则，将VC的引脚变更为A10进行配置，此时IABC三个通道同时进行采样，UABC三个通道同时进行采样。接下来约定首先进行三个通道的电流采样，接下来进行三个通道的电压采样，之后进行母线电压的采样。

接下来分别对ADC A的三个转换通道进行设置，下面展示了电流通道和电压通道的设置。需要关注对应的通道、触发事件、采样时间。



下面展示母线电压的采样设置：

截图里有图片

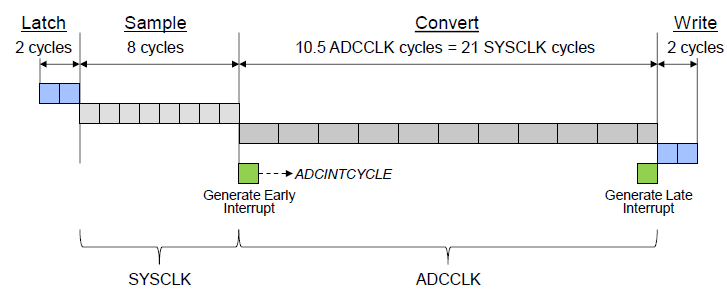
描述已自动生成

将以上三个通道设置好后，将不会有任何的错误和警告，如下图所示：

图形用户界面, 文本

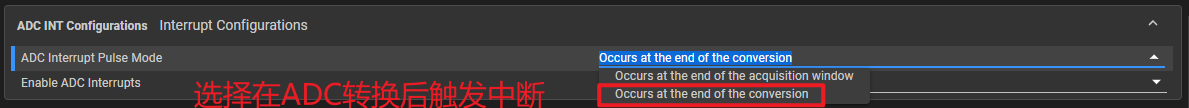
描述已自动生成

其中，ADC采样时间的计算原理如下：

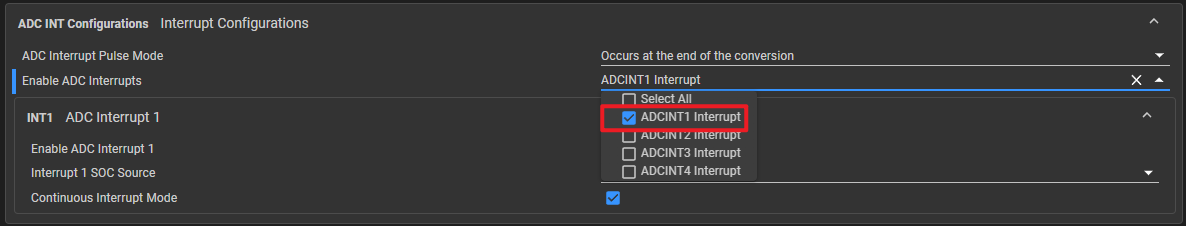


通常情况下，F280049器件的采样时间设置为8tick，转换时间为10.5tick，总计耗时290ns，相当于采样速率3.45MHz。

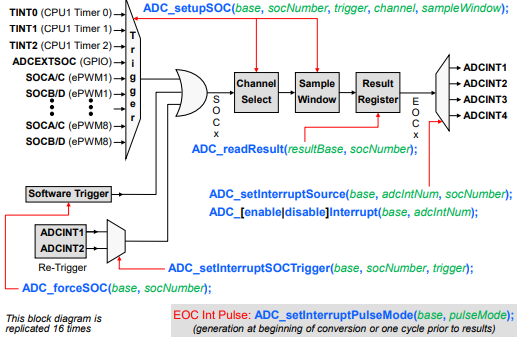
接下来配置了三个转换通道后，需要设置中断源，如下：



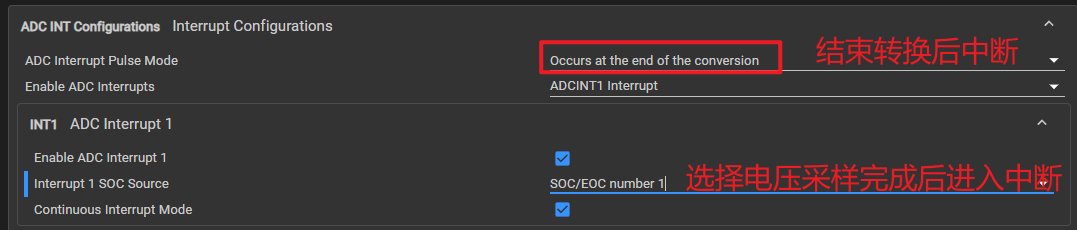
接下来使能中断通道，通常情况下，只需要主中断，以INT1作为中断源。



中断通道的选择可以依据下图，选择合适的EOC作为中断触发源。通常情况下由于电流和电压是控制的关键变量，所以可以用EOC1作为中断触发源，也即在完成电流（EOC0）和电压（EOC1）时触发中断。



具体配置如下：



其中Continuous Interrupt Mode是指当中断重复发生时，如果中断标志位没有被清除，仍然重新触发并进入中断。

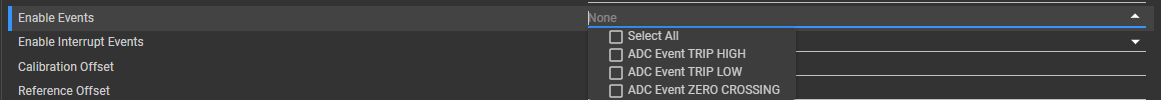
在进入转换完成中断后ADC的结果将保存在排序器的结果寄存器中。

在对于资源紧缺的计算场景下，可以使用后处理模块（PPB）来实现偏移量矫正、跳闸信号产生等动作。

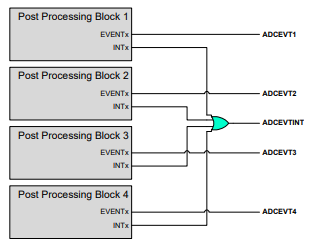
图形用户界面, 文本, 应用程序

描述已自动生成

在选定PPB对应的SOC通道后，可以选择PPB模块的使能的功能，如下图所示。



在选择了TZ事件和中断后，可以触发过限的TZ中断，并且数个事件可以组合起来构成了一个合并的安全事件中断。

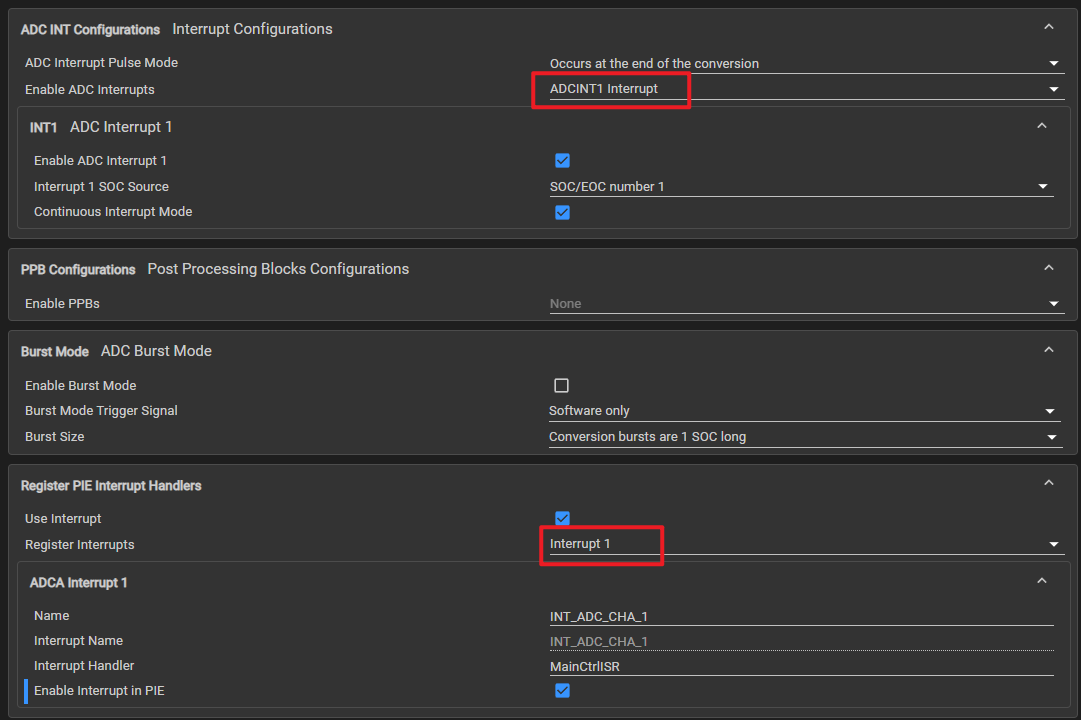


在中断选项中启用中断1，并在中断设置中启用PIE。

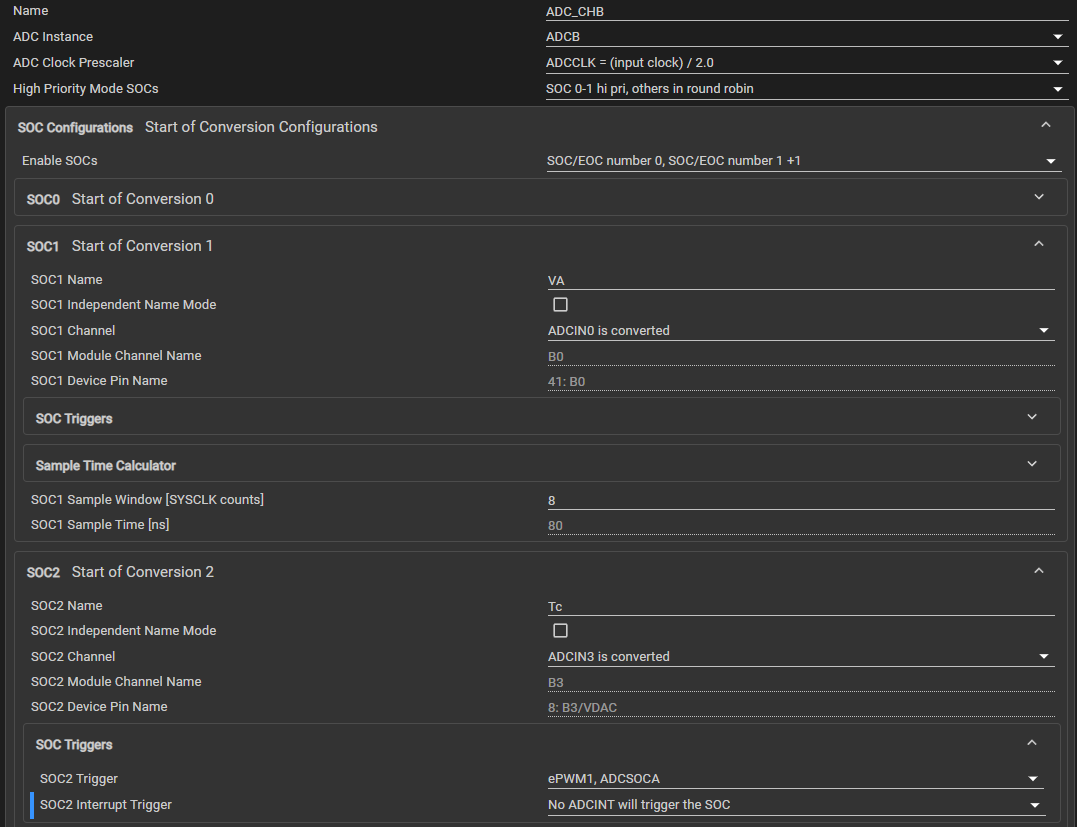
图形用户界面, 文本, 应用程序

描述已自动生成

此处中断1应当和ADC中断触发源设置中的中断编号匹配。

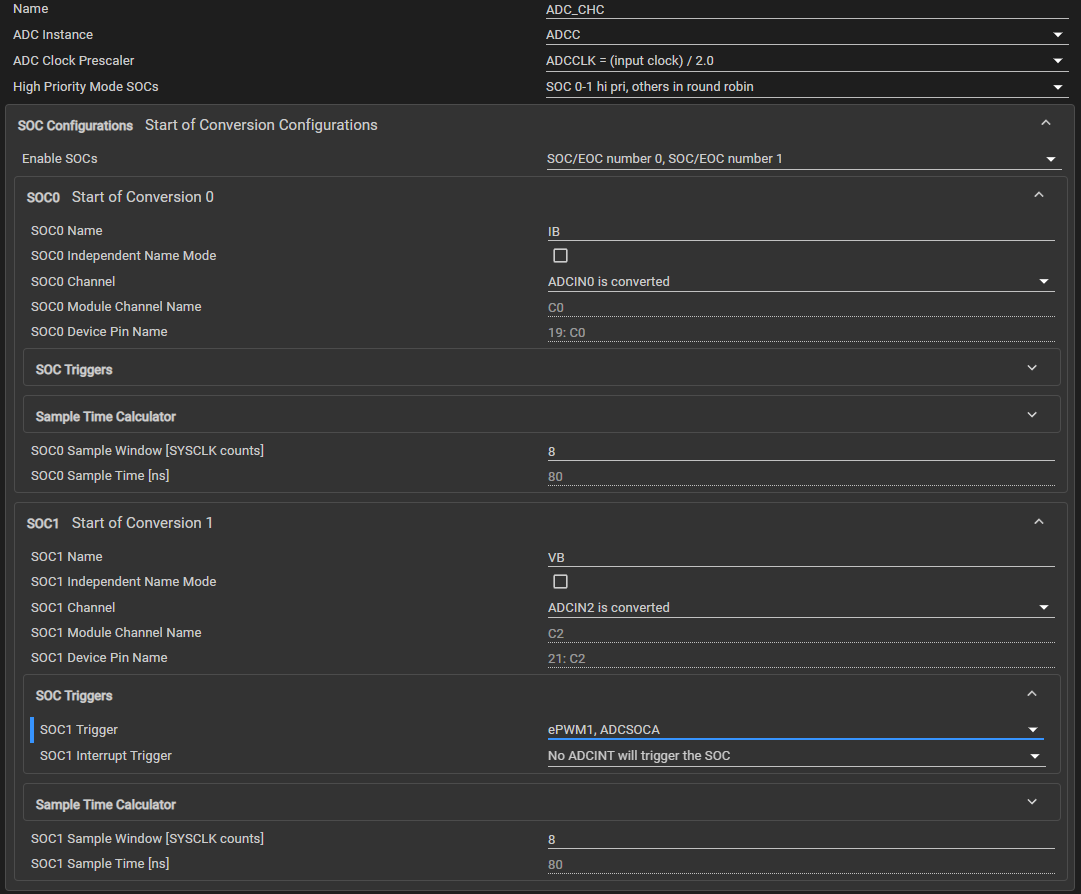


相似地，根据上表设置ADC2的相关内容。应当注意到Tc并不参与高优先级，但是触发源仍然可以使用Epwm1。



主中断和ADC1绑定，使用ADC2时不需要额外配置中断。

相似地，可以完成ADC3的设置



读取ADC的数据，可以使用下面的函数示例：

|  |
| --- |
|  |
| **#define** IFBU ADC\_readResult(ADC\_A\_RESULT\_BASE, ADC\_A\_MOTOR\_IU)  **#define** IFBV ADC\_readResult(ADC\_B\_RESULT\_BASE, ADC\_B\_MOTOR\_IV)  **#define** IFBW ADC\_readResult(ADC\_A\_RESULT\_BASE, ADC\_A\_MOTOR\_IW)  **#define** IFBU\_PPB ADC\_readPPBResult(ADC\_A\_RESULT\_BASE, ADC\_A\_MOTOR\_IU\_PPB)  **#define** IFBV\_PPB ADC\_readPPBResult(ADC\_B\_RESULT\_BASE, ADC\_B\_MOTOR\_IV\_PPB)  **#define** IFBW\_PPB ADC\_readPPBResult(ADC\_A\_RESULT\_BASE, ADC\_A\_MOTOR\_IW\_PPB)  **#define** VDC\_EVT ADC\_readResult(ADC\_A\_RESULT\_BASE, ADC\_A\_MOTOR\_VDC) |

CMPSS是比较器子模块，可以考虑使用这一模块实现过电流保护等功能。

四. PWM发生器

接下来设置PWM发生器，并且使用PWM发生器正确触发ADC中断。

图形用户界面, 文本, 应用程序

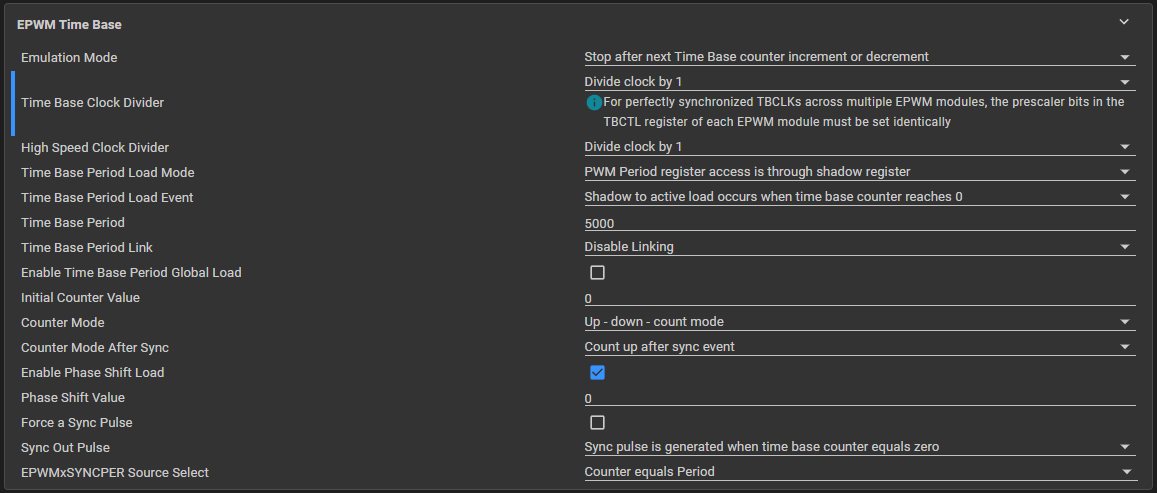
描述已自动生成

如果没有指定硬件的选项，可以在末尾PinMux中设置正确的PWM引脚。

图形用户界面, 文本, 应用程序

描述已自动生成

下面正确设置ePWM模块的时基



接下来设置EPWM的比较器值。

图形用户界面, 文本, 应用程序

描述已自动生成

设置比较器触发动作

文本

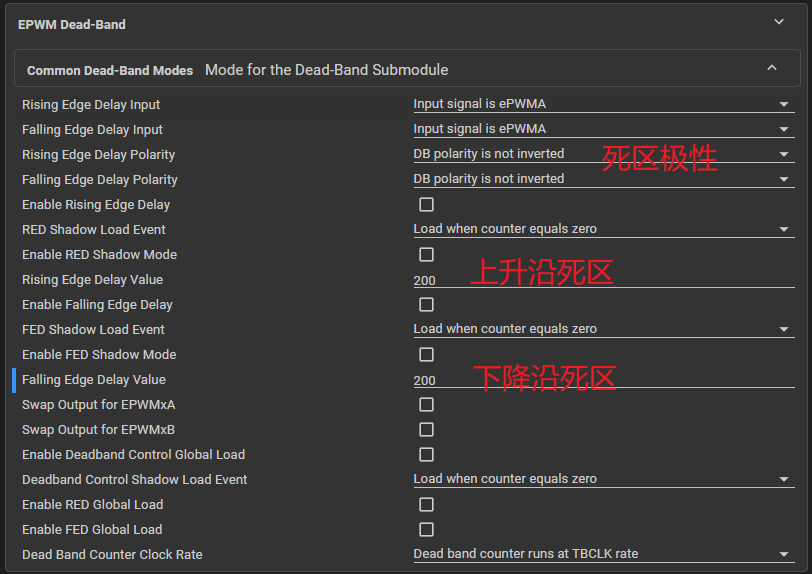
描述已自动生成

ePWMxB的触发动作由死区控制器发生，因此ePWMxB的行为是全空。

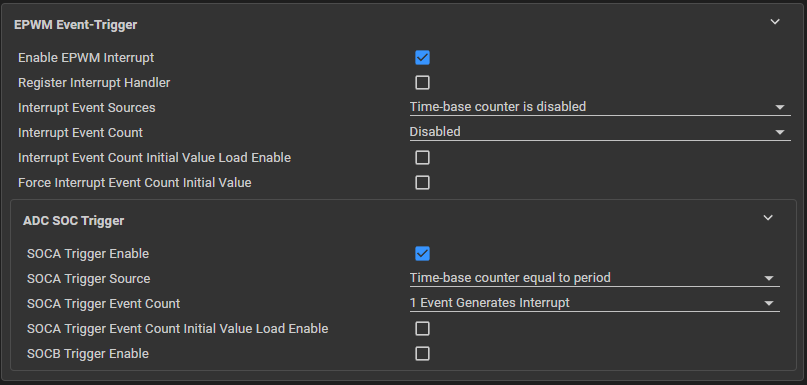
文本

中度可信度描述已自动生成

死区控制器设置如下：



在完成以上设置之后需要完成EPWM事件触发。



|  |
| --- |
|  |
| **EPWM\_setCounterCompareValue**(PHASE\_U\_PWM\_BASE, *EPWM\_COUNTER\_COMPARE\_A*,  (uint16\_t)((INV\_PWM\_HALF\_TBPRD \* pwm1.Vabc\_pu[0]) +  INV\_PWM\_HALF\_TBPRD));  **EPWM\_setCounterCompareValue**(PHASE\_V\_PWM\_BASE, *EPWM\_COUNTER\_COMPARE\_A*,  (uint16\_t)((INV\_PWM\_HALF\_TBPRD \* pwm1.Vabc\_pu[1]) +  INV\_PWM\_HALF\_TBPRD));  **EPWM\_setCounterCompareValue**(PHASE\_W\_PWM\_BASE, *EPWM\_COUNTER\_COMPARE\_A*,  (uint16\_t)((INV\_PWM\_HALF\_TBPRD \* pwm1.Vabc\_pu[2]) +  INV\_PWM\_HALF\_TBPRD)); |

六. GPIO使用

可以用Sysconfig非常方便地配置GPIO

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

在GMP环境中，可以使用下面的程序控制GPIO的输出：（此处以阴极接GPIO的情况为例）

|  |
| --- |
|  |
| gmp\_hal\_gpio\_write(LEDR, 0); // 点亮红灯  gmp\_hal\_gpio\_write(LEDG, 1); // 关闭绿灯 |

七. CLA的使用

以CLA只使用一个任务为例，需要在CLA配置中选择Task1，并给出Task1的名字，如下图所示：

文本

中度可信度描述已自动生成

接下来使能CLA的CPU端中断：这个中断将会在CLA运行结束之后被调用。

图形用户界面

低可信度描述已自动生成

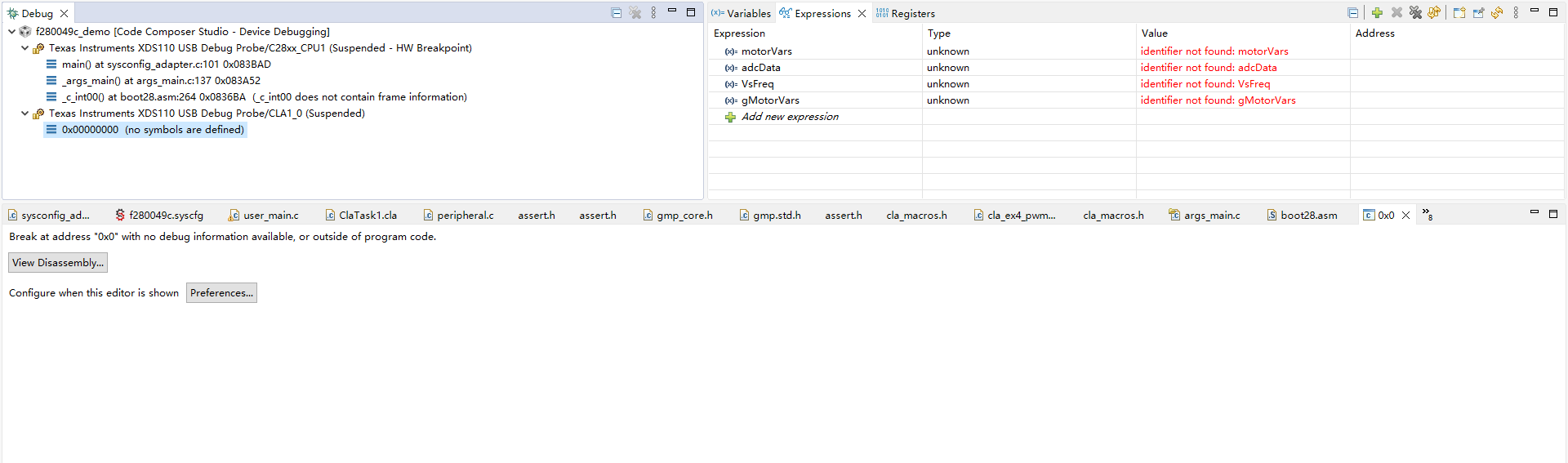
之后需要赋给CLA程序的运行空间，所以需要在MEMCFG页面中设置CLA的运行程序空间。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

使用了CLA之后应当使用新的连接命令文件，否则连接器会找不到几个关键变量的定义。

在调试CLA的步骤如下，在正确部署CLA程序之后，进入调试模式如下：

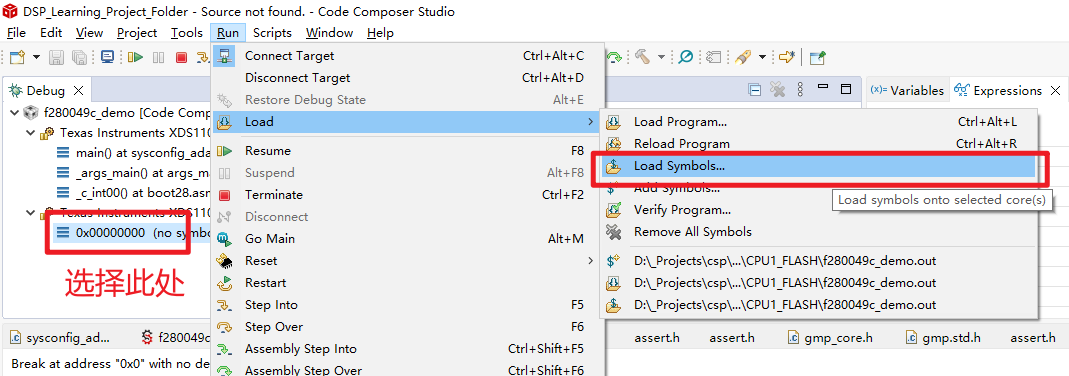


如果CLA没有链接，需要右键选择链接

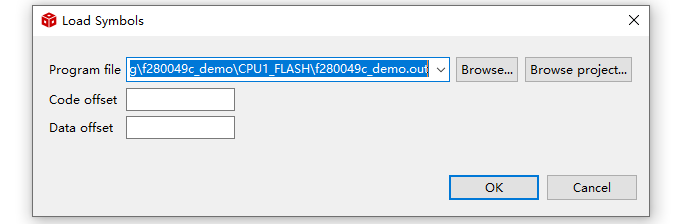
图形用户界面, 文本, 应用程序

描述已自动生成

此时CLA的程序并没有正确加载符号，所以可以，可以在下图所示的路径中找到加载符号：



进入符号导入界面，



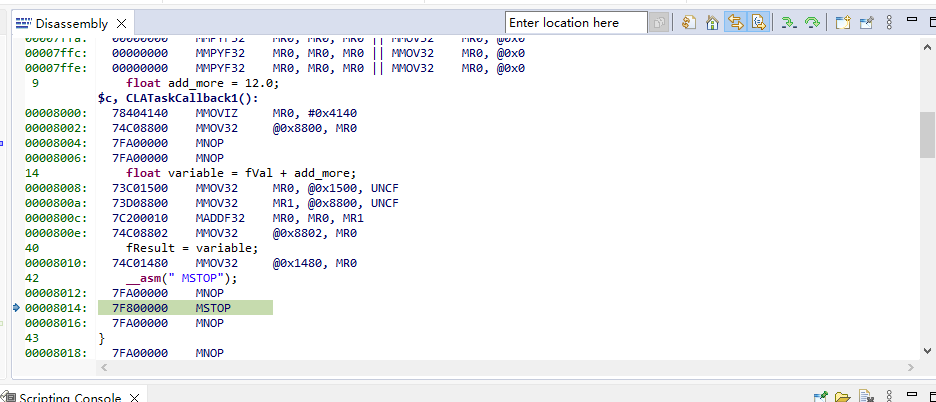
在CLA的程序中断中，C28x内核会直接清空流水线，CLA并不会清空流水线，所以会对于数值计算指令可能在经过3个或更多指令之后才会正确刷新结果。

如果没有正确加载符号，则只能显示CLA执行的代码，但并不会显示对应的C语言代码。

图形用户界面, 文本, 应用程序

描述已自动生成

正确导入符号之后，则能够正确显示对应的C语言代码：



CLA函数的结尾应当使用，这一条汇编指令将会在函数结束时自动执行，不需要用户显示手动给出。

\_\_asm(" MSTOP");

作为任务结束的标志，并且触发中断。

目前仍然不能正确触发CLA的完成中断，不明白为什么。解决问题，sysconfig生成的程序不会自动启动CPU全局中断开关，中断程序并不会执行。已经在CSP中补充了启动全局中断的程序。

在实际使用中，应当注意CLA中的代码如果涉及判断语句，那么CLA编译生成的汇编代码将会非常严重的膨胀。

八. SPI外设的使用

特别注意，使用SPI发送数据时发送的是txdata的高8位（左对齐），接收数据时时右对齐。

基本示例参考如下：

|  |
| --- |
|  |
| **#define** writeSPI(tx\_data) SPI\_writeDataBlockingNonFIFO(DRV\_SPI\_BASE, tx\_data)  **#define** readSPI() SPI\_readDataNonBlocking(DRV\_SPI\_BASE)  **#define** getStatus\_RxDataTxEmpty() (HWREGH(DRV\_SPI\_BASE + SPI\_O\_STS) & SPI\_STS\_INT\_FLAG) |