第5章 基础应用

CHAPTER 5

本章主要介绍智能平衡移动机器人 TMS320F28069 片内外设的使用，了解片内外设的基本原理，并结合 MATLAB/Simulink 实现 TMS320F28069 单片机外设的模型搭建实验环境及自动代码生成。

5.1 GPIO

TMS320F28069 有 54 个 GPIO，对应芯片输出的 54 个引脚。这些引脚分为 A、B 两组：A 组包括 GPIO0～GPIO31，B 组包括 GPIO32～GPIO58，其中 B 组的引脚中不含 GPIO45～GPIO49 引脚。每个引脚都有自己的复用功能，可以根据使用手册进行配置。

GPIO 在当作通用 I/O 口使用的时候，可以通过 GPxDIR 配置 I/O 口的方向（1 为输出，0 为输入）。GPxMUXn 用于配置 GPIOn 的复用功能，GPxPUD 用于配置 I/O 口的上拉功能（0 为使能上拉），也可以使用量化寄存器 GPxQSEL 对输入信号进行量化限制，从而消除数字量 I/O 引脚的噪声干扰。

此外，还有 4 种方式可对 GPIO 引脚进行读写操作：可以通过 GPxDAT 寄存器独立读/写 I/O 口信号；使用 GPxSET 寄存器写 1 对 I/O 口进行置位操作；使用 GPxCLEAR 寄存器写 1 对 I/O 口进行清零操作；使用 GPxTOOGLE 寄存器写 1 对 I/O 口进行电平翻转操作。需要注意的是，以上操作写 0 时均无效。

5.1.1 GPIO_OUTPUT 控制 LED 灯


视频讲解

作为第一个应用案例，以“点亮一个 LED 灯”开始。硬件上的蓝色 LED 灯对应的引脚为 GPIO31，与红色 LED 灯对应的引脚为 GPIO25，如图 5.1 所示。由于这两个 LED 灯都是共阴连接方式，当 GPIO 为高电平（置 1）时，则 LED 灯被点亮；当 GPIO 为低电平（置 0）的时候，则 LED 灯熄灭。首先对整个模型进行配置[①]。

打开 Simulink 的模型在 Simulation 工具栏下打开模型配置参数（Model Configuration

① 此处基本配置适用于所有模型，后续实验模型的基本配置都可参考本节设置。

Pararmeters)界面或者直接单击 工具,如图 5.2 所示。

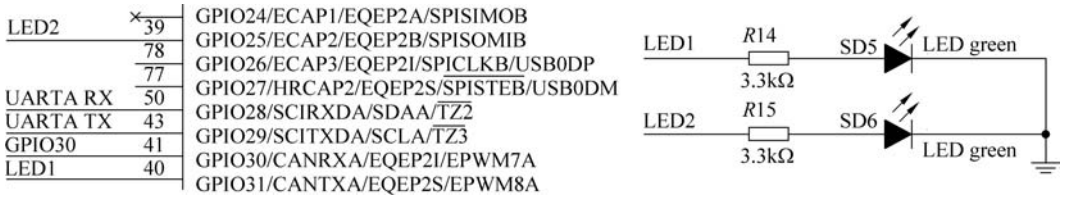


图 5.1 LED灯引脚连接图

图 5.2 模型参数配置

首先配置 Solver 选项卡,如图 5.3 所示,设置 Stop Time 为 inf。Solver selection 为定步长离散解算器,也就是 Type 选择 Fixed-tep,并选择 discrete(no continuous states)。定步长设置根据实际情况确定,默认为 auto,单位为秒,此定步长相当于定时器 0 的中断时间间隔。

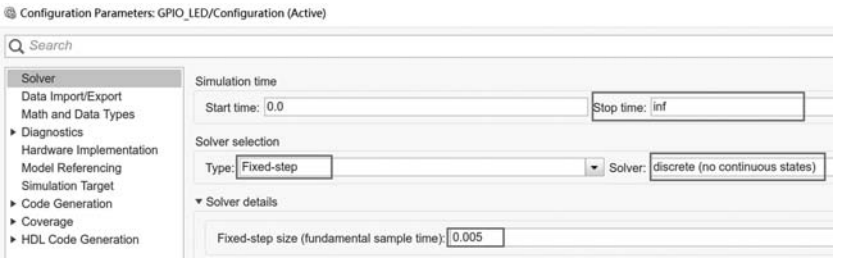


图 5.3 解算器配置

在配置过程中需要注意的是,解算器类型必须选择固定点解算器。固定点 Solver 中提供了多种算法,此模型由于没有连续状态,所以可以选择 discrete 方法。步长默认为 auto,在简单的通用嵌入式代码生成过程中此参数没有实际作用,可以采用默认或设置 0.005s。而在针对目标芯片定制的代码生成过程中,硬件驱动工具箱往往会将步长 step size 作为其外设或内核中定时器的中断周期,使得生成的算法代码在硬件芯片中以同样的时间间隔执行。由于解算器步长为整个模型提供了一个基础采样频率,故被称为基采样率(base-rate)。

在 Diagonostics 选项卡的 Data Validity 选项中,将 Multitask data store 配置为 none。

在 Hardware Implementation 中选择 Hardware board 为 TI Piccolo F2806x 或者 TI Piccolo F2806x(boot from flash),前者为将程序烧写到 RAM 中,适合调试,程序在芯片掉电后丢失;后者为将程序烧写到 Flash 中,适合运用到实际应用中,掉电后不丢失,提供的示例模型选择为后者。这时 Simulink 会在 Device type 中自动选定 TI C2000 系列。然后配置该界面下 Build action 为 Build,load and run(默认)。Device Name 为 F28069。选中 Use custom linker command file 复选框,如图 5.4 所示。然后在 Clocking 选项中,外部晶振默认为 10MHz,将系统时钟配置最高为 90MHz,LSPCLK 低速时钟外设设置为

SYSCLKOUT/4 分频，如图 5.5 所示。其他外设模块按自己需要配置。

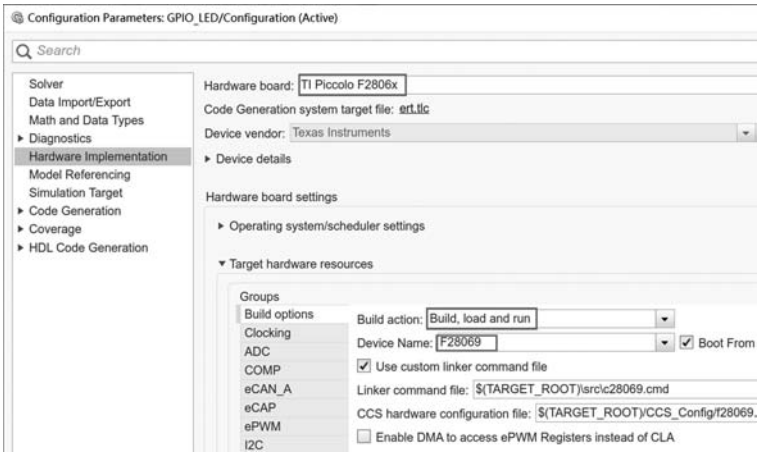


图 5.4 硬件配置

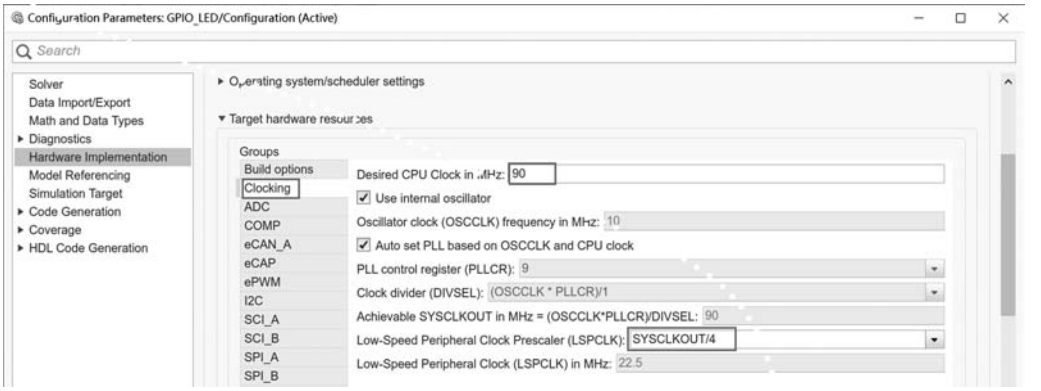


图 5.5 时钟配置

在 Code Generation 选项卡中使用的 Toolchain 为 Texas Instruments Code Composer (C2000)。在 Code generation objectives 的 Prioritized objectives 中将执行效率、ROM 效率、RAM 效率设置为优先的代码生成目标，如图 5.6 所示。在 Report 中选中 Generate model Web view 复选框，使生成的代码可以进行模型与代码之间相互的跟踪，如图 5.7 所示。

以上便完成了一个模型最基本的配置。

在 Simulink 界面上，搭建点亮一个 LED 灯的模型，系统步长为 0.005s，Counter Limited 设置 Upper limit 为 400，Compare To Constant 设置 Constant value 为 200，将 GPIOx 选择为 GPIO31，如图 5.8 所示，LED 流水灯模型如图 5.9 所示。

整个控制效果为：当计数值大于或等于 200 时，也就是大概有 $0.005 \times 200 = 1$s GPIO31 是置 1 的，另外 1s 是置 0 的。这样便实现了一个 LED 灯一亮一灭的效果。

要想实现流水灯的实验，可搭建如图 5.10 所示的模型。在下载模型之前也可以通过仿

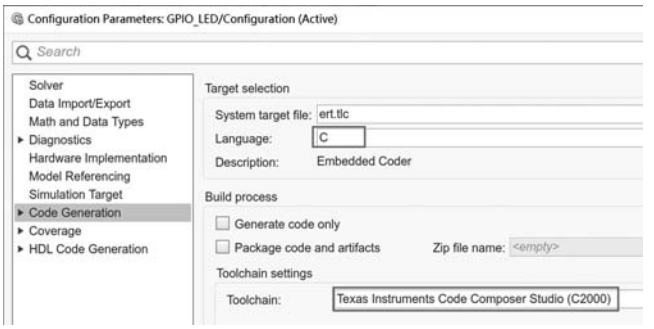


图 5.6 自动代码生成配置

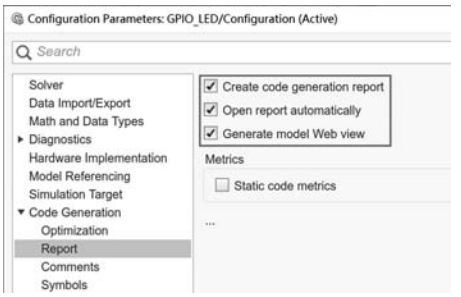


图 5.7 自动代码生成报告

真查看是不是预计的控制效果,如图 5.11 所示,加入 Scope 模块,单击“运行”按钮或者按 Ctrl+R 键,观察仿真结果如图 5.12 所示。由仿真结果可以看出,此控制模型为我们所要的结构,然后单击“编译”按钮或者按 Ctrl+B 键将此模型编译下载到主控板,观察实验现象,验证控制算法在实际硬件上运行时是否跟仿真结果一致。

图 5.8 GPIO 模块配置

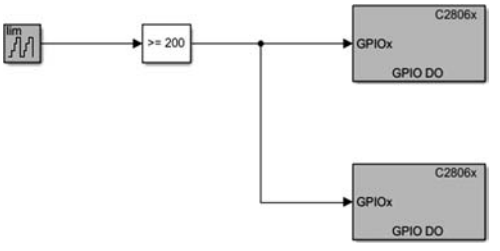


图 5.9 LED 流水灯模型

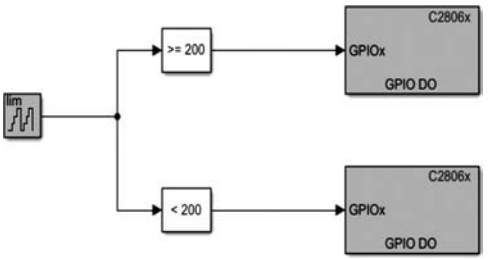


图 5.10　LED 流水灯模型

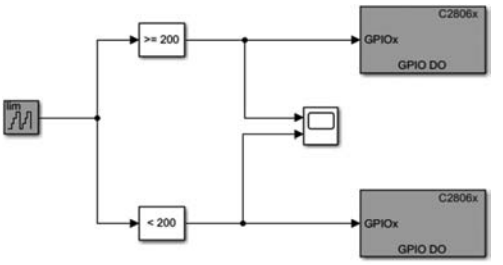


图 5.11　LED 流水灯模型

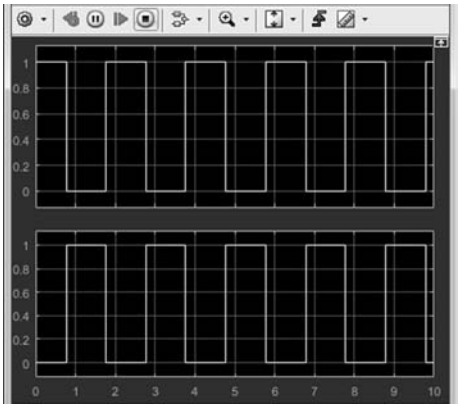


图 5.12　控制波形图

5.1.2　GPIO_INPUT 扫描_NORMAL 模式


视频讲解

在配置 GPIO 输入模式的时候，使用按键的端口输入值，作为 GPIO 输出模块的控制量。使用按键（对应 GPIO42）控制其中一个 LED 灯（对应 GPIO25）的亮灭，另一个 LED 灯（GPIO31）用于验证模型是否下载到硬件中，按键硬件原理图如图 5.13 所示。

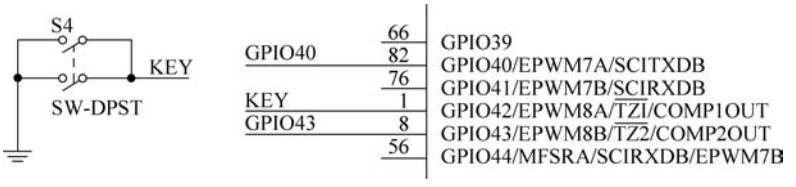


图 5.13　按键硬件原理图

配置 GPIO 的 PUD（上拉禁止寄存器）使能 GPIO42 的电平上拉，当按键按下时，GPIO42 检测到低电平（置 0），当没按下时 GPIO42 为高电平（置 1）。因此需要在模型中先将 GPIO42 使能上拉，如图 5.14 所示。

注意：通过执行 EALLOW 指令允许 CPU 自由写入受保护的寄存器。在修改寄存器之后，可以通过执行 EDIS 指令清除 EALLOW 位使它们再次受到保护，EALLOW 和 EDIS

图 5.14　初始化配置 GPIO42

一般是成对出现的。

GPIO DI 选择 GPIO25，GPIO31 需要选择 Toggle，Toggle 表示电平翻转。搭建的按键控制模型如图 5.15 所示，仿真步长为 0.5s。

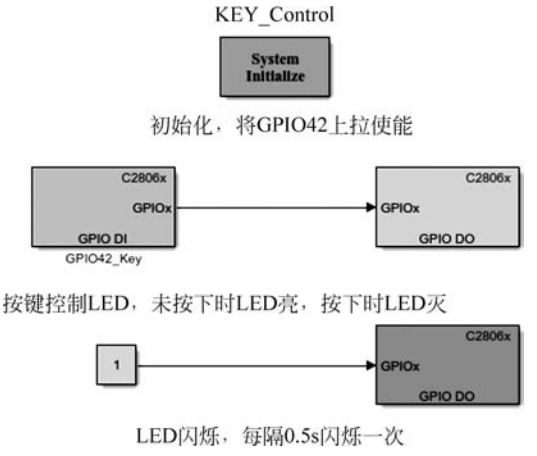


图 5.15　按键实验模型

在上述配置完成，并完成模型的搭建后，将模型编译下载到主控板。先观察红色的 LED 灯有没有闪烁，再通过控制按键按下与否来控制蓝色 LED 灯的状态。


视频讲解

5.1.3　GPIO_INPUT 扫描_EXTERNAL 模式

需要注意的是，Simulink 的 External 模式默认使用的是 SCIA，波特率为 115200b/s，并且默认复用 I/O 口是 GPIO28、GPIO29，对应开发板上的 SCIA 接口是 RXA、TXA，如图 5.16 所示。所以在进行 External 模式操作的时候，通过一根 USB 转 TTL 通信线，将 USB 的 RX、TX 分别接在 F28069 主控板的 TXA、RXA 引脚。注意，USB 和 F28069 的控制板要共地，同时不能将线序接反，否则 Simulink 无法通过 SCI 转 USB 与 DSP 进行通信。使用 External 模式可极大地方便对数据的观测。

首先需要在模型配置时对外部模式进行设置，如图 5.17 所示。将 Communication interface 设置为串行通信，然后在“设备管理器”中找到串行通信的 COM 口，输入到 Serial port 中，如图 5.18 所示。GPIO 外部模式模型如图 5.19 所示。

注意：GPIO DI 模块输出数据类型选择 uint8，如图 5.20 所示。

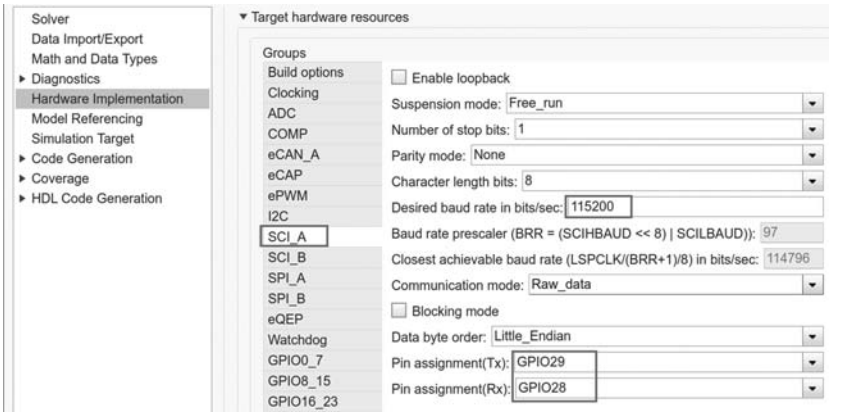


图 5.16　SCI 配置

图 5.17　配置外部模式

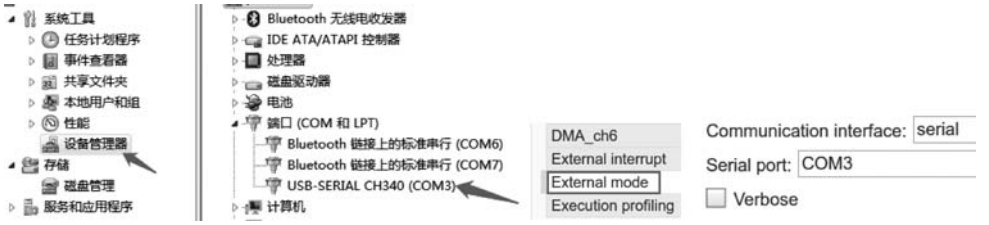


图 5.18　串口配置

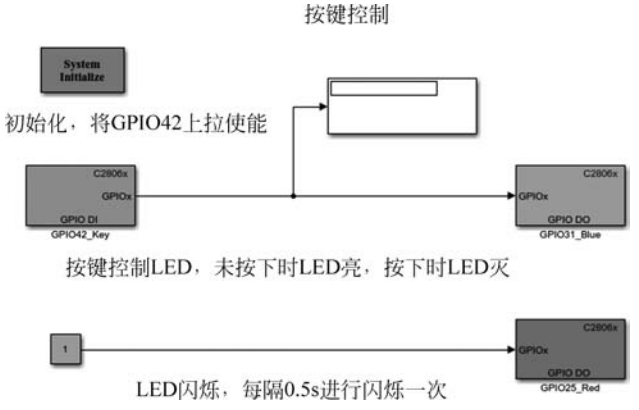


图 5.19　GPIO 外部模式模型

图 5.20　GPIO DI 模块数据类型配置

在上述配置完成后,再次单击 Simulink 界面的“运行”按钮将代码下载到开发板上,可以看到,Simulink 处于仿真运行状态,按下主控板按键,可以在 Display 模块中看到数值的变化。

由于 GPIO4 默认上拉使能,所以在按键未按下前 Dispaly 显示 1,按下后显示 0,如图 5.21 所示。同时观察蓝色 LED 灯的亮灭情况可以判断 Display 显示是否正确,如图 5.22 所示。

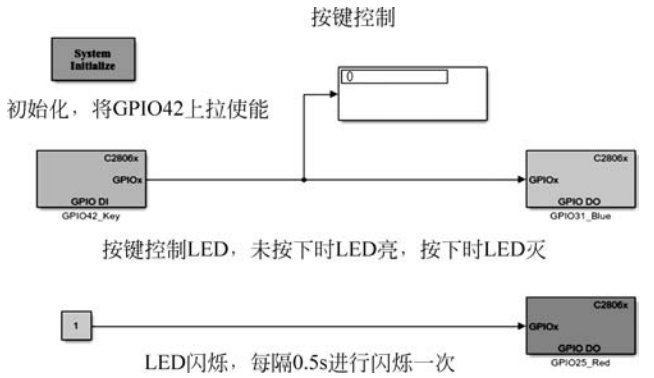


图 5.21 按键按下实验现象

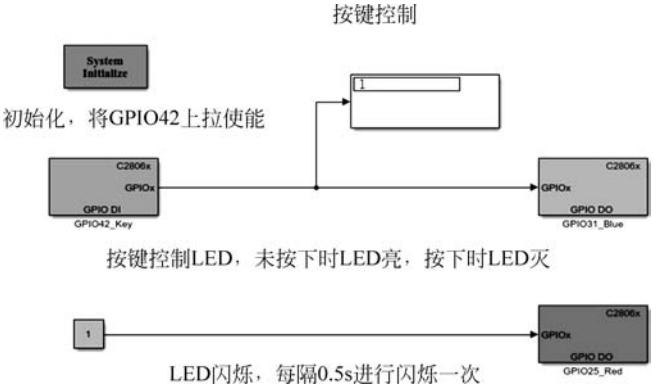


图 5.22 按键松开实验现象

5.1.4 GPIO_INPUT 中断控制 LED 灯

GPIO 不仅可以实现通用 I/O 数字量输出的功能,还可以配置成外部中断实现控制目的。比如将最常用的按键配置成外部中断,在中断中执行参数或模式的修改,这样比常用的按键扫描方式要节省资源。

在 Simulink 中搭建模型,配置 GPIO42 为外部中断触发源,并将中断配置为下降沿触发,对应外部中断 3。在中断函数中分别进行对应的 LED 的翻转,GPIO42 对应 LED1 灯(GPIO31)、LED2 灯(GPIO25)。仿真模型如图 5.23 所示,系统仿真步长为 0.05s。

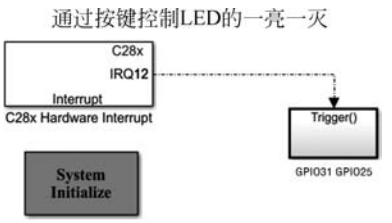


图 5.23 中断仿真模型

其中 System Initialize 中写入的是 GPIO42 的初始化代码，如图 5.24 所示。

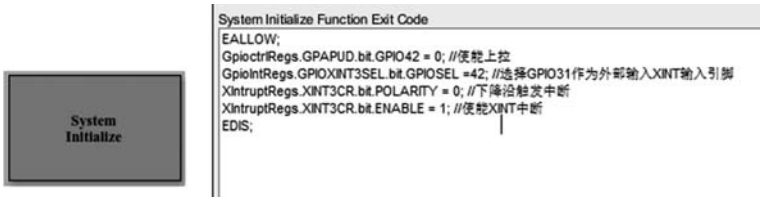


图 5.24 GPIO 初始化配置

硬件中断模块配置如图 5.25 所示，中断号为 CPU-12，PIE-1 也就是外部中断 XTIN3，任务优先级默认，Preemption flags 输入 1 表示中断可以被抢占。输入 0 表示中断不能被抢占。更多的详细信息请单击该模块的 Help 按钮了解。

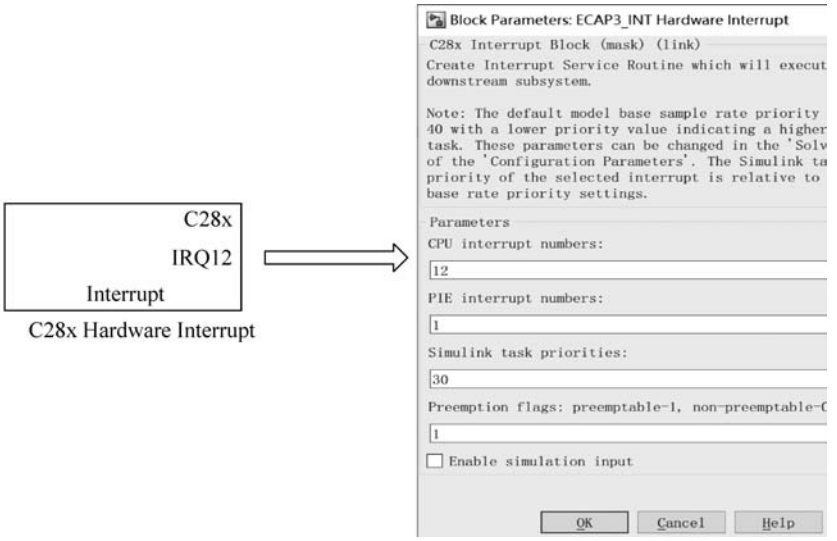


图 5.25 中断配置

Trigger 子系统里面为 LED 灯的控制模型，如图 5.26 所示，且在控制模型的 GPIO Do 模块中不选中 Toggle。

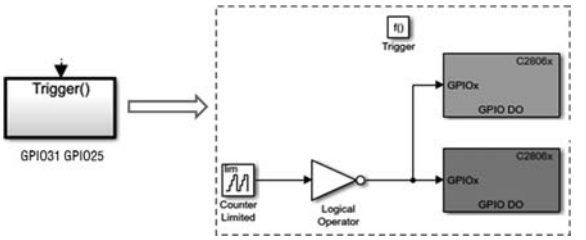


图 5.26 GPIO 中断触发模型

在上述配置完成,并完成模型的搭建后,将生成的工程代码下载到主控板上。按下一次按键,可以发现对应的 LED 灯电平便翻转一次。


视频讲解

5.2 ADC

5.2.1 ADC 基本原理

1. ADC 转换步骤

A/D 转换器(ADC)将模拟量转换为数字量通常要经过 4 个步骤:采样、保持、量化和编码。

所谓采样,就是将一个时间上连续变化的模拟量转化为时间上离散变化的模拟量,如图 5.27 所示。

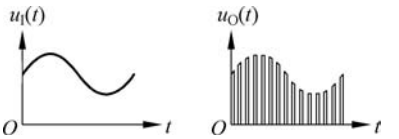


图 5.27 A/D 采样原理

将采样结果存储起来,直到下次采样,这个过程称作保持。一般地,采样器和保持电路一起总称为采样保持电路。

将采样电平归化为与之接近的离散数字电平,这个过程称作量化。

将量化后的结果按照一定数制形式表示就是编码。

将采样电平(模拟值)转换为数字值时,主要有两类方法:直接比较型与间接比较型。

(1) 直接比较型:就是将输入模拟信号直接与标准的参考电压比较,从而得到数字量。这种类型常见的有并行 ADC 和逐次比较型 ADC。

(2) 间接比较型:输入模拟量不是直接与参考电压比较,而是将二者变为中间的某种物理量再进行比较,然后将比较所得的结果进行数字编码。这种类型常见的有双积分型的 ADC。

2. ADC 转换原理

采用逐次逼近法的 A/D 转换器由比较器、D/A 转换器、N 位寄存器及控制逻辑电路组成,如图 5.28 所示。

基本原理是从高位到低位逐位试探比较,好像用天平称物体,从重到轻逐级增减砝码进

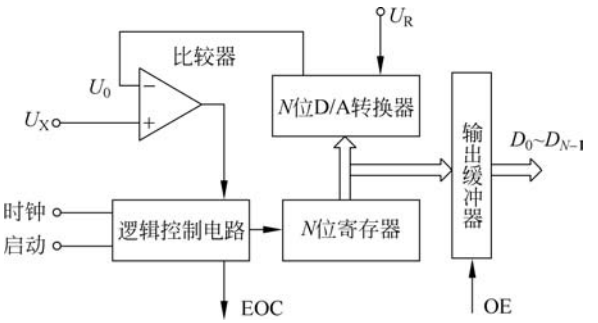


图 5.28　逐次逼近式 A/D 转换器原理图

行试探。逐次逼近法的转换过程是：初始化时将逐次逼近寄存器各位清零，转换开始时，先将逐次逼近寄存器最高位置 1，送入 D/A 转换器，经 D/A 转换后生成的模拟量送入比较器，称为 U_0，与送入比较器的待转换的模拟量 U_X 进行比较，若 $U_0 < U_X$，则该位 1 被保留，否则被清除。然后再置逐次逼近寄存器次高位为 1，将寄存器中新的数字量送 D/A 转换器，输出的 U_0 再与 U_X 比较，若 $U_0 < U_X$，则该位 1 被保留，否则被清除。重复此过程，直至逼近寄存器最低位。转换结束后，将逐次逼近寄存器中的数字量送入缓冲寄存器，得到数字量的输出。逐次逼近的操作过程是在一个控制电路的控制下进行的。

采用双积分法的 A/D 转换器由电子开关、积分器、比较器和控制逻辑等部件组成，如图 5.29 所示。

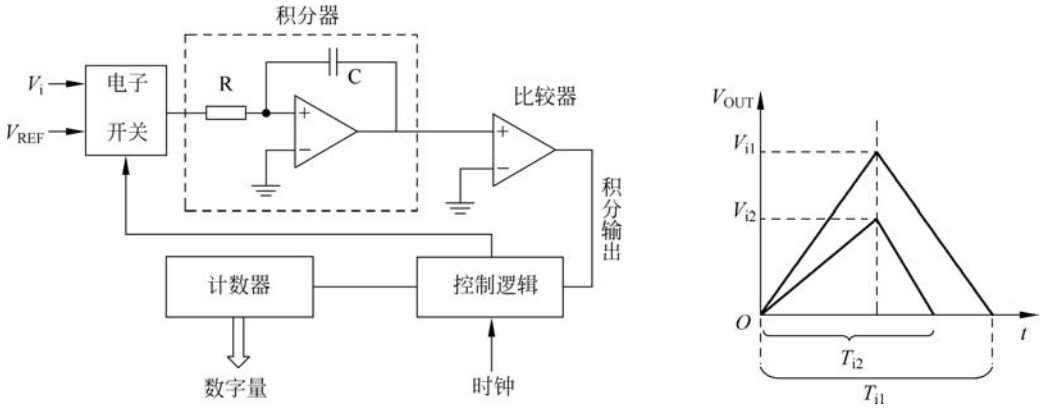


图 5.29　双积分法的 A/D 转换器

基本原理是将输入电压变换成与其平均值成正比的时间间隔，再把此时间间隔转换成数字量，属于间接转换。双积分法 A/D 转换的过程是：先将开关接通待转换的模拟量 V_i，V_i 采样输入到积分器，积分器从零开始进行固定时间 T 的正向积分，时间 T 到后，开关再接通与 V_i 极性相反的基准电压 V_{REF}，将 V_{REF} 输入到积分器，进行反向积分，直到输出为 0V 时停止积分。V_i 越大，积分器输出电压越大，反向积分时间也越长。计数器在反向积分

时间内所计的数值,就是输入模拟电压 V_i 所对应的数字量,实现了 A/D 转换。

3. ADC 关键技术指标

(1) 分辨率(Resolution)指数字量变化一个最小量时模拟信号的变化量,定义为满刻度与 2^n 的比值。分辨率又称精度,通常以数字信号的位数来表示。

(2) 转换速率(Conversion Rate):也可以称为 A/D 采样率,是 A/D 转换一次所需要时间的倒数。单位时间内,完成从模拟转换到数字的次数。积分型 A/D 的转换时间是毫秒级,属低速 A/D;逐次比较型 A/D 是微秒级,属中速 A/D,全并行/串并行型 A/D 可达到纳秒级。采样时间则是另外一个概念,是指两次转换的间隔。为了保证转换的正确完成,采样速率(Sample Rate)必须小于或等于转换速率。因此有人习惯上将转换速率在数值上等同于采样速率也是可以接受的。常用单位是 ksps 和 Msps,表示每秒采样千/百万次(kilo/Million Samples per Second)。

(3) 量化误差(Quantizing Error):由于 A/D 的有限分辨率而引起的误差,即有限分辨率 A/D 的阶梯状转移特性曲线与无限分辨率 A/D(理想 A/D)的转移特性曲线(直线)之间的最大偏差。通常是 1 个或半个最小数字量的模拟变化量,表示为 1LSB、1/2LSB。

(4) 偏移误差(Offset Error):输入信号为零时输出信号不为零的值,可外接电位器调至最小。

(5) 满刻度误差(Full Scale Error):满度输出时对应的输入信号与理想输入信号值之差。

(6) 线性度(Linearity):实际转换器的转移函数与理想直线的最大偏移,不包括以上 3 种误差。

其他指标还有绝对精度(Absolute Accuracy)、相对精度(Relative Accuracy)、微分非线性、单调性和无错码、总谐波失真(Total Harmonic Distotortion,THD)和积分非线性。

TMS320F2806x 的 ADC 模块主要包括以下内容。

- 12 位模数转换。
- 2 个采样保持器(S/H)。
- 同步采样或顺序采样。
- 模拟电压输入范围 0～3.3V。
- 16 通道模拟输入。
- 16 个结果寄存器存放 ADC 转换的结果。
- 多个触发源:S/W——软件立即启动,ePWM1～ePWM8,外部中断 2 脚,定时器 0、1、2 以及 A/D 中断 1、2。

关于 ADC 单元寄存器的具体描述在这里不再进行具体介绍,感兴趣的读者可以参考 TMS320F28069(以下简写为 F28069)的数据手册,里面有详细讲述,F28069 主控板关于 A/D 引脚原理图如图 5.30 所示。

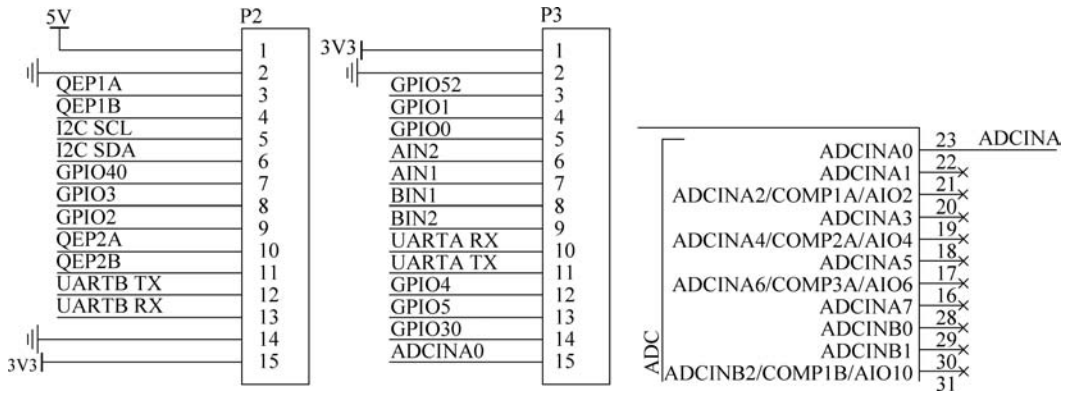


图 5.30　A/D 引脚原理图

5.2.2　ADC_NORMAL 模式

ADC_NORMAL 模式配置先从配置时钟开始，ADC 模块是挂在高速外设时钟线上的外设，Clocking 配置为系统 90MHz 时钟 2 分频。在 ADC 配置中，使用默认的分频系数 ADCLK＝2，得到 ADC 模块时钟为 45MHz，其他使用默认配置。具体配置如图 5.31 所示。

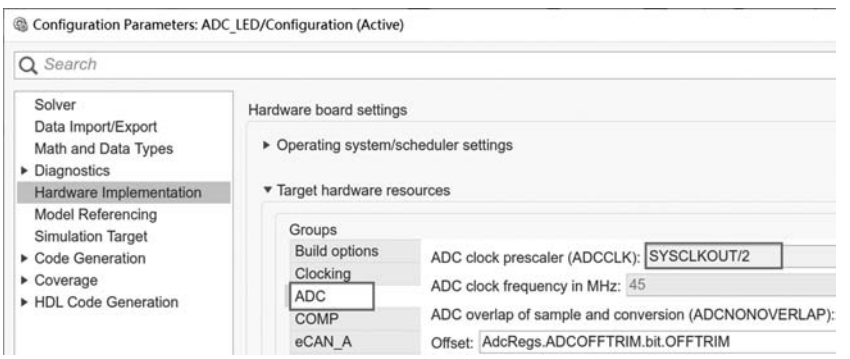


图 5.31　ADC 模块配置

采样模式为单个采样模式，SOC 触发数为 SOC0，采样窗口为 7，SOCx 触发源为软件触发，输出数据类型为 uint16，输入通道配置为 A0，此模块数据基本为默认。具体配置如图 5.32 和图 5.33 所示。

在以上配置完成后，对 ADC 采样的值进行转换。由于 ADC 转换结果寄存器是 16 位的，且数值是左对齐的 12 位数据，所以要进行左移 4 位的操作。官方支持包人性化地做了这一点，只需直接对输出的结果进行转换就可以得到实际的采样值。本例中采集的是外部电位器的 0～3.3V 的电压，所以直接进行转换（因为是 12 位的 ADC，故满量程为 4096，也就是 4096 对应实际的参考电压 3.3V）。当采集到的模拟量少于 2048 时，使 LED2 灯闪烁，否则使 LED1 灯闪烁。模型搭建如图 5.34 所示。模型中的 Rate Transition 为高采样速率向低采样速率转换。

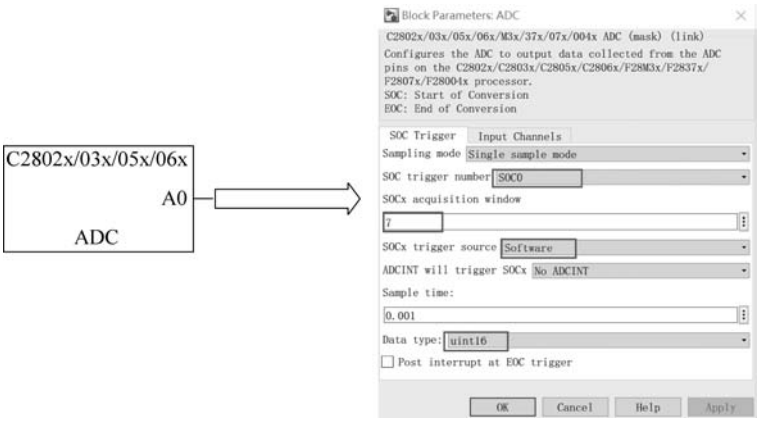


图 5.32　ADC 模块配置

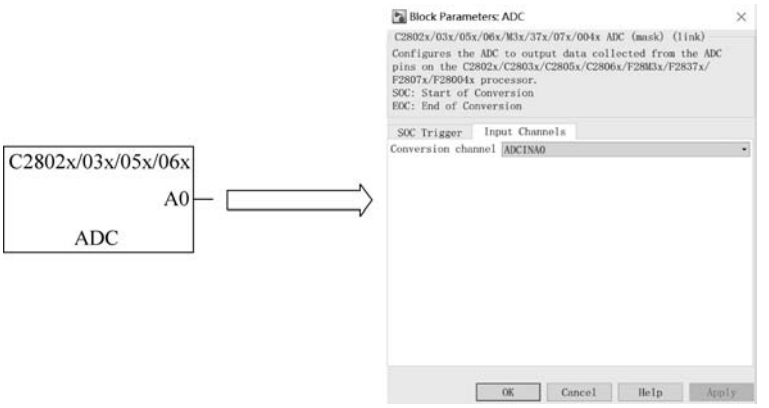


图 5.33　ADC 模块输入通道配置

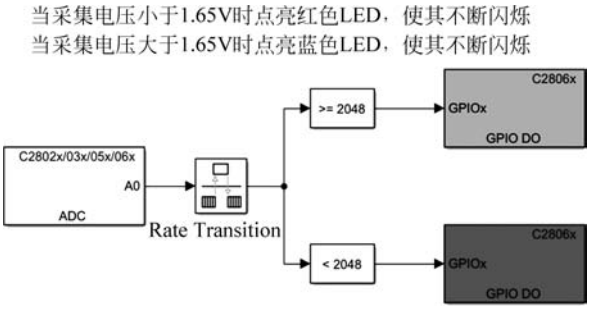


图 5.34　电压采集模型图

在上述配置完成，并完成模型的搭建后，将模型编译下载到主控板。将电位器的中间引脚接在主控板的 AD0 引脚，其他两个引脚分别接在主控板的 3.3V 和 GND 引脚。可以发现，在顺时针和逆时针调节电位器时，两个 LED 灯都发生了变化。

5.2.3 ADC_EXTERNAL 模式

先将模型按照前述方法配置为外部模式。

在 Simulink 中搭建模型，如图 5.35 所示，将 ADC 输出值通过一个增益模块赋给 PWM 的占空比输入，并在显示模块上显示。其中，Gain 模块的输出数据为 uint16，ePWM 模块选择 ePWM1，这里 WA 相当于占空比的大小，相关配置在 5.3 节有详细介绍。

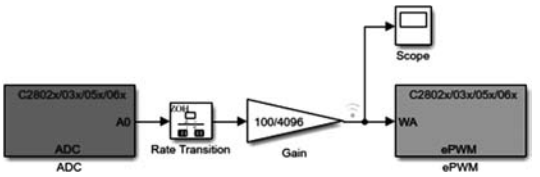


图 5.35 外部模式配置

在上述配置完成，并完成模型的搭建后，用 USB 转 TTL 线，分别将 USB 的 RX、TX 接在 F28069 主控板的 TXA、RXA 引脚上。启动仿真，通过调节电位器便能观察到 Scope 模块的数据变化，如图 5.36 所示。

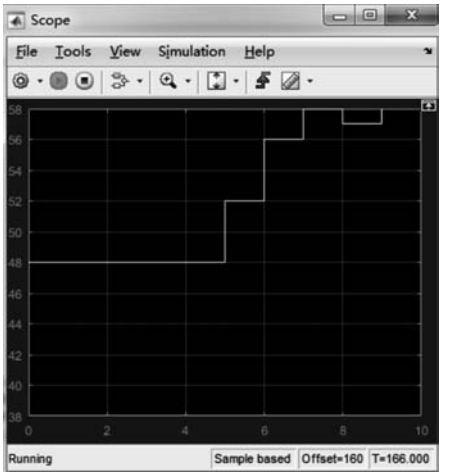


图 5.36 Scope 模块显示

5.3 Timer_IT


视频讲解

在多数情况下 Timer0 都作为系统默认的时基，如图 5.37 所示的 Base rate trigger 作为 Simulink 模型的触发率。那么触发步长在 Slover 中为 Fixed-step size(fundamental sample time)，如图 5.38 所示，配置 Simulink 模型的执行周期为 1s，也就是 1Hz 的执行频率。

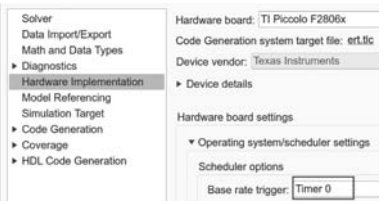


图 5.37　定时器中断配置图

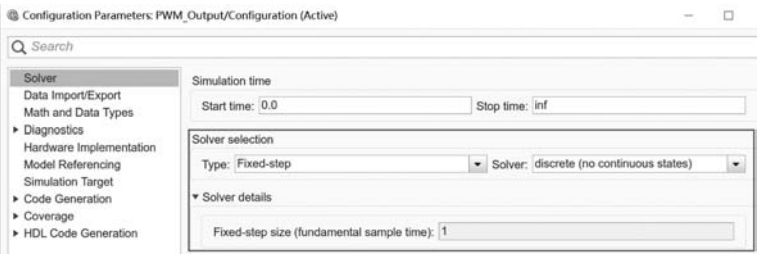


图 5.38　步长设置图

在上述配置完成,并完成模型的搭建后,如图 5.39 所示,将模型编译下载到主控板上。最终可以观察到,LED1 灯每秒闪烁一次。

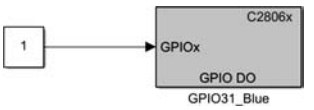


图 5.39　模型搭建图

可以把此模型生成的工程文件在 CCS 编译器中打开,复制 MTLAB 的工作路径 D:\MBD28069_BalanceCar\Chapter2\2.3Timer_IT,导入到 CCS 中,可以看到如图 5.40 所示的一些文件。

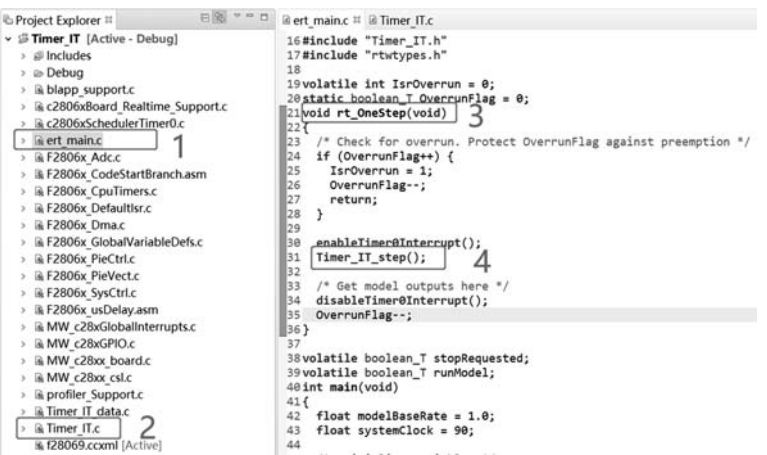


图 5.40　CCS 程序图

（1）ert. main. c：主要是对控制芯片一些初始化操作、while 循环和中断函数的调用。

（2）Timer_IT. c：主要包括初始化函数、定时器 0 中断操作函数、主函数里面的初始化函数。

（3）图 5. 40 中的 3 和 4 为定时器 0 中断执行函数，Timer_IT_step()函数为 rt_One_Step()函数的子函数，在 Simulink 中搭建的控制模型基本在定时器 0 中断函数执行，除了 C28x Hardware Interrupt、Idle Task 等一些特别声明的模块。

（4）如图 5. 41 所示，是 main 主函数的一些命令。

```
int main(void)
{
  float modelBaseRate = 1.0;
  float systemClock = 90;

  /* Initialize variables */
  stopRequested = false;
  runModel = false;
  c2000_flash_init();
  init_board();

#ifdef MW_EXEC_PROFILER_ON

  config_profilerTimer();

#endif

  ;
  bootloaderInit();
  rtmSetErrorStatus(Timer_IT_M, 0);
  Timer_IT_initialize();
  globalInterruptDisable();
  configureTimer0(modelBaseRate, systemClock);
  runModel =
    rtmGetErrorStatus(Timer_IT_M) == (NULL);
  enableTimer0Interrupt();
  globalInterruptEnable();
  while (runModel) {
    stopRequested = !(
                      rtmGetErrorStatus(Timer_IT_M) == (NULL));
  }
```

图 5. 41　CCS 程序图一

（5）modelBaseRate 也就是之前设定的定步长大小，systemClock 就是系统时钟大小，Flash 烧写初始化。

（6）init_board()主要是对 Simulink 模型配置参数中的 I/O 口进行初始化，Timer_IT_initialize()主要是对 Simulink 中搭建的一些 GPIO 进行初始化，如图 5. 42 所示，其中的 0xFCFFFFFF 是对 GPIO31 的配置。

```
/* Model step function */
void Timer_IT_step(void)
{
  /* S-Function (c280xgpio_do): '<Root>/GPIO31_Blue1' incorporates:
   *  Constant: '<Root>/Constant'
   */
  {
    if (Timer_IT_P.Constant_Value)
      GpioDataRegs.GPASET.bit.GPIO25 = 1;
    else
      GpioDataRegs.GPACLEAR.bit.GPIO25 = 1;
  }
}

/* Model initialize function */
void Timer_IT_initialize(void)
{
  /* Registration code */

  /* initialize error status */
  rtmSetErrorStatus(Timer_IT_M, (NULL));

  /* Start for S-Function (c280xgpio_do): '<Root>/GPIO31_Blue1' incorporates:
   *  Constant: '<Root>/Constant'
   */
  EALLOW;
  GpioCtrlRegs.GPAMUX2.all &= 0xFFFFFFCF;
  GpioCtrlRegs.GPADIR.all |= 0x40000;
  EDIS;
}
```

图 5. 42　CCS 程序图二

(7) configureTimer0()是计算定时器0产生中断的时间间隔。

(8) void ConfigCpuTimer(struct CPUTIMER_VARS * Timer, float Freq, float Period),CPUTIMER_VARS * Timer 表示选择哪个定时器,Freq 表示定时器频率,Period 表示定时器周期值。

(9) ConfigCpuTimer(&CpuTimer0,systemClock,baseRate * 1000000)。

(10) 计算公式为: T = systemClock * baseRate * 1000000/CpuTimer0,如图 5.43 所示。

```
    CpuTimer0Regs.TPR.all  = 0;
    CpuTimer0Regs.TPRH.all = 0;

    //
    // Make sure timer is stopped:
    //
    CpuTimer0Regs.TCR.bit.TSS = 1;

    //
    // Reload all counter register with period value:
    //
    CpuTimer0Regs.TCR.bit.TRB = 1;

    //
    // Reset interrupt counters:
    //
    CpuTimer0.InterruptCount = 0;
    #endif

    /* Configure CPU-Timer 0 to interrupt every 0.0025 sec. */
    /* Parameters:  Timer Pointer, CPU Freq in MHz, Period in usec. */
    ConfigCpuTimer(&CpuTimer0, systemClock, baseRate * 1000000);
    StartCpuTimer0();

    EALLOW;
    PieVectTable.TINT0 = &TINT0_isr;     /* Hook interrupt to the ISR*/
    EDIS;

    PieCtrlRegs.PIEIER1.bit.INTx7 = 1; /* Enable interrupt TINT0 */
    IER |= M_INT1;
}
```

图 5.43 CCS 程序图三

(11) 根据之前的设置,可以得到 T=90 * 1 * 1000000/90000000=1s。

(12) enableTimer0Interrupt()使能定时器0中断,globalInterruptEnable()使能全局中断,如图 5.44 所示。

```
    #ifdef PIEMASK10
      PieCtrlRegs.PIEIER11.all &= ~PIEMASK10;    /* disable
    #endif
    #ifdef PIEMASK11
      PieCtrlRegs.PIEIER12.all &= ~PIEMASK11;    /* disable
    #endif
    #ifdef PIEMASK12
      IER &= ~(M_INT13);
    #endif
    #ifdef PIEMASK13
      IER &= ~(M_INT14);
    #endif

    asm(" RPT #5 || NOP");              /* wait 5 cycles */
    IFR &= ~IFRMASK;                         /* eventually
    PieCtrlRegs.PIEACK.all = IFRMASK;        /* ACK to all
    IER |= 1;
    EINT;                               /* global interrupt
    rt_OneStep();
    DINT;                               /* disable global i
    #ifdef PIEMASK0
      PieCtrlRegs.PIEIER1.all = PIEIER1_stack_save;/*restore
    #endif
    #ifdef PIEMASK1
      PieCtrlRegs.PIEIER2.all = PIEIER2_stack_save;/*restore
    #endif
    #ifdef PIEMASK2
      PieCtrlRegs.PIEIER3.all = PIEIER3_stack_save;/*restore
    #endif
    #ifdef PIEMASK3
      PieCtrlRegs.PIEIER4.all = PIEIER4_stack_save;/*restore
```

图 5.44 CCS 程序图四

(13) while 为函数的循环语句，其中 Simulink 中 Idle Task 模块就是运行在 while 中。

首次在 CCS 中编译 Simulink 生成的代码前，需要先打开 f28069.ccxml [Active]，然后选择对应的仿真器型号，第一次打开默认是 XDS100v1，由于主控板使用的是 v2，所以选择 XDS100v2，先单击右侧的 Save 按钮，然后再选择对应的设备 TMS320F28069，最后单击 Test Connection 按钮，如图 5.45 所示，此时计算机会跟主控板通信，最终出现如图 5.46 所示的界面，说明已经配置成功。

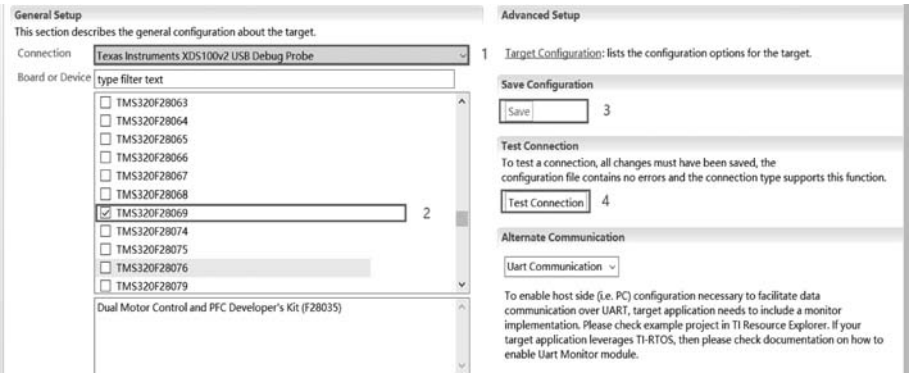


图 5.45　CCS 生成代码设置

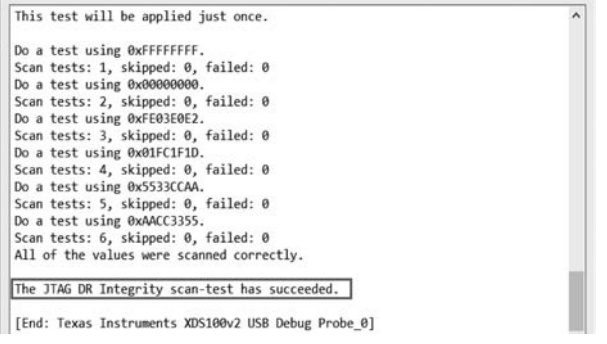


图 5.46　CCS 生成代码配置

当配置好 ccxml 文件后，下次编译工程时就不需要再进行操作了。现在就可以单击 形状的“编译”按钮，确认无误，如图 5.47 所示。

图 5.47　程序编译

然后再单击 形状的“调试”按钮，程序会下载到主控板，之后再单击“运行”按钮 ，就可观察到 LED1 灯开始闪烁，证明程序已经运行了。当单击“暂停”按钮 时，则程序会暂时停止在芯片上运行，这是 LED1 灯停止闪烁，当单击“终止”按钮 时，则程序停止在线

运行,如图 5.48 所示。关于 CCS 更多的操作请看合动智能科技公司的《CCS6.2 开发入门手册》。

图 5.48 程序运行与调试


视频讲解

5.4 ePWM

脉冲宽度调制(PWM)是一种对模拟信号电平进行数字编码的方法,其根据相应载荷的变化来调制晶体管栅极或基极的偏置,来实现开关稳压电源输出晶体管或晶体管导通时间的改变,这种方式能使电源的输出电压在工作条件变化时保持恒定,是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术,广泛应用在从测量、通信到功率控制与变换的许多领域中。

关于 PWM 的控制方法,采样控制理论中有一个重要结论:冲量相等而形状不同的窄脉冲加在具有惯性的环节上时,其效果基本相同,如图 5.49 所示。

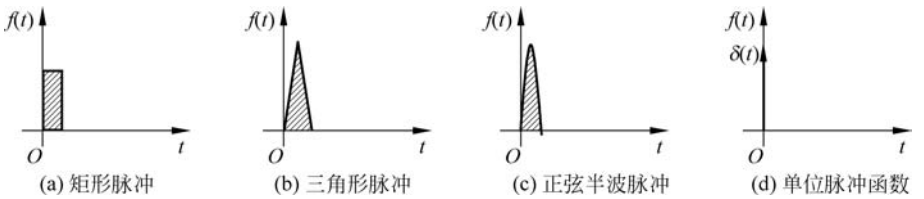


图 5.49 采样原理

PWM 控制技术就是以该结论为理论基础,对半导体开关器件的导通和关断进行控制,使输出端得到一系列幅值相等而宽度不相等的脉冲,用这些脉冲来代替正弦波或其他所需要的波形。按一定的规则对各脉冲的宽度进行调制,既可改变逆变电路输出电压的大小,也可改变输出频率,如图 5.50 所示。

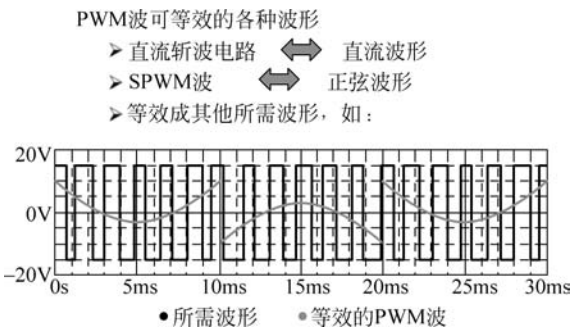


图 5.50 等效 PWM 波

TMS320F28069 有 19 路脉宽调制输出引脚(这些引脚需要软件进行配置,类似于前面的普通 GPIO 的配置),这 19 路脉宽调制输出引脚包括 16 路增强型 PWM(ePWM)和 3 路普通 PWM(ECap 模块复用),16 路增强型 PWM 中又有 8 路可以配置为高分辨率的 PWM(HRPWM)。8 组 PWM 模块,每一组又有 2 路 PWM,分别是 PWMA 和 PWMB。

一个 ePWM 模块包括 Time-base(TB) module(时基模块)、Counter-compare(CC) module(计数器比较模块)、Action-qualifier(AQ) module(比较方式预设模块)、Dead-band(DB) module(死区模块)、PWM-chopper(PC) module(斩波模块)、Event-trigger(ET) module(事件触发模块)、Trip-zone(TZ) module(行程区模块)共 7 个模块。关于各个模块的介绍请参考刘杰编写的《Simulink 建模基础及 C2000 DSP 代码自动生成》。

本次应用案例利用的是 EPWM1 模块,其原理图如图 5.51 所示。

图 5.51 EPWM1 模块接口

以下通过 3 个例子对 PWM 的功能进行说明。

5.4.1 ePWM_单路输出

1. 方法一

ePWM 模块输入时钟,直接由系统时钟而来,使用的时候可以根据用户需要进行分频。在 ePWM 的设置界面的 General 选项卡中,配置为 ePWM1 模式,对 PWM 模块进行初始化配置,时钟周期单元选择 Clock cycles,计数模式选择 Up-Down 计数,如图 5.52 所示。

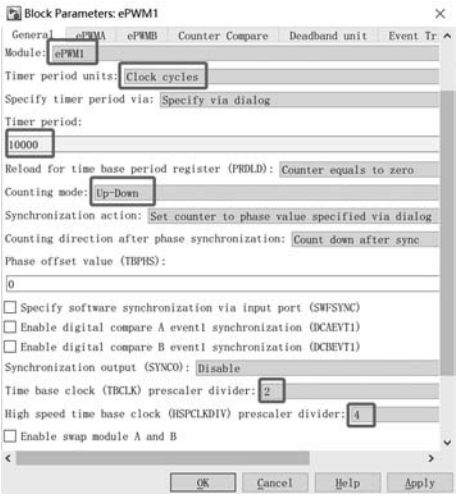


图 5.52 ePWM 模块配置

TBCLK=SYSCLKOUT/(HSPCLKDIV * CLKDIV),本次实验系统时钟 SYSCLKOUT 设置为 80MHz,取 HSPCLKDIV 选择 4 分频,TBCLK 分频选择 2 分频,所以 ePWM1 的时基时钟为 TBCLK=80/(2 * 4)=10MHz,设置周期寄存器值为 10000,因此 ePWM1 的输出频率为 10MHz/10000=1kHz。

在 ePWMA 选项卡配置使能 ePWMA,当计数等于 0 的时候不动作,等于比较寄存器 CAU 的值的时候复位,等于比较寄存器 CAD 的值的时候置高,其他时间不动作。ePWMB、死区时间、事件触发以及错误联防均不配置,如图 5.53 所示。

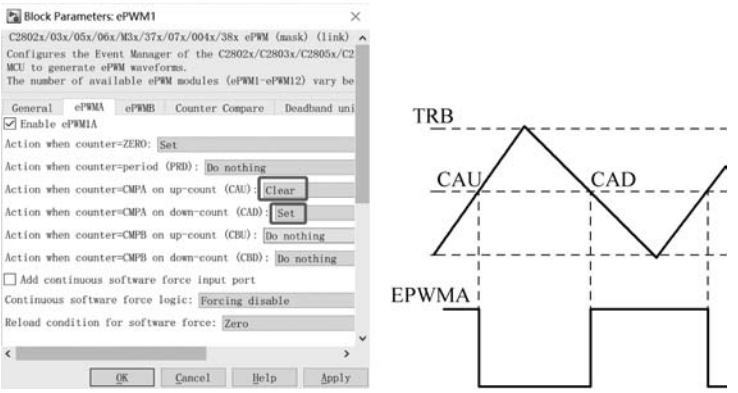


图 5.53 ePWMA 计数模式配置

对 Counter Compare 的配置:将 CMPA units 设置为 Percentage(百分比);比较寄存器 CMPA 设置成 Specify via dialog 内部指定为 50,即占空比为 50%,如图 5.54 所示。

通过以上配置,可以得到模型如图 5.55 所示。

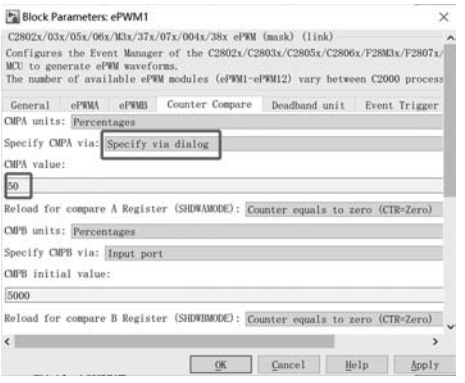


图 5.54 占空比配置

图 5.55 ePWM1 模块

2. 方法二

在 General 选项卡中,配置为 ePWM1 模式,对 PWM 模块进行初始化配置,时钟周期单元选择 Seconds,计数模式选择 Up-Down 计数,关闭同步功能,分频系数 TBCLK 和

HSPCLKDIV 均为 1(不对系统时钟分频),如图 5.56 所示。

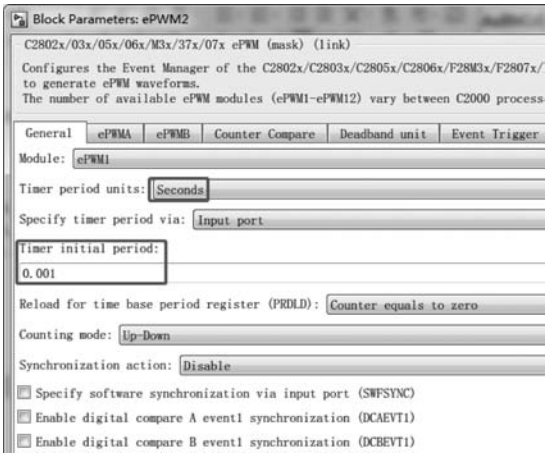


图 5.56 PWM 周期配置

对 ePWMA 配置同方法一,这里配置 CMPA units 设置为 Percentage(百分比);比较寄存器 CMPA 设置成 Input port(由外部输入),CMPA 初始值设为 0,如图 5.57 所示。

通过以上配置,可以得到模型如图 5.58 所示。

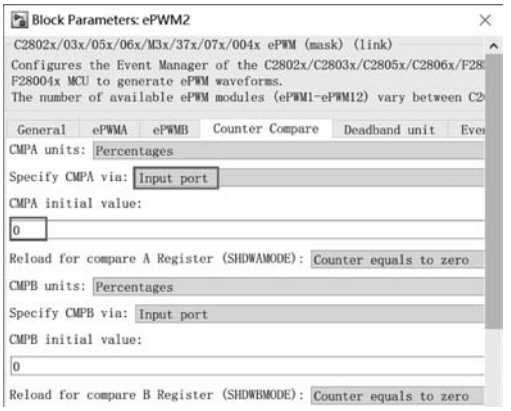


图 5.57 占空比配置

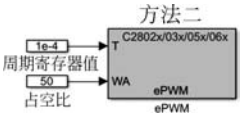


图 5.58 ePWM 模块配置

在上述配置完成,并完成模型的搭建后,将模型编译下载到主控板,然后在 CCS 中打开模型生成的工程文件,先单击“编译”按钮,再单击“调试”按钮,然后将图 5.59 中的两个变量添加到 Expressions 中,单击“运行”按钮,可以看到 TBPRD、CMPA 的值,如图 5.59 所示。可以将主控板的 PWM1A 引脚连接到示波器的信号线,主控板和示波器记得要共地,便能观测到输出的 PWM 的频率是否为 1kHz,占空比是否为 50%。注意,要单击 Expressions 的实时刷新按钮,否则数据不会变化。

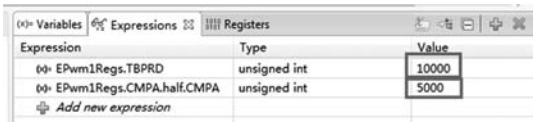


图 5.59　变量观察

5.4.2　ePWM_双路互补输出

对 General 选项卡和 ePWMA 选项卡的配置同 5.4.1 节中的方法一，因为要输出两路互补的 PWM，所以这里使能 ePWM1B 配置与 ePWM1A 相反，如图 5.60 所示。

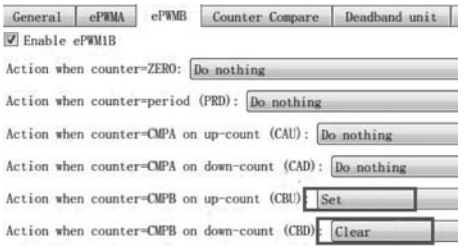


图 5.60　ePWMB 计数模式配置

对 Counter Compare 的配置：将 CMPA units 设置为 Clock cycles(时钟周期)；比较寄存器 CMPA 设置成 Input port 输入端口，初始值设为 0；对 CMPB 的操作一样，如图 5.61 所示。

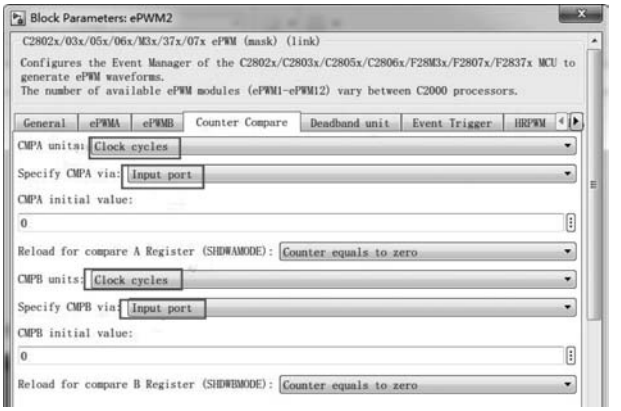


图 5.61　ePWM 周期配置

搭建的模型如图 5.62 所示。

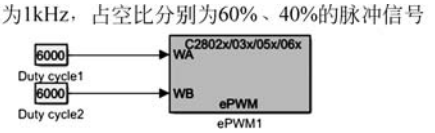


图 5.62　两路占空比不同的 PWM

在上述配置完成，并完成模型的搭建后，将模型编译下载到主控板，然后在 CCS 中打开模型生成的工程文件，先单击“编译”按钮，再单击“调试”按钮，然后将图 5.63 中的 3 个变量添加到 Expressions 中，单击“运行”按钮，可以看到 TBPRD、CMPA、CMPB 的值，如图 5.63 所示。可以将主控板的 PWM1A 引脚和 PWM1B 引脚分别连接到示波器的信号线，主控板和示波器记得要共地，便能观测到两路 PWM 的频率是否都为 1kHz，占空比是否分别为 60%和 40%，波形是否互补。注意，要单击图 5.63 右上方的实时刷新按钮，否则数据不会变化。

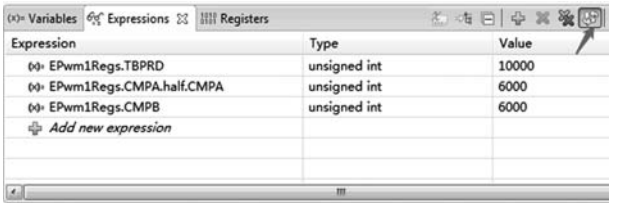


图 5.63 变量观察

5.4.3 ePWM_中断

PWM 中断配置在 5.4.2 节的 PWM 实验的基础上改动。在前面的配置不变的情况下，在 Event Trigger 选项卡中，使能 ePWM 中断，并配置为在第一个事件发生时就进入中断，其他设置默认。配置如图 5.64 所示。

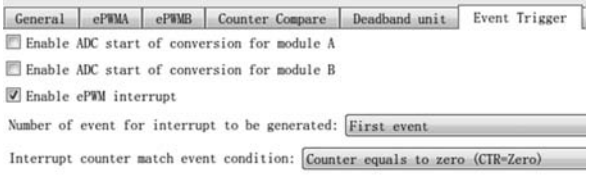


图 5.64 ePWM 中断配置

在上述配置完成后，调用 C28x Hardware Interrupt 模块，将 PWM1 中断的 CPU 级中断号 3 以及 PIE 级中断号 1 写入，其余配置默认，如图 5.65 所示。

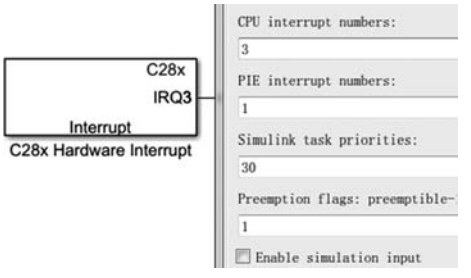


图 5.65 中断模块配置

中断函数中,使用 StateFlow 搭建了一个简单的逻辑功能模块,使 LED 灯每秒翻转一次。其中计数器计数最大值为 1000,如图 5.66 所示。

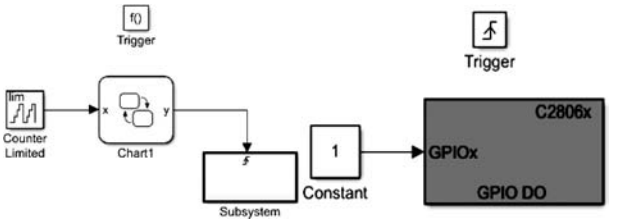


图 5.66 中断函数

该模型的系统时钟为 80MHz,定步长为 0.5s。完成上面的配置之后,将生成的代码下载到开发板上,可以看到,在 PWM 波产生的同时,LED2 灯在以 1Hz 的频率闪烁,LED1 灯以每 0.5s 的频率闪烁,如图 5.67 所示。

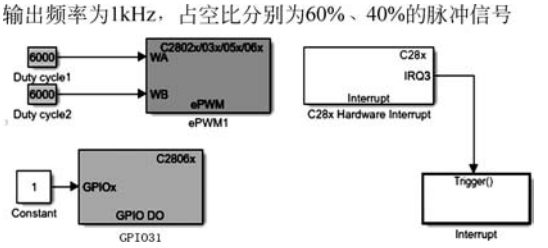


图 5.67 ePWM 中断实现


视频讲解

5.5 eCAP

5.5.1 eCAP 基本介绍

TMS320F28069 的捕获单元模块能够捕获外部输入引脚的逻辑状态(电平的高或低、电平翻转时的上升沿或下降沿),并利用内部定时器对外部事件或者引脚状态变化进行处理。控制器给每个捕获单元模块都分配了一个捕获引脚,在捕获引脚上输入待测波形,捕获模块就会捕获到指定捕获的逻辑状态,所以捕获单元可以用于测量脉冲周期以及脉冲的宽度。TMS320F28069 上面有 4 路增强型捕捉模块 eCAP,CAP 模块是应用定时器来实现事件捕获功能的,主要应用在速度测量、脉冲序列周期测量等方面。

eCAP 模块包括以下资源。

- 可分配的输入引脚。
- 32b 时间基准(计数器)。
- 4 个 32b 时间窗捕获控制寄存器。
- 独立的边缘极性选择。

- 输入信号分频(2～62)。
- 4 个捕获事件均可引起中断。

1. eCAP 模块的组成

eCAP 模块可以设置为捕捉模式或者是 APWM 模式,一般而言,前者比较常用,因此在这里只对第一种情况进行分析介绍。在捕捉模式下,一般可以将 eCAP 模块分为以下几部分:事件分频、边沿极性选择与验证、中断控制。

1) 事件分频

输入事件信号可通过分频器分频处理(分频系数 2～62),或直接跳过分频器。这个功能通常适用输入事件信号频率很高的情况。

2) 边沿极性选择与验证

4 个独立的边沿极性(上升沿/下降沿)选择通道;Modulo4 序列发生器对 Eachedge(共 4 路)进行事件验证;CAPx 通过 Mod4 对事件边沿计数,CAPx 寄存器在下降沿时被装载。32b 计数器(TSCTR)此计数器为捕捉提供时钟基准,而时钟的计数则是基于系统时钟的。当此计数器计数超过范围时,则会产生相应的溢出标志,若溢出中断使能,则产生中断。此计数器在计算事件周期时非常有效,关于 CAP 的详细资料请参阅数据手册。

3) 中断控制

中断能够被捕获事件(CEVT1～CEVT4、CTROVF)触发,计数器(TSCTR)计数溢出同样会产生中断。事件单独地被极性选择部分以及序列验证部分审核。这些事件中的一个被选择用来作为中断源送入 PIE。

设置 CAP 中断的过程可表述如下。

(1) 关闭全局中断。

(2) 停止 eCAP 计数。

(3) 关闭 eCAP 的中断。

(4) 设置外设寄存器。

(5) 清除 eCAP 中断标志位。

(6) 使能 eCAP 中断。

(7) 开启 eCAP 计数器。

(8) 使能全局中断。

2. 加深理解 eCAP 模块原理

配置好 eCAP 模块的引脚后,外部事件由引脚输入,首先通过模块的分频部分,分频系数为 2～62,也可以选择跳过分频部分,此功能主要是针对输入事件信号频率很高的情况。经过分频后的信号(通常频率会降低),送至边沿及序列审核部分,边沿审核即设置为上升沿或下降沿有效,序列审核则是指分配当前对哪个寄存器(CAP1～CAP4)作用的问题,之后就是中断执行控制部分。

5.5.2 eCAP 捕获 PWM 脉冲

在 5.4 节中知道了如何配置 TMS320F28069 的 ePWM 模块使其输出 PWM 波形，而本实验则是通过利用捕获模块功能来测量配置的 PWM 波的占空比是否正确。实验时，只需要将 ECAP1 引脚用杜邦线先后与 EPWM1A 引脚和 EPWM1B 连接起来，即可完成实验。其原理图如图 5.68 所示。

第一步，在 Solver 中设置定步长为 0.5s，在 Hardware Implementation 中配置系统时钟为 90MHz，LSPCLK 低速时钟外设 4 分频，eCAP 的 ECAP1 pin assignment 引脚选择 GPIO5，如图 5.69 所示。

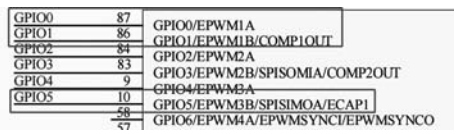


图 5.68 eCAP 原理图

图 5.69 eCAP 模块配置

第二步，初始化好 10 个全局变量，如图 5.70 所示。用来实现对一方波信号的频率、周期、占空比的测量。

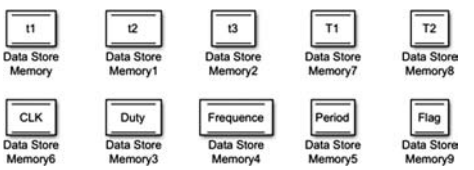


图 5.70 变量初始化

(1) t1 表示第一个捕捉事件发生时计数器的值。

(2) t2 表示第二个捕捉事件发生时计数器的值。

(3) t3 表示第三个捕捉事件发生时计数器的值。

(4) T1 表示 t2－t1 的差，也就是测试方波的高电平时间对应的计数值。

(5) T2 表示 t3－t1 的差，也就是测试方波的整个周期对应的计数值。

(6) CLK 表示系统时钟周期，Duty 表示实际方波的占空比，Frequence 表示方波的频率，Period 表示方波的周期，Flag 表示检测完成标志位。

第三步，生成两路频率为 10kHz，一个是占空比为 50%的 PWM，另一个是占空比为 60%的 PWM，如图 5.71 所示。

第四步，设置 eCAP 模块，中断中配置如图 5.72 和图 5.73 所示，使用的是 eCAP1 模块，配置为连续控制模式，并在第三次事件之后停止计数并重置，第一次事件的触发极性为上升沿，第二次事件的触发极性为下降沿，第三次触发极性为上升沿，计数时间的数据类型为无符号的 32 位整型。中断配置为捕捉到第三次事件之后触发中断。在 eCAP 输出接上 demux 模块，并将输出的值赋予 t1、t2、t3。

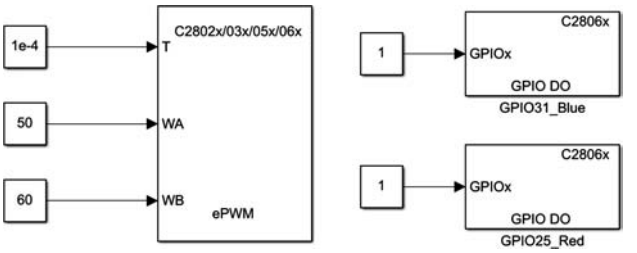
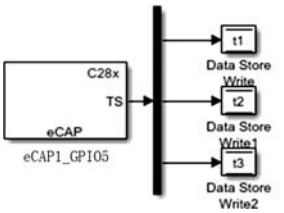


图 5.71　ePWM 模块配置



t1表示第一个捕捉事件发生时计数器的值
t2表示第二个捕捉事件发生时计数器的值
t3表示第三个捕捉事件发生时计数器的值

图 5.72　eCAP 模块搭建

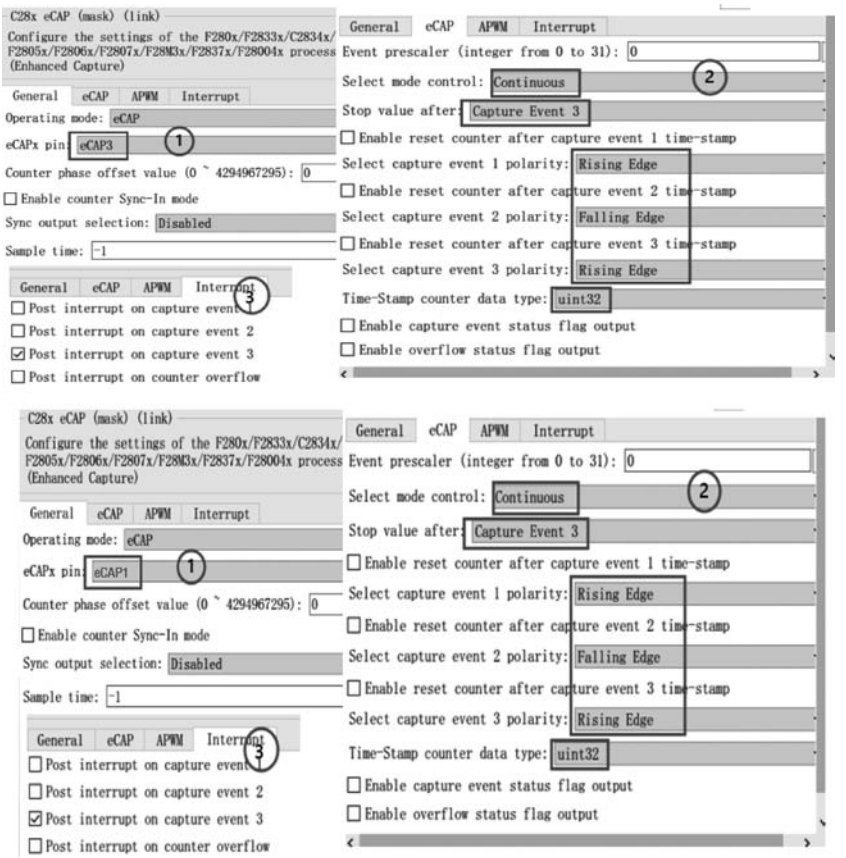
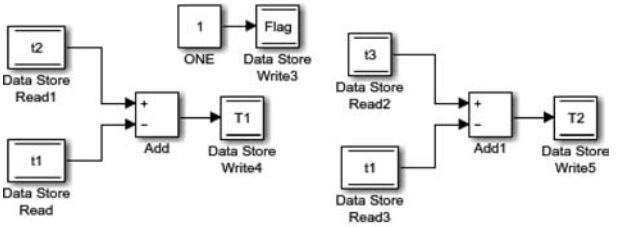


图 5.73　eCAP 模块配置

第五步,计算 T1、T2 并对 Flag 置位,表示一次检测完毕,如图 5.74 所示。



注:Flag为标志位,作为while子系统的触发信号

图 5.74 计算 T1、T2

第六步,封装子系统,配置硬件中断,根据 eCAP1 的中断向量号(它属于 CPU 中断的第四组下的第 1 个 PIE 中断),配置 C28x 硬件中断模块,如图 5.75 所示。

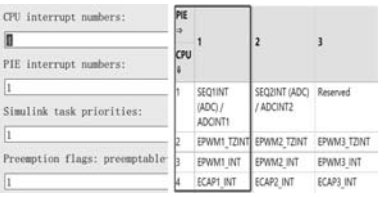


图 5.75 eCAP 中断配置

第七步,标志位触发,在 Whlie(1)中触发计算模块,对输入方波的频率、周期、占空比进行计算,如图 5.76 所示。其中,CLK = 1/90000000。如图 5.77 所示为 eCAP 的 Simulink 模型图。

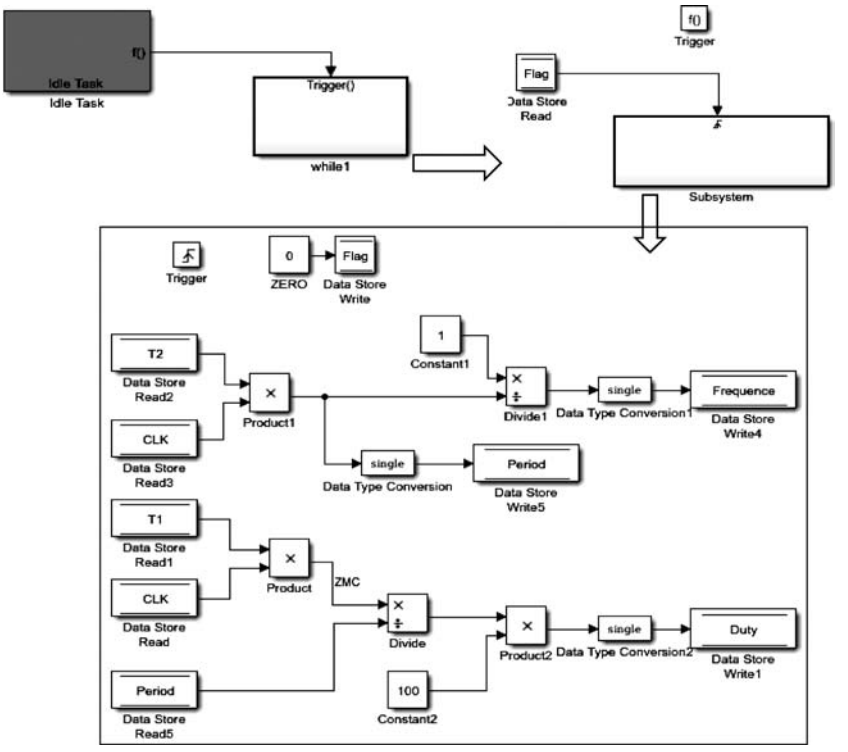


图 5.76 PWM 参数计算

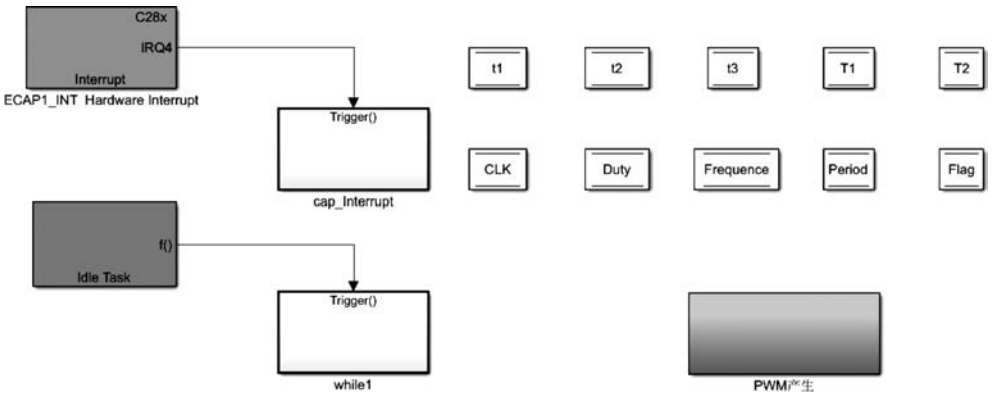


图 5.77 eCAP 实验

这里需要注意一点，请在 Diagonostics 选项卡中的 Data Validity 选项中，将 Multitask data store 配置为 none，否则会出现如图 5.78 所示的错误。

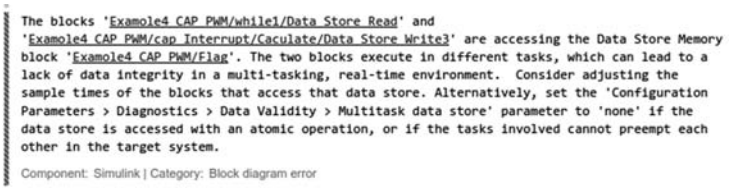
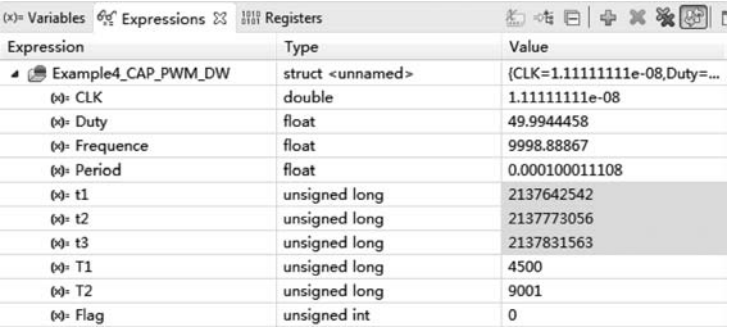


图 5.78 错误诊断

在上述配置完成，并完成模型的搭建后，将模型编译下载到主控板，然后在 CCS 中打开模型生成的工程文件，先单击“编译”按钮，再单击“调试”按钮，然后将图 5.79 和图 5.80 中的 10 个变量添加到 Expressions 中，在单击“运行”按钮前，先将 eCAP3 引脚用杜邦线与 EPWM1A 引脚连接起来。然后再单击“运行”按钮可以看到各个变量的值。要想看 EPWM1B 的数值，ECAP1 引脚用杜邦线与 EPWM1B 引脚连接起来。这里有一点点误差，是芯片自身问题。图 5.79 为 EPWM1A 数据监测图，图 5.80 为 EPWM1B 数据监测图。



Expression	Type	Value
Example4_CAP_PWM_DW	struct <unnamed>	{CLK=1.11111111e-08,Duty=...
CLK	double	1.11111111e-08
Duty	float	49.9944458
Frequence	float	9998.88867
Period	float	0.000100011108
t1	unsigned long	2137642542
t2	unsigned long	2137773056
t3	unsigned long	2137831563
T1	unsigned long	4500
T2	unsigned long	9001
Flag	unsigned int	0

图 5.79 EPWM1A 数据监测

Variables | Expressions | Registers

Expression	Type	Value
Example4_CAP_PWM_DW	struct <unnamed>	{CLK=1.11111111e-08,Duty=...
CLK	double	1.11111111e-08
Duty	float	60.0044479
Frequence	float	9998.88867
Period	float	0.000100011108
t1	unsigned long	3463601723
t2	unsigned long	3463679132
t3	unsigned long	3463736738
T1	unsigned long	5401
T2	unsigned long	9001
Flag	unsigned int	0

图 5.80 EPWM1B 数据监测


视频讲解

5.6 SCI 串行通信

5.6.1 SCI 通信基本原理

SCI(Serial Communication Interface)意为“串行通信接口”,是相对于并行通信来说的,是串行通信技术的一种总称,最早由 Motorola 公司提出。它是一种通用异步通信接口,与 MCS-51 的异步通信功能基本相同。

SCI 模块用于串行通信,如 RS422、RS485、RS232,在 SCI 中,通信协议体现在 SCI 的数据格式上。通常将 SCI 的数据格式称为可编程的数据格式,原因就是它可以通过 SCI 的通信控制寄存器 SCICCR 来进行设置,规定通信过程中所使用的数据格式,如图 5.81 所示。

图 5.81 典型的 SCI 数据帧格式

TMS320F28069 上面有两个 UART 口:SCIA、SCIB。本实验针对 SCIB。SCI 的原理以及特点如下所述。

(1) 外部引脚。

发送引脚:SCITXD,接收引脚:SCIRXD。

(2) 波特率可编程。

当 BRR≠0 时:波特率=LSRCLK÷((BRR+1)×8)。

当 BRR=0 时:波特率=LSPCLK÷16。

(3) 数据格式。

1 个开始位,1~8 个数据位,奇校验/偶检验/无校验可选,1 或 2 个停止位。

(4) 4 个错误检测标志:校验、溢出、帧、断点检测。

(5) 全双工和半双工模式可以缓冲接收和发送。

(6) 串口数据发送和接收过程可以通过中断方式或查询方式。

本实验选取 TMS320F28069 的 SCIB 模块进行实验,分别设置 GPIO15 和 GPIO58 作为 SCIB 的发送和接收功能引脚,并配置为波特率 9600b/s,8 位数据位,1 位停止位,无奇偶校验,原理图如图 5.82 所示。

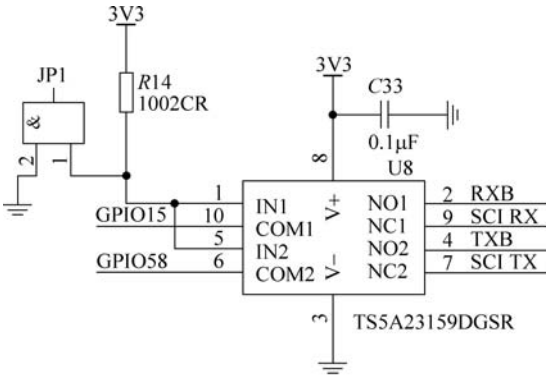


图 5.82 SCIB 原理图

5.6.2 SCI 收发数据

在 Solver 中设置定步长为 0.1s,在 Hardware Implementation 中配置系统时钟为 90MHz,LSPCLK 低速时钟外设 4 分频,SCI_B 的字符长度为 8 位,波特率设为 115000b/s,Tx 的引脚设为 GPIO58,Rx 的引脚设为 GPIO15,如图 5.83 所示。

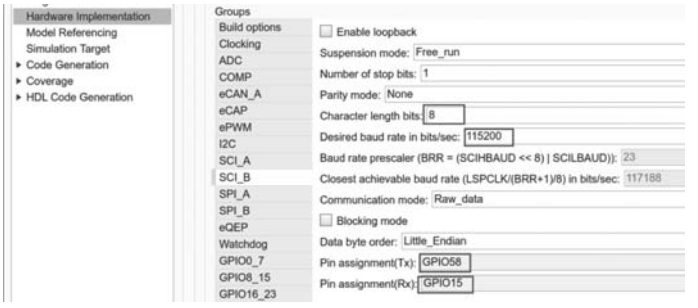


图 5.83 SCIB 模块配置

设置 SCI Receive 的 SCI module 为 B,其他的参数默认,此模块负责接收串口助手发送过来的数据; SCI Transmit 的 SCI module 为 B,如图 5.84 所示,其他参数默认,此模块负责发送数据给串口助手。

搭建整个模型如图 5.85 所示,控制流程为: SCI 接收到数据与 53 比较,如果等于 53 则输出 1,1 大于 0 则输出 Counter Limited 的数(Upper limit 设定为 10,当累加到 10 变归 0 重新计数),否则一直输出 2。

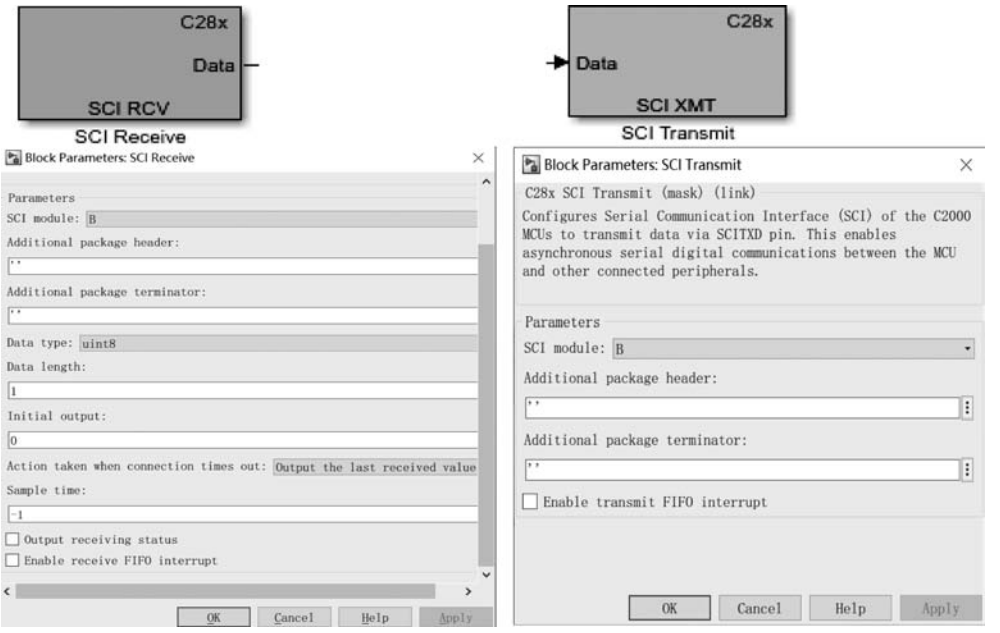


图 5.84 SCI 收发配置

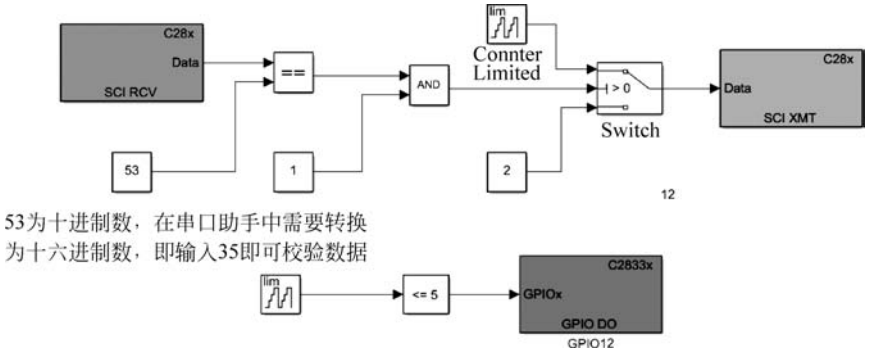


图 5.85 整个模型搭建

在上述配置完成,并完成模型的搭建后,将模型编译下载到主控板。

将套件中附带的 USB 线一端与计算机连接,并将另一端插到智能平衡移动机器人套件中主控制板的 USB 插口处。当用 USB 直接连接计算机时,要想看到系统识别串口,需要打开设备管理器,选择 TI XDS100 ChannelB,在右键快捷菜单中选择“属性”命令,在“高级”选项卡中选中“加载 VCP”选项,拔掉 USB 连接线,再次插入时将显示系统已经识别到 COM 口,可以打开串口调试工具进行下一步实验,如图 5.86 所示。

接下来打开串口猎人软件。可以看见几个主要的部分,如图 5.87 所示。

选择对应的端口号 COM9,波特率为 115200b/s,在没有发送数据之前,数据接收界面一直显示 2,说明搭建的模型是正确的,如图 5.88 所示。

图 5.86　串口配置

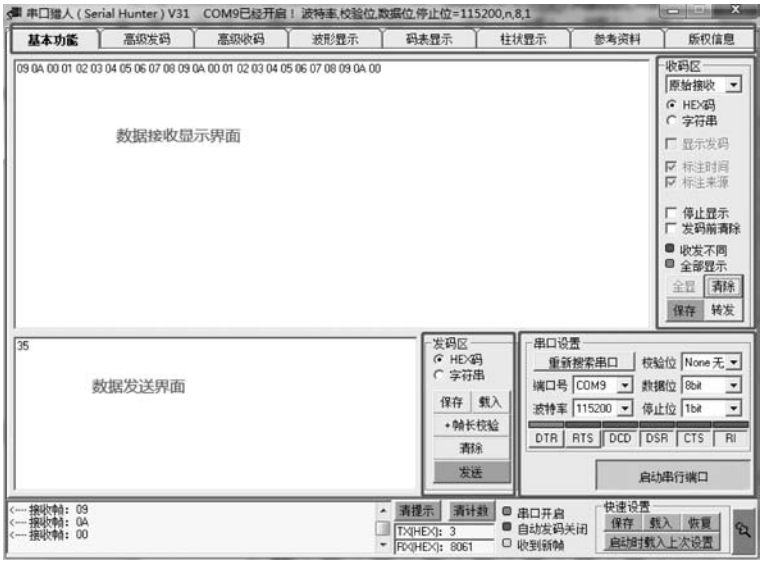


图 5.87　串口软件配置

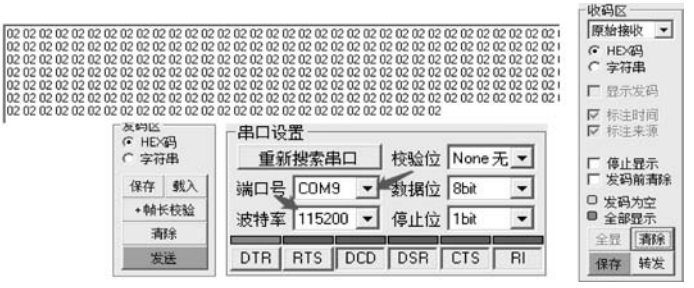


图 5.88　串口软件显示

在左下角的方框中输入 35(十六进制),则数据接收显示界面会一直显示 0～10 的数,如图 5.89 所示。

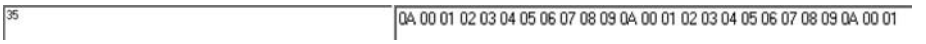


图 5.89　串口软件测试

在高级收码工具栏下方的通道设置的来源选择提取每一帧,然后单击“启动高级收码”按钮,便可以在历史数据看见有 0～10 的十进制数显示,如图 5.90 所示。

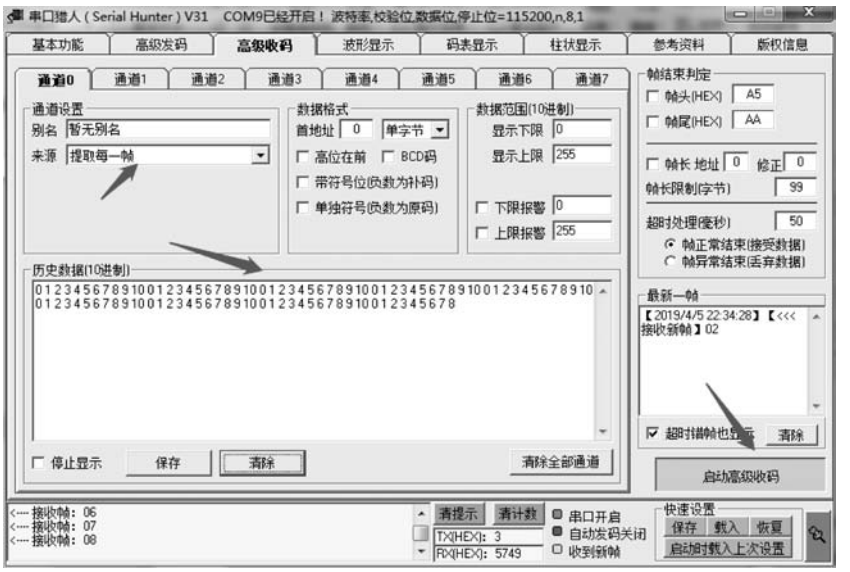


图 5.90　串口软件测试

在波形显示界面将 Y 轴倍率设为 10/格,周期为 0.1/格,可以看见一个 0～10 的阶梯状图形,如图 5.91 所示。

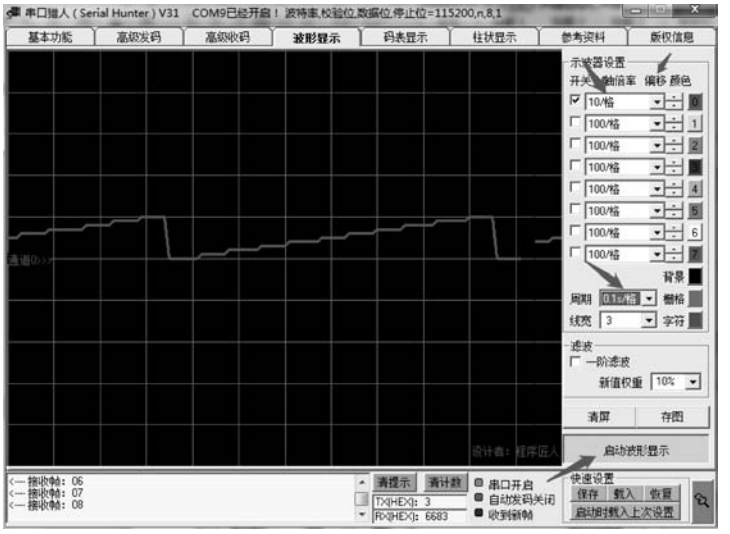


图 5.91　串口软件波形显示

5.7 SPI 串行通信

5.7.1 SPI 概述

SPI 即 Serial Peripheral Interface 是高速同步串行输入/输出端口，SPI 最早是由 Freescale(原属 Motorola)公司在其 MC68HCxx 系列处理器上定义的一种高速同步串行接口。SPI 目前被广泛用于外部移位寄存器、D/A、A/D、串行 EEPROM、LED 显示驱动器等外部芯片的扩展。与前面介绍的 SCI 的最大区别是，SPI 是同步串行接口。SPI 总线包括 1 根串行同步时钟信号线(SCI 不需要)以及 2 根数据线，实际总线接口一般使用 4 根线，即 SPI 四线制：串行时钟线、主机输入/从机输出数据线、主机输出/从机输入数据线和低电平有效的从机片选线。有的 SPI 接口带有中断信号线，有的 SPI 接口没有主机输出/从机输入线。在 TMS320F28069 中使用的是前面介绍的 SPI 四线制。

SPI 接口的通信原理简单，以主从方式进行工作。在这种模式中，必须要有一个主设备，可以有多个从设备。通过片选信号来控制通信从机，SPI 时钟引脚提供串行通信同步时钟，数据通过从入主出引脚输出，从出主入引脚输入。通过波特率寄存器设置数据速率。SPI 向输入数据寄存器或发送缓冲器写入数据时就启动了从入主出引脚上的数据发送，先发送最高位。同时，接收数据通过从出主入引脚移入数据寄存器最低位。选定数量位发送结束，则整个数据发送完毕。收到的数据传送到 SPI 接收寄存器，右对齐供 CPU 读取。SPI 的通信链接如图 5.92 所示。

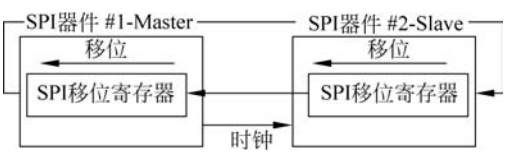


图 5.92 SPI 通信原理

TMS320F28069 的 SPI 接口具有以下特点。

1. 4 个外部引脚

- SPISOMI：SPI 从输出/主输入引脚。
- SPISIMO：SPI 从输入/主输出引脚。
- SPISTE：SPI 从发送使能引脚。
- SPICLK：SPI 串行时钟引脚。

2. 2 种工作方式：主和从工作方式

波特率：125 种可编程波特率。

当 SPIBRR＝3～127 时，波特率＝$\dfrac{\text{LSPCLK}}{(\text{SPIBRR}+1)}$。

当 SPIBRR＝0,1,2 时，波特率＝$\dfrac{\text{LSPCLK}}{4}$。

数据字长：可编程的 1～16 个数据长度。

3. 4 种计时机制(由时钟极性和时钟相应控制)

(1) 无相位延时的下降沿：SPICLK 为高电平有效。在 SPICLK 信号的下降沿发送数据，在 SPICLK 信号的上升沿接收数据。

(2) 有相位延时的下降沿：SPICLK 为高电平有效。在 SPICLK 信号的下降沿之前的半个周期发送数据，在 SPICLK 信号的下降沿接收数据。

(3) 无相位延迟的上升沿：SPICLK 为低电平有效。在 SPICLK 信号的上升沿发送数据，在 SPICLK 信号的下降沿接收数据。

(4) 有相位延迟的上升沿：SPICLK 为低电平有效。在 SPICLK 信号的下降沿之前的半个周期发送数据，而在 SPICLK 信号的上升沿接收数据。

- 接收和发送可同时操作(可以通过软件屏蔽发送功能)。
- 通过中断或查询方式实现发送和接收操作。
- 9 个 SPI 模块控制寄存器：位于控制寄存器内，帧开始地址 7040H。

注意：这个模块中的所有寄存器是被连接至外设帧 2 的 16 位寄存器。当一个寄存器被访问时，低字节(7～0)和高字节(15～8)内的寄存器数据被读作零。对高字节的写入没有效果。

4. 增强型特性

- 4 级发送/接收 FIFO。
- 经延迟的发射控制。
- 支持双向 3 线 SPI 模式。
- 借助 SPISTE 翻转的音频数据接收支持。

5.7.2 SPI 控制 PWM 占空比

第一步，在 Solver 中设置定步长为 0.05s，在 Hardware Implementation 中配置系统时钟为 90MHz，LSPCLK 低速时钟外设 4 分频，主要设置 SPI_A 的 Enable loopback，将 SIMO pin assignment、SOMI pin assignment、CLK pin assignment、STE pin assignment 这 4 个引脚配置为如图 5.93 所示的 I/O 口。

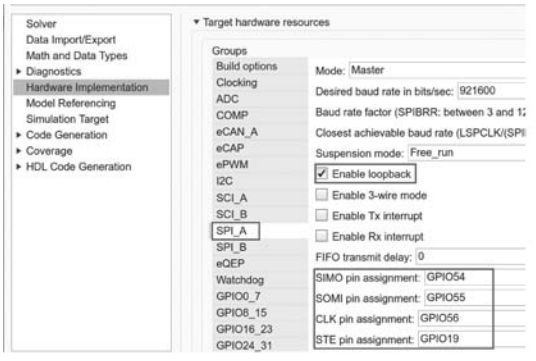


图 5.93 SPIA 配置

第二步，搭建占空比选择子系统，如图 5.94 所示，关于 MASK 子系统的介绍可以查看网页：https://ww2.mathworks.cn/help/simulink/gui/mask-editor-overview.html?

searchHighlight＝Mask%20Editor%20Overview&s_tid＝doc_srchtitle。

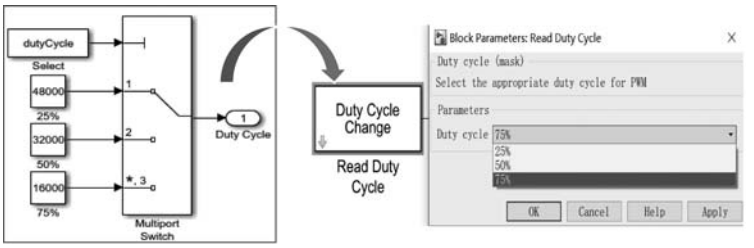


图 5.94　占空比配置

第三步，选择 SPI Transmit 和 SPI Receive 的模块为 SPI_A，如图 5.95 所示，其他参数默认。

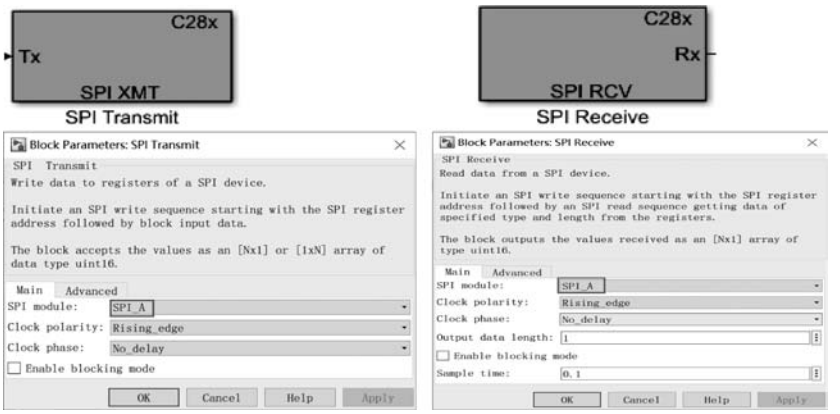


图 5.95　收发配置

第四步，配置 PWM 输出，定时器计数次数为 64000，计数方式为向上计数 UP，不分频。

使能 ePWM1A，当计数值等于定时器周期值 PRD 时置低，当计数值等于向上计数的 CAU 时置高，其他计数为 Do nothing(不动作)。

使能 ePWM1B，当计数值等于定时器周期值 PRD 时置高，当计数值等于向上计数的 CBU 时置低，其他计数为 Do nothing(不动作)。

比较计数，选择 Clock cycles，CMPA initial value、CMPB initial value 的值由外部输入，初值为 0，如图 5.96 所示。

第五步，在上述配置完成，并完成模型的搭建后，如图 5.97 所示。选择占空比为 75%，将模型编译下载到主控板。

第六步，然后在 CCS 中打开模型生成的工程文件，先单击“编译”按钮，再单击“调试”按钮，然后将图 5.98 中的两个变量添加到 Expressions 中，单击“运行”按钮，可以看到 TBPRD、CMPA、CMPB 的值。可以将主控板的 PWM1A 引脚和 PWM1B 引脚分别连接到示波器的信号线，主控板和示波器记得要共地，便能观测到两路 PWM 的频率是否都

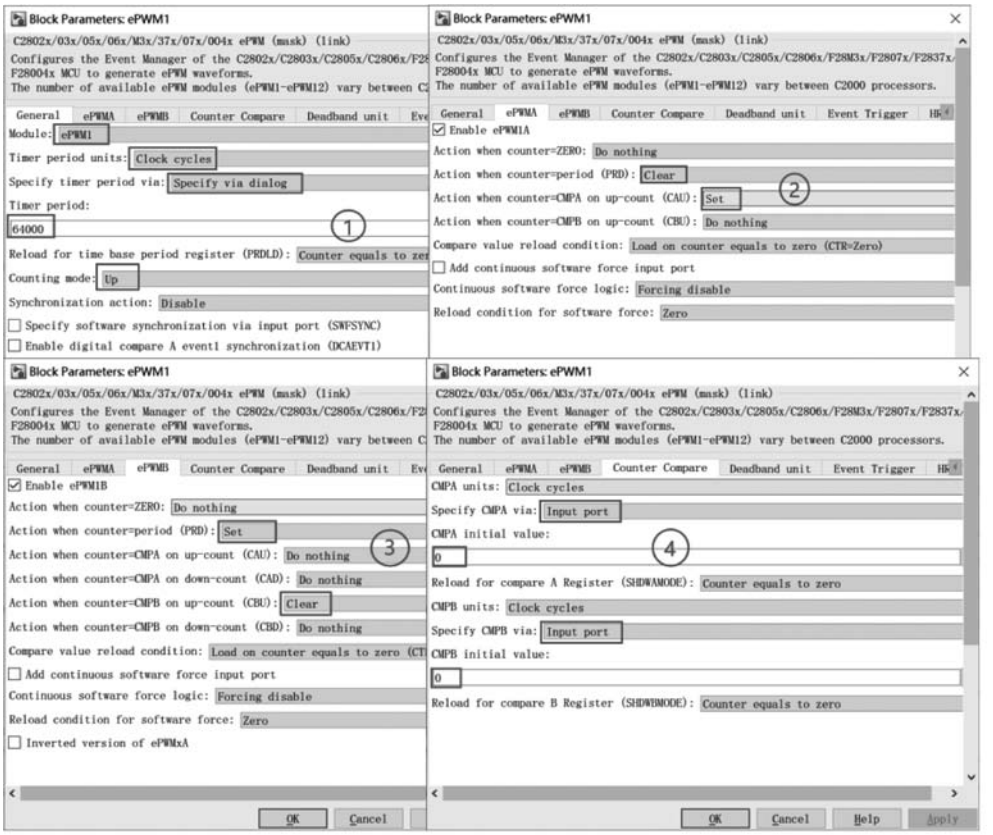


图 5.96 ePWM 配置

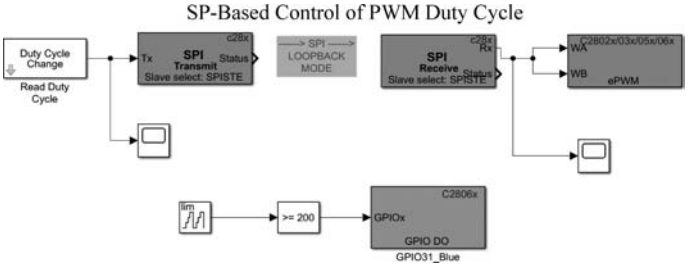


图 5.97 整个模型搭建

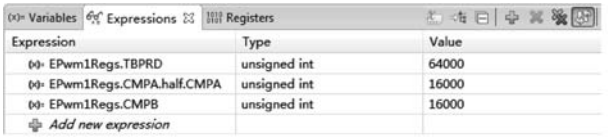


图 5.98 两路 PWM 的占空比分别为 75%,25%

为 1kHz，占空比是否分别为 75%和 25%，波形是否互补。注意，要单击右上方的实时刷新按钮，否则数据不会变化。

第七步，在 Simulink 中选择占空比为 50%，重新编译下载，然后在 CCS 中观察数据，如图 5.99 所示。

图 5.99 两路 PWM 的占空比都为 50%

第八步，在 Simulink 添加两个 Scope，如图 5.100 所示，选择 External 模式，主控板连接 USB 转 TTL 模块，选择对应的 COM 口，波特率为 115200b/s，选择占空比为 75%，单击“运行”按钮，可以观测 Scope1 的数据是否和 Scope 的数据一致，在运行状态下可以改变占空比的大小，查看 Scope1 的数据变化，如图 5.101 所示。

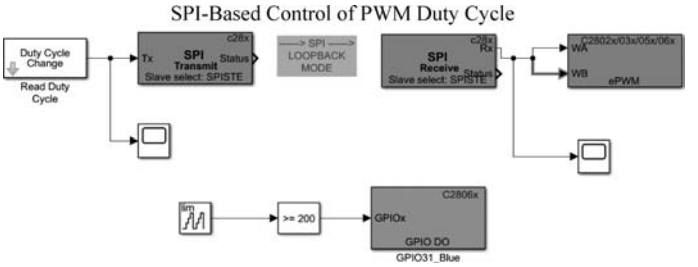


图 5.100 模型图

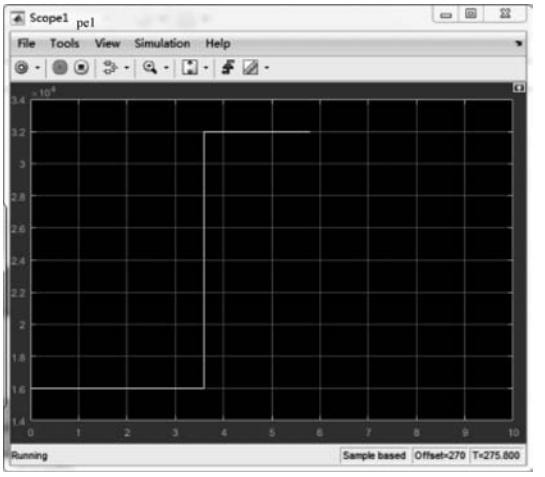


图 5.101 波形图

5.8 IdelTask 模块介绍

在常规的 MCU 编程中,Whlie(1)是几乎都会使用到的一个函数,它可以使一些对工作周期要求不高的外设对所要执行的功能进行循环扫描,例如,矩阵键盘或者 OLED 显示等功能。在 TI 提供的硬件支持包中,也很人性化地提供了这个功能模块——Idle Task。添加 Idle Task 模块后生成的代码如下。

```
while (runModel) {
    stopRequested = !( rtmGetErrorStatus(Ideltask_M) == (NULL));
    runModel = !(stopRequested);
    idletask_num1();
    idletask_num2();
  }
```

Idle Task 模块的配置界面如图 5.102 所示,分别设置 Task numbers 以及 Preemption flags。

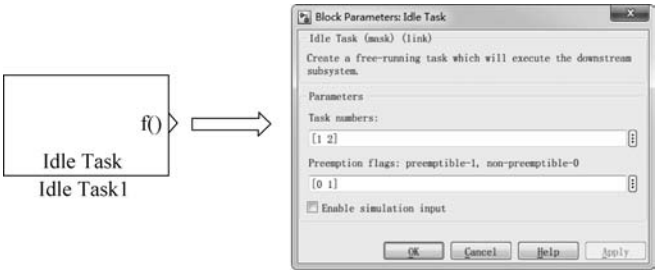


图 5.102 Idle Task 模块配置

这里的抢占式优先级不同于中断的抢占式优先级。它的功能是:当 Preemption flag 设置为 0 的时候,每次进入 Whlie 循环的开始,关掉全局中断,执行完毕后打开。当 Preemption flag 设置为 1 时,则不做上述处理,当中断来临时,优先执行中断。

下面为 Preemption flag 设置为的 0 时生成的代码,可以看到生成的代码中出现了开关中断的指令,而从 Preemption flag 设置为 1 时的生成代码可以看到,没有开关中断的指令。

```
Void idletask_num1(void)
{
  DINT;
  Idle_num1_task_fcn();
  EINT;
}
Void idletask_num2(void)
{
```

```
  Idle_num2_task_fcn();
}
Void enable_interrupts()
{
  EINT;
  ERTM;
  }
```

所以在使用的时候，可以根据自己的需要选择。搭建如图 5.103 所示模型，左边子系统是给 GPIO31 一个高电平，右边子系统是给 GPIO25 一个低电平，电平不翻转。编译下载可以观察主控板的 LED1 灯亮，LED2 灯灭。

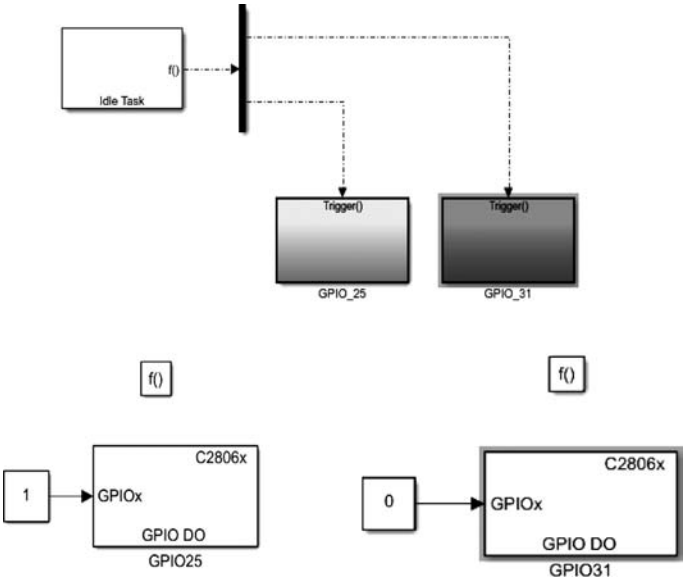


图 5.103　模型搭建

下面为该模型生成的代码，idle_num1_task_fcn()函数为图 5.103 中右边子系统生成的代码，idle_num2_task_fcn()函数为左边子系统生成的代码。

```
void idle_num1_task_fcn(void)
{
  /* Call the system: <Root>/GPIO_25 */
  {
    /* S-Function (idletask): '<Root>/Idle Task' */
    /* Output and update for function-call system: '<Root>/GPIO_25' */
    /* S-Function (c280xgpio_do): '<S2>/GPIO25' incorporates:
     * Constant: '<S2>/Constant'
     */
```

```
    {
      if (Ideltask_P.Constant_Value)
        GpioDataRegs.GPASET.bit.GPIO25 = 1;
      else
        GpioDataRegs.GPACLEAR.bit.GPIO25 = 1;
    }
    /* End of Outputs for S-Function (idletask): '<Root>/Idle Task' */
  }
}
/* Idle Task Block: '<Root>/Idle Task' */
void idle_num2_task_fcn(void)
{
  /* Call the system: <Root>/GPIO_31 */
  {
    /* S-Function (idletask): '<Root>/Idle Task' */
    /* Output and update for function-call system: '<Root>/GPIO_31' */
    /* S-Function (c280xgpio_do): '<S1>/GPIO31' incorporates:
     * Constant: '<S1>/Constant'
     */
    {
      if (Ideltask_P.Constant_Value_a)
        GpioDataRegs.GPASET.bit.GPIO31 = 1;
      else
        GpioDataRegs.GPACLEAR.bit.GPIO31 = 1;
    }
    /* End of Outputs for S-Function (idletask): '<Root>/Idle Task' */
  }
}
```

可以从 Idletask.h 头文件代码中看到当 Constant_Value 为 0,LED2 灯灭,Constant_Value_a 为 1,LED1 灯亮。

```
/* Parameters (default storage) */
struct P_Ideltask_T_ {
  real_T Constant_Value;                /* Expression: 0
                                         * Referenced by: '<S2>/Constant'
                                         */
  real_T Constant_Value_a;              /* Expression: 1
                                         * Referenced by: '<S1>/Constant'
                                         */
};
P_Ideltask_T Ideltask_P = {
0.0,                                    /* Expression: 0
                                         * Referenced by: '<S2>/Constant'
                                         */
```

```
    1.0                                /* Expression: 1
                                        * Referenced by: '<S1>/Constant'
                                        */
    }
```

5.9 WatchDog 模块介绍

意外难免会发生，部分意外发生的时候，系统程序会跑飞或进入死循环，系统需要有一定自恢复的功能，这就需要看门狗。意外有很多，对强电类控制电路来说，最让人头疼的就是琢磨不透又抓不着的 EMI 干扰，以及电源设计，对于软件而言有内存泄漏、程序健壮性等问题。看门狗(Watchdog timer)从本质上来说就是一个定时器电路，一般有一个输入和一个输出，其中的输入叫作喂狗，输出一般连接到另外一个部分的复位端，在这里就是 TMS320F28069 的复位端。CPU 工作正常时，按照设定的程序，每隔一段时间就输出一个信号到喂狗端，实际操作是给看门狗计数器清零，如果超过了一定时间没有信号到喂狗端进行喂狗，来做清零操作，一般就认为程序运行出了意外，不管意外类型是怎样的，这时候看门狗电路就会给出一个复位信号给 CPU 的复位端，使 CPU 强制复位，从而可能改变程序跑飞或死循环的状态。设计者必须了解看门狗的溢出时间以决定在适当的时候清看门狗。清看门狗也不能太过频繁，否则会造成资源浪费。在系统设计初以及调试的时候，不建议使用看门狗，因为系统设计初的时候意外的可能性太多，且有些意外是必须处理的，看门狗电路的复位信号很可能会引入更多的困扰。合理利用看门狗电路，可以检测软件和硬件运行的状态，进一步提高系统的可靠性。

在使用看门狗之前首先要对看门狗的内部计数器进行配置。打开模型配置界面，在 Watchdog 选项卡中，使能看门狗，再配置看门狗计数器的计时时间为 0.209715s 触发一次 Time out event(从配置中可以看到，看门狗的时钟直接来自 OSCCLK 也就是外部晶振的时钟 10MHz)，然后配置 Time Out event 为 Chip reset。具体配置如图 5.104 所示。

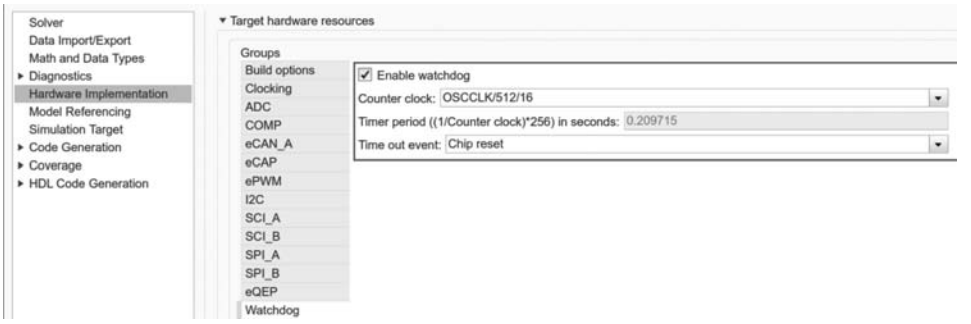


图 5.104 看门狗模块

将看门狗模块的复位源信号设置为 Timer0 中断的周期,在 Sample time 中填入-1,如图 5.105 所示。

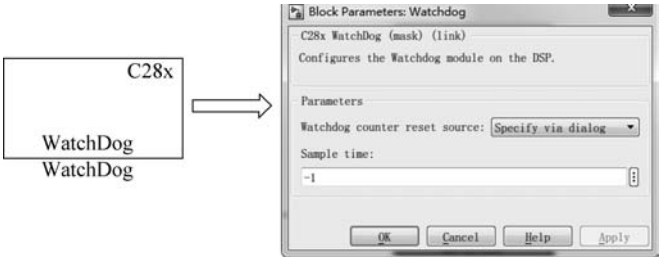


图 5.105 看门狗模块配置

同时在 Timer0 的中断中使两 LED 的电平在中断中翻转。模型如图 5.106 所示。

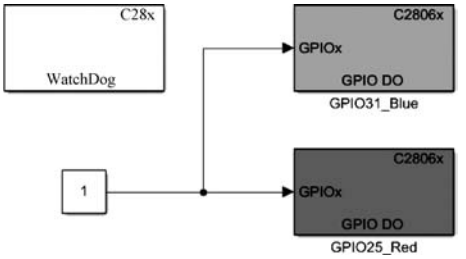


图 5.106 模型搭建

然后可以做两次对比性实验,一次实验的系统步长设置为 0.5s(大于看门狗的溢出时间),另一次设置为 0.1s(小于看门狗的溢出时间)。分别生成代码之后下载到开发板上观察实验现象。

从图 5.107 中的代码可以看到,每进行一次定时器中断,便将看门狗计数器复位,也就是所说的喂狗。

```
/* Model step function */
void WatchDog_step(void)
{
  /* S-Function (c280xgpio_do): '<Root>/GPIO25_Red' incorporates:
   *  Constant: '<Root>/Constant'
   */
  {
    GpioDataRegs.GPATOGGLE.bit.GPIO25 = (WatchDog_P.Constant_Value != 0);
  }

  /* S-Function (c280xgpio_do): '<Root>/GPIO31_Blue' incorporates:
   *  Constant: '<Root>/Constant'
   */
  {
    GpioDataRegs.GPATOGGLE.bit.GPIO31 = (WatchDog_P.Constant_Value != 0);
  }

  /* S-Function (c28xwatchdog): '<Root>/Watchdog' */
  {
    KickDog();
  }
}
```

图 5.107 自动生成代码

在对比实验中，可以清楚地看到，当中断周期大于看门狗定时器的时间，芯片会一直复位，LED灯无法按照设定的频率翻转；反之则LED灯正常工作。

5.10 本章小结

本章主要介绍了智能平衡移动机器人片内外设模块的基本原理及MATLAB/Simulink的仿真实验，包括GPIO模块、ADC模块、Timer_IT模块、ePWM、eCAP及SCI串行通信和SPI串口通信的基本原理，以及对MATLAB/Simulink系统的设置过程，并结合MATLAB/Simulink对每个模块进行了模型搭建实验及自动代码生成。