日程表：

|  |  |  |
| --- | --- | --- |
|  | 硬件 | 软件 |
| 5月19号 | ti主控拓展板绘制并打板 | 显示屏显示时间 |
| 5月20号 | ti主控拓展板绘制并打板 | 编码器读取速度，显示在显示屏上 |
| 5月21号 | 清点比赛所需模块，并购买 | 编写ldc1000代码并读取传感器的值 |
| 5月22号 | 清点比赛所需模块，并购买 | 位移计算并显示在显示屏上 |
| 5月23号 | 清点比赛所需模块，并购买 | 编写电机驱动代码，编写pid算法控制电机恒定转速代码 |
| 5月24号 | 焊接与电路搭建 | 编写舵机摆动跟随金属丝代码 |
| 5月25号 | 焊接与电路搭建 | 调车，在行驶时转向跟随金属丝 |
| 5月26号 | 调车 | 调车 |
| 5月27号 | 调车 | 调车 |
| 5月28号 | 调车 | 调车 |
| 5月29号 | 调车 | 调车 |

软件任务：

1. oled显示位移与时间
2. 编码器读取速度以及位移计算
3. 对于方向pid的设想
4. Ldc1000的mspm0g系列的移植代码
5. 速度环pid的编写与调参
6. 无线串口通信调节速度pid的设想

细分：

1. 显示屏显示时间
2. 编码器读取速度，显示在显示屏上
3. 位移计算并显示在显示屏上
4. 编写电机驱动代码
5. 编写pid算法控制电机恒定转速代码
6. 编写ldc1000代码并读取传感器的值
7. 编写舵机摆动跟随金属丝代码
8. 调车，在行驶时转向跟随金属丝

##### 日志：

###### 5月19号

1. 使用tft屏作为显示屏，在csdn上找到了移植的代码并成功运行。
2. 为tftlcd驱动程序添加了行号常量，方便赋值行数。

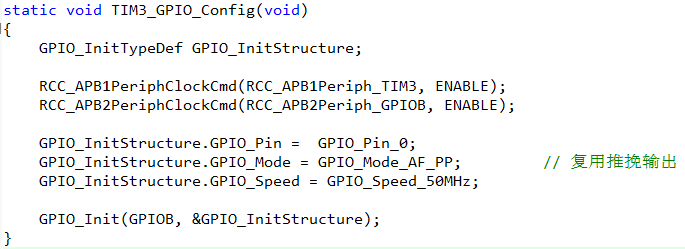
###### 5月21号：

1. 实现了编码器读取、位移计算和电机驱动。

###### 5月22号：

Ldc1000启动流程（stm32例程）：

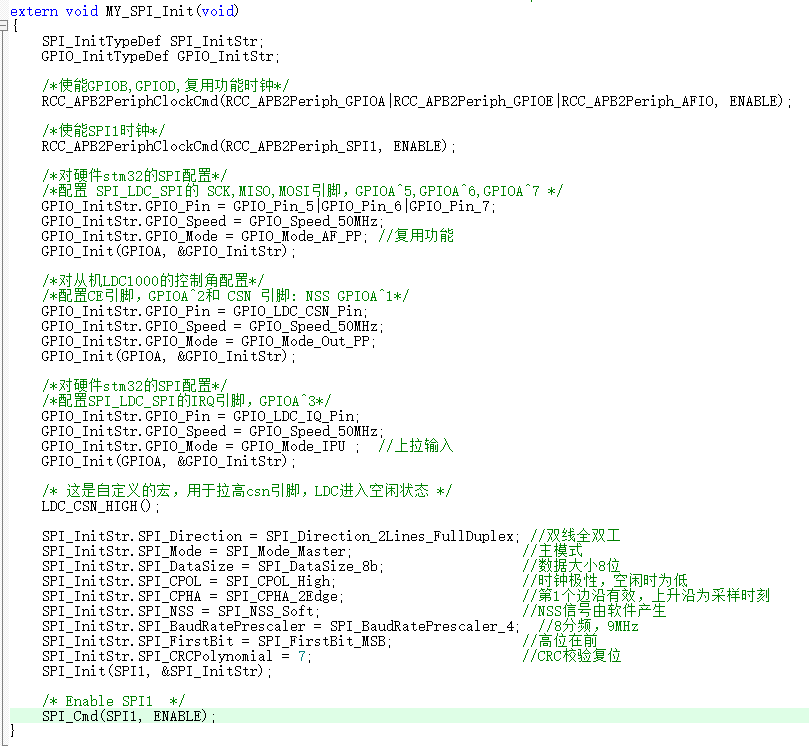
tim3\_gpio的启动



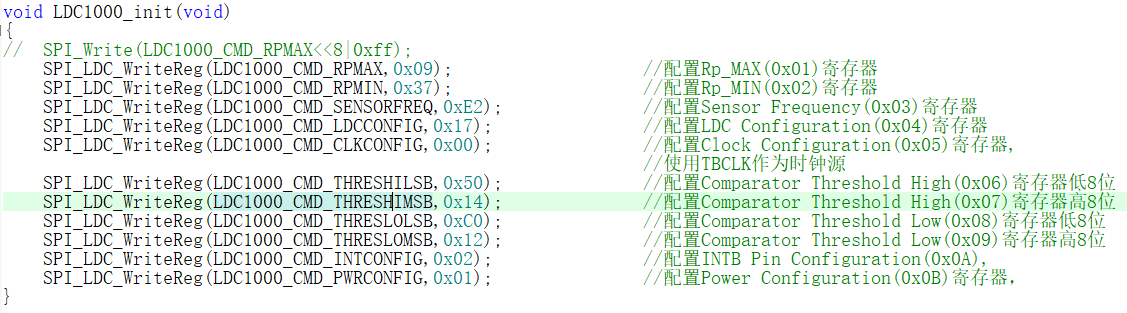
配置 TIM3 通道3生成占空比50%、频率18MHz的PWM信号。



主要配置spi



通过spi与ldc1000通信，发送初始指令配置寄存器



至此初始化结束，以下为调用ldc\_read()：

读取中断引脚直至低电平，然后发送读取指令，以读取Proximity低八位。

读取中断引脚直至低电平，然后发送读取指令，以读取Proximity高八位。

读取中断引脚直至低电平，然后发送读取指令，以读取Frequency低八位。

读取中断引脚直至低电平，然后发送读取指令，以读取Frequency中八位。

读取中断引脚直至低电平，然后发送读取指令，以读取Frequency高八位。

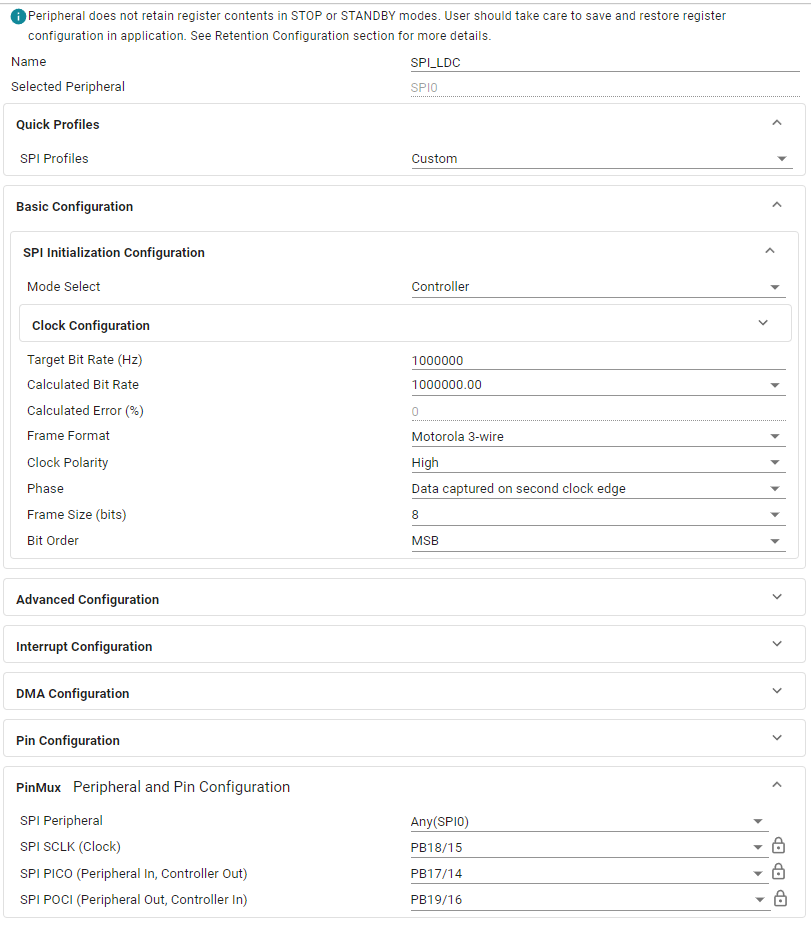


###### 5月23日：

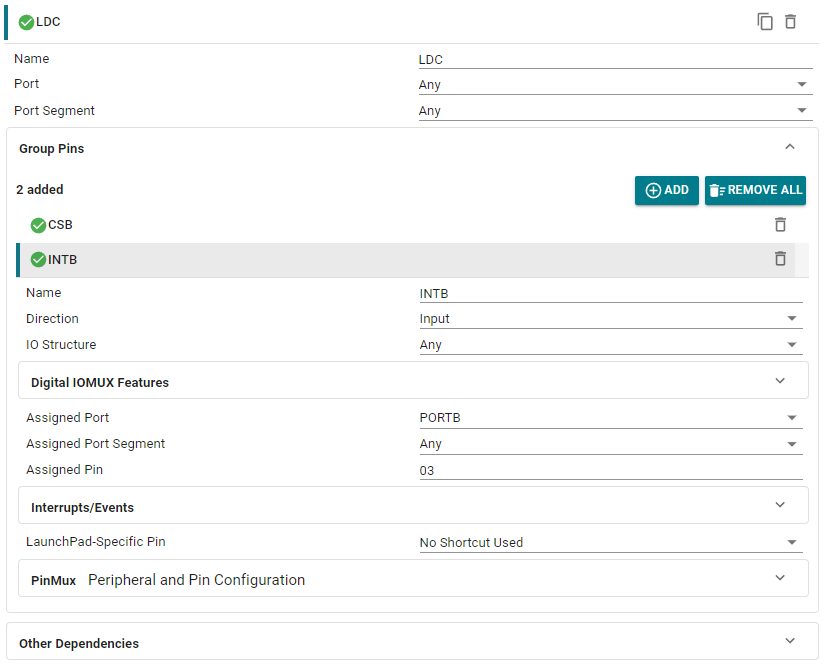
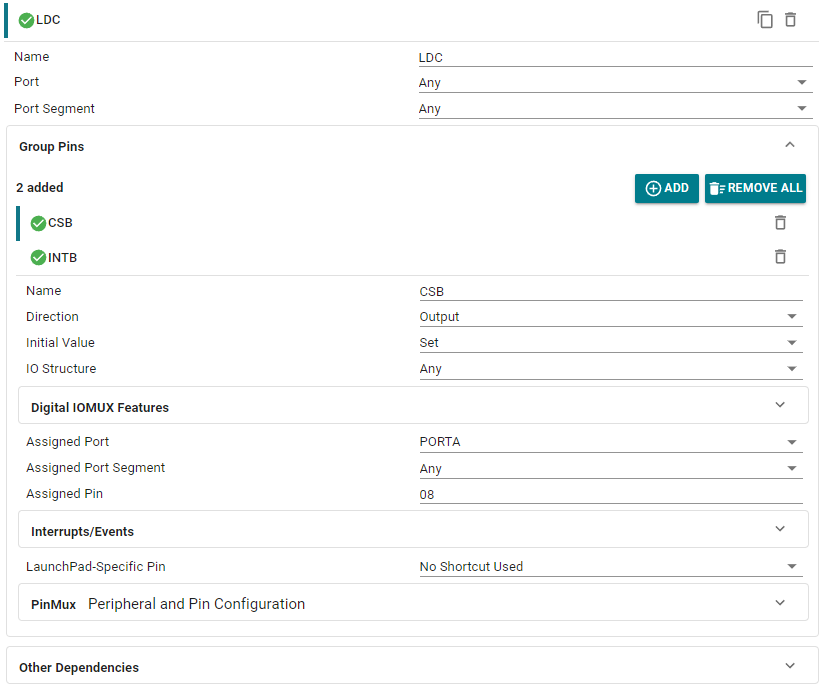
1. 编码器读取速度值和位移，将编码器外部中断优先级设为最高，避免了右侧电机旋转时导致程序卡死的情况。

###### 5月24日：

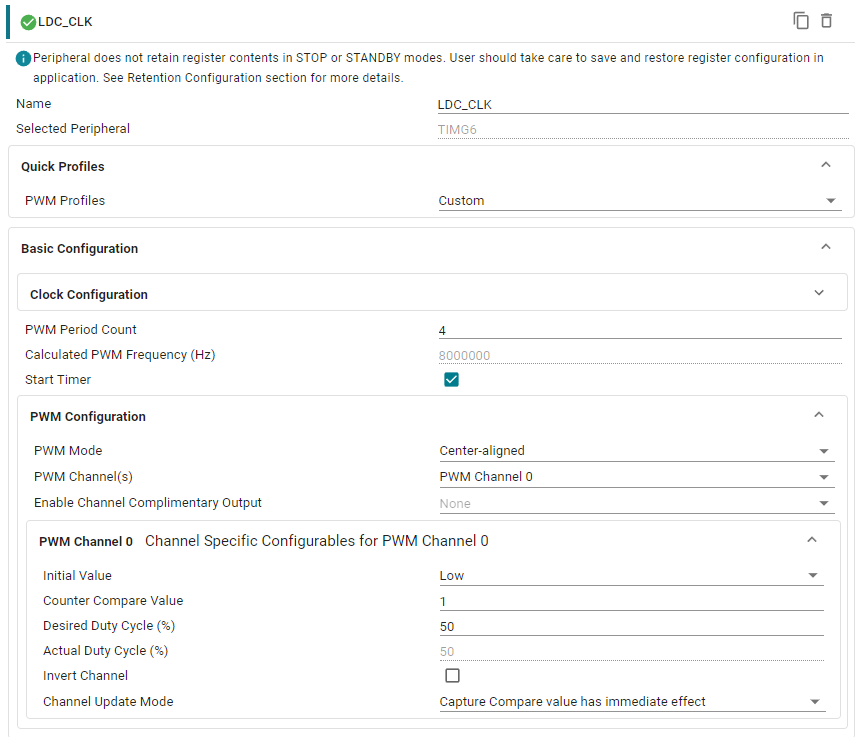
1. 将ldc1000移植至mspm0g3507，主要是配置spi接口和配置（POSI,PISO,SCLK）



gpio引脚(CS,INTB)



8MHz的pwm时钟(ECLK),



1. 程序：

***#ifndef \_\_LDC1000\_H\_\_***

***#define \_\_LDC1000\_H\_\_***

***#include <stdint.h>***

***#include "ti\_msp\_dl\_config.h"***

***#define CSB\_HIGH() DL\_GPIO\_setPins(LDC\_CSB\_PORT, LDC\_CSB\_PIN)***

***#define CSB\_LOW() DL\_GPIO\_clearPins(LDC\_CSB\_PORT, LDC\_CSB\_PIN)***

***#define INTB\_READ() DL\_GPIO\_readPins(LDC\_INTB\_PORT, LDC\_INTB\_PIN)***

***#define SPI\_RWBIT 0x80***

***#define SPI\_NOP 0XFF//空指令用来读状态寄存器***

***/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*寄存器地址区\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***

***#define LDC1000\_REVID 0x00***

***#define LDC1000\_RPMAX 0x01***

***#define LDC1000\_RPMIN 0x02***

***#define LDC1000\_SENSORFREQ 0x03***

***#define LDC1000\_LDCCONFIG 0x04***

***#define LDC1000\_CLKCONFIG 0x05***

***#define LDC1000\_THRESHILSB 0x06***

***#define LDC1000\_THRESHIMSB 0x07***

***#define LDC1000\_THRESLOLSB 0x08***

***#define LDC1000\_THRESLOMSB 0x09***

***#define LDC1000\_INTCONFIG 0x0A***

***#define LDC1000\_PWRCONFIG 0x0B***

***#define LDC1000\_STATUS 0x20***

***#define LDC1000\_PROXLSB 0x21***

***#define LDC1000\_PROXMSB 0x22***

***#define LDC1000\_FREQCTRLSB 0x23***

***#define LDC1000\_FREQCTRMID 0x24***

***#define LDC1000\_FREQCTRMSB 0x25***

***// LDC BITMASKS***

***#define LDC1000\_BIT\_AMPLITUDE 0x18***

***#define LDC1000\_BIT\_RESPTIME 0x07***

***#define LDC1000\_BIT\_CLKSEL 0x02***

***#define LDC1000\_BIT\_CLKPD 0x01***

***#define LDC1000\_BIT\_INTMODE 0x07***

***#define LDC1000\_BIT\_PWRMODE 0x01***

***#define LDC1000\_BIT\_STATUSOSC 0x80***

***#define LDC1000\_BIT\_STATUSDRDYB 0x40***

***#define LDC1000\_BIT\_STATUSWAKEUP 0x20***

***#define LDC1000\_BIT\_STATUSCOMP 0x10***

***#define TEST\_RPMAX\_MAX 0x13 /\*\*< maximum calibration value for RPMAX \*/***

***#define TEST\_RPMAX\_MIN 0x10 /\*\*< minimum calibration value for RPMAX \*/***

***#define TEST\_RPMAX\_INIT TEST\_RPMAX\_MIN+1 /\*\*< RPMAX initial value \*/***

***#define TEST\_RPMIN\_MAX 0x3D /\*\*< maximum calibration value for RPMIN \*/***

***#define TEST\_RPMIN\_MIN 0x3A /\*\*< minimum calibration value for RPMIN \*/***

***#define TEST\_RPMIN\_INIT TEST\_RPMIN\_MIN+1 /\*\*< RPMIN initial value \*/***

***// Final Test Range***

***#define TEST\_RP\_MSB\_MAX 0x12 /\*\*< maximum value for proximity data \*/***

***#define TEST\_RP\_MSB\_MIN 0x0A /\*\*< minimum value for proximity data \*/***

***#define TEST\_FC\_MAX 0x0D5D /\*\*< maximum value for frequency counter \*/***

***#define TEST\_FC\_MIN 0x0D39 /\*\*< minimum value for frequency counter \*/***

***extern volatile uint32\_t ProximityData;***

***extern volatile uint32\_t FrequencyData;***

***void LDC1000\_Init(void);***

***void LDC1000\_ReadData(void);***

***uint8\_t LDC1000\_ReadReg(uint8\_t reg);***

***void LDC1000\_WriteReg(uint8\_t reg, uint8\_t data);***

***#endif***

***#include "ldc1000.h"***

***volatile uint32\_t ProximityData = 0;***

***volatile uint32\_t FrequencyData = 0;***

***/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SPI 底层读写 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***

***uint8\_t SPI\_LDC\_RW(uint8\_t dat)***

***{***

***uint8\_t data = 0;***

***//发送数据***

***DL\_SPI\_transmitData8(SPI\_LDC\_INST,dat);***

***//等待SPI总线空闲***

***while(DL\_SPI\_isBusy(SPI\_LDC\_INST));***

***//接收数据***

***data = DL\_SPI\_receiveData8(SPI\_LDC\_INST);***

***//等待SPI总线空闲***

***while(DL\_SPI\_isBusy(SPI\_LDC\_INST));***

***return data;***

***}***

***/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 寄存器操作 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***

***void LDC1000\_WriteReg(uint8\_t reg, uint8\_t data)***

***{***

***CSB\_LOW();***

***SPI\_LDC\_RW(reg);***

***SPI\_LDC\_RW(data);***

***CSB\_HIGH();***

***}***

***uint8\_t LDC1000\_ReadReg(uint8\_t reg)***

***{***

***uint8\_t Reg\_Value;***

***CSB\_LOW();//拉低csn片选信号，使能LDC的spi传输***

***SPI\_LDC\_RW(reg|SPI\_RWBIT);//选择寄存器***

***Reg\_Value = SPI\_LDC\_RW(SPI\_NOP);***

***CSB\_HIGH();//拉高csn片选信号，即释放LDC的spi传输完毕；***

***return Reg\_Value;***

***}***

***/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 初始化函数 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***

***void LDC1000\_Init(void)***

***{***

***// 配置LDC1000寄存器***

***LDC1000\_WriteReg(LDC1000\_RPMAX, 0x09);***

***LDC1000\_WriteReg(LDC1000\_RPMIN, 0x37);***

***LDC1000\_WriteReg(LDC1000\_SENSORFREQ, 0xE2);***

***LDC1000\_WriteReg(LDC1000\_LDCCONFIG, 0x17);***

***LDC1000\_WriteReg(LDC1000\_CLKCONFIG, 0x00);***

***LDC1000\_WriteReg(LDC1000\_THRESHILSB, 0x50);***

***LDC1000\_WriteReg(LDC1000\_THRESHIMSB, 0x14);***

***LDC1000\_WriteReg(LDC1000\_THRESLOLSB, 0xC0);***

***LDC1000\_WriteReg(LDC1000\_THRESLOMSB, 0x12);***

***LDC1000\_WriteReg(LDC1000\_INTCONFIG, 0x02);***

***LDC1000\_WriteReg(LDC1000\_PWRCONFIG, 0x01);***

***}***

***/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 数据读取函数 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/***

***void LDC1000\_ReadData(void)***

***{***

***ProximityData = 0;***

***FrequencyData = 0;***

***// 等待INTB变低（数据就绪）***

***while(INTB\_READ()!= 0);***

***ProximityData = LDC1000\_ReadReg(LDC1000\_PROXLSB);***

***while(INTB\_READ()!= 0);***

***ProximityData |= LDC1000\_ReadReg(LDC1000\_PROXMSB) << 8;***

***while(INTB\_READ()!= 0);***

***FrequencyData = LDC1000\_ReadReg(LDC1000\_FREQCTRLSB);***

***while(INTB\_READ()!= 0);***

***FrequencyData |= LDC1000\_ReadReg(LDC1000\_FREQCTRMID) << 8;***

***while(INTB\_READ()!= 0);***

***FrequencyData |= LDC1000\_ReadReg(LDC1000\_FREQCTRMSB) << 16;***

***}***

Ps：若ldc1000板上已经有时钟供给eclk，则无需配置pwm。

###### 5月25日：

1. 编写了舵机驱动

舵机逻辑为在小于21650时左转，大于21650时右转，并加上角度限制（80到100），小车将如z形前进。

1. mspm0g3507程序卡死

直接原因是舵机中使用了一个变量判断（servo\_cnt）,其放在tick中断中会增加异常，改名为cnt\_s后正常显示。在timer0中断中使用extern cnt，cnt++的写法，在舵机proc使用该变量是会卡死，在debug中查到是timer0中断导致错误，具体原因不明。解决方法是改为在tick中断内自增且改名。

1. 编写蜂鸣器逻辑。

在检测到硬币（ldc1000读值为23000以上）时，持续响声且记录。