实验7 STM32数据采集与控制-实验报告

电 25 吴晨聪 2022010311

# 7.1 用定时器实现PWM输出，并用ADC采集输出波形

任务1可以分解为3个部分：互补PWM输出；双路ADC采样；串口数据传输及接收。

互补PWM波输出利用STM32里TIM1的CH1和CH1N来实现，设置时钟频率为120MHz，配置Prescaler和Counter Period分别为1699和267，计算的PWM波的频率为

再设置Pulse=200，该系数影响的是占空比。在”main.c”文件的开始再添加以下代码开启PWM波，如此即完成了互补PWM波的硬件设置。

HAL\_TIM\_Base\_Start(&htim1);

HAL\_TIM\_PWM\_Start(&htim1,TIM\_CHANNEL\_1);

HAL\_TIMEx\_PWMN\_Start(&htim1,TIM\_CHANNEL\_1);

双路ADC采样采用的是ADC1的CH1和ADC2的CH2双通道采样，采样的触发由TIM3来进行，如此可以通过配置TIM3的频率来配置采样的频率，在此处设置TIM3的频率为120MHz÷120÷20=50kHz，设置较高的采样频率使波形尽量完整。同时配置ADC1工作模式为Dual regular simultaneous mode only，再设置DMA的储存长度和地址长度均为word，使得DMA能够存下两路ADC采样的数据。同样在”main.c”的开始先启动两路ADC：

HAL\_ADCEx\_Calibration\_Start(&hadc1, ADC\_SINGLE\_ENDED);

HAL\_ADCEx\_Calibration\_Start(&hadc2, ADC\_SINGLE\_ENDED);

HAL\_ADC\_Start(&hadc2);

HAL\_ADCEx\_MultiModeStart\_DMA(&hadc1,(uint32\_t\*)&ADC1ConvertedData,ADC\_CONVERTED\_DATA\_BUFFER\_SIZE);

再在DMA中断处发送数据：

void HAL\_ADC\_ConvCpltCallback(ADC\_HandleTypeDef\* hadc) {

HAL\_UART\_Transmit(&huart2, (uint8\_t \* ) &FrameHeader, 2, 0xFFFF);

HAL\_UART\_Transmit(&huart2,(uint8\_t\*)ADC1ConvertedData,ADC\_CONVERTED\_DATA\_BUFFER\_SIZE\* 4, 0xFFFF);

HAL\_UART\_Transmit(&huart2, (uint8\_t \* ) &FrameTerm, 2, 0xFFFF);

}

此处ADC\_CONVERTED\_DATA\_BUFFER\_SIZE是一个设置在”main.h”中的参数，表示DMA存储的每路ADC转化的数据数，当DMA中存到了该数目的数据以后会调用DMA的中断函数，然后根据中断函数的内容把数据打包发送出去。这里为了保证接收数据的完整性（以免打开时恰好一个数据被拆开接收），设置了字头和字尾。

STM32的串口部分设置波特率为5000000Bit/s（因为采样率较高而且是双路ADC，需要较高的波特率才能完整地传输实时数据），发送部分就在DMA中断函数中实现。接收部分用Simulink完成，接收时要重点注意传输数据的格式。首先一个DMA是32位，里面以little-endian模式存放了16位CH1数据和16位CH2数据，一次发送ADC\_CONVERTED\_DATA\_BUFFER\_SIZE=16个DMA的数据。由于Matlab将数据存入矩阵的顺序是按列存，所以需要设置一个[2 16]的矩阵接收数据，再转置得到两路波形信号。此外波特率、校验位、停止位、帧头帧尾等设置也需要完全相同才能保证数据接收成功。

一張含有 螢幕擷取畫面, 正方形, Rectangle 的圖片

AI 產生的內容可能不正確。

图1 互补Pulse 200 PWM波的波形（死区时间=100）

一張含有 螢幕擷取畫面, Rectangle, 行 的圖片

AI 產生的內容可能不正確。

图1 互补Pulse 200 PWM波的波形（死区时间=50）

看到由于采样率足够高，采样的波形还是很完整的，PWM的互补性也可以观察到。输出波形并不是完美的矩形波，这可能是由STM32本身的局限性导致的，但这并不影响对结果的观测。

至于传输方式的选择，实际上经过了好几次的变化。首先采用的是ADC采样中断并存在数组内分别传输，但这种方法显然效率太低；此后采用了ADC的分别DMA采样再发送的方式，但这个方法也出现了问题，经过观察发现仅第一路ADC的数据能完整传输到PC端，第二路的ADC数据在传输时会有很大的失真，在每次打包的数据数目越大时，这样失真现象更为严重。经过仔细分析，认为问题的原因在于传输时直接将DMA内的数据通过串口传输，而两路DAC的输出通过两行代码进行传输，而在传输第一个通道的数据时DMA仍然在更新，因此第二路DAC信号在传输时变有部分发生了变化，从而得不到正确的波形；因此，最后将两路DAC的数据存在一路DMA中，既可以用一次传输发送两路的信号，经过尝试，这样的变化能够得到正确的波形。

# 7.2 用定时器实现双脉冲输出

任务2可以被分解成两个部分：一个用来实现timer驱动的双脉冲产生，另一个是串口传输波形并显示。串口传输部分可以直接沿用任务1的结果，所以只需要设计timer驱动的双脉冲发生电路即可。

双脉冲信号产生的思路为：利用两个计时器tim1和tim2，其中tim1用来输出一个占空比为100%的PWM波形（即高电平），tim2用来计数并操纵tim1的开启和关断。理论上就能设置任意个、任意持续时间的脉冲波形。

为了实现这样的想法，首先要配置两个定时器的时钟频率为

设置tim2的频率远大于tim1是为了方便用tim2的计数来控制tim1。同时要打开tim2的中断使能和tim1的PWM输出CH1。

在”main.c”中，首先打开tim2的中断使能

HAL\_TIM\_Base\_Start\_IT(&htim2);

同时设置一个变量count用于计数，每次进入tim2的中断中都会将count的值+1。

再利用count来控制tim1的启停，在count=0时使能PWM输出，在count=100时关闭PWM输出，在count=300时再次开启PWM输出，在count=400时再次关闭PWM输出。则从输出的角度看，就相当于输出了两次高电位的脉冲信号。

1. 中断回调函数

void HAL\_ADC\_ConvCpltCallback(ADC\_HandleTypeDef \*hadc)

{

HAL\_UART\_Transmit(&huart2, (uint8\_t \*)&ADC1ConvertedData, LEN\*2, 0xFFFF);

}

1. 串口输出数据

void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef\*htim)

{

if(count == 1000){

HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_1);

count = 0;

}

else{

count++;

if(count == 100){ // t1=100/100000s

HAL\_TIM\_PWM\_Stop(&htim1, TIM\_CHANNEL\_1);

}

if(count == 300){ // t2=0.002s

HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_1);

}

if(count == 400){ // t3=0.001s

HAL\_TIM\_PWM\_Stop(&htim1, TIM\_CHANNEL\_1);

}

}

}

通过改变开启和关闭TIM1的时间间隔实现t1、t2、t3的配置。

如图2所示是Simulink接收到的波形结果，同样通过调整if语句中count的值，可以改变脉冲的宽度、间隔。改变后的波形如图3所示：

一張含有 螢幕擷取畫面 的圖片

AI 產生的內容可能不正確。

图2 Simulink观测到的双脉冲波形

一張含有 螢幕擷取畫面, 行 的圖片

AI 產生的內容可能不正確。

图3 改变脉冲宽度、脉冲间隔的双脉冲波形

如图2所示是Simulink接收到的波形结果，同样通过调整if语句中count的值，可以改变脉冲的宽度、间隔。

改变后的波形如图3所示，理论上该方法可以得到任意宽度、任意间隔的双脉冲甚至多脉冲波形。虽然对于时间的刻画是利用tim2和count来体现的，tim2的频率决定了可操纵时间的精度，但越高的精度就需要越大的count来存储。