

(8条消息) FOC和SVPWM的C语言代码实现_qlexcel的专栏-CSDN博客

SVPWM的原理讲解在这儿: <https://blog.csdn.net/qlexcel/article/details/74787619#comments>

现在开始分析C语言的代码（代码建议复制到notepad++中查看），为方便读者试验，每个代码都是独立的子模块，复制到工程中就可以编译运行：

一、配置高级定时器TIM1产生6路互补PWM，带刹车保护

详细配置代码如下，把下面的程序段拷贝到main.c中直接就可以输出PWM波形（要保证BKIN下拉），方便读者验证：

```
1. static void TIM1_GPIO_Config(void)
2. {
3.     GPIO_InitTypeDef GPIO_InitStructure;
4.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
5.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB, ENABLE);
6.
7.     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9| GPIO_Pin_10 | GPIO_Pin_11;
8.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
9.     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
10.    GPIO_Init(GPIOA, &GPIO_InitStructure);
11.
12.    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13| GPIO_Pin_14 | GPIO_Pin_15;
13.    GPIO_Init(GPIOB, &GPIO_InitStructure);
14.
15.    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
16.    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
17.    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
18.    GPIO_Init(GPIOB, &GPIO_InitStructure);
19.
20.    GPIO_PinLockConfig(GPIOA, GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 );
21. }
22.
23. #define CKTIM      ((u32)72000000uL)
24. #define PWM_PRSC    ((u8)0)
25. #define PWM_FREQ    ((u16) 15000)
26. #define PWM_PERIOD  ((u16) (CKTIM / (u32) (2 * PWM_FREQ * (PWM_PRSC+1))))
27. #define REP_RATE    (1)
28.
29. #define DEADTIME_NS    ((u16)1000)
30. #define DEADTIME      (u16)((unsigned long long)CKTIM/2 * (unsigned long long)DEADTIME_NS/1000000000uL)
31.
32. static void TIM1_Mode_Config(void)
33. {
34.     TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
35.     TIM_OCInitTypeDef TIM_OCInitStructure;
36.     TIM_BDTRInitTypeDef TIM1_BDTRInitStructure;
37.     NVIC_InitTypeDef NVIC_InitStructure;
38.
39.     TIM_TimeBaseStructure.TIM_Period = PWM_PERIOD;
40.     TIM_TimeBaseStructure.TIM_Prescaler = PWM_PRSC;
41.     TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV2;
42.     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_CenterAligned1;
43.     TIM_TimeBaseStructure.TIM_RepetitionCounter = REP_RATE;
44.     TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
45.
46.     TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
47.     TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
48.     TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
49.     TIM_OCInitStructure.TIM_Pulse = 800;
50.     TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
51.     TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
52.     TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Reset;
53.     TIM_OCInitStructure.TIM_OCNIIdleState = TIM_OCNIIdleState_Set;
54.
55.     TIM_OC1Init(TIM1, &TIM_OCInitStructure);
56.
57.     TIM_OCInitStructure.TIM_Pulse = 800;
58.     TIM_OC2Init(TIM1, &TIM_OCInitStructure);
59.
60.     TIM_OCInitStructure.TIM_Pulse = 800;
61.     TIM_OC3Init(TIM1, &TIM_OCInitStructure);
```

```

62.
63.
64.     TIM1_BDTRInitStructure.TIM_OSSRState = TIM_OSSRState_Enable;
65.     TIM1_BDTRInitStructure.TIM_OSSIState = TIM_OSSIState_Enable;
66.     TIM1_BDTRInitStructure.TIM_LOCKLevel = TIM_LOCKLevel_1;
67.     TIM1_BDTRInitStructure.TIM_DeadTime = DEADTIME;
68.     TIM1_BDTRInitStructure.TIM_Break = TIM_Break_Enable;
69.     TIM1_BDTRInitStructure.TIM_BreakPolarity = TIM_BreakPolarity_High;
70.     TIM1_BDTRInitStructure.TIM_AutomaticOutput = TIM_AutomaticOutput_Disable;
71.     TIM_BDTRConfig(TIM1, &TIM1_BDTRInitStructure);
72.
73.     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
74.
75.     NVIC_InitStructure.NVIC_IRQChannel = TIM1_BRK_IRQn;
76.     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
77.     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
78.     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
79.     NVIC_Init(&NVIC_InitStructure);
80.
81.     TIM_SelectOutputTrigger(TIM1, TIM_TRGOSource_Update);
82.     TIM_ClearITPendingBit(TIM1, TIM_IT_Break);
83.     TIM_ITConfig(TIM1, TIM_IT_Break, ENABLE);
84.     TIM_CtrlPWMOutputs(TIM1, ENABLE);
85.     TIM_Cmd(TIM1, ENABLE);
86. }
87. void TIM1_PWM_Init(void)
88. {
89.     TIM1_GPIO_Config();
90.     TIM1_Mode_Config();
91. }
92.
93.
94.
95.
96.
97. void TIM1_BRK_IRQHandler(void)
98. {
99.
100.     TIM_ClearITPendingBit(TIM1, TIM_IT_Break);
101. }

```

1、配置TIM1的CH1--A8、CH2--A9、CH3--A10、CH4-A11、CH1N-B13、CH2N-B14、CH3N-B15、BKIN-B12

BKIN作为报警信号或者刹车信号的输入，当检测此信号时，TIM1的PWM会硬件上停止输出，实时性好，起到保护硬件电路的作用。

2、观察SVPWM的PWM波形是对称的：

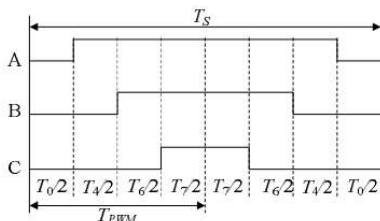
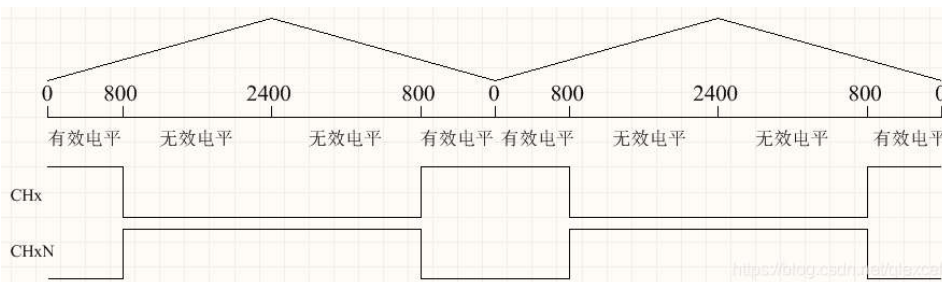


图 2-11 扇区 I 内的三相波形

正好配置TIM1为中央对齐模式1，在上面代码的配置中，载波周期为15KHz，TIM_Period(ARR)=2400，CH1的TIM_Pulse(CCR)=800。采用的PWM1模式，即CNT小于CCR时，输出有效电平，大于CCR小于ARR时，输出无效电平，又配置CHx的有效电平为高电平，CHxN的有效电平为高电平，则可以得到下面的PWM波形：



如果CHxN的有效电平是低电平，则输出的CHx和CHxN的波形是相同的。（可能CHx和CHxN有效电平的叫法相反）

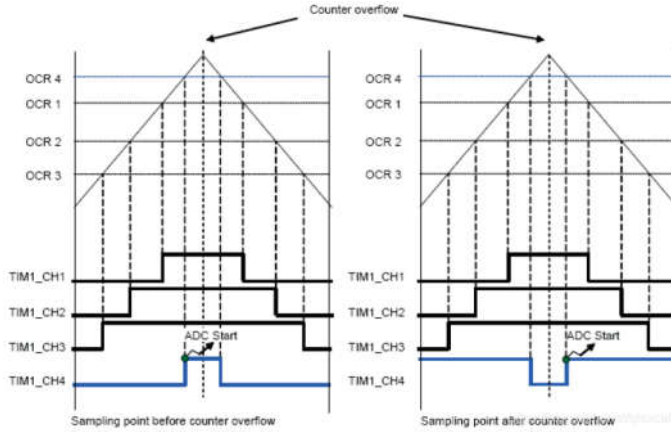
3、配置CHx和CHxN空闲时的电平，调用TIM_CtrlPWMOutputs(TIM1, DISABLE)后，就进入空闲状态了，高侧没什么用，让空闲时低侧的管子导通，可以使相线连在一起，起到锁定的作用。

4、改变 REP_RATE 的值，可以更改TRGO信号输出的频率。因为相电流采样由TIM1的TRGO信号触发，故更改REP_RATE可以调整电流环的计算频率(每次相电流采样后，会进行一次FOC运算)。采样频率关系： $(2 * PWM_FREQ) / (REP_RATE + 1)$ ，如：当PWM_FREQ=15KHz，REP_RATE=0，则采样频率为30KHz。

5、TIM_OSSRState和TIM_OSSIState直接Enable就可以了，详情可以去看用户手册。

6、使用TIM_SelectOutputTrigger(TIM1, TIM_TRGOSource_Update);函数设置TRGO信号的产生源，TIM_TRGOSource_Update参数代表TIM1产生一次更新事件，就输出一一次TRGO信号。TRGO信号用来触发相电流的采样。

当然也可以使用TIM1的CH4来触发相电流采样，参数为TIM_TRGOSource_OC4Ref，再打开CH4，并配置CH4的比较值，比如配置比较值为PWM_PERIOD-5。这样当CNT计数到PWM_PERIOD-5时就会触发相电流的ADC采样，这种方法比较灵活，可以合理设置CH4的比较值，来让相电流采样点避开开关噪声。



二、配置双ADC模式和规则组、注入组，其中注入组由TIM1的TRGO触发

```
1.
2. #define PHASE_A_ADC_CHANNEL          ADC_Channel_11
3. #define PHASE_A_GPIO_PORT            GPIOC
4. #define PHASE_A_GPIO_PIN             GPIO_Pin_1
5.
6. #define PHASE_B_ADC_CHANNEL          ADC_Channel_10
7. #define PHASE_B_GPIO_PORT            GPIOC
8. #define PHASE_B_GPIO_PIN             GPIO_Pin_0
9.
10. #define TEMP_FDBK_CHANNEL            ADC_Channel_13
11. #define TEMP_FDBK_CHANNEL_GPIO_PORT   GPIOC
12. #define TEMP_FDBK_CHANNEL_GPIO_PIN    GPIO_Pin_3
13.
14. #define BUS_VOLT_FDBK_CHANNEL         ADC_Channel_14
15. #define BUS_VOLT_FDBK_CHANNEL_GPIO_PORT GPIOC
16. #define BUS_VOLT_FDBK_CHANNEL_GPIO_PIN GPIO_Pin_4
17.
18. #define POT1_VOLT_FDBK_CHANNEL        ADC_Channel_12
19. #define POT1_VOLT_FDBK_CHANNEL_GPIO_PORT GPIOC
20. #define POT1_VOLT_FDBK_CHANNEL_GPIO_PIN GPIO_Pin_2
21.
22. #define BUS_SHUNT_CURR_CHANNEL        ADC_Channel_15
23. #define BUS_SHUNT_CURR_CHANNEL_GPIO_PORT GPIOC
24. #define BUS_SHUNT_CURR_CHANNEL_GPIO_PIN GPIO_Pin_5
25.
26. #define BRK_SHUNT_CURR_CHANNEL        ADC_Channel_7
27. #define BRK_SHUNT_CURR_CHANNEL_GPIO_PORT GPIOA
28. #define BRK_SHUNT_CURR_CHANNEL_GPIO_PIN GPIO_Pin_7
29.
30. #define AIN0_VOLT_FDBK_CHANNEL        ADC_Channel_8
31. #define AIN0_VOLT_FDBK_CHANNEL_GPIO_PORT GPIOB
32. #define AIN0_VOLT_FDBK_CHANNEL_GPIO_PIN GPIO_Pin_0
33.
34. #define AIN1_VOLT_FDBK_CHANNEL        ADC_Channel_9
35. #define AIN1_VOLT_FDBK_CHANNEL_GPIO_PORT GPIOB
36. #define AIN1_VOLT_FDBK_CHANNEL_GPIO_PIN GPIO_Pin_1
37.
38. #define ADC1_DR_Address                ((uint32_t)0x4001244C)
39. #define BufferLenght                   36
40. volatile u32 ADC_DualConvertedValueTab[BufferLenght];
41. volatile u16 ADC1_RegularConvertedValueTab[BufferLenght];
42. volatile u16 ADC2_RegularConvertedValueTab[BufferLenght];
43. static u16 hPhaseAOffset;
44. static u16 hPhaseBOffset;
45.
46. void ADC_DMA_Init(void)
47. {
48.     u8 bIndex;
49.
```

```

50.     GPIO_InitTypeDef GPIO_InitStructure;
51.     ADC_InitTypeDef ADC_InitStructure;
52.     DMA_InitTypeDef DMA_InitStructure;
53.     NVIC_InitTypeDef NVIC_InitStructure;
54.
55.     RCC_ADCCLKConfig(RCC_PCLK2_Div6);
56.     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
57.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
58.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC2, ENABLE);
59.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO | RCC_APB2Periph_GPIOA |
60.                             RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOE, ENABLE);
61.
62.     GPIO_StructInit(&GPIO_InitStructure);
63.     GPIO_InitStructure.GPIO_Pin = PHASE_A_GPIO_PIN;
64.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
65.     GPIO_Init(PHASE_A_GPIO_PORT, &GPIO_InitStructure);
66.
67.     GPIO_InitStructure.GPIO_Pin = PHASE_B_GPIO_PIN;
68.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
69.     GPIO_Init(PHASE_B_GPIO_PORT, &GPIO_InitStructure);
70.
71.     GPIO_InitStructure.GPIO_Pin = TEMP_FDBK_CHANNEL_GPIO_PIN;
72.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
73.     GPIO_Init(TEMP_FDBK_CHANNEL_GPIO_PORT, &GPIO_InitStructure);
74.
75.     GPIO_InitStructure.GPIO_Pin = BUS_VOLT_FDBK_CHANNEL_GPIO_PIN;
76.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
77.     GPIO_Init(BUS_VOLT_FDBK_CHANNEL_GPIO_PORT, &GPIO_InitStructure);
78.
79.     GPIO_InitStructure.GPIO_Pin = POT1_VOLT_FDBK_CHANNEL_GPIO_PIN;
80.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
81.     GPIO_Init(POT1_VOLT_FDBK_CHANNEL_GPIO_PORT, &GPIO_InitStructure);
82.
83.     GPIO_InitStructure.GPIO_Pin = BUS_SHUNT_CURR_CHANNEL_GPIO_PIN;
84.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
85.     GPIO_Init(BUS_SHUNT_CURR_CHANNEL_GPIO_PORT, &GPIO_InitStructure);
86.
87.     GPIO_StructInit(&GPIO_InitStructure);
88.     GPIO_InitStructure.GPIO_Pin = BRK_SHUNT_CURR_CHANNEL_GPIO_PIN;
89.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
90.     GPIO_Init(BRK_SHUNT_CURR_CHANNEL_GPIO_PORT, &GPIO_InitStructure);
91.
92.     GPIO_StructInit(&GPIO_InitStructure);
93.     GPIO_InitStructure.GPIO_Pin = AIN0_VOLT_FDBK_CHANNEL_GPIO_PIN | AIN1_VOLT_FDBK_CHANNEL_GPIO_PIN;
94.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
95.     GPIO_Init(AIN0_VOLT_FDBK_CHANNEL_GPIO_PORT, &GPIO_InitStructure);
96.
97.
98.     DMA_DeInit(DMA1_Channel1);
99.     DMA_StructInit(&DMA_InitStructure);
100.    DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;
101.    DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)ADC_DualConvertedValueTab;
102.    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
103.    DMA_InitStructure.DMA_BufferSize = BufferLenght;
104.    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
105.    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
106.    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
107.    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
108.    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
109.    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
110.    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
111.    DMA_Init(DMA1_Channel1, &DMA_InitStructure);
112.
113.    DMA_ClearITPendingBit(DMA1_IT_TC1);
114.    DMA_ITConfig(DMA1_Channel1, DMA_IT_TC, ENABLE);
115.    DMA_Cmd(DMA1_Channel1, ENABLE);
116.
117.
118.
119.    ADC_DeInit(ADC1);
120.    ADC_DeInit(ADC2);

```

```

121.     ADC_StructInit(&ADC_InitStructure);
122.     ADC_InitStructure.ADC_Mode = ADC_Mode_RegInjecSimult;
123.     ADC_InitStructure.ADC_ScanConvMode = ENABLE;
124.     ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
125.     ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
126.     ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Left;
127.     ADC_InitStructure.ADC_NbrOfChannel = 3;
128.     ADC_Init(ADC1, &ADC_InitStructure);
129.
130.     ADC_DMACmd(ADC1, ENABLE);
131.
132.     ADC_StructInit(&ADC_InitStructure);
133.     ADC_InitStructure.ADC_Mode = ADC_Mode_RegInjecSimult;
134.     ADC_InitStructure.ADC_ScanConvMode = ENABLE;
135.     ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
136.     ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
137.     ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Left;
138.     ADC_InitStructure.ADC_NbrOfChannel = 3;
139.     ADC_Init(ADC2, &ADC_InitStructure);
140.     ADC_ExternalTrigConvCmd(ADC2, ENABLE);
141.
142.     ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_239Cycles5);
143.     ADC_RegularChannelConfig(ADC1, BRK_SHUNT_CURR_CHANNEL, 2, ADC_SampleTime_239Cycles5);
144.     ADC_RegularChannelConfig(ADC1, BUS_SHUNT_CURR_CHANNEL, 3, ADC_SampleTime_239Cycles5);
145.     ADC_RegularChannelConfig(ADC2, POT1_VOLT_FDBK_CHANNEL, 1, ADC_SampleTime_239Cycles5);
146.     ADC_RegularChannelConfig(ADC2, AIN0_VOLT_FDBK_CHANNEL, 2, ADC_SampleTime_239Cycles5);
147.     ADC_RegularChannelConfig(ADC2, AIN1_VOLT_FDBK_CHANNEL, 3, ADC_SampleTime_239Cycles5);
148.
149.     ADC_Cmd(ADC1, ENABLE);
150.     ADC_TempSensorVrefintCmd(ENABLE);
151.
152.     ADC_ResetCalibration(ADC1);
153.     while(ADC_GetResetCalibrationStatus(ADC1));
154.     ADC_StartCalibration(ADC1);
155.     while(ADC_GetCalibrationStatus(ADC1));
156.
157.     ADC_Cmd(ADC2, ENABLE);
158.     ADC_ResetCalibration(ADC2);
159.     while(ADC_GetResetCalibrationStatus(ADC2));
160.     ADC_StartCalibration(ADC2);
161.     while(ADC_GetCalibrationStatus(ADC2));
162.
163.
164.     ADC_InjectedSequencerLengthConfig(ADC1,2);
165.     ADC_ITConfig(ADC1, ADC_IT_JEOC, DISABLE);
166.     hPhaseAOffset=0;
167.     hPhaseBOffset=0;
168.     ADC_ExternalTrigInjectedConvConfig(ADC1, ADC_ExternalTrigInjecConv_None);
169.     ADC_ExternalTrigInjectedConvCmd(ADC1,ENABLE);
170.     ADC_InjectedChannelConfig(ADC1, PHASE_A_ADC_CHANNEL,1,ADC_SampleTime_7Cycles5);
171.     ADC_InjectedChannelConfig(ADC1, PHASE_B_ADC_CHANNEL,2,ADC_SampleTime_7Cycles5);
172.
173.     ADC_ClearFlag(ADC1, ADC_FLAG_JEOC);
174.     ADC_SoftwareStartInjectedConvCmd(ADC1,ENABLE);
175.
176. for(bIndex=16; bIndex !=0; bIndex--)
177.     {
178. while(!ADC_GetFlagStatus(ADC1,ADC_FLAG_JEOC)) { }
179.
180.
181.         hPhaseAOffset += (ADC_GetInjectedConversionValue(ADC1,ADC_InjectedChannel_1)>>3);
182.         hPhaseBOffset += (ADC_GetInjectedConversionValue(ADC1,ADC_InjectedChannel_2)>>3);
183.         ADC_ClearFlag(ADC1, ADC_FLAG_JEOC);
184.         ADC_SoftwareStartInjectedConvCmd(ADC1,ENABLE);
185.     }
186.
187.
188.     ADC_InjectedChannelConfig(ADC1, PHASE_A_ADC_CHANNEL, 1, ADC_SampleTime_7Cycles5);
189.     ADC_InjectedChannelConfig(ADC1, BUS_VOLT_FDBK_CHANNEL,2, ADC_SampleTime_7Cycles5);
190.     ADC_ExternalTrigInjectedConvConfig(ADC1, ADC_ExternalTrigInjecConv_T1_TRGO);
191.     ADC_ITConfig(ADC1, ADC_IT_JEOC, ENABLE);

```

```

192.
193.     ADC_InjectedSequencerLengthConfig(ADC2,2);
194.     ADC_InjectedChannelConfig(ADC2, PHASE_B_ADC_CHANNEL, 1,ADC_SampleTime_7Cycles5);
195.     ADC_InjectedChannelConfig(ADC2, TEMP_FDBK_CHANNEL, 2,ADC_SampleTime_7Cycles5);
196.     ADC_ExternalTrigInjectedConvCmd(ADC2,ENABLE);
197.
198.     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
199.
200.     NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQn;
201.     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
202.     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
203.     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
204.     NVIC_Init(&NVIC_InitStructure);
205.
206.     NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel1_IRQn;
207.     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
208.     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
209.     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
210.     NVIC_Init(&NVIC_InitStructure);
211.
212.
213.
214.     ADC_SoftwareStartConvCmd(ADC1, ENABLE);
215. }
216.
217. u16 h_ADCBusvolt;
218. u16 h_ADCTemp;
219. u16 h_ADCPhase_A;
220. u16 h_ADCPhase_B;
221. void ADC1_2_IRQHandler(void)
222. {
223.     if((ADC1->SR & ADC_FLAG_JEOC) == ADC_FLAG_JEOC)
224.     {
225.         ADC1->SR = ~(u32)ADC_FLAG_JEOC;
226.
227.
228.         h_ADCTemp=ADC_GetInjectedConversionValue(ADC2,ADC_InjectedChannel_2);
229.         h_ADCBusvolt=ADC_GetInjectedConversionValue(ADC1,ADC_InjectedChannel_2);
230.
231.
232.         h_ADCPhase_A=ADC_GetInjectedConversionValue(ADC1,ADC_InjectedChannel_1);
233.         h_ADCPhase_B=ADC_GetInjectedConversionValue(ADC2,ADC_InjectedChannel_1);
234.     }
235. }
236.
237. void DMA1_Channel1_IRQHandler(void)
238. {
239.     u8 i,j=0;
240.
241.     if(DMA_GetITStatus(DMA1_IT_TC1))
242.     {
243.         DMA_ClearITPendingBit(DMA1_IT_GL1);
244.         for(i=0;i<BufferLenght;i++)
245.         {
246.             ADC1_RegularConvertedValueTab[j++] = (uint16_t)(ADC_DualConvertedValueTab[i]>>4);
247.         }
248.
249.         j = 0;
250.
251.         for(i=0;i<BufferLenght;i++)
252.         {
253.             ADC2_RegularConvertedValueTab[j++] = (uint16_t)(ADC_DualConvertedValueTab[i] >> 20);
254.         }
255.
256.     }
257. }

```

1、stm32的ADC转换速度为1MHz，精度为12位。采样时间可设置（1.5到239.5个周期），最小采样时间107ns。使用双ADC模式，同时触发ADC1、ADC2采集电机的两相电流，可保证采集到的两相电流值时间误差最小。配置ADC1为主，ADC2为从，用ADC1触发ADC2。

2、在众多的ADC采样通道中，A相、B相、母线电压值、散热器温度是对实时性要求比较高的，于是把他们配置成注入组通道，其余的配置成规则组。（注入组与规则组的关系和main中的while循环与中断类似，当注入组被触发时会打断规则组的ADC转换，优先转换注入组的通道，当注入组转换完成，规则组才继续转换）

3、在双ADC模式下，DR寄存器的高16位存储了ADC2的转换数据，低16位存储了ADC1的转换数据：

ADC 规则数据寄存器(ADC_DR)

地址偏移: 0x4C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADC2DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

位数	描述
位 31:16	ADC2DATA[15:0]: ADC2 转换的数据 (ADC2 data) - 在 ADC1 中: 双模式下, 这些位包含了 ADC2 转换的规则通道数据。 - 在 ADC2 和 ADC3 中: 不使用这些位。
位 15:0	DATA[15:0]: 规则转换的数据 (Regular data) 这些位为只读, 包含了规则通道的转换结果。数据是左对齐或右对齐。

4、左右对齐的问题, ADC的转换精度只有12位, 要保存的数据宽度为16位, 因此存在靠左还是靠右的问题。

数据右对齐

注入组

SEX	SEX	SEX	SEX	D1	D1	D	D	D	D	D	D	D	D	D	D
T	T	T	T	1	0	9	8	7	6	5	4	3	2	1	0

规则组

0	0	0	0	D1	D1	D	D	D	D	D	D	D	D	D	D
1	0	0	0	1	0	9	8	7	6	5	4	3	2	1	0

数据左对齐

注入组

SEX	D11	D10	D9	D8	D7	D	D	D	D	D	D	D	0	0	0
T						6	5	4	3	2	1	0			

规则组

D11	D10	D9	D8	D7	D6	D	D	D	D	D	D	0	0	0	0
						5	4	3	2	1	0				

如上, 注入组合规则组的左右对齐并不一样。推荐使用右对齐, 直接取低12位即可。(上面的例程使用的左对齐, 要做修改)

5、为了获取A、B相零电流时的值用于后面电机运行电流的矫正, 先把注入组设置为软件触发, 把零电流值采集完成后, 再把配置修改成用TIM1的TRGO信号触发。因为是双ADC模式, 只需要配置ADC1的触发信号就可以了。

6、求Q1.15格式的零电流值, 16个(零电流值/8)的累加, 把最高符号位溢出:

我们都知道MCU处理定点数会很快, 处理浮点数比较慢, 但是相电流采样值一般都比较小, 会有小数, 因此我们要使用Q格式来让浮点数据转化为定点数, 提高处理速度。

对于 16 位的 DSP 而言, Q 数定义共有 16 种, 其简化写法分别是 Q15、Q14、Q13、Q12、Q11、Q10、Q9、Q8、Q7、Q6、Q5、Q4、Q3、Q2、Q1、Q0, 其数学含义可以在其标准定义中更加明确, 分别应当是: Q1. 15、Q2. 14、Q3. 13、Q4. 12、Q5. 11、Q6. 10、Q7. 9、Q8. 8、Q9. 7、Q10. 6、Q11. 5、Q12. 4、Q13. 3、Q14. 2、Q15. 1、Q16. 0, 即标准形式为 Qn:m 其数学意义是 Q 数的最大整数的绝对值 $\leq 2^{n-1}$; 例如 Q15 其整数位小于等于 1; Q14 的整数位小于等于 2; Q13 的整数位小于等于 4; ...Q1 的整数位小于等于 16384; Q0 的整数位小于等于 32768。其最大整数位数是 n 位。小数位的最小刻度为 $=2^{-m}$, 由 m 位二进制数表示。例如: Q15 小数位的最小刻度为 $=2^{-m}=2^{-15}=3.0518 \times 10^{-5}$, Q14 小数位的最小刻度为 $=2^{-m}=2^{-14}=6.1035 \times 10^{-5}$, ...Q1 小数位的最小刻度为 $=2^{-m}=2^{-1}=0.5$ 。

因此我们上面使用的Q1.15格式(也称Q15), 就是用15位来表示小数部分, 最高位是符号位。浮点数转化为Q15, 要将数据乘以2的15次方。Q15数据转化为浮点数, 将数据除以2的15次方。

更多可以看这两篇文章: [文章1](#) [文章2](#), 不想深究的, 只需要导致浮点数和Q格式数的转换方法即可。

因此得到A、B相零电流的过程如下:

已知: 寄存器中的值/4096*Vref 是实际采样电阻上的电压值。hPhaseAOffset、hPhaseBOffset是16个零电流值的和。hPhaseAOffset、hPhaseBOffset是U16类型。

于是A相零电流的Q15格式值为: $(hPhaseAOffset/16)/4096*Vref*2^{15}=(((hPhaseAOffset>>4)>>12)*Vref)<<15$, 当Vref=2V, A相零电流的Q15格式值就刚好等于hPhaseAOffset。为方便计算, 我们就统一Vref=2了。(在程序中, Vref取多少都没有关系, 只要统一就行)

三、编码器的配置

```
1. #define U32_MAX ((u32)4294967295uL)
2. #define POLE_PAIR_NUM (u8)2
3. #define ENCODER_PPR (u16)(1000)
4. #define ALIGNMENT_ANGLE (u16)90
5. #define COUNTER_RESET (u16)((((s32)(ALIGNMENT_ANGLE)*4*ENCODER_PPR/360)-1)/POLE_PAIR_NUM)
6. #define ICx_FILTER (u8) 8
7.
8. void ENC_Init(void)
9. {
10.     TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
11.     TIM_ICInitTypeDef TIM_ICInitStructure;
```

```

12.     GPIO_InitTypeDef GPIO_InitStructure;
13.     NVIC_InitTypeDef NVIC_InitStructure;
14.
15.     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
16.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
17.
18.     GPIO_StructInit(&GPIO_InitStructure);
19.     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
20.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
21.     GPIO_Init(GPIOA, &GPIO_InitStructure);
22.
23.
24.     NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
25.     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
26.     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
27.     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
28.     NVIC_Init(&NVIC_InitStructure);
29.
30.     TIM_DeInit(TIM2);
31.     TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
32.     TIM_TimeBaseStructure.TIM_Prescaler = 0x0;
33.
34.     TIM_TimeBaseStructure.TIM_Period = (4*ENCODER_PPR)-1;
35.     TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
36.     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
37.     TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
38.
39.     TIM_EncoderInterfaceConfig(TIM2, TIM_EncoderMode_TI12, TIM_ICPolarity_Rising, TIM_ICPolarity_Rising);
40.
41.     TIM_ICStructInit(&TIM_ICInitStructure);
42.     TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
43.     TIM_ICInitStructure.TIM_ICFilter = ICx_FILTER;
44.     TIM_ICInit(TIM2, &TIM_ICInitStructure);
45.
46.     TIM_ICInitStructure.TIM_Channel = TIM_Channel_2;
47.     TIM_ICInit(TIM2, &TIM_ICInitStructure);
48.
49.     TIM_ClearFlag(TIM2, TIM_FLAG_Update);
50.     TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
51.     TIM2->CNT = COUNTER_RESET;
52.     TIM_Cmd(TIM2, ENABLE);
53. }
54.
55. void TIM2_IRQHandler(void)
56. {
57.     TIM_ClearFlag(TIM2, TIM_FLAG_Update);
58.
59. }
60.
61.
62.
63.
64.
65.
66.
67. s16 ENC_Get_Electrical_Angle(void)
68. {
69.     s32 temp;
70.
71.     temp = (s32)(TIM_GetCounter(TIM2)) * (s32)(U32_MAX / (4*ENCODER_PPR));
72.     temp *= POLE_PAIR_NUM;
73.     return((s16)(temp/65536));
74. }

```

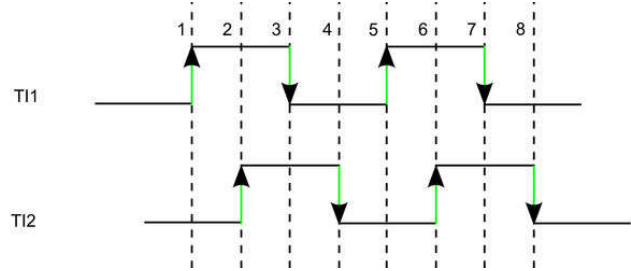
1、定时器的编码器模式只用CH1、CH2，因此把编码器的A、B信号接在TIM2_CH1、TIM2_CH2即可。

2、编码器的模式选择TIM_EncoderMode_TI12，即TI1和TI2都要计数，用两张图就能说明白编码器模式的原理了：

Table 92. Counting direction versus encoder signals					
Active edge	Level on opposite signal (TI1 for TI2, TI2 for TI1)	TI1		TI2	
		Rising	Falling	Rising	Falling
Counting on TI1 only	High	Down	Up	No Count	No Count
	Low	Up	Down	No Count	No Count
Counting on TI2 only	High	No Count	No Count	Up	Down
	Low	No Count	No Count	Down	Up
Counting on TI1 and TI2	High	Down	Up	Up	Down
	Low	Up	Down	Down	Up

TI1和TI2对应了编码器的A、B信号，第一列是指编码器的3种模式，即只计数TI1、只计数TI2和都要计数。第二列是相对信号的电平，比如我们讨论TI1的边沿，它的相对信号就是TI2。

我们就可以来看了，首先看第二行，只计数TI1信号的模式。当它的相对信号(即TI2)是高电平时，如果TI1来一个上升沿，那么CNT就要向下计数(Down)，如果来一个下降沿，那么CNT就要向上计数(Up)。示意图如下：



在时刻1，当TI2为低电平时，TI1来一个上升沿，CNT向上计数1。

在时刻3，当TI2为高电平时，TI1来一个下降沿，CNT又向上计数1。

同样的道理你也可以理解只计数TI2信号的模式，然后把这两个模式加起来你也可以理解同时计数TI1和TI2的模式。

3、TIM_ICPolarity_Rising 表示极性不反相。TIM_ICPolarity_falling:表示极性反相。库函数中的配置代码是这样的：

```

1.
2.  tmpccer &= (uint16_t)((uint16_t)~((uint16_t)TIM_CCER_CC1P)) & ((uint16_t)~((uint16_t)TIM_CCER_CC2P));
3.  tmpccer |= (uint16_t)(TIM_IC1Polarity | (uint16_t)(TIM_IC2Polarity << (uint16_t)4));
4.
5.  TIMx->CCER = tmpccer;

```

配置选项有3种：

```

1. #define TIM_ICPolarity_Rising      ((uint16_t)0x0000)
2. #define TIM_ICPolarity_Falling    ((uint16_t)0x0002)
3. #define TIM_ICPolarity_BothEdge   ((uint16_t)0x000A)

```

寄存器：

13.4.9 TIM1 和TIM8 捕获/比较使能寄存器(TIMx_CCER)

偏移地址：0x20

复位值：0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E	

实际配置的是CC1P、CC1NP、CC2P、CC2NP位，改位描述如下：

CC1P：输入/捕获1输出极性 (Capture/Compare 1 output polarity)

CC1通道配置为输出：

0：OC1高电平有效；

1：OC1低电平有效。

CC1通道配置为输入：

该位选择是IC1还是IC1的反相信号作为触发或捕获信号。

0：不反相：捕获发生在IC1的上升沿；当用作外部触发器时，IC1不反相。

1：反相：捕获发生在IC1的下降沿；当用作外部触发器时，IC1反相。

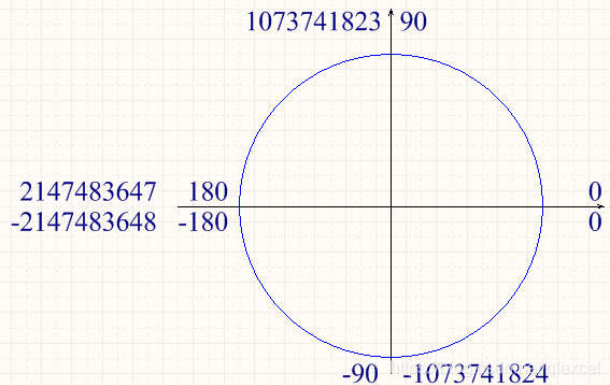
注：一旦LOCK级别(TIMx_BDTR寄存器中的LOCK位)设为3或2，则该位不能被修改。

4、ENC_Get_Electrical_Angle函数的问题

1)、首先是S16_MAX对应180度、S16_MIN对应-180的问题。这里其实就是利用了u32和s32数据类型表示的范围不同，而巧妙的产生了负数。我们知道u32的范围是：0到4294967295。而s32的范围是：-2147483648到2147483647。

(s32)(U32_MAX / (4*ENCODER_PPR))就是把4294967295分成4000份，如果CNT的值在0-2000，那么得到的结果最大也就是4294967295的一半，即2147483647，这时还没有超过s32的范围。当CNT=2001，得到2148555741，超过了s32范围，那么会怎么样呢？会从s32范围的最小值开始往上增加，就像一个环一样，最大值和最小值之间只差1。于是2148555741超出了2147483647：2148555741-2147483647=1072094，绕了一圈后得到：-2147483648+1072094=-2146411555。

有没有发现数据类型的这种特性和电机的电角度也是类似的？电机的电角度从0度增加到180度，然后再增加就变成-179度，又从-179度增加到0度，完成一圈。和数据类型的：从0增加到2147483647，再增加就变成-2147483648，又从-2147483648增加到0，完成一圈。



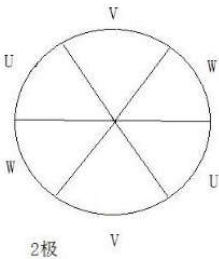
最后再把s32的数整除65536，就可以得到s16的数据类型了。（不能移位，会把符号位也移动了）

2)、极对数和电角度的关系

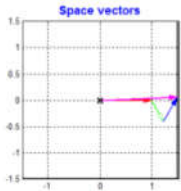
在函数还有一句：temp *= POLE_PAIR_NUM，即电角度要被极对数放大，这是为什么呢？

首先看极对数是什么：极对数是每相励磁绕组含有的磁极个数。

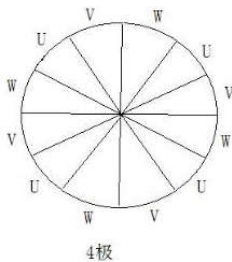
如果极对数是1，即每相只有一对磁极（一对磁极=2极，两对磁极=4极）



那么这3个相的磁极互差120度分布，相电流呈正弦规律变化一次，合成电压矢量旋转一圈，旋转磁场也会旋转一圈：



如果极对数是2，即每相有2对磁极：



那么这3个相的磁极互差60度分布，相电流呈正弦规律变化一次，合成电压矢量旋转半圈，旋转磁场也旋转半圈：

（此处差一个gif。。。有没有谁知道上面那种gif怎么画的。。。）

因此SVPWM输入的电角度和电机转子的机械角度之间就有极对数的倍数关系了：

如果极对数是1，那么SVPWM输出的磁场旋转一圈，电机转子也旋转一圈，电角度和电机转子角度是一一对应的。

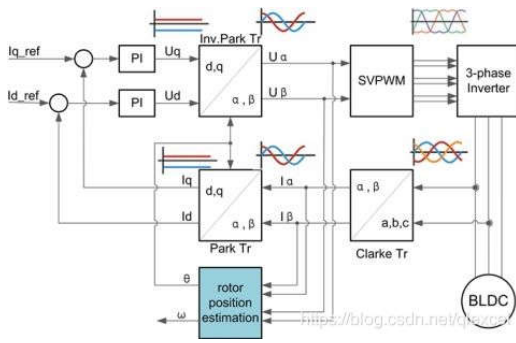
如果极对数是2，那么SVPWM输出的磁场旋转一圈，电机转子只旋转半圈，电角度是电机转子角度的2倍。

编码器的角度反映的是电机转子的机械角度。

后话：既然1对极电机就能转了，干嘛还要2对、4对呢？虽然极对数越多，转速越慢，但是扭矩可以越大。参考：<https://toutiao.1688.com/article/1067976.htm>

四、FOC相关变换的代码实现

前面中的代码中我们用TIM1的TRGO信号触发ADC注入组的转换。然后ADC的注入组转换完成后会产生中断，然后在中断函数对相电流进行采样，再经过clark变换把Ia, Ib, Ic变换成Ia, Ib, 再经过park变换，把Ia, Ib变换成Iq, Id。



获取相电流采样值、clark变换、park变换、反park变换的函数如下：

```

1. typedef struct
2. {
3.     s16 qI_Component1;
4.     s16 qI_Component2;
5. } Curr_Components;
6.
7. typedef struct
8. {
9.     s16 qV_Component1;
10.    s16 qV_Component2;
11. } Volt_Components;
12.
13. typedef struct
14. {
15.    s16 hCos;
16.    s16 hSin;
17. } Trig_Components;
18.
19. #define S16_MAX    ((s16)32767)
20. #define S16_MIN    ((s16)-32768)
21.
22.
23.
24.
25. Curr_Components GET_PHASE_CURRENTS(void)
26. {
27.    Curr_Components Local_Stator_Currents;
28.    s32 wAux;
29.
30.    wAux = ((ADC1->JDR1)<<1)-(s32)(hPhaseAOffset);
31.    if (wAux < S16_MIN)
32.        Local_Stator_Currents.qI_Component1= S16_MIN;
33.    else if (wAux > S16_MAX)
34.        Local_Stator_Currents.qI_Component1= S16_MAX;
35.    else
36.        Local_Stator_Currents.qI_Component1= wAux;
37.
38.    wAux = ((ADC2->JDR1)<<1)-(s32)(hPhaseBOffset);
39.    if (wAux < S16_MIN)
40.        Local_Stator_Currents.qI_Component2= S16_MIN;
41.    else if (wAux > S16_MAX)
42.        Local_Stator_Currents.qI_Component2= S16_MAX;
43.    else
44.        Local_Stator_Currents.qI_Component2= wAux;
45.
46.    return(Local_Stator_Currents);
47. }
48.
49. #define divSQRT_3    (s16)0x49E6
50.
51.
52.
53.
54. Curr_Components Clarke(Curr_Components Curr_Input)
55. {
56.    Curr_Components Curr_Output;
57.    s32 qIa_divSQRT3_tmp;
58.    s32 qIb_divSQRT3_tmp;
59.    s16 qIa_divSQRT3;

```

```

60.     s16 qIb_divSQRT3 ;
61.
62.     Curr_Output.qI_Component1 = Curr_Input.qI_Component1;
63.
64.     qIa_divSQRT3_tmp = divSQRT_3 * Curr_Input.qI_Component1;
65.     qIa_divSQRT3_tmp /=32768;
66.     qIb_divSQRT3_tmp = divSQRT_3 * Curr_Input.qI_Component2;
67.     qIb_divSQRT3_tmp /=32768;
68.
69.     qIa_divSQRT3=((s16)(qIa_divSQRT3_tmp));
70.     qIb_divSQRT3=((s16)(qIb_divSQRT3_tmp));
71.
72.     Curr_Output.qI_Component2=(-(qIa_divSQRT3)-(qIb_divSQRT3)-(qIb_divSQRT3));
73. return (Curr_Output);
74. }
75.
76. #define SIN_MASK    0x0300
77. #define U0_90       0x0200
78. #define U90_180     0x0300
79. #define U180_270    0x0000
80. #define U270_360    0x0100
81. const s16 hSin_Cos_Table[256] = { \
82. 0x0000,0x00C9,0x0192,0x025B,0x0324,0x03ED,0x04B6,0x057F,\
83. 0x0648,0x0711,0x07D9,0x08A2,0x096A,0x0A33,0x0AFB,0x0BC4,\
84. 0x0C8C,0x0D54,0x0E1C,0x0EE3,0x0FAB,0x1072,0x113A,0x1201,\
85. 0x12C8,0x138F,0x1455,0x151C,0x15E2,0x16A8,0x176E,0x1833,\
86. 0x18F9,0x19BE,0x1A82,0x1B47,0x1C0B,0x1CCF,0x1D93,0x1E57,\
87. 0x1F1A,0x1FDD,0x209F,0x2161,0x2223,0x22E5,0x23A6,0x2467,\
88. 0x2528,0x25E8,0x26A8,0x2767,0x2826,0x28E5,0x29A3,0x2A61,\
89. 0x2B1F,0x2BDC,0x2C99,0x2D55,0x2E11,0x2ECC,0x2F87,0x3041,\
90. 0x30FB,0x31B5,0x326E,0x3326,0x33DF,0x3496,0x354D,0x3604,\
91. 0x36BA,0x376F,0x3824,0x38D9,0x398C,0x3A40,0x3AF2,0x3BA5,\
92. 0x3C56,0x3D07,0x3DB8,0x3E68,0x3F17,0x3FC5,0x4073,0x4121,\
93. 0x41CE,0x427A,0x4325,0x43D0,0x447A,0x4524,0x45CD,0x4675,\
94. 0x471C,0x47C3,0x4869,0x490F,0x49B4,0x4A58,0x4AFB,0x4B9D,\
95. 0x4C3F,0x4CE0,0x4D81,0x4E20,0x4EBF,0x4F5D,0x4FFB,0x5097,\
96. 0x5133,0x51CE,0x5268,0x5302,0x539B,0x5432,0x54C9,0x5560,\
97. 0x55F5,0x568A,0x571D,0x57B0,0x5842,0x58D3,0x5964,0x59F3,\
98. 0x5A82,0x5B0F,0x5B9C,0x5C28,0x5CB3,0x5D3E,0x5DC7,0x5E4F,\
99. 0x5ED7,0x5F5D,0x5FE3,0x6068,0x60EB,0x616E,0x61F0,0x6271,\
100. 0x62F1,0x6370,0x63EE,0x646C,0x64E8,0x6563,0x65DD,0x6656,\
101. 0x66CF,0x6746,0x67BC,0x6832,0x68A6,0x6919,0x698B,0x69FD,\
102. 0x6A6D,0x6ADC,0x6B4A,0x6BB7,0x6C23,0x6C8E,0x6CF8,0x6D61,\
103. 0x6DC9,0x6E30,0x6E96,0x6EFB,0x6F5E,0x6FC1,0x7022,0x7083,\
104. 0x70E2,0x7140,0x719D,0x71F9,0x7254,0x72AE,0x7307,0x735E,\
105. 0x73B5,0x740A,0x745F,0x74B2,0x7504,0x7555,0x75A5,0x75F3,\
106. 0x7641,0x768D,0x76D8,0x7722,0x776B,0x77B3,0x77FA,0x783F,\
107. 0x7884,0x78C7,0x7909,0x794A,0x7989,0x79C8,0x7A05,0x7A41,\
108. 0x7A7C,0x7AB6,0x7AEE,0x7B26,0x7B5C,0x7B91,0x7BC5,0x7BF8,\
109. 0x7C29,0x7C59,0x7C88,0x7CB6,0x7CE3,0x7D0E,0x7D39,0x7D62,\
110. 0x7D89,0x7DB0,0x7DD5,0x7DFA,0x7E1D,0x7E3E,0x7E5F,0x7E7E,\
111. 0x7E9C,0x7EB9,0x7ED5,0x7EEF,0x7F09,0x7F21,0x7F37,0x7F4D,\
112. 0x7F61,0x7F74,0x7F86,0x7F97,0x7FA6,0x7FB4,0x7FC1,0x7FCD,\
113. 0x7FD8,0x7FE1,0x7FE9,0x7FF0,0x7FF5,0x7FF9,0x7FFD,0x7FFE};
114.
115.
116.
117.
118.
119.
120. Trig_Components Trig_Functions(s16 hAngle)
121. {
122.     u16 hindex;
123.     Trig_Components Local_Components;
124.
125.
126.     hindex = (u16)(hAngle + 32768);
127.     hindex /= 64;
128.
129. switch (hindex & SIN_MASK)
130. {
131. case U0_90:

```

```

132.     Local_Components.hSin = hSin_Cos_Table[(u8) (hindex)];
133.     Local_Components.hCos = hSin_Cos_Table[(u8) (0xFF-(u8) (hindex))];
134. break;
135.
136. case U90_180:
137.     Local_Components.hSin = hSin_Cos_Table[(u8) (0xFF-(u8) (hindex))];
138.     Local_Components.hCos = -hSin_Cos_Table[(u8) (hindex)];
139. break;
140.
141. case U180_270:
142.     Local_Components.hSin = -hSin_Cos_Table[(u8) (hindex)];
143.     Local_Components.hCos = -hSin_Cos_Table[(u8) (0xFF-(u8) (hindex))];
144. break;
145.
146. case U270_360:
147.     Local_Components.hSin = -hSin_Cos_Table[(u8) (0xFF-(u8) (hindex))];
148.     Local_Components.hCos = hSin_Cos_Table[(u8) (hindex)];
149. break;
150. default:
151. break;
152. }
153. return (Local_Components);
154. }
155.
156. Trig_Components Vector_Components;
157.
158.
159.
160. Curr_Components Park(Curr_Components Curr_Input, s16 Theta)
161. {
162.     Curr_Components Curr_Output;
163.     s32 qId_tmp_1, qId_tmp_2;
164.     s32 qIq_tmp_1, qIq_tmp_2;
165.     s16 qId_1, qId_2;
166.     s16 qIq_1, qIq_2;
167.
168.     Vector_Components = Trig_Functions(Theta);
169.
170.     qIq_tmp_1 = Curr_Input.qI_Component1 * Vector_Components.hCos;
171.     qIq_tmp_1 /= 32768;
172.     qIq_tmp_2 = Curr_Input.qI_Component2 * Vector_Components.hSin;
173.     qIq_tmp_2 /= 32768;
174.
175.     qIq_1 = ((s16) (qIq_tmp_1));
176.     qIq_2 = ((s16) (qIq_tmp_2));
177.     Curr_Output.qI_Component1 = ((qIq_1)-(qIq_2));
178.
179.     qId_tmp_1 = Curr_Input.qI_Component1 * Vector_Components.hSin;
180.     qId_tmp_1 /= 32768;
181.     qId_tmp_2 = Curr_Input.qI_Component2 * Vector_Components.hCos;
182.     qId_tmp_2 /= 32768;
183.
184.     qId_1 = (s16) (qId_tmp_1);
185.     qId_2 = (s16) (qId_tmp_2);
186.     Curr_Output.qI_Component2 = ((qId_1)+(qId_2));
187.
188. return (Curr_Output);
189. }
190.
191.
192.
193.
194. Volt_Components Rev_Park(Volt_Components Volt_Input)
195. {
196.     s32 qValpha_tmp1,qValpha_tmp2,qVbeta_tmp1,qVbeta_tmp2;
197.     s16 qValpha_1,qValpha_2,qVbeta_1,qVbeta_2;
198.     Volt_Components Volt_Output;
199.
200.     qValpha_tmp1 = Volt_Input.qV_Component1 * Vector_Components.hCos;
201.     qValpha_tmp1 /= 32768;
202.     qValpha_tmp2 = Volt_Input.qV_Component2 * Vector_Components.hSin;

```

```
203.  qValpha_tmp2 /= 32768;
204.
205.  qValpha_1 = (s16)(qValpha_tmp1);
206.  qValpha_2 = (s16)(qValpha_tmp2);
207.  Volt_Output.qV_Component1 = ((qValpha_1)+(qValpha_2));
208.
209.  qVbeta_tmp1 = Volt_Input.qV_Component1 * Vector_Components.hSin;
210.  qVbeta_tmp1 /= 32768;
211.  qVbeta_tmp2 = Volt_Input.qV_Component2 * Vector_Components.hCos;
212.  qVbeta_tmp2 /= 32768;
213.
214.  qVbeta_1 = (s16)(qVbeta_tmp1);
215.  qVbeta_2 = (s16)(qVbeta_tmp2);
216.  Volt_Output.qV_Component2 = -(qVbeta_1)+(qVbeta_2);
217.
218. return(Volt_Output);
219. }
```

1、读取ADC注入组的转换值：ADC1->JDR1和ADC2->JDR1。数据的对齐还是满足下表的关系：

数据右对齐

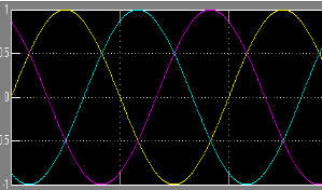
注入组				SEX	SEX	SEX	SEX	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
SEX	SEX	SEX	SEX	T	T	T	T	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
规则组	规则组	规则组	规则组	0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

数据左对齐

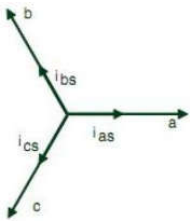
注入组				SEX	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0
SEX	SEX	SEX	SEX	T	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0
规则组	规则组	规则组	规则组	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0

我们是注入组左对齐，因此右移3位才是真实的转换值。电流采样值的Q15格式为：(((ADC1->JDR1<<1)>>4)/4096*Vref*2^15=(((ADC1->JDR1<<1)>>4)>>12)*Vref)<<15=ADC1->JDR1<<1.

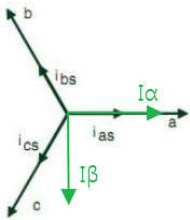
2、用ADC采集电机的两相电流，即可通过Ia+Ib+Ic=0得到第三相的电流。电机的三相电流是在时间上互差120度，成正弦规律变化的波形。电流波形如下：



表示为矢量：



我们通过clark变换，把Ia、Ib、Ic变换为Iα、Iβ：



变换公式：（可以当成矢量分解来理解）

$$\begin{cases} I_{\alpha} = k(I_a - \frac{1}{2}I_b - \frac{1}{2}I_c) \\ I_{\beta} = k(\frac{\sqrt{3}}{2}I_c - \frac{\sqrt{3}}{2}I_b) \end{cases}$$

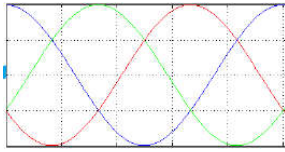
公式中有一个系数k，一般取2/3。有兴趣可以去这儿了解：<https://blog.csdn.net/daidi1989/article/details/89926324>。带入2/3：

$$\begin{cases} I_{\alpha} = \frac{2}{3}I_a - \frac{1}{3}I_b - \frac{1}{3}I_c \\ I_{\beta} = \frac{\sqrt{3}}{3}I_c - \frac{\sqrt{3}}{3}I_b \end{cases}$$

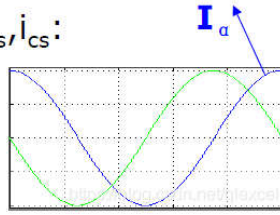
再代入 $I_a+I_b+I_c=0$:

$$\begin{cases} I_\alpha = I_a \\ I_\beta = -\frac{I_a+2I_b}{\sqrt{3}} \end{cases}$$

 Clarke变换应用于定子电流: i_{as}, i_{bs}, i_{cs} :



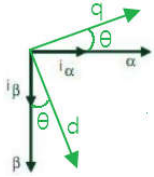
$$\begin{aligned} i_\alpha &= i_{as} \\ i_\beta &= -\frac{i_{as} + 2i_{bs}}{\sqrt{3}} \end{aligned}$$



注: ST的FOC库, 采用的 I_β 轴(即虚轴)滞后 I_α 90度的表示方法。与其他地方说的 I_β 提前 I_α 90度是一样的效果。只是表示方法不同而已, 选择了这种表示方法, 后面的park变换也和提前90度的不同。


3、Trig_Functions函数, 使用查表的方法计算目标角度的三角函数。速度快, 精度也还可以, 读者可以自己测试一下精度。

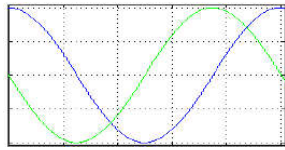
4、使用park变换将电流 I_α 、 I_β 和转子的电角度 θ 转化为电流 I_q 、 I_d 。



公式为:

$$\begin{aligned} i_q &= i_\alpha \cos \theta - i_\beta \sin \theta \\ i_d &= i_\alpha \sin \theta + i_\beta \cos \theta \end{aligned}$$


 Park变换应用于定子电流: i_α, i_β :

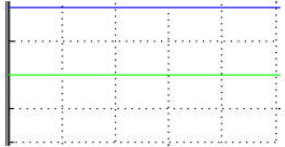


$$\begin{aligned} i_{qs} &= i_\alpha \cos \theta_r - i_\beta \sin \theta_r \\ i_{ds} &= i_\alpha \sin \theta_r + i_\beta \cos \theta_r \end{aligned}$$

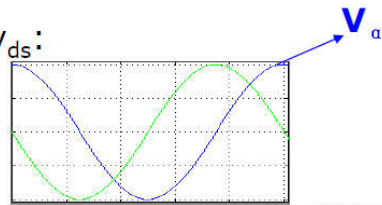


5、反park变换就是上面park变换的逆过程, 公式就不推了。

 反Park变换应用于定子电压: v_{qs}, v_{ds} :



$$\begin{aligned} v_\alpha &= v_{qs} \cos \theta_r + v_{ds} \sin \theta_r \\ v_\beta &= -v_{qs} \sin \theta_r + v_{ds} \cos \theta_r \end{aligned}$$



五、PID控制

```
1. typedef struct
2. {
3.     s16 hKp_Gain;                //比例系数
4.     u16 hKp_Divisor;            //比例系数因子
5.     s16 hKi_Gain;                //积分系数
6.     u16 hKi_Divisor;            //积分系数因子
7.     s16 hLower_Limit_Output;    //总输出下限
8.     s16 hUpper_Limit_Output;    //总输出上限
9.     s32 wLower_Limit_Integral;  //积分项下限
10.    s32 wUpper_Limit_Integral;  //积分项上限
11.    s32 wIntegral;               //积分累积和
12.    s16 hKd_Gain;                //微分系数
13.    u16 hKd_Divisor;            //微分系数因子
14.    s32 wPreviousError;          //上次误差
15. } PID_Struct_t;
16.
17. /***** 扭矩的PID参数, 即q轴 *****/
18. #define PID_TORQUE_REFERENCE    (s16)3000 //q轴的设定值, PID的目的就是要让测量的q轴值与设定值误差为0
19. #define PID_TORQUE_KP_DEFAULT   (s16)1578 //Kp默认值
20. #define PID_TORQUE_KI_DEFAULT   (s16)676  //Ki默认值
21. #define PID_TORQUE_KD_DEFAULT   (s16)100  //Kd默认值
22.
23. /***** 转子磁通的PID参数, 即d轴 *****/
```

```

24. #define PID_FLUX_REFERENCE    (s16)0          //d轴的设定值
25. #define PID_FLUX_KP_DEFAULT   (s16)1578
26. #define PID_FLUX_KI_DEFAULT   (s16)676
27. #define PID_FLUX_KD_DEFAULT   (s16)100
28.
29. /***** q轴和d轴PID参数的放大倍数 *****/
30. #define TF_KP_DIV ((u16)(1024))    //因为Kp、Ki、Kd值很小，而我们需要整数计算，所以需要放大。得出计算结果之后，再缩小。
31. #define TF_KI_DIV ((u16)(16384))
32. #define TF_KD_DIV ((u16)(8192))
33.
34. /***** 速度环的PID参数 *****/
35. #define PID_SPEED_REFERENCE_RPM (s16)1500      //电机的设定转速
36. #define PID_SPEED_REFERENCE      (u16)(PID_SPEED_REFERENCE_RPM/6) //电机转速和速度环的设定值一般都不相等，电机不同，它们的关系也不同
37. #define PID_SPEED_KP_DEFAULT     (s16)300
38. #define PID_SPEED_KI_DEFAULT     (s16)100
39. #define PID_SPEED_KD_DEFAULT     (s16)0000
40. #define NOMINAL_CURRENT          (s16)5289     //motor nominal current (0-pk), 3倍的额定电流
41. #define IQ_MAX                   NOMINAL_CURRENT //速度环输出最大值
42.
43. /***** 速度环PID参数的放大倍数 *****/
44. #define SP_KP_DIV ((u16)(16))
45. #define SP_KI_DIV ((u16)(256))
46. #define SP_KD_DIV ((u16)(16))
47.
48. volatile s16 hTorque_Reference; //q轴设定值
49. volatile s16 hFlux_Reference;  //d轴设定值
50. volatile s16 hSpeed_Reference; //速度环设定值
51. void PID_Init (PID_Struct_t *PID_Torque, PID_Struct_t *PID_Flux, PID_Struct_t *PID_Speed)
52. {
53.     hTorque_Reference = PID_TORQUE_REFERENCE; //q轴设定值初始化
54. /***** 下面是控制扭矩的PID参数，即q轴大小 *****/
55.     PID_Torque->hKp_Gain = PID_TORQUE_KP_DEFAULT; //Kp参数，放大了hKp_Divisor倍。调节结果除以hKp_Divisor才是真实结果
56.     PID_Torque->hKp_Divisor = TF_KP_DIV; //Kp参数分数因子
57.     PID_Torque->hKi_Gain = PID_TORQUE_KI_DEFAULT; //Ki参数
58.     PID_Torque->hKi_Divisor = TF_KI_DIV; //Ki参数分数因子
59.     PID_Torque->hKd_Gain = PID_TORQUE_KD_DEFAULT; //Kd参数
60.     PID_Torque->hKd_Divisor = TF_KD_DIV; //Kd参数分数因子
61.     PID_Torque->wPreviousError = 0; //上次计算的误差值，用于D调节
62.     PID_Torque->hLower_Limit_Output = S16_MIN; //PID输出下限幅
63.     PID_Torque->hUpper_Limit_Output = S16_MAX; //PID输出上限幅
64.     PID_Torque->wLower_Limit_Integral = S16_MIN * TF_KI_DIV; //I调节的下限幅
65.     PID_Torque->wUpper_Limit_Integral = S16_MAX * TF_KI_DIV; //I调节的上限幅
66.     PID_Torque->wIntegral = 0; //I调节的结果，因为是积分，所以要一直累积
67. /***** 上面是控制扭矩的PID参数，即q轴大小 *****/
68.
69.     hFlux_Reference = PID_FLUX_REFERENCE; //对于SM-PMSM电机，Id = 0
70. /***** 下面是控制转子磁通的PID参数，即d轴大小 *****/
71.     PID_Flux->hKp_Gain = PID_FLUX_KP_DEFAULT;
72.     PID_Flux->hKp_Divisor = TF_KP_DIV;
73.     PID_Flux->hKi_Gain = PID_FLUX_KI_DEFAULT;
74.     PID_Flux->hKi_Divisor = TF_KI_DIV;
75.     PID_Flux->hKd_Gain = PID_FLUX_KD_DEFAULT;
76.     PID_Flux->hKd_Divisor = TF_KD_DIV;
77.     PID_Flux->wPreviousError = 0;
78.     PID_Flux->hLower_Limit_Output = S16_MIN;
79.     PID_Flux->hUpper_Limit_Output = S16_MAX;
80.     PID_Flux->wLower_Limit_Integral = S16_MIN * TF_KI_DIV;
81.     PID_Flux->wUpper_Limit_Integral = S16_MAX * TF_KI_DIV;
82.     PID_Flux->wIntegral = 0;
83. /***** 上面是控制转子磁通的PID参数，即d轴大小 *****/
84.
85.     hSpeed_Reference = PID_SPEED_REFERENCE;
86. /***** 下面是速度环的PID参数 *****/
87.     PID_Speed->hKp_Gain = PID_SPEED_KP_DEFAULT;
88.     PID_Speed->hKp_Divisor = SP_KP_DIV;
89.     PID_Speed->hKi_Gain = PID_SPEED_KI_DEFAULT;
90.     PID_Speed->hKi_Divisor = SP_KI_DIV;
91.     PID_Speed->hKd_Gain = PID_SPEED_KD_DEFAULT;

```



```

92.     PID_Speed->hKd_Divisor = SP_KDDIV;
93.     PID_Speed->wPreviousError = 0;
94.     PID_Speed->hLower_Limit_Output= -IQMAX;
95.     PID_Speed->hUpper_Limit_Output= IQMAX;
96.     PID_Speed->wLower_Limit_Integral = -IQMAX * SP_KIDIV;
97.     PID_Speed->wUpper_Limit_Integral = IQMAX * SP_KIDIV;
98.     PID_Speed->wIntegral = 0;
99.  /****** 上面是速度环的PID参数 *****/
100. }
101.
102. //define DIFFERENTIAL_TERM_ENABLED    //不使用PID的D调节
103. typedef signed long long s64;
104. s16 PID_Regulator(s16 hReference, s16 hPresentFeedback, PID_Struct_t *PID_Struct)
105. {
106.     s32 wError, wProportional_Term, wIntegral_Term, houtput_32;
107.     s64 dwAux;
108.     #ifdef DIFFERENTIAL_TERM_ENABLED    //如果使能了D调节
109.         s32 wDifferential_Term;
110.     #endif
111.
112.     wError= (s32)(hReference - hPresentFeedback);    //设定值-反馈值, 取得需要误差量delta_e
113.     wProportional_Term = PID_Struct->hKp_Gain * wError; //PID的P调节, 即比例放大调节: wP = Kp * delta_e
114.
115.     if (PID_Struct->hKi_Gain == 0)    //下面进行PID的I调节, 即误差的累积调节
116.     {
117.         PID_Struct->wIntegral = 0;    //如果I参数=0, I调节就=0
118.     }
119.     else
120.     {
121.         wIntegral_Term = PID_Struct->hKi_Gain * wError;    //wI = Ki * delta_e , 本次积分项
122.         dwAux = PID_Struct->wIntegral + (s64)(wIntegral_Term); //积分累积的调节量 = 以前的积分累积量 + 本次的积分项
123.
124.         if (dwAux > PID_Struct->wUpper_Limit_Integral)    //对PID的I调节做限幅
125.         {
126.             PID_Struct->wIntegral = PID_Struct->wUpper_Limit_Integral; //上限
127.         }
128.         else if (dwAux < PID_Struct->wLower_Limit_Integral)    //下限
129.         {
130.             PID_Struct->wIntegral = PID_Struct->wLower_Limit_Integral;
131.         }
132.         else
133.         {
134.             PID_Struct->wIntegral = (s32)(dwAux);    //不超限, 更新积分累积项为dwAux
135.         }
136.     }
137.     #ifdef DIFFERENTIAL_TERM_ENABLED    //如果使能了D调节
138.     {
139.         s32 wtemp;
140.
141.         wtemp = wError - PID_Struct->wPreviousError;    //取得上次和这次的误差之差
142.         wDifferential_Term = PID_Struct->hKd_Gain * wtemp;    //D调节结果, wD = Kd * delta_d
143.         PID_Struct->wPreviousError = wError;    //更新上次误差, 用于下次运算
144.     }
145.
146.     houtput_32 = (wProportional_Term/PID_Struct->hKp_Divisor+    //输出总的调节量 = 比例调节量/分数因子 +
147.         PID_Struct->wIntegral/PID_Struct->hKi_Divisor + //    + 积分调节量/分数因子
148.         wDifferential_Term/PID_Struct->hKd_Divisor);    //    + 微分调节量/分数因子
149.
150. #else
151.     //把P调节和I调节结果除以分数因子再相加, 得到PI控制的结果
152.     houtput_32 = (wProportional_Term/PID_Struct->hKp_Divisor + PID_Struct->wIntegral/PID_Struct->hKi_Divisor);
153. #endif
154.     if (houtput_32 >= PID_Struct->hUpper_Limit_Output)    //PI控制结果限幅
155.     {
156.         return(PID_Struct->hUpper_Limit_Output);
157.     }
158.     else if (houtput_32 < PID_Struct->hLower_Limit_Output) //下限
159.     {
160.         return(PID_Struct->hLower_Limit_Output);

```

```

161.     }
162.     else
163.     {
164.         return((s16)(houtput_32)); //不超限。输出结果 houtput_32
165.     }
166. }

```

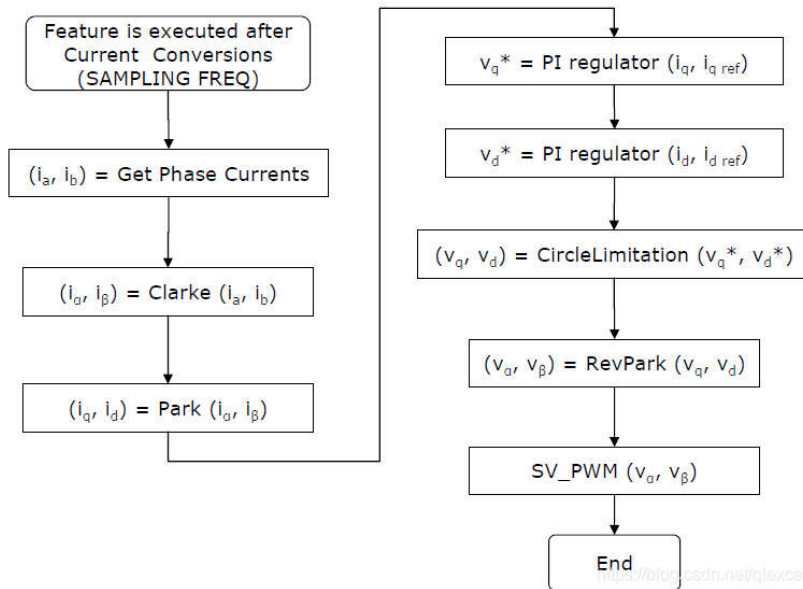
PID控制没什么好说的，网上资料很多。此处用的PID也很普通，很容易看懂。

使用了宏定义，选择是否使用PID的D调节。一般PI控制就已经足够。

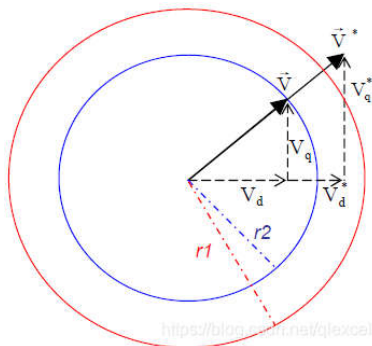
6、使用RevPark_Circle_Limitation函数对PID的输出进行归一化

看下面的FOC的处理过程：

FOC_Model (MC_FOC_DRIVE.c)



可以发现在PID计算后，要进行CircleLimitation处理，为什么呢？因为我们需要控制旋转磁场大小的恒定，如果d、q轴给得太大了，那么输出的旋转磁场就会过调制，圆形磁场就会凸起来一块。PID控制器只能对单独的d、q轴大小进行限制，可是它控制不了d、q轴合成矢量的大小。比如：最大值要求限制在1，d=0.8，q=0.9，他们各自的大小都没有超过1，可是它们的合成矢量大小却超过了1。示意图如下，就是要用比例关系，把外圆上的d、q值等比例缩小到内圆上：



```

1.  /***** 根据载波频率来选择调制系数，频率越大，调制系数越小（实质是控制的最大占空比） *****/
2.  //define MAX_MODULATION_100_PER_CENT // up to 11.4 kHz PWM frequency
3.  //define MAX_MODULATION_99_PER_CENT // up to 11.8 kHz
4.  //define MAX_MODULATION_98_PER_CENT // up to 12.2 kHz
5.  //define MAX_MODULATION_97_PER_CENT // up to 12.9 kHz
6.  //define MAX_MODULATION_96_PER_CENT // up to 14.4 kHz
7.
8.  //define MAX_MODULATION_94_PER_CENT // up to 15.2 kHz
9.  //define MAX_MODULATION_93_PER_CENT // up to 16.7 kHz
10. //define MAX_MODULATION_92_PER_CENT // up to 17.1 kHz
11. //define MAX_MODULATION_89_PER_CENT // up to 17.5 kHz
12.
13. /***** 以下是根据选择的调制系数，计算d、q轴合成矢量模的最大值。用宏定义，优化计算速度 *****/
14.
15.
16.
17.
18.
19.
20.
21.
22.
23.
24.
25.
26.
27.
28.
29.

```

```
30.
31.
32.
33.
34.
35.
36.
37.
38.
39.
40.
41.
42.
43.
44.
45.
46.
47.
48.
49.
50.
51.
52.
53.
54.
55.
56.
57.
58.
59.
60.
61.
62.
63.
64.
65.
66.
67.
68.
69.
70.
71.
72.
73.
74.
75.
76.
77.
78.
79.
80.
81.
82.
83.

84. static const u16 circle_limit_table[91]=

85. {

86. 32468,32176,31893,31347,31084,30578,30334,29863,29636,29414,28984,28776,28373,

87. 28178,27987,27616,27436,27086,26916,26585,26425,26267,25959,25809,25518,25375,

88. 25098,24962,24829,24569,24442,24194,24072,23953,23719,23604,23381,23271,23056,

89. 22951,22848,22644,22545,22349,22254,22066,21974,21883,21704,21616,21444,21359,

90. 21275,21111,21030,20872,20793,20640,20564,20490,20343,20270,20128,20058,19920,

91. 19852,19785,19653,19587,19459,19396,19271,19209,19148,19028,18969,18852,18795,

92. 18738,18625,18570,18460,18406,18299,18246,18194,18091,18040,17940,17890,17792

93. };

94.
95.
96.
97.

98. static const u16 circle_limit_table[89]=

99. {

100. 32489,32217,31952,31442,31195,30719,30489,30045,29830,29413,29211,28819,28629,

101. 28442,28080,27904,27562,27396,27072,26914,26607,26457,26165,26022,25882,25608,

102. 25475,25214,25086,24836,24715,24476,24359,24130,24019,23908,23692,23586,23378,

103. 23276,23077,22979,22787,22692,22507,22416,22326,22150,22063,21893,21809,21645,

104. 21564,21405,21327,21173,21097,21022,20875,20802,20659,20589,20450,20382,20247,

105. 20181,20051,19986,19923,19797,19735,19613,19553,19434,19375,19260,19202,19090,

106. 19034,18979,18871,18817,18711,18659,18555,18504,18403,18354,18255

107. };

108.
109.
110.
111.

112. static const u16 circle_limit_table[87]=

113. {

114. 32508,32255,32008,31530,31299,30852,30636,30216,30012,29617,29426,29053,28872,

115. 28520,28349,28015,27853,27536,27382,27081,26934,26647,26507,26234,26101,25840,

116. 25712,25462,25340,25101,24984,24755,24643,24422,24315,24103,24000,23796,23696,

117. 23500,23404,23216,23123,22941,22851,22763,22589,22504,22336,22253,22091,22011,

118. 21854,21776,21624,21549,21401,21329,21186,21115,20976,20908,20773,20706,20575,

119. 20511,20383,20320,20196,20135,20015,19955,19838,19780,19666,19609,19498,19443,

120. 19334,19280,19175,19122,19019,18968,18867,18817,18719

121. };

122.
123.
124.
125.

126. static const u16 circle_limit_table[84]=

127. {
```

```
128. 32291,32060,31613,31397,30977,30573,30377,29996,29811,29451,29276,28934,28768,
129. 28444,28286,27978,27827,27533,27390,27110,26973,26705,26574,26318,26069,25948,
130. 25709,25592,25363,25251,25031,24923,24711,24607,24404,24304,24107,24011,23821,
131. 23728,23545,23456,23279,23192,23021,22854,22772,22610,22530,22374,22297,22145,
132. 22070,21922,21850,21707,21636,21497,21429,21294,21227,21096,21032,20904,20778,
133. 20717,20595,20534,20416,20357,20241,20184,20071,20015,19905,19851,19743,19690,
134. 19585,19533,19431,19380,19280,19182
135. };
136.
137.
138.
139.
140. static const u16 circle_limit_table[82]=
141. {
142. 32324,32109,31691,31489,31094,30715,30530,30170,29995,29654,29488,29163,29005,
143. 28696,28397,28250,27965,27825,27552,27418,27157,26903,26779,26535,26416,26182,
144. 26067,25842,25623,25515,25304,25201,24997,24897,24701,24605,24415,24230,24139,
145. 23960,23872,23699,23614,23446,23282,23201,23042,22964,22810,22734,22584,22437,
146. 22365,22223,22152,22014,21945,21811,21678,21613,21484,21421,21296,21234,21112,
147. 21051,20932,20815,20757,20643,20587,20476,20421,20312,20205,20152,20048,19996
148. ,19894,19844,19744,19645
149. };
150.
151.
152.
153.
154. static const u16 circle_limit_table[81]=
155. {
156. 32559,32154,31764,31575,31205,31025,30674,30335,30170,29847,
157. 29689,29381,29083,28937,28652,28375,28239,27974,27844,27589,
158. 27342,27220,26983,26866,26637,26414,26305,26090,25984,25777,
159. 25575,25476,25280,25184,24996,24811,24720,24542,24367,24281,
160. 24112,24028,23864,23703,23624,23468,23391,23240,23091,23018,
161. 22874,22803,22662,22524,22456,22322,22191,22126,21997,21934,
162. 21809,21686,21625,21505,21446,21329,21214,21157,21045,20990,
163. 20880,20772,20719,20613,20561,20458,20356,20306,20207,20158,
164. 20109
165. };
166.
167.
168.
169.
170. static const u16 circle_limit_table[78]=
171. {
172. 32574,32197,32014,31656,31309,31141,30811,30491,30335,30030,
173. 29734,29589,29306,29031,28896,28632,28375,28249,28002,27881,
174. 27644,27412,27299,27076,26858,26751,26541,26336,26235,26037,
175. 25844,25748,25561,25378,25288,25110,24936,24851,24682,24517,
176. 24435,24275,24118,24041,23888,23738,23664,23518,23447,23305,
177. 23166,23097,22962,22828,22763,22633,22505,22442,22318,22196,
178. 22135,22016,21898,21840,21726,21613,21557,21447,21338,21284,
179. 21178,21074,21022,20919,20819,20769,20670,20573
180. };
181.
182.
183.
184.
185. static const u16 circle_limit_table[76]=
186. {
187. 32588,32411,32066,31732,31569,31250,30940,30789,30492,30205,
188. 29925,29788,29519,29258,29130,28879,28634,28395,28278,28048,
189. 27823,27713,27497,27285,27181,26977,26777,26581,26485,26296,
190. 26110,26019,25840,25664,25492,25407,25239,25076,24995,24835,
191. 24679,24602,24450,24301,24155,24082,23940,23800,23731,23594,
192. 23460,23328,23263,23135,23008,22946,22822,22701,22641,22522,
193. 22406,22291,22234,22122,22011,21956,21848,21741,21636,21584,
194. 21482,21380,21330,21231,21133,21037
195. };
196.
197.
198.
199.
200.
201. const u16 circle_limit_table[74]=
202. {
203. 32424,32091,31929,31611,31302,31002,30855,30568,30289,30017,
204. 29884,29622,29368,29243,28998,28759,28526,28412,28187,27968,
```

```
205. 27753,27648,27441,27238,27040,26942,26750,26563,26470,26288,
206. 26110,25935,25849,25679,25513,25350,25269,25111,24955,24803,
207. 24727,24579,24433,24361,24219,24079,23942,23874,23740,23609,
208. 23479,23415,23289,23165,23042,22982,22863,22745,22629,22572,
209. 22459,22347,22292,22183,22075,21970,21917,21813,21711,21610,
210. 21561,21462,21365,21268
211. };
212.
213.
214.
215.
216. const ul6 circle_limit_table[73]=
217. {
218. 32437,32275,31959,31651,31353,31207,30920,30642,30371,30107,
219. 29977,29723,29476,29234,29116,28883,28655,28433,28324,28110,
220. 27900,27695,27594,27395,27201,27011,26917,26733,26552,26375,
221. 26202,26116,25948,25783,25621,25541,25383,25228,25076,25001,
222. 24854,24708,24565,24495,24356,24219,24084,24018,23887,23758,
223. 23631,23506,23444,23322,23202,23083,23025,22909,22795,22683,
224. 22627,22517,22409,22302,22250,22145,22042,21941,21890,21791,
225. 21693,21596,21500
226. };
227.
228.
229.
230.
231. const ul6 circle_limit_table[72]=
232. {
233. 32607,32293,31988,31691,31546,31261,30984,30714,30451,30322,
234. 30069,29822,29581,29346,29231,29004,28782,28565,28353,28249,
235. 28044,27843,27647,27455,27360,27174,26991,26812,26724,26550,
236. 26380,26213,26049,25968,25808,25652,25498,25347,25272,25125,
237. 24981,24839,24699,24630,24494,24360,24228,24098,24034,23908,
238. 23783,23660,23600,23480,23361,23245,23131,23074,22962,22851,
239. 22742,22635,22582,22477,22373,22271,22170,22120,22021,21924,
240. 21827,21732
241. };
242.
243.
244.
245.
246. const ul6 circle_limit_table[71]=
247. {
248. 32613,32310,32016,31872,31589,31314,31046,30784,30529,30404,
249. 30158,29919,29684,29456,29343,29122,28906,28695,28488,28285,
250. 28186,27990,27798,27610,27425,27245,27155,26980,26808,26639,
251. 26473,26392,26230,26072,25917,25764,25614,25540,25394,25250,
252. 25109,24970,24901,24766,24633,24501,24372,24245,24182,24058,
253. 23936,23816,23697,23580,23522,23408,23295,23184,23075,23021,
254. 22913,22808,22703,22600,22499,22449,22349,22251,22154,22059,
255. 21964
256. };
257.
258.
259.
260.
261. const ul6 circle_limit_table[70]=
262. {
263. 32619,32472,32184,31904,31631,31365,31106,30853,30728,30484,
264. 30246,30013,29785,29563,29345,29238,29028,28822,28620,28423,
265. 28229,28134,27946,27762,27582,27405,27231,27061,26977,26811,
266. 26649,26489,26332,26178,26027,25952,25804,25659,25517,25376,
267. 25238,25103,25035,24903,24772,24644,24518,24393,24270,24210,
268. 24090,23972,23855,23741,23627,23516,23461,23352,23244,23138,
269. 23033,22930,22828,22777,22677,22579,22481,22385,22290,22196
270. };
271.
272.
273.
274.
275. const ul6 circle_limit_table[68]=
276. {
277. 32483,32206,31936,31672,31415,31289,31041,30799,30563,30331,
278. 30105,29884,29668,29456,29352,29147,28947,28750,28557,28369,
279. 28183,28002,27824,27736,27563,27393,27226,27062,26901,26743,
280. 26588,26435,26360,26211,26065,25921,25780,25641,25504,25369,
```

```

281. 25236,25171,25041,24913,24788,24664,24542,24422,24303,24186,
282. 24129,24015,23902,23791,23681,23573,23467,23362,23258,23206,
283. 23105,23004,22905,22808,22711,22616,22521,22429
284. );
285.
286.
287.
288.
289. const u16 circle_limit_table[67]=
290. {
291. 32494,32360,32096,31839,31587,31342,31102,30868,30639,30415,
292. 30196,29981,29771,29565,29464,29265,29069,28878,28690,28506,
293. 28325,28148,27974,27803,27635,27470,27309,27229,27071,26916,
294. 26764,26614,26467,26322,26180,26039,25901,25766,25632,25500,
295. 25435,25307,25180,25055,24932,24811,24692,24574,24458,24343,
296. 24230,24119,24009,23901,23848,23741,23637,23533,23431,23331,
297. 23231,23133,23036,22941,22846,22753,22661
298. };
299.
300.
301.
302.
303.
304. const u16 circle_limit_table[66]=
305. {
306. 32635,32375,32121,31873,31631,31394,31162,30935,30714,30497,
307. 30284,30076,29872,29672,29574,29380,29190,29003,28820,28641,
308. 28464,28291,28122,27955,27791,27630,27471,27316,27163,27012,
309. 26864,26718,26575,26434,26295,26159,26024,25892,25761,25633,
310. 25569,25444,25320,25198,25078,24959,24842,24727,24613,24501,
311. 24391,24281,24174,24067,23963,23859,23757,23656,23556,23458,
312. 23361,23265,23170,23077,22984,22893
313. };
314.
315.
316.
317.
318. const u16 circle_limit_table[65]=
319. {
320. 32767,32390,32146,31907,31673,31444,31220,31001,30787,30577,30371,
321. 30169,29971,29777,29587,29400,29217,29037,28861,28687,28517,
322. 28350,28185,28024,27865,27709,27555,27404,27256,27110,26966,
323. 26824,26685,26548,26413,26280,26149,26019,25892,25767,25643,
324. 25521,25401,25283,25166,25051,24937,24825,24715,24606,24498,
325. 24392,24287,24183,24081,23980,23880,23782,23684,23588,23493,
326. 23400,23307,23215,23125
327. };
328.
329.
330. Volt_Components Stat_Volt_q_d;
331. /*****
332. 把经过PId调整过的d、q值的合成矢量模值限制在最大值，如果超过了，那么等比例缩小d、q值
333. *****/
334. void RevPark_Circle_Limitation(void)
335. {
336. s32 temp;
337. //计算Vd^2+Vq^2
338. temp = Stat_Volt_q_d.qV_Component1 * Stat_Volt_q_d.qV_Component1 + Stat_Volt_q_d.qV_Component2 * Stat_Volt_q_d.qV_Component2;
339.
340. if ( temp > (u32)(( MAX_MODULE * MAX_MODULE) ) ) //如果(Vd^2+Vq^2)大于MAX_MODULE^2，就要进行比例缩小
341. {
342. u16 index; //假设Vq=x*32767, Vd=y*32767
343.
344. temp /= (u32)(512*32768); // (Vq^2+Vd^2)/(512*32768) = (x^2+y^2)*32767^2/(512*32768)=64(x^2+y^2)
345. temp -= START_INDEX; //程序中，载波频率=15K，因此调制系数选择95%，MAX_MODULE=32767*95%=31128，START_INDEX=57
346. index = circle_limit_table[(u8)temp]; //当Vq、Vd接近于最大值32767，即x、y接近于1，temp=64(x^2+y^2)-START_INDEX=128-57有最大值71
347. //当Vd^2+Vq^2的值只比MAX_MODULE^2大一点，temp=(Vq^2+Vd^2)/(512*32768)-START_INDEX=MAX_MODULE^2/(512*32768)-57有最小值0
348. //因此调制系数为95%的circle_limit_table表中只有72个数，并且根据Vd^2+Vq^2的大小来选择缩小的系数
349. temp = (s16)Stat_Volt_q_d.qV_Component1 * (u16)(index); //使用缩小系数来缩小Vq
350. Stat_Volt_q_d.qV_Component1 = (s16)(temp/32768);
351.
352. temp = (s16)Stat_Volt_q_d.qV_Component2 * (u16)(index); //使用缩小系数来缩小Vd
353. Stat_Volt_q_d.qV_Component2 = (s16)(temp/32768);
354. }

```

355. }

上面的过程也没有什么好说的，具体可以看注释。为了优化处理速度，能用查表的地方都用查表了。

7、SVPWM的实现

根据FOC的处理过程，相电流采样后经过clark变换、park变换得到d、q轴值，然后和参考值做PID运算，再经过归一化，反park变换得到 V_α 、 V_β ，这就是FOC的处理结果，然后输出给SVPWM去执行，下面我们来看看SVPWM处理函数：

```
1. #define SQRT_3          1.732051
2. #define T                (PWM_PERIOD * 4)
3. #define T_SQRT3          (u16)(T * SQRT_3)
4. #define SECTOR_1          (u32)1
5. #define SECTOR_2          (u32)2
6. #define SECTOR_3          (u32)3
7. #define SECTOR_4          (u32)4
8. #define SECTOR_5          (u32)5
9. #define SECTOR_6          (u32)6
10.
11. void CALC_SVPWM(Volt_Components Stat_Volt_Input)
12. {
13.     u8 bSector;
14.     s32 wX, wY, wZ, wUAlpha, wUBeta;
15.     u16 hTimePhA=0, hTimePhB=0, hTimePhC=0;
16.
17.     wUAlpha = Stat_Volt_Input.qV_Component1 * T_SQRT3;
18.     wUBeta = -(Stat_Volt_Input.qV_Component2 * T);
19.
20.     wX = wUBeta;
21.     wY = (wUBeta + wUAlpha)/2;
22.     wZ = (wUBeta - wUAlpha)/2;
23.
24.
25. if (wY<0)
26. {
27. if (wZ<0)
28. {
29.         bSector = SECTOR_5;
30.     }
31. else
32. if (wX<=0)
33. {
34.         bSector = SECTOR_4;
35.     }
36. else
37. {
38.         bSector = SECTOR_3;
39.     }
40. }
41. else
42. {
43. if (wZ>=0)
44. {
45.         bSector = SECTOR_2;
46.     }
47. else
48. if (wX<=0)
49. {
50.         bSector = SECTOR_6;
51.     }
52. else
53. {
54.         bSector = SECTOR_1;
55.     }
56. }
57.
58. switch(bSector)
59. {
60. case SECTOR_1:
61. case SECTOR_4:
62.         hTimePhA = (T/8) + (((T + wX) - wZ)/2)/131072;
63.         hTimePhB = hTimePhA + wZ/131072;
64.         hTimePhC = hTimePhB - wX/131072;
```

```

65. break;
66. case SECTOR_2:
67. case SECTOR_5:
68.     hTimePhA = (T/8) + (((T + wY) - wZ)/2)/131072;
69.     hTimePhB = hTimePhA + wZ/131072;
70.     hTimePhC = hTimePhA - wY/131072;
71. break;
72.
73. case SECTOR_3:
74. case SECTOR_6:
75.     hTimePhA = (T/8) + (((T - wX) + wY)/2)/131072;
76.     hTimePhC = hTimePhA - wY/131072;
77.     hTimePhB = hTimePhC + wX/131072;
78. break;
79. default:
80. break;
81. }
82.
83. TIM1->CCR1 = hTimePhA;
84. TIM1->CCR2 = hTimePhB;
85. TIM1->CCR3 = hTimePhC;
86. }

```

1、上面的处理的处理函数，先用输入的 V_α 、 V_β 计算 wU_{Alpha} 、 wU_{Beta} ，再根据 wU_{Alpha} 、 wU_{Beta} 计算 wX 、 wY 、 wZ ，再由 wX 、 wY 、 wZ 判断扇区号和占空比，计算过程完全同手册说的一样：

计算 wU_{Alpha} 、 wU_{Beta} ：

$$U_\alpha = \sqrt{3} \times T \times V_\alpha, \quad U_\beta = -T \times V_\beta$$

计算 wX 、 wY 、 wZ ：

$$X = U_\beta, \quad Y = \frac{U_\alpha + U_\beta}{2} \text{ and } Z = \frac{U_\beta - U_\alpha}{2}$$

判断扇区号：

	Y < 0			Y ≥ 0		
	Z < 0	Z ≥ 0		Z < 0	Z ≥ 0	
		X ≤ 0	X > 0	X ≤ 0	X > 0	
Sector	V	IV	III	VI	I	II

计算占空比：

$$\text{Sector I, IV: } t_A = \frac{T+X-Z}{2}, \quad t_B = t_A + Z, \quad t_C = t_B - X$$

$$\text{Sector II, V: } t_A = \frac{T+Y-Z}{2}, \quad t_B = t_A + Z, \quad t_C = t_A - Y$$

$$\text{Sector III, VI: } t_A = \frac{T-X+Y}{2}, \quad t_B = t_C + X, \quad t_C = t_A - Y$$

2、扇区判断和占空比计算的原理与这篇文章说的一样，这里就不细说了。代码中有疑虑就是那个 $T/8$ 和 131072 是什么意思。

1. $hTimePhA = (T/8) + (((T + wX) - wZ)/2)/131072$
2. $hTimePhB = hTimePhA + wZ/131072$
3. $hTimePhC = hTimePhB - wX/131072$

因为 T 等于4倍的 ARR 值，所以 $T/8$ 就等于 $ARR/2$ ，相当于把3个通道的比较值平移了半个周期。因为当3个通道的波形相同时，实际上并没有输出（只有3个低侧开关管或高侧开关管打开），真正有输出的是3个通道波形不一样的地方（有高侧管子打开也有低侧管子打开），所以让3个通道同时移动相同 $ARR/2$ 对输出并没有什么影响，同时方便了计算，避免出现通道比较值为负的情况。

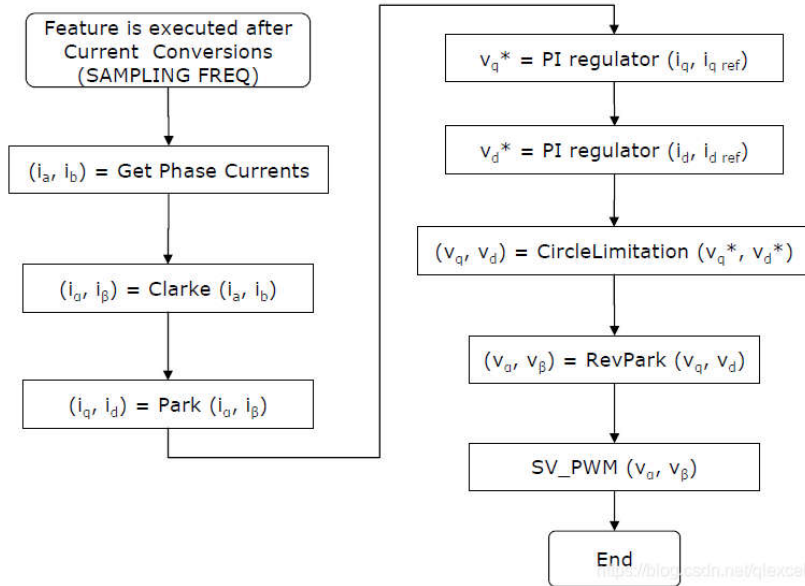
至于 131072 ，网上有人是这么说的：
 $(2^{15}) * 4 = 32768 * 4 = 131072$
 $\#define T (PWM_PERIOD * 4)$ ，这里有一个4倍的放大。
然后电流采用了 $Q15$ 表示（左对齐）， $2^{15} = 32768$ 。
所以最后计算需要除以 131072 。

这篇帖子也有相关讨论：[链接](#)

因为用的 st 的库， $svpwm$ 的实现函数也没怎么去研究过。感觉这个函数有点绕，让人晕，其实知道了原理后，读者可以自己实现。

8、FOC总的处理函数：

FOC_Model (MC_FOC_DRIVE.c)



```

1. Curr_Components Stat_Curr_a_b;
2. Curr_Components Stat_Curr_alpha_beta;
3. Curr_Components Stat_Curr_q_d;
4. Curr_Components Stat_Curr_q_d_ref_ref;
5. Volt_Components Stat_Volt_q_d;
6. Volt_Components Stat_Volt_alpha_beta;
7.
8. void FOC_Model(void)
9. {
10.     Stat_Curr_a_b = GET_PHASE_CURRENTS();
11.     Stat_Curr_alpha_beta = Clarke(Stat_Curr_a_b);
12.     Stat_Curr_q_d = Park( Stat_Curr_alpha_beta, ENC_Get_Electrical_Angle() );
13.
14.
15.     Stat_Volt_q_d.qV_Component1 = PID_Regulator(Stat_Curr_q_d_ref_ref.qI_Component1, Stat_Curr_q_d.qI_Component1, &PID_Torque_InitStructure);
16.
17.     Stat_Volt_q_d.qV_Component2 = PID_Regulator(Stat_Curr_q_d_ref_ref.qI_Component2, Stat_Curr_q_d.qI_Component2, &PID_Flux_InitStructure);
18.
19.     RevPark_Circle_Limitation();
20.     Stat_Volt_alpha_beta = Rev_Park(Stat_Volt_q_d);
21.     CALC_SVPWM(Stat_Volt_alpha_beta);
22. }
  
```

把这个函数放在ADC注入组转换完成中断中调用即可。

现在我们梳理一下整个过程：配置TIM1、ADC，并设置ADC的注入组转换由TIM1触发。这样在每个载波周期，都会触发一次ADC注入组采样，相电流采样完成后调用FOC_Model进行FOC运算处理，然后把最后的计算结果更新到TIM1的输出。现在还有一个问题：现在自己手中有上面的代码、相关的硬件和一个电机，怎么让电机开始转起来呢？那就要看这篇文章了。

9、弱磁控制

待续。。

PS：工程要的人多，我放在这儿了<https://www.cirmall.com/circuit/16544>，卖3块，平台收30%，得2.1，当赏瓶可乐吧。

工程中的代码在本文中已经基本给出来了，为了方便大家直接看，才给出工程的，少去了你自己去整理代码。至于为什么卖3块，我已经说了是“赏”，你觉得看了文章有帮助，“赏”给我的。

那些因为我卖钱，就骂我的白嫖党也是牛逼，你觉得我花时间写这些东西给你们看，是天经地义的？爱看看，不看滚。