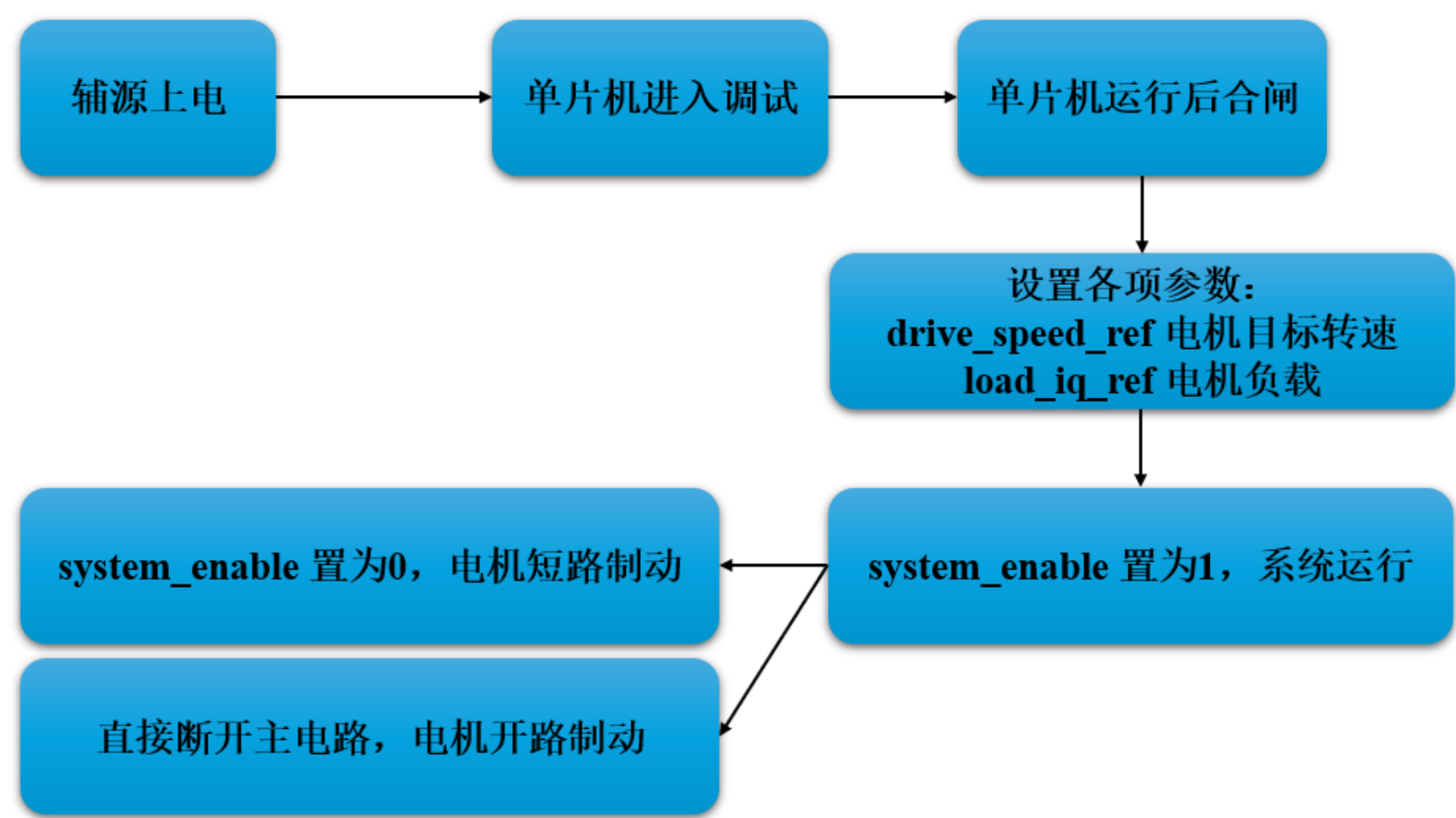


Chapter1 快速上手

1. 运行流程



Chapter2 CubeMX 配置指南

1. 旋转变压器(AD2S1210)配置

GPIO 配置

GPIO 参考配置如下:

- 驱动电机旋转变压器:

引脚	默认输出电平(GPIO output level)	GPIO模式(GPIO mode)	上拉/下拉(GPIO Pull-up/Pull down)	输出速率(Maximum output speed)	旋变引脚
PA11	High	Output Push Pull	Pull-up	High	AD2S1_PCS
PA15(JTDI)	Low	Output Push Pull	Pull-up	Low	AD2S1_A0
PB2	High	Output Push Pull	Pull-up	Low	AD2S1_RD
PB6	Low	Output Push Pull	Pull-up	Low	AD2S1_DIR
PB7	High	Output Push Pull	Pull-up	High	SPI1_CS
PB9	Low	Output Push Pull	Pull-up	Low	AD2S1_A1
PF0	Low	Output Push Pull	Pull-up	High	AD2S1_SAM
PF1	High	Output Push Pull	Pull-up	Low	AD2S1_RESET

- 负载电机旋转变压器：

引脚	默认输出电平(GPIO output level)	GPIO模式(GPIO mode)	上拉/下拉(GPIO Pull-up/Pull down)	输出速率(Maximum output speed)	旋变引脚
PB10	High	Output Push Pull	Pull-up	Low	AD2S2_PCS
PB12	Low	Output Push Pull	Pull-up	Low	AD2S2_A0
PC2_C	Low	Output Push Pull	Pull-up	Low	AD2S2_SAM
PC3_C	High	Output Push Pull	Pull-up	Low	AD2S2_RESET
PC13	High	Output Push Pull	Pull-up	Low	AD2S2_RD
PE0	Low	Output Push Pull	Pull-up	Low	AD2S2_DIR
PE1	High	Output Push Pull	Pull-up	Low	SPI2_CS
PE4	Low	Output Push Pull	Pull-up	Low	AD2S2_A1

SPI 配置

SPI 参考配置如下，一般配置 SPI 波特率在 25 MBits/s 以下：

SPI1 Mode and Configuration

Mode

ModeFull-Duplex Master

Hardware NSS SignalDisable

Configuration

Reset Configuration

Parameter Settings

User Constants

NVIC Settings

DMA Settings

GPIO Settings

Configure the below parameters :

Search (Ctrl+F)

Basic Parameters

Frame FormatMotorola

Data Size8 Bits

First BitMSB First

Clock Parameters

Prescaler (for Baud Rate)8

Baud Rate22.916666 MBits/s

Clock Polarity (CPOL)Low

Clock Phase (CPHA)2 Edge

Advanced Parameters

CRC CalculationDisabled

NSSP ModeEnabled

NSS Signal TypeSoftware

Fifo ThresholdFifo Threshold 01 Data

Tx Crc Initialization PatternAll Zero Pattern

Rx Crc Initialization PatternAll Zero Pattern

Nss PolarityNss Polarity Low

Master Ss Idleness00 Cycle

Master Inter Data Idleness00 Cycle

Master Receiver Auto SuspDisable

Master Keep Io StateMaster Keep Io State Disable

IO SwapDisabled

8位数据
高位先行
预分频器(选择波特率)
CPOL 选择 Low
CPHA 选择 2 Edge
其余选项保持默认

SPI 的引脚:

引脚	SPI引脚
PB3(JTDO/TRACESWO)	SPI1_SCK
PB4(NJTRST)	SPI1_MISO
PB5	SPI1_MOSI
PB13	SPI2_SCK
PB14	SPI2_MISO
PB15	SPI2_MOSI

2. 逆变器配置

TIM 配置

引脚分配

TIM1 负责负载电机的逆变器控制，TIM8 负责驱动电机的逆变器控制。TIM 的引脚分配如下：

引脚	TIM引脚
PA7	TIM8_CH1N
PB0	TIM8_CH2N
PB1	TIM8_CH3N
PC6	TIM8_CH1
PC7	TIM8_CH2
PC8	TIM8_CH3
PE8	TIM1_CH1N
PE9	TIM1_CH1
PE10	TIM1_CH2N
PE11	TIM1_CH2
PE12	TIM1_CH3N
PE13	TIM1_CH3

CHx 对应逆变器上桥臂，CHxN 对应逆变器下桥臂。

对应的 TIM 配置如下：

TIM1 Mode and Configuration

Mode

Slave Mode

Disable

▼

Trigger Source

Disable

▼

Clock Source

Internal Clock

系统时钟作为时钟源

▼

Channel1

PWM Generation CH1 CH1N

1通道互补输出(U相)

▼

Channel2

PWM Generation CH2 CH2N

2通道互补输出(V相)

▼

Channel3

PWM Generation CH3 CH3N

3通道互补输出(W相)

▼

Channel4

Disable

▼

Channel5

Disable

▼

Channel6

Disable

▼

Combined Channels

Disable

▼

Activate-Break-Input

Disable

▼

Activate-Break-Input-2

Disable

▼

Use ETR as Clearing Source

Disable

▼

☐ XOR activation

☐ One Pulse Mode

TIM8 Mode and Configuration

Mode

Slave Mode

Disable

▼

Trigger Source

Disable

▼

Clock Source

Internal Clock

系统时钟作为时钟源

▼

Channel1

PWM Generation CH1 CH1N

1通道互补输出(U相)

▼

Channel2

PWM Generation CH2 CH2N

2通道互补输出(V相)

▼

Channel3

PWM Generation CH3 CH3N

3通道互补输出(W相)

▼

Channel4

PWM Generation No Output

4通道不输出，用于触发ADC采样

▼

Channel5

Disable

▼

Channel6

Disable

▼

Combined Channels

Disable

▼

Activate-Break-Input

Disable

▼

Activate-Break-Input-2

Disable

▼

Use ETR as Clearing Source

Disable

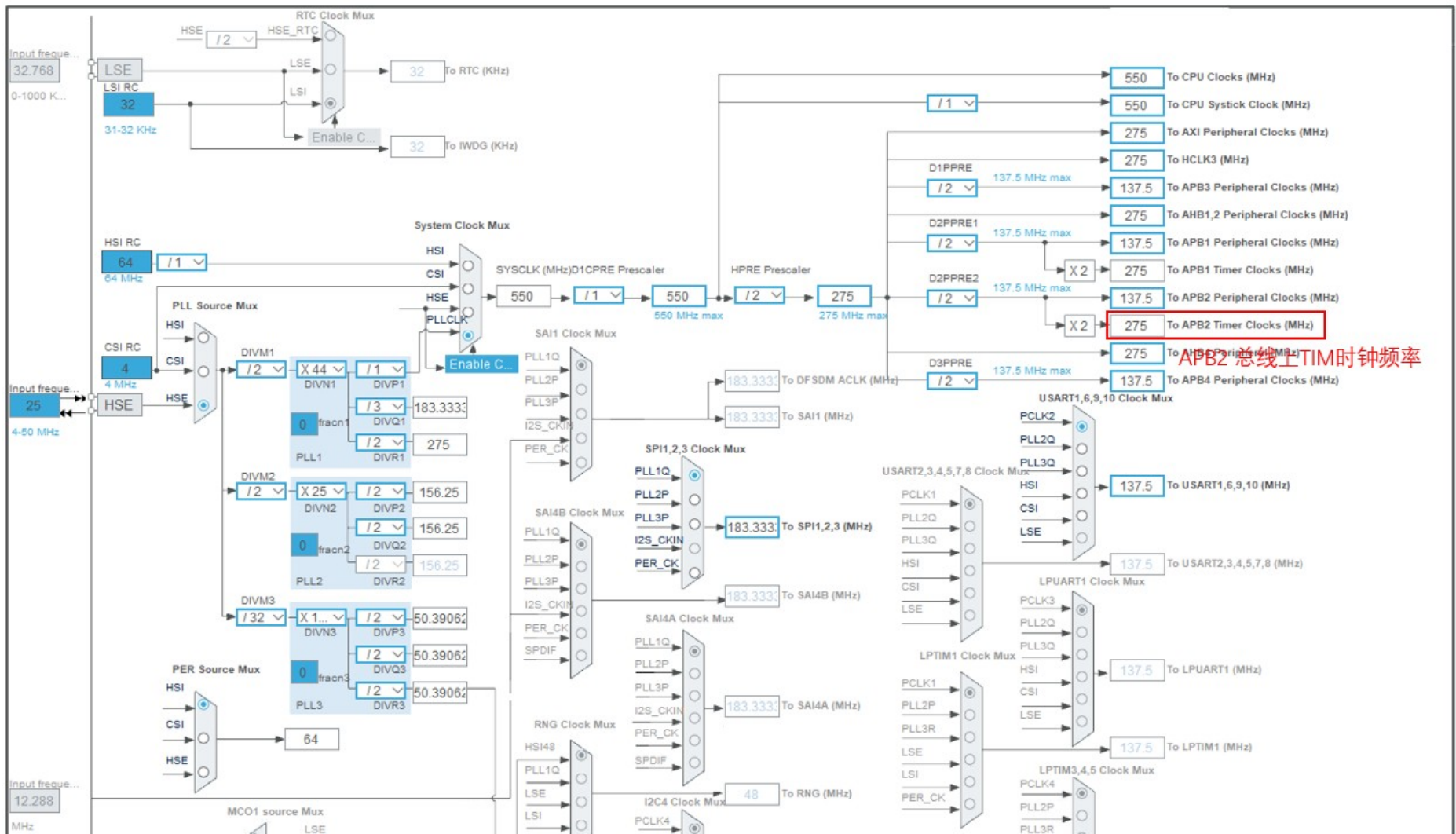
▼

☐ XOR activation

☐ One Pulse Mode

时基单元配置

在时钟树中可以看到 TIM1 和 TIM8 的输入频率均为 275MHz(TIM1 和 TIM8 在 APB2 总线上):



APB2 总线上TIM时钟频率

时基单元配置如下:

TIM8 Mode and Configuration

Configuration

Reset Configuration

Parameter Settings

User Constants

NVIC Settings

DMA Settings

GPIO Settings

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

Prescaler (PSC - 16 bits value)

Counter Mode

Counter Period (AutoReload Register - 16 bits value)

Internal Clock Division (CKD)

Repetition Counter (RCR - 16 bits value)

auto-reload preload

Trigger Output (TRGO) Parameters

Break And Dead Time management - BRK Configuration

Break And Dead Time management - BRK2 Configuration

Break And Dead Time management - Output Configuration

Clear Input

PWM Generation Channel 1 and 1N

PWM Generation Channel 2 and 2N

PWM Generation Channel 3 and 3N

PWM Generation Channel 4

11-1

Center Aligned mode1

1250

No Division

1

Disable

TIM8 预分频值(PSC)

TIM8 计数模式

TIM8 重装载值(ARR)

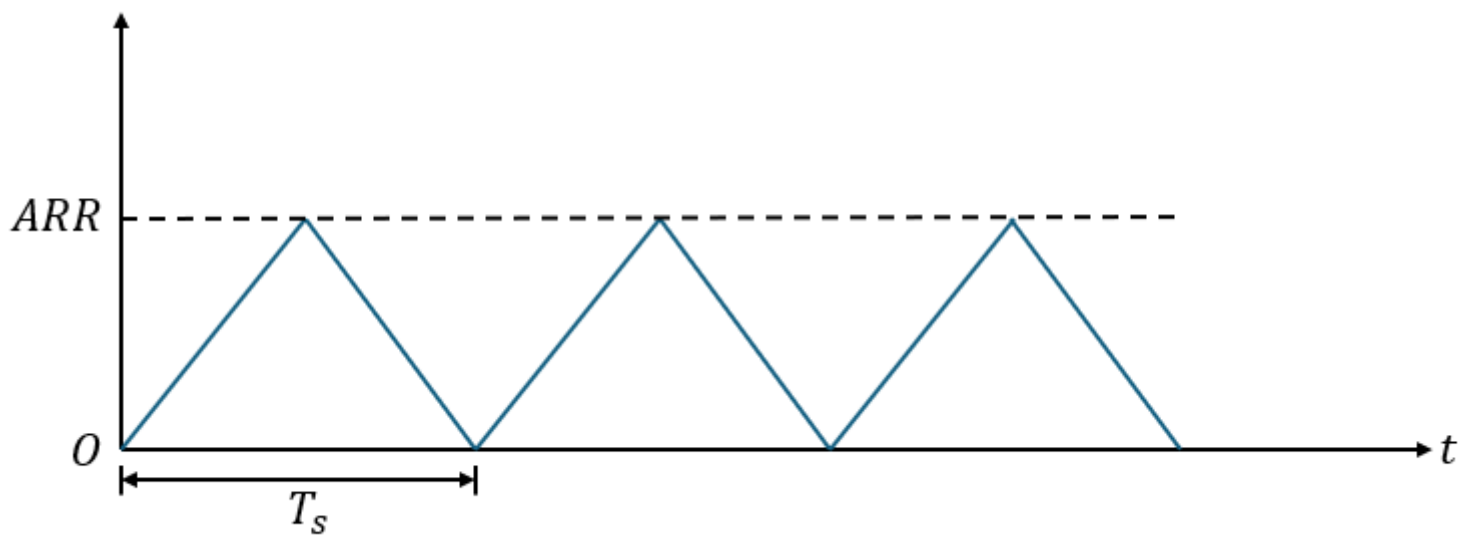
TIM8 重复计数器值(RCR)

不使能ARR影子寄存器

1. 定时器单次计数频率:

f_{TIM} = \frac{f_{APB}}{PSC + 1} = \frac{275MHz}{11} = 25MHz

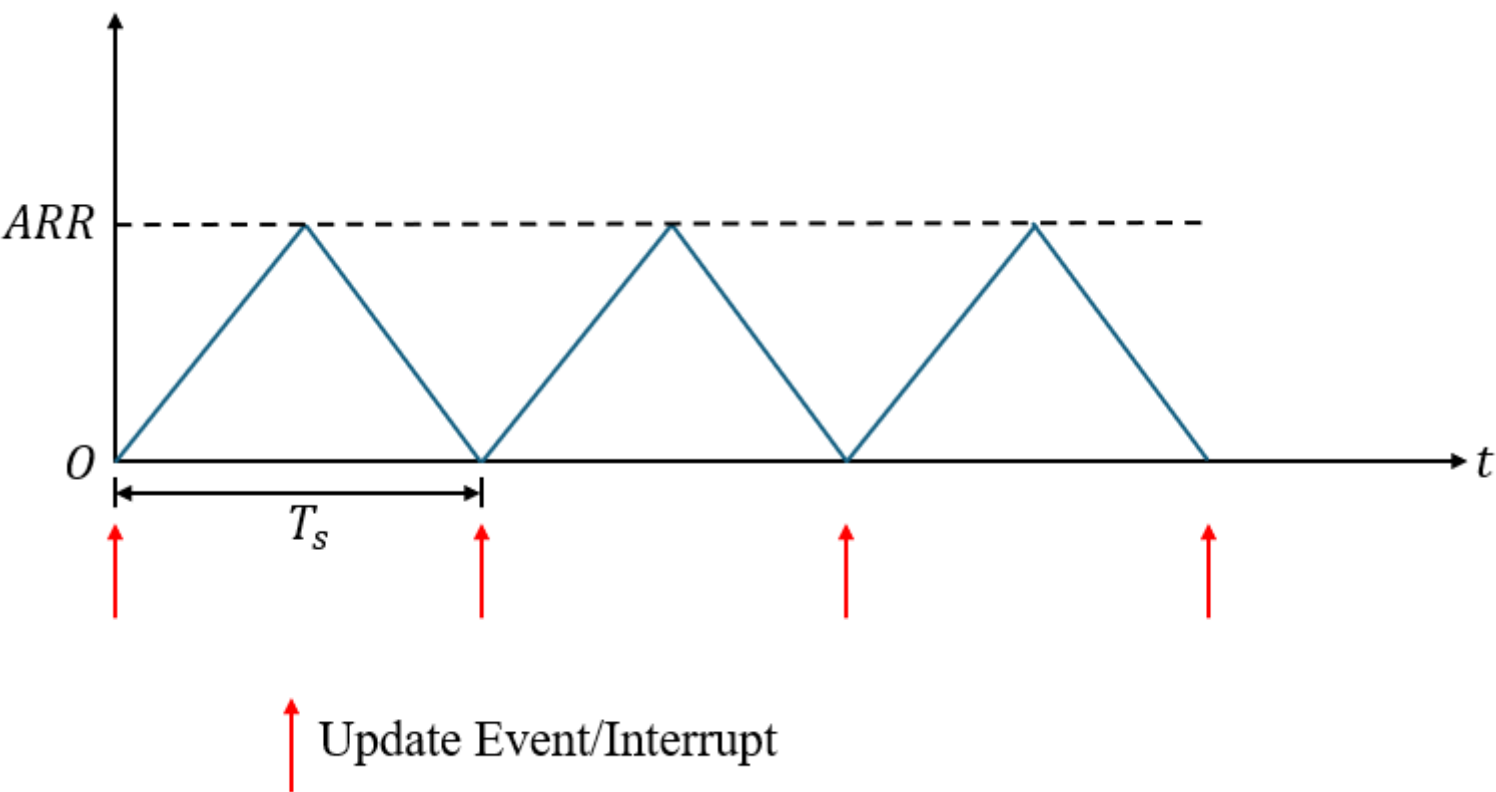
2. 由于使用中心对齐模式, PWM 载波波形如下:



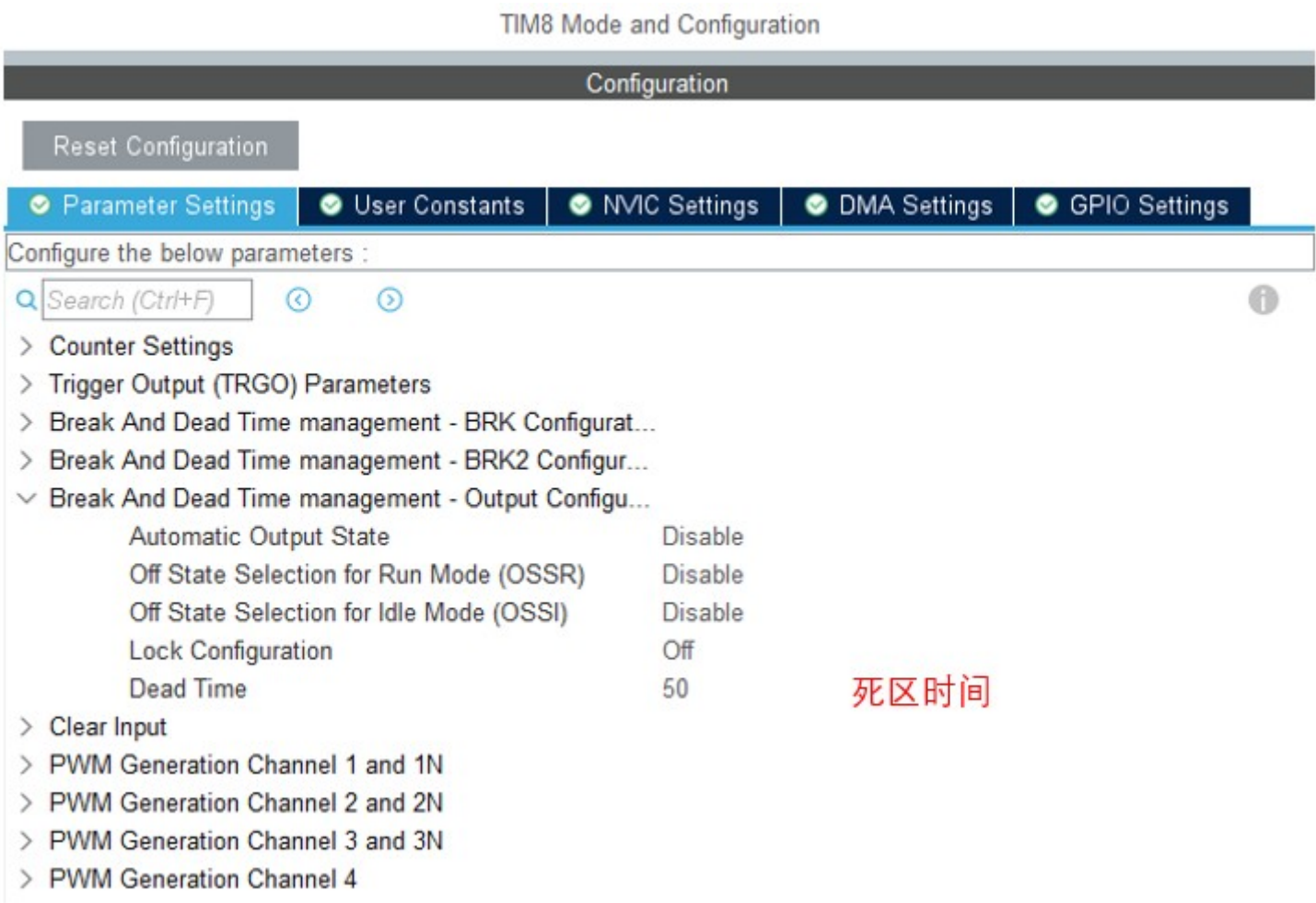
载波频率(逆变器开关频率):

$$f_s = \frac{f_{TIM}}{2 \times ARR} = \frac{25MHz}{2 \times 1250} = 10kHz$$

3. 配置重复计数器 RCR 为 1, 此时 TIM 定时器更新事件/更新中断在下图所示的地方产生:



死区配置



死区时间:

$$T_s = Dead_Time \times \frac{1}{f_{TIM}} = 50 \times \frac{1}{275MHz} = 181ns$$

PWM 输出配置

Configuration	
Reset Configuration	
✓ Parameter Settings	✓ User Constants
✓ NVIC Settings	✓ DMA Settings
✓ GPIO Settings	

Configure the below parameters :

Search (Ctrl+F) ⏪ ⏩ i

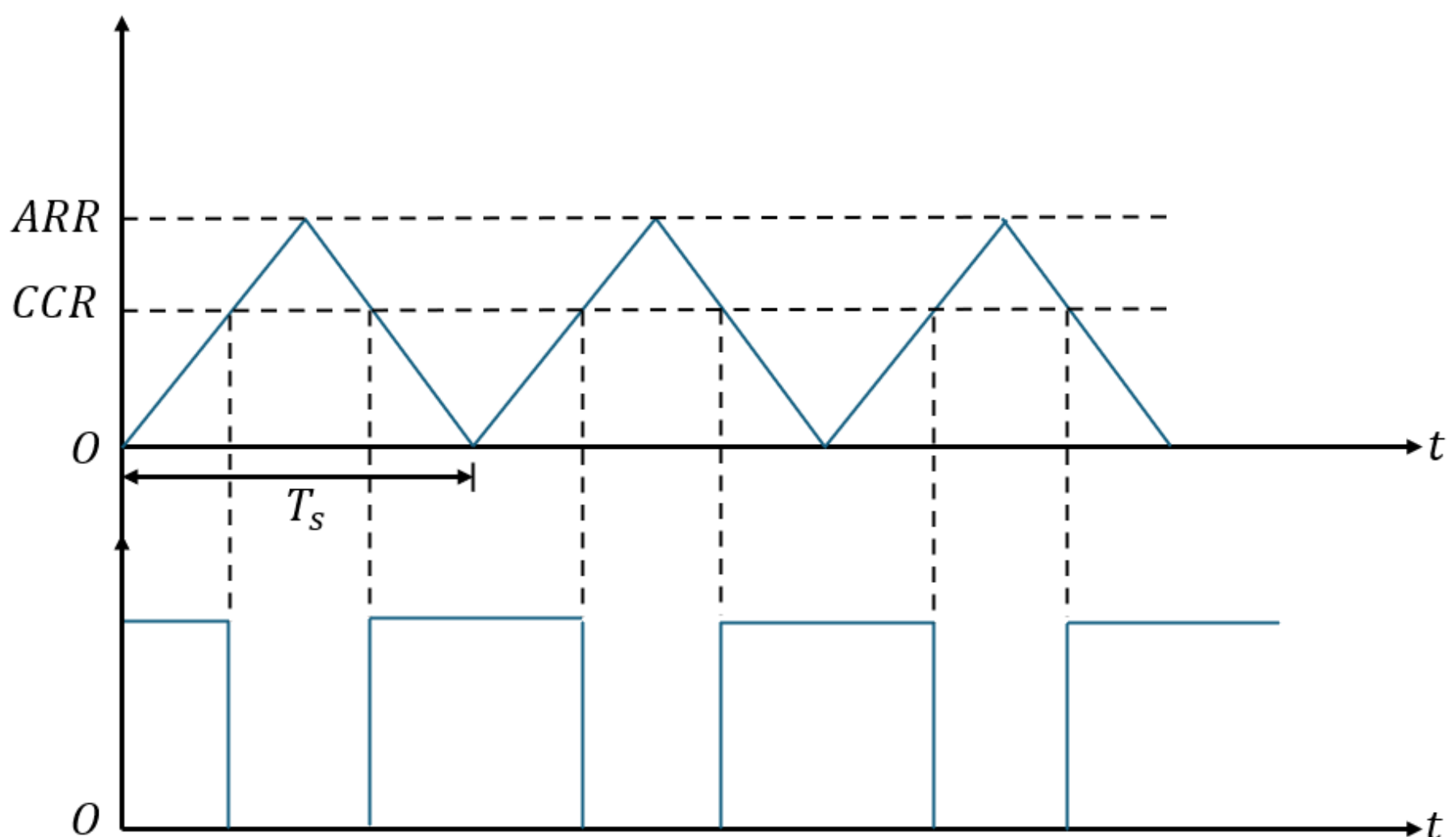
- > Counter Settings
- > Trigger Output (TRGO) Parameters
- > Break And Dead Time management - BRK Configura...
- > Break And Dead Time management - BRK2 Configur...
- > Break And Dead Time management - Output Config...
- > Clear Input
- ✓ PWM Generation Channel 1 and 1N

Mode	PWM mode 1
Pulse (16 bits value)	0
Output compare preload	Enable
Fast Mode	Enable
CH Polarity	High
CHN Polarity	High
CH Idle State	Reset
CHN Idle State	Reset
- ✓ PWM Generation Channel 2 and 2N

Mode	PWM mode 1
Pulse (16 bits value)	0
Output compare preload	Enable
Fast Mode	Enable
CH Polarity	High
CHN Polarity	High
CH Idle State	Reset
CHN Idle State	Reset
- ✓ PWM Generation Channel 3 and 3N

Mode	PWM mode 1
Pulse (16 bits value)	0
Output compare preload	Enable
Fast Mode	Enable
CH Polarity	High
CHN Polarity	High
CH Idle State	Reset
CHN Idle State	Reset

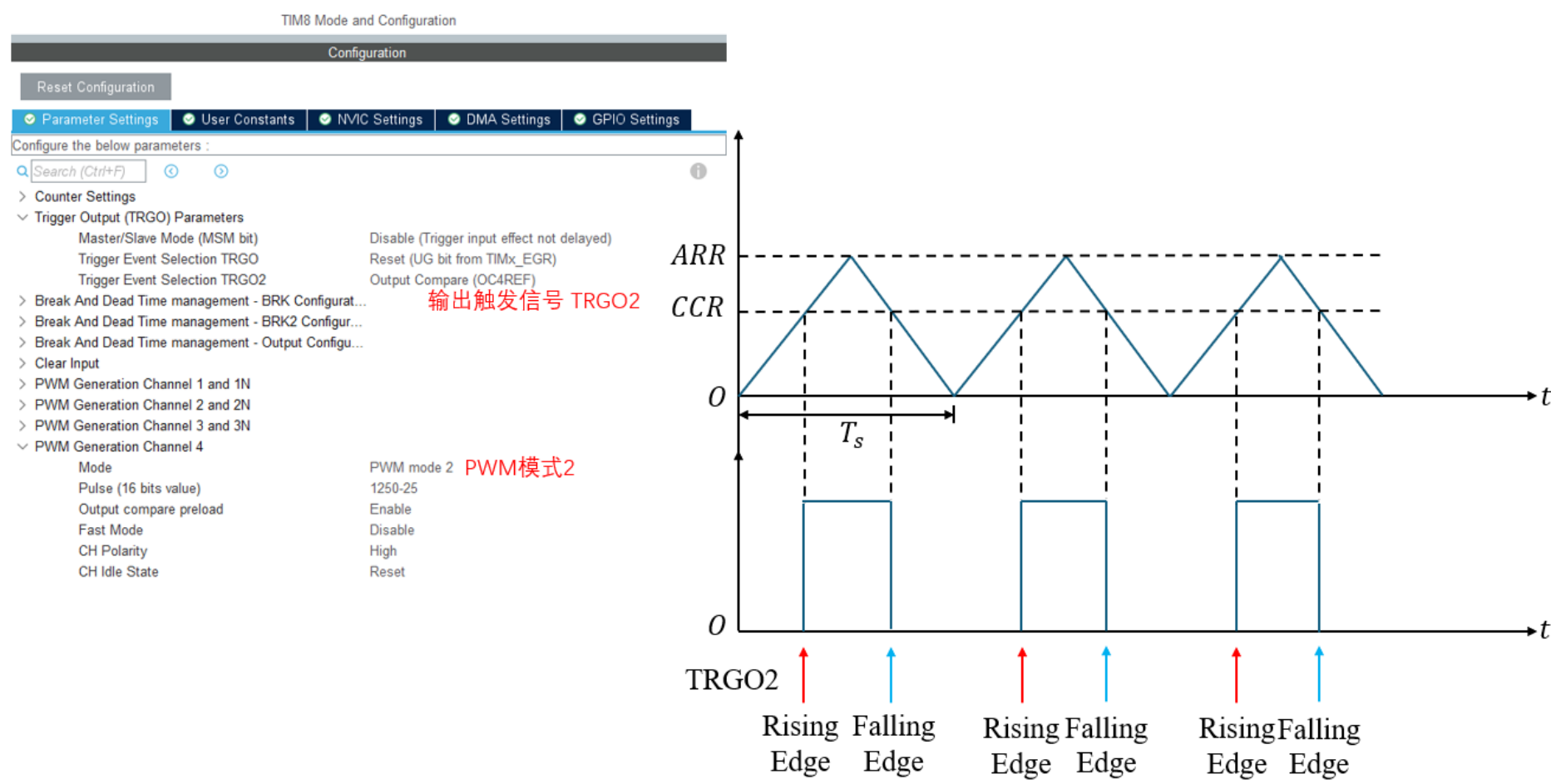
此时的 PWM 波形(逆变器上桥臂开关信号)如下:



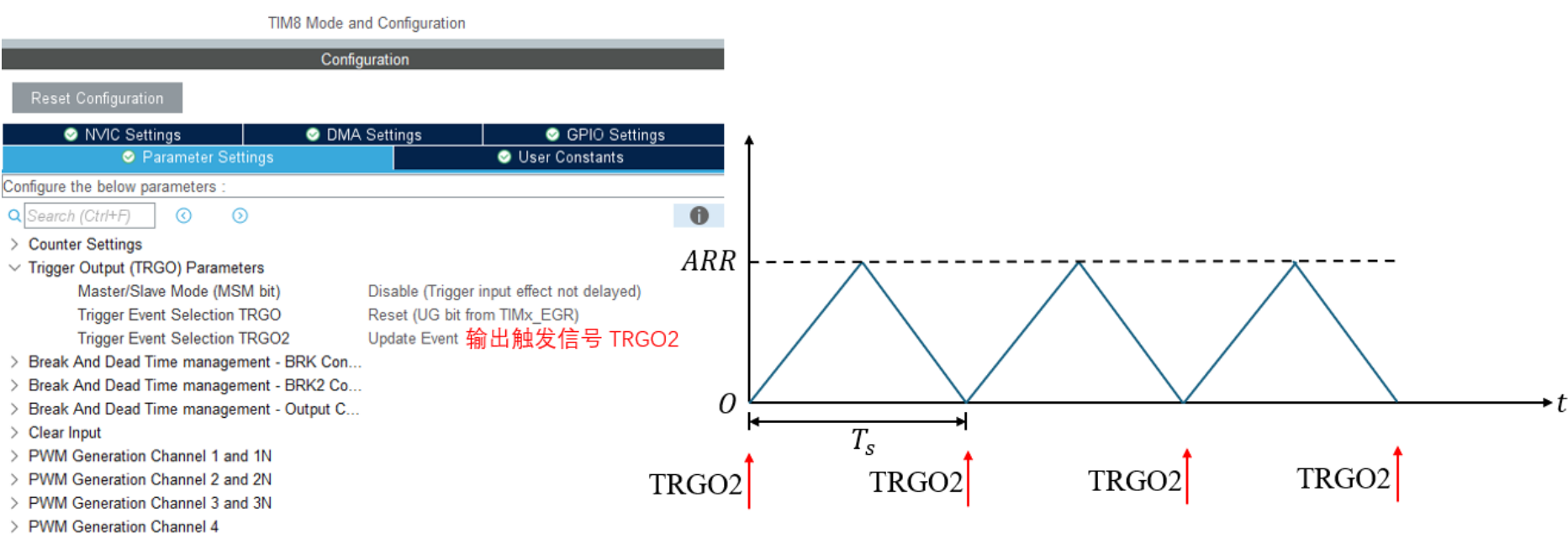
ADC 触发信号配置

配置模式1

使用 PWM 通道 4 形成 TRGO2 触发信号。



配置模式2

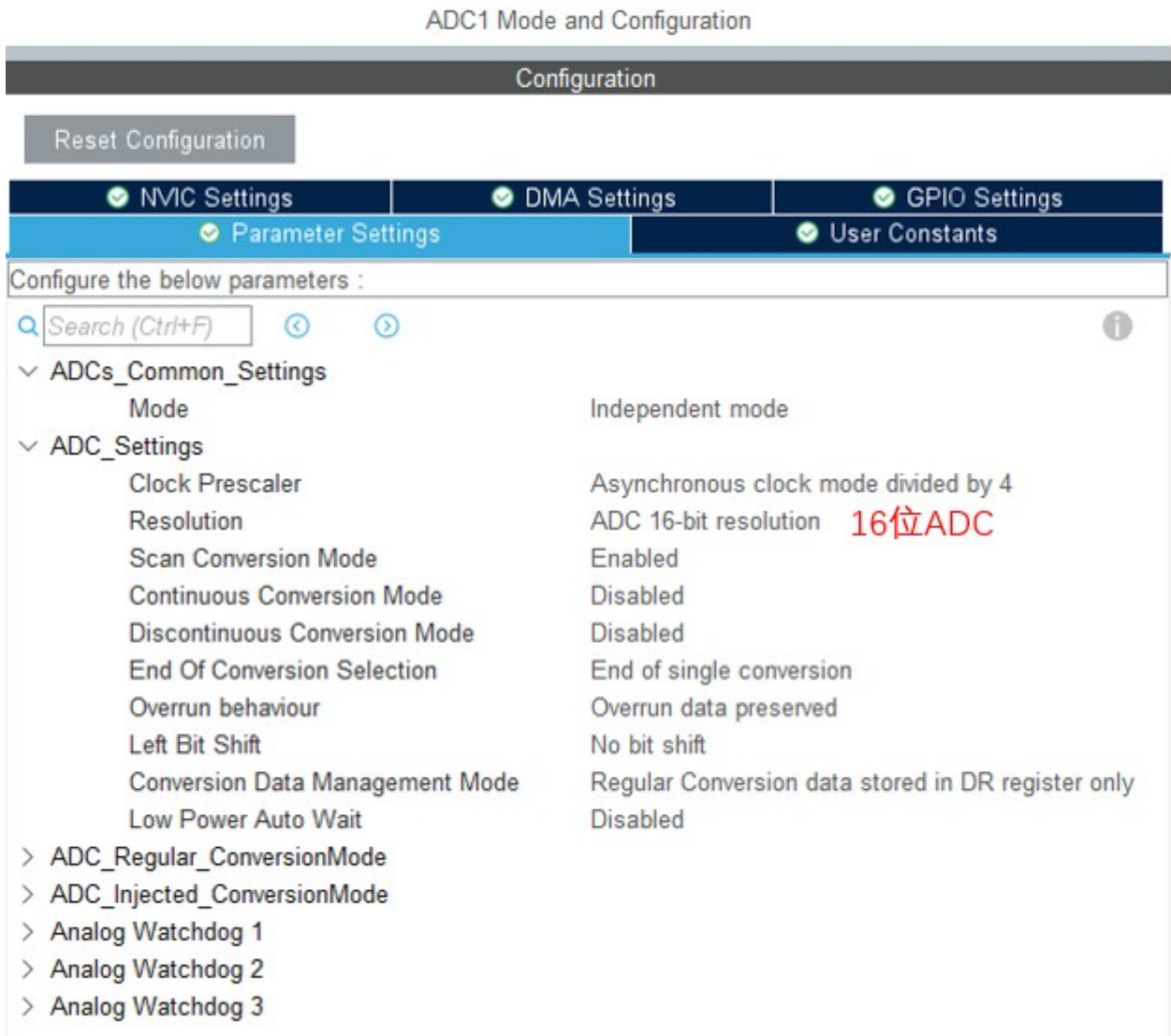


3. 采样配置

引脚分配

引脚	ADC引脚	用途
PA2	ADC1_INP14	负载电机u相
PA3	ADC1_INP15	负载电机v相
PC4	ADC1_INP4	驱动电机v相
PC5	ADC1_INP8	驱动电机u相

ADC基础配置



注入组配置

ADC 的注入组可以使用采用事件触发机制，且优先级最高，适合进行电流采样。



ADC1 Mode and Configuration

Configuration

Reset Configuration

✓ NVIC Settings

✓ DMA Settings

✓ GPIO Settings

✓ Parameter Settings

✓ User Constants

Configure the below parameters :

Q Search (Ctrl+F)

⌂

⌂

i

▼ ADC_Injected_ConversionMode

Enable Injected Conversions

Enable Injected Oversampling

Number Of Conversions

External Trigger Source

External Trigger Conversion Edge

Injected Conversion Mode

Injected Queue

Rank

Channel

Sampling Time

Offset Number

Offset Signed Saturation

Injected Offset Saturation

Rank

Channel

Sampling Time

Offset Number

Offset Signed Saturation

Injected Offset Saturation

Rank

Channel

Sampling Time

Offset Number

Offset Signed Saturation

Injected Offset Saturation

Rank

Channel

Sampling Time

Offset Number

Offset Signed Saturation

Injected Offset Saturation

Enable

Disable

4

Timer 8 Capture Compare 4 event

Trigger detection on the rising edge

None

Injected Queue Disable

1

Channel 4

1.5 Cycles

No offset

Disable

Disable

2

Channel 8

1.5 Cycles

No offset

Disable

Disable

3

Channel 14

1.5 Cycles

No offset

Disable

Disable

4

Channel 15

1.5 Cycles

No offset

Disable

Disable

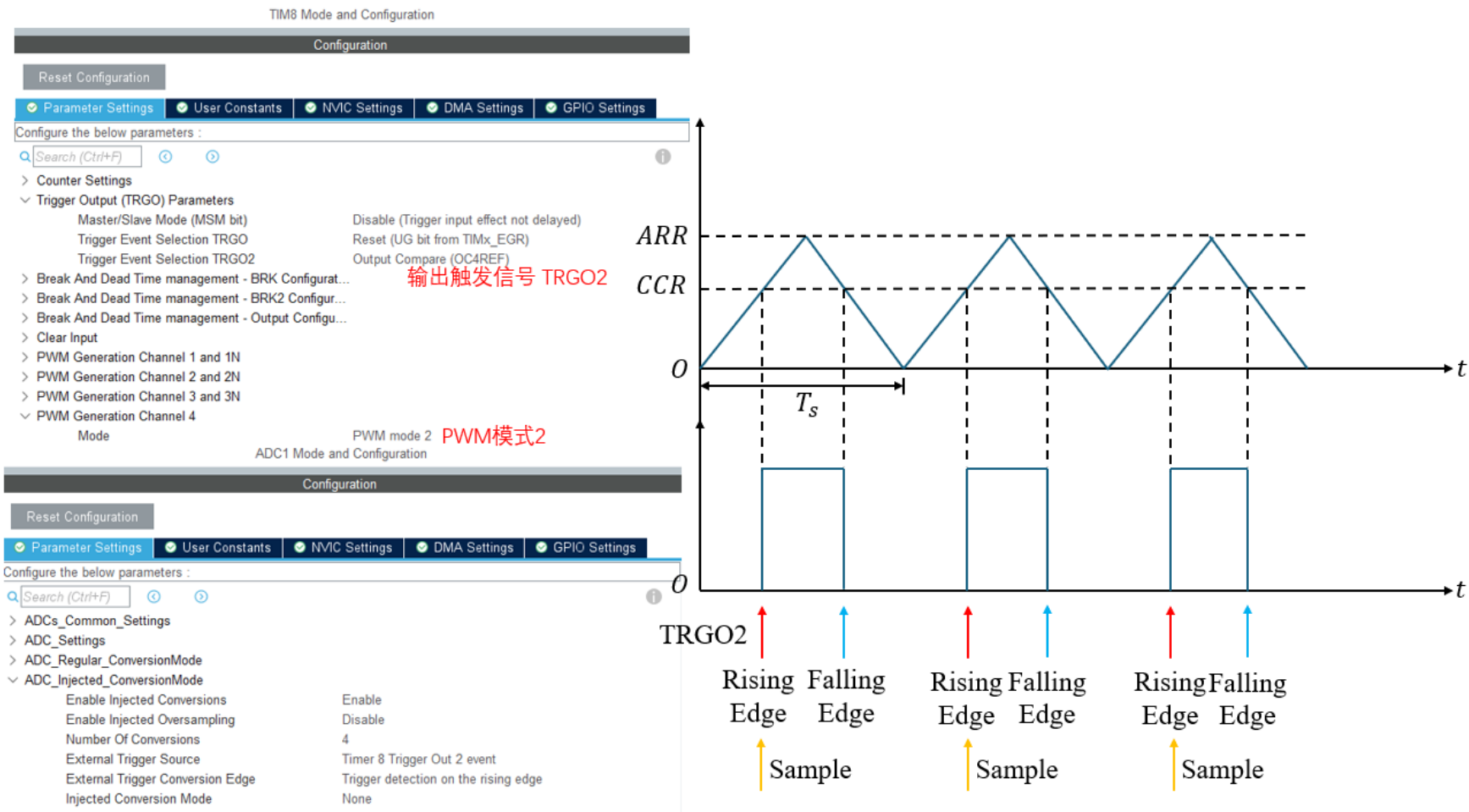
注入组1通道，对应JDR1

ADC_INP4

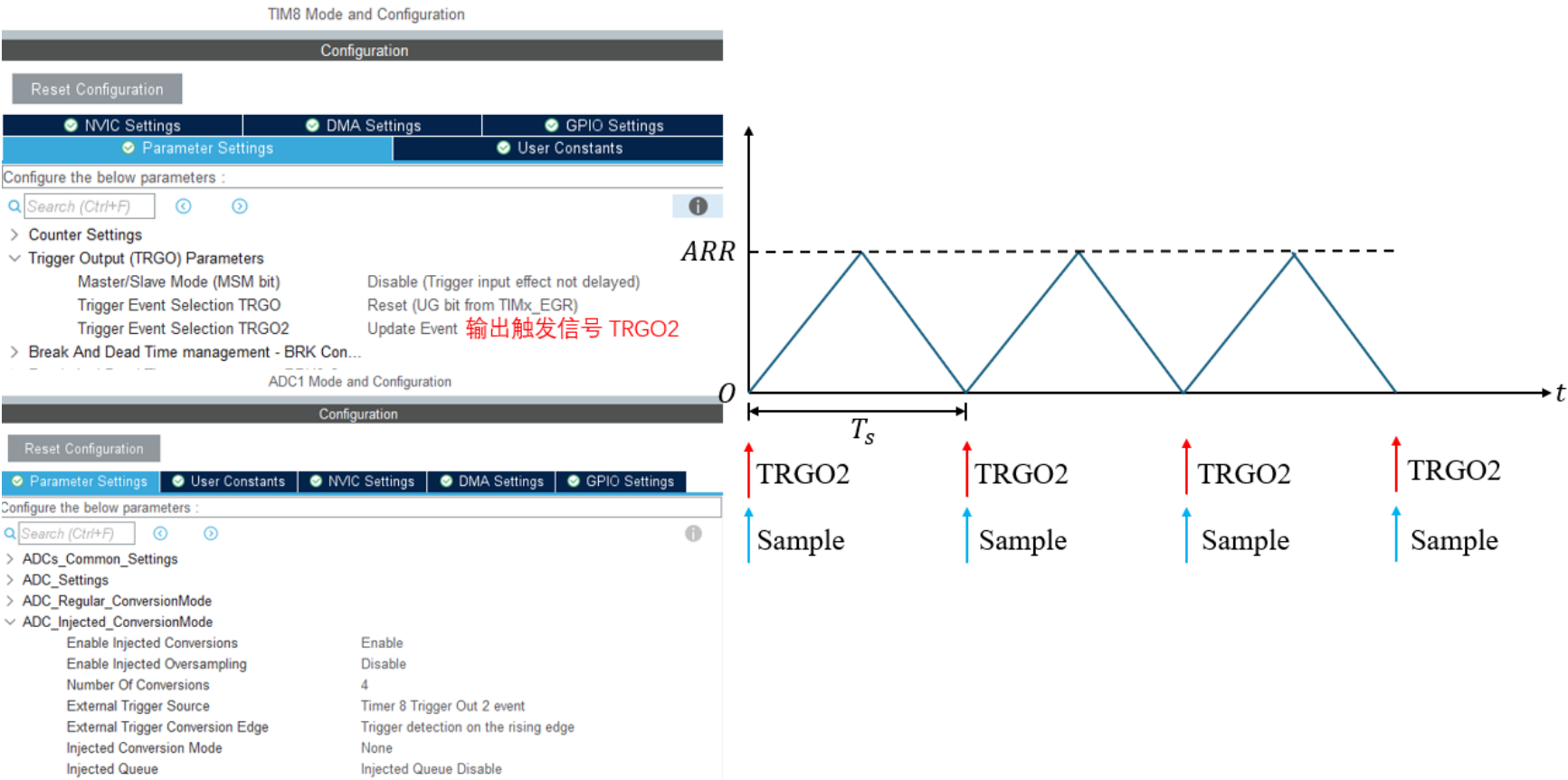
1.5个ADC周期用于采样保持

配置模式1

ADC 的触发事件可以为 Capture Compare 4 event :



配置模式2

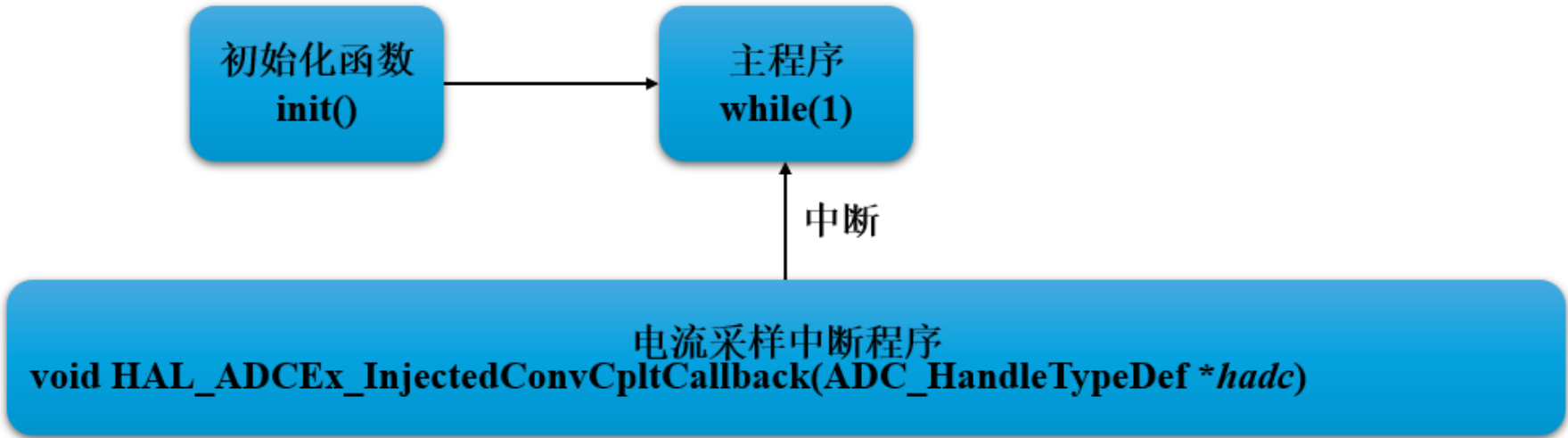


采样完成后会触发注入组采样中断，在 CubeMX 里面使能中断：

ADC1 Mode and Configuration			
Configuration			
Reset Configuration			
Parameter Settings	User Constants	NVIC Settings	DMA Settings
NVIC Interrupt Table			
ADC1 and ADC2 global interrupts	Enabled	Preemption Priority	Sub Priority
	<input checked="" type="checkbox"/>	0	0

Chapter3 用户代码编写指南

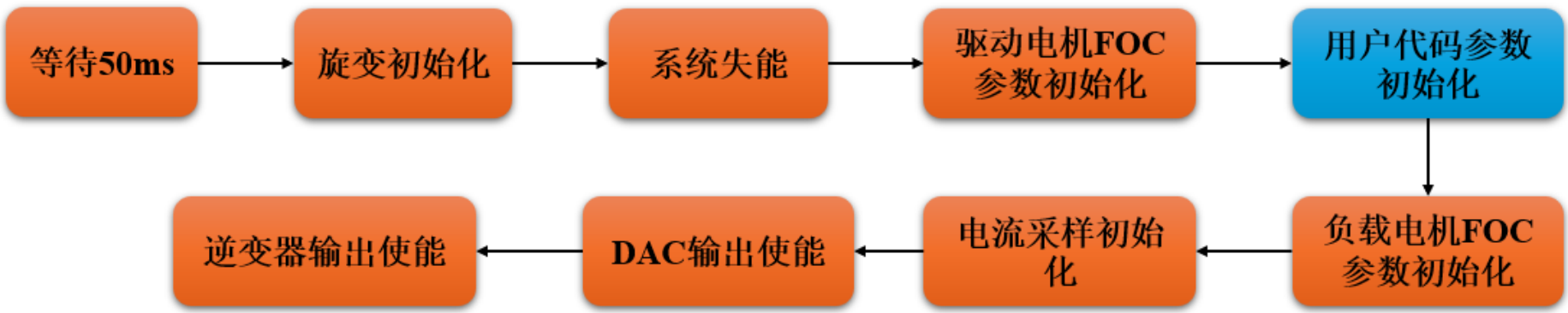
1. 代码结构



初始化程序

```
1  /**
2   * @brief  Init Program
3   */
4  static void init(void)
5  {
6      // Wait
7      HAL_Delay(50);
8      // Init AD2S1210
9      AD2S1210_Init();
10     // system init
```

```
11  system_enable = 0;
12  // Init FOC parameters
13  // Drive Motor
14  drive_speed_ref          = 0;
15  Drive_curr.adc_val_u_offset = 0;
16  Drive_curr.adc_val_v_offset = 0;
17  Drive_curr.sample_flag    = CURR_SAMPLE_GET_OFFSET; // Read ADC Offset
18  PID_init(&Drive_id_pi, 0.005, 12.5, 0, 80);          // Current PI Init
19  PID_init(&Drive_iq_pi, 0.005, 12.5, 0, 80);
20  PID_init(&Drive_speed_pi, 0.1, 0.1, 0, 1); // Speed PI Init
21  // 用户在此处定义另外的初始化代码
22
23  // Load Motor
24  load_iq_ref              = 0;
25  Load_curr.sample_flag   = CURR_SAMPLE_GET_OFFSET; // Read ADC Offset
26  Load_curr.adc_val_u_offset = 0;
27  Load_curr.adc_val_v_offset = 0;
28  PID_init(&Load_id_pi, 0.005, 12.5, 0, 80); // Current PI Init
29  PID_init(&Load_iq_pi, 0.005, 12.5, 0, 80);
30  // Current Sample
31  HAL_ADCEx_Calibration_Start(&hadc1, ADC_CALIB_OFFSET, ADC_SINGLE_ENDED);
32  __HAL_ADC_CLEAR_FLAG(&hadc1, ADC_FLAG_JEOC);
33  __HAL_ADC_CLEAR_FLAG(&hadc1, ADC_FLAG_EOC);
34  HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_4);
35  HAL_ADCEx_InjectedStart_IT(&hadc1);
36  // Wait for Read ADC Offset
37  while (Drive_curr.sample_flag == CURR_SAMPLE_GET_OFFSET || Load_curr.sample_flag ==
CURR_SAMPLE_GET_OFFSET) {
38      ;
39  }
40  // DAC Init
41  HAL_DAC_Start(&hdac1, DAC1_CHANNEL_1);
42  // Open Inverter
43  HAL_TIM_Base_Start_IT(&htim8);
44  HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_1);
45  HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_2);
46  HAL_TIM_PWM_Start(&htim8, TIM_CHANNEL_3);
47  HAL_TIMEx_PWMN_Start(&htim8, TIM_CHANNEL_1);
48  HAL_TIMEx_PWMN_Start(&htim8, TIM_CHANNEL_2);
49  HAL_TIMEx_PWMN_Start(&htim8, TIM_CHANNEL_3);
50  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
51  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
52  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
53  HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_1);
54  HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_2);
55  HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_3);
56 }
```

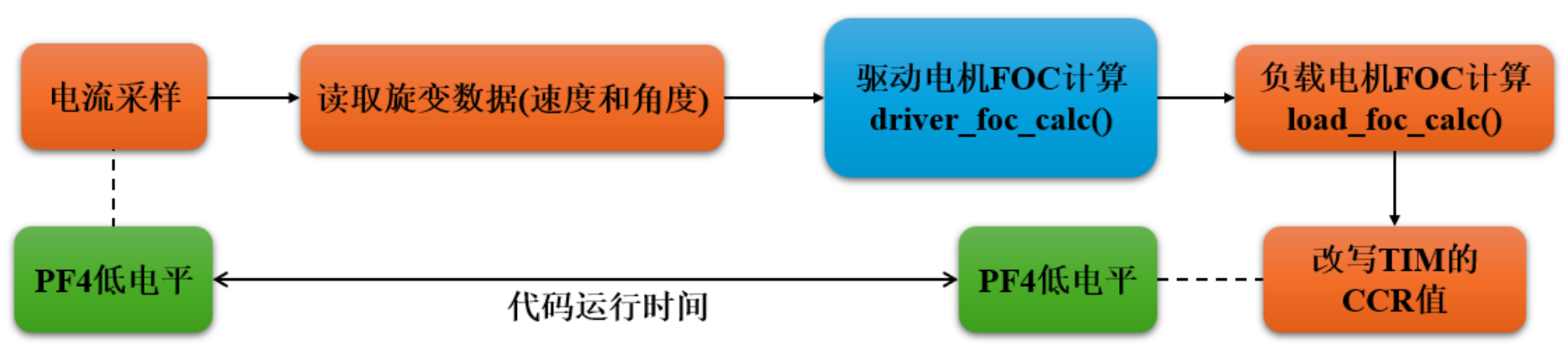


电流采样中断程序

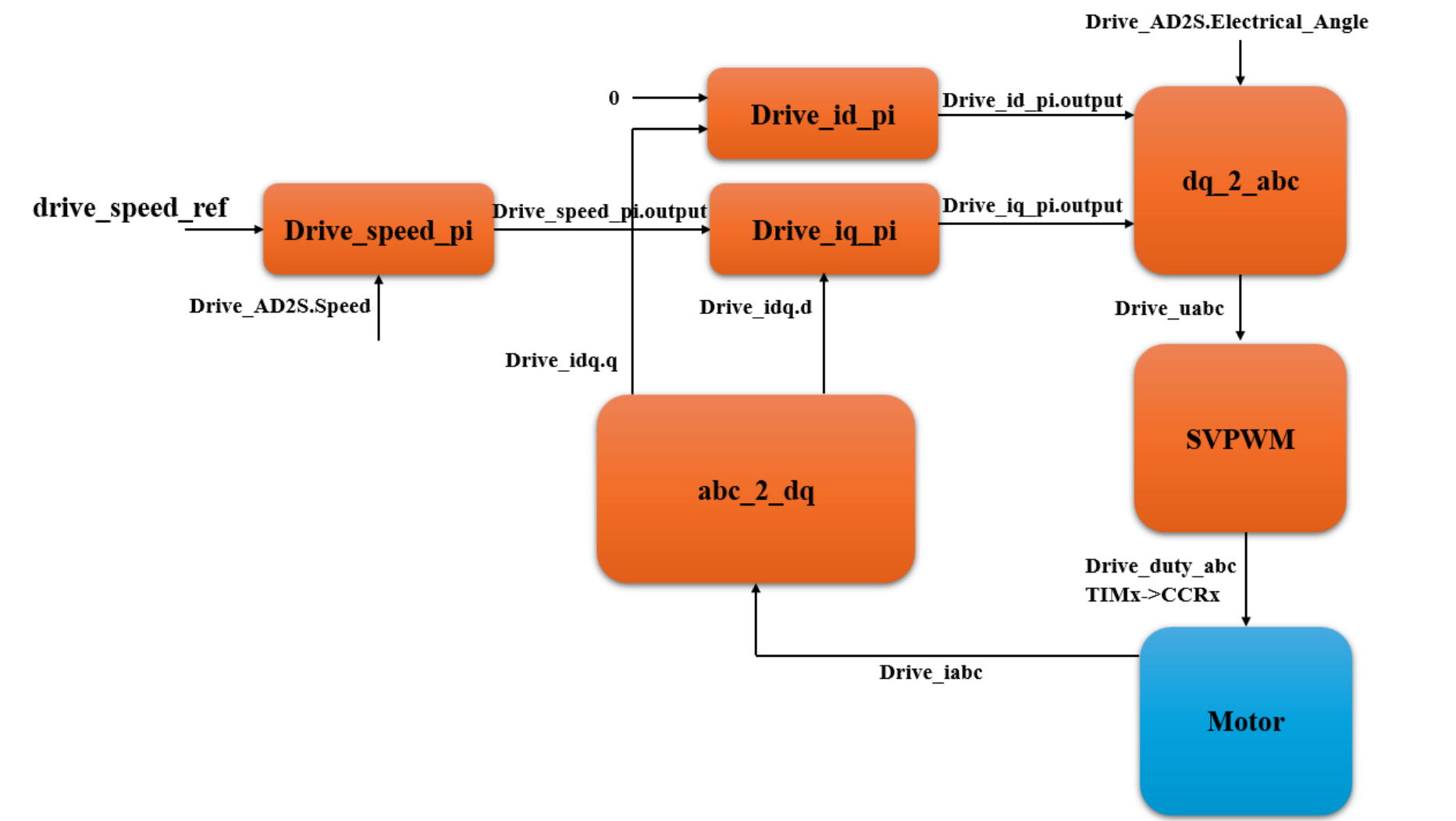
```
1  /**
2   * @brief   ADC Injected Channel interrupt Callback function
3   */
4  void HAL_ADCEx_InjectedConvCpltCallback(ADC_HandleTypeDef *hadc)
5  {
6      static int adc_cnt          = 0;
7      static uint32_t adc_u1_offset_sum = 0;
8      static uint32_t adc_v1_offset_sum = 0;
9      static uint32_t adc_u2_offset_sum = 0;
10     static uint32_t adc_v2_offset_sum = 0;
11     HAL_GPIO_WritePin(GPIOF, GPIO_PIN_4, 0); // Caculate running time
12     UNUSED(hadc);
13     /*****
14      * @brief   Sample
15      */
16     if (hadc == &hadc1) {
17         // Read Offset
18         if (Drive_curr.sample_flag == CURR_SAMPLE_GET_OFFSET) {
19             adc_u1_offset_sum += hadc1.Instance->JDR1;
20             adc_v1_offset_sum += hadc1.Instance->JDR2;
21             adc_u2_offset_sum += hadc1.Instance->JDR3;
22             adc_v2_offset_sum += hadc1.Instance->JDR4;
23             adc_cnt++;
24             if (adc_cnt == 20) {
25                 adc_cnt          = 0;
26                 Drive_curr.adc_val_u_offset = adc_u1_offset_sum / 20.0f;
27                 Drive_curr.adc_val_v_offset = adc_v1_offset_sum / 20.0f;
28                 Drive_curr.sample_flag      = CURR_SAMPLE_RUNNING;
29                 adc_u1_offset_sum           = 0;
30                 adc_v1_offset_sum           = 0;
31                 Load_curr.adc_val_u_offset = adc_u2_offset_sum / 20.0f;
32                 Load_curr.adc_val_v_offset = adc_v2_offset_sum / 20.0f;
33                 Load_curr.sample_flag      = CURR_SAMPLE_RUNNING;
34                 adc_u2_offset_sum           = 0;
35                 adc_v2_offset_sum           = 0;
36             }
37         }
38         // Read Current
39         else {
40             Drive_curr.adc_val_u = hadc1.Instance->JDR1;
41             Drive_curr.adc_val_v = hadc1.Instance->JDR2;
42             Load_curr.adc_val_u  = hadc1.Instance->JDR3;
43             Load_curr.adc_val_v  = hadc1.Instance->JDR4;
44             adc_2_curr(&Drive_curr);
45             adc_2_curr(&Load_curr);
46             Drive_iabc.a = Drive_curr.curr_u;
47             Drive_iabc.b = Drive_curr.curr_v;
48             Drive_iabc.c = Drive_curr.curr_w;
49             Load_iabc.a  = Load_curr.curr_u;
50             Load_iabc.b  = Load_curr.curr_v;
51             Load_iabc.c  = Load_curr.curr_w;
52             // software protection
53             if ((Drive_iabc.a * Drive_iabc.a > MAX_CURRENT * MAX_CURRENT) || (Drive_iabc.b *
Drive_iabc.b > MAX_CURRENT * MAX_CURRENT) || (Drive_iabc.c * Drive_iabc.c > MAX_CURRENT *
MAX_CURRENT) || (Load_iabc.a * Load_iabc.a > MAX_CURRENT * MAX_CURRENT) || (Load_iabc.b * Load_iabc.b
> MAX_CURRENT * MAX_CURRENT) || (Load_iabc.c * Load_iabc.c > MAX_CURRENT * MAX_CURRENT)) {
54                 system_enable = 0;
55             }
56         }
57     }
58     // Angle and Speed Sample
59     AD2S1210_Angle_Get(); // Angle Sample
60     AD2S1210_Speed_Get(system_sample_time); // Speed Sample
```



```
61  /*****
62  * @brief   FOC Calculate
63  */
64  drive_foc_calc();
65  load_foc_calc();
66
67  /*****
68  * @brief   Voltage-Source Inverter Control
69  */
70  if (system_enable == 0) {
71      TIM8->CCR1 = 0;
72      TIM8->CCR2 = 0;
73      TIM8->CCR3 = 0;
74      TIM1->CCR1 = 0;
75      TIM1->CCR2 = 0;
76      TIM1->CCR3 = 0;
77  } else {
78      TIM8->CCR1 = Drive_duty_abc.dutya * TIM8->ARR;
79      TIM8->CCR2 = Drive_duty_abc.dutyb * TIM8->ARR;
80      TIM8->CCR3 = Drive_duty_abc.dutyc * TIM8->ARR;
81      TIM1->CCR1 = Load_duty_abc.dutya * TIM1->ARR;
82      TIM1->CCR2 = Load_duty_abc.dutyb * TIM1->ARR;
83      TIM1->CCR3 = Load_duty_abc.dutyc * TIM1->ARR;
84  }
85
86  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_4, 1); // Caculate running time
87 }
```

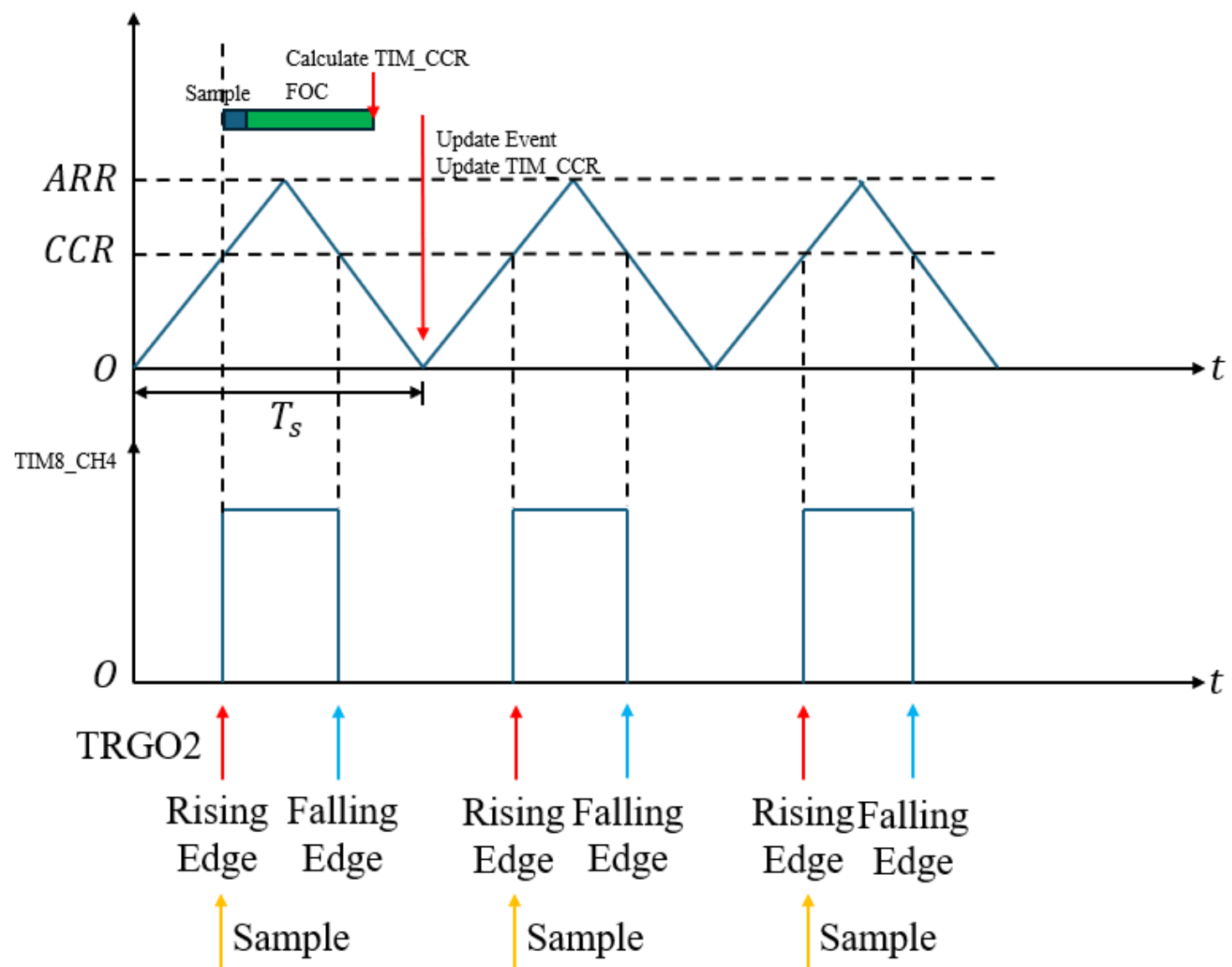


用户可以改写驱动电机的 FOC 代码(增加无感算法)。 `drive_foc_calc()` 函数运行逻辑如下:

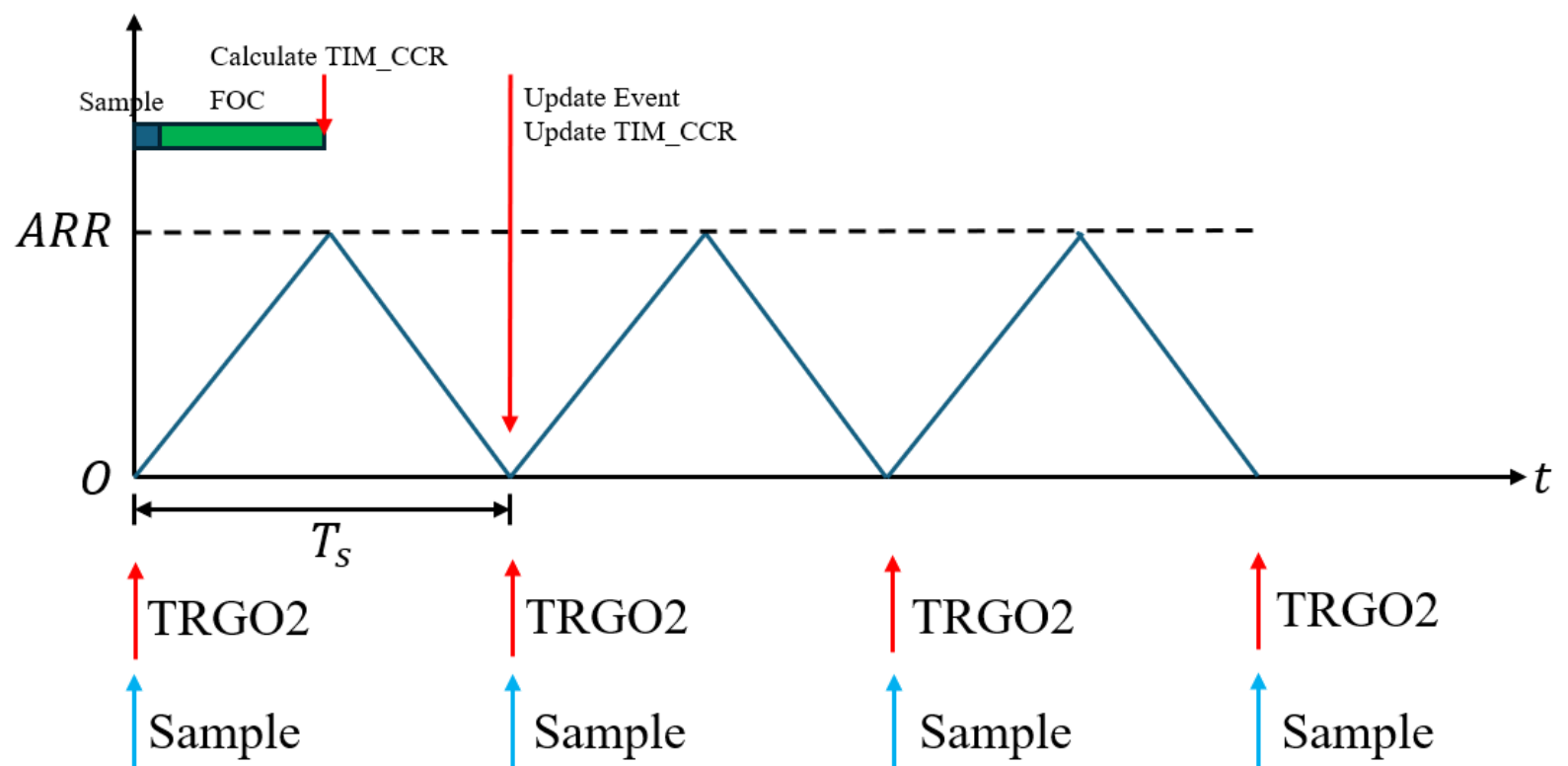


不同配置下的采样-更新时序:

- 配置模式1



- 配置模式2



2. 用户编写代码

用户应当在 `userpara.c / userpara.h` , `drivefoc.c / drivefoc.h` , `userinit.c / userinit.h` 三组文件中编写用户自定义代码。

userpara.c/h 文件

该文件进行全局变量定义，将全局变量单独列出一个文件以便于调试时观察变量。

userinit.c/h 文件

该文件内是用户代码参数初始化函数 `user_init()`。

用户在该函数内编写代码以便进行算法参数初始化。

drivefoc.c/h 文件

该文件内是用于进行驱动电机 FOC 算法的函数 `drive_foc_calc()`。

用户在该函数内编写 **针对驱动电机的无感算法**。