22 تایمر

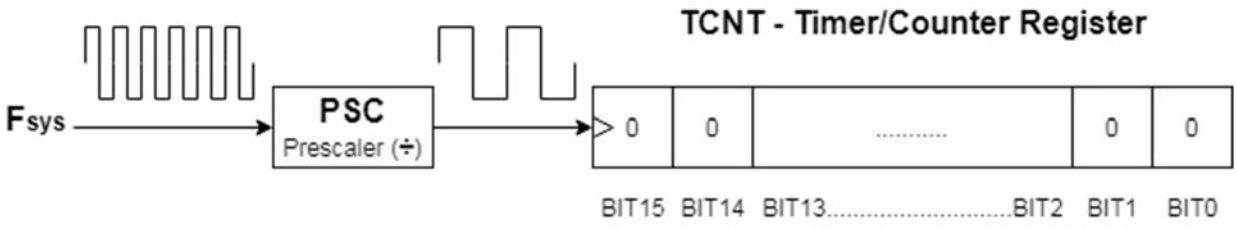
معمولاً تعداد زیادی از کارهای کنترلی، نیازمند اندازهگیری زمان یا شمارش یک اتفاق هستند. برای پیاده سازی چنین عملیاتهایی، در ریزپردازندهها یک یا چند تایمر تعبیه شده است. در این جلسه نحوه کار با تایمرها، حالت های کاری مختلف آنها و وقفه های مرتبط مورد بحث قرار خواهند گرفت.

22.1 مقدمه

تایمرها در حقیقت شمارنده های سختافزاری مجزایی هستند که در صورت فعال بودن به طور موازی با CPU عمل شمارش را انجام داده و مقدار آنها افزایش یا کاهش مییابد. از این رو تایمرها مهمترین ابزار برای سنجش زمان در ریزپردازنده ها میباشند.

یک ماژول تایمر در اساسی ترین شکل آن یک مدار منطقی دیجیتال است که در هر چرخه ساعت یک شماره انجام میدهد . قابلیت های بیشتری در سخت افزار برای پشتیبانی از ماژول تایمر وجود دارد تا بتواند بصورت بالا یا پایین بشمارد. تایمر می تواند یک پیش مقیاس (Prescaler)برای تقسیم فرکانس ساعت ورودی با یک مقدار قابل انتخاب داشته باشد. همچنین می تواند مدارهایی ضبط ورودی(Input Capture) ، تولید سیگنال PWM و موارد دیگر را داشته باشد همانطور که در ادامه خواهیم دید.

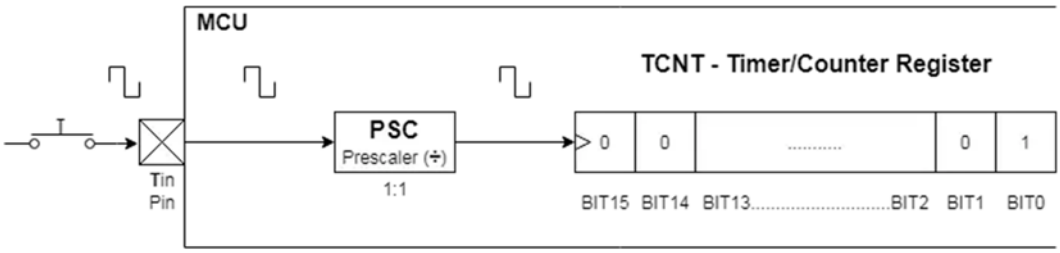
اگر یک تایمر پایه 16 بیتی مانند شکل زیر را در نظر بگیریم که به عنوان یک زمان سنج 16بیتی می تواند از 0 تا 65535 بشمارد میبینیم که در هر چرخه ساعت ، مقدار تایمر به اندازه یکی افزایش می یابد و همانطور که می بینید ، Fsys فرکانسی نیست که ماژول تایمر را افزایش می دهد، بلکه توسط پیش مقیاس تقسیم می شود و سپس به تایمر اعمال می شود.



اساسا ، در حالت تایمر ، رجیستر TCNT با هر چرخه ساعت 1 عدد با فرکانس Fsys/PSC افزایش می یابد. این به این معنی است که اگر Fsys برابر با 80MHz باشد و PSC برابر یا 1024 باشد، TCNT در هر 12.8 میکروثانیه یک عدد افزایش میابد. بنابراین اگر شما این تایمر را راه اندازی کنید تا از مقدار 0 تا زمانی که سرریز شود(65535) بشمارد، به شما یک سیگنال وقفه در هر 0.839 ثانیه خواهد داد.

خب اگر لازم باشد این تایمر را تنظیم کنیم تا یک بار در هر 1 ثانیه سیگنال وقفه به ما بدهد چه ؟ در واقع ما این فاصله زمانی 0.839 ثانیه ای را نمی خواهیم. خوب ، به همین دلیل ، یک ویژگی سخت افزاری به نام preload register وجود دارد که تایمر را مجبور می کند تا از هر مقدار دلخواه انتخاب شده تا سرریز بشمارد. بنابراین ، دیگر لازم نیست از صفر شروع به شمارش کنید. بنابراین ، هر فاصله زمانی را می توان با یک ماژول تایمر بدست آورد.

یک ماژول تایمر همچنین می تواند در حالت شمارنده کار کند که منبع ساعت شناخته شده نیست و در واقع یک سیگنال خارجی است، شاید از یک دکمه فشاری. ، بنابراین شمارنده هر لبه بالا یا پایین از فشار دکمه افزایش می یابد. این حالت می تواند در کاربردهای متعدد سودمند باشد که در ادامه نشان خواهیم داد. نمودار زیر را در نظر بگیرید.



همانطور که می توانید ببینید ، سیگنال ساعت اکنون از دکمه فشار ایجاد می شود و از طریق پیش مقیاس به ورودی ساعت تایمر می رسد و شما می توانید اطلاعات چند بار فشار دادن دکمه را با خواندن مقدار ثبات TCNT دریافت کنید.

انواع مختلفی از تایمر ها در میکروکنترلر های STM32 وجود دارد. هر کدام می توانند در حالت های مختلف کار کنند و کارهای زیادی انجام دهند. ما می توانیم حالت ها و قابلیت های تایمر را با استفاده از رجیسترهای خاص مربوط به تایمر کنترل یا تغییر دهیم. قبل از اینکه به این موضوع برسیم ، باید درباره ساعت(clock) در STM32 بیشتر بدانیم.

4.3 ساعت سیستم

ساعت سیستم(System Clock) قلب میکروکنترلر است. بدون ساعت ، ریزپردازنده نمیتواند کار کند. ما می توانیم ساعت را به سه روش مختلف به STM32 ارائه دهیم:

- ساعت نوسان ساز HSI(High-Speed Internal Clock)
- ساعت نوسان ساز HSE(High-Speed External Clock)
- ساعت PLL اصلی(Main PLL Clock)

HSI یا HSE بر اساس پیکربندی ما انتخاب می شود و خروجی به PLL داده می شود. با استفاده از این PLL می توانیم مقدار System Clock را کنترل (افزایش یا کاهش) کنیم. هنگامی که کلاک سیستم تولید شد، این کلاک با استفاده از گذرگاه AHB(باس پیشرفته با

کارایی بالا)، گذرگاه APB1(باس پیشرفته جانبی کم سرعت)، و گذرگاه APB2(باس پیشرفته جانبی پرسرعت) بین واحدهای جانبی و سایر واحدها توزیع می‌شود. . هر یک از این کلاک ها را می توان با استفاده از پیش مقیاس کننده ها تنظیم کرد.

22.2 حالت های کاری تایمر در STM32

با استفاده ار ثبات ها، ما میتوانیم تایمرها را تنظیم کنیم تا کارهای زیادی را انجام دهند. مهمترین حالت هایی که تایمر ها میتوانند مورداستفاده قرار بگیرند به قرار زیر است:

- **حالت زمان سنج(Timer mode)**
- **حالت شمارنده(Counter mode)**
- **حالت ضبط ورودی(Input Capture mode)**
- **حالت PWM(PWM mode)**

22.2.1حالت زمان سنج

تایمر می تواند به عنوان یک مولد پایه زمانی استفاده شود. بسته به پارامترهای کلاک ، پیش مقیاس و بارگذاری مجدد خودکار(auto-reload) ، تایمر 16 بیتی می تواند یک رویداد به روز رسانی را از نانو ثانیه تا چند دقیقه ایجاد کند که میتواند برای تولید تاخیر ، عملیات خاص انجام شده در هر فاصله زمانی خاص (عملیات دوره ای) و غیره استفاده شود.

22.2.2حالت شمارنده

در این حالت می توانیم رویدادهای خارجی را بشماریم. مقدار تایمر بر اساس رویدادهای خارجی افزایش خواهد یافت. در این حالت ، تایمر ساعت را از منبع خارجی از طریق پین خارجی ورودی تایمر دریافت می کند. این حالت در بسیاری از موقعیت ها مفید است به خصوص وقتی که شما نیاز به پیاده سازی یک شمارنده دیجیتال بدون استفاده از خواندن دوره ای GPIO دارید.

22.2.3حالت ضبط ورودی

تایمر می تواند در حالت ضبط ورودی برای اندازه گیری یک سیگنال خارجی استفاده شود. حالت ضبط ورودی یکی دیگر از ویژگی های مفید تایمر است که معمولا برای شمارش فرکانس استفاده می شود. ماژول Input Capture وظیفه دارد که مقدار فعلی شمارنده تایمر را برای یک رویداد ورودی ضبط کند. قابلیت ضبط ورودی در بسیاری از کاربردها استفاده میشود مانند:

- اندازه گیری عرض پالس
- اندازه گیری دوره یا زمان بندی) به طور دقیق مدت زمان بین افزایش و کاهش لبه های ورودی های دیجیتال
- ثبت زمان یک رویداد

22.2.4 حالت PWM

در این حالت ، ما می توانیم یک موج مربع با یک چرخه کار و فرکانس خاص تولید کنیم. با استفاده از این ، ما می توانیم اقدامات زیادی مانند تغییر روشنایی نور ، سرعت موتور و غیره انجام دهیم.

22.3 انواع تایمر در STM32

هر ریزپردازنده نوع STM32 دارای چندین تایمر داخلی است. آنها از TIM1 تا TIM20 شماره گذاری می شوند و به انواع مختلف گروه بندی می شوند:

- تایمرهای با همه منظوره(General Purpose Timers)
- تایمرهای با کنترل پیشرفته(Advance Control Timers)
- تایمرهای پایه(Basic Timers)
- تایمرهای کم مصرف(Low Power Timers)

ریزپردازنده های سری STM32F4 که در این آزمایشگاه استفاده میشوند دارای 13 ماژول تایمر میباشند(TIM1-TIM14) (به استثنای TIM8)که به صورت زیر دسته بندی میشوند:

- TIM1- تایمر با کنترل پیشرفته
- TIM2 تا TIM5- تایمرهای همه منظوره
- TIM9 تا TIM14- تایمرهای همه منظوره
- TIM6 و TIM7- تایمرهای پایه

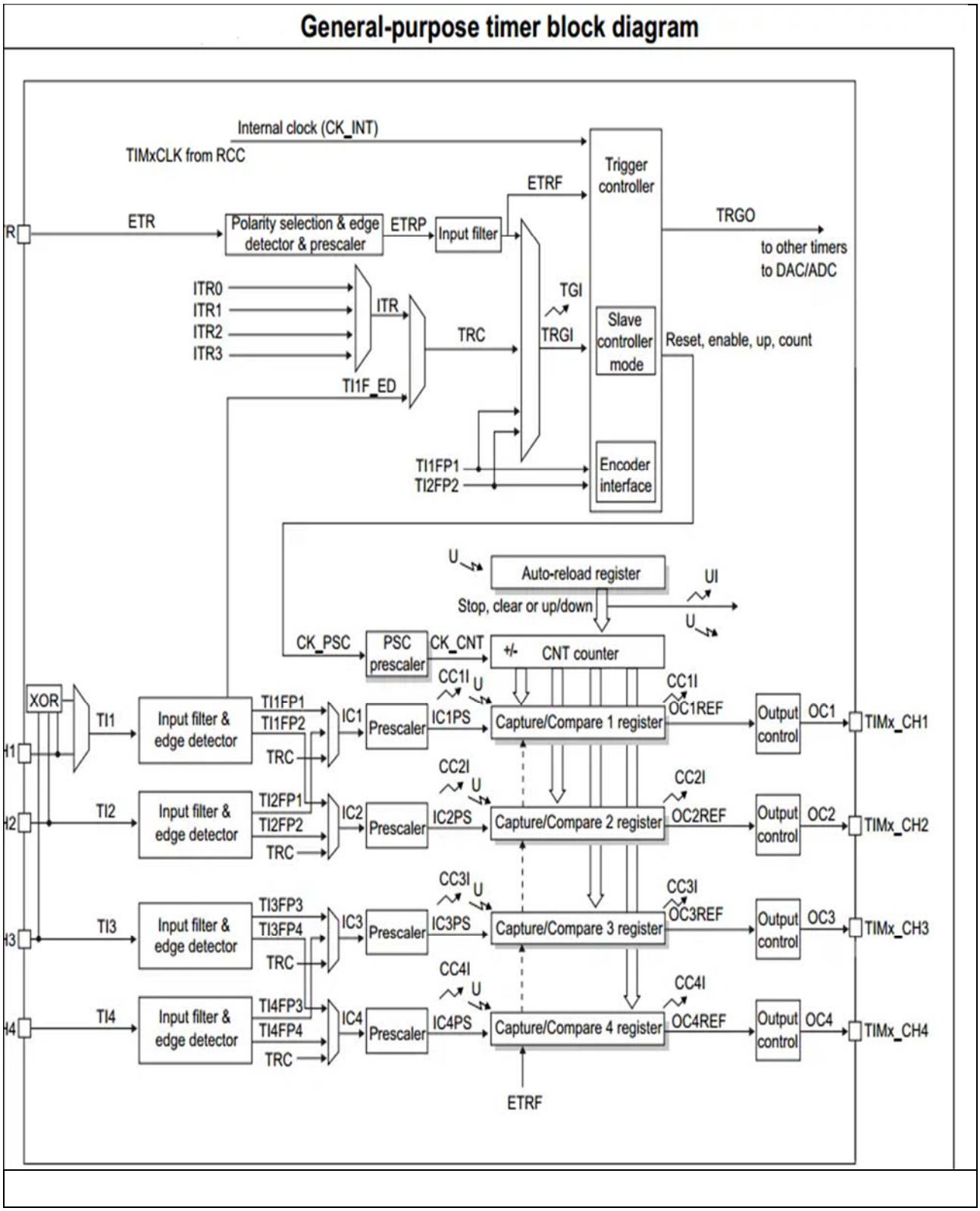
به دلیل گستردگی مبحث تایمرها، در این آزمایشگاه فقط تایمرهای با کارکرد کلی را بررسی میکنیم.

22.4 تایمرهای همه منظوره(General-Purpose Timers)

تایمرهای همه منظوره شامل یک شمارنده بارگیری مجدد خودکار(auto-reload counter) 16 بیتی یا 32 بیتی است که توسط یک پیش مقیاس کننده قابل برنامه ریزی هدایت می شود. این تایمر ها برای اهداف مختلفی استفاده می شوند و می توانند در حالت ضبط ورودی ، حالت مقایسه خروجی و حالت PWM با برنامه نویسی پیکربندی های مختلف استفاده شوند .تایمر ها کاملا مستقل هستند و هیچ منبعی را به اشتراک نمی گذارند. برخی از مهمترین ویژگی های تایمرهای همه منظوره عبارتست از:

- شمارنده بارگیری مجدد خودکار 16 یا 32 بیتی (بر اساس میکروکنترلر) بالا، پایین، بالا/پایین.
- پیش مقیاس کننده قابل برنامه ریزی 16 بیتی که برای تقسیم فرکانس ساعت شمارنده بر هر عاملی بین 1 و 65535 استفاده می شود.
- حداکثر 4 کانال مستقل برای:
  1. ضبط ورودی
  2. مقایسه خروجی(Output Compare)
  3. PWM
  4. خروجی حالت تک پالس(One Pulse Mode)
- تولید Interrupt/DMA برای حالت های مختلف
- پشتیبانی از مدارهای انکودر افزایشی و سنسور هال برای اهداف موقعیت یابی

در این آزمایشگاه ما تایمرهای همه منظوره(TIM1 تا TIM5) را در STM32F4بررسی خواهیم کرد. نمودار بلوکی تایمرهای همه منظوره در شکل زیر آمده است:



حال نگاهی به ثبات های موجود در تایمر های همه منظوره STM32F4 می اندازیم:

1. TIMx control register 1 (TIMx_CR1)
2. TIMx control register 2 (TIMx_CR2)
3. TIMx slave mode control register (TIMx_SMCR)
4. TIMx DMA/Interrupt enable register (TIMx_DIER)
5. TIMx status register (TIMx_SR)
6. TIMx event generation register (TIMx_EGR)
7. TIMx capture/compare mode register 1 (TIMx_CCMR1)
8. TIMx capture/compare mode register 2 (TIMx_CCMR2)
9. TIMx capture/compare enable register (TIMx_CCER)
10. TIMx counter (TIMx_CNT)
11. TIMx prescaler (TIMx_PSC)
12. TIMx auto-reload register (TIMx_ARR)
13. TIMx capture/compare register 1 (TIMx_CCR1)
14. TIMx capture/compare register 2 (TIMx_CCR2)
15. TIMx capture/compare register 3 (TIMx_CCR3)
16. TIMx capture/compare register 4 (TIMx_CCR4)
17. TIMx DMA control register (TIMx_DCR)
18. TIMx DMA address for full transfer (TIMx_DMAR)
19. TIM2 option register (TIM2_OR)
20. TIM5 option register (TIM5_OR)

20 ثبات وجود دارند و قصد نداریم همه ثبات ها را توضیح دهیم؛ ما فقط چند ثبات اساسی را در پیکربندی تایمر توضیح خواهیم داد. بنابراین ، لطفا برای جزئیات بیشتر به کتابچه راهنمای کاربر (STM32F407 user manual) مراجعه کنید.

قبل از اینکه سراغ برنامه ریزی تایمرها برویم لازم است بدانیم هر یک از تایمرهای روی چه باسی قرار دارند. در حالت کلی باس APB1 و APB2 به تایمرها متصل هستند. تایمر 2، 3، 4، 5، 6، 7، 12، 13 و 14 روی باس APB1 هستند.

(RCC_APB1ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	Reser-ved	CAN2 EN	CAN1 EN	Reser-ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	Reser-ved
		rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWDG EN	Reserved		TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw			rw	rw	rw	rw	rw	rw	rw	rw	rw

(RCC_APB2ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													TIM11 EN	TIM10 EN	TIM9 EN
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reser-ved	SYSCFG EN	Reser-ved	SPI1 EN	SDIO EN	ADC3 EN	ADC2 EN	ADC1 EN	Reserved		USART6 EN	USART1 EN	Reserved		TIM8 EN	TIM1 EN
	rw		rw	rw	rw	rw	rw			rw	rw			rw	rw

و تایمر های 1، 8، 9، 10 و11 به APB2 متصل هستند.

ثبات های اصلی که در پیکر بندی تایمر 3 استفاده می شوند شامل رجیسترهای ذیل هستند.

- رجیستر شمارنده (TIMx_CNT)

TIMx counter (TIMx_CNT)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

حاوی مقدار شمارنده تایمر در هر لحظه می‌باشد.

- رجیستر پری‌اسکالر (TIMx_PSC)

TIMx prescaler (TIMx_PSC)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

فرکانسی که به شمارنده تایمر اعمال میگردد طبق رابطه ذیل محاسبه می شود.

$$f_{timer} = \frac{f_{clk_{PSC}}}{PSC[15:0] + 1}$$

$f_{clk_{PSC}}$ فرکانس APB1 است.

- رجیستر بارگذاری خودکار (TIMx_ARR)

TIMx auto-reload register (TIMx_ARR)

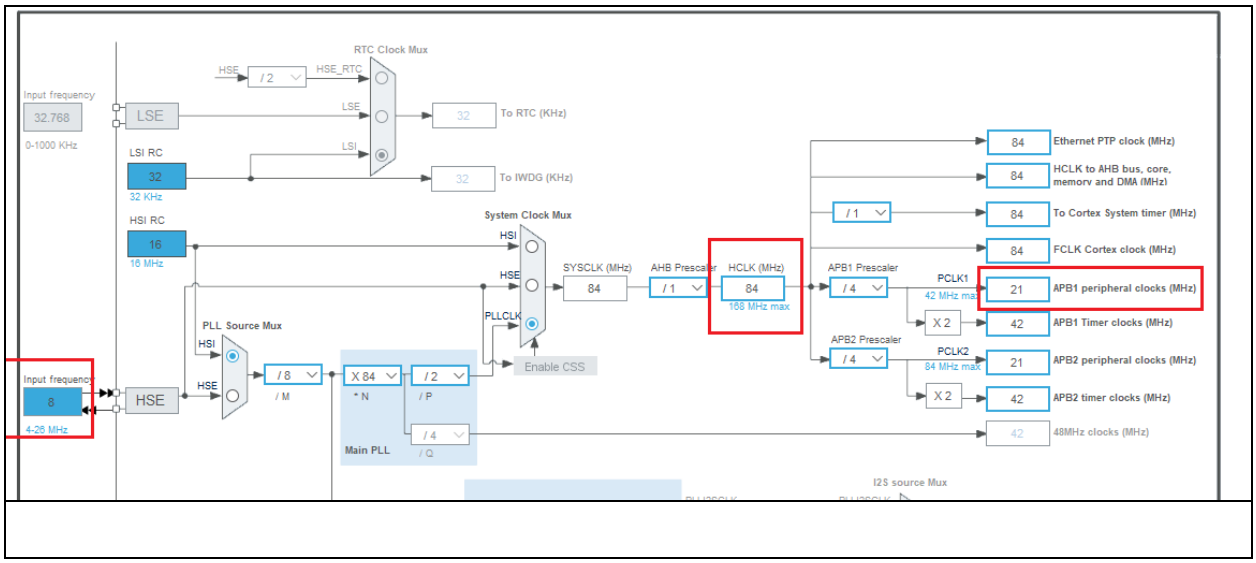
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

در این رجیستر حد شمارش تایمر محدود میگردد. در حالت عادی تایمر میتواند تا 65535 بشمارد و پس از وقفه سرریز مجدد از صفر شروع میکند. ولی با تنظیم این رجیستر حد شمارش صفر تا محتوای TIMx_ARR میباشد.

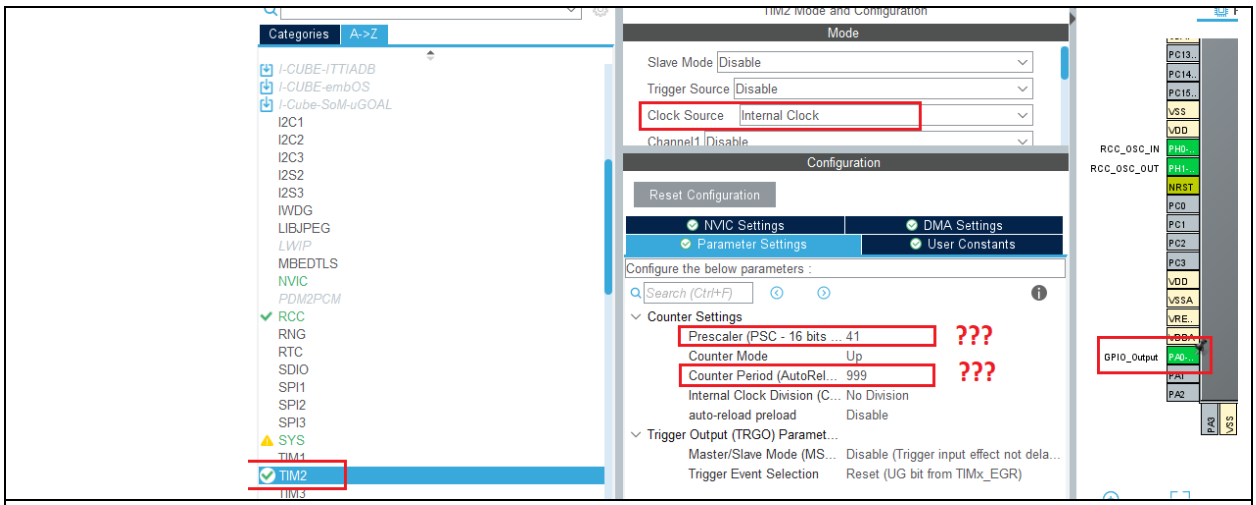
22.5 پروژه LED چشمک زن با استفاده از تایمر 2

برای طراحی LED چشمک زن با فرکانس دقیق به تایمر ها نیاز داریم بدین منظور از تایمر 2 استفاده شده است. ابتدا در محیط stm32cubemx تنظیمات لازم را مطابق ذیل انجام داده و سپس در محیطKeil کدهای لازم را اضافه نمایید.

ابتدا کلاک باس APB1 متصل به تایمر دو را تنظیم نمایید.



تنظیمات تایمر دو را مانند شکل انجام دهید. همچنین یک پایه دلخواه را به عنوان خروجی در نظر گرفته تا به LED متصل گردد.



در تنظیمات دو تایمر دو پارامتری که با علامت سوال ?? مشخص شده از اهمیت زیادی برخوردارند و محتوای رجیستر TIMx_PSC و TIMx_ARR را تنظیم می نماید.

فرض نمایید فرکانس APB1 برابر با 21MHz می باشد. اگر فرکانس تایمر برابر با 250KHz باشد مقدار رجیستر TIMx_PSC برابر است با :

$$\mathrm{TIMx_{PSC}} = \frac{21MHZ}{250KHz} - 1 = 83$$

با فرکانس تایمر برابر با 250KHz هر واحد شمارش 4us طول میکشد. لذا اگر مقدار رجیستر TIMx_ARR به مقدار 49999 تنظیم گردد تایمر هر 0.2 ثانیه وقفه تولید مینماید. و در زیر برنامه وقفه میتوان وضعیت LED را تغییر داد.

$$T_{overflow} = \frac{(PSC + 1)(ARR + 1)}{f_{APB1}}$$

کدهای تهیه شده را در محیط keil مشابه کد زیر تغییر دهید.

Main.c

```
#include "main.h"

TIM_HandleTypeDef htim2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
if(htim->Instance == TIM2){
        HAL_GPIO_TogglePin(GPIOC,GPIO_PIN_0);
                                    میتوان وضعیت led را در این زیربرنامه یا در TIM2_IRQHandler(void) تغییر داد.

}
}

int main(void)
{

  HAL_Init;()

  SystemClock_Config;()

  MX_GPIO_Init;()
  MX_TIM2_Init;()

 HAL_TIM_Base_Start_IT(&htim2);

  while(1)
  {
}
}
```

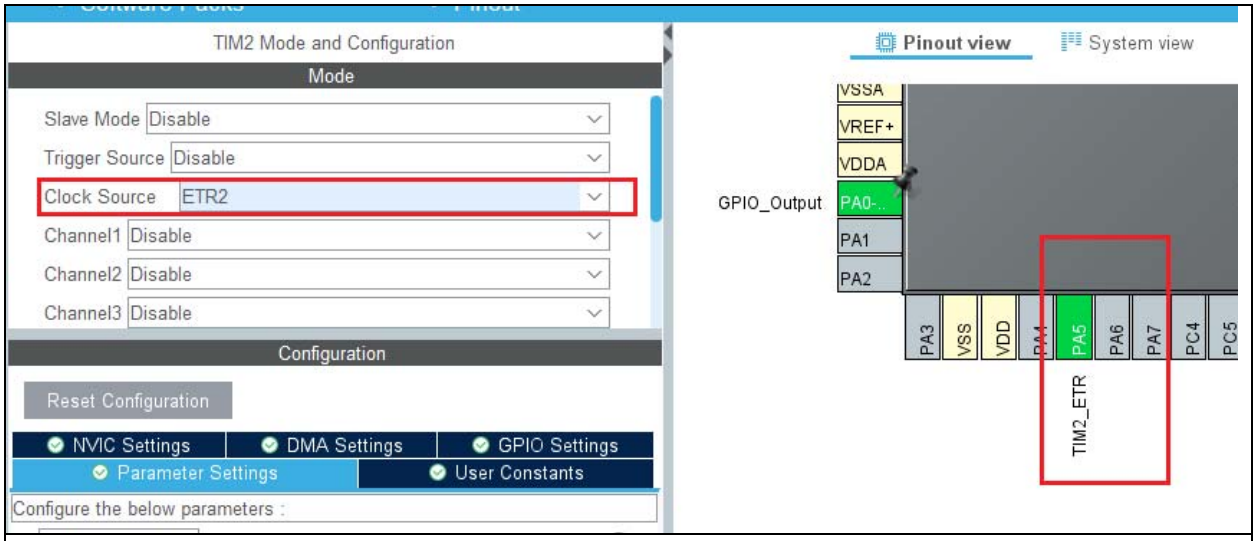

نسخه پیشنویس(ارسال اشکال فنی و نگارشی به zmohammadzadeh@of.iut.ac.ir)

stm32f4xx_it.c

```
void TIM2_IRQHandler(void)
{

  HAL_TIM_IRQHandler(&htim2);
        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_0);
                                    میتوان وضعیت led را در این زیربرنامه یا در HAL_TIM_PeriodElapsedCallback تغییر داد.

}
```

22.6 مد شمارنده در تایمر

اگر از تایمر کاربرد شمارنده انتظار داشته باشید میتوان از کلاک خارجی برای آآن استفاده کرد که تنظیمات آن در شکل نشان داده شده است. در این حالت پایه مشخصی برای ورودی کلاک نیز به صورت خودکار در نظر گرفته می شود.



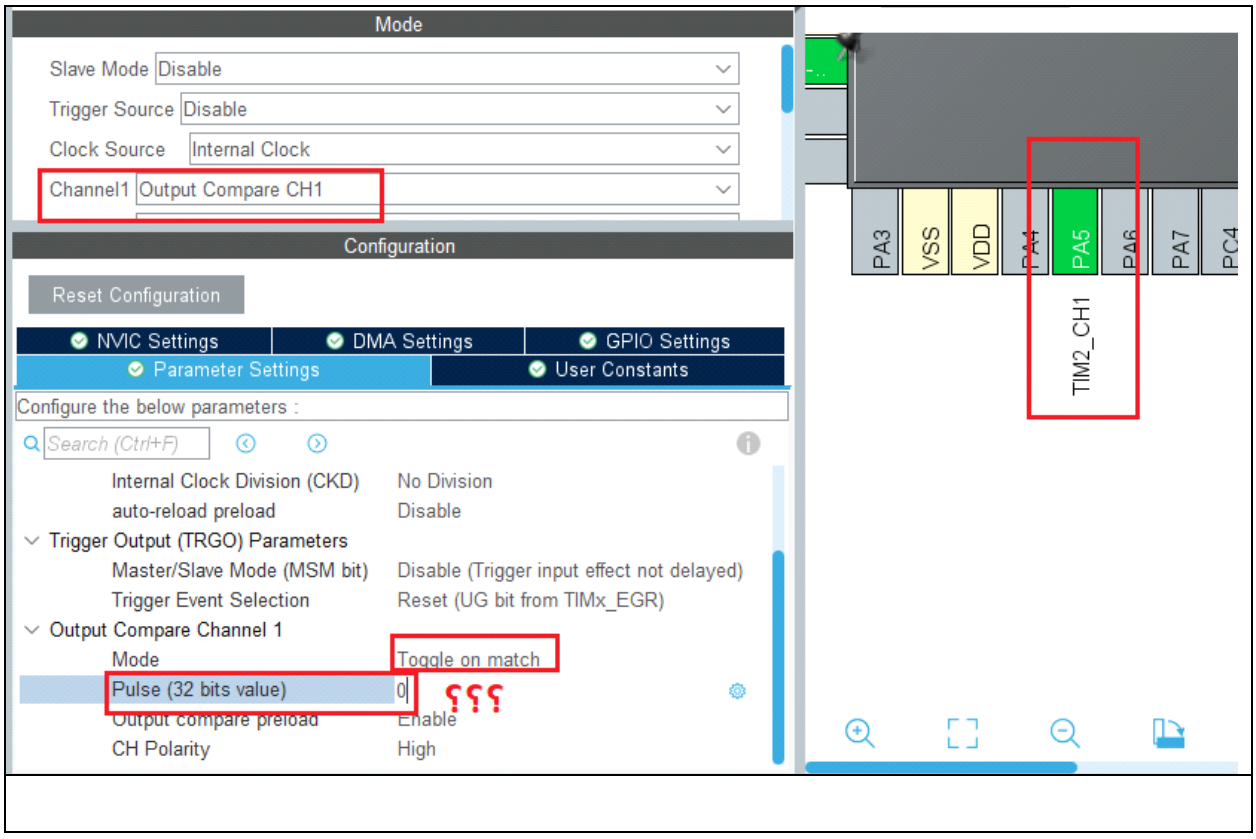
TIMx capture/compare register 1 (TIMx_CCR1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCR1[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCR1[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

شرح	نام	بیت
مقدار باارزش در تایمر 2 و 5	**CCR1[31:16]**	31:16
مقدار کم ارزش	CCR1[15:0]	15:0

در این مد هرگاه مقدار رجیستر شمارنده TIMx_CNTبه مقدار موجود در رجیستر TIMx_CCR1 برسد، یکی از پایه های میکرو که به تایمر متصل است تغییر میکند و امکان فعال کردن وقفه نیز وجود دارد.

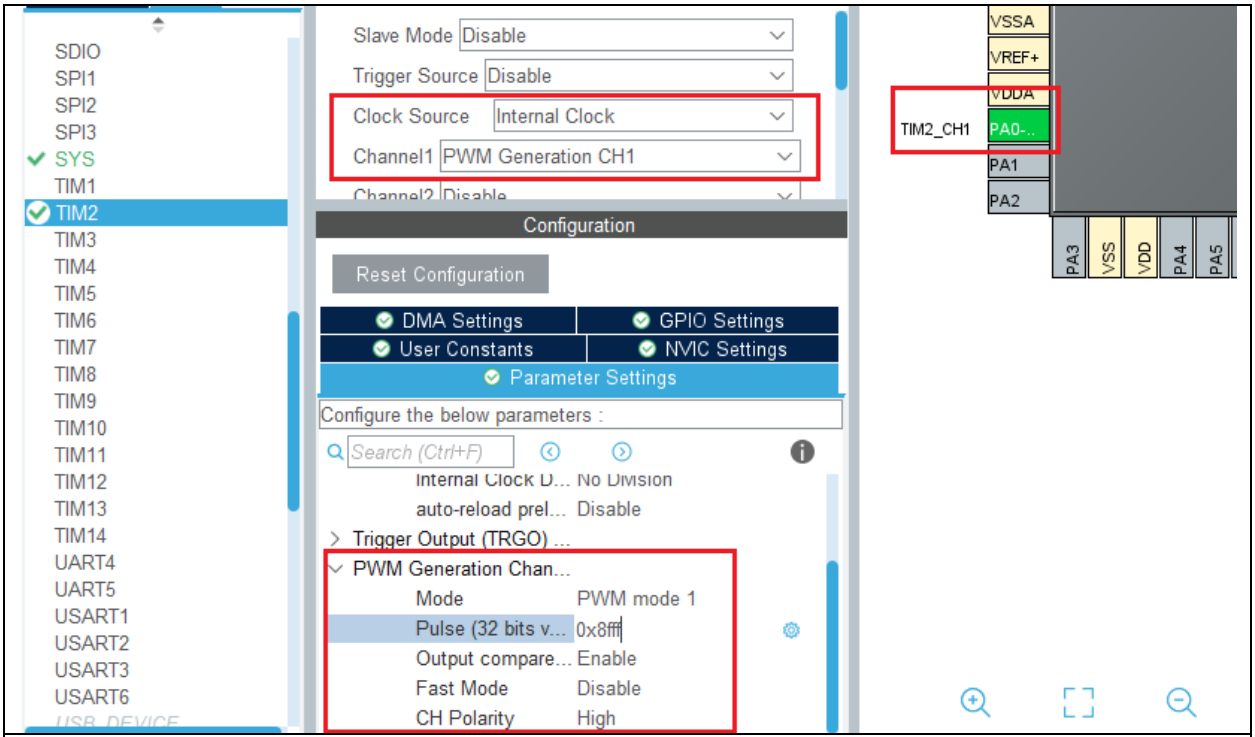


در تصویر مقدار رجیستر TIMx_CCR1 تنظیم میشود که بایستی بین صفر و TIMx_ARR تنظیم شود.

```
#include "main.h"

TIM_HandleTypeDef htim2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);

int main(void)
{

  HAL_Init;()


  SystemClock_Config;()

  MX_GPIO_Init;()
  MX_TIM2_Init;()

  HAL_TIM_Base_Start_IT(&htim2);
 HAL_TIM_OC_Start(&htim2, TIM_CHANNEL_1);

  while(1)
{

}

}
```

22.8 مد PWM

در این مد میتوان شکل موج های مربعی با دوره تناوب مشخص و عرض پالس متغیر ایجاد نمود. فرکانس شکل موج به رجیستر TIMx_ARR و عرض پالس به TIMx_CCRx بستگی دارد. در این حالت هرگاه محتوای شمارنده یا TIMx_CNT به TIMx_ARR و TIMx_CCRx میرسد پایه خروجی تغییر وضعیت می‌دهد.

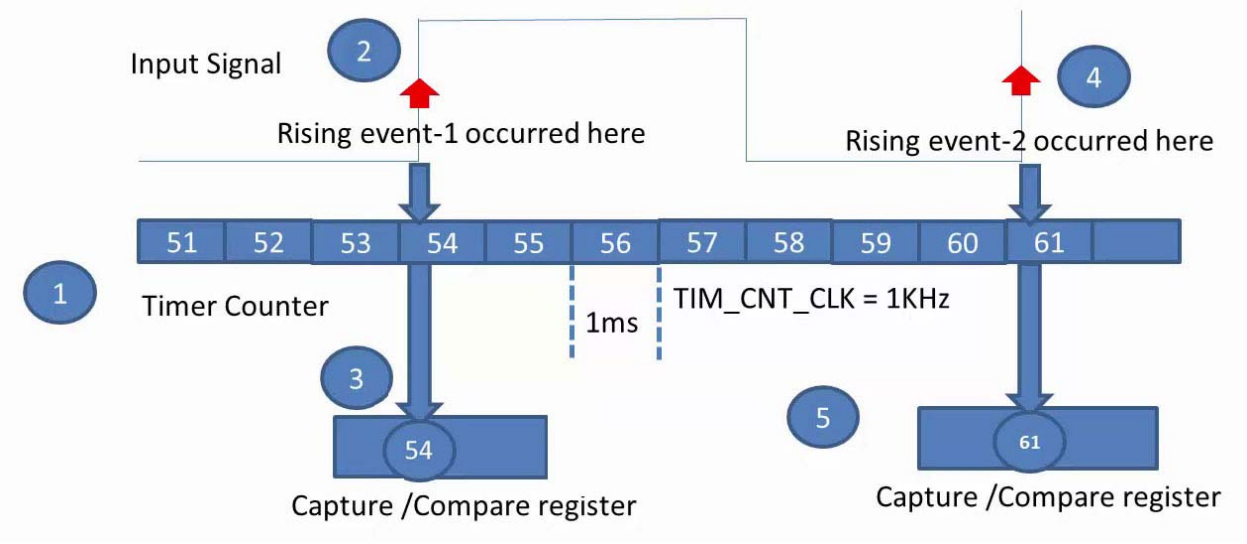



```
#include "main.h"

TIM_HandleTypeDef htim2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);

int main(void)
{

  HAL_Init;()

  SystemClock_Config;()

  MX_GPIO_Init;()
  MX_TIM2_Init;()



        HAL_TIM_PWM_Init(&htim2);
        HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

```
while(1)
{

}
}
```

22.9 مد input capture

در حالت ضبط ورودی، میتوان زمان رخداد یک پدیده یا سیگنال دیجیتال را با شناسایی لبه بالارونده یا پایین رونده ثبت نمود. این پدیده به پایه ICx متصل میشود و و در صورت شناسایی لبه مورد نظر مقدار شمارنده تایمر در یک ثبات به نام TIMx_CCRx ذخیره میگردد و پرچم CCxIF در رجیست TIMx_SR تنظیم میگردد. در سرویس دهی به این رخداد اولین کاری که انجام میشود منتقل کردن مقدار ثبات TIMx_CCRx به یک متغیر دلخواه میباشد تا در صورتی که داده جدیدی ثبت شد، داده قبلی از بین نرود. مراحل این پروسه در شکل نشان داده شده است.



1- برای اندازه‌گیری سیگنال ورودی ابتدا باید شمارنده‌ی تایمر را با فرکانس مشخص، به‌صورت بالا شمار (یا پایین شمار) راه‌اندازی کنیم

2- سپس سیگنالی که می‌خواهیم فرکانس آن را اندازه‌گیری کنیم را به یکی از کانال‌های تایمر که بر روی پین میکروکنترلر قرار دارد، متصل می‌کنیم و وقفه‌ی این کانال را فعال و حساس به لبه‌ی بالارونده (یا پایین‌رونده) قرار می‌دهیم. در این صورت در هر لبه‌ی بالارونده، یک وقفه رخ می‌دهد

3- پس‌ازاینکه وقفه‌ی مربوط به لبه‌ی بالارونده رخ داد، مقدار شمارنده در لحظه وقوع وقفه، در رجیستر TIMx_CCRxذخیره می‌شود

4- شمارنده همچنان به شمارش خود ادامه خواهد داد. در حالی که شمارنده به شمارش خود ادامه می‌دهد، بر روی دومین لبه‌ی بالارونده سیگنال یک وقفه‌ی دیگر رخ می‌دهد

5- پس‌ازاینکه وقفه‌ی مربوط به دومین لبه‌ی بالارونده رخ داد، مقدار شمارنده در لحظه وقوع وقفه، دوباره در رجیستر TIMx_CCRx ذخیره می‌شود

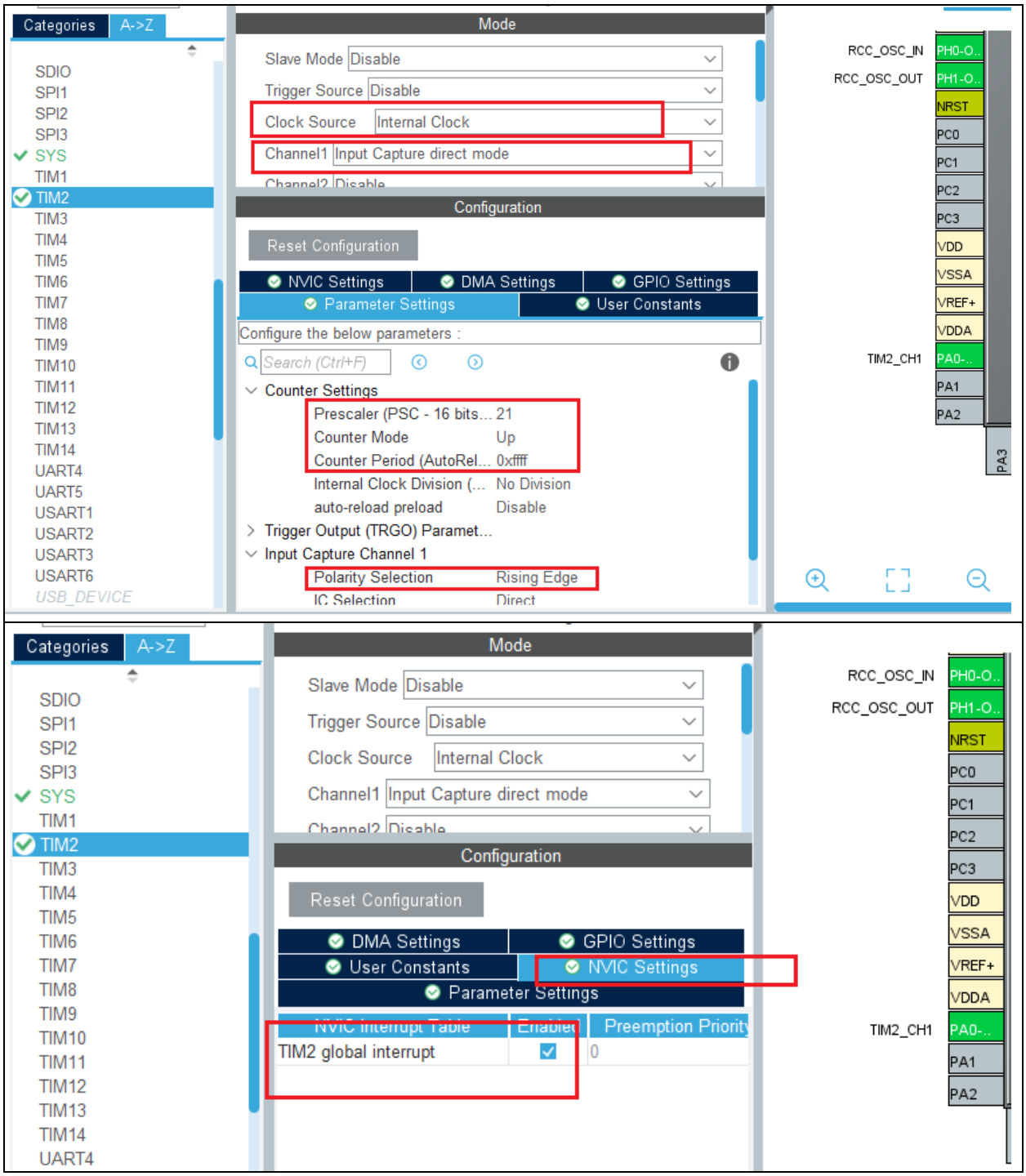
6- محاسبه فرکانس سیگنال ورودی:

$$f_{in} = \frac{f_{timer}}{D2 - D1}$$

F_{in} فرکانس سیگنال ورودی ، F_{timer}: فرکانس تایمر ، D2 دومین داده ضبط شده در رجیستر TIMx_CCRx و D1 اولین داده ضبط شده در رجیستر TIMx_CCRx میباشد.

نکته ضروری که باید به آن توجه داشت این است که در هر سرریز تایمر چندین داده ضبط شده باشد و چنانچه داده ای در دو سیکل متفاوت شمارش تایمر ضبط گردد برای محاسبه فرکانس ورودی استفاده نگردد.

تنظیمات لازم برای فعال کردن مد input capture در محیط Cubemx در شکل نشان داده شده است.



بعد از تولید کد در محیط keil تغییرات ذیل را در برنامه ایجاد نمایید.

Filename: main.c
#include "main.h"

```
TIM_HandleTypeDef htim2;

uint32_t capture_value = 0;


void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);


void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
   if (htim->Instance == TIM2)
   {
      capture_value = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1)  ;
      HAL_GPIO_TogglePin(GPIOC,GPIO_PIN_2) ;
   }
 در این قسمت می توان فرآیند محاسبه فرکانس ورودی را اضافه کرد.

}


int main(void)
{


  HAL_Init;()

  SystemClock_Config;()

  MX_GPIO_Init;()
  MX_TIM2_Init;()

  HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1);


  while(1)
  {
                                                  در این قسمت میتوان فرکانس سیگنال ورودی را روی LCD نمایش داد.
}

}
```

Filename: stm32f4xx_it.c

```
void TIM2_IRQHandler(void)
{
```

```
  if (TIM2->SR & TIM_SR_UIF) {
      TIM2->SR &= ~TIM_SR_UIF;
     HAL_GPIO_TogglePin(GPIOC,GPIO_PIN_1);
سرریز شد تایمر را نشان میدهد.

   }
         if (TIM2->SR & TIM_SR_CC1IF) {
      //TIM2->SR &= ~TIM_SR_CC1IF;
اگر این خط اجراشود پرچم مربوطه پاک شده و وارد زیربرنامه محاسبه فرکانس ورودی نمیشود.
     HAL_GPIO_TogglePin(GPIOC,GPIO_PIN_0);

   }


  HAL_TIM_IRQHandler(&htim2);

}


}
```

22.10 برنامه های HAL

به منظور توضیح عملکرد زیر برنامه‌های ارائه شده در کد، می‌توان آن‌ها را به چهار گروه اصلی تقسیم کرد: توابع مدیریت زمان، توابع خروجی مقایسه (Output Compare)، توابع PWM و توابع ورودی Capture (Input Capture). هر کدام از این توابع برای کنترل و مدیریت تایمرها در میکروکنترلرها طراحی شده‌اند. در ادامه به شرح این توابع می‌پردازیم:

1. توابع مدیریت زمان (Time Base Functions)

این توابع برای تنظیم و کنترل عملیات کلی تایمر (Timer) استفاده می‌شوند.

- HAL_TIM_Base_Init(TIM_HandleTypeDef *htim)

- شرح: این تابع برای مقداردهی اولیه تایمر استفاده می‌شود. با فراخوانی این تابع، تنظیمات پیش‌فرض تایمر بارگذاری می‌شود.

- خروجی: وضعیت عملیات (مانند HAL_OK یا HAL_ERROR).

- HAL_TIM_Base_DeInit(TIM_HandleTypeDef *htim)

- شرح: این تابع برای آزادسازی منابع و غیرفعال‌سازی تایمر استفاده می‌شود.

- HAL_TIM_Base_MspInit(TIM_HandleTypeDef *htim)

- شرح: این تابع برای انجام پیکربندی‌های سخت‌افزاری مرتبط با تایمر در سطح *MCU* که مخصوص هر میکروکنترلر است، فراخوانی می‌شود. مثل فعال‌سازی کلاک یا تنظیمات GPIO.

- HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef *htim)

- شرح: این تابع برای غیرفعال کردن پیکربندی‌های سخت‌افزاری مربوط به تایمر استفاده می‌شود.

- HAL_TIM_Base_Start(TIM_HandleTypeDef *htim)

- شرح: این تابع تایمر را در حالت Blocking و با استفاده از Polling آغاز می‌کند.

- HAL_TIM_Base_Stop(TIM_HandleTypeDef *htim)

- شرح: با این تابع می‌توان تایمر را متوقف کرد.

- HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)

- شرح: این تابع تایمر را در حالت Interrupt آغاز می‌کند، به طوری که پس از رسیدن به پریود، یک وقفه ایجاد می‌شود.

- HAL_TIM_Base_Stop_IT(TIM_HandleTypeDef *htim)

- شرح: این تابع برای متوقف کردن تایمر در حالت Interrupt استفاده می‌شود.

- HAL_TIM_Base_Start_DMA(TIM_HandleTypeDef *htim, const uint32_t *pData, uint16_t Length)

- شرح: این تابع تایمر را برای انتقال داده‌ها با استفاده از DMA آغاز می‌کند.

- HAL_TIM_Base_Stop_DMA(TIM_HandleTypeDef *htim)

- شرح: این تابع تایمر را از حالت DMA متوقف می‌کند.

2. توابع خروجی مقایسه (Output Compare Functions)

این توابع برای تنظیم و کنترل حالت خروجی مقایسه (Output Compare) تایمرها طراحی شده‌اند.

- HAL_TIM_OC_Init(TIM_HandleTypeDef *htim)

- شرح: مشابه تابع Init برای تایمرهای پایه، این تابع برای مقداردهی اولیه تنظیمات خروجی مقایسه استفاده می‌شود.

- HAL_TIM_OC_DeInit(TIM_HandleTypeDef *htim)

- شرح: برای غیرفعال‌سازی منابع و تنظیمات خروجی مقایسه استفاده می‌شود.

- HAL_TIM_OC_MspInit(TIM_HandleTypeDef *htim)

- شرح: برای انجام تنظیمات سخت‌افزاری خاص برای خروجی مقایسه.

- HAL_TIM_OC_MspDeInit(TIM_HandleTypeDef *htim)

- شرح: برای پاک‌سازی تنظیمات سخت‌افزاری خروجی مقایسه.

- HAL_TIM_OC_Start(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: آغاز کار خروجی مقایسه برای یک کانال خاص.

- HAL_TIM_OC_Stop(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: توقف کار خروجی مقایسه برای یک کانال خاص.

- HAL_TIM_OC_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: آغاز عملیات خروجی مقایسه در حالت Interrupt.

- HAL_TIM_OC_Stop_IT(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: توقف عملیات خروجی مقایسه در حالت Interrupt.

- HAL_TIM_OC_Start_DMA(TIM_HandleTypeDef *htim, uint32_t Channel, const uint32_t *pData, uint16_t Length)

- شرح: آغاز عملیات خروجی مقایسه با استفاده از DMA.

- HAL_TIM_OC_Stop_DMA(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: توقف عملیات خروجی مقایسه در حالت DMA.

3. توابع PWM (Pulse Width Modulation Functions)

این توابع برای ایجاد سیگنال‌های PWM (مدولاسیون عرض پالس) استفاده می‌شوند.

- HAL_TIM_PWM_Init(TIM_HandleTypeDef *htim)

- شرح: مقداردهی اولیه تایمر برای تولید سیگنال PWM.

- HAL_TIM_PWM_DeInit(TIM_HandleTypeDef *htim)

- شرح: غیرفعال‌سازی منابع PWM و تنظیمات مربوطه.

- HAL_TIM_PWM_MspInit(TIM_HandleTypeDef *htim)

- شرح: انجام تنظیمات سخت‌افزاری خاص برای PWM.

- HAL_TIM_PWM_MspDeInit(TIM_HandleTypeDef *htim)

- شرح: پاک‌سازی تنظیمات سخت‌افزاری مربوط به PWM.

- HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: آغاز کار PWM برای یک کانال خاص.

- HAL_TIM_PWM_Stop(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: توقف کار PWM برای یک کانال خاص.

- HAL_TIM_PWM_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: آغاز PWM در حالت Interrupt.

- HAL_TIM_PWM_Stop_IT(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: توقف PWM در حالت Interrupt.

- HAL_TIM_PWM_Start_DMA(TIM_HandleTypeDef *htim, uint32_t Channel, const uint32_t *pData, uint16_t Length)

- شرح: آغاز PWM با استفاده از DMA.

- HAL_TIM_PWM_Stop_DMA(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: توقف PWM در حالت DMA.

4. توابع ورودی Capture (Input Capture Functions)

این توابع برای دریافت داده‌های سیگنال‌های ورودی استفاده می‌شوند.

- HAL_TIM_IC_Init(TIM_HandleTypeDef *htim)

- شرح: مقداردهی اولیه برای حالت ورودی Capture.

- HAL_TIM_IC_DeInit(TIM_HandleTypeDef *htim)

- شرح: غیرفعال‌سازی منابع و تنظیمات مربوط به ورودی Capture.

- HAL_TIM_IC_MspInit(TIM_HandleTypeDef *htim)

- شرح: انجام تنظیمات سخت‌افزاری خاص برای ورود Capture.

- HAL_TIM_IC_MspDeInit(TIM_HandleTypeDef *htim)

- شرح: پاک‌سازی تنظیمات سخت‌افزاری مربوط به ورودی Capture.

- HAL_TIM_IC_Start(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: آغاز کار ورودی Capture برای یک کانال خاص.

- HAL_TIM_IC_Stop(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: توقف کار ورودی Capture برای یک کانال خاص.

- HAL_TIM_IC_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: آغاز ورودی Capture در حالت Interrupt.

- HAL_TIM_IC_Stop_IT(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: توقف ورودی Capture در حالت Interrupt.

- HAL_TIM_IC_Start_DMA(TIM_HandleTypeDef *htim, uint32_t Channel, uint32_t *pData, uint16_t Length)

- شرح: شروع ورودی Capture با استفاده از DMA.

- HAL_TIM_IC_Stop_DMA(TIM_HandleTypeDef *htim, uint32_t Channel)

- شرح: توقف ورودی Capture در حالت DMA.

در STM32، تأسیس و استفاده از تایمرها نیاز به پیاده‌سازی برخی از زیر برنامه‌ها (callback functions) دارد. هر کدام از این زیر برنامه‌ها برای یک نوع خاص از event یا رخداد مربوط به تایمرها طراحی شده‌اند. در زیر، توضیحات مربوط به هر یک از این زیر برنامه‌ها و شرایط فعال شدن آن‌ها آورده شده است:

1. void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)

- شرح: این تابع زیر برنامه در زمانی فراخوانی می‌شود که تایمر به پریود (Period) خود برسد، یعنی زمان تعیین‌شده (مقدار پیش‌فرض یا پیکربندی‌شده) به پایان رسیده است. معمولاً برای انجام کارهایی مانند تغییر وضعیت LED، ضبط داده‌ها یا راه‌اندازی سایر زیرروال‌ها استفاده می‌شود.

- شرایط فعال شدن: زمانی که تایمر در حال شمارش است و به مقدار پریود خود می‌رسد.

2. void HAL_TIM_PeriodElapsedHalfCpltCallback(TIM_HandleTypeDef *htim)

- شرح: این تابع مشابه تابع قبلی است و زمانی فراخوانی می‌شود که تایمر به نیمه پریود (Half Period) خود برسد.

- شرایط فعال شدن: اگر پریود تایمر به دو نیمه تقسیم شود (در حالت‌های خاصی از کار با تایمرها) و به نیمه اول آن برسد.

3. void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim)

- شرح: این تابع زمانی فراخوانی می‌شود که تایمر در حالت «خروجی مقایسه» (Output Compare - OC) و پس از یک تأخیر مشخص شده، خرجی خود را فعال یا غیرفعال می‌کند.

- شرایط فعال شدن: وقتی که رویداد خروجی مقایسه با تأخیر به پایان می‌رسد.

4. void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)

- شرح: این تابع برای زمان‌هایی است که تایمر در حالت «ورودی (Input Capture - IC) «Capture و یک ورودی شناسایی شده را ثبت می‌کند.

- شرایط فعال شدن: زمانی که لبه (rising/Falling edge) قابل‌تشخیص از سیگنال ورودی ثبت شود.

5. void HAL_TIM_IC_CaptureHalfCpltCallback(TIM_HandleTypeDef *htim)

- شرح: عملکردی مشابه با تابع `HAL_TIM_IC_CaptureCallback` دارد، اما فقط برای نیمه اول داده‌های Capture.

- شرایط فعال شدن: وقتی که رویداد Capture به نیمه اول داده‌ها برسد.

6. void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim)

- شرح: این تابع زمانی فراخوانی می‌شود که پالس PWM به انتهای خود رسیده باشد.

- شرایط فعال شدن: در پایان پالس خروجی PWM، معمولاً برای مدیریت پالس‌های PWM در سیستم‌های کنترل موتور یا روشنایی.

7. void HAL_TIM_PWM_PulseFinishedHalfCpltCallback(TIM_HandleTypeDef *htim)

- شرح: مشابه تابع قبلی، ولی برای نیمه اول پالس PWM.

- شرایط فعال شدن: در انتهای نیمه اول پالس خروجی PWM.

8. void HAL_TIM_TriggerCallback(TIM_HandleTypeDef *htim)

- شرح: این تابع در پاسخ به یک رویداد Trigger از تایمر فراخوانی می‌شود. معمولاً برای شروع یک عمل ثبت داده یا ایجاد سیگنال‌های کنترلی استفاده می‌شود.

- شرایط فعال شدن: زمانی که یک رویداد Trigger در تایمر به وقوع بپیوندد.

9. void HAL_TIM_TriggerHalfCpltCallback(TIM_HandleTypeDef *htim)

- شرح: مشابه تابع TriggerCallback است، ولی برای نیمه اول پالس Trigger.

- شرایط فعال شدن: وقتی که Trigger به نیمه اول آن برسد.

10. void HAL_TIM_ErrorCallback(TIM_HandleTypeDef *htim)

- شرح: این تابع در صورتی که یک خطا در تایمر رخ دهد (مانند overrun یا configuration error) فراخوانی می‌شود.

- شرایط فعال شدن: زمانی که تایمر با یک مشکل مواجه می‌شود؛ در این صورت، می‌تواند به کاربر اطلاعاتی در مورد وضعیت بدهد.