SysTic timer

- Every ARM Cortex M.

contador

- Down counter STCURRENT (Systick->VAL). cAN RECEIVE CLOCK from 2 diff sources.
- The clock ssc is chosen using CLK_SRC bit of STCTRL (SysTick->CTRL) register.

Registros

STCTRL (SysTick Control and Status)

- 0 ENABLE: 0 diasbled, 1 enabes systick
- 1 TICKINT: 0 int dis, 1 when systick counts to 0 int is generated.
- 2 CLKSOURCE: 0 AHB clock divideb by 8 - AHB clock
- 16 COUNTFLAG: 0 SysTick has not counted dt0 - 1: systick has counted to zero
- NOTE: COUNTLAG is clared by reading STRCTRL or writin STCURRENT.

SysTick Reload value register (STRELOAD)

STREOAD is used to program de start value of ystick down counter. the stcurrent register, dth streload sohuld contain the value n0 1 for d=the count to fire envery n clock cycles

STM32F10x TIMERS

Delay generation

Types of Timers

- Basic
- General purpose (Timers 2, 3 and 4)
- Advanced control timers (Timer 1)

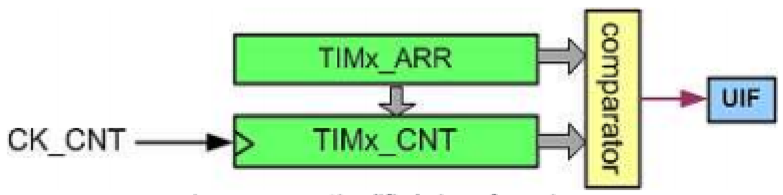


Figure 11-11: A Simplified View of TIM timers

Registers

CNT (Counter)

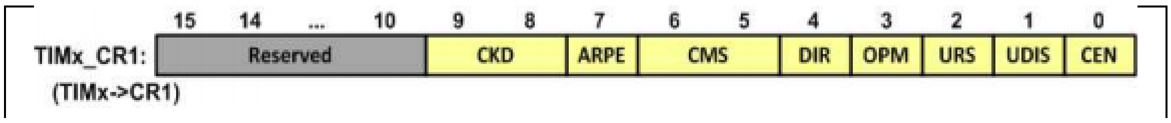
- 16-bit up/down counter.
- TIMx_CNT to load value

ARR Auto Reload Register

- 16 bit
- Continously compared with the TIMx_CNT reg

TIMx_CR1 (Control Registe)

- ways: up, down, and both like a yoyo



- 9-8 CKD: Clock division used by dead-time
- 7 ARPE: Auto reload preload enable 0 ARR not buff, 1 Buffered
- 6-5 Center-aligned mode selection
  - 00 up or down
- 4 DIR: is CMS is 00, dir chooses te direction of counting. 0 ups, 1 down
- 3 OPM: one pulse mode 0 continously, 1 counter stops at the next update event
- 2 URS update request source
- 1 UDIS update disable (mask)
- 0 CEN counter enable 0 DIS, 1 ENA

TIMx_SR (Status Register)

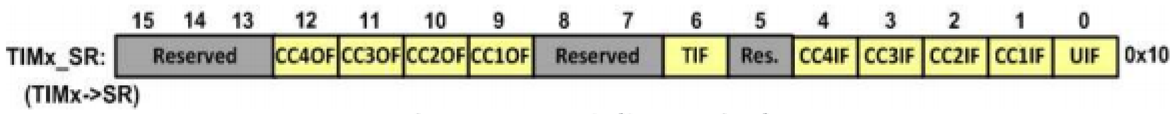


Figure 11-15: TIMx_SR (Status Register)

- 0 UIF: Update Interrupt Flag.
- When conuting up mode, when the value CNT reaches ARR, UIF flag is SET. The lag remains high until it is clered by software in the counting down mode. The flag is se twhn conuter sreaches 0

Counting Up

- When TIMx_CNT counter is counting up, it is compared with the ocuntents of TIMx_ARR register.
- When CNT and ARR are equal UIF flag goes up indicating is a match and CNT rolls over to zero.

Enabling Clock

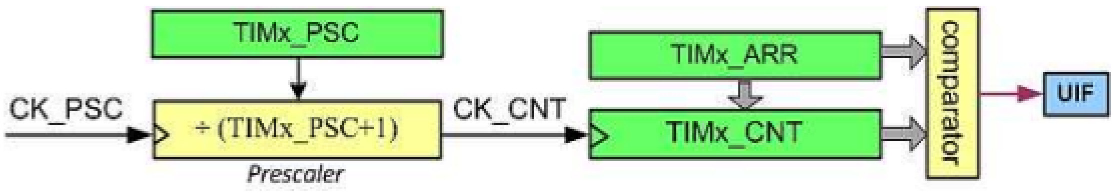
For each timer we have a separate clock enable bit in RCC_APB1ENR and RCC_APB2ENr registers.

Making Delays using TIM timer in up counting mode

1. Enable the clock to TIMx module
2. load TIMx_ARR register with proper value
3. Clear UIF flag.
4. Set mode as up couter timer and enable timer
5. Wait for UIF flag to go HIGH

6. Stop the timer

Prescaler

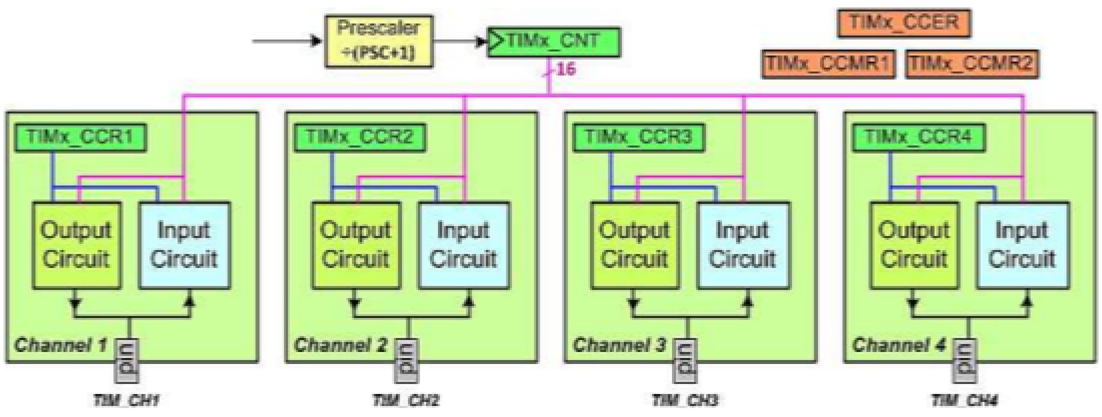


- Contains a 16 bit up counter and a comparator
- up-counter counts from - t value of TIMx_PSC reg.

```
/*
- ARR = 71 -> 72 states (clk goes from 0 to 71)
- default timer clk *72MHz -> each clock lasts 1/72MHz
- so delay = 71 * 1/72Mhz = 1us
*/
void delay()
{
    RCC->APB1ENR |= (1<<0); /* enable TIM2 clk */
    TIM2->ARR = 71;
    TIM2->SR  = 0; /* clear UIF flag */
    TIM2->CR1 = 1; /* up counting */
    while((TIM2->SR & 1) == 0); /* wait until UIF flag is set */
    TIM2->CR1 = 0; /* stop counting */
}

void delayConPrescaler()
{
    RCC->APB1ENR |= (1<<0);
    TIM2->PSC = 7200-1;   /* psc = 7199 */
    TIM2->ARR = 500-1;

    TIM2->SR = 0;  /* clear UIF */
    TIM2->CR1 = 1; /* up counting */
    while((TIM2->SR & 1) == 0);
    TIM2->CR1 = 0; /* stop counting */
}


void delayOnePulseMode() //timer stops automatically
{
    RCC->APB1ENR |= (1<<0); /* enable TIM2 clk */
    TIM2->PSC = 7200-1;
    TIM2->ARR = 500 - 1;
    TIM2->SR  = 0; /* clear UIF flag */
    TIM2->CR1 = 9; /* up counting & OnePulseMode*/
    while((TIM2->SR & 1) == 0); /* wait until UIF flag is set */
}
```

Programming Output Compare

- In some apps we need control digital output transition with PRECISION TIMING.
- In STM32 each TIMx has 4 channels
- Each ch has an input circuit for input capturing and output circuit for output compare.

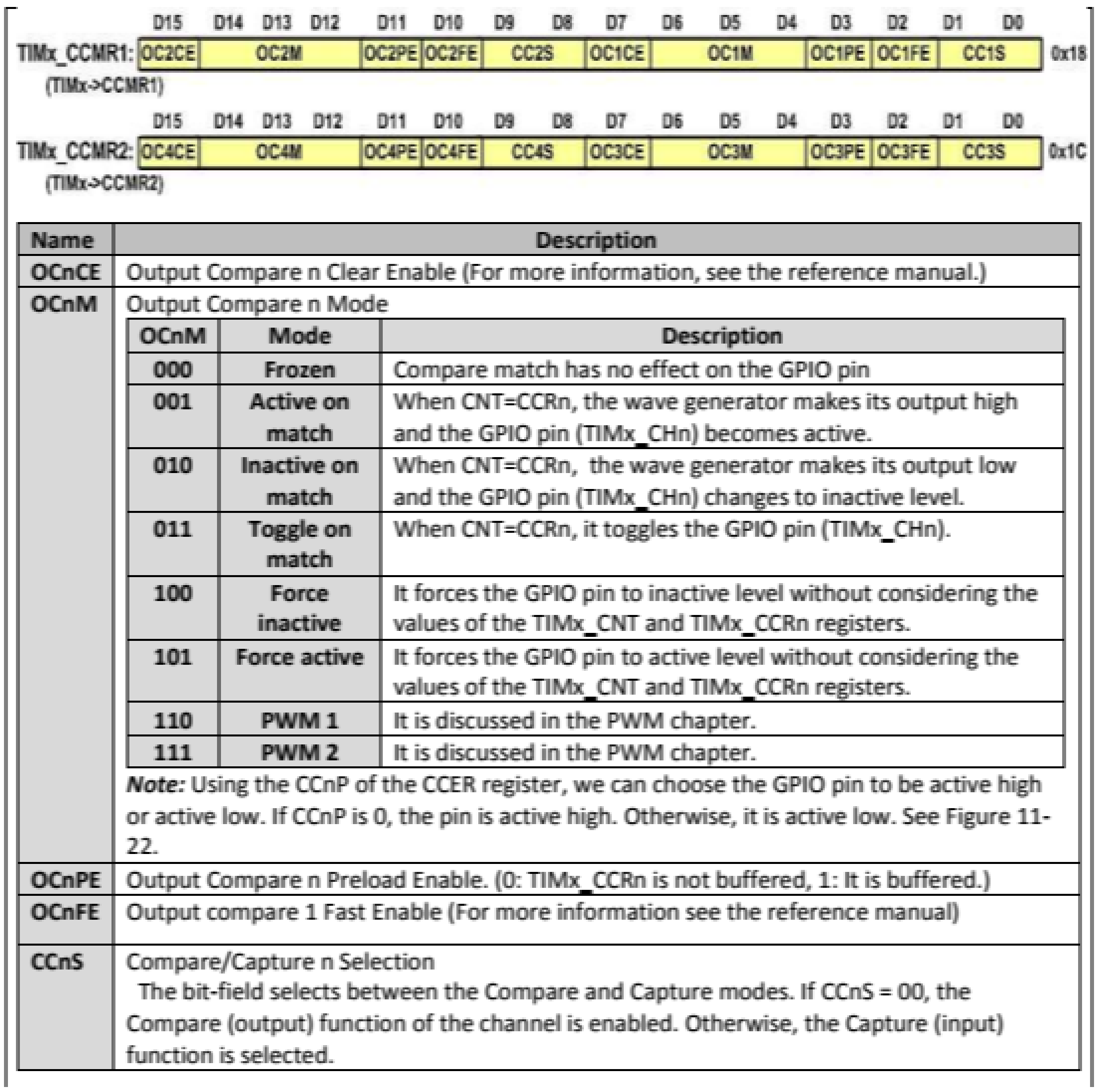


Cap/Comp Reg (TIMx_CCRn)

- CCR1 to 4.
- Readable and writable
- TIMx_CCRn is compared with TIMx_CNT after each clock cycle.
- When CNT and CCRn match, CCnOF flag is set HIGH
- There is a WAVE GENERATOR in each chanel, making pin go low to high (toggling ping).

Cap/Comp Mode Reg (TIMx_CCMR1 TIM_CCMR2)

- D0-7, D8-15, D16-31, ... for ch1, ch2, ch3, and ch4
- Each ch has CCnS (Comp/Capt Selection)
- CCnS bits are used to enable the input or output circuit of the ch
- CCnS = 00, output circuit is enable and channel is in Compare mode.
- OCnM select the behavior of the wave generator con copare match (TIMx_CNT = TIMx_CCRn). -> the wave generator might SET the pin, make low or toggle it. Example: if OCnM = 011 the wave gen toggles the GPIO on compare match



	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
TIMx_CCMR1: (TIMx->CCMR1)	OC2CE	OC2M			OC2PE	OC2FE	CC2S		OC1CE	OC1M			OC1PE	OC1FE	CC1S		0x18
TIMx_CCMR2: (TIMx->CCMR2)	OC4CE	OC4M			OC4PE	OC4FE	CC4S		OC3CE	OC3M			OC3PE	OC3FE	CC3S		0x1C

Name	Description
OCnCE	Output Compare n Clear Enable (For more information, see the reference manual.)
OCnM	Output Compare n Mode

OCnM	Mode	Description
000	Frozen	Compare match has no effect on the GPIO pin
001	Active on match	When CNT=CCRn, the wave generator makes its output high and the GPIO pin (TIMx_CHn) becomes active.
010	Inactive on match	When CNT=CCRn, the wave generator makes its output low and the GPIO pin (TIMx_CHn) changes to inactive level.
011	Toggle on match	When CNT=CCRn, it toggles the GPIO pin (TIMx_CHn).
100	Force inactive	It forces the GPIO pin to inactive level without considering the values of the TIMx_CNT and TIMx_CCRn registers.
101	Force active	It forces the GPIO pin to active level without considering the values of the TIMx_CNT and TIMx_CCRn registers.
110	PWM 1	It is discussed in the PWM chapter.
111	PWM 2	It is discussed in the PWM chapter.

Note: Using the CCnP of the CCER register, we can choose the GPIO pin to be active high or active low. If CCnP is 0, the pin is active high. Otherwise, it is active low. See Figure 11-22.

Name	Description
OCnPE	Output Compare n Preload Enable. (0: TIMx_CCRn is not buffered, 1: It is buffered.)
OCnFE	Output compare 1 Fast Enable (For more information see the reference manual)
CCnS	Compare/Capture n Selection The bit-field selects between the Compare and Capture modes. If CCnS = 00, the Compare (output) function of the channel is enabled. Otherwise, the Capture (input) function is selected.

Cap/Comp Enable Register (TIMx_CCER)

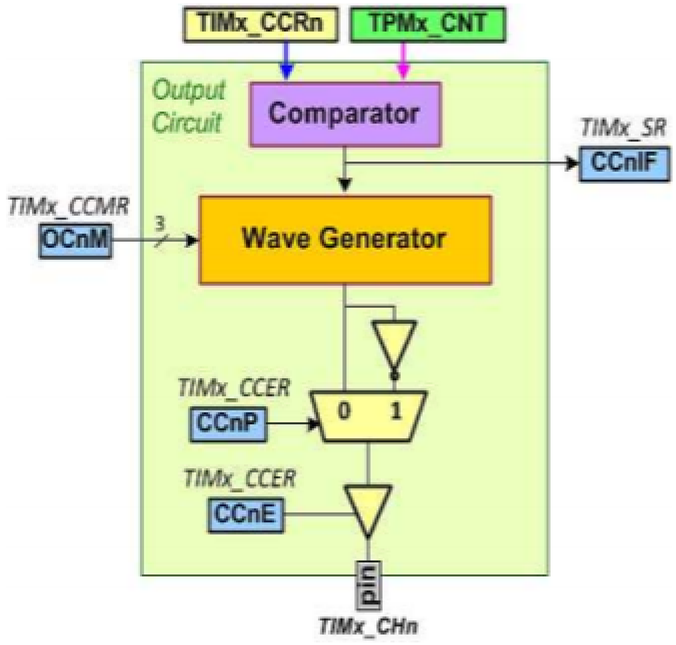
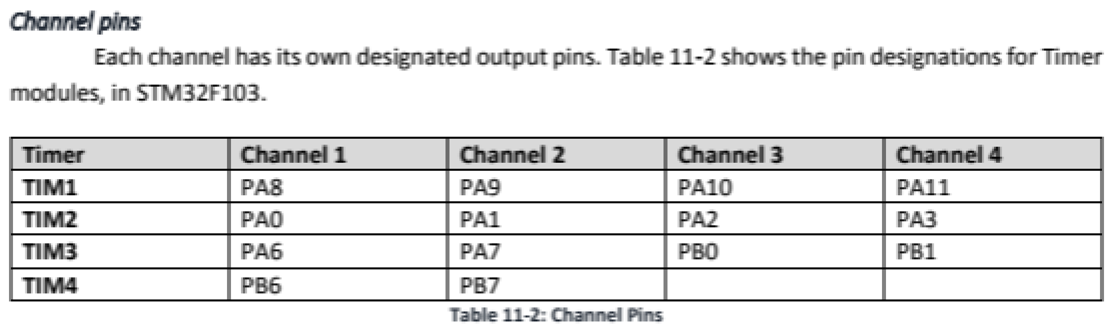


Figure 11-22: Output Circuit

- if CCnP=0 the output of the wave generator goes directly to the GPIO pin.
- if CCnP=1 the output signal becomes INVERTED and goes to the GPIO pin.

- We must set the CCnE bit to 1 to let the wave generator change the GPIO pin.

Channels



Channel pins

Each channel has its own designated output pins. Table 11-2 shows the pin designations for Timer modules, in STM32F103.

Timer	Channel 1	Channel 2	Channel 3	Channel 4
TIM1	PA8	PA9	PA10	PA11
TIM2	PA0	PA1	PA2	PA3
TIM3	PA6	PA7	PB0	PB1
TIM4	PB6	PB7		

Table 11-2: Channel Pins

Wave Generation Examples

Example 1: Simple Wave

- Timer counts and toggles CH3 when CNT=CCR3.
- CCR3 is set to 200, so CH3 toggles after 200 clocks
- The CH3 pin toggles agin on next CNT=CCR3

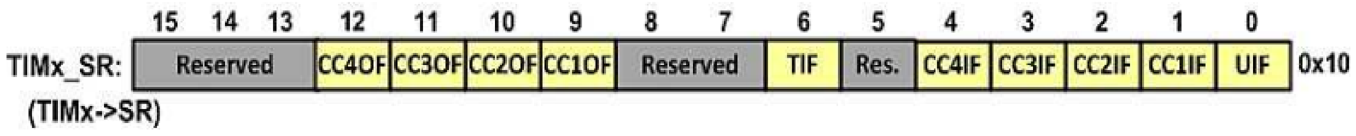
```
int main()
{
    RCC->APB2ENR |= 0xFC; /* enable gpio clocks */
    RCC->APB1ENR |= (1<<0); /* enable TIM2 clock */
    GPIO->CRL = 0x44444B44 /* PA2: Alternate func output */

    TIM2->CCR3 = 200;
    TIM2->CCER = 0x1 <<8; /* CC3P = 0, CC3E = 1 */
    TIM2->CCMR2= 0x0030 /*toggle channel 3 */
    TIM2->ARR = 10000-1;
    TIM2->CR1 = 1; /* start counting */

    while(1)
    {
    }
}
```

Example 2

```
int main()
{
    RCC->APB2ENR |= 0xFC;   /* enable GPIO clocks */
    RCC->APB1ENR |= (1<<0); /* enable TIM2 clock  */
    GPIO->CRL = 0x44444BB4; /* PA2 (ch3), PA1 (ch2): alternate func. output */

    TIM2->CCR2 = 1000;
    TIM2->CCR3 = 3000;
    TIM2->CCER = (0x1<<8) | (0x1<<4); /* CC3P=0 CC3E=1 CC2E=1 */
    TIM2->CCMR1 = 0x3000; /* toggle channel 2 */
    TIM2->CCMR2 = 0x0030; /* toggle channel 3 */
    TIM2->PSC = 7200-1;
    TIM2->ARR = 10000-1;
    TIM2->CR1 = 1;
}
```

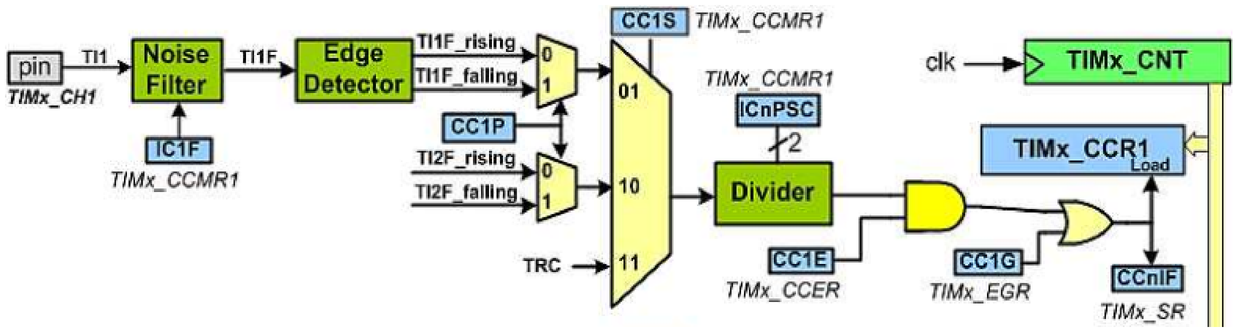
Timer Status Register (TIMx_SR)

- Status flags for all the channels to monitor the status of all channels with a single read of the register.



Timer for input capturing

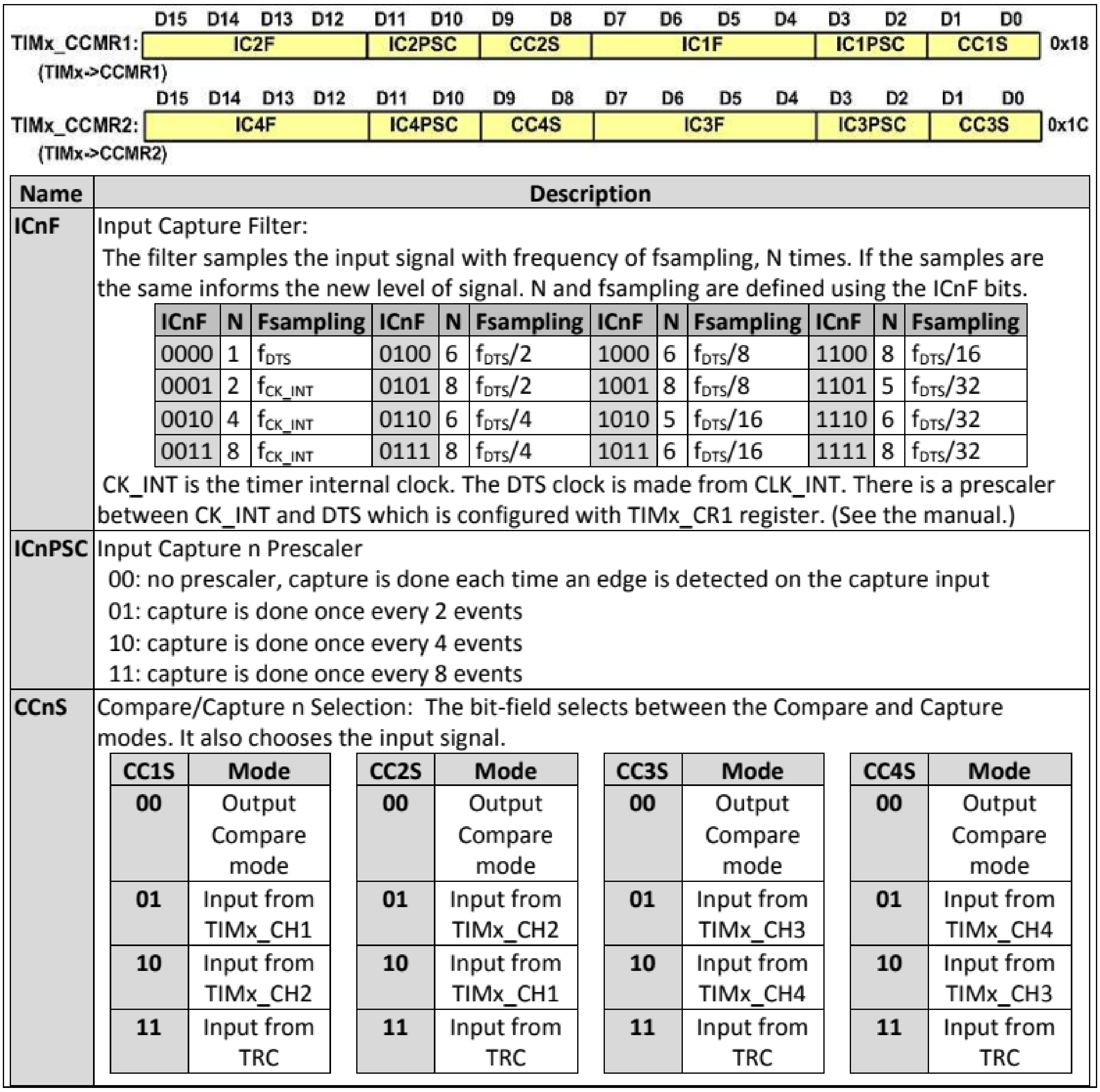
- the IO pin of the channel is used to capture the signal transition events.
- when event occurs, content of TIMx_CNT is captured and saved in the CCRn register of the channel



1. The singal is filtered by the Noise filter: NF makes sure the signal is stable for a given number of clocks and then informs
2. Edge detector detects the rising and falling edges of the signal.

- Using CCnP of CCER we choose to capture RISING or FALLING edge.
- Using CCnS we choose the channel to be INPUT or OUTPUT mode. Also to choose the input signal. Example CC1S=01 captures TIMx_CH1 pin.

3. The divider can be used to set the ratio if you like to capture after 4 edges you can set ICnPSC to 10.



	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
TIMx_CCMR1: (TIMx->CCMR1)	IC2F				IC2PSC		CC2S		IC1F				IC1PSC		CC1S		0x18
TIMx_CCMR2: (TIMx->CCMR2)	IC4F				IC4PSC		CC4S		IC3F				IC3PSC		CC3S		0x1C

Name	Description
ICnF	Input Capture Filter: The filter samples the input signal with frequency of fsampling, N times. If the samples are the same informs the new level of signal. N and fsampling are defined using the ICnF bits.

ICnF	N	Fsampling	ICnF	N	Fsampling	ICnF	N	Fsampling	ICnF	N	Fsampling
0000	1	f_{DTS}	0100	6	$f_{DTS}/2$	1000	6	$f_{DTS}/8$	1100	8	$f_{DTS}/16$
0001	2	f_{CK_INT}	0101	8	$f_{DTS}/2$	1001	8	$f_{DTS}/8$	1101	5	$f_{DTS}/32$
0010	4	f_{CK_INT}	0110	6	$f_{DTS}/4$	1010	5	$f_{DTS}/16$	1110	6	$f_{DTS}/32$
0011	8	f_{CK_INT}	0111	8	$f_{DTS}/4$	1011	6	$f_{DTS}/16$	1111	8	$f_{DTS}/32$

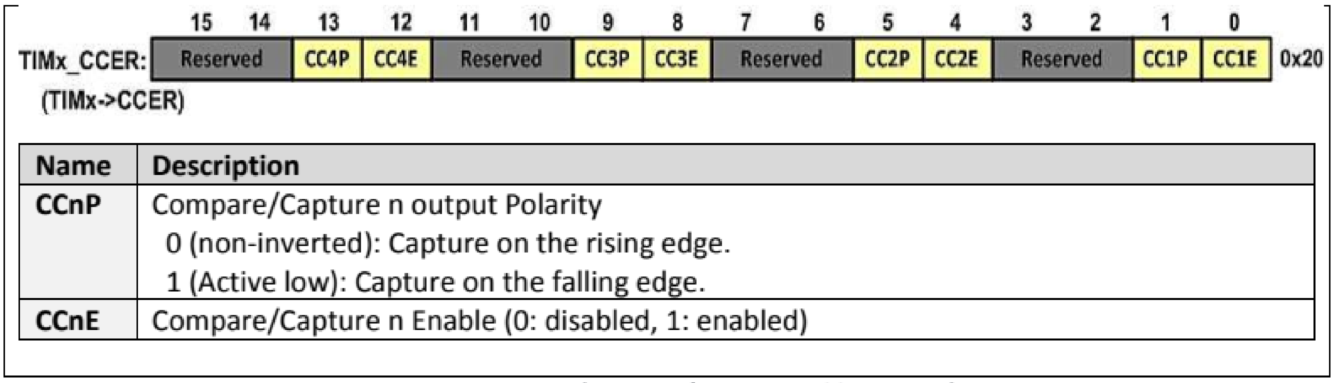
CK_INT is the timer internal clock. The DTS clock is made from CLK_INT. There is a prescaler between CK_INT and DTS which is configured with TIMx_CR1 register. (See the manual.)

Name	Description
ICnPSC	Input Capture n Prescaler 00: no prescaler, capture is done each time an edge is detected on the capture input 01: capture is done once every 2 events 10: capture is done once every 4 events 11: capture is done once every 8 events
CCnS	Compare/Capture n Selection: The bit-field selects between the Compare and Capture modes. It also chooses the input signal.

CC1S	Mode	CC2S	Mode	CC3S	Mode	CC4S	Mode
00	Output Compare mode	**00**	Output Compare mode	**00**	Output Compare mode	**00**	Output Compare mode
01	Input from TIMx_CH1	**01**	Input from TIMx_CH2	**01**	Input from TIMx_CH3	**01**	Input from TIMx_CH4
10	Input from TIMx_CH2	**10**	Input from TIMx_CH1	**10**	Input from TIMx_CH4	**10**	Input from TIMx_CH3
11	Input from TRC	**11**	Input from TRC	**11**	Input from TRC	**11**	Input from TRC

4. CCnE (CC Enable) is used to enable capturing. To capture, CCnE must be 1.
5. CCnG (CC Generation) of TIMx_EGR can be used to make a capture. If we set the bit, one capture takes place and then CCnG bit is cleared by hardware.

- On each capture, CCnIF flag of TIMx_SR is also set to 1 and remains 1 until is cleared by software.
- If we read TIMx_CCRn, CCnIF is cleared automatically.
- The bits of CCMR differ in output compare and input capture modes.
  - both modes the same bits are used as CCnS bits
  - When we set ch to input capture mode, the rolls of the other bits of the channel changes as shown



Name	Description
CCnP	Compare/Capture n output Polarity 0 (non-inverted): Capture on the rising edge. 1 (Active low): Capture on the falling edge.
CCnE	Compare/Capture n Enable (0: disabled, 1: enabled)

Figure 11-31: CCER (Compare/Capture Enable Register)

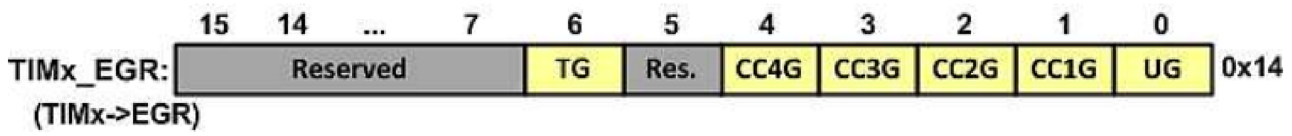
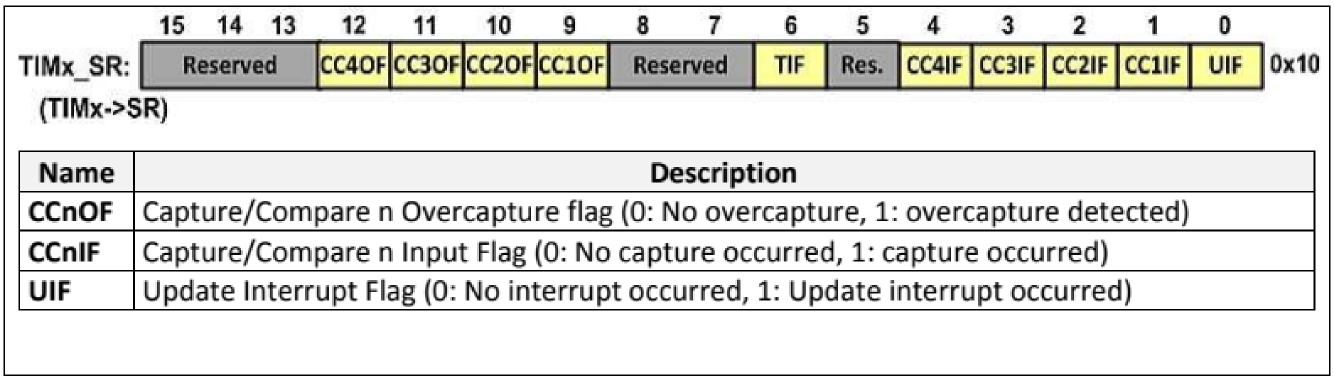


Figure 11-32: TIMx_EGR (Event Generation Register)



Name	Description
CCnOF	Capture/Compare n Overcapture flag (0: No overcapture, 1: overcapture detected)
CCnIF	Capture/Compare n Input Flag (0: No capture occurred, 1: capture occurred)
UIF	Update Interrupt Flag (0: No interrupt occurred, 1: Update interrupt occurred)

Steps to Program Input Capture

1. ENABLE THE CLOCK to the input pin GPIO port and TIMx module.
2. SELECT THE CHANNEL PIN as INPUT in the CRL/CRH registers.
3. SET the timer PRESCALER value.
4. Select the INPUT SIGNAL using CCMR1 and CCMR2.
5. Select the RISING/FALLING edge and ENABLE THE CAPTURE using CCER.
6. set the value of TIMx_ARR register
7. ENABLE TIMER
8. wait until the CCNIF bit is set in TIMx_SR.
9. READ the current counter value captured
10. calculate the current counter value difference from the last value.
11. save the current value for next calculation
12. repeat from step 8

Example: Measure the Freq of the wave and reports through usart1 */

```
#include <stm32f10x.h>
#include <stdio.h>
```

```
void usart1_init(void);
void usart1_sendByte(unsigned char c);
void usart1_sendInt(unsigned int i);
void usart1_sendStr(char *str);

int main()
{
    RCC->APB2ENR |= (0xFC | (1<<14)); /* enable GPIO clocks and USaRT1 clock */
    RCC->APB1ENR |= (1<<0); /* enable TIM2 clock */

    usart1_init();

    GPIOA->CRL = 0x44444844; /* PA2 (CH3): Input pull-up */
    GPIOA->ODR |= (1<<2);

    TIM2->CCMR2 = 0x001; /* pin TIM2_CH3 as input for Ch3 */
    TIM2->CCER = 0x1 << 8; /* CC3P=0 (rising), CC3E=1 (enable) */
    TIM2->PSC = 7200-1;
    TIM2->ARR = 50000-1;
    TIM2->CR1 = 1; /* start counting */

    uint16_t t = 0;
    uint16_t t0 = 0;

    while(1)
    {
        while((TIM2->SR & (1<<3)) == 0); /* wait until cc3if flag sets */
        t = TIM2->CCR3; /* read the captured value */

        usart1_sendInt(t - t-); /* send the difference */
        usart1_sendStr("\n\r"); /* go to new line */

        t0 = t;
    }
}

void usart1_init()
{
    GPIOA->ODR |= (1<<10); //pull-up PA10
    GPIOA->CRH |= 0x444448B4; //RX1=input with pull-up, TX1=alt. func output
    USART1->CR1 = 0x200C;
    USART1->BRR = 7500; // 72MHz/9600bps = 7500
}

void usart1_sendByte(unsigned char c)
{
    while((USART1->SR& (1<<6))==0); //wait until TC flag is set
    USART1->DR = c;
}

void usart1_sendStr(char *str)
{
    while(*str != 0)
    {
        usart1_sendByte(*str);
        str++;
    }
}

void usaer1_sendInt(unsigned int i)
{
    char str[10];
    sprintf(str, "%d", i);

    usart1_sendStr(str);
}
```