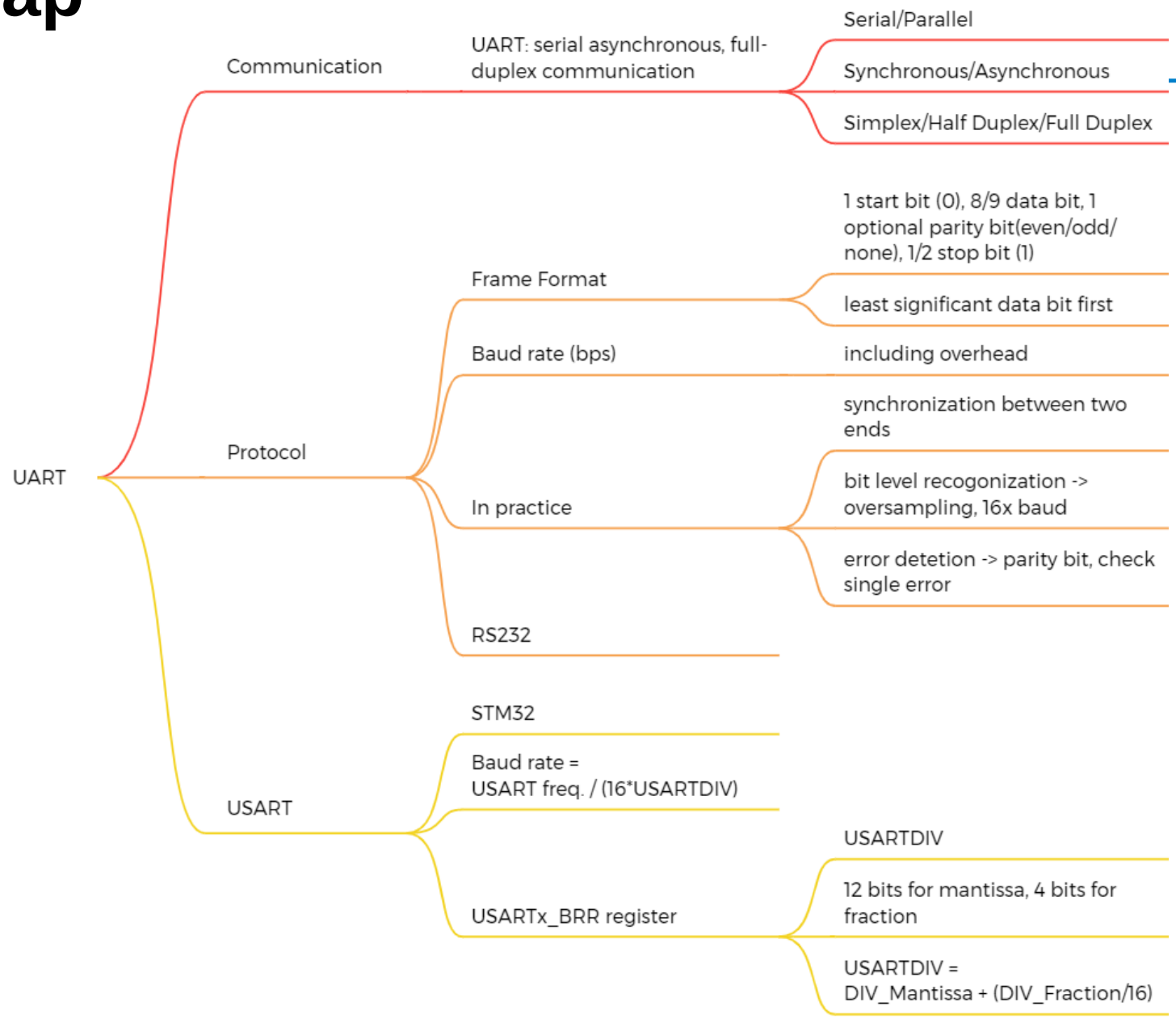
CS301
Embedded System and Microcomputer Principle

Lecture 7:Timer Part1

2023 Fall

Recap


UART
Communication
UART: serial asynchronous, full-duplex communication
Serial/Parallel
Synchronous/Asynchronous
Simplex/Half Duplex/Full Duplex
Protocol
Frame Format
1 start bit (0), 8/9 data bit, 1 optional parity bit(even/odd/none), 1/2 stop bit (1)
least significant data bit first
Baud rate (bps)
including overhead
In practice
synchronization between two ends
bit level recogonization -> oversampling, 16x baud
error detetion -> parity bit, check single error
RS232
USART
STM32
Baud rate = USART freq. / (16*USARTDIV)
USARTx_BRR register
USARTDIV
12 bits for mantissa, 4 bits for fraction
USARTDIV = DIV_Mantissa + (DIV_Fraction/16)

Outline

- **Introduction to timers**
- Clock Tree
- STM32 Timers

Time-based Control

- Many embedded systems are used to control things based on time or that have time constraints
  - Traffic light controller
  - Power meter
  - Pacemaker
  - Subway collision avoidance system
  - Airbag
  - ...
  - How to track real (wall clock) time?

Recall First Sample Code

- Toggling PA2 every second
  - This time we use pure delay loop

```
#include <stm32f10x.h>
int main()
{
    /* System clock initial */
    RCC->APB2ENR |= 0xFC; /* Enable clocks for GPIO ports */
    GPIOA->CRL = 0x44444344; /* PA2 as output */
    for (;;)
    {
        volatile unsigned int i;
        GPIOA->ODR ^= (1 << 2); /* toggle PA2 */
        i = 5000000; /* Delay */
        do
            (i--);
        while (i != 0);
    }
}
```

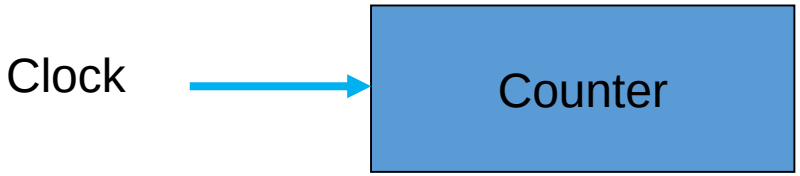
How much time?

Problems Regarding Time

- Using software delay loops
  - Waste of processor because it is not available for other operations
  - Difficult to translate into actual time
  - Given a time for the delay, difficult to translate into number of iterations
  - The delays are unpredictable, e.g., compiler optimization, interrupts
- We need an independent reference of time!

Reference of Time

- The simplest hardware to provide a reference of time is a counter that counts every fixed unit of time → timer



- The actual time can be obtained by multiplying the counter with the clock interval time
  - The accuracy and stability of the clock is critical
- How is clock generated?

Outline

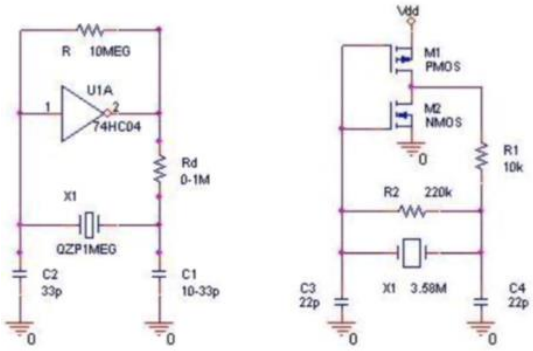
- Introduction to timers
- **Clock Tree**
- STM32 Timers

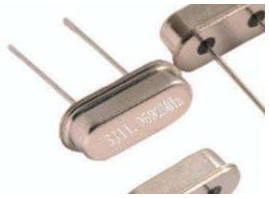
Clock

- Clock refers to the basic time interval for instruction execution, and a high clock frequency means a strong computing capability for the CPU.
- Stable clocks are essential for watchdog timers, timers, asynchronous communications, and more.
- STM32 Clocks
  - Quartz Crystal Oscillators
  - RC/LC/RLC oscillators
  - PLL

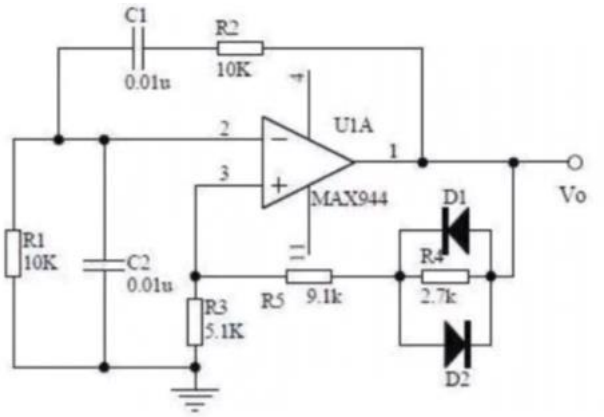
Clock Sources

- STM32 Clocks
  - Quartz Crystal Oscillators
    - High Accuracy
    - Stable frequency
  - RC/LC/RLC oscillators
    - Simplicity
    - Low Cost
  - PLL

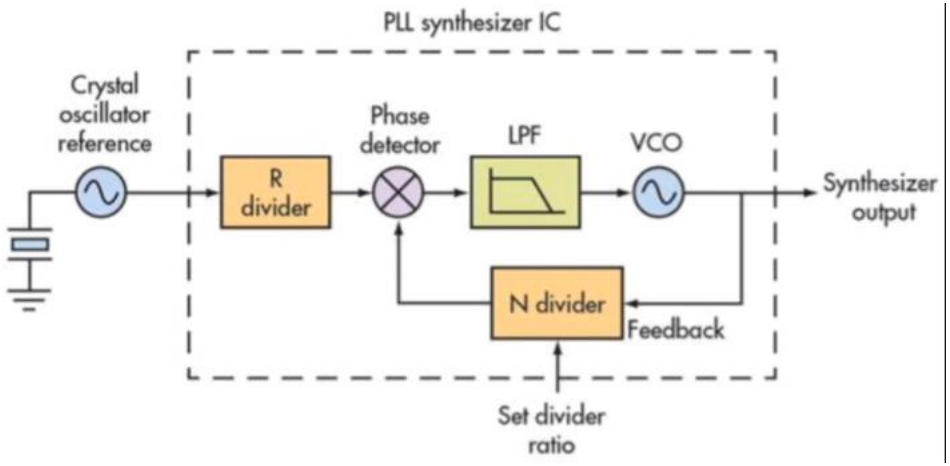




Quatz

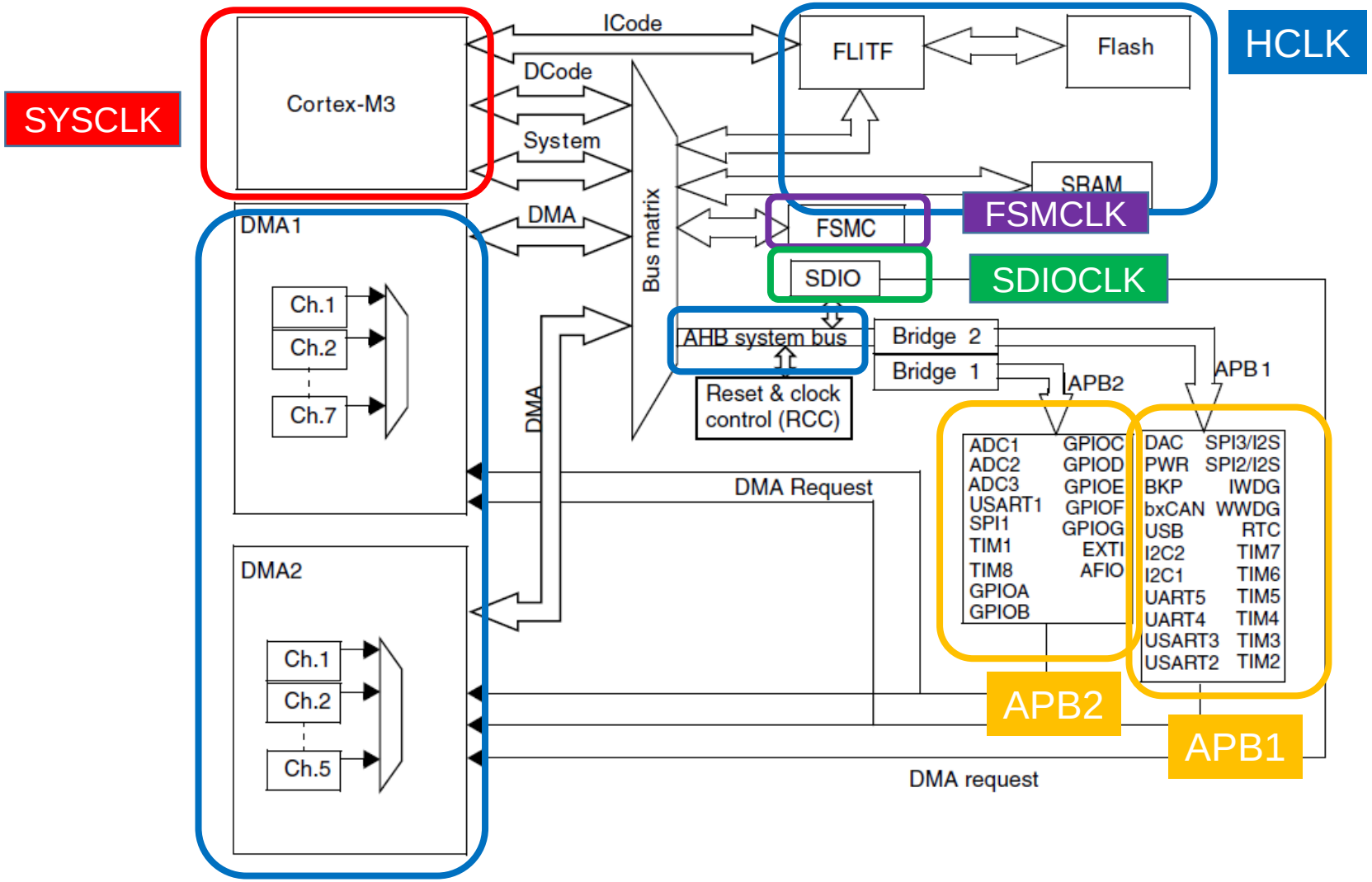


RC



PLL

STM32 Clocks



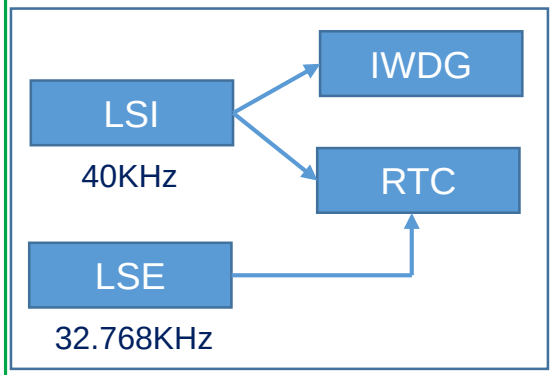
Multi-Clock Management

- Different types of buses, interfaces, external components, and power management may require different clocks.
  - Inside the MCU, there is a unified clock tree, and peripheral clocks are typically derived by dividing the system clock.
  - These clocks are usually provided by external crystals oscillators, and the clock source is selected using external configuration pins during reset.
- Phase-locked loop (PLL) technology is widely used to increase the external lower-frequency clock to a higher-frequency internal clock

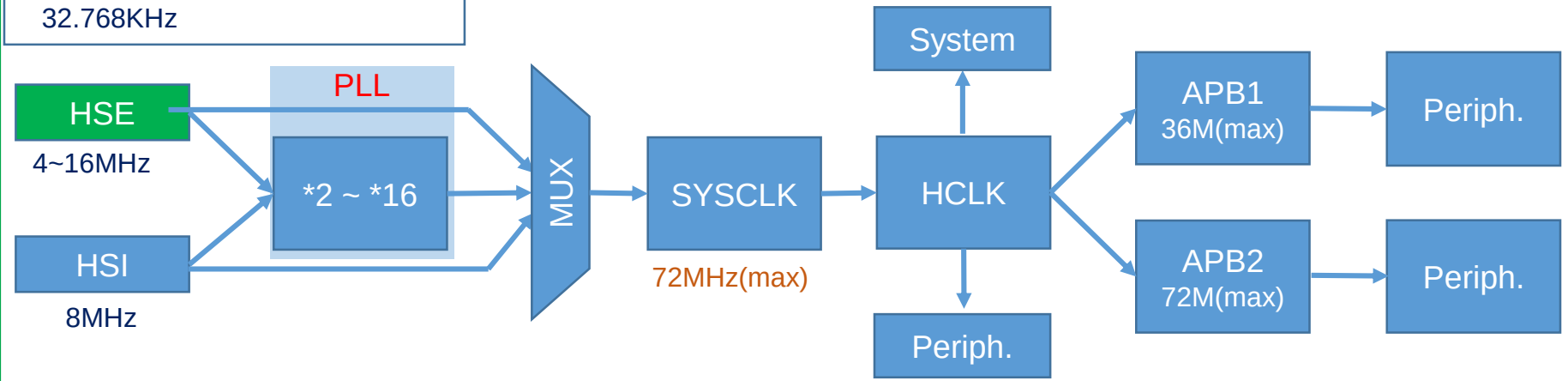
Clock Tree

- STM32F103 Clock Config
  - H：high
  - L：low
  - S：speed
  - I：internal
  - E：external

Source	Freq.	Material	Usage
HSE	4~16MHz	Oscillator	SYSCLK
LSE	32.768KHz	Oscillator	RTC
HSI	8MHz	RC	SYSCLK
LSI	40KHz	RC	RTC/IWDG



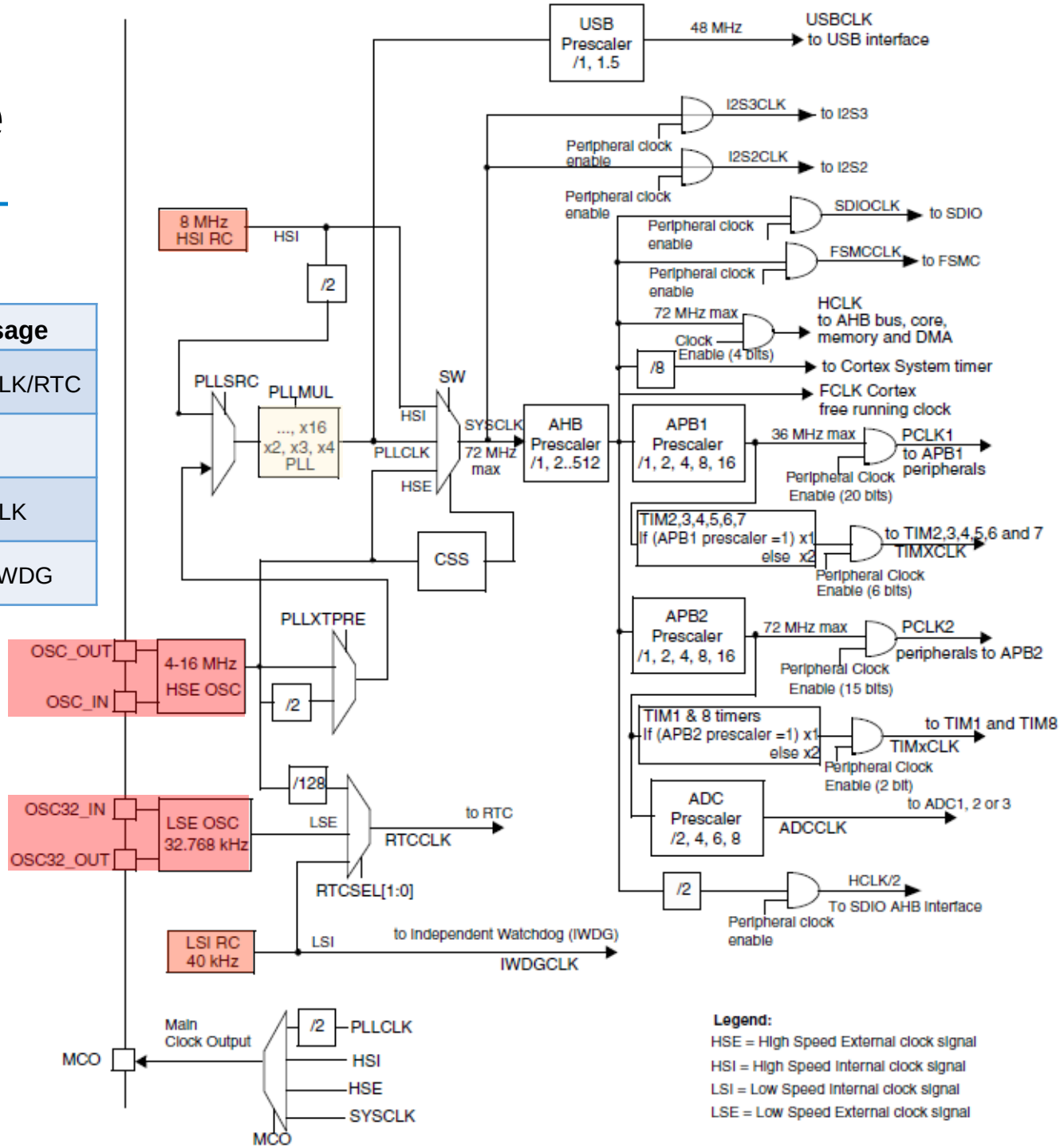
Max System Freq. 72MHz



Clock Tree

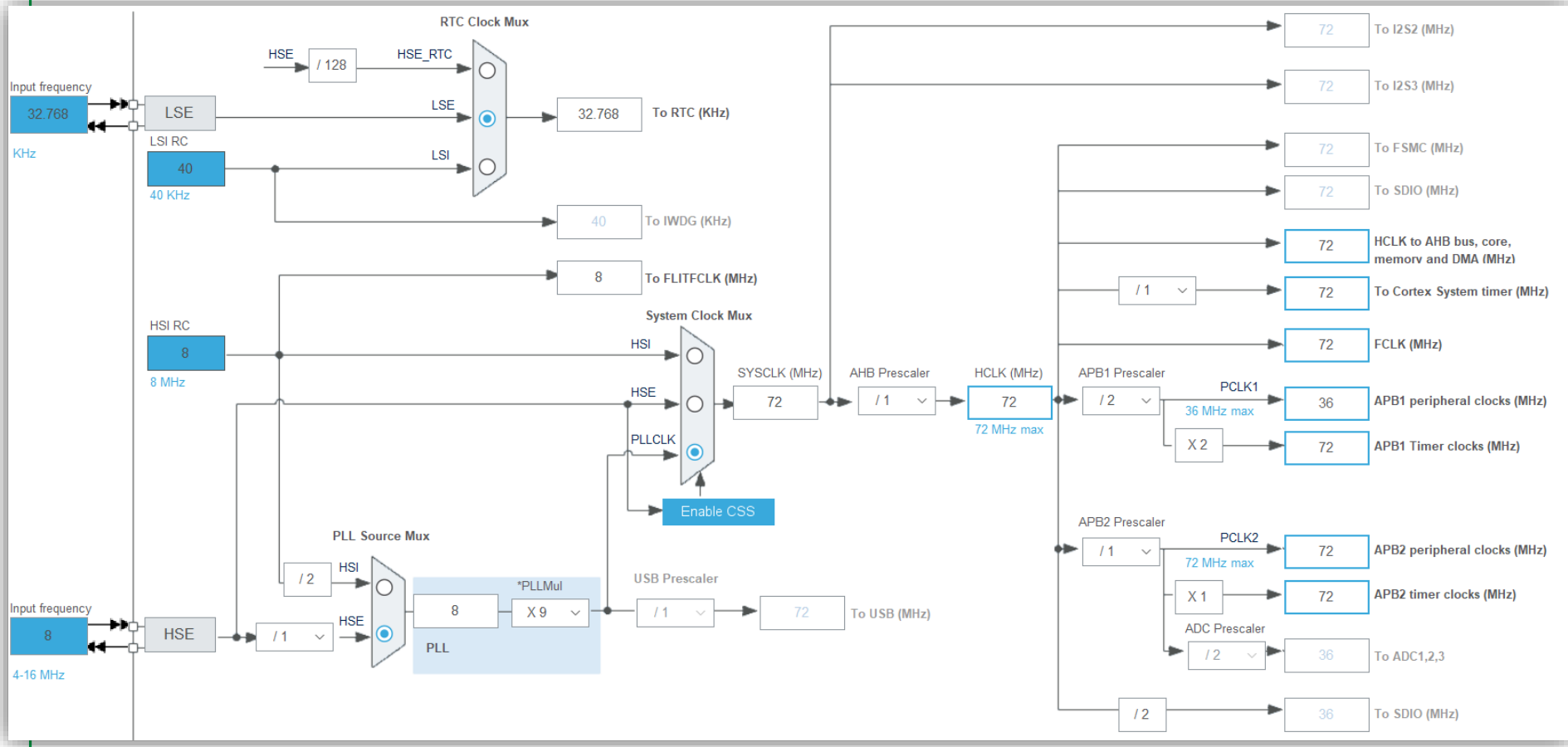
- STM32F103

Source	Freq.	Material	Usage
HSE	4~16MHz	Oscillator	SYSCLK/RTC
LSE	32.768KHz	Oscillator	RTC
HIS	8MHz	RC	SYSCLK
LSI	40KHz	RC	RTC/IWDG



Clock Tree

- STM32F103 Clock Config in CubeIDE

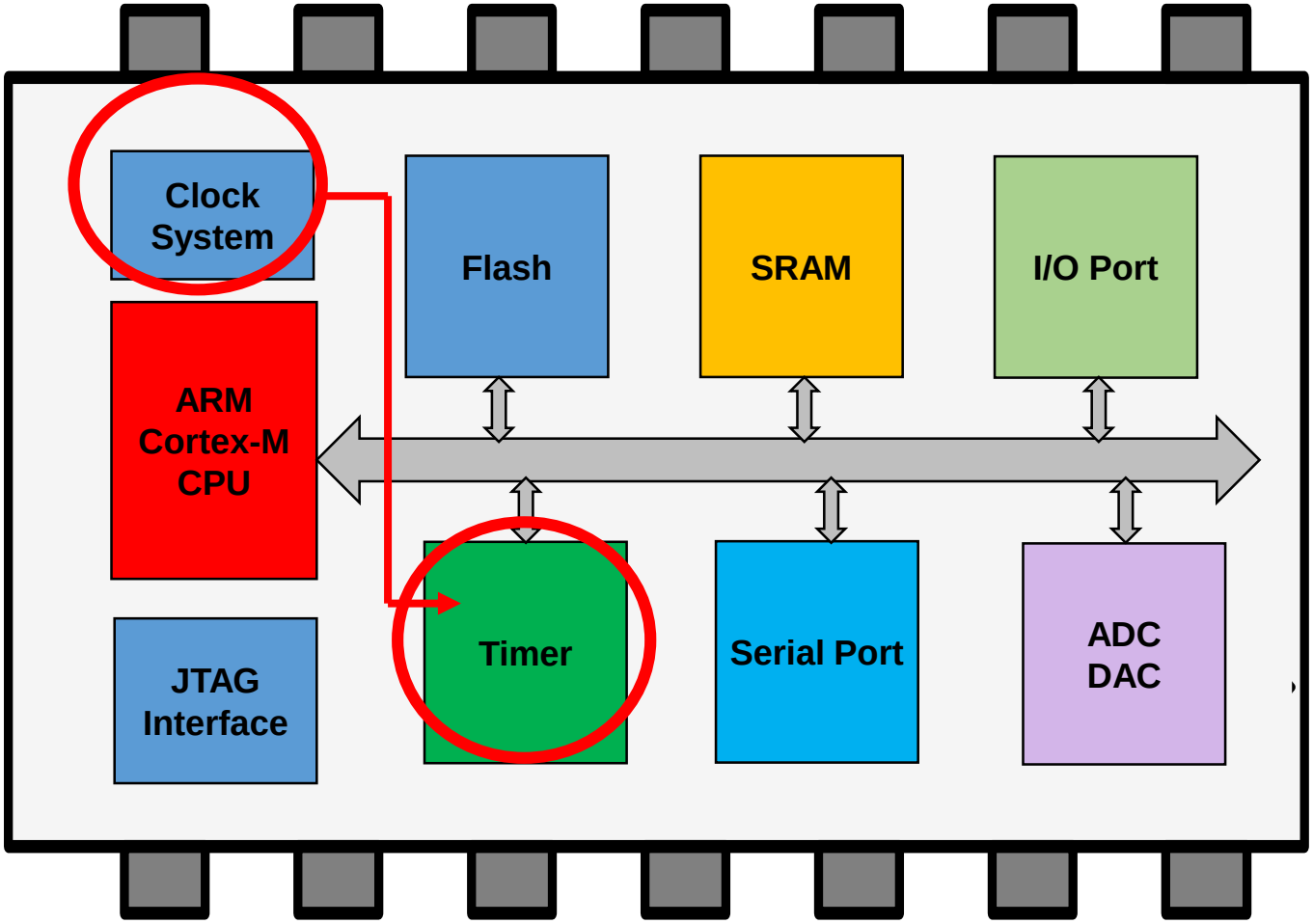


Outline

- Introduction to timers
- Clock Tree
- **STM32 Timers**

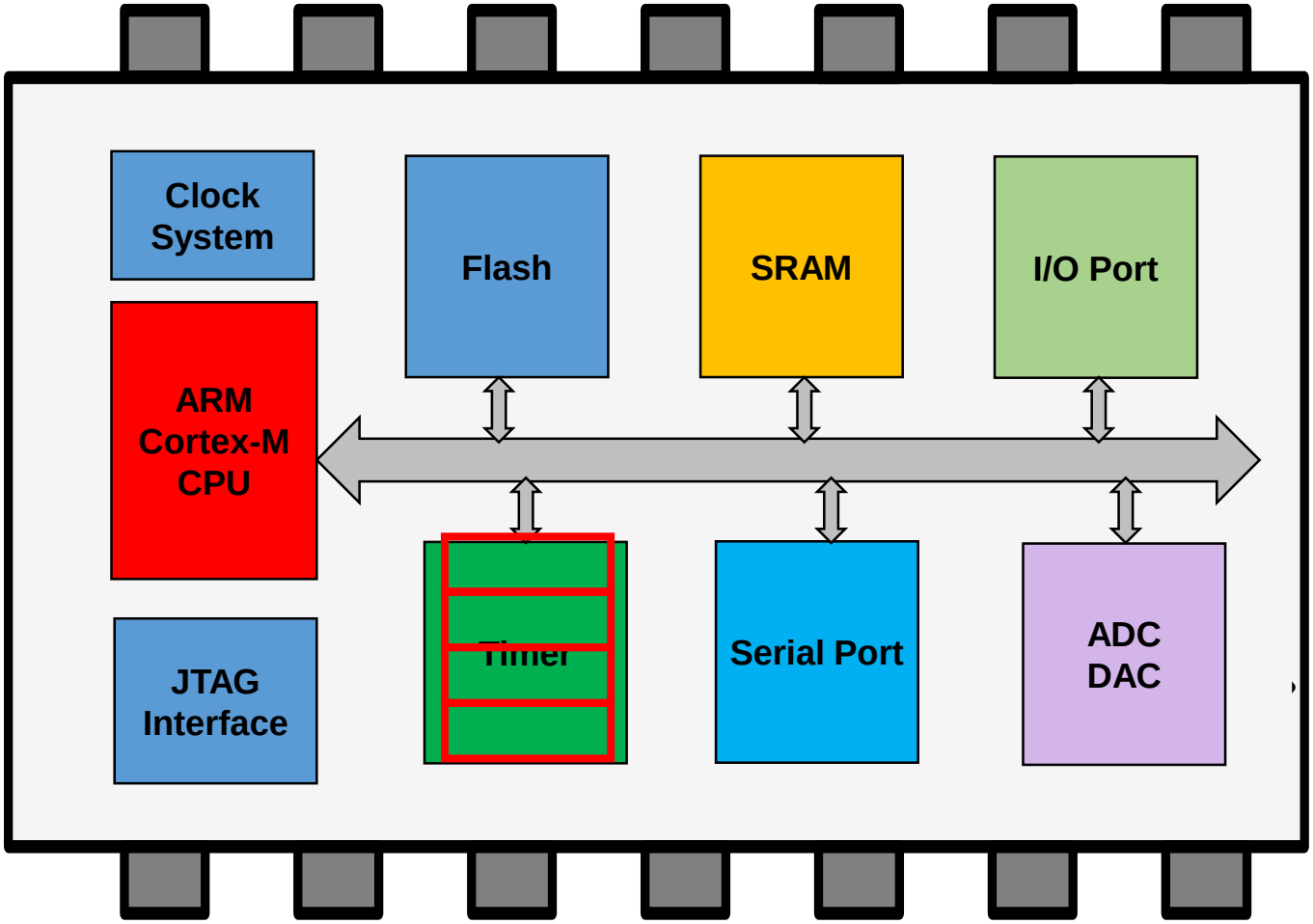
Timer in MCU

- Make Timer an IO Device based on Reference Clock



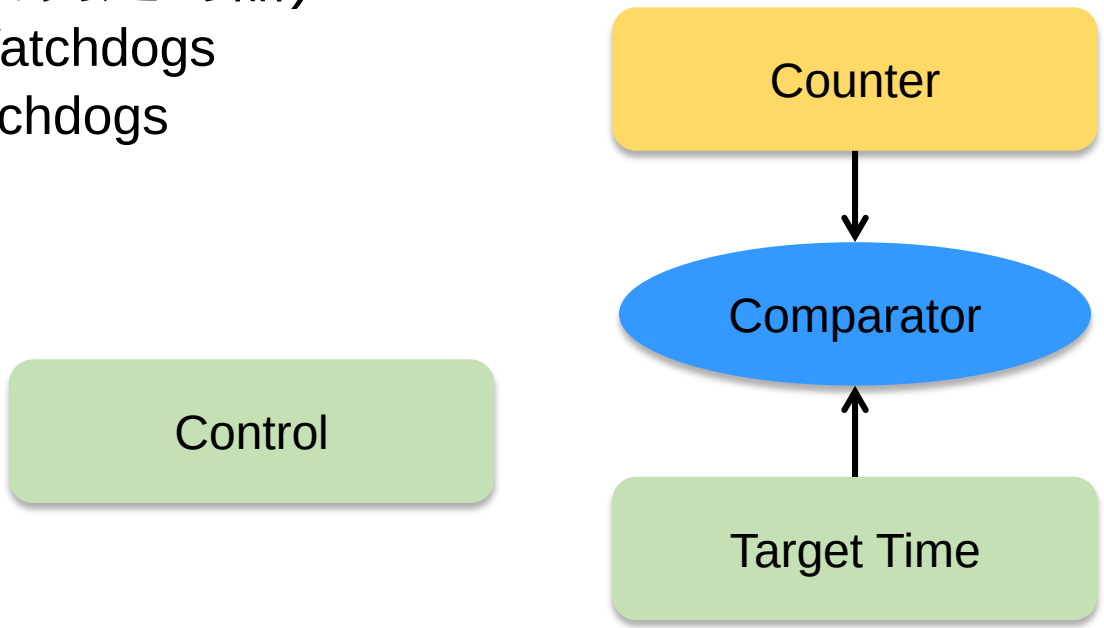
Timer in MCU

- Have internal registers with addresses in the memory space for the CPU to access



STM32 Timers

- Timers(常规定时器)
  - Basic Timers
  - General Purpose Timers
  - Advanced Timers
- SysTick(系统嘀嗒定时器)
- Watchdogs(看门狗定时器)
  - Independent Watchdogs
  - Windowed Watchdogs
- RTC(实时时钟)



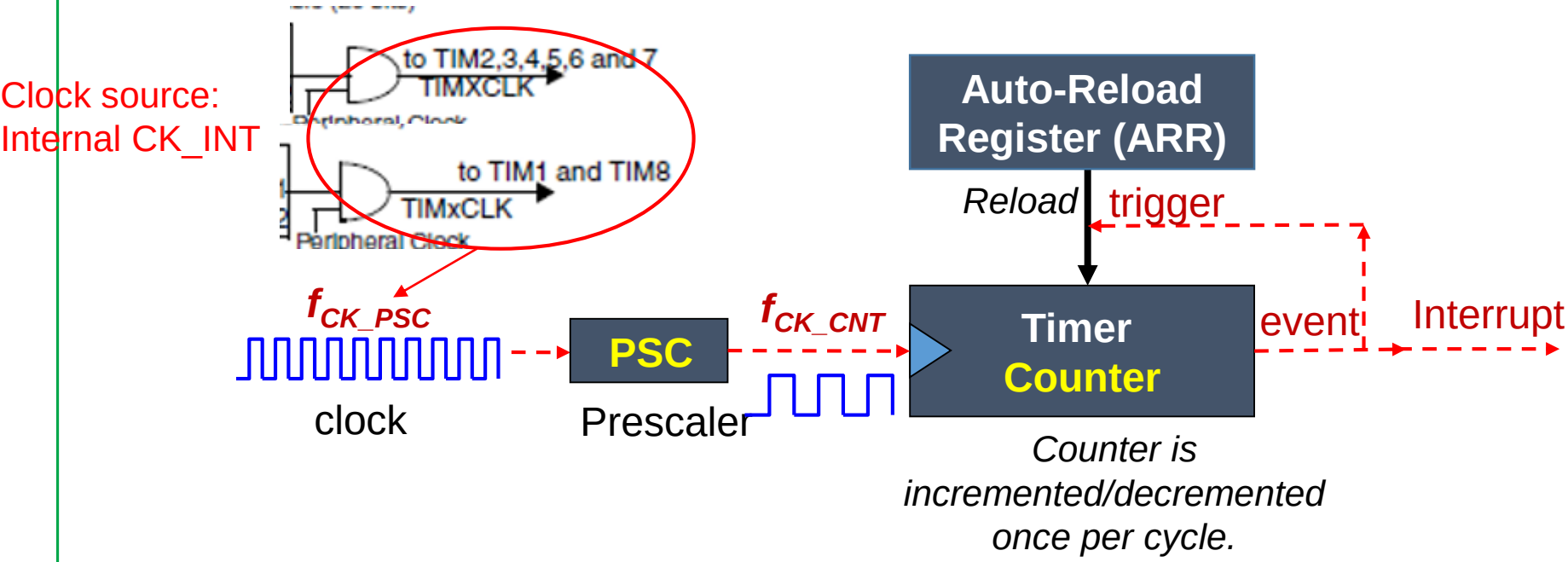
STM32 Timers

- Advanced-control timers (TIM1&TIM8)
- General-purpose timers (TIM2 to TIM5)
- Basic timers (TIM6&TIM7)
- All are 16 bits timer, but Baisc timers only support counting up

Type	Num	Bus	Counting mode	Feature
Basic	TIM6, TIM7	APB1	Up	Time-base generation (Timer)
General-purpose	TIM2 to TIM5	APB1	Up, down, up/down	Time-base generation, Input capture(measuring input pulse lengths), Output compare (PWM waveform)
Advanced	TIM1, TIM8	APB2	Up, down, up/down	Time-base generation, Input capture, Output compare, and more advanced features

Basic Block Diagram

- Basic timer block diagram
  - Prescaler
    - divide counter clock frequency by any factor between 1~65536
  - Counter
    - counts from 0 to the auto-reload value (contents of the ARR register), then restarts from 0 and generates an overflow event.
  - Auto-Reload Register



Timer Registers

- Generalized for all timers
- TIMx_CNT (Counter)

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

Bits 15:0 **CNT[15:0]**: Counter value

- TIMx_ARR (Auto-Reload Register)

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

Bits 15:0 **ARR[15:0]**: Auto-reload value

$$f_{CK\_CNT} = \frac{f_{CK\_PSC}}{TIMx\_PSC + 1}$$

- TIMx_PSC (Prescaler value)

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

Timer Registers

- TIMx_CR1 (Control Register)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

0	0	0	0	0	0	0	0	0	1	up
0	0	0	0	0	1	0	0	0	1	down
0	0	0	0	0	0	0	0	0	0	stop

  - CEN (Counter Enable)
    - 0: counter disabled
    - 1: count
  - OPM (One Pulse Mode)
    - 0: the counter counts continuously
    - 1: the counter stops at the next update event.
  - CMS (Center-aligned Mode Selection)
  - DIR (Direction)

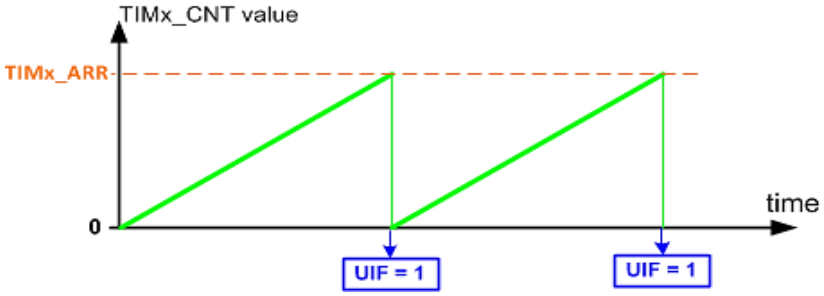
CMS	DIR	Counting mode
00	0	Counting up
00	1	Counting down
01	X	Count up and down
10	X	Count up and down
11	X	Count up and down

Timer Registers

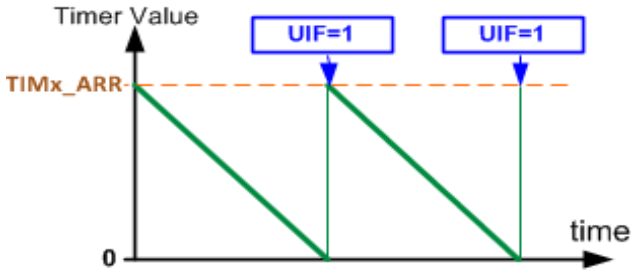
- TIMx_SR (Status Register)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

  - The bit 0 of TIMx_SR is used as UIF (Update Interrupt Flag)
    - Set by hardware while update_event occurs, cleared by sorftware
  - In counting up mode, when the value of CNT (counter) reaches ARR, UIF flag is set.
    - The flag remains high until it is cleared by software
  - In the counting down mode, the flag is set when the counter reaches zero.



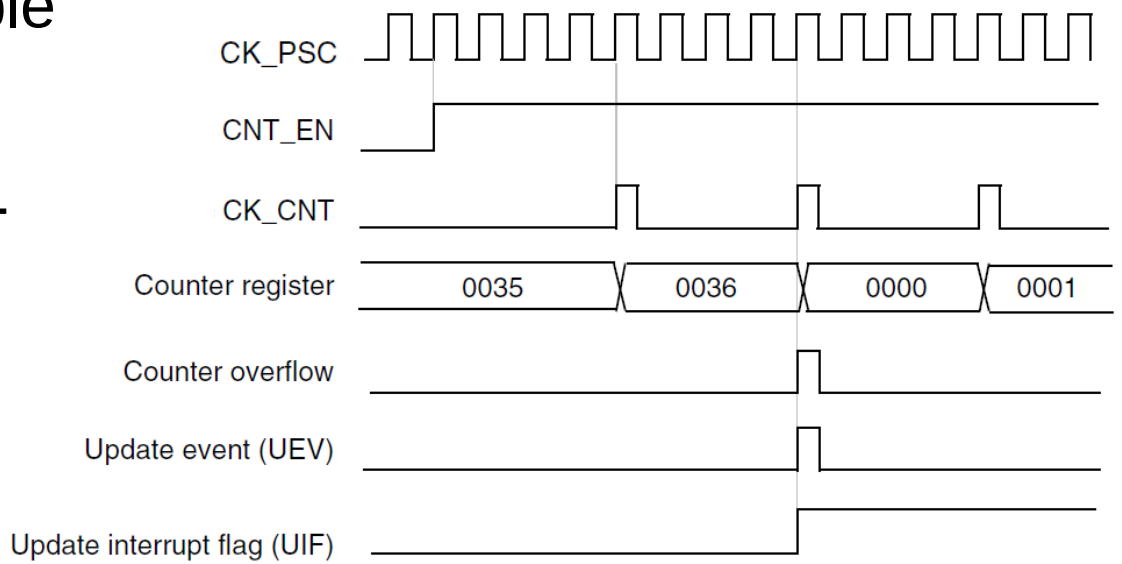
Counting up UIF

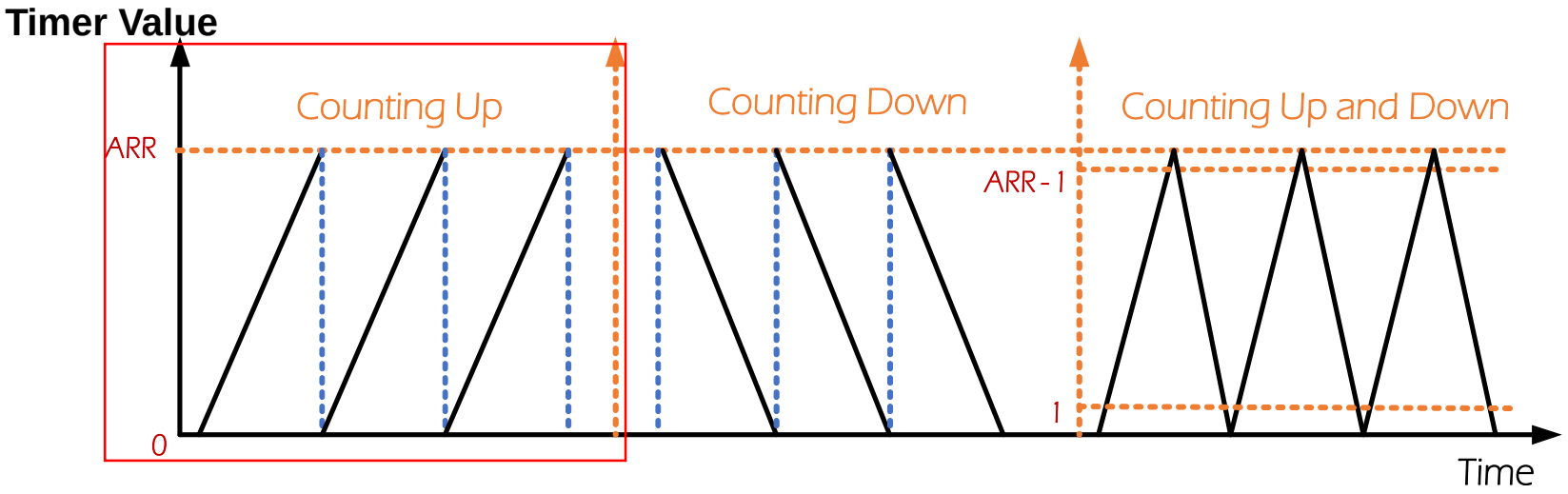


Counting down UIF

Counting up

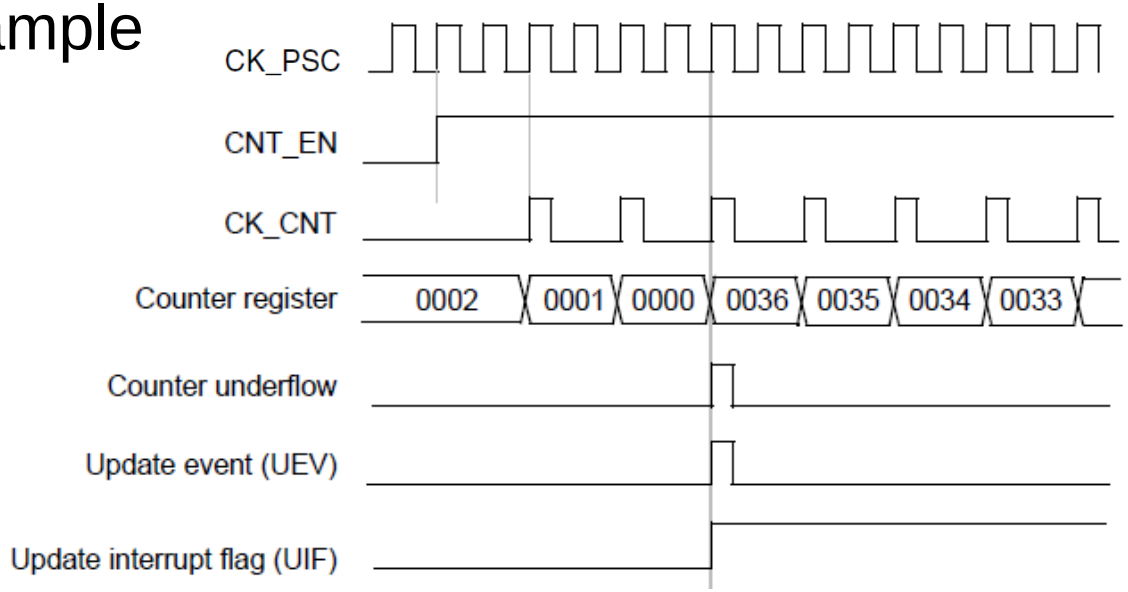
- Counting Up Example
  - PSC = 3
  - ARR = 36
- What's the CK_CNT frequency and what period will the timer generate?

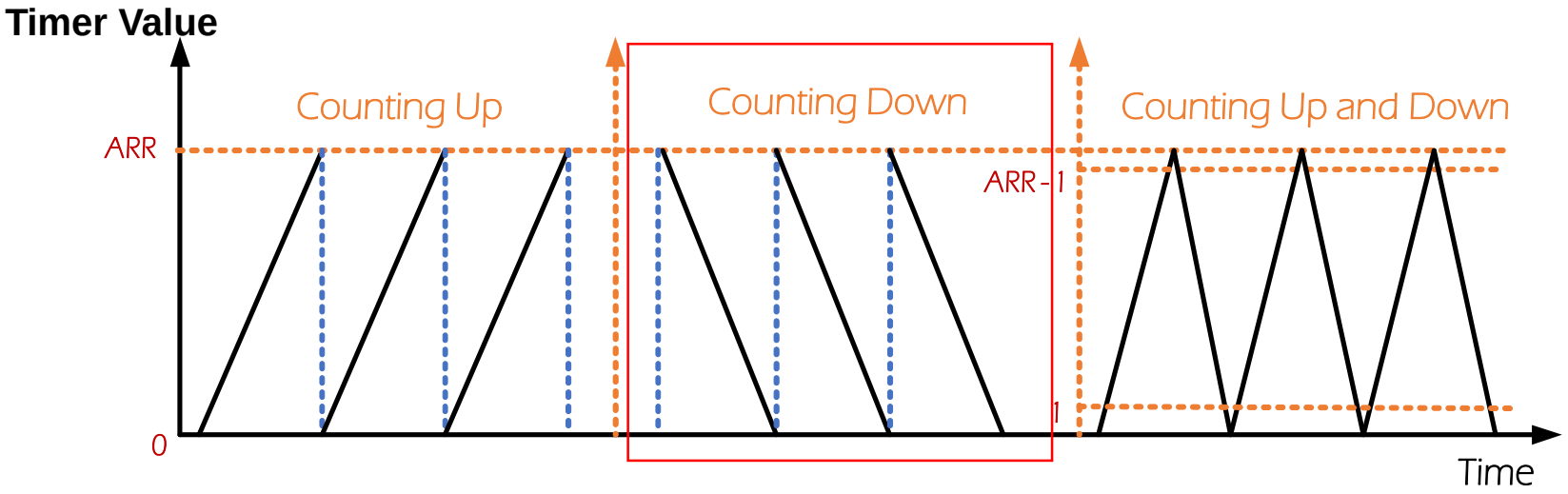




Counting down

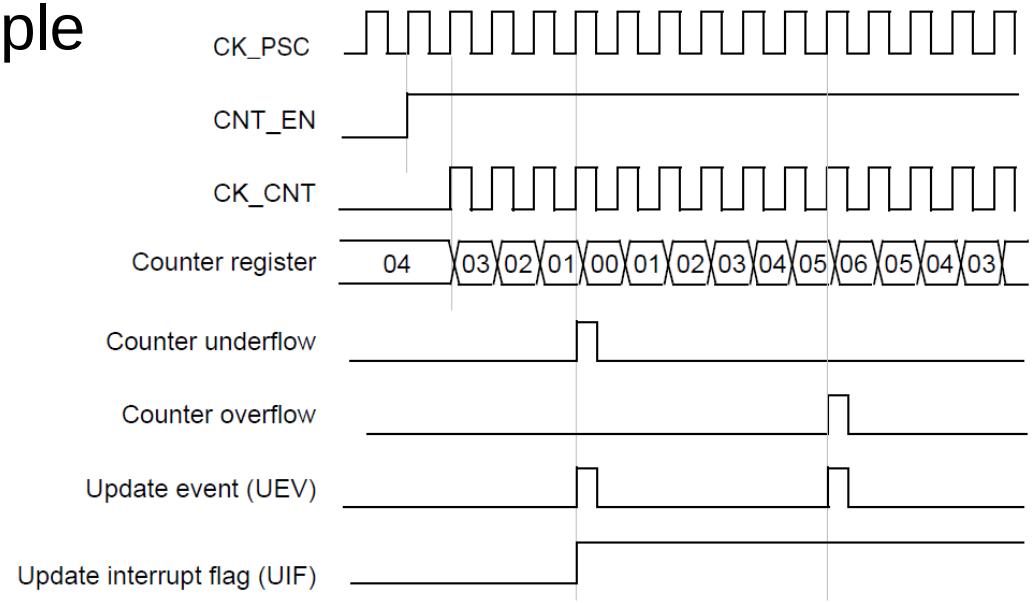
- Counting Down Example
  - PSC = 1
  - ARR = 36

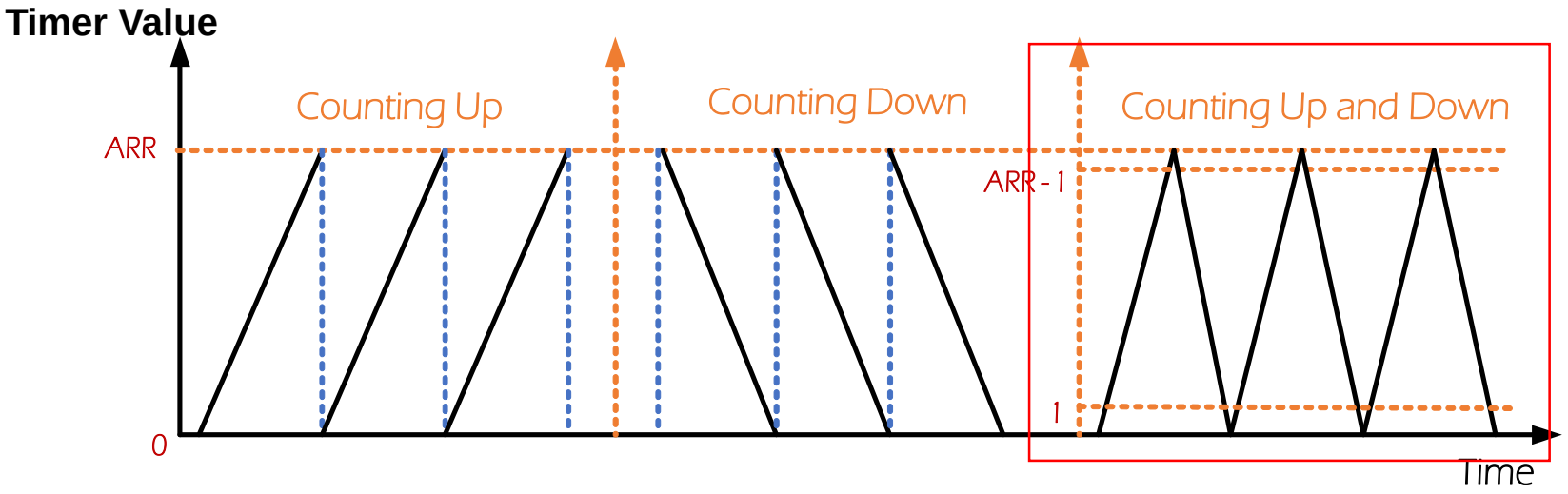




Center-aligned mode

- Center-aligned Example
  - PSC = 0
  - ARR = 6





How to make delays using TIMx

- Steps Making delays using the 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 the mode as up-counter timer and enable timer,
  - 5) wait for UIF flag to go HIGH,
  - 6) Stop the timer.
- For up counter, how to calculate the delay?

$$T = \frac{(\text{TIMx\_ARR} + 1) * (\text{TIMx\_PSC} + 1)}{f_{CK\_PSC}}$$

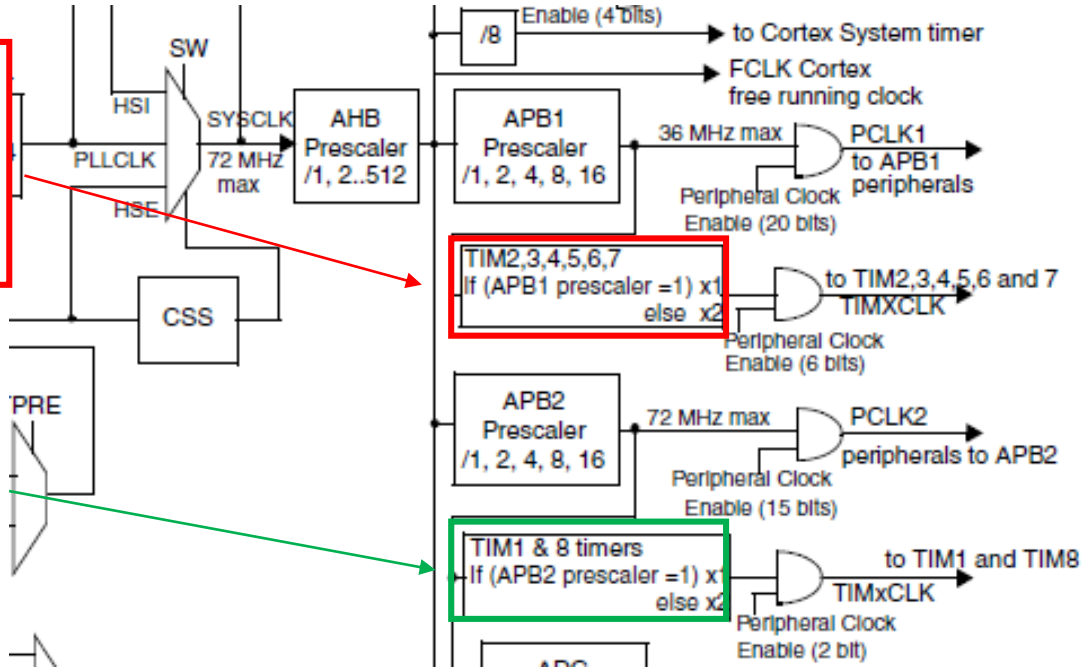
Example1

- By default, the AHB clock frequency is 72MHz; the APB1 clock is set to 36MHz, and APB2 is 72MHz. Calculate the frequency of the clock that is fed to the timer 7 and timer 8.

Solution:

F_{AHB}= 72MHz, f_{APB1} = 36 MHz→
APB1 prescaler = 2
TIM7 is connected to APB1 bus
f_{TIM7} = 36MHz x 2

F_{AHB} = 72MHz, f_{APB2} = 72 MHz→
APB2 prescaler = 1
TIM8 is connected to APB2 bus
f_{TIM8} = 72MHz x 1



Example2

$$T = \frac{(TIMx\_ARR + 1) * (TIMx\_PSC + 1)}{f_{CK\_PSC}}$$

- Example: Calculate the delay which is made by the following function. Supposing TIM2 CK_PSC frequency is 72MHz

```
void delay()
{
    RCC->APB1ENR |= (1 << 0); /* enable TIM2 clock */
    TIM2->PSC = 7200 - 1;     /* PSC = 7199 */
    TIM2->ARR = 500 - 1;
    TIM2->SR = 0;  /* clear the UIF flag */
    TIM2->CR1 = 1; /* enable TIM2 with up counting */
    while ((TIM2->SR & 1) == 0)
        ;          /* wait until the UIF flag is set */
    TIM2->CR1 = 0; /*stop counting */
}
```

- Solution
  - PSC = 7199, ARR = 499,
  - Delay = (7199+1)×(499+1)/72MHz = 0.05s = 50ms

Example3

$$T = \frac{(TIMx\_ARR + 1) * (TIMx\_PSC + 1)}{f_{CK\_PSC}}$$

- Example: Using TIM2 to write a program that toggles PC13 every second. Calculate value of TIM2->ARR to complete the program. Supposing TIM2 CK_PSC frequency is 72MHz

```
int main() {
    RCC->APB2ENR |= 0xFC;      /* enable GPIO clocks */
    RCC->APB1ENR |= (1 << 0); /* enable TIM2 clock */
    GPIOC->CRH = 0x44344444;  /* PC13 as output */
    while (1) {
        GPIOC->ODR ^= (1 << 13); /* toggle PC13 */
        delay();
    }
}
void delay() {
    TIM2->PSC = 7200 - 1; /* PSC = 7199 */
    TIM2->ARR = ________?________;
    TIM2->SR = 0;  /* clear the UIF flag */
    TIM2->CR1 = 1; /* enable TIM2 with up counting */
    while ((TIM2->SR & 1) == 0); /* wait until the UIF flag is set */
    TIM2->CR1 = 0; /*stop counting */
}
```

Solution

PSC = 7199, Delay = 1s

ARR = 1s×72MHz/(7199+1) – 1

= 10000 - 1