ELEC 3300 – Tutorial for LAB4

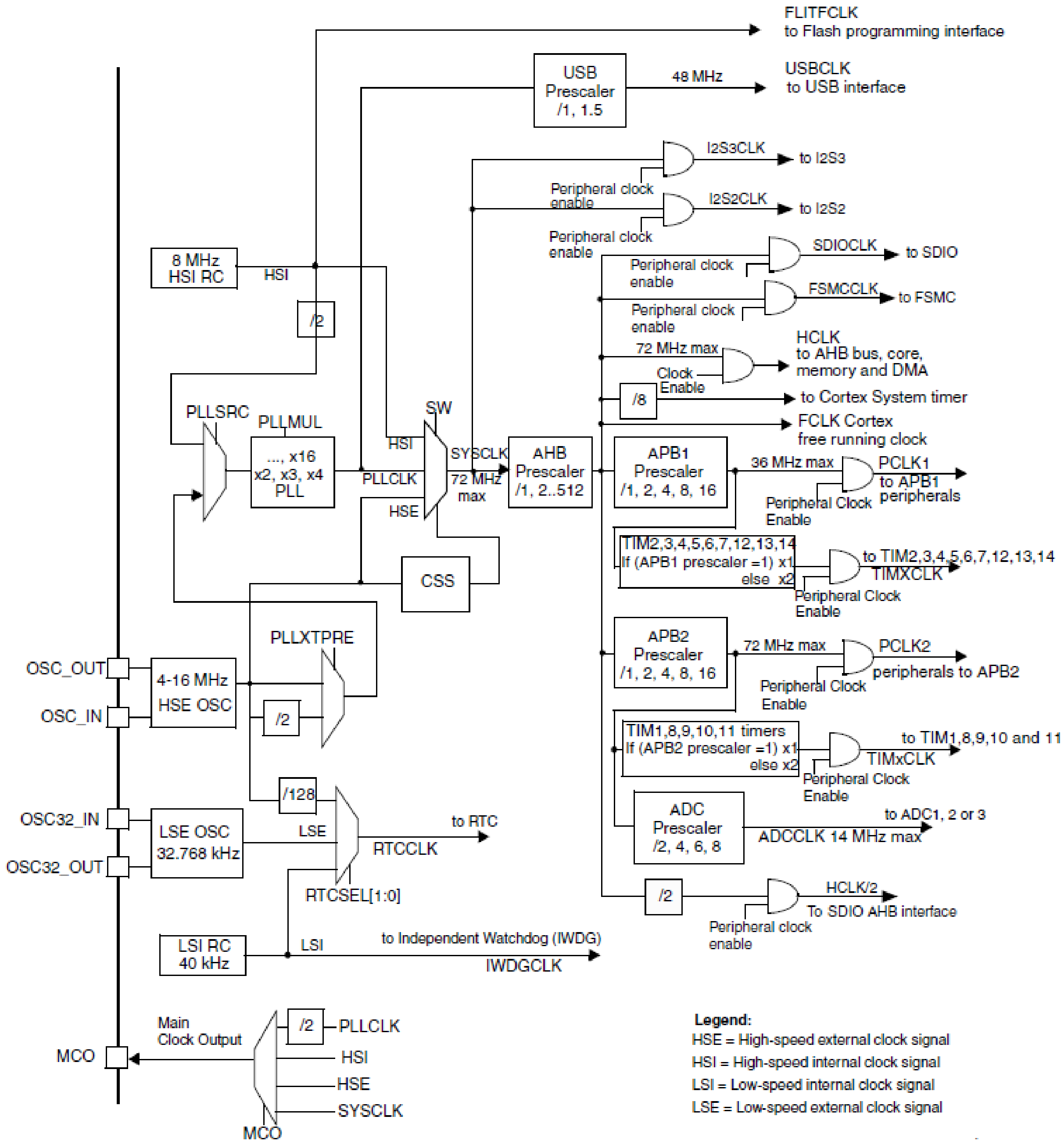
Department of Electronic and Computer Engineering
HKUST
by WU Chi Hang 

Clock in STM32

- In LAB2, you already understand that there is a clock that governs the speed of the STM32.
- The running clock of the STM32 is called the System Clock (SYSCLK).
- The SYSCLK is the global clock that will be further distributed to the AHB and APB to be the clock of rest of the STM32.
- Recall the clock tree diagram.

AHB APB

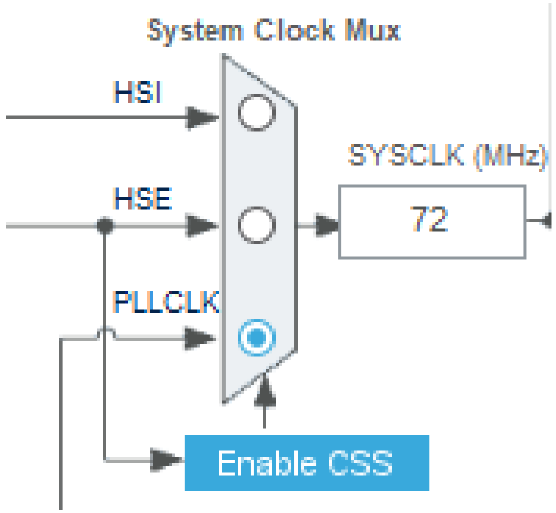
- **SYSCLK is the System Clock Frequency (max 72 MHz)**
- **AHB is the System Bus**
- **APB is Peripherals Bus**
- The two AHB/APB bridges provide full synchronous connections between the AHB and the 2 APB buses.
- APB1 is limited to 36 MHz
- APB2 can operates at full speed (i.e. max 72 MHz)

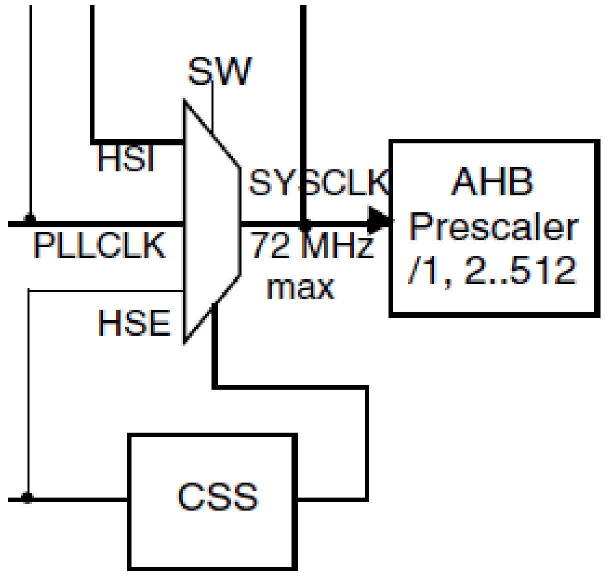


Clock in STM32

- Actually, the SYSCLK clock is originated from the following 3 sources
  - HSI = High Speed Internal clock signal.
  - HSE = High Speed External clock signal.
  - PLLCLK = Phase Locked Loop CLK signal.

- You can see the SYSCLK is 72MHz max.



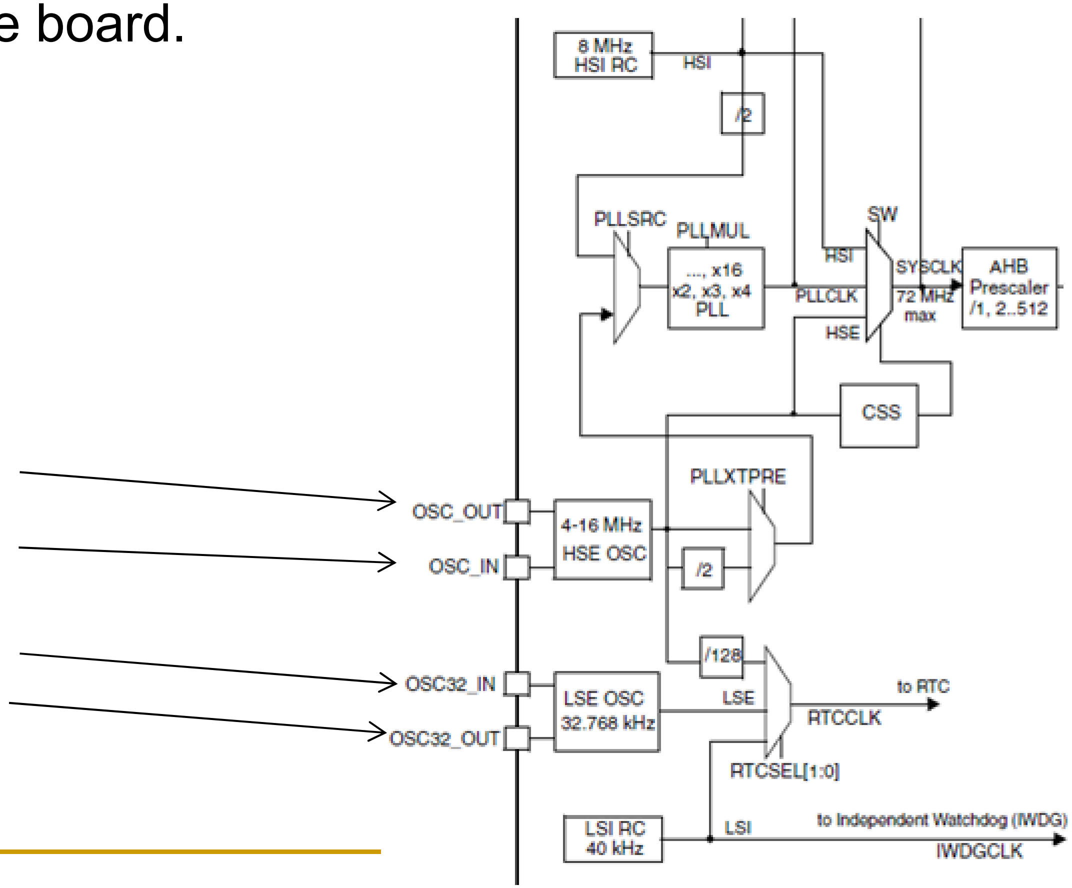


Clock in STM32

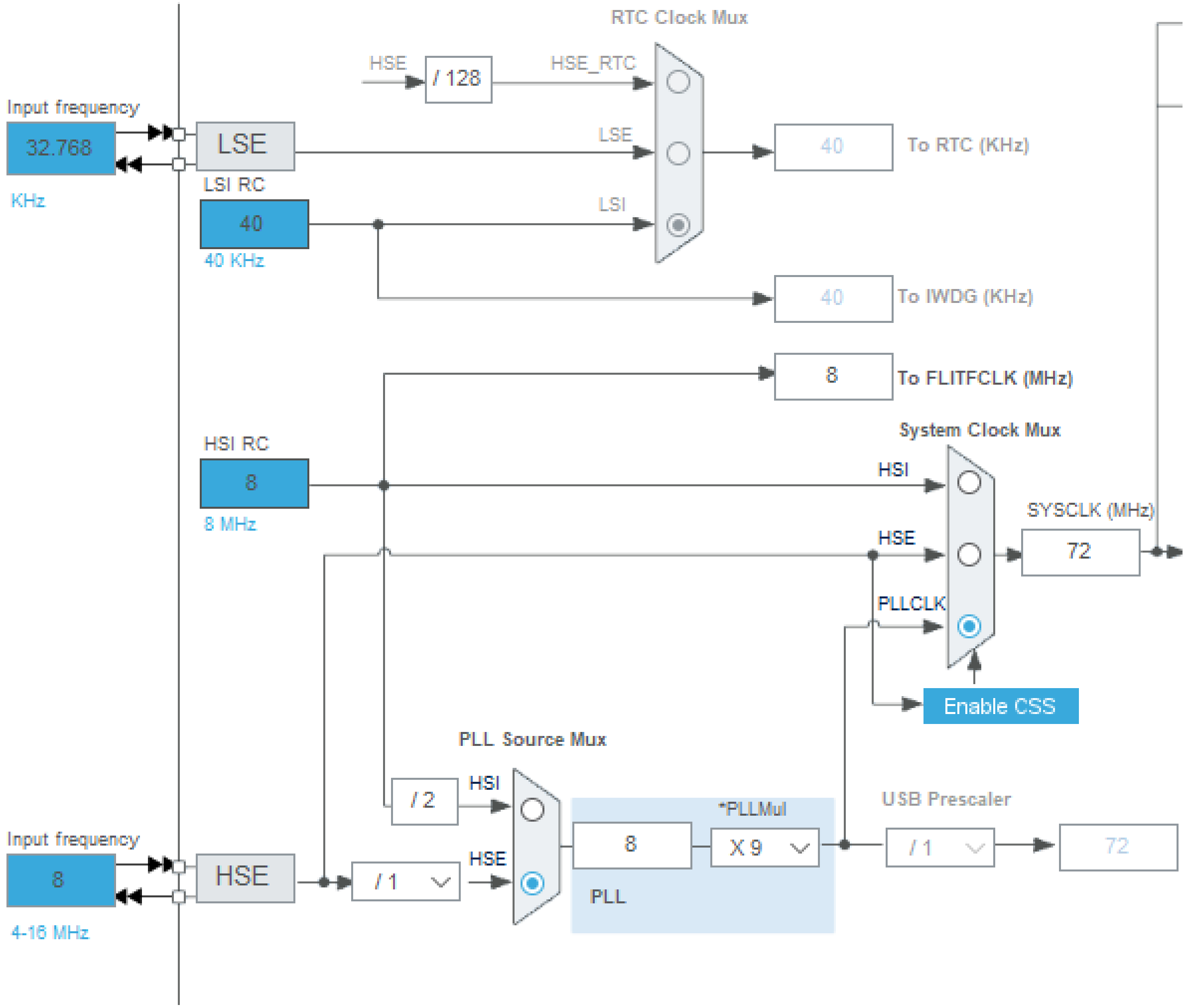
- In the MINI-V3 Development board, PLLCLK is selected the as the input to the SYSCLK because it is programmable, and it is originated from the 8MHz crystal clock on the board.

8MHz Crystal

32768Hz Crystal

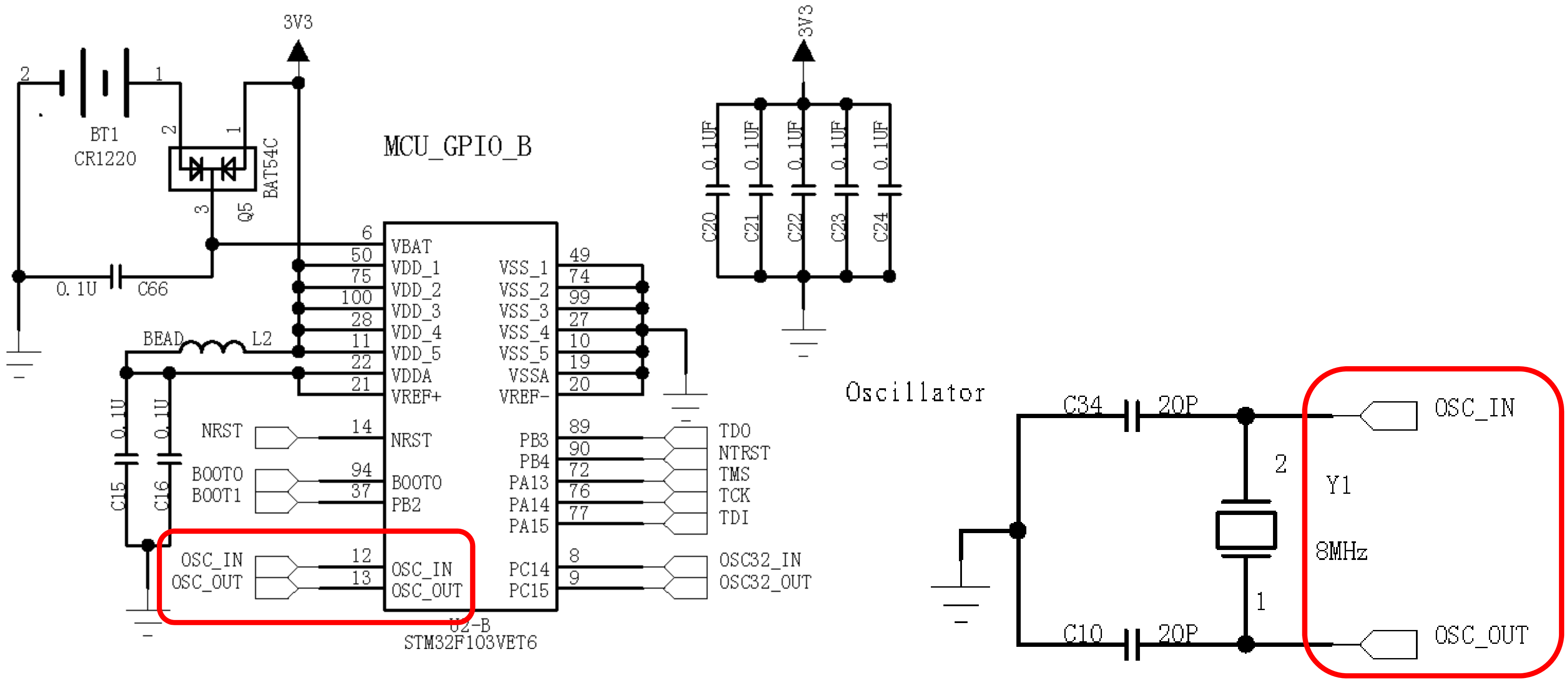


Clock in STM32



Clock in STM32

- As shown in the schematic



MCO (Master Clock Output)

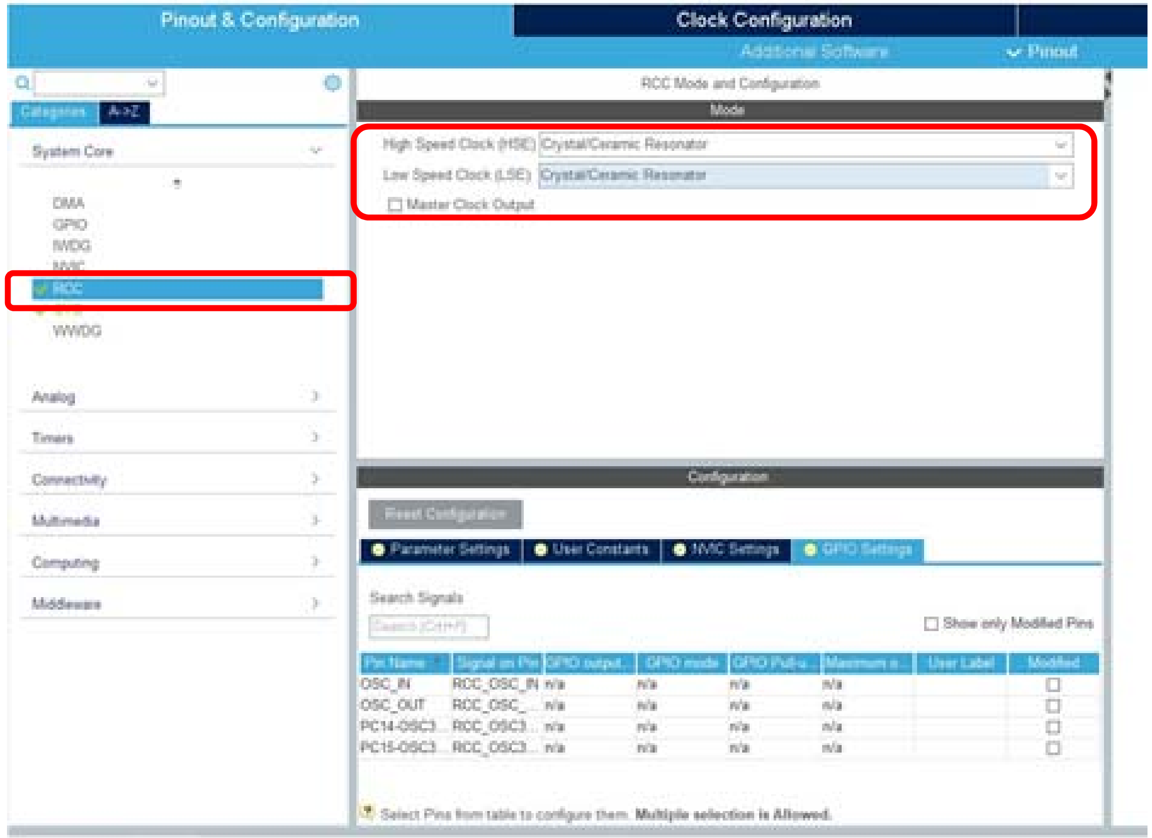
- In STM32, there is a pin called Master Clock Output (MCO) that allows you to output the clock to view it in the oscilloscope.

- The MCO pin is mapped to PA.8 of the STM32.

E11	E9	D1	40	66	99	PC9	I/O	FT	PC9	TIM8_CH4/SDIO_D1	TIM3_CH4
E12	D9	E4	41	67	100	PA8	I/O	FT	PA8	USART1_CK/ TIM1_CH1(8)/MCO	
D12	C9	D2	42	68	101	PA9	I/O	FT	PA9	USART1_TX(8)/	

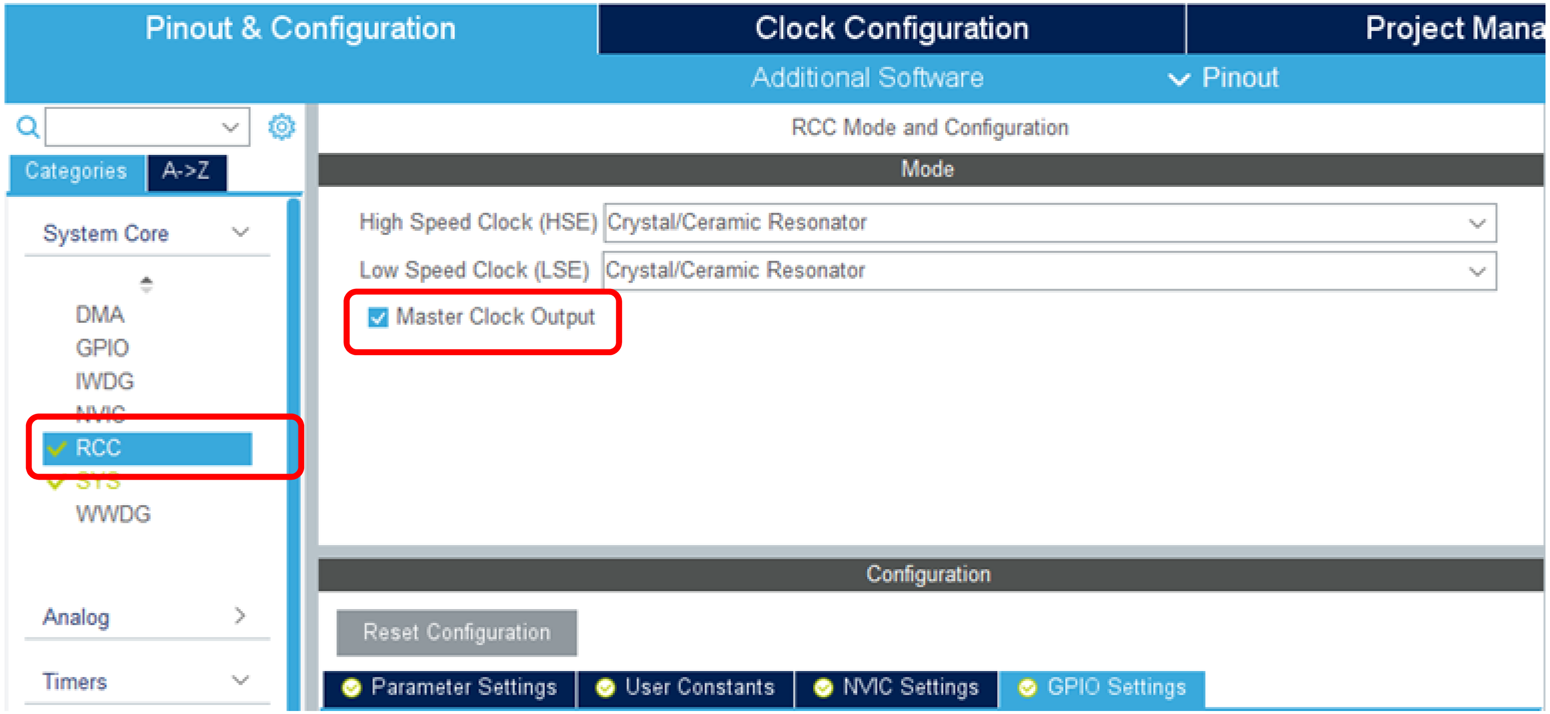
Change Clock to Crystal

- Click RCC, enable the High Speed Clock and Low Speed Clock to
  - Crystal/Creamic Resonator



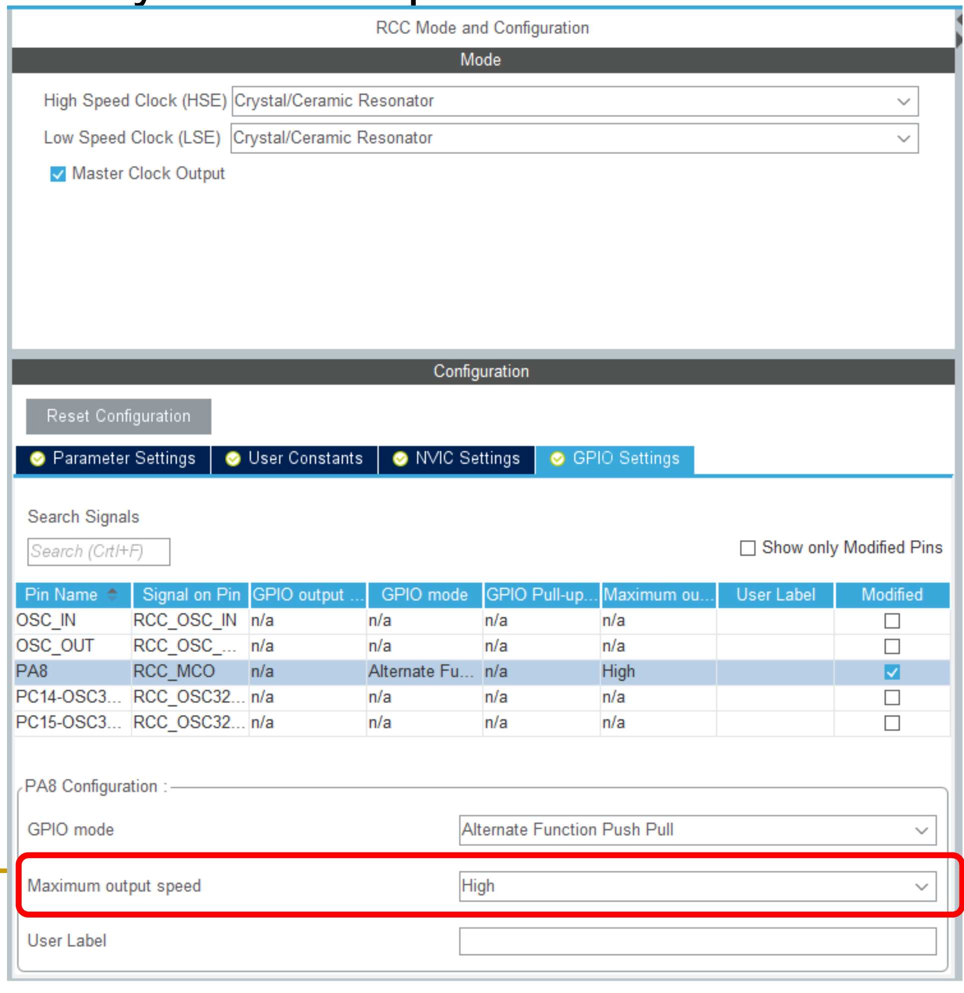
MCO (Master Clock Output)

- In order to enable the clock, you need to enable the function in CubeIDE
- On RCC Page, when you enable the Clock



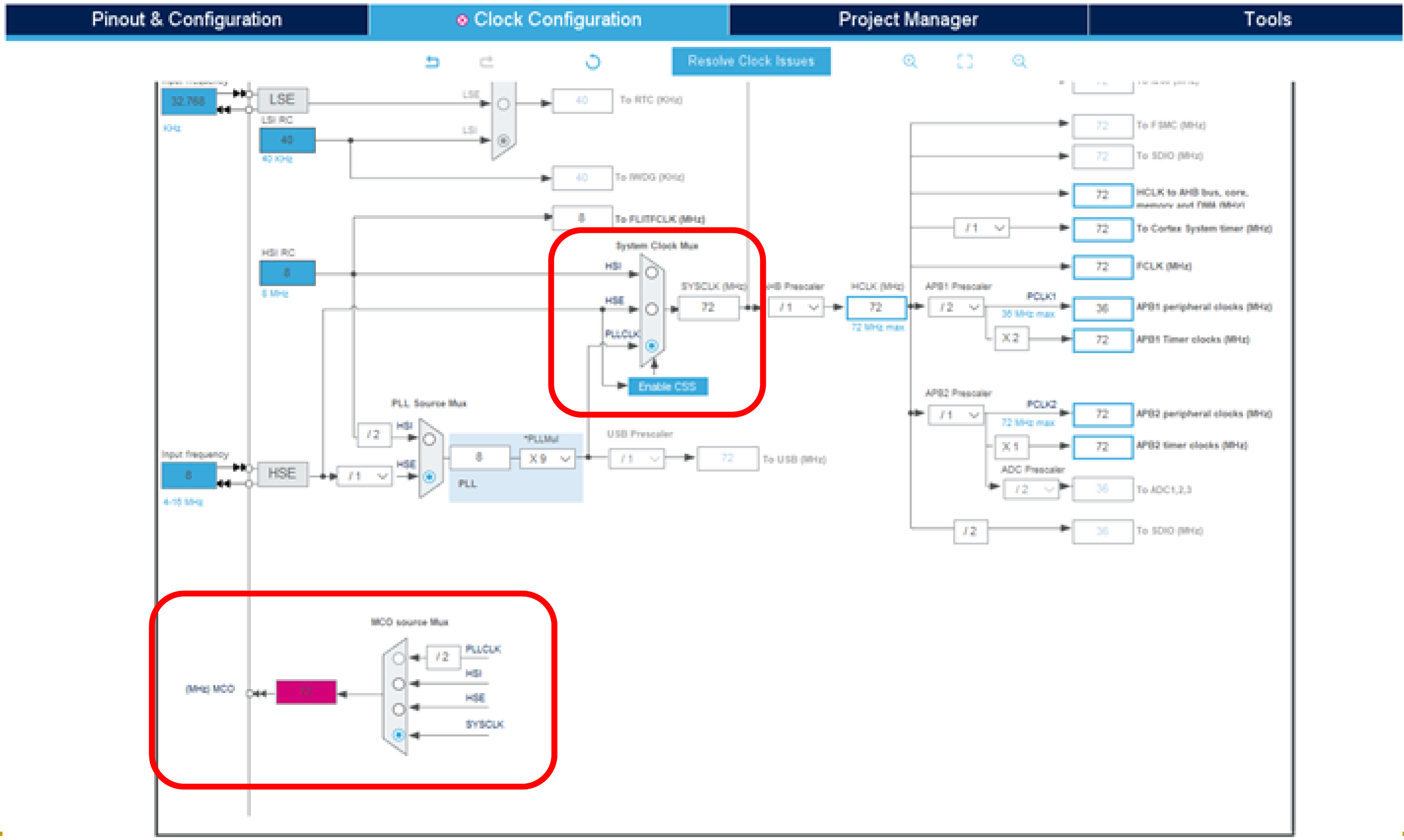
MCO (Master Clock Output)

- Once you enabled it, you will see the actual pin is PA.8, modify the speed to High, so that you can output a faster clock



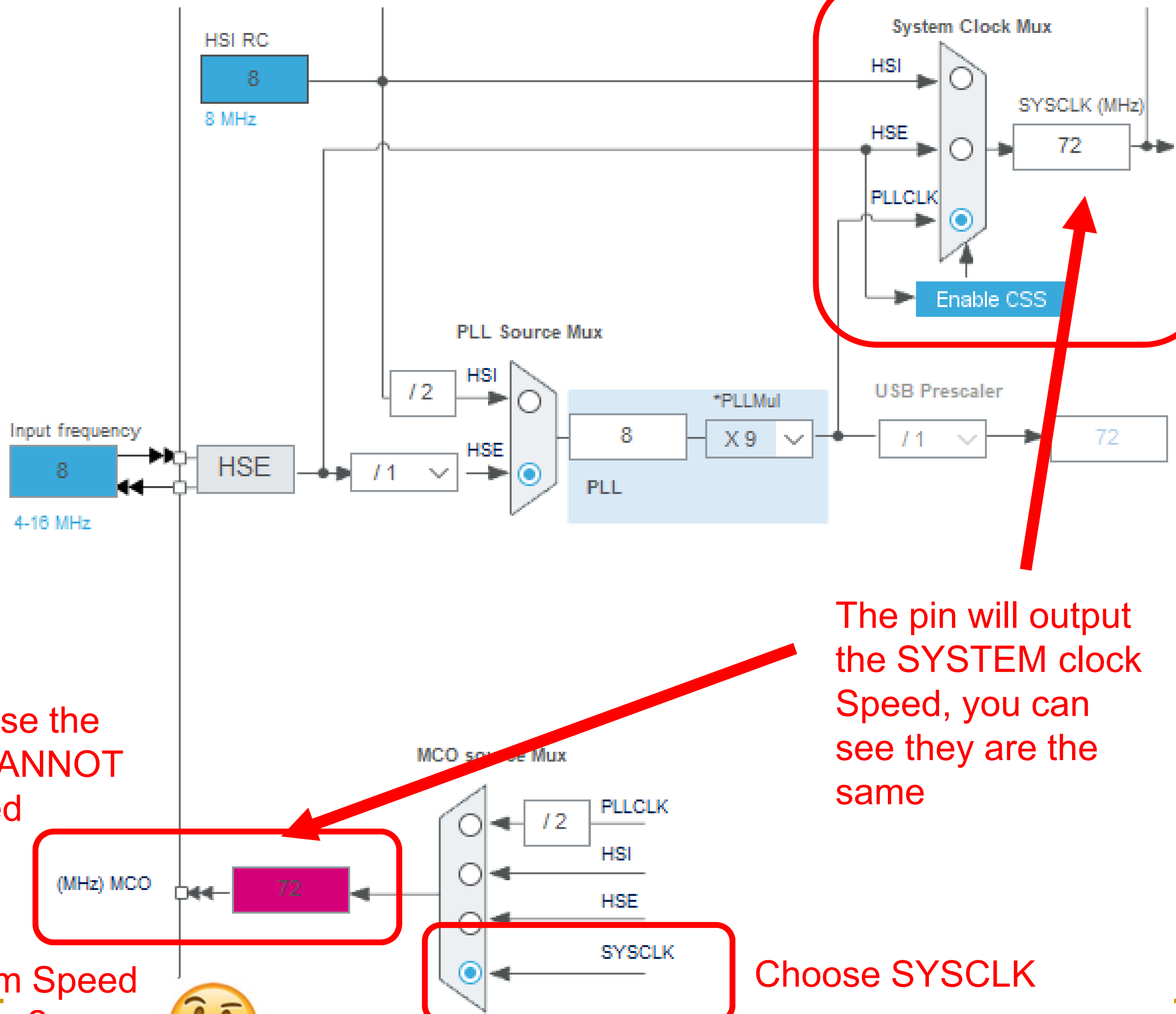
MCO (Master Clock Output)

- On Clock Configuration Page, you will see the bottom part is enabled



MCO (Master Clock Output)

- Close up to the required part



The pin will output the SYSTEM clock Speed, you can see they are the same

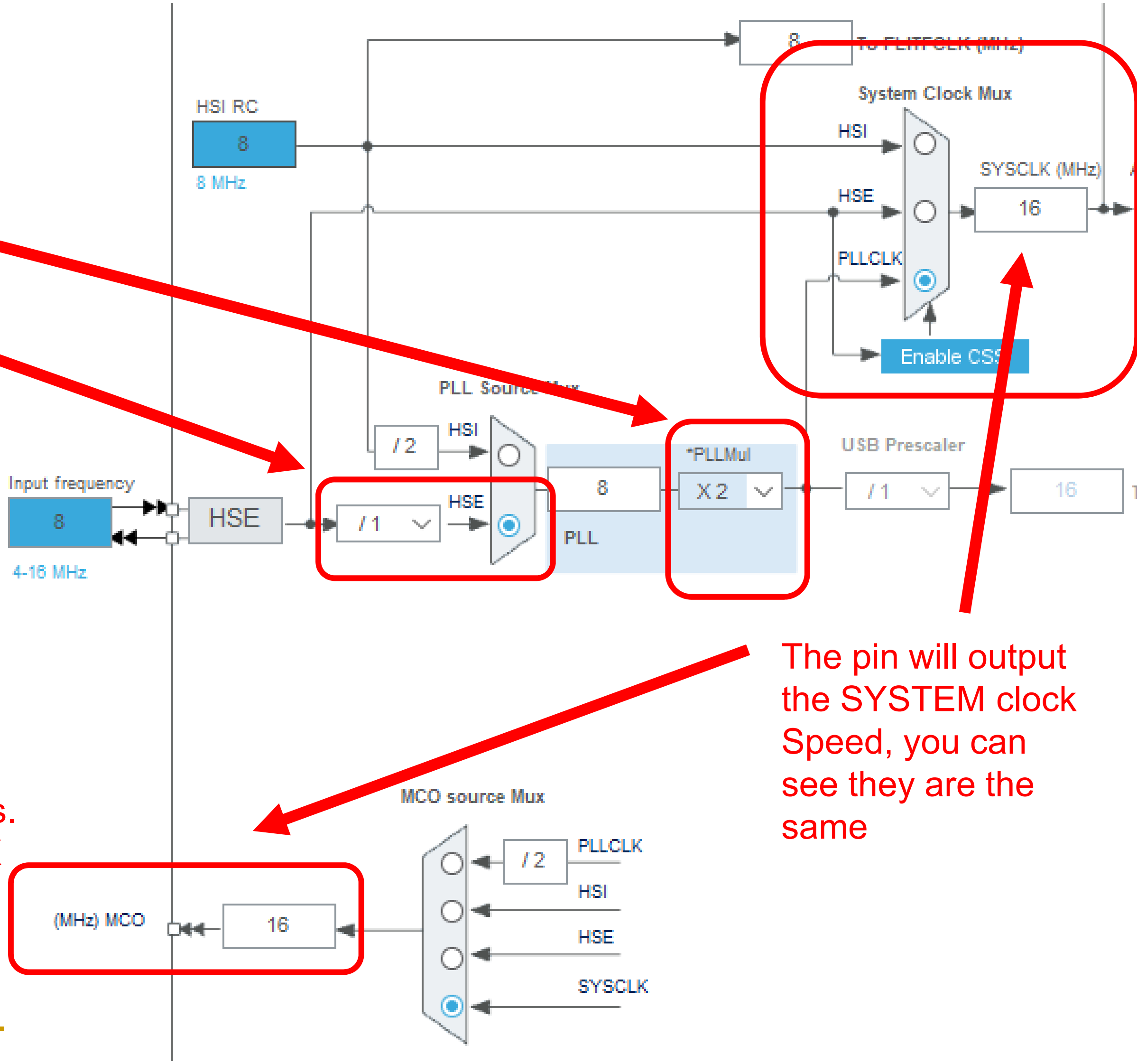
Red here because the I/O pin (PA.8) CANNOT output that speed

Choose SYSCLK

Question : What is the Maximum Speed that can be output for the I/O pin ?

MCO (Master Clock Output)

Change the PLLMul, and HSE divisor such that it uses a lower speed SYSCLK



The pin will output the SYSTEM clock Speed, you can see they are the same

You can try any combinations. In this example, the SYSCLK is 8MHz x 2 = 16MHz, which can be output to PA.8

LAB4 – Task 1

- Task 1 requires you to output the SYSCLK via the MCO pin and display the SYSCLK on the DMM.
  1. Refer to CubeIDE Video, create a simple Project that allows you to output the SYSLCK.
  2. Follow the steps before, change the HSE divisor PLLMul, such that you can set the SYSCLK to 8MHz.
  3. The reason for setting to 8MHz is because our DMM can only measure frequency less than 10MHz.
  4. Connect the Red Terminal of your DMM to the PA.8. Try to locate where is PA.8 by going through the MINI V3 Schematics.
  5. Run your program, you will be able to see a 8MHz signal on the DMM.

LAB4 – Task 1 Hint

- For changing the HSE divisor or PLLMul, you can either generate the code again or try to modify the code generated

- In main.c

```
void SystemClock_Config(void)

RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV2;
RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
```

- You can change the code there instead of re-generating the code.

LAB4 – Task 1 Hint

- Jumper Location 3

Default as shown

- Left <J18-J19> PA1 <-----> Cap T_KEY
- Right <J20-J21> PA8 <-----> Buzzer

- By default connects PA1 to Cap T_KEY, if PA1 has other use, the jumper needs to be removed.
- By default connects PA8 to Buzzer, if PA8 has other use, the jumper needs to be removed.

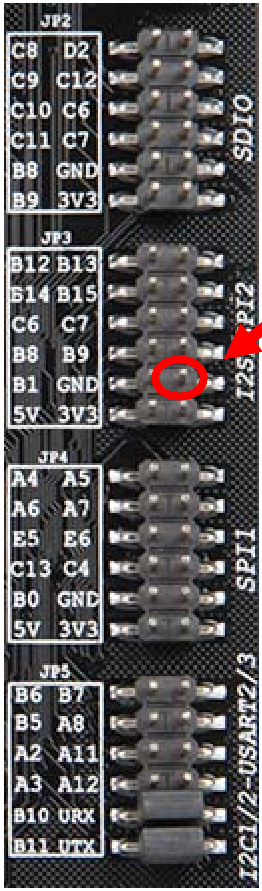
LAB4 – Task 1 Hint

- Right <J20-J21> PA8 <-----> Buzzer

By default connects PA8 to Buzzer, if PA8 has other use, the jumper needs to be removed

- Question : After you removed the Jumper, there are 2 points
- Which point connects to PA.8 ? Which point connects to Buzzer ?

Task 1 – Viewing the output



Connect to PA.8

Display

Hz for masuring Freq

% for measuring Duty Cycle

Switch between Hz / %

Set to Hz/Duty

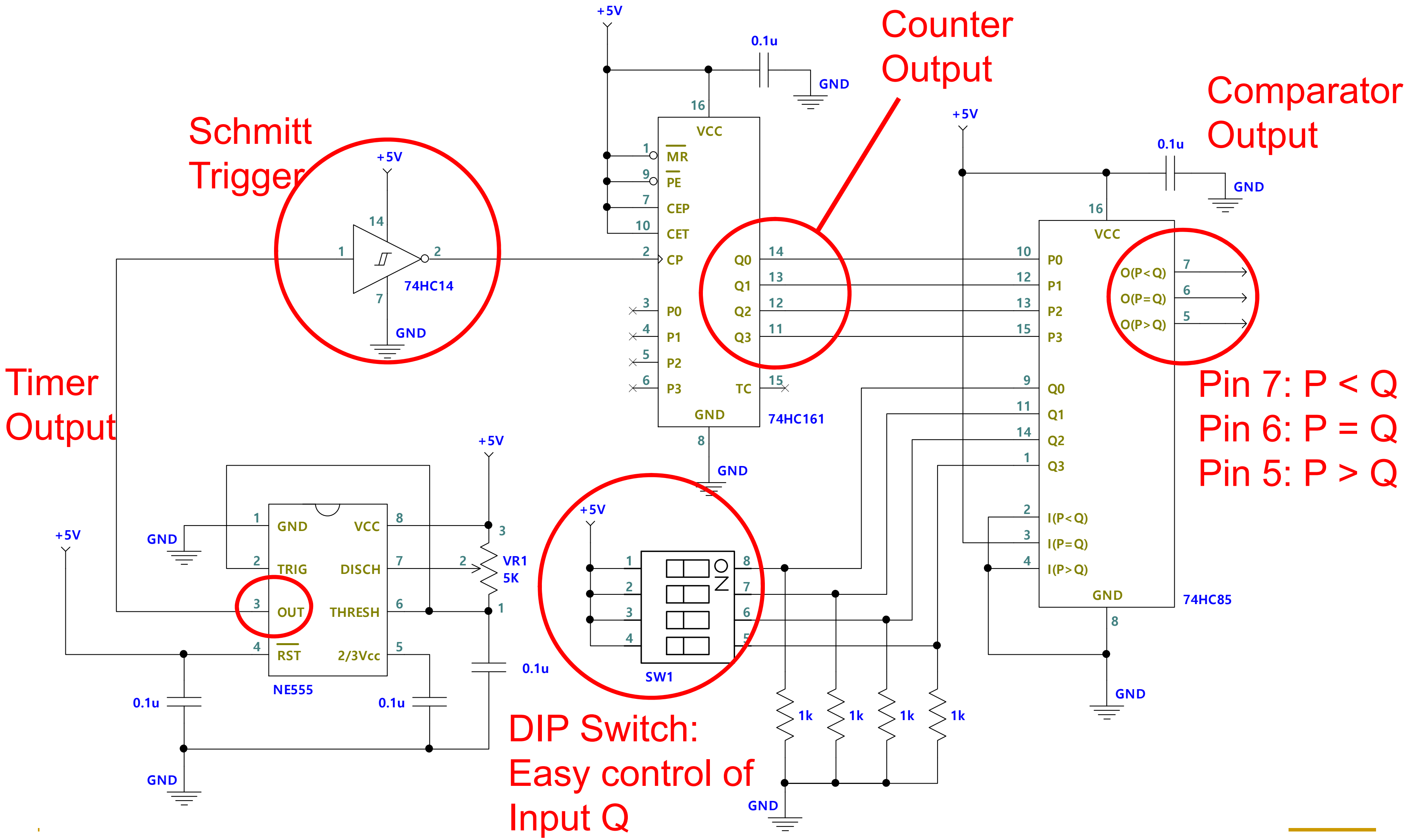
DO NOT HOOK DIRECTLY TO THE BOARD

USE the CONNECTION WIRES PROVIDED to lead out PA.8 PIN

Timers in STM32

- The high-density STM32F103xx performance line devices include up to two advanced control timers, up to four general-purpose timers, two basic timers, two watchdog timers and a SysTick timer.
  - TIM1 / TIM8 – advanced control timers
  - TIM2 / TIM3 / TIM4 / TIM5 – general purpose timers
  - TIM6 / TIM7 – basic timers

Recall from your ELEC 1100



Timers in STM32

- TIM1 / TIM8 – advanced control timers
- TIM2 / TIM3 / TIM4 / TIM5 – general purpose timers
- TIM6 / TIM7 – basic timers

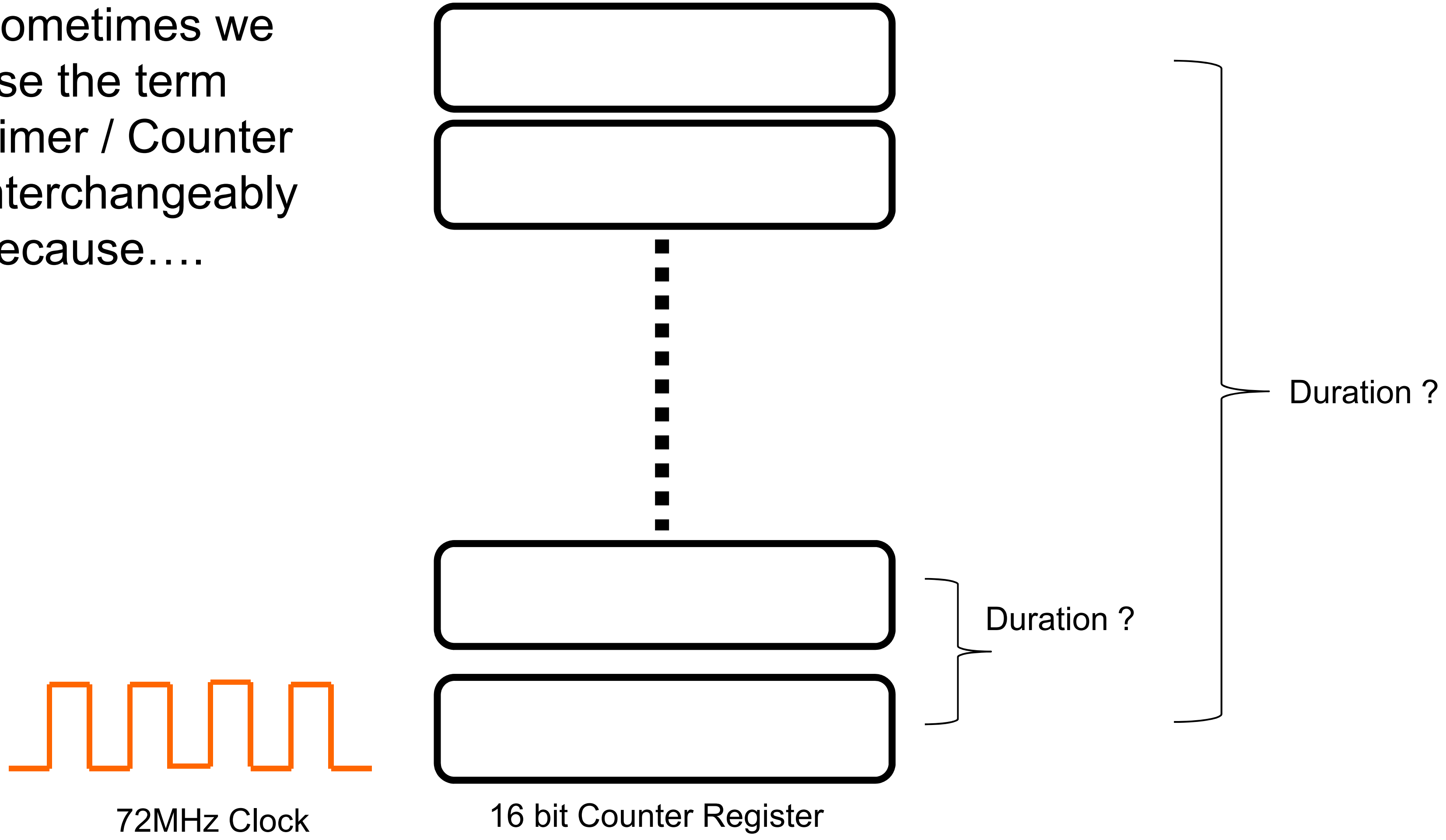
Table 4 compares the features of the advanced-control, general-purpose and basic timers.

Table 4. Timer feature comparison

Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/compare channels	Complementary outputs
TIM1, TIM8	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	Yes
TIM2, TIM3, TIM4, TIM5	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	No
TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No

16-bit Timer ? Counter ?

- Sometimes we use the term Timer / Counter interchangeably because….



Advanced Timers (TIM1 / TIM8)

- The two advanced-control timers (TIM1 and TIM8) can each be seen as a three-phase PWM multiplexed on 6 channels. They have complementary PWM outputs with programmable inserted dead-times. They can also be seen as a complete general-purpose timer. The 4 independent channels can be used for:
  - Input capture
  - Output compare
  - PWM generation (edge or center-aligned modes)
  - One-pulse mode output
- If configured as a standard 16-bit timer, it has the same features as the TIMx timer. If configured as the 16-bit PWM generator, it has full modulation capability (0-100%).

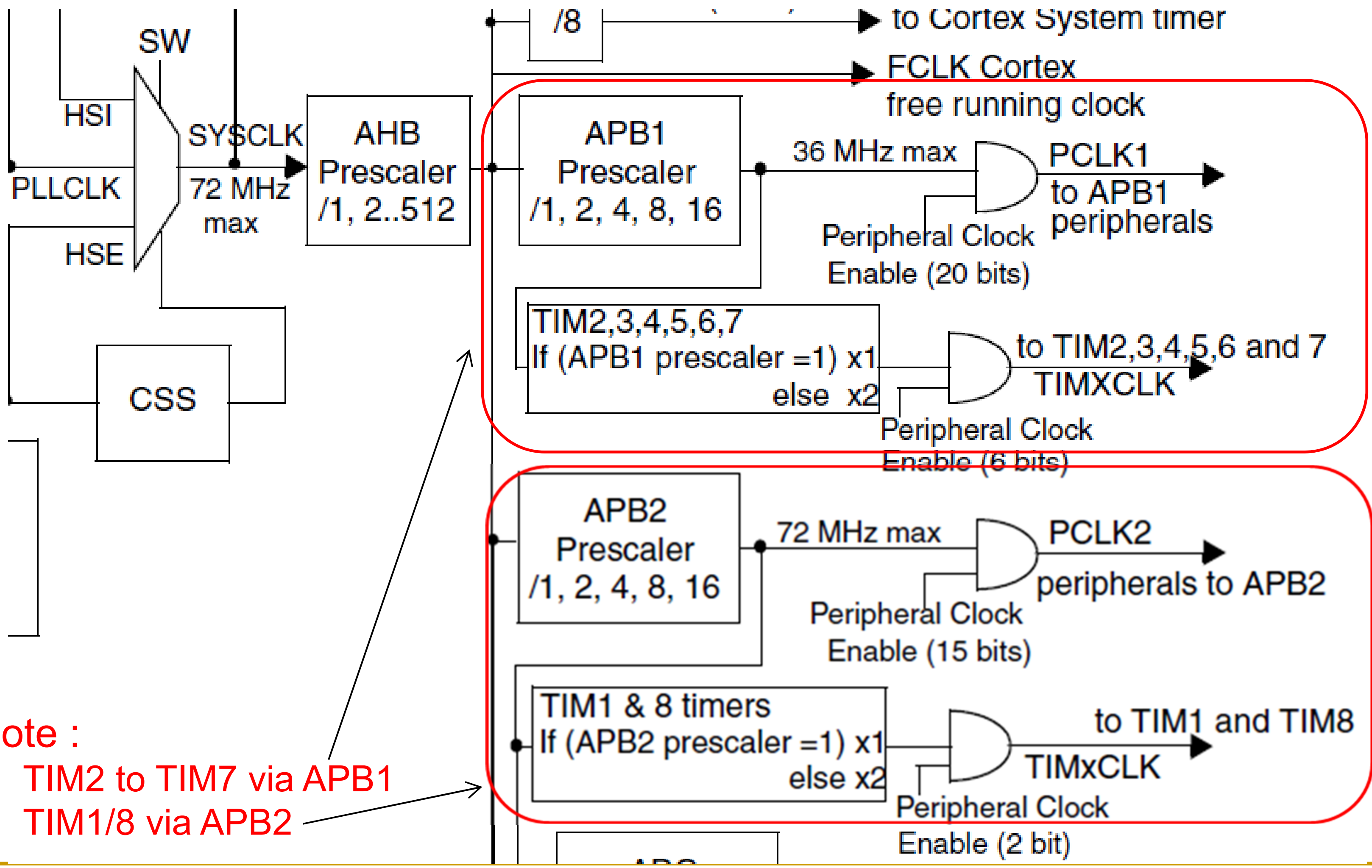
General-purpose Timers (TIMx)

- There are up to 4 synchronizable general-purpose timers (TIM2, TIM3, TIM4 and TIM5) embedded in the STM32F103xC, STM32F103xD and STM32F103xE performance line devices.
- These timers are based on a 16-bit auto-reload up/down counter, a 16-bit prescaler and feature 4 independent channels each for input capture/output compare, PWM or onepulse mode output.
- The general-purpose timers can work together with the advanced-control timer via the Timer Link feature for synchronization or event chaining. Their counter can be frozen in debug mode.
- Any of the general-purpose timers can be used to generate PWM outputs. They all have independent DMA request generation.
- These timers are capable of handling quadrature (incremental) encoder signals and the digital outputs from 1 to 3 hall-effect sensors.

Basic Timers (TIM6 / TIM7)

- These timers are mainly used for DAC trigger generation.
- They can also be used as a generic 16-bit time base.

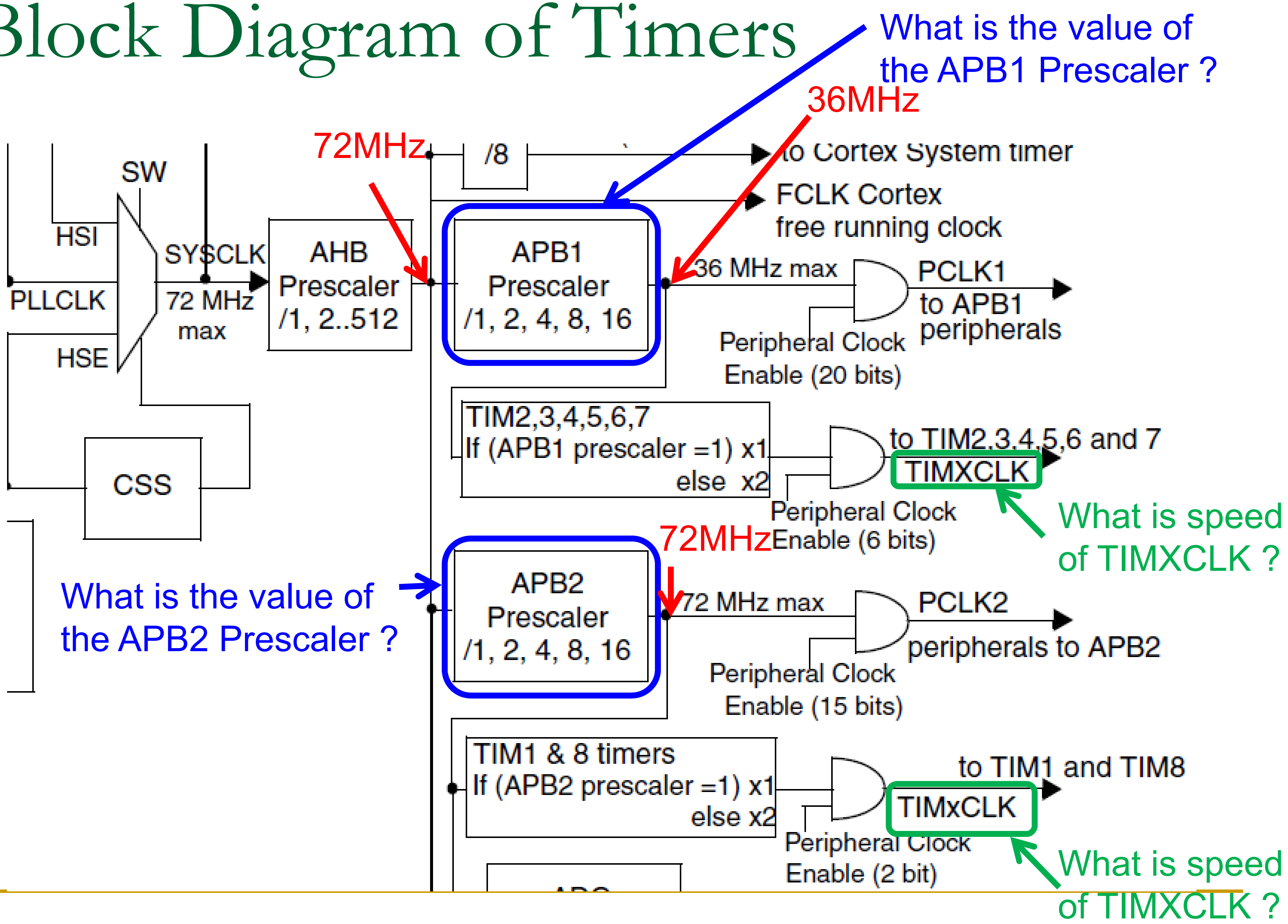
Block Diagram of Timers



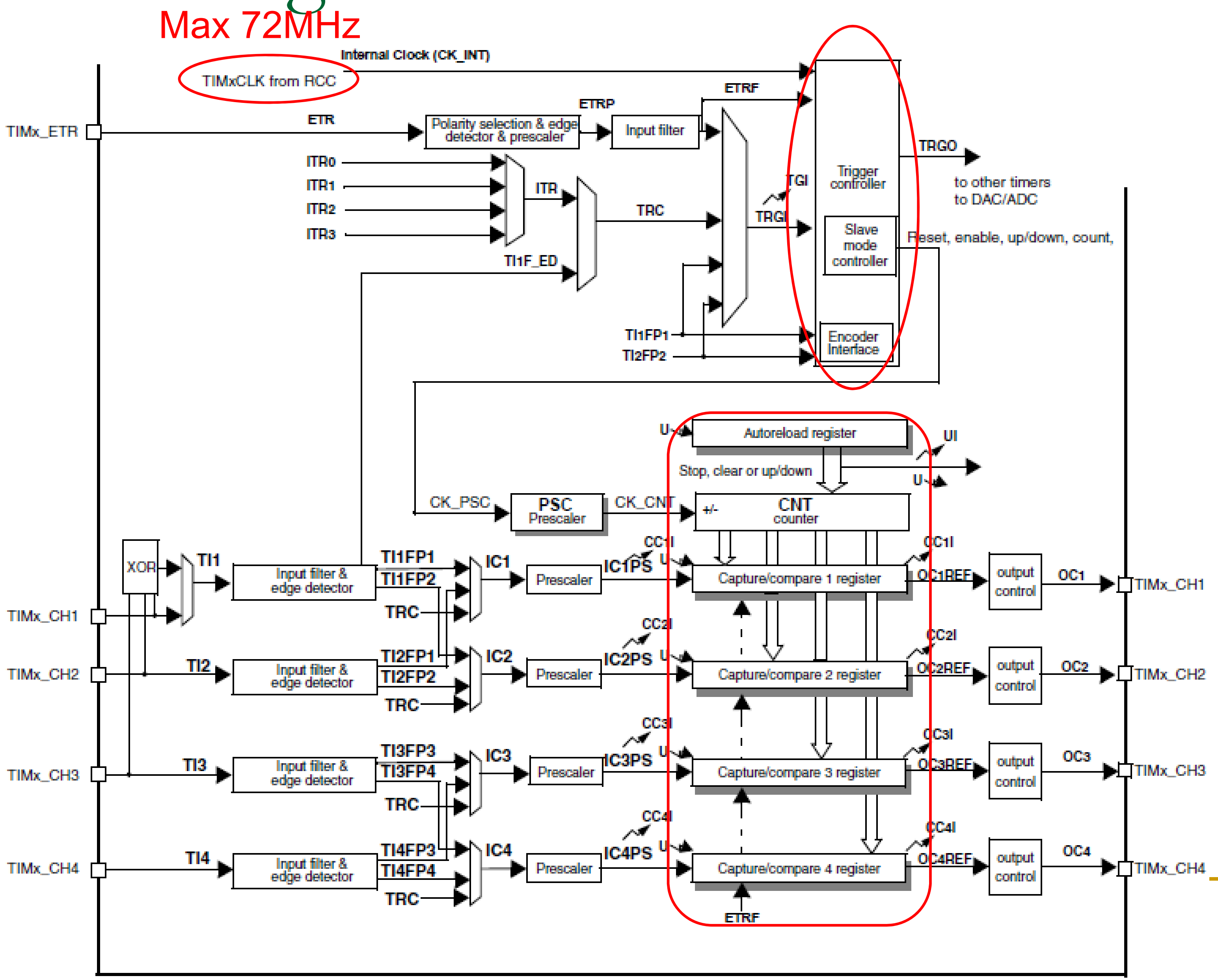
Note :

- TIM2 to TIM7 via APB1
- TIM1/8 via APB2

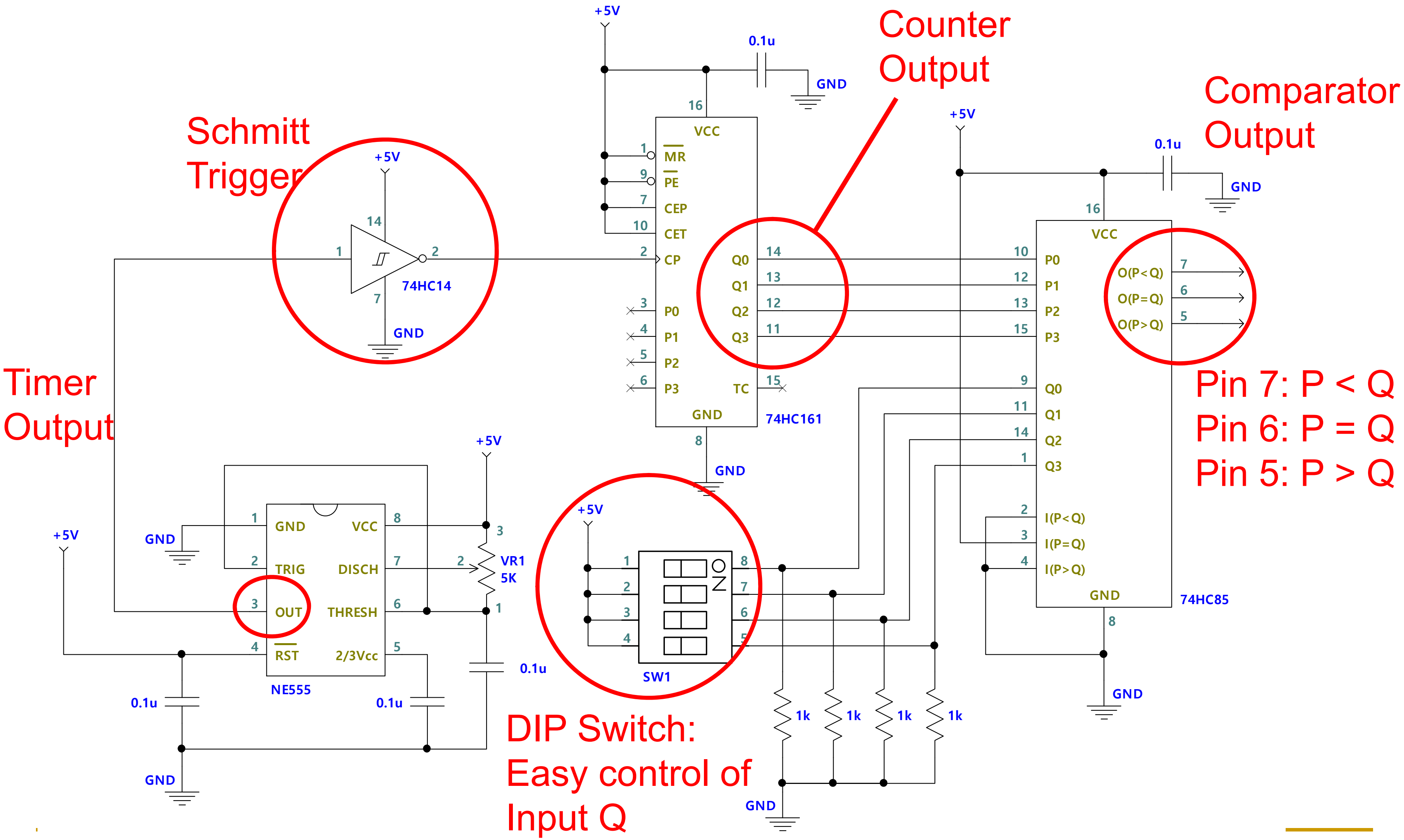
Block Diagram of Timers



Block Diagram of Timers



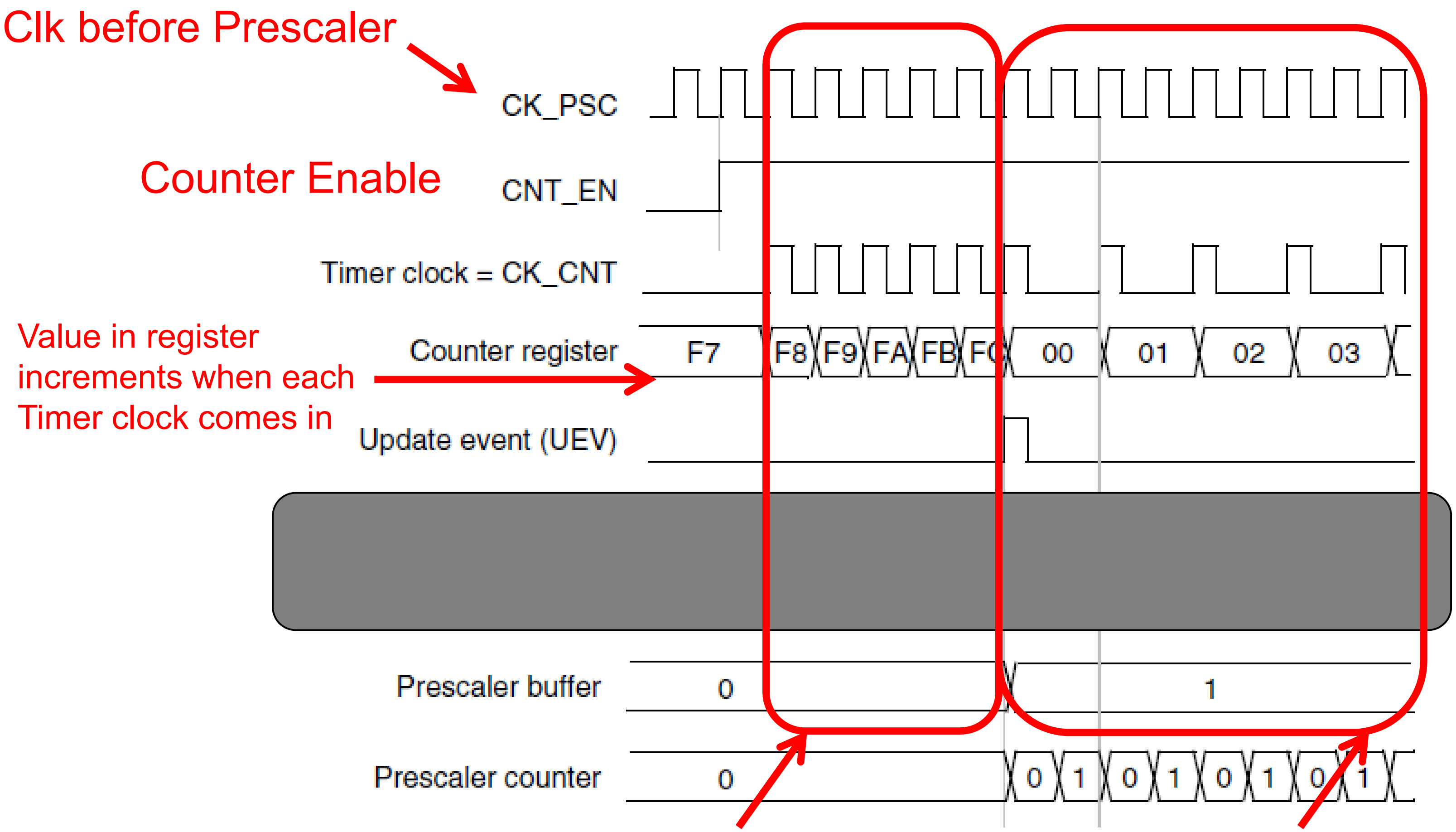
Recall from your ELEC 1100



Functional Description of Timer

- The counter, the auto-reload register and the prescaler register can be written or read by software. This is true even when the counter is running.
- The time-base unit includes:
  - Counter Register (TIMx_CNT)
  - Prescaler Register (TIMx_PSC)
  - Auto-Reload Register (TIMx_ARR)
- The counter is clocked by the prescaler output CK_CNT, which is enabled only when the counter enable bit (CEN) in TIMx_CR1 register is set.

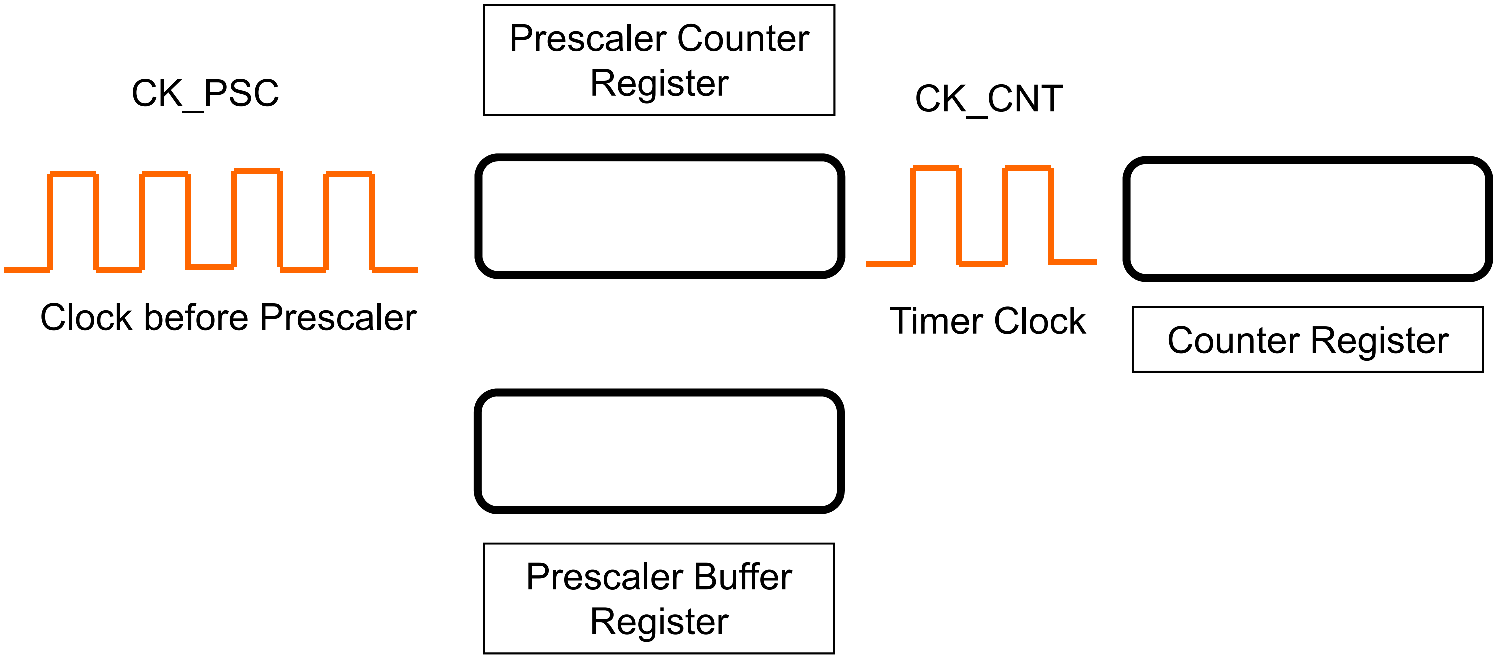
Functional Description of Timer



When prescaler = 0, what is the relation of CK_PSC and CK_CNT ?

When prescaler = 1, what is the relation of CK_PSC and CK_CNT ?

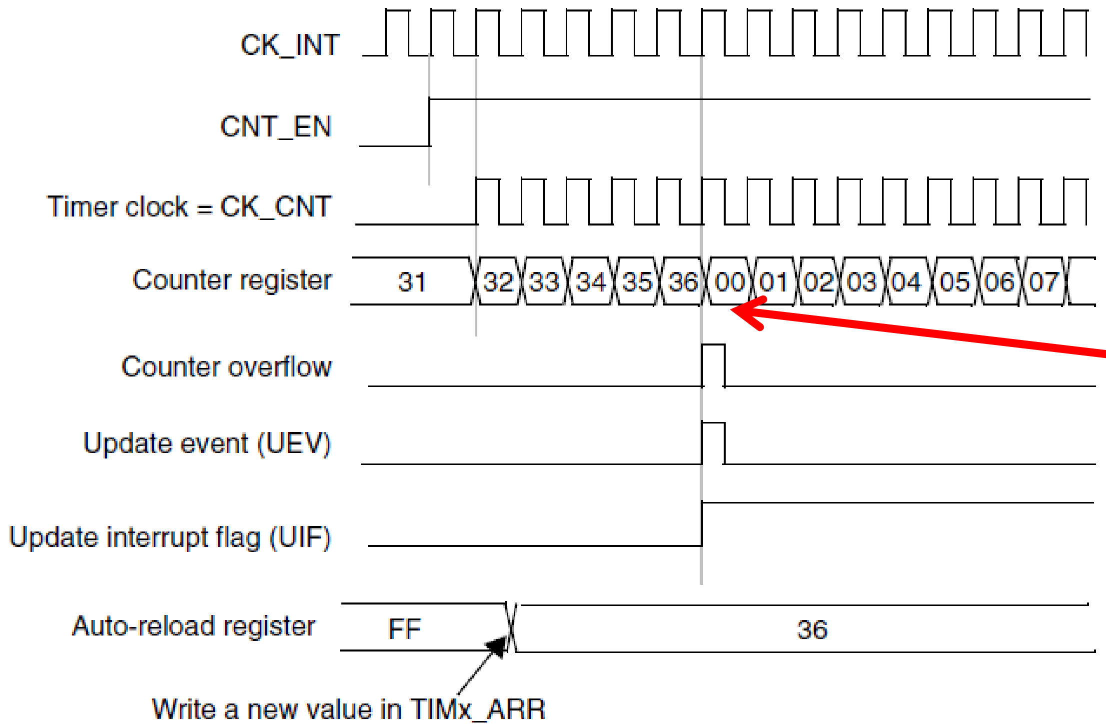
Prescaler



- What is the use of the Prescaler ?

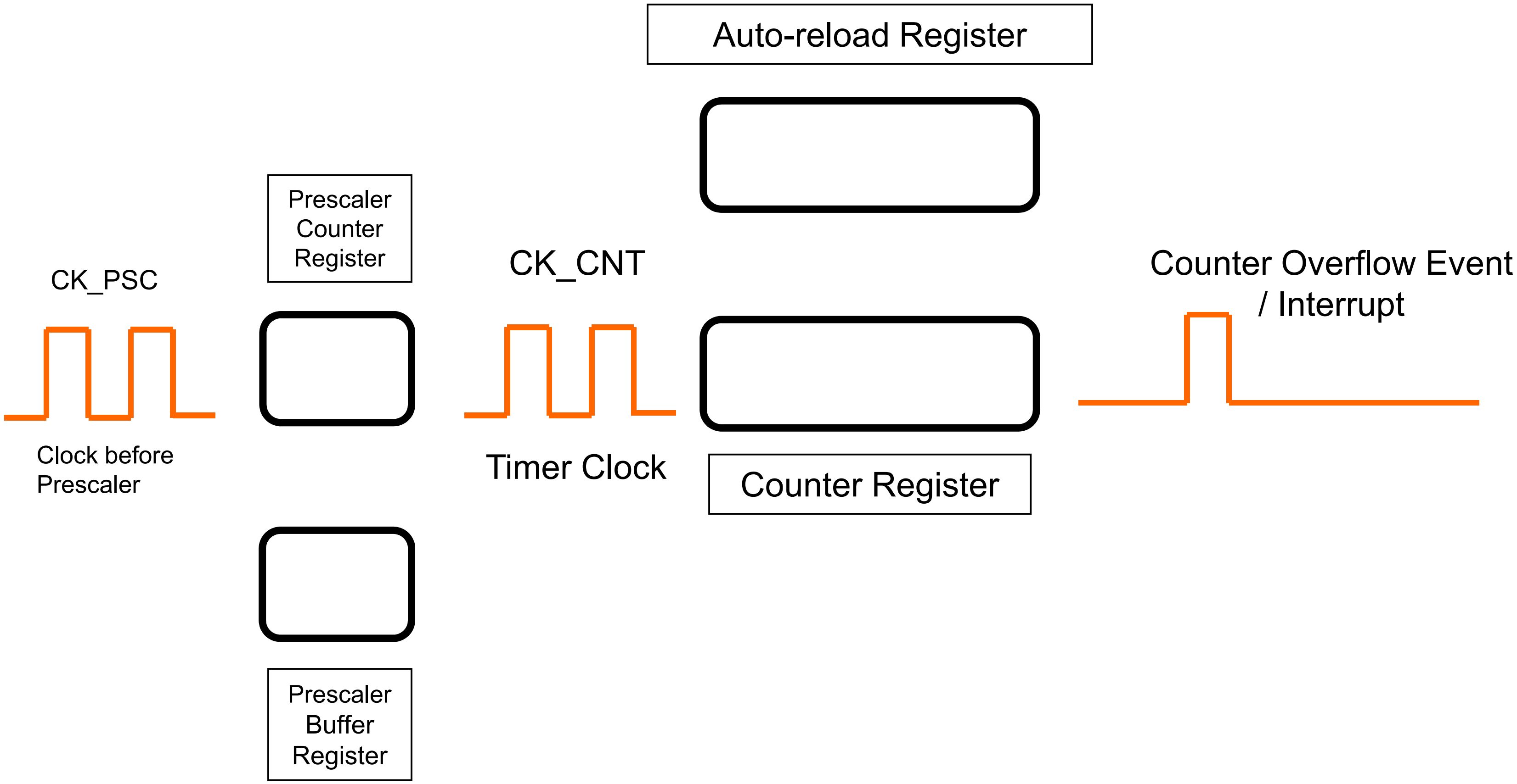
Autoreload

- In upcounting mode, the counter counts from 0 to the auto-reload value (content of the TIMx_ARR register), then restarts from 0 and generates a counter overflow event.
- The following figures show some examples of the counter behavior for different clock frequencies when TIMx_ARR=0x36.



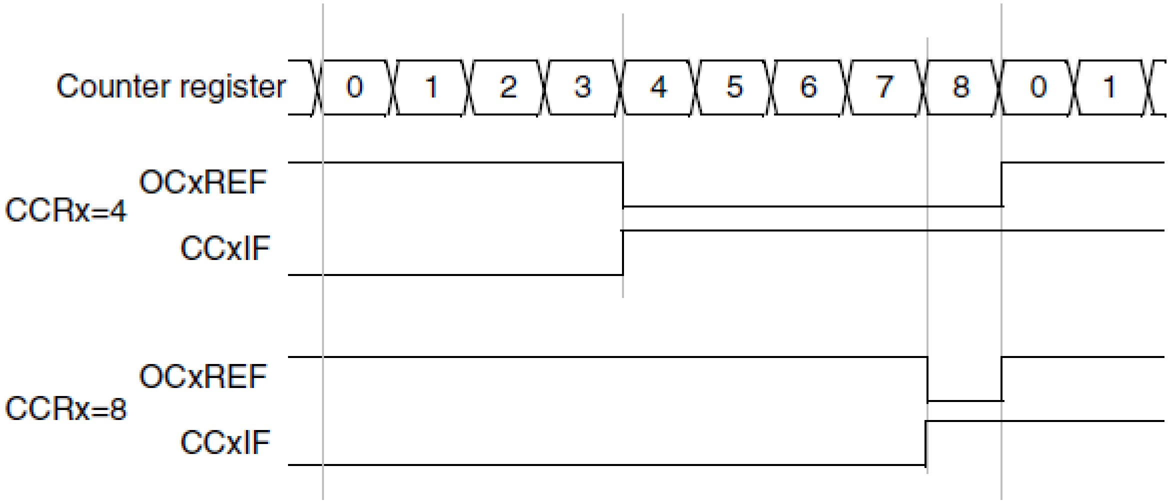
Note : Start with 0, end at 0x36. What is the relation between CK_INT and Counter overflow ?

Auto-reload Register



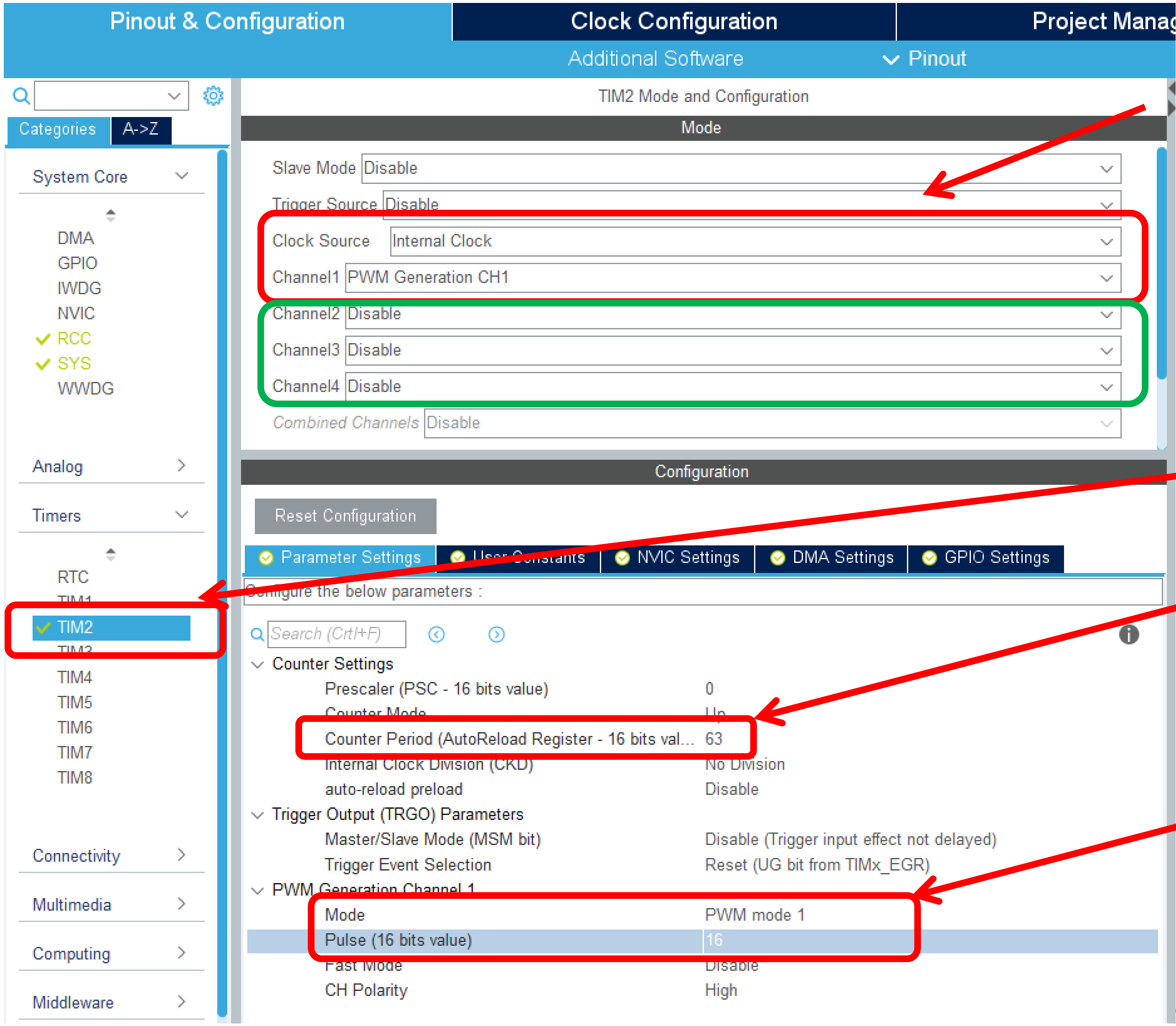
PWM Output using TIMx

- Pulse width modulation mode allows you to generate a signal with a frequency determined by the value of the TIMx_ARR register and a duty cycle determined by the value of the TIMx_CCRx register.
- The following shows and **Edge-aligned PWM waveforms (ARR=8)**



Generating PWM in STM32

- You can use CubeIDE to Initialize the PWM
- Let’s use TIM2 as an example



Using Internal Clock as Clock Source to generate the PWM in Channel 1

Actually there are 4 channels, each channel can output different duty cycle PWM

TIM2

Period = 63

Pulse = 16

Setup the Period (Frequency)

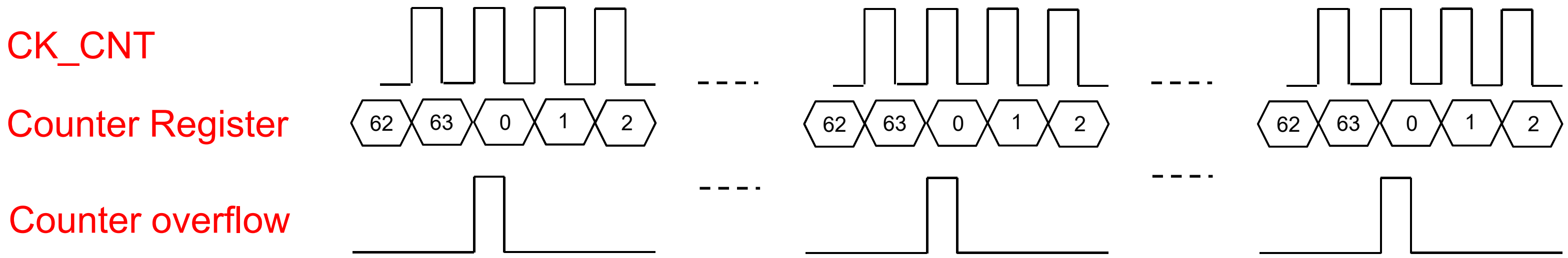
- You can check the code, initializations for the Period is shown

void MX_TIM2_Init(void)

```
htim2.Instance = TIM2;
htim2.Init.Prescaler = 0;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 63;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
```

Setup the Period (Frequency)

- From the above setting
- If CK_CNT = 72MHz, what is the frequency of Counter overflow ?



Setup the Pulse (Duty Cycle)

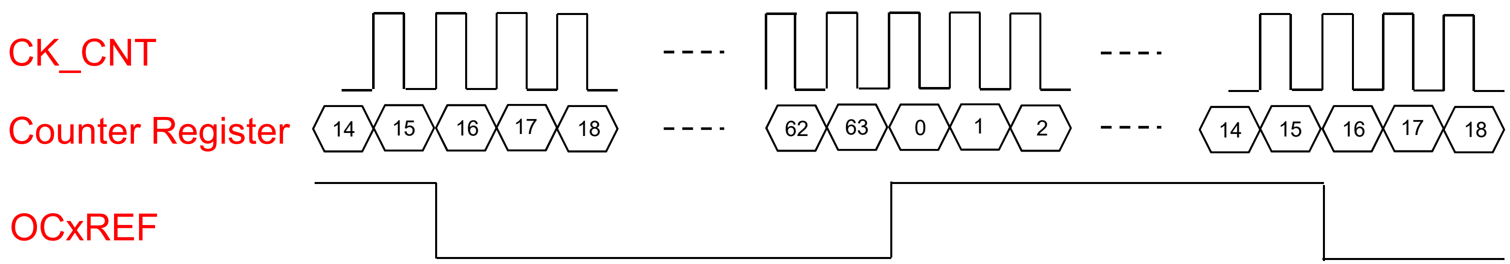
- You can check the code, initializations for the Pulse is shown

void MX_TIM2_Init(void)

```
if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
{
   Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
  Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 16;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
   Error_Handler();
}
```

Setup the Pulse (Duty Cycle)

- The Pulse argument is the number of High count.



- What is the duty cycle of OCxREF ?

Enable the PWM

- You configured the Timer in CubeIDE, if you want the Timer to run, you need to add a code on

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

Task 2 – Generating a PWM

- You need to output a PWM using TIM3
- The specification of PWM is as follows :

  Assuming your student ID digits are abcdefgh
  - Frequency = z0kHz z = ((g x 10 + h) mod 9) + 1
  - Duty Cycle = y0% y = ((f x 10 + g) mod 7) + 2

- Example, if your student ID is 20123456
  - z = ((5 x 10 + 6) mod 9) + 1 = 3 → Frequency = 30kHz
  - y = ((4 x 10 + 5) mod 7) + 2 = 5 → Duty Cycle = 50%

Task 2 – Generating a PWM

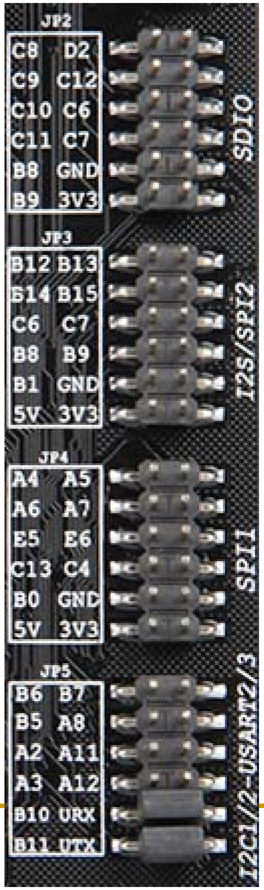
- You need to take the following note for finishing Task 2.
  1. Please **KEEP YOUR SYSCLK** at **8MHz as in Task 1**
  2. The output will be in PA.6 for TIM3_CH1

Pins						Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Alternate functions(4)	
BGA144	BGA100	WLCSP64	LQFP64	LQFP100	LQFP144					Default	Remap
L3	J3	G5	22	31	42	PA6	I/O		PA6	SPI1_MISO(8) TIM8_BKIN/ADC12_IN6 TIM3_CH1(8)	TIM1_BKIN

  3. Enable PWM for TIM3

Task 2 – Viewing the output

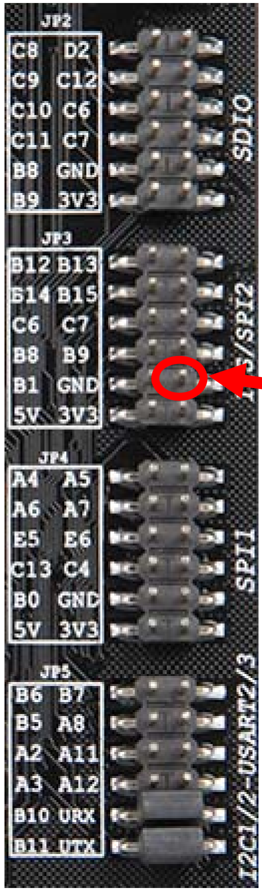
You need to connect PA.6 to the DMM, locate the PA.6 from the right side of the board



Make SURE you use the Connection Wires provided to lead out the pin then connect to the DMM Probe.

DO NOT connect the DMM directly to the board as it might short circuit to other pin and hence damage the board.

Task 2 – Viewing the output



Connect to PA.6

Display

Hz for masuring Freq

% for measuring Duty Cycle

Switch between Hz / %

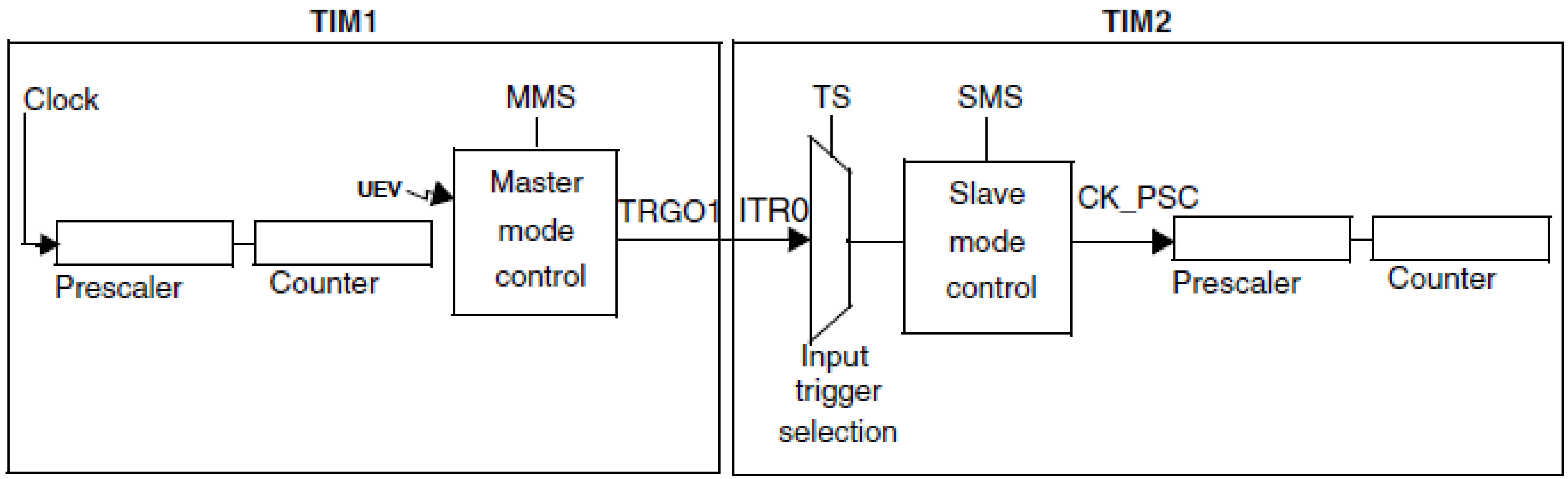
Set to Hz/Duty

DO NOT CONNECT DMM Cable DIRECTLY TO THE BOARD

USE the CONNECTION WIRES PROVIDED to lead out PA.6 PIN

Timer synchronization

- The TIMx timers are linked together internally for timer synchronization or chaining. When one Timer is configured in Master Mode, it can reset, start, stop or clock the counter of another Timer configured in Slave Mode.



Timer synchronization

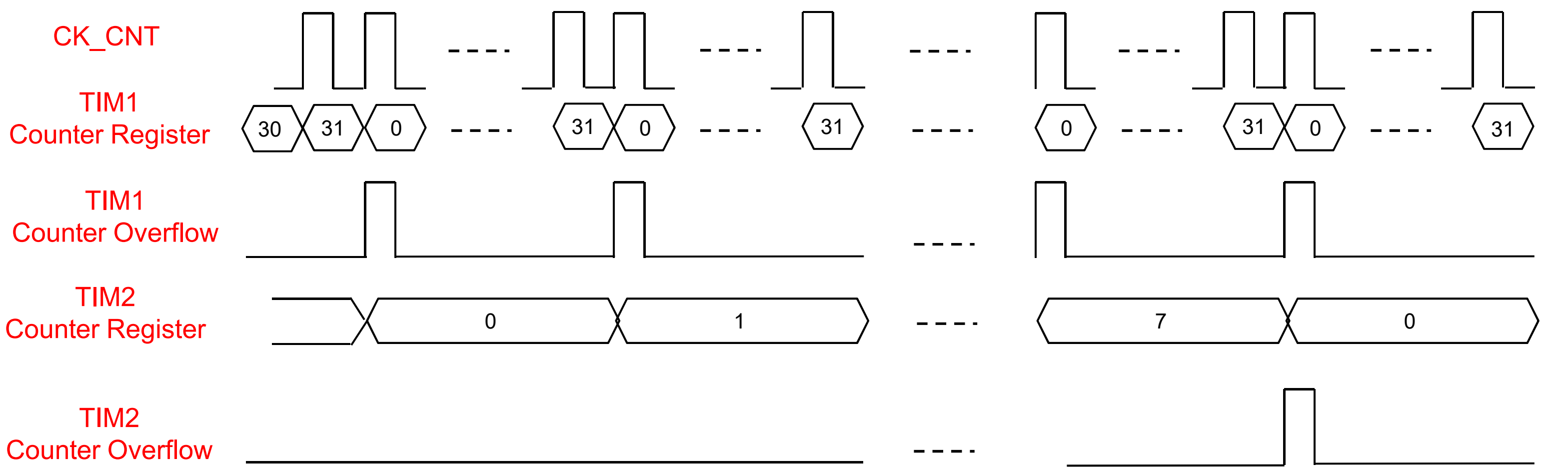
- In the previous figure,
- You can configure Timer 1 to act as a prescaler for Timer 2
- To do this, you need to
  1. Configure Timer 1 in master mode so that it outputs a periodic trigger signal on each update event UEV. A rising edge is output on TRGO1 each time an update event is generated.
  2. To connect the TRGO1 output of Timer 1 to Timer 2, Timer 2 must be configured in slave mode using ITR0 as internal trigger. You select this through the TS bits in the TIM2_SMCR register (writing TS=000).
  3. Then you put the slave mode controller in external clock mode 1 (write SMS=111 in the TIM2_SMCR register). This causes Timer 2 to be clocked by the rising edge of the periodic Timer 1 trigger signal (which correspond to the timer 1 counter overflow).
  4. Finally both timers must be enabled by setting their respective CEN bits (TIMx_CR1 register).

Timer synchronization

- Details steps
- To do this, you need to
  1. Configure Timer 1 master mode to send its Update Event (UEV) as trigger output (MMS=010 in the TIM1_CR2 register), then it outputs a periodic signal on each counter overflow.
  2. Configure the Timer 1 period (TIM1_ARR registers).
  3. Configure Timer 2 to get the input trigger from Timer 1 (TS=000 in the TIM2_SMCR register).
  4. Configure Timer 2 in external clock mode 1 (SMS=111 in TIM2_SMCR register).
  5. Start Timer 2 by writing ‘1 in the CEN bit (TIM2_CR1 register).
  6. Start Timer 1 by writing ‘1 in the CEN bit (TIM1_CR1 register).

Timer synchronization

- Below shows the case when TIM1_ARR = 31, TIM2_ARR = 7

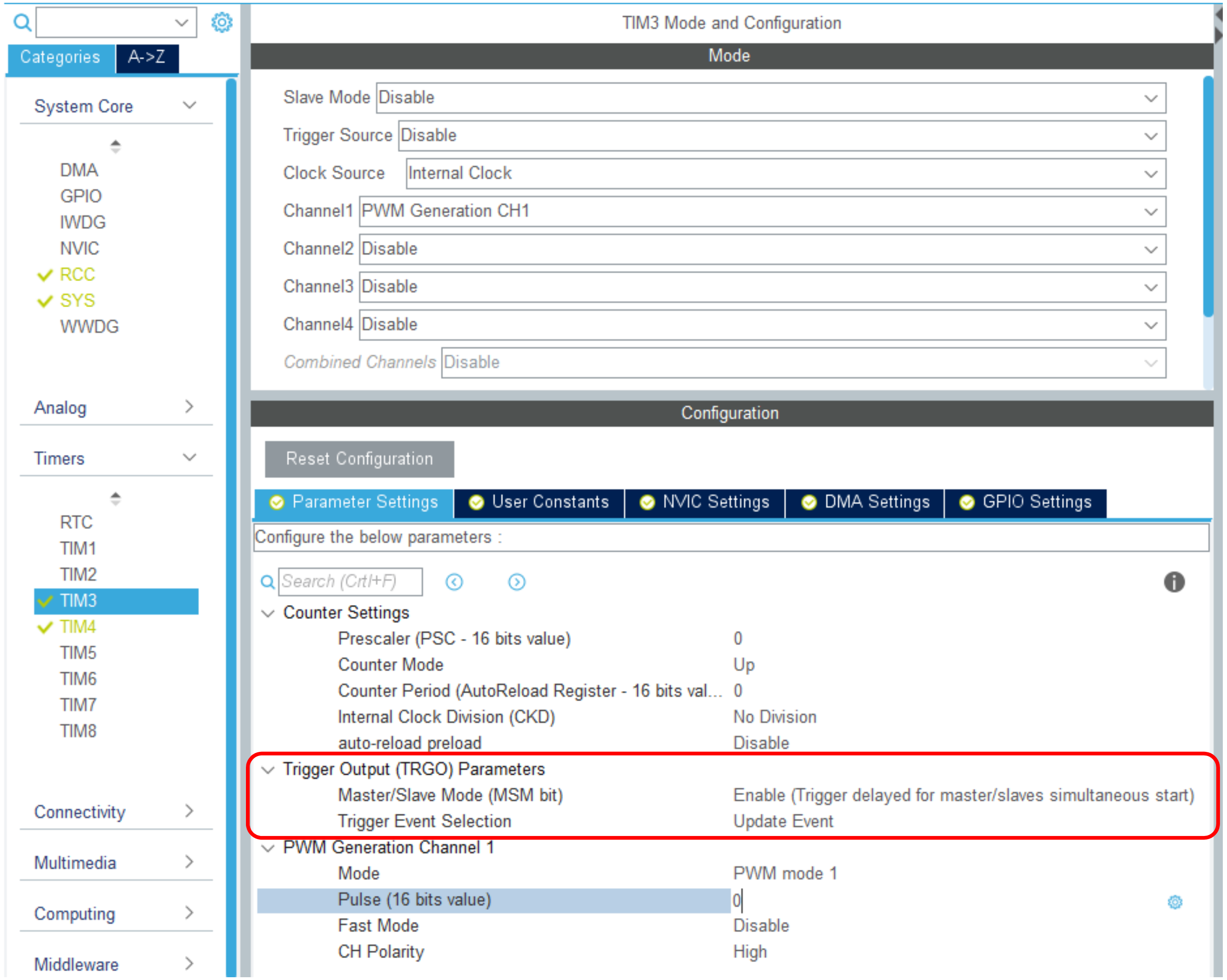


- If CK_CNT = 72MHz
  - What is the frequency of TIM1 Counter overflow ?
  - What is the frequency of TIM2 Counter overflow ?

Task 3 – Generating a PWM

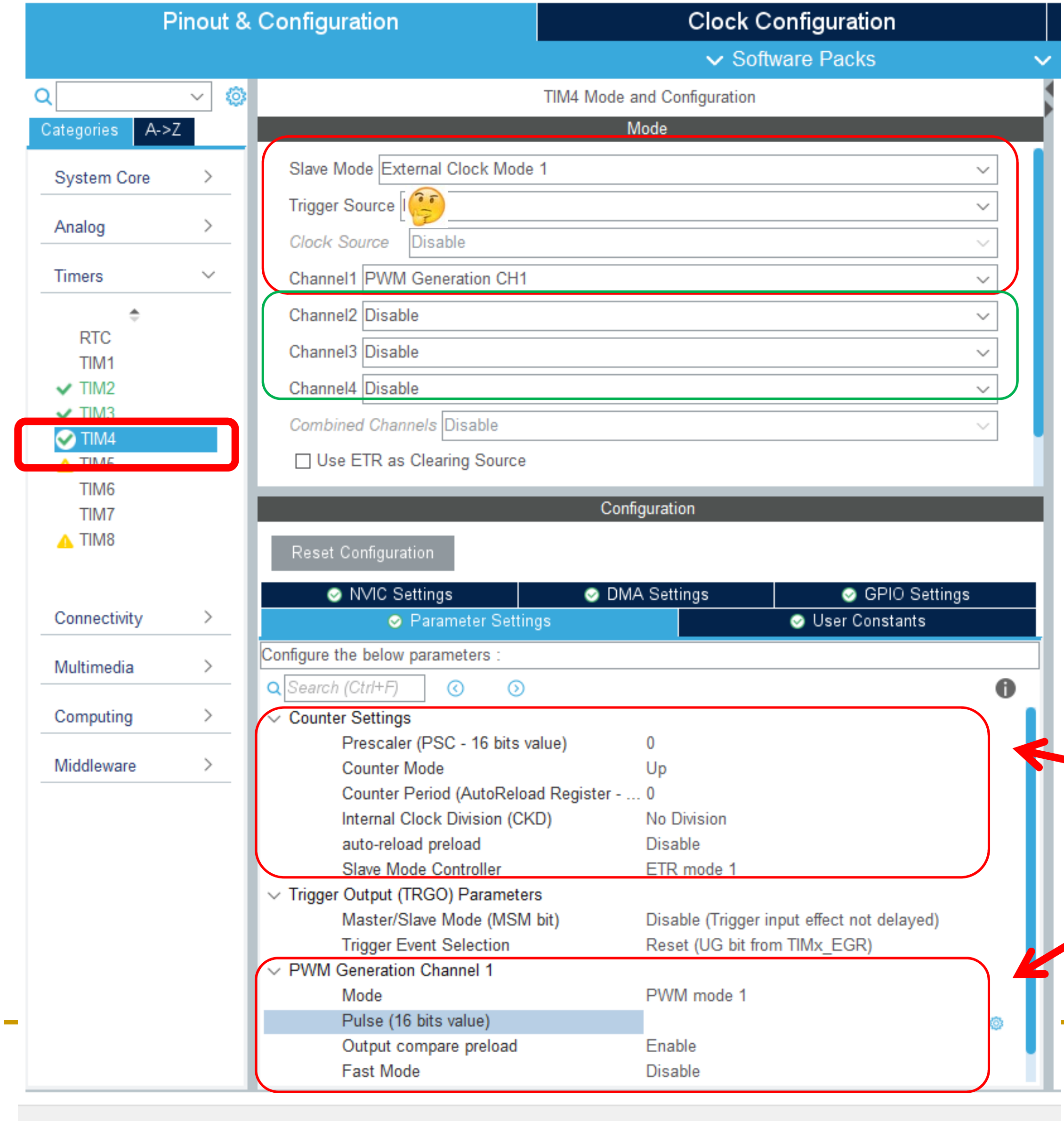
- You need to output a PWM using TIM4 using TIM3 as an input
  - You need to use your Task 2 result to do Task 3
  - If number is not divisible, please use closest match.
- The specification of PWM is as follows :
- Assuming your student ID digits are abcdefgh
  - Frequency = w00Hz   w = ((d *10 + e) mod 9) + 1
  - Duty Cycle = x0%   x = ((c * 10 + d) mod 7) + 2
- Example, if your student ID is 20123456
  - w = ((2 x 10 + 3) mod 9) + 1 = 6 → Frequency = 600Hz
  - x = ((1 x 10 + 2) mod 7) + 2 = 7 → Duty Cycle = 70%

Task 3 – For TIM3



Enable the Master Mode
Trigger Event to Update

Task 3 – For TIM4



Slave Mode : External Clock Mode 1
Trigger Source : Refer to Page 57

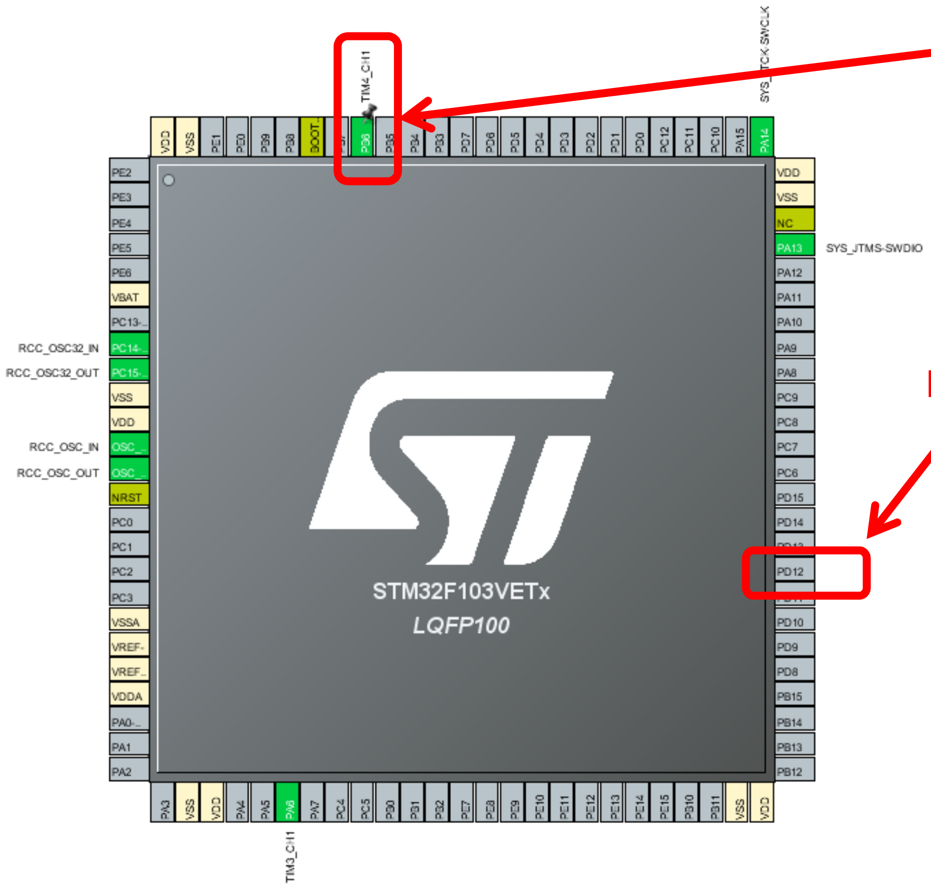
Channel 1 : PWM G. CH1

Same as TIM3, there are 4 channels, each channel can output different duty cycle PWM

Set suitable Period and Pulse for your Student ID

Task 3 – For TIM4

- Please note that the CubeIDE may use PD.12 as TIM4_CH1. If it is the case, please choose PB.6 to be the TIM4_CH1.



Use PB.6

DO NOT Use PD.12

Challenge : Try to locate where PD.12 is connected in the MINI V3 development board. Understand why we do not use PD.12 ?

Task 3 – Generating a PWM

- You need to take the following note for finishing Task 3.
  1. Please **KEEP YOUR SYSCLK** at **8MHz as in Task 1**
  2. The output will be in PB.6 for TIM4_CH1

C6	B5	B5	58	92	136	PB6	I/O	FT	PB6	I2C1_SCL[8]/ TIM4_CH1[8]	USART1_TX

  3. Enable PWM for TIM4

TIMx Internal trigger connection

- The internal trigger connection by different timers is specified in the in the TIMx_SMCR Register (Bit 6:4) **TS:** Trigger selection

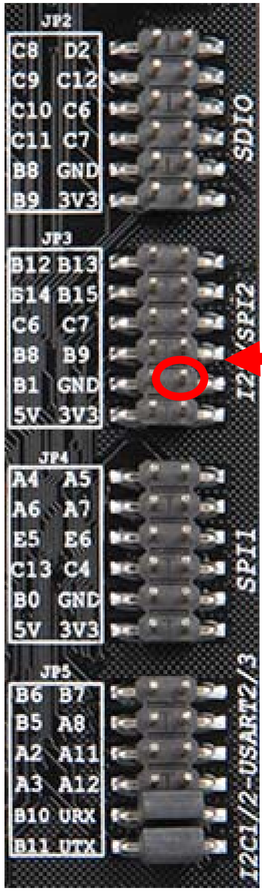
Table 86. TIMx Internal trigger connection[1]

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1	TIM8	TIM3	TIM4
TIM3	TIM1	TIM2	TIM5	TIM4
TIM4	TIM1	TIM2	TIM3	TIM8
TIM5	TIM2	TIM3	TIM4	TIM8

1. When a timer is not present in the product, the corresponding trigger ITRx is not available.

- So, in Task 3, Master is TIM3, Slave is TIM4, you need to enable which ITRx as the internal trigger ?

Task 3 – Viewing the output



Connect to PB.6

Display

Hz for masuring Freq

% for measuring Duty Cycle

Switch between Hz / %

Set to Hz/Duty

DO NOT CONNECT DMM Cable DIRECTLY TO THE BOARD

USE the CONNECTION WIRES PROVIDED to lead out PB.6 PIN

Task 2 and 3 – Debug Skills

- How can you debug if PA.6 (Task 2) or PB.6 (Task 3) have no output after you run the program ?
- How do you know if it is a hardware or software problem ?

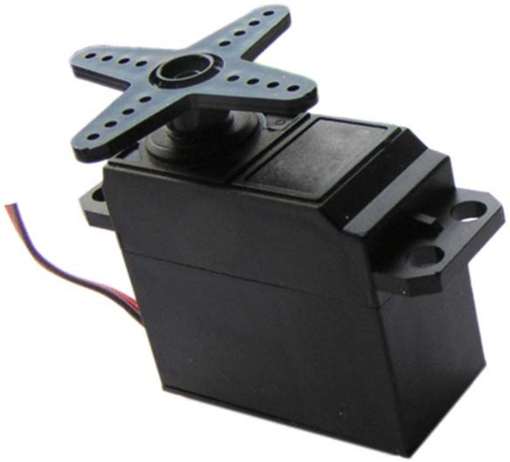
```
if (Software problem) {
      most likely your problem… 😬
}
if (Hardware problem) {
      Can solve in existing board ? 🤔
}
```

Task 4 – Change Optimization

- Your C++ Optimization should by default set to None (-O0) when generated by CubeIDE.
- Try to change your Optimization to Optimize most (-O3).
- Compile your program check the frequency by using DMM.
- Answer the TA if the frequency changed

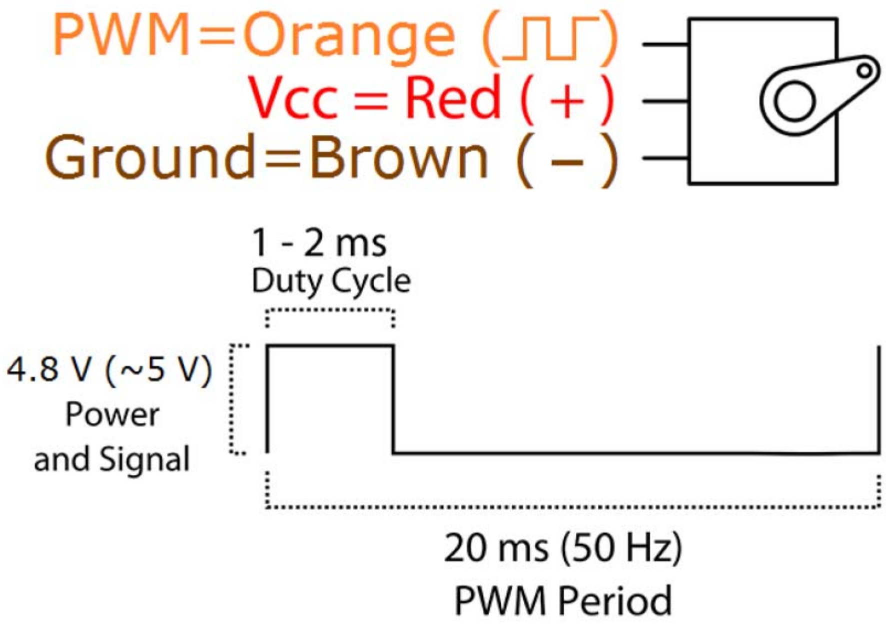
Servo Motor

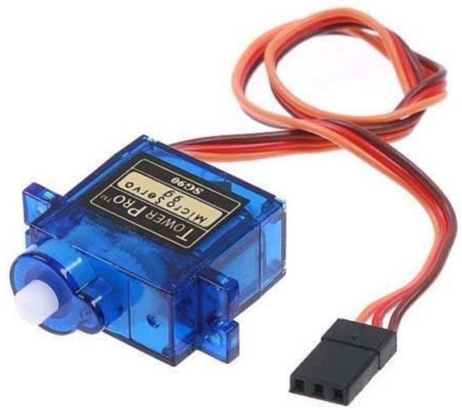
- One use of the PWM is to control a Servo Motor.
- A servomotor is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration. [Wiki]



Servo Motor

- In this LAB, we will use a SG90 Servo motor.
  - 1.5 ms pulse will set the motor in middle,
  - ~2 ms pulse will set the motor 45 degrees to the right
  - ~1 ms pulse will set the motor 45 degrees to the left





Task 5 – Control a Servo Motor

- Combining with your knowledge of LAB2, write a program to perform the following task.

At start, servo will stay at the middle

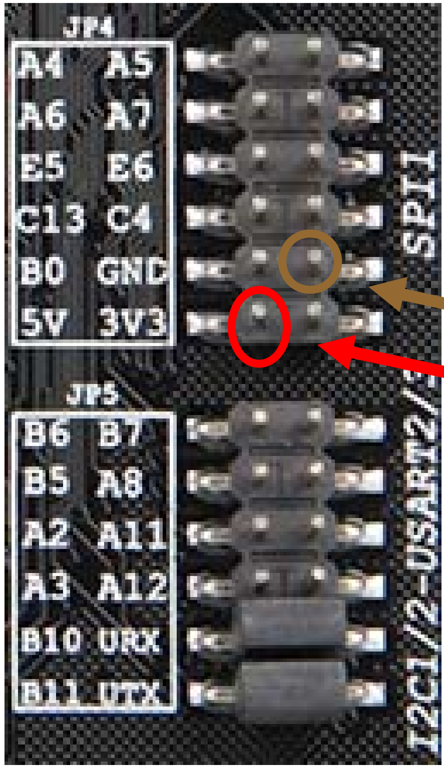
If K1 is pressed, servo turns to one side by 30 degrees from the middle,
    when K1 is released it will stay at that position

If K2 is pressed, servo turns to the opposite side by 30 degrees from the middle
    when K2 is released it will stay at that position

If both K1 AND K2 are pressed together, servo will stay at the middle.
    when both K1 and K2 are released it will still stay at middle

Task 5 – Control a Servo Motor

- Connect the Signal pin (ORANGE) of the Servo to your PWM
- You can choose any PWM pin you want



DO NOT CONNECT the Servo DIRECTLY TO THE BOARD USE the CONNECTION WIRES PROVIDED to lead out GND PIN

BROWN Connect to GND

RED Connect to 5V

ORANGE Connect to PWM

Task 5 – Hint

- Per your previous tasks, you might use CubeIDE to generate your PWM code, once PWM changed, you might think to regenerate the code again.
- However, like Task 5, you need to alter the PWM Pulse (Duty Cycle) dynamically. Try to Google and see how can you write your own function to achieve that.

- You can also think how you can alter the PWM Period (Frequency) by referring to the generated code.
  - `static void MX_TIM4_Init(void); // This is for TIM4 only`

  Can you add some parameter/argument instead of void ?

- To ease your demo without re-compiling your code, can you use another Timer to generate the PWM ? Noted that TIM3 and TIM4 already used.

After finishing LAB4 you are expected to…

1. Understand how clock governs the speed of the STM32.
2. Relate the function of timers to what you learnt in ELEC 1100.
3. Use CubeIDE to initialize different Timers interface.
4. Calculate the value used based on the clock and timer to generate a specific frequency.
5. Generate different PWM by using Timers in STM32
6. Cascade the Timers in STM32 together.
7. Control the different device by PWM (servo/motor).
8. Integrate the knowledge of LAB2 (key input) to add to LAB4 by referring to the schematic.

END