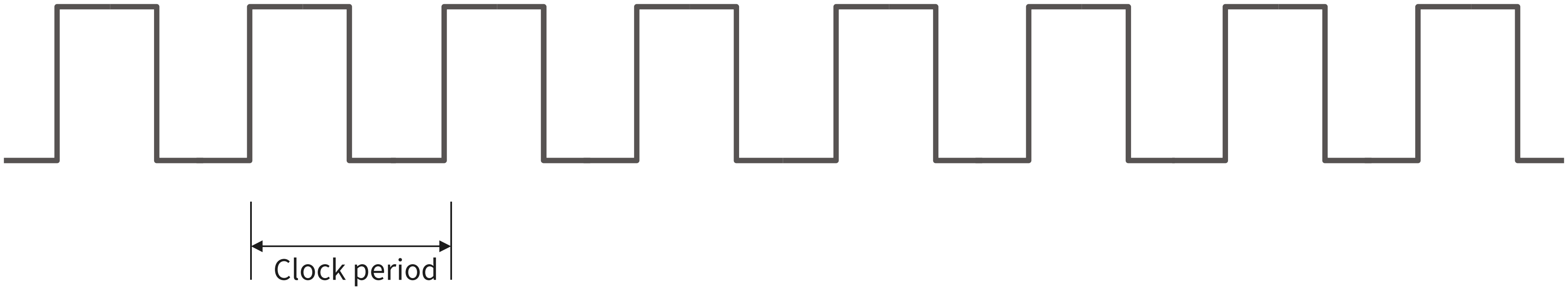
Embedded Systems Course - Timer, Counter and PWM

Boya Vinay Kumar

restricted

Timekeeping on microcontrollers – 1/2



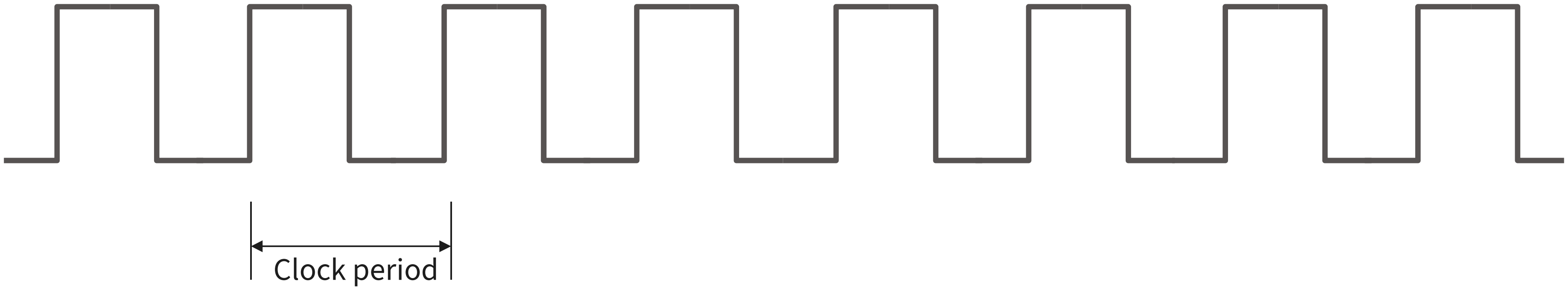
Clock Period T = 1/Frequency

Example:

Frequency: 100 Hz

Period: 1/100 = 0.01s or 10ms

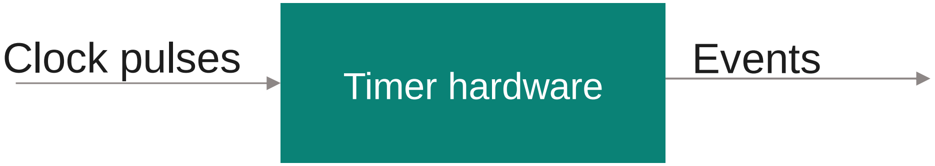
Timekeeping on microcontrollers – 2/2



Quiz: If each clock period is 10ms, what is the overall duration of 200 clock periods?

Answer: 2s

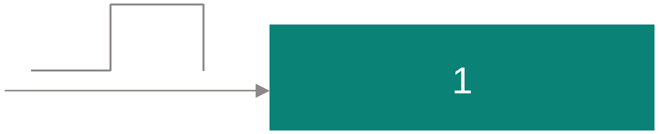
Introduction of Timers and Counters



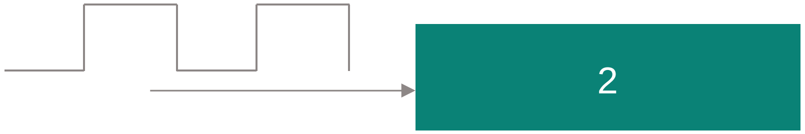
Clock pulses

Timer hardware

Events



1



2

1 clock pulse given to timer hardware

Timer counts 1

2 clock pulse given to timer hardware

Timer counts 2

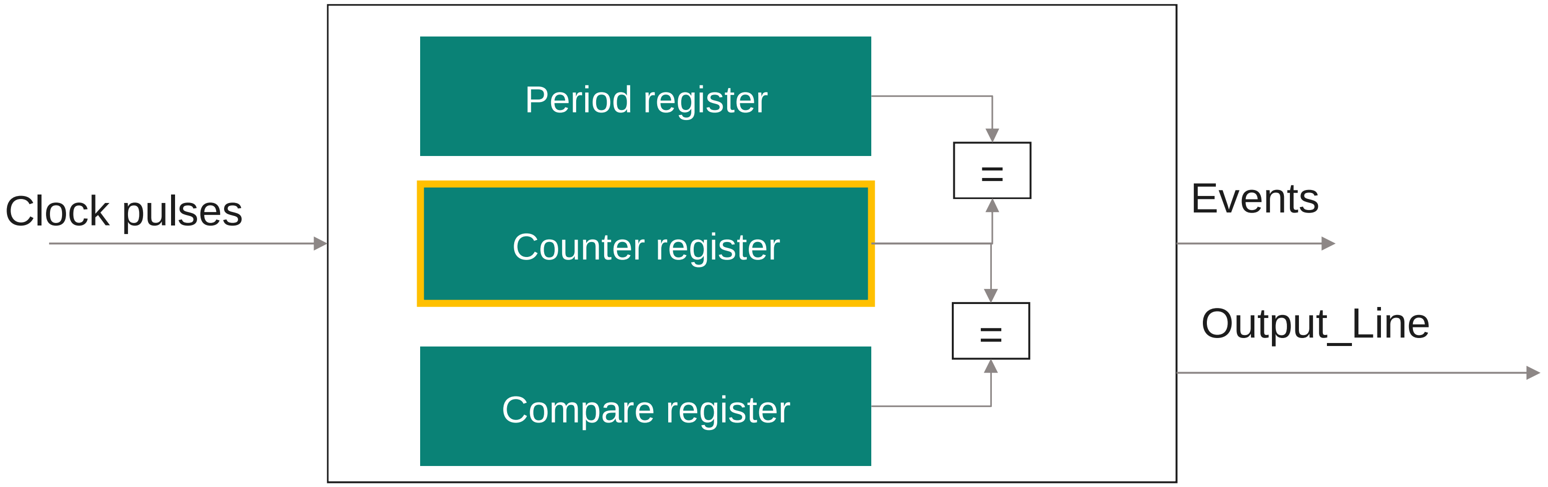
Example

Timer has counted up to 5.

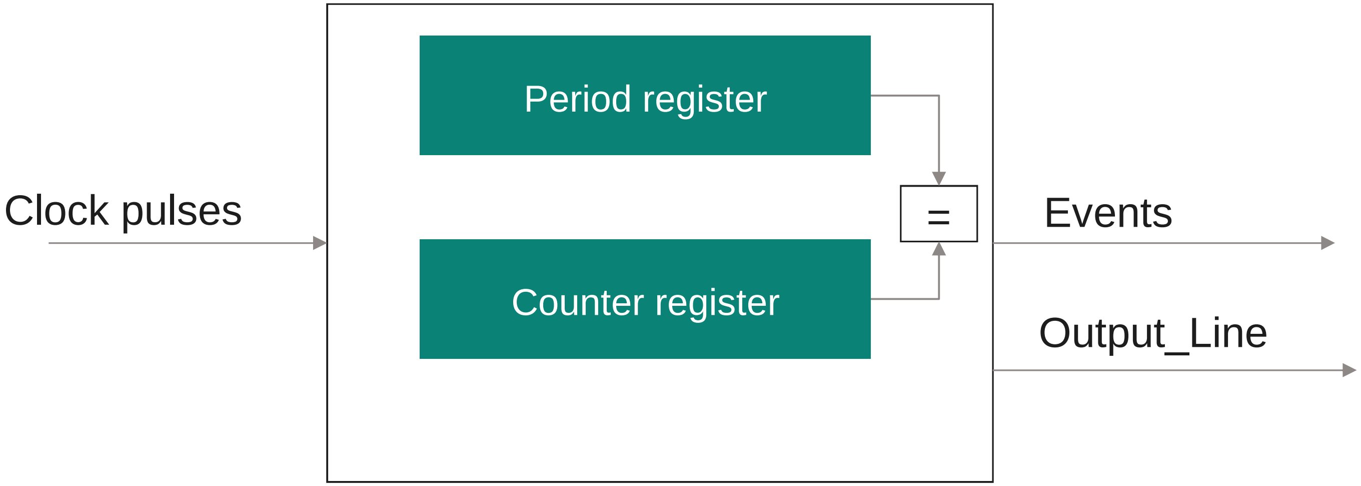
It means the timer hardware has received 5 clock pulses.

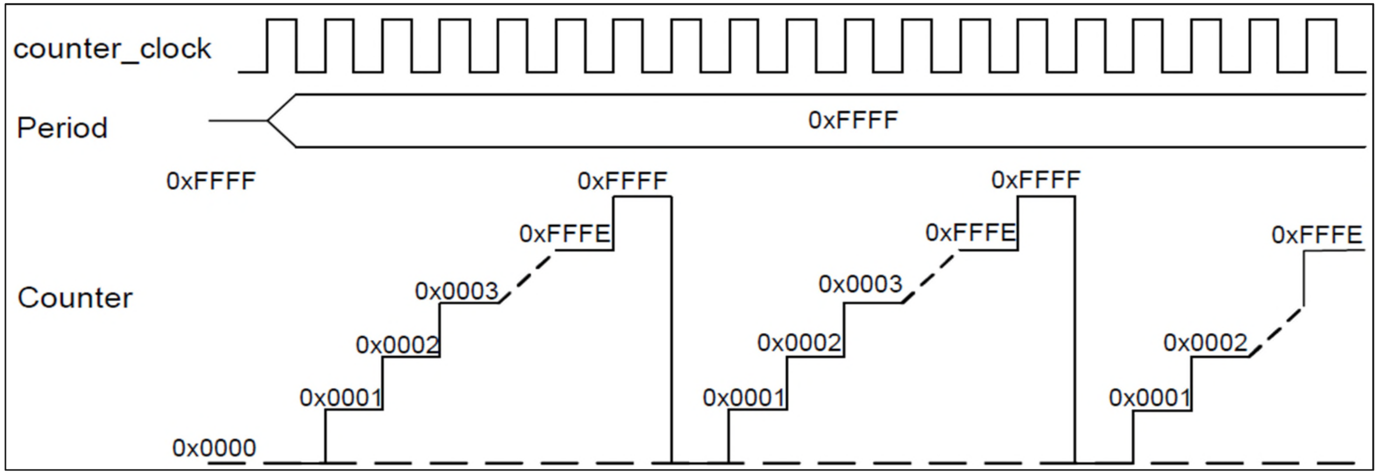
If each clock pulse is of period 0.5 S, we may say that 5 x 0.5 S = 2.5 S have lapsed

General architecture of hardware timer



Hardware timer as a simple timer

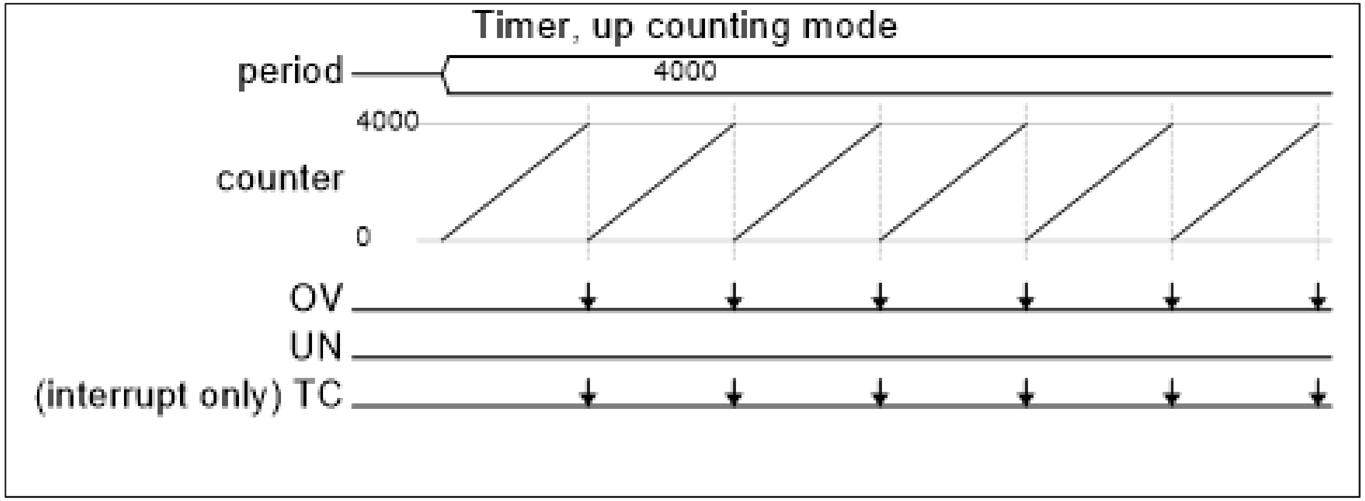




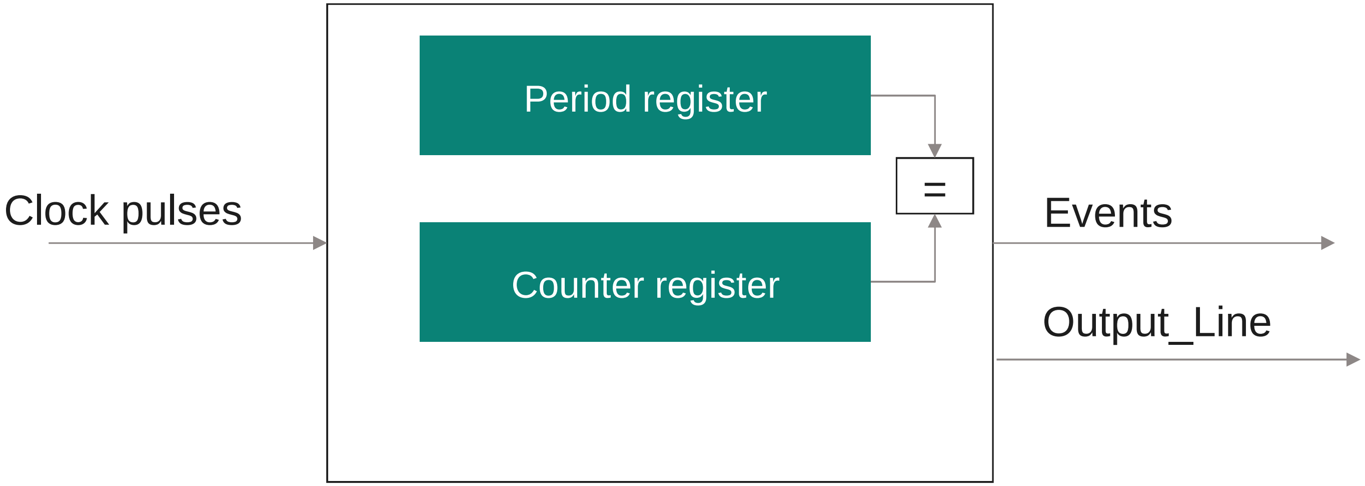
Timer counting up from 0 to terminal count

Example:

1. Your application has a need to wait for exactly 2 Sec before doing something else.
2. You know that the clock to the timer hardware is 2000 Hz and therefore the clock period is 0.5 milli-second
3. You program the Period register with a value of 4000 (Terminal Count Value) and START the timer
4. The counter register which was initialized to 0 increments by 1 for every clock pulse
5. After reaching a value of 4000, the Counter and Period registers are equal.
6. Timer hardware generates an event. Before starting the timer, you configured this event to interrupt the CPU
7. If the Interrupt Service Routine executes, it will mean that the wait time of 2 Sec has lapsed.



Hardware timer as a code profiler



You want to find out how long does a piece of software take to execute.

For example, you wrote FFT algorithm and now you want to know how long it takes to fully execute.

1. Load the Period register with the highest value (e.g. 0xFFFFFF).
2. Start the timer.
3. Run the FFT code
4. Now stop the timer and make a note of the value in the counter register
5. Let us assume that you read out a value of 12780 from the counter register.
6. If the clock frequency was 2000 Hz, then the 12780 clock pulses total to 6390 ms or 6.39 Sec

Applications of Timer/counter

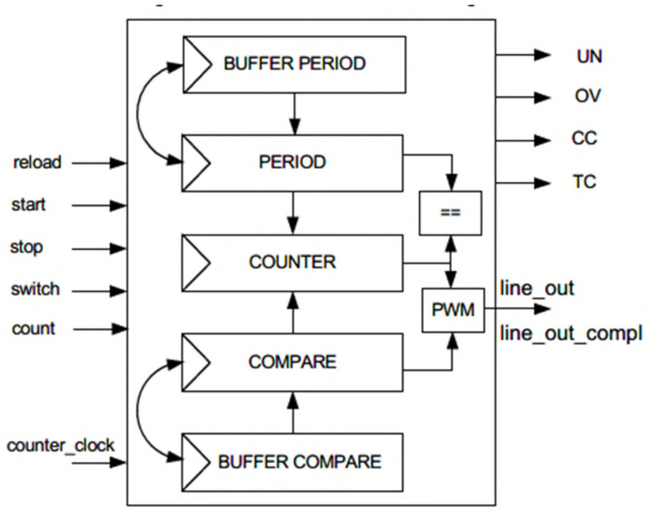
Precision Timing Operations	Timer modules ensure accurate time management for various applications, enhancing system reliability.
Delay Generation	They generate precise delays required for synchronizing processes within microcontroller applications.
Event Counting	They can count specific events, enabling monitoring and control of various system parameters.
Input Capture	Timers can capture the timestamp of an external event. This is useful for measuring the time interval between events, such as the period of a waveform or the duration of a pulse.
Output Compare	Timers can be configured to generate an output signal when the timer reaches a specific count value. This can be used to generate precise periodic signals, trigger interrupts, or control timing for other tasks.
Frequency Measurement	Timers can be used to measure the frequency of an input signal by counting the number of events in a fixed time period. This is useful in applications like frequency counters and signal analysis.
Watchdog Timer	A special type of timer, the watchdog timer, is used to reset the microcontroller if the software fails to operate correctly. This helps in recovering from software malfunctions and ensures system reliability
Task Scheduling	In real-time operating systems (RTOS), timers are used to schedule tasks to run at specific intervals, manage time slices in multitasking systems, and ensure that time-critical tasks are executed as needed.

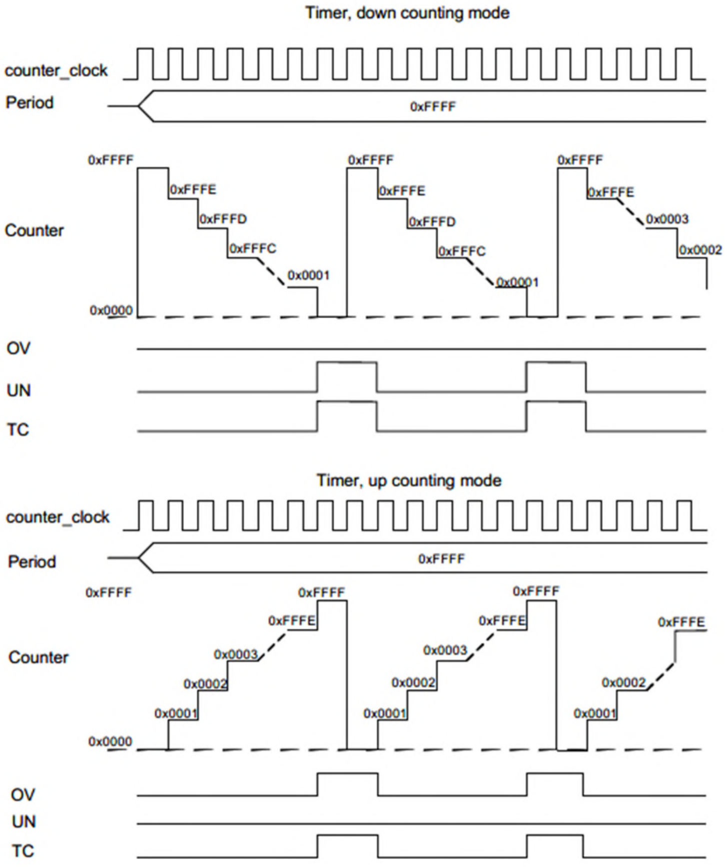
Introduction to Timer module - TCPWM in PSoC

The block can be used to measure the

- Period and pulse width of an input signal (Timer),
- find the number of times a particular event occurs (Counter),
- generate PWM signals

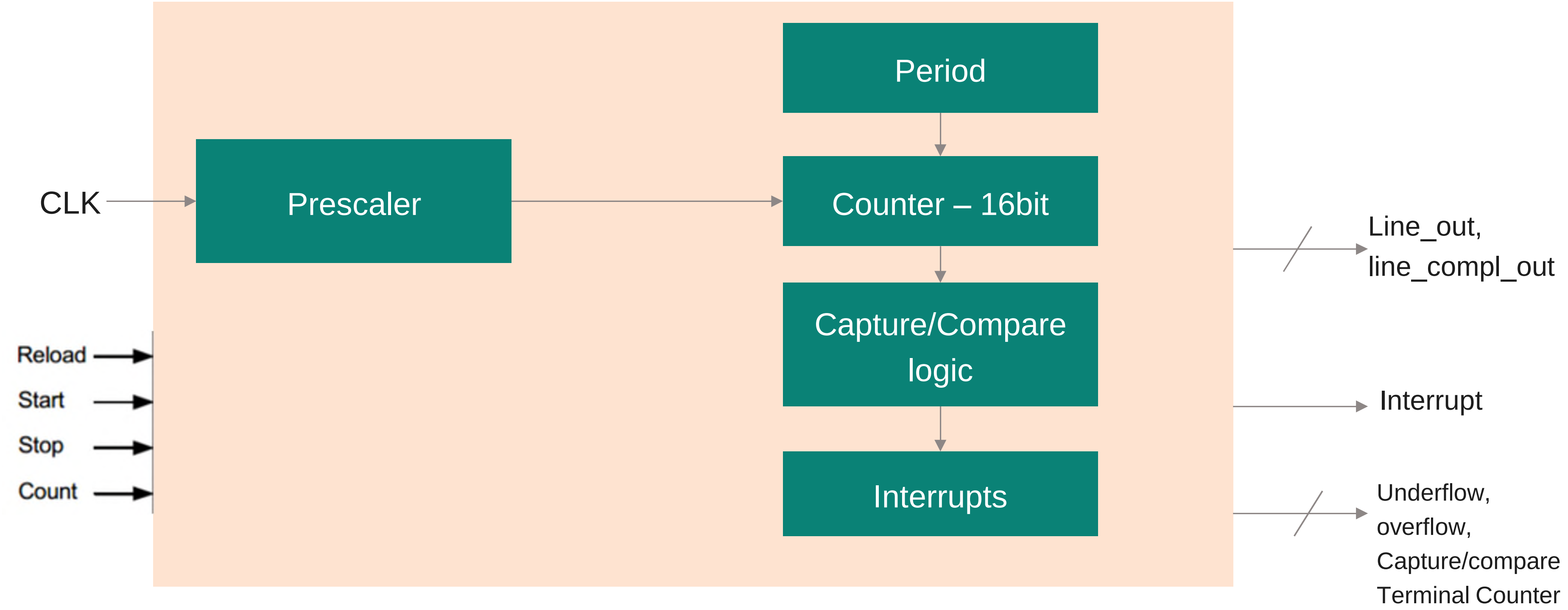
PSoC Timers


BUFFER PERIOD
PERIOD
COUNTER
COMPARE
BUFFER COMPARE
==
PWM
reload
start
stop
switch
count
counter_clock
UN
OV
CC
TC
line_out
line_out_compl

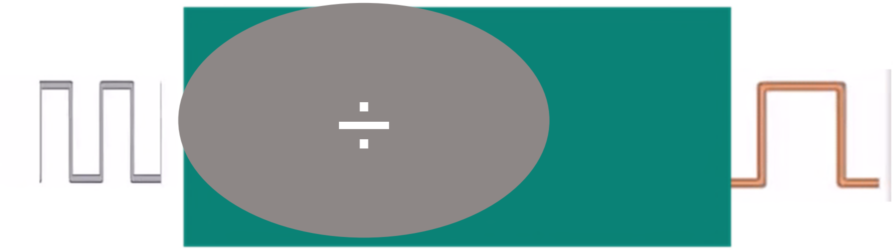

Timer, down counting mode
counter_clock
Period
0xFFFF
Counter
0xFFFF
0xFFFE
0xFFFD
0xFFFC
0x0001
0x0000
0x0003
0x0002
OV
UN
TC
Timer, up counting mode
counter_clock
Period
0xFFFF
Counter
0xFFFF
0xFFFE
0x0003
0x0002
0x0001
0x0000
OV
UN
TC

Key Components of Timer module

- Prescaler - Divides the input clock frequency to extend the timer range.
- Counter - Registers the number of clock cycles for accurate timing.
- Capture/Compare Units - Captures external events or compares timer values to trigger actions.
- Interrupts - Allow timers to signal the CPU asynchronously for timely responses.



Pre-scalar

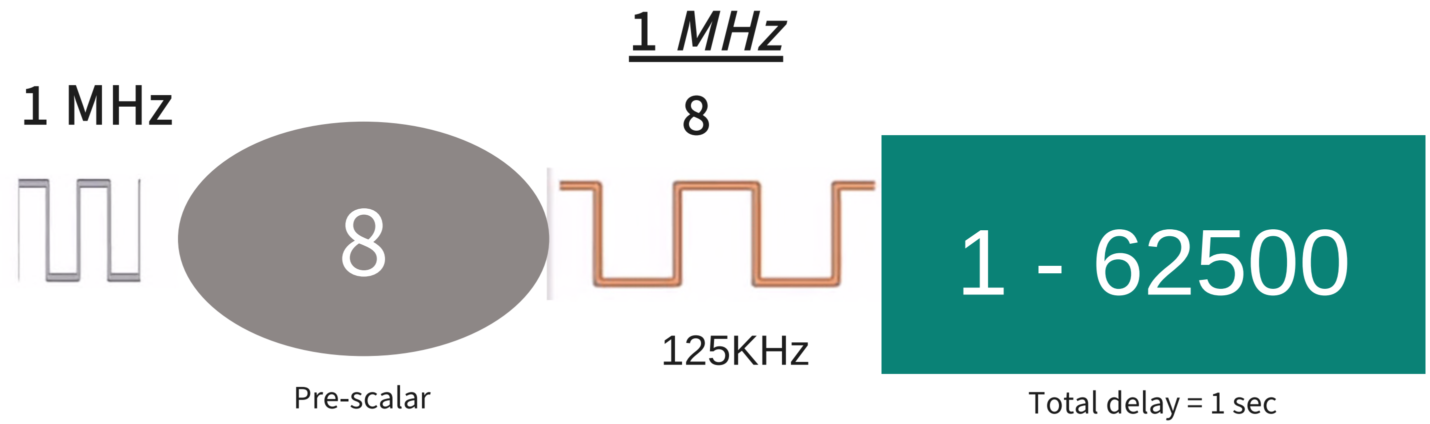


Pre-scalar – 1,2,4….64,128

2^16…16bit

Motivation for pre-scalar

- If you need to measure lot of time (more seconds) with an 8-bit timer that overflows at 256, you can use a pre-scaler to give your timer a slow clock for desired requirement.
- For shorter (8 and 16-bit) timers, there will often be a tradeoff between resolution (high resolution requires a high clock rate) and range (high clock rates cause the timer to overflow more quickly).



1 MHz

8

Pre-scalar

$\frac{1\ MHz}{8}$

125KHz

1 - 62500

Total delay = 1 sec

Without Pre-scaler:

Delay = Count * (1/1MHz)

where Count = 62500

Delay = 0.0625

Resolution = 1/1MHz = 1uS

With 8 as Pre-scaler:

Delay = Count * (8/1MHz)

where: Count = 62500

Delay = 0.5seconds

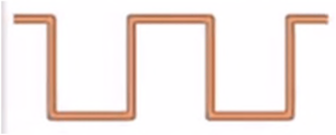
Resolution = 1/125KHz = 8uS

What about Resolution? Will it change with Pre-scalar?

COUNTER

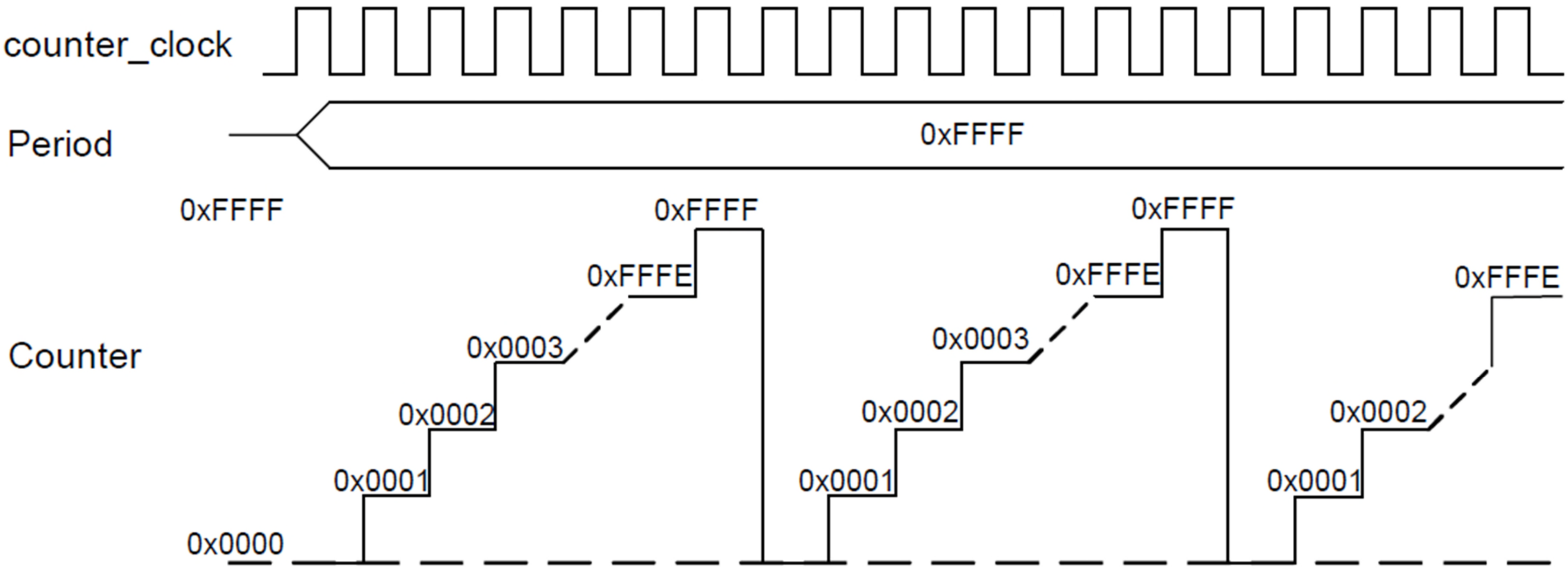
Modes of counter

Up Counter



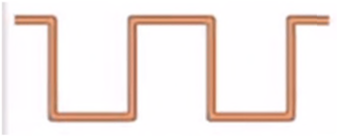
Period = 9

When Count = Period value it resets to 0


counter_clock
Period
0xFFFF
0xFFFF
0xFFFF
0xFFFF
0xFFFE
0xFFFE
0xFFFE
0x0003
0x0003
0x0002
0x0002
0x0002
0x0001
0x0001
0x0001
0x0000
Counter

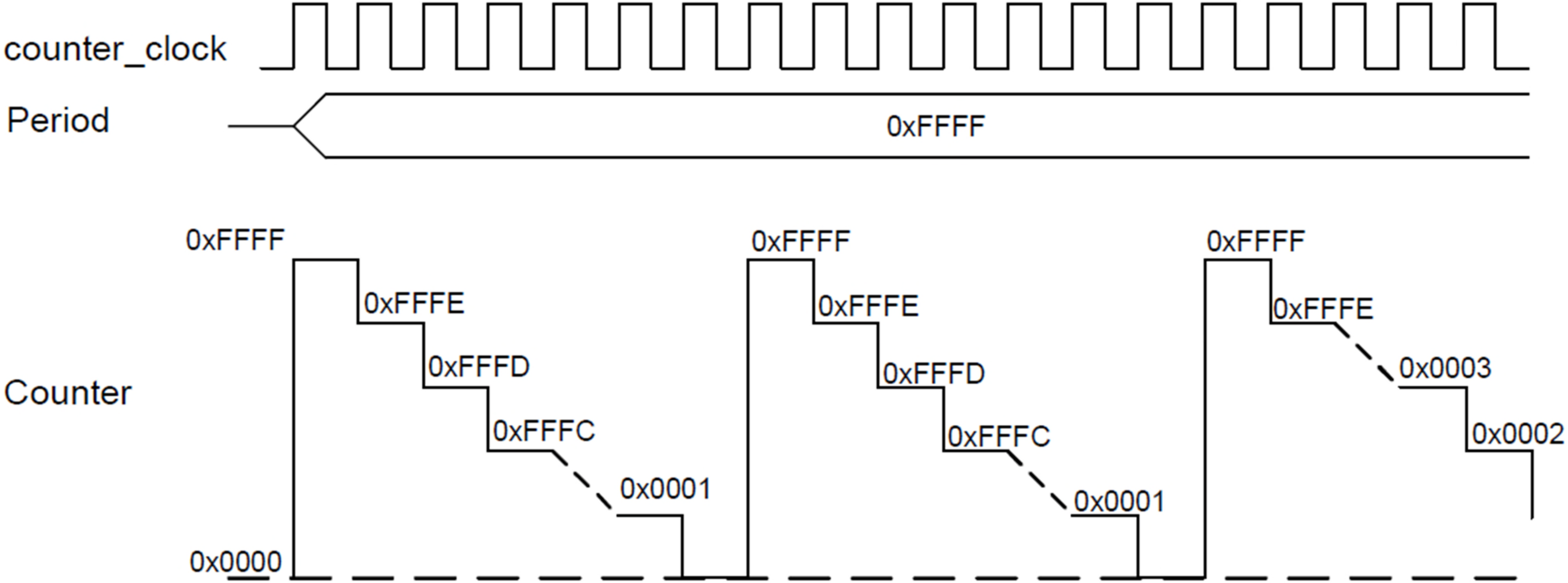
Modes of counter

Down Counter



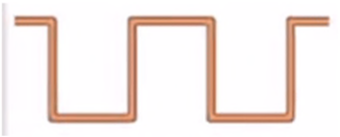
Period = 9

When Count = 0 value it resets to Period value


counter_clock
Period
0xFFFF
0xFFFF
0xFFFE
0xFFFD
0xFFFC
0x0001
0x0000
Counter
0xFFFF
0xFFFE
0xFFFD
0xFFFC
0x0001
0xFFFF
0xFFFE
0x0003
0x0002

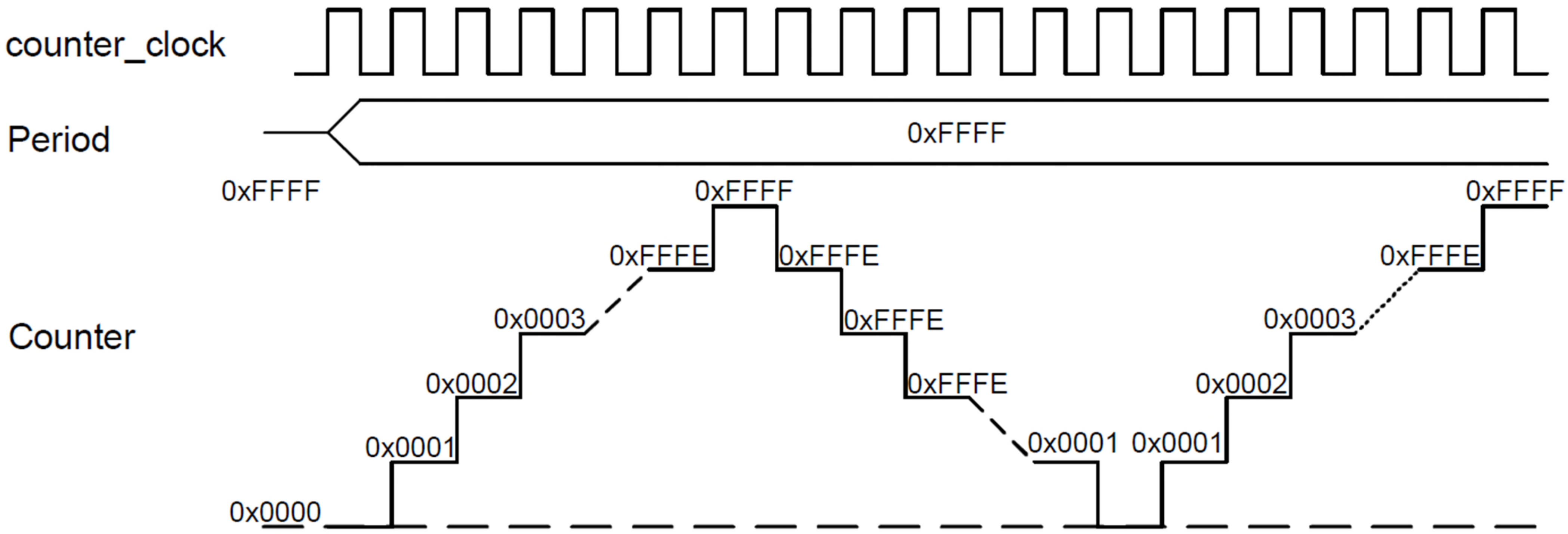
Modes of counter

Up/Down Counter



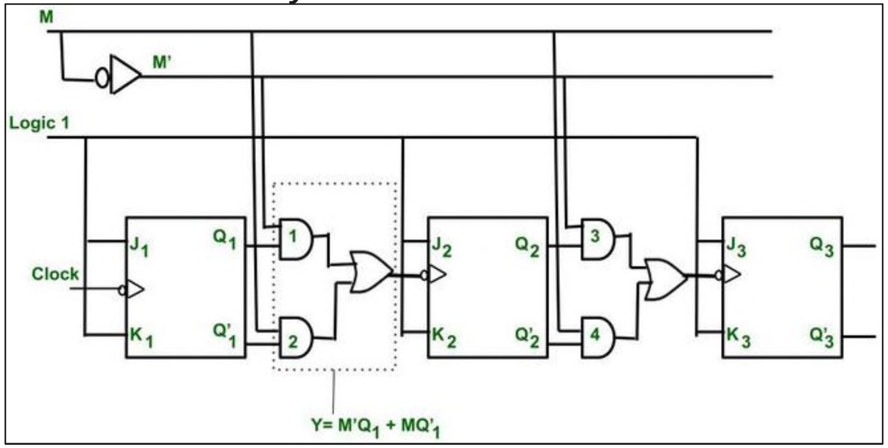
Period = 9

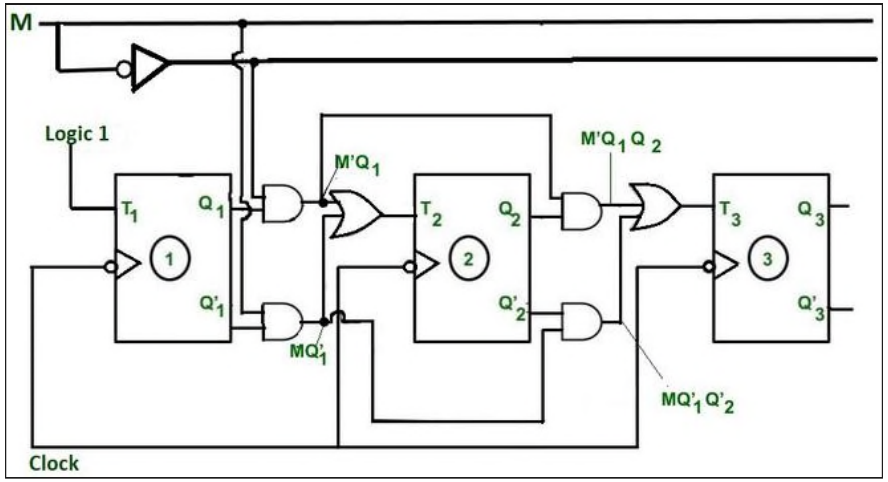
When Count = Period value it decrements to 0 and then again it starts incrementing till it reaches Period values and continues like this.


counter_clock
Period
0xFFFF
Counter
0xFFFF
0xFFFF
0xFFFF
0xFFFE
0xFFFE
0xFFFE
0xFFFE
0xFFFE
0x0003
0x0003
0x0002
0x0002
0x0001
0x0001
0x0001
0x0000

Example Counter Implementation:
FLIP FLOP UP/DOWN COUNTER (JK for Async, T for Sync)

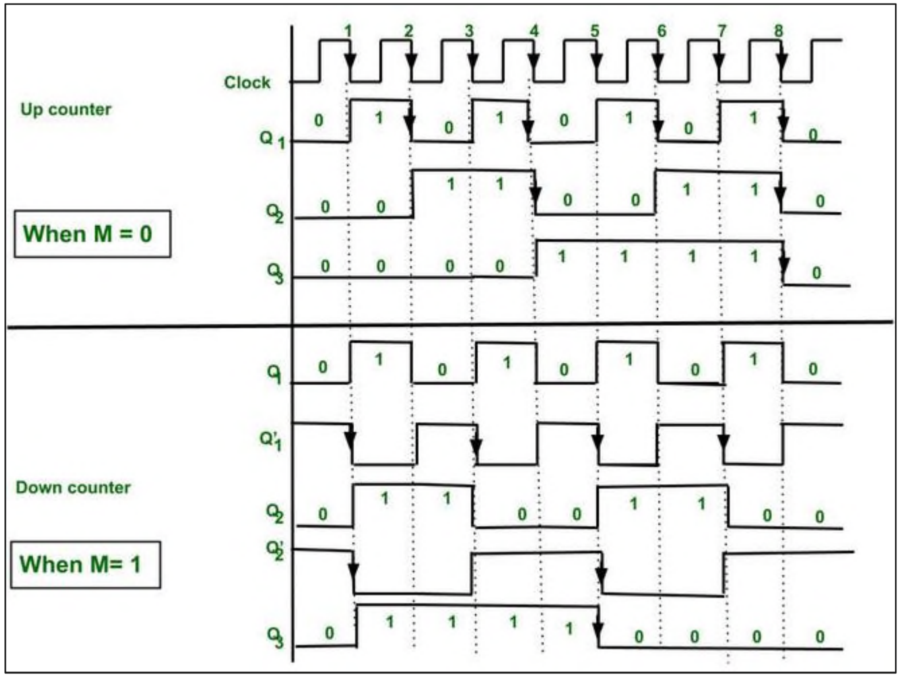
Asynchronous





synchronous

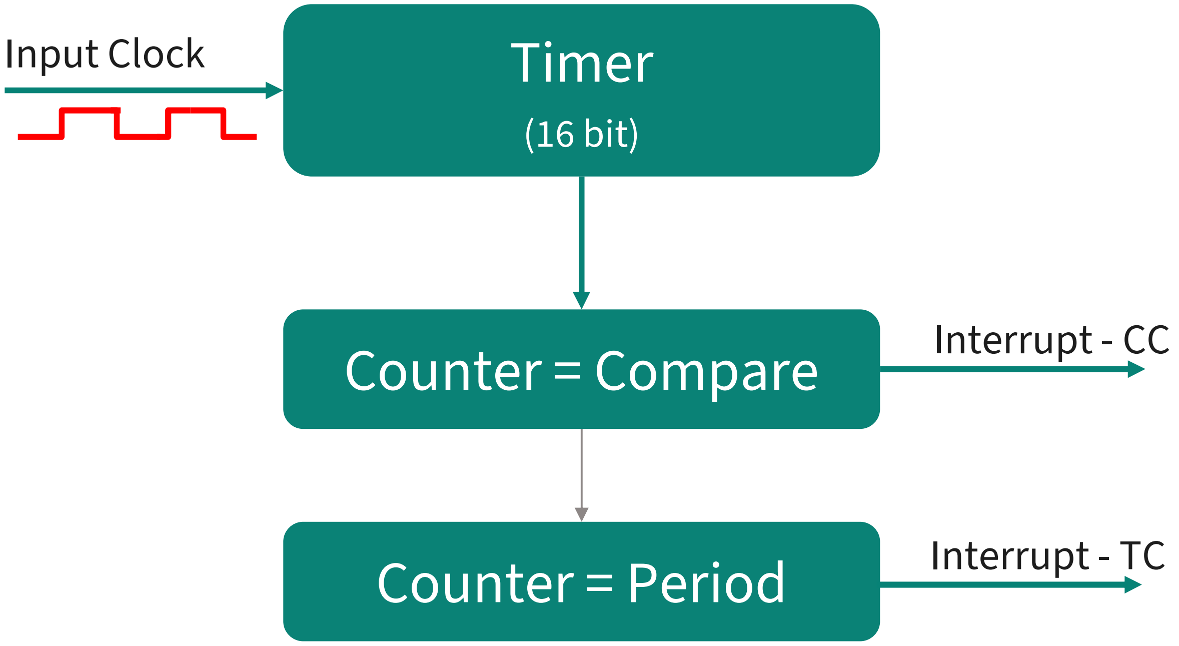
Timing Diagram

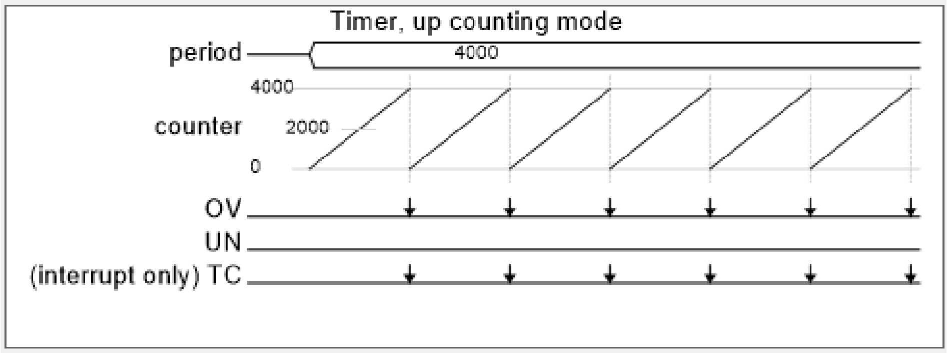


There are two types of counter mode

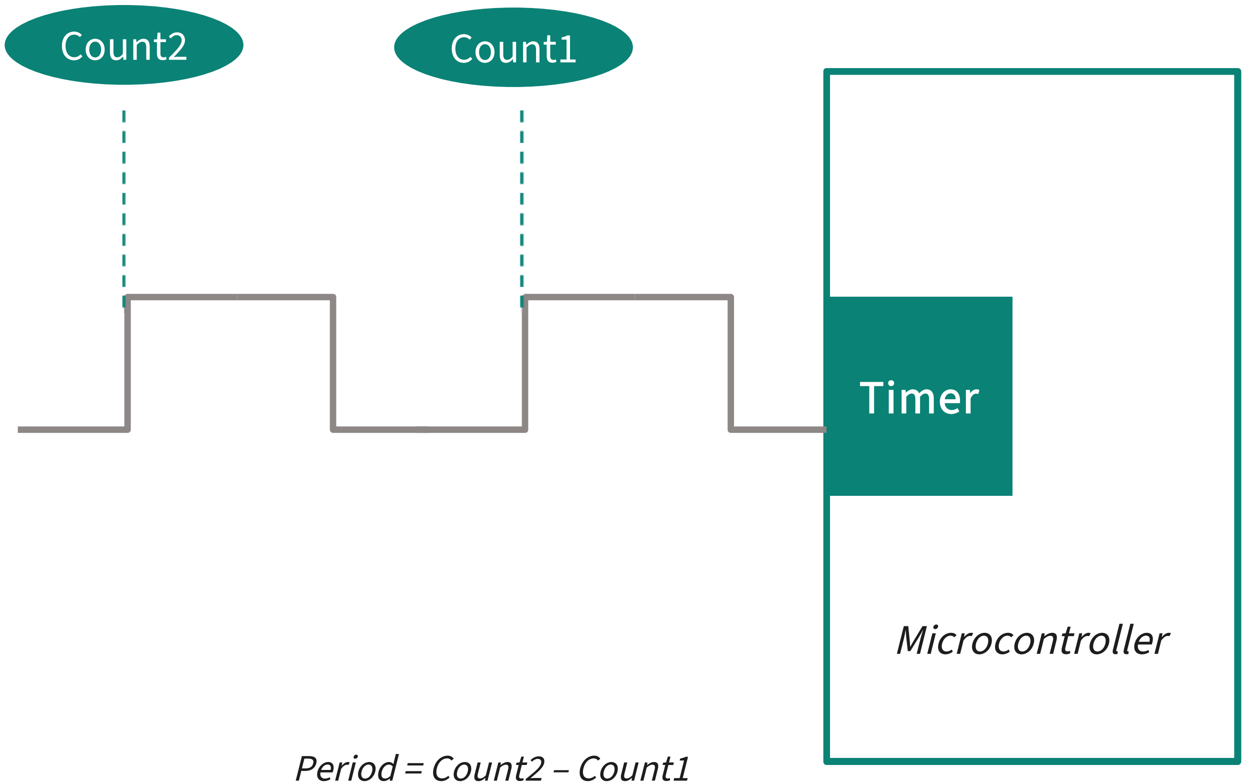
- Compare Mode
- Capture Mode

Compare mode

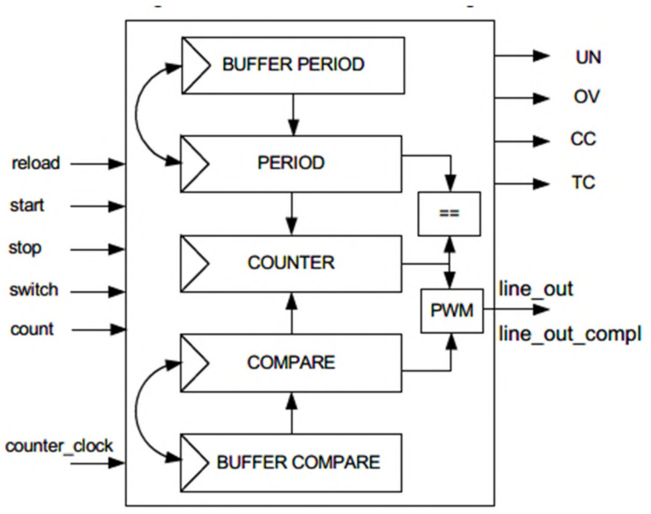




Capture mode

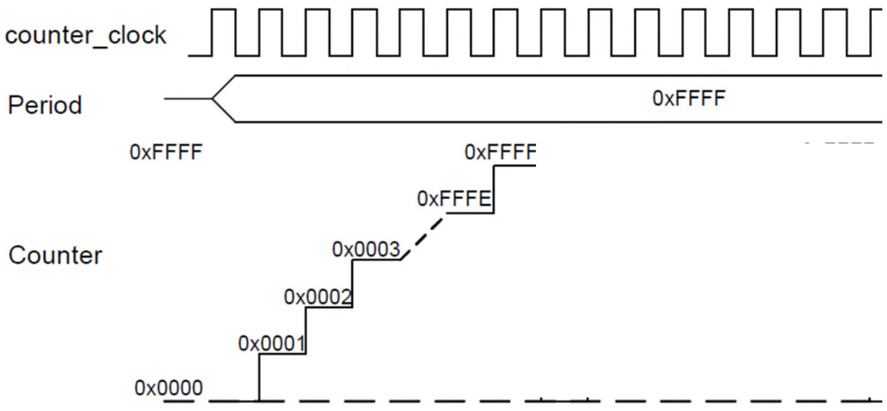


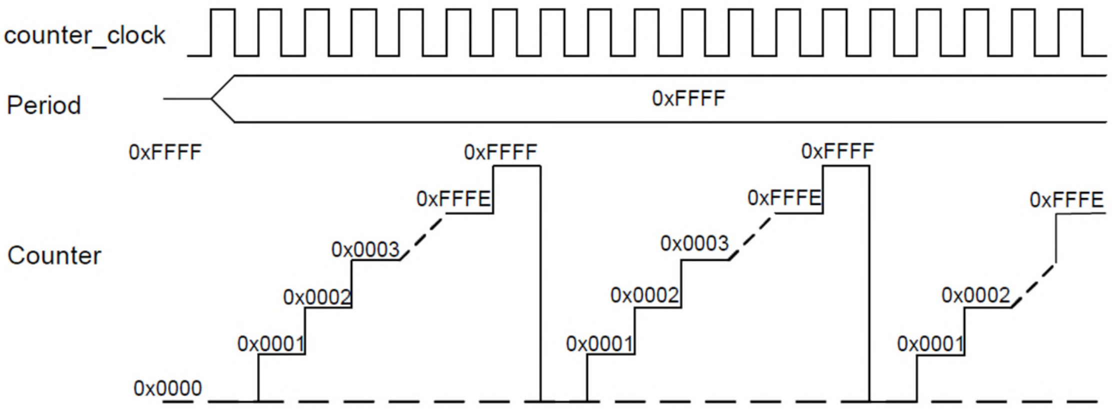
Period = Count2 – Count1



Hardware timer: Single shot vs Continuous modes

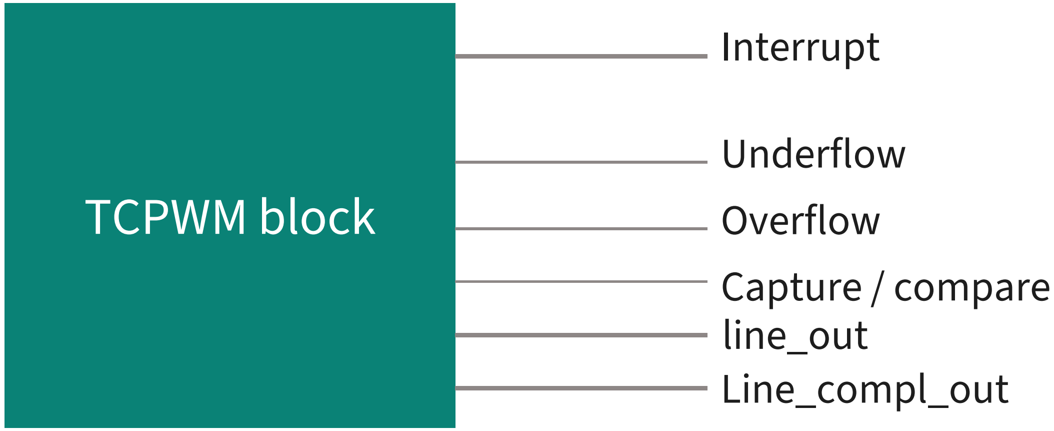
- A timer can count up to the PERIOD register value and two things can happen now
  - The counter can reset to 0 and start incrementing again [Up-counting, Continuous Mode]
  - The Counter can reload itself with value from PERIOD register and start decrementing again [Down-Counting, Continuous mode]
  - Or the timer simply stops [Single shot mode]





TCPWM output signals

Counter generates an **internal underflow** (UN) condition when counting down and the count register reaches zero.



TCPWM block

Interrupt

Underflow

Overflow

Capture / compare

line_out

Line_compl_out

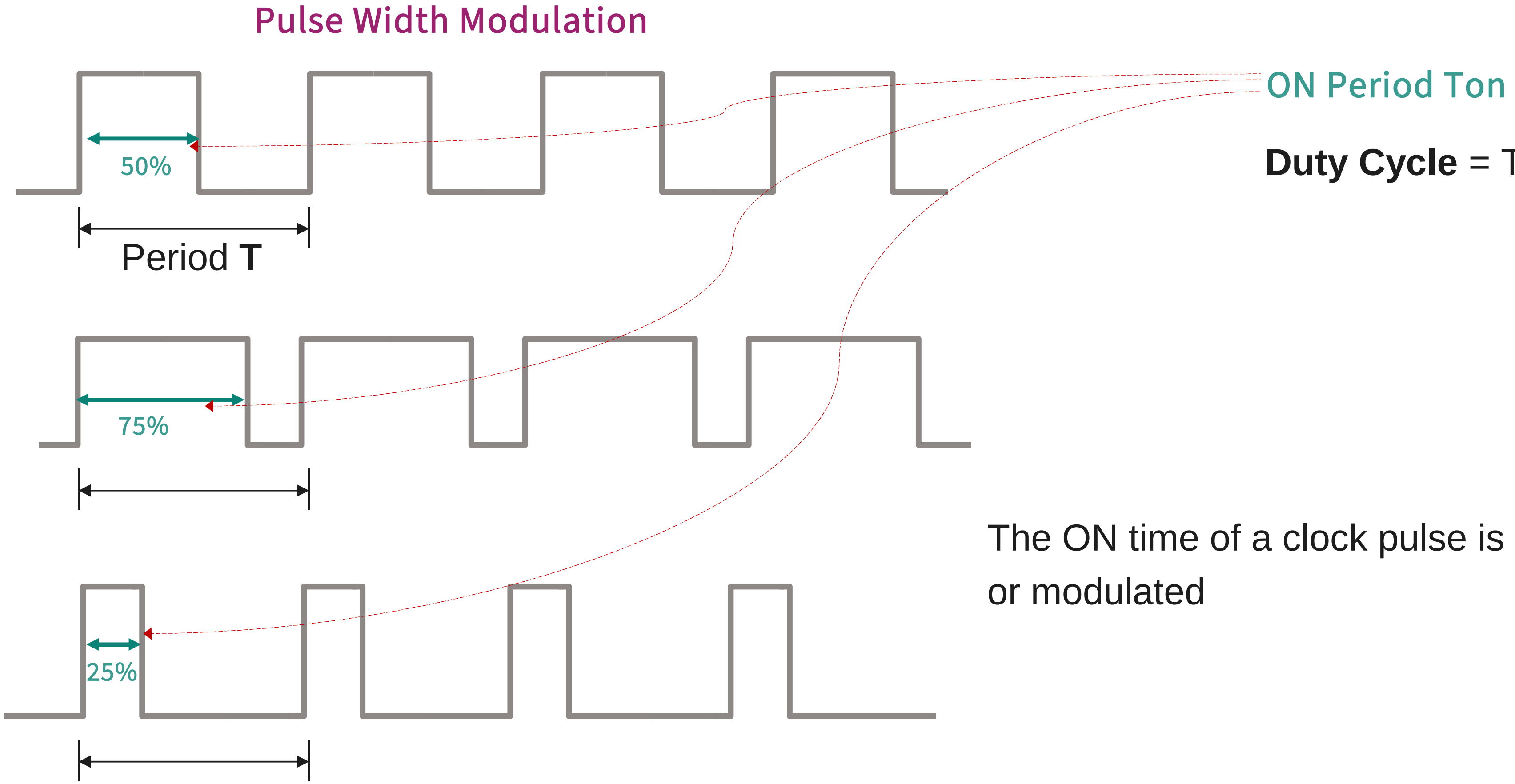
Counter generates an **internal overflow** (OV) condition when counting up and the count register reaches the period value.

The **capture/compare** (CC) condition is generated by the TCPWM when the counter is running and one of the following conditions occur:

- The counter value equals the compare value.
- When a capture event occurs

PSoC 4100S Plus have 8 16-bit TCPWM blocks

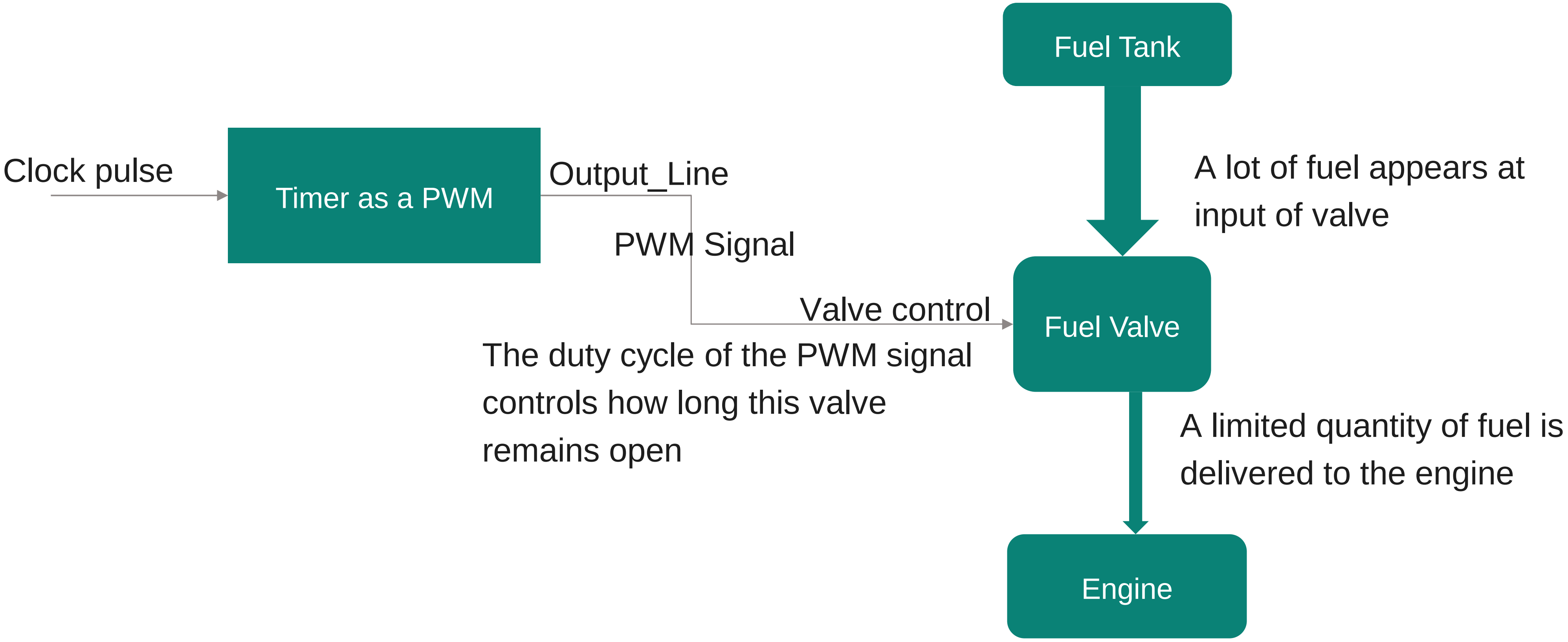
Hardware timer as PWM (Pulse Width Modulator)



Duty Cycle = Ton / T

The ON time of a clock pulse is being changed or modulated

Hardware timer as PWM (Pulse Width Modulator) – Example 1



Clock pulse

Timer as a PWM

Output_Line

PWM Signal

Valve control

The duty cycle of the PWM signal controls how long this valve remains open

Fuel Tank

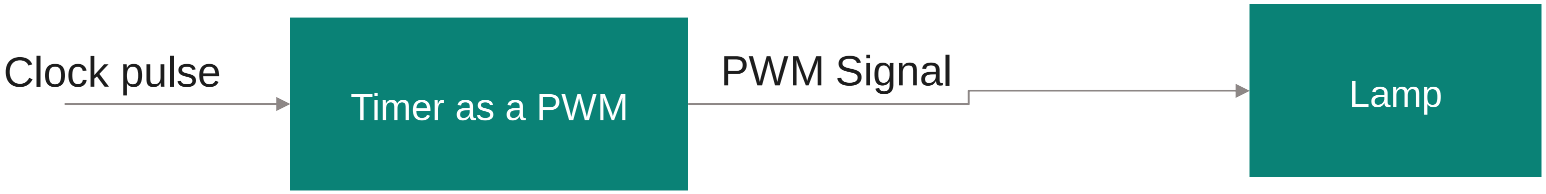
A lot of fuel appears at input of valve

Fuel Valve

A limited quantity of fuel is delivered to the engine

Engine

Hardware timer as PWM (Pulse Width Modulator) – Example 2



The duty cycle of the PWM signal controls how bright can this lamp burn

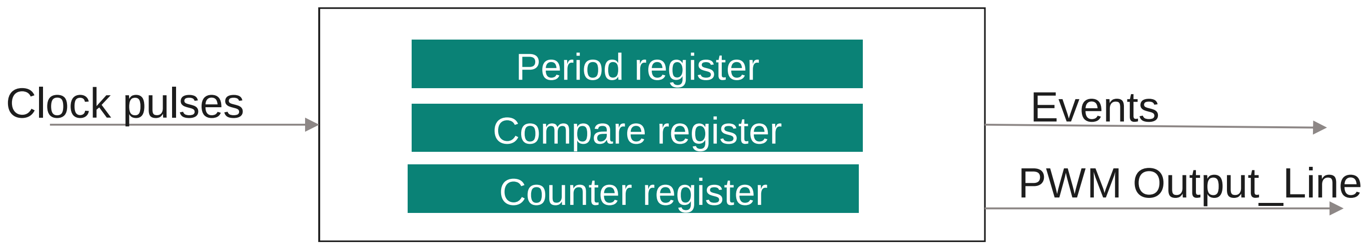
Example:

80% Duty Cycle = Bright

40% Duty Cycle = Normal

15% Duty Cycle = Dim

Hardware timer as Pulse Width Modulator



1. You want to generate a 1 KHz square wave with a duty cycle of 40%.

   Period of 1 KHz square wave = 1 ms

2. Clock frequency of the Timer is 100 KHz (Clock Period: 10 us)

   So Period register is loaded with a value of **100** (10 us x **100** = 1 ms)

   40% of 100 = 40

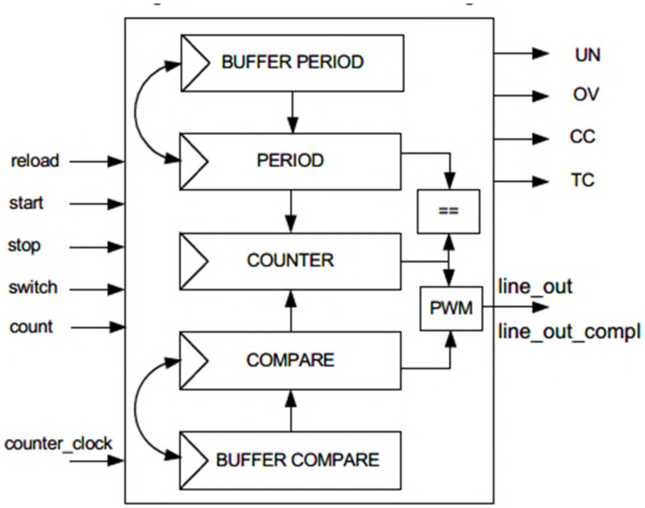
   So Compare register is loaded with a value of 40

3. Start the timer
4. Counter starts counting from 0.
5. PWM Output remains 0 until as long as Counter register is less than Compare register
6. Once Counter register equals Compare register, PWM Output flips to HIGH
7. Once Counter register equals Period register, PWM Output resets to 0 and the story continues

How to change PERIOD and COMPARE registers

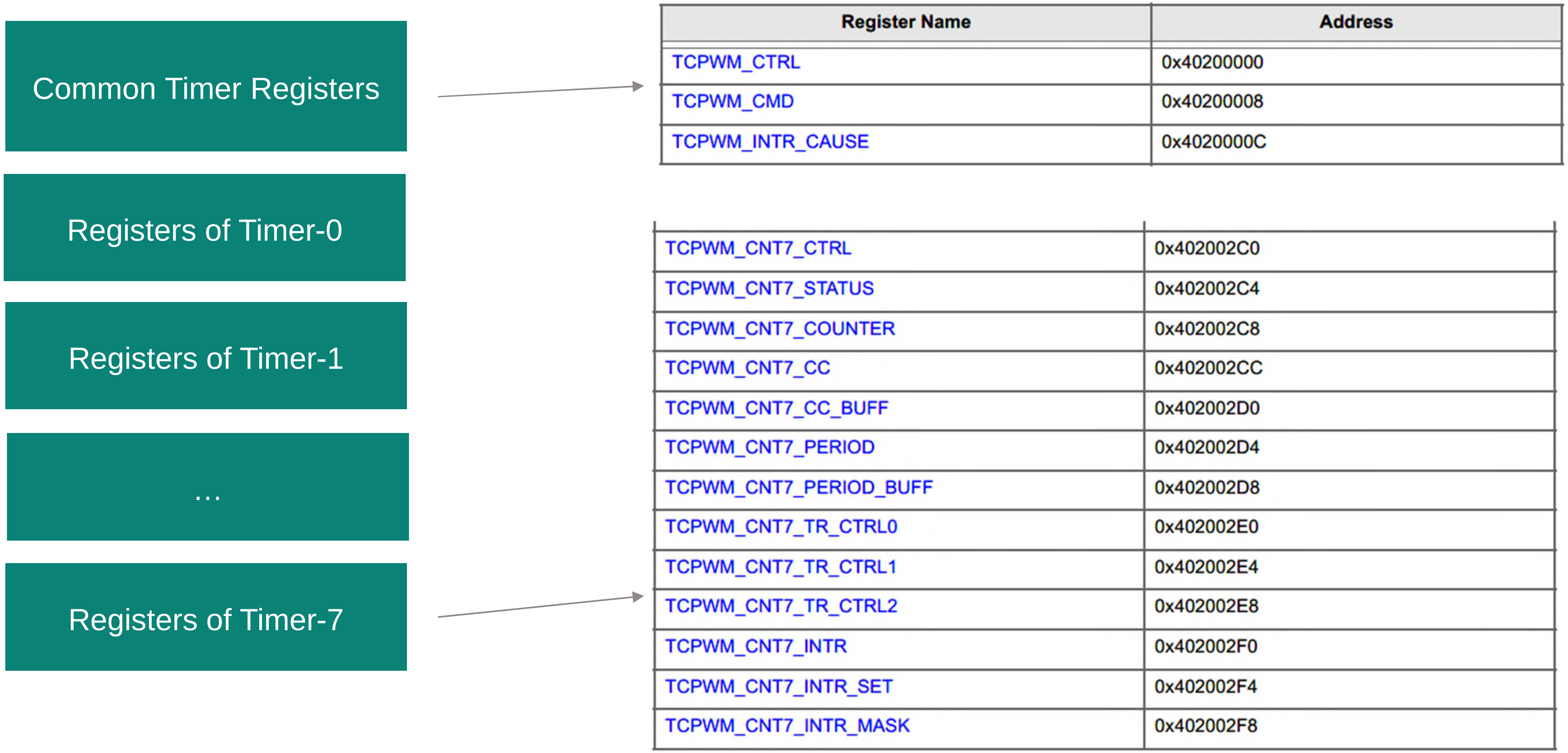
- If your timer has stopped, you may directly program PERIOD and COMPARE registers

- If the timer is already running with a certain value programmed in PERIOD and COMPARE register
  - Program PERIOD buffer register and COMPARE buffer register
  - Values are automatically transferred from buffer registers to actual registers only after the timer has finished counting up to the value currently programmed in the PERIOD register – Useful in Continuous mode of operation



Register configuration

PSoC timer registers



Common Timer Registers

Registers of Timer-0

Registers of Timer-1

…

Registers of Timer-7

Register Name	Address
TCPWM_CTRL	0x40200000
TCPWM_CMD	0x40200008
TCPWM_INTR_CAUSE	0x4020000C

TCPWM_CNT7_CTRL	0x402002C0
TCPWM_CNT7_STATUS	0x402002C4
TCPWM_CNT7_COUNTER	0x402002C8
TCPWM_CNT7_CC	0x402002CC
TCPWM_CNT7_CC_BUFF	0x402002D0
TCPWM_CNT7_PERIOD	0x402002D4
TCPWM_CNT7_PERIOD_BUFF	0x402002D8
TCPWM_CNT7_TR_CTRL0	0x402002E0
TCPWM_CNT7_TR_CTRL1	0x402002E4
TCPWM_CNT7_TR_CTRL2	0x402002E8
TCPWM_CNT7_INTR	0x402002F0
TCPWM_CNT7_INTR_SET	0x402002F4
TCPWM_CNT7_INTR_MASK	0x402002F8

Understanding PSoC Timer registers – 1/2

Common registers

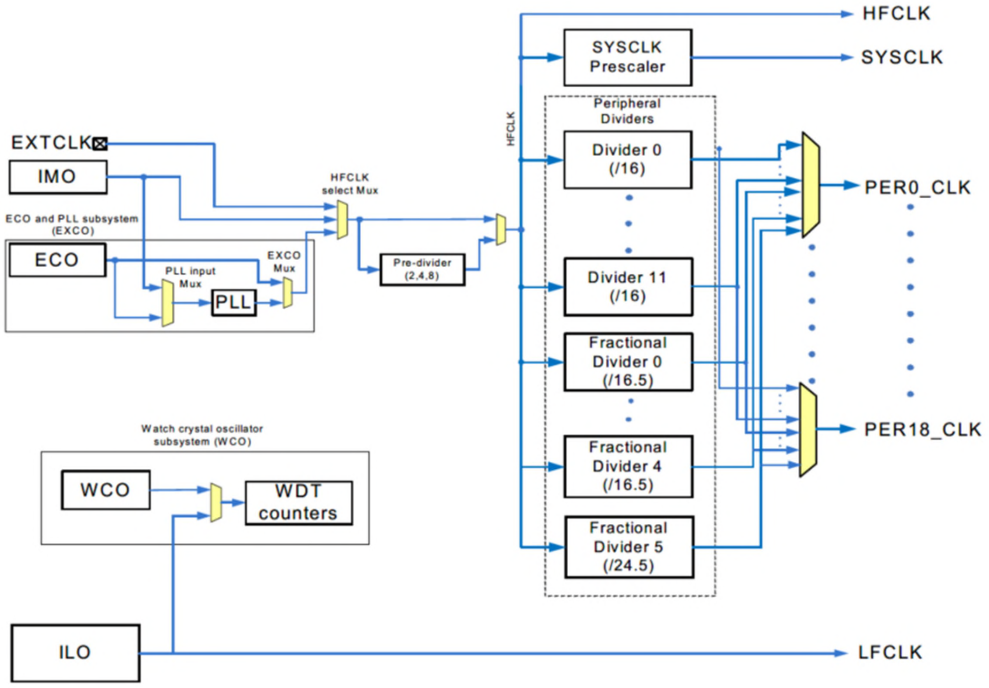
Register Name	Description
TCPWM_CTRL	Contains one Enable bit for each of the 8 timers. Your chosen timer will simply not work without activating the corresponding Enable bit here in this register
TCPWM_CMD	Contains the activation bits for generating control signals (START, STOP, RELOAD) to your chosen timer “by software”. If you prefer that other peripherals generate these control signals instead of software, use the timer specific register set
TCPWM_INTR_CAUSE	Contains one interrupt status bit for each of the 8 timers. If the bit corresponding to your chosen timer is found to be a 1 here, it means that your timer has generated an event (overflow, underflow, compare match, period match etc.) and that event has generated an interrupt to the CPU

Understanding PSoC Timer registers – 2/2

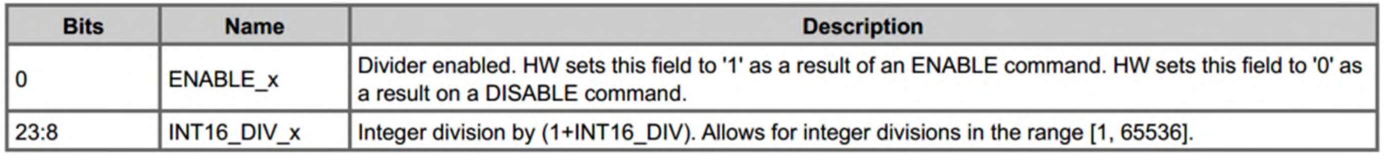
Timer specific registers

Register Name	Description
TCPWM_CNTx_CTRL	Contains bits that define operational mode of the timer (Timer/PWM mode, One-shot/Continuous, Up/Down/Up-Down mode, Reload from Shadow register
TCPWM_CNTx_STATUS	1 bit to indicate whether the timer is running or not
TCPWM_CNTx_CC, TCPWM_CNTx_PERIOD, TCPWM_CNTx_COUNTER	COMPARE, PERIOD and COUNTER registers of this timer. Their shadow registers are TCPWM_CNTx_CC_Buff and TCPWM_CNTx_PERIOD_Buff
TCPWM_CNTx_CTRL0	Remember that the control signals (START, STOP and RELOAD) can be generated by signals from 16 other peripherals. You can select for each of the three control signals a preferred peripheral from the list of 16.
TCPWM_CNTx_CTRL1	After choosing the source of a control signal, you can configure how your chosen signal can generate the control signal [Rising edge/Falling edge/Both edges]
TCPWM_CNTx_CTRL2	Bits to determine how hould Line_Output signal be modified. What should happen to Line_Out when a COMAPARE match event is detected? What should happen to Line_Out when a PERIOD match event is detected?
TCPWM_CNTx_INTRyy	Interrupt management registers. You can find out whether or not is an event active, you can even clear the active event or even mask an event from happening if you prefer!

Clocking system in PSoC



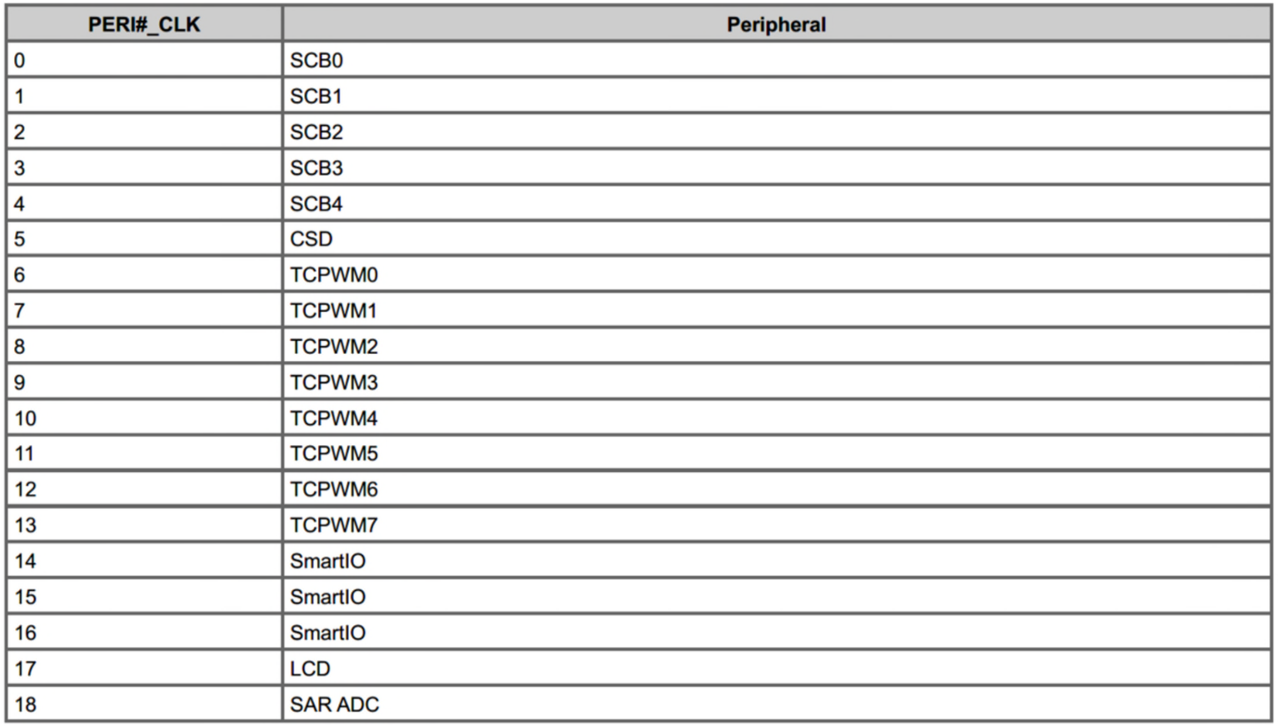
Peripheral Clock Divider Configuration Register PERI_DIV_16_CTLx



Bits	Name	Description
0	ENABLE_x	Divider enabled. HW sets this field to '1' as a result of an ENABLE command. HW sets this field to '0' as a result on a DISABLE command.
23:8	INT16_DIV_x	Integer division by (1+INT16_DIV). Allows for integer divisions in the range [1, 65536].

Clocking system in PSoC

```
//TIMER 2 TIMER- CLOCK
SysClk_PeriphDisableDivider(CY_SYSCLK_DIV_16_BIT, 3U);
SysClk_PeriphSetDivider(CY_SYSCLK_DIV_16_BIT, 3U, 24000 - 1);
SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_16_BIT, 3U);
SysClk_PeriphAssignDivider(PCLK_TCPWM_CLOCKS2, CY_SYSCLK_DIV_16_BIT, 3U);
```



PERI#_CLK	Peripheral
0	SCB0
1	SCB1
2	SCB2
3	SCB3
4	SCB4
5	CSD
6	TCPWM0
7	TCPWM1
8	TCPWM2
9	TCPWM3
10	TCPWM4
11	TCPWM5
12	TCPWM6
13	TCPWM7
14	SmartIO
15	SmartIO
16	SmartIO
17	LCD
18	SAR ADC

Infineon