EE 186 – Introduction to Embedded Systems

Lecture 5

Zerina Kapetanovic

Announcements

- Project idea submission due **TODAY at 10 PM**,
- Project selection on Thursday
  - Attendance required
  - Come eat some pizza!
- HW 2 due tomorrow (10/14), Lab 2 due on Friday (10/16)
- HW 3, Lab 3 released today
- Sign up for Lab 1-2 Discussion next Friday (10/23) (will post link on Ed & Canvas announcement)
  - https://docs.google.com/spreadsheets/d/1khJmYLUElnuXQuG3AafNfZIY93zjGVoeSeGV6i8b9p8/edit?usp=sharing

Component of the day…

Wireless Modules!

- LoRa
- Cellular
- Wi-Fi
- NFC
- Satellite
- GPS
- BLE
- RFID









Nest Interrupts / Preemption

Interrupt occurs with **higher priority** while CPU is handling another interrupt of **lower importance** (i.e. incoming request has a lower priority number)

New interrupt will interrupt the current ISR

Will push a new ISR stack on the stack

Tail Chaining

Same or lower important interrupt occurs while CPU is handling an interrupt of **same/higher priority**

New exception will enter pending state….but will be **executed before register unstacking** (saving unnecessary unstacking/stacking operations)

Late Arrivals

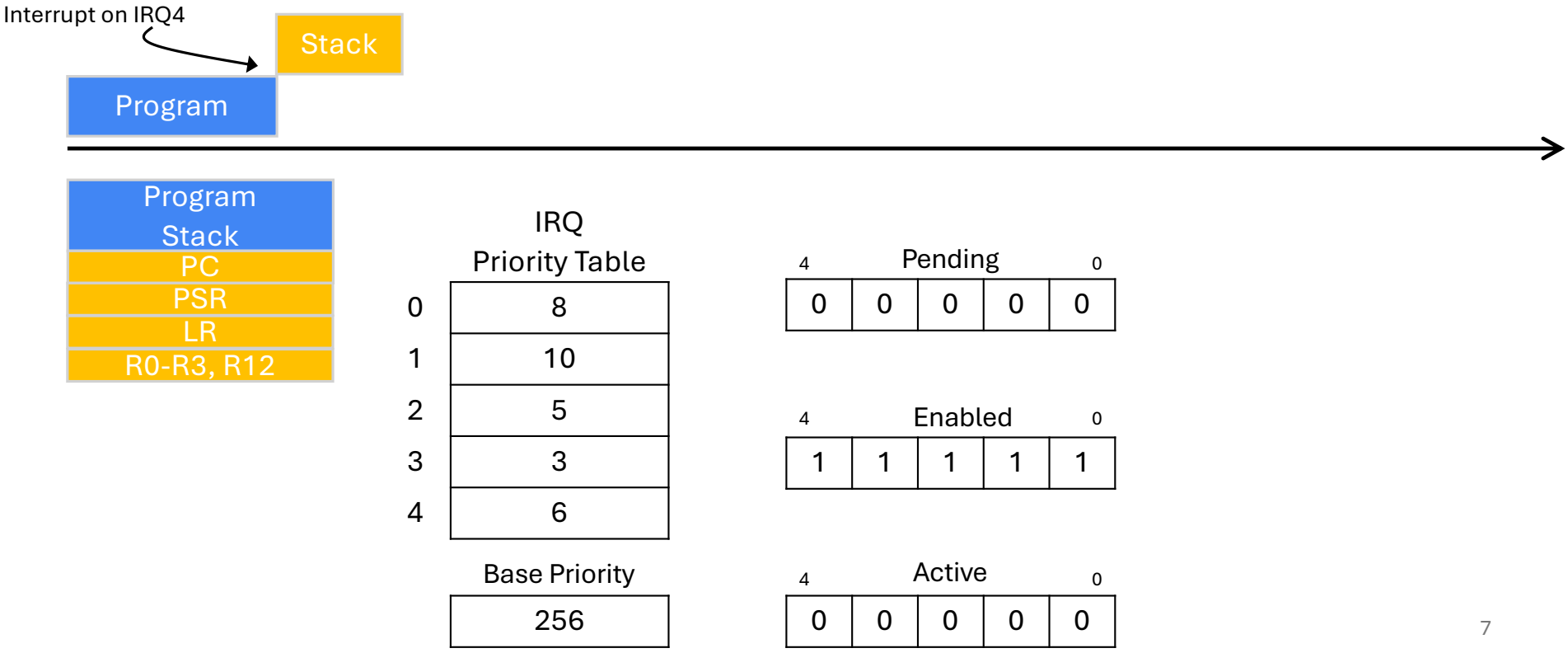
An exception occurs and stacking has started

Then another exception occurs before stacking completes

The second exception has higher preempt priority and will be processed first

Example: Late Arrivals

What happens if an interrupt occurs while you are stacking?



Example: Late Arrivals



Program Stack
PC
PSR
LR
R0-R3, R12

IRQ Priority Table

0	8
1	10
2	5
3	3
4	6

Base Priority

256

Pending

4				0
1	1	0	0	0

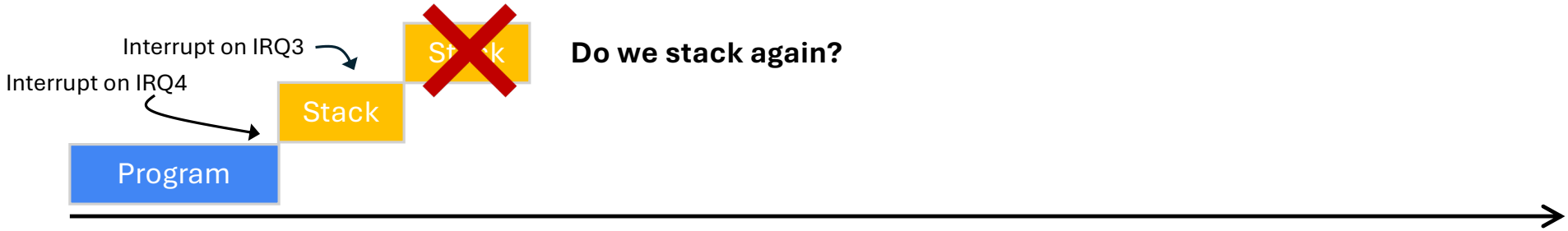
Enabled

4				0
1	1	1	1	1

Active

4				0
0	0	0	0	0

Example: Late Arrivals



Do we stack again?

Program Stack
PC
PSR
LR
R0-R3, R12

IRQ Priority Table

0	8
1	10
2	5
3	3
4	6

Base Priority

6

Pending

4				0
1	1	0	0	0

Enabled

4				0
1	1	1	1	1

Active

4				0
0	0	0	0	0

Example: Late Arrivals

Once stacking is complete, find vector for the highest priority pending



Program Stack
PC
PSR
LR
R0-R3, R12

IRQ Priority Table

0	8
1	10
2	5
3	3
4	6

Base Priority

256

Pending

4				0
1	1	0	0	0

Enabled

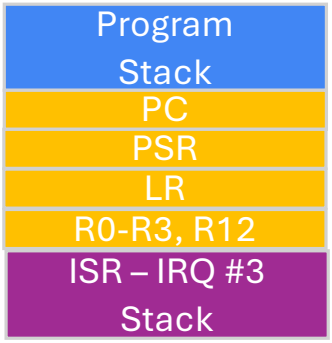
4				0
1	1	1	1	1

Active

4				0
0				

Example: Late Arrivals





IRQ Priority Table

0	8
1	10
2	5
3	3
4	6

Base Priority

3

Pending

4				0
1	0	0	0	0

Enabled

4				0
1	1	1	1	1

Active

4				0
0	1	0	0	0

Example: Late Arrivals



Program Stack
PC
PSR
LR
R0-R3, R12

IRQ Priority Table

0	8
1	10
2	5
3	3
4	6

Base Priority

3

Pending

4				0
1	0	0	0	0

Enabled

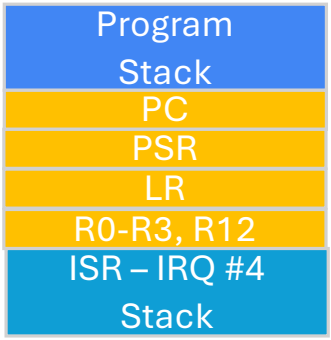
4				0
1	1	1	1	1

Active

4				0
0	1	0	0	0

Example: Late Arrivals





IRQ Priority Table

0	8
1	10
1	5
3	3
4	6

Base Priority

4

Pending

4				0
0	0	0	0	0

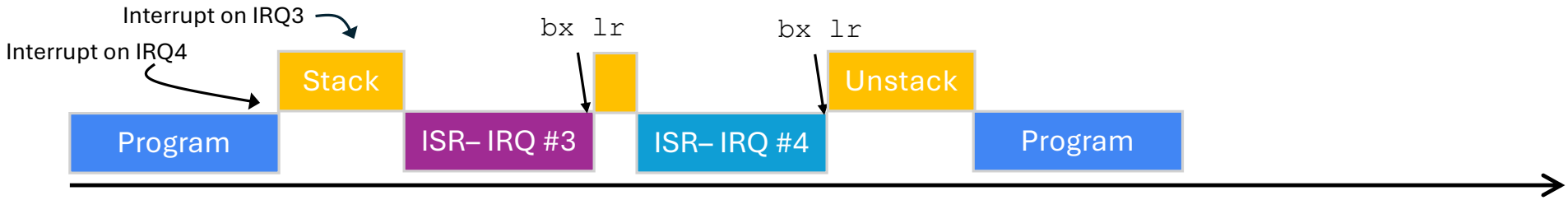
Enabled

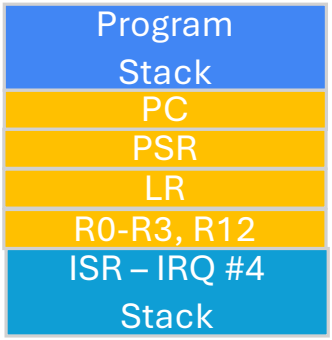
4				0
1	1	1	1	1

Active

4				0
1	0	0	0	0

Example: Late Arrivals





IRQ Priority Table

0	8
1	10
1	5
3	3
4	6

Base Priority

256

Pending

4				0
0	0	0	0	0

Enabled

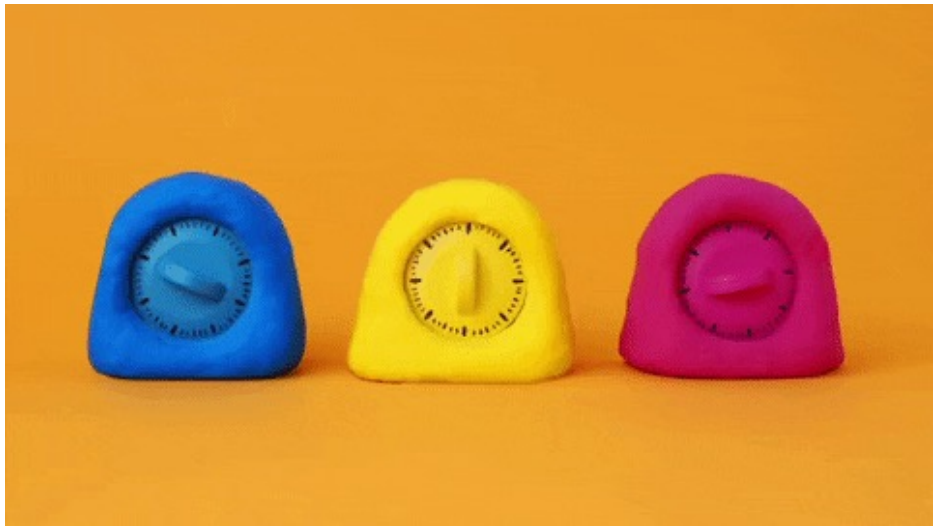
4				0
1	1	1	1	1

Active

4				0
0	0	0	0	0

Today we are talking about…

- Timers
  - Background
  - Timer Anatomy
  - Capture and Compare
- Pulse Width Modulation
- SysTick Timers



Reading

- STM32L4 Application Developers Reference Manual (RM0432)
  - Section 37: Advanced-control timers (TIM1/TIM8)
  - Section 38: General-purpose timers (TIM2/TIM3/TIM4/TIM5)

- Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C (Yifeng Zhu)
  - Chapter 15: General Purpose Timers

What is a timer?

A **timer** is a peripheral for tracking time

One of the most important features of a microcontroller!

Many use cases:

Scheduled Tasks – trigger an interrupt to read in sensor data, update a display, transmit a data packet, etc.

Generate PMW Signals – control the speed of a motor, the brightness of an LED, etc.

Event Synchronization – synchronize events internally or externally

Generate Timeouts – maximum wait time





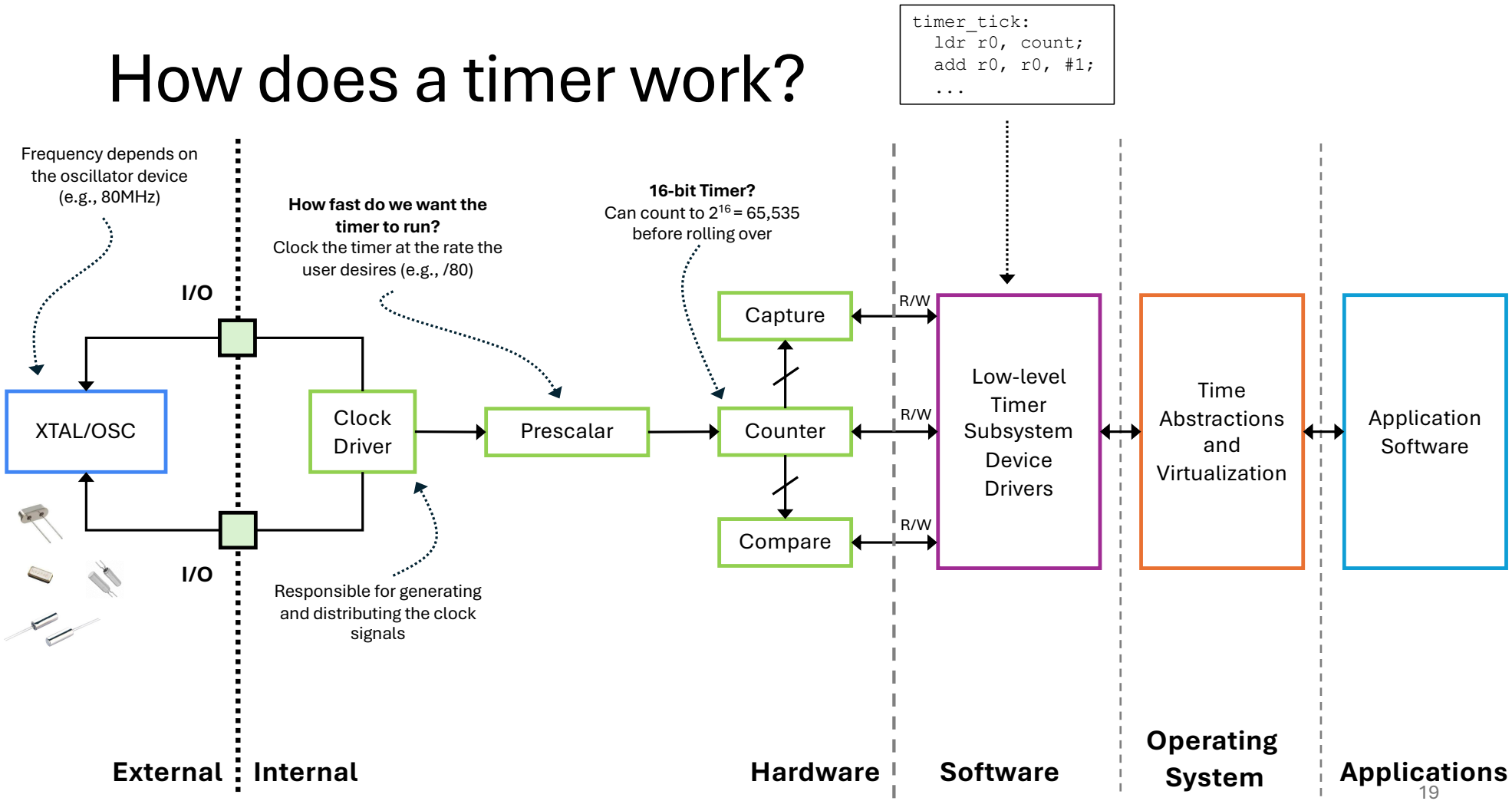


60 second brainstorm (talk with your neighbor)

Electronic devices need some physical principle to mark the passage of time.

What do they use?

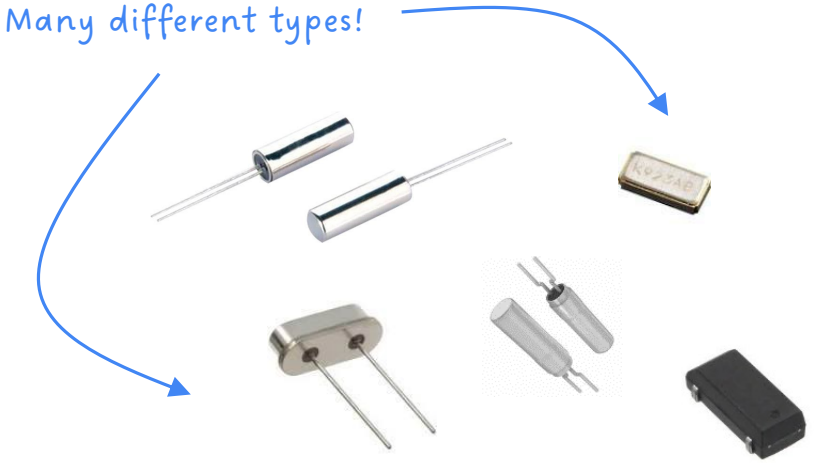
How does a timer work?

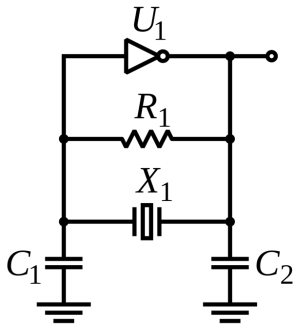


Crystal Oscillators

- A quartz crystal resonator consists of a small, thin sheet of quartz mounted between two metallic plates (electrodes)

- Lower frequencies



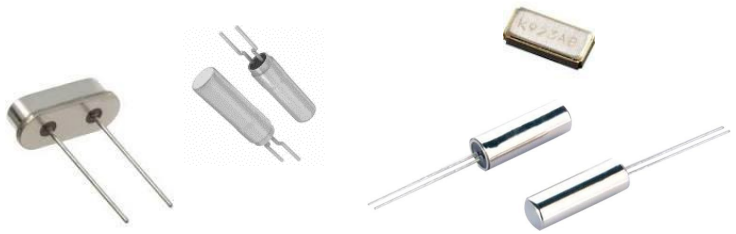


Pierce Gate Oscillator

Types of Clock Sources

Quartz Crystal

- Typically 32.768 KHz for Real Time Clocks
  - Used to track time
  - Years, months, days, hours, mins, seconds
- Typically kHz to few MHz range

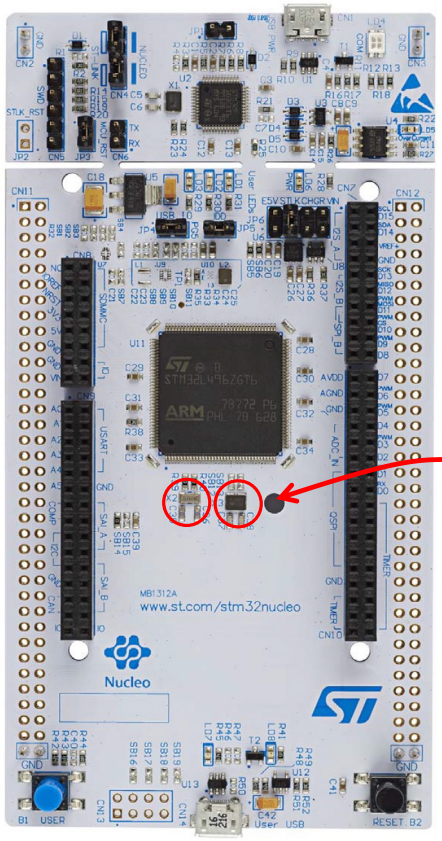


SAW Resonator

- Surface Acoustic Wave Resonator
- Typically 10– 100 MHz
- Very accurate



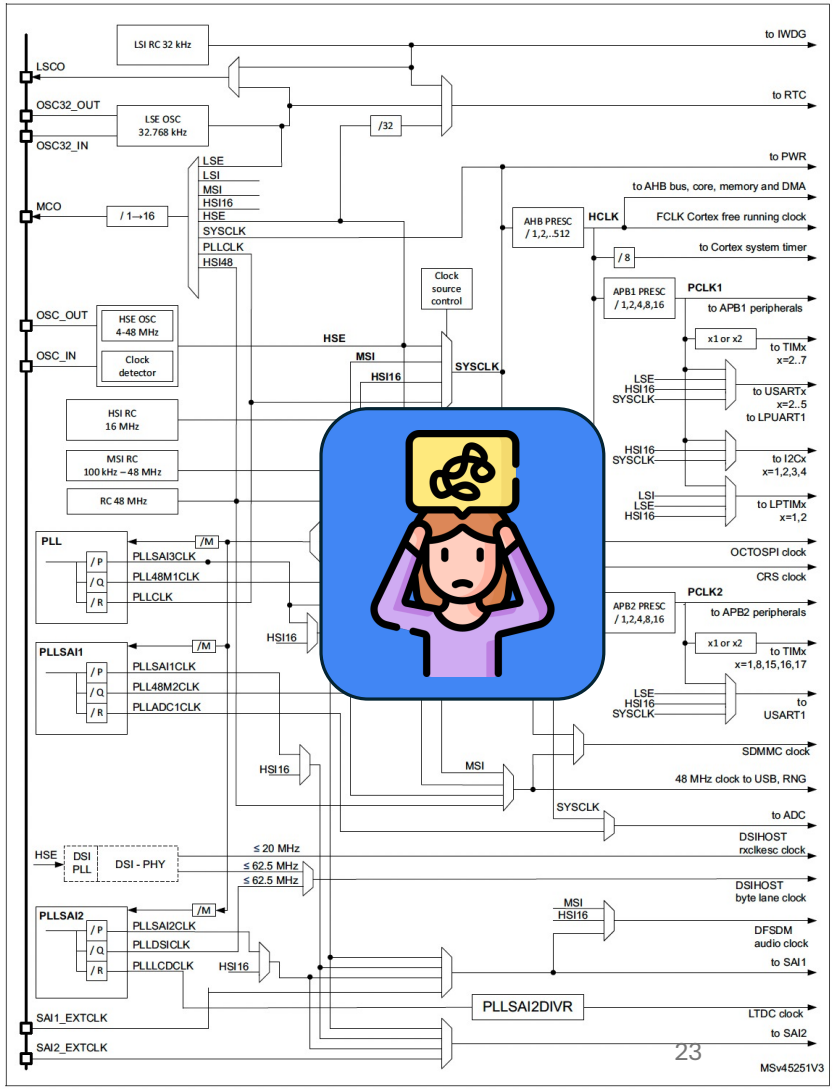
STM32L4 Clock Tree



External Crystals

From the STM32 Manual

Figure 16. Clock tree for STM32L4Rxxx and STM32L4Sxxx devices



STM32 Clock Tree

Some Terminology

Low Speed External (LSE)
Low Speed Internal (LSI)

High Speed External (HSE)
High Speed Internal (HSI)

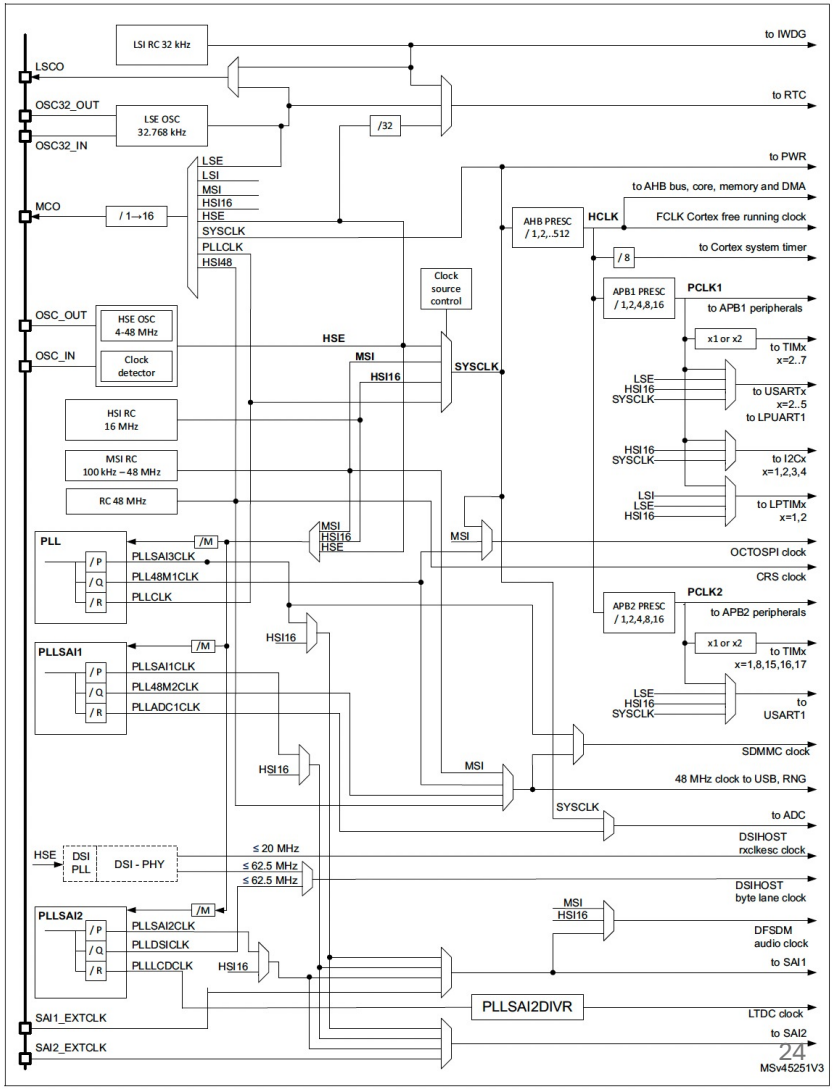
Multi-Speed Internal (MSI)
Output Clock (MCO)

Real Time Clock (RTC)

System Clock (SYSCLK)

Phase Locked Loop (PLL)

Figure 16. Clock tree for STM32L4Rxxx and STM32L4Sxxx devices



STM32 Clock Tree

Select what clock source should be used for SYSCLK

Prescalars

Low Speed Internal Resistor-Capacitor (RC) 32kHz Oscillator

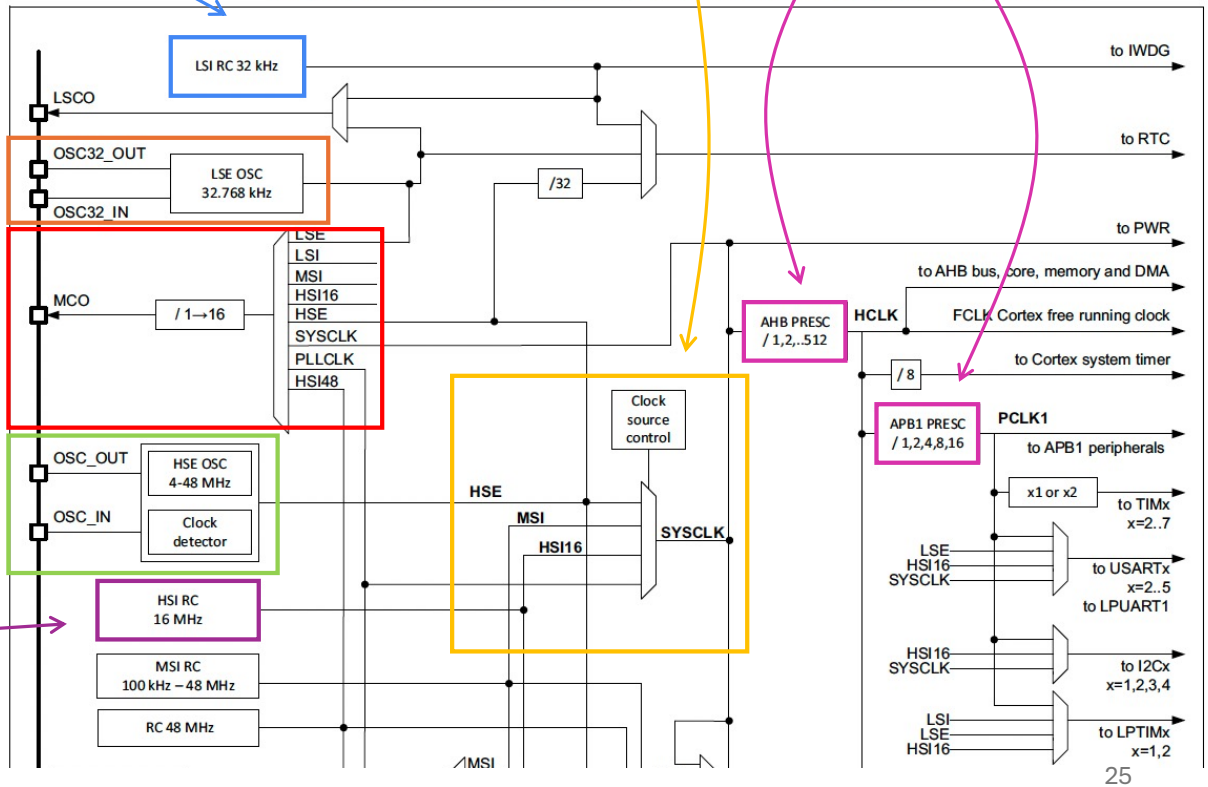
Low Speed External 32.768 kHz Crystal Oscillator (X2 = 32.768kHz)

Microcontroller Clock Output

High Speed External Oscillator (X3 = 8MHz)

High Speed Internal RC 16MHz Oscillator

Figure 16. Clock tree for STM32L4Rxxx and STM32L4Sxxx devices



STM32 Clock Tree

Phase Locked Loop

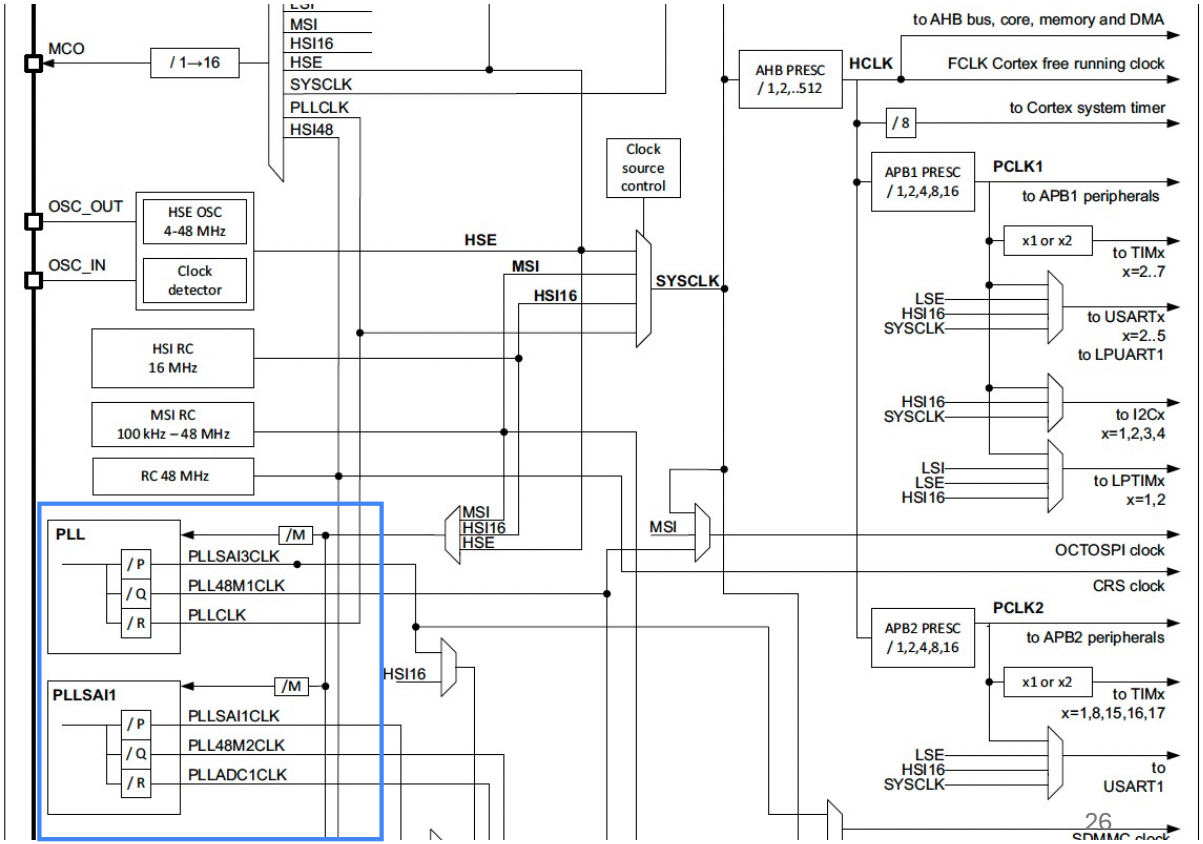
Generate a stable frequency that is a *multiple* of the input frequency

Higher clock noise

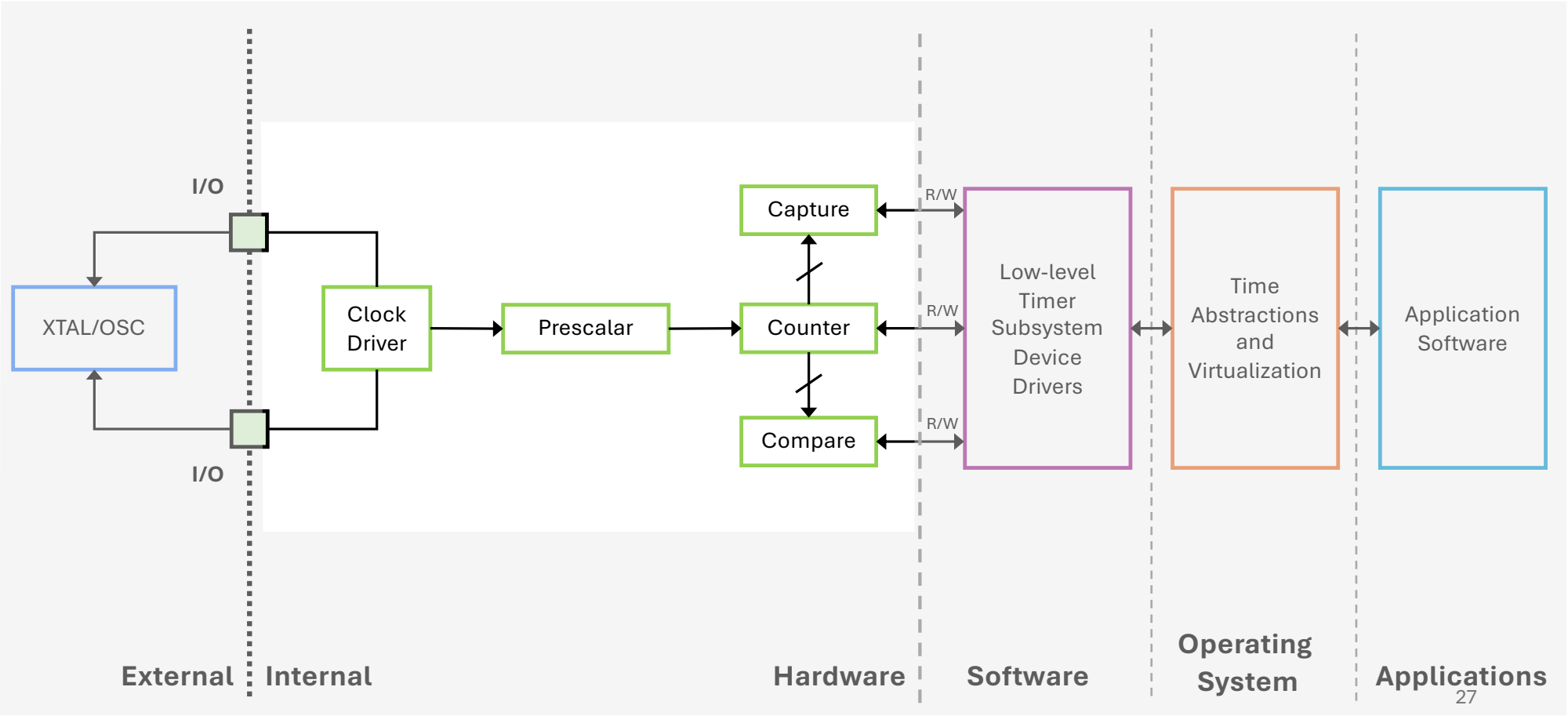
Increased power consumption

$$f = \frac{f_{PLLSource}}{M} \times N$$

(divide by /R for PLLCLK, etc.)

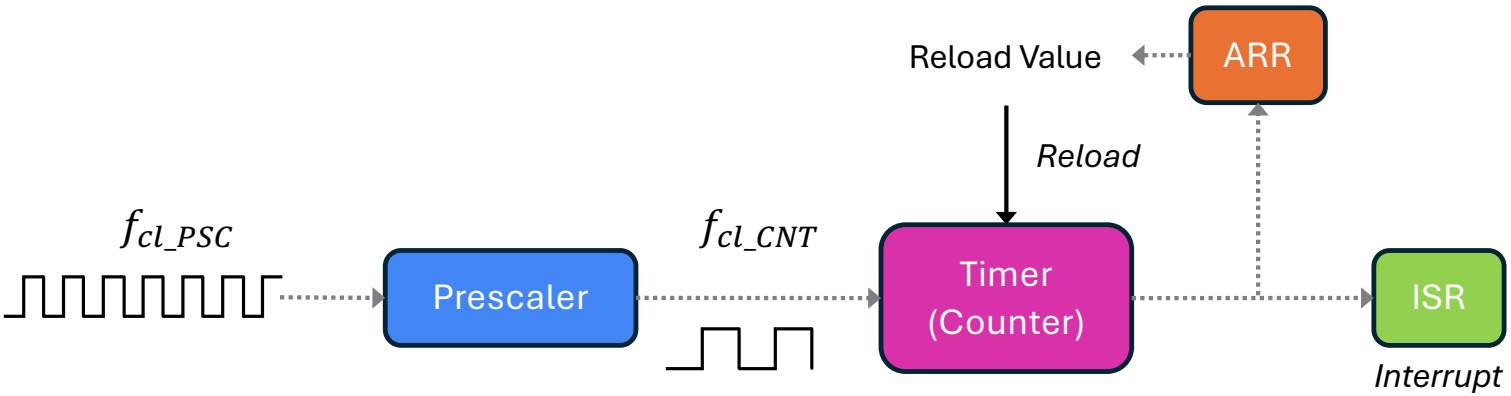


How does a timer work?



Counter Basics

Auto Reload Register (ARR)
Interrupt Service Routine (ISR)
Prescaler (PSC)



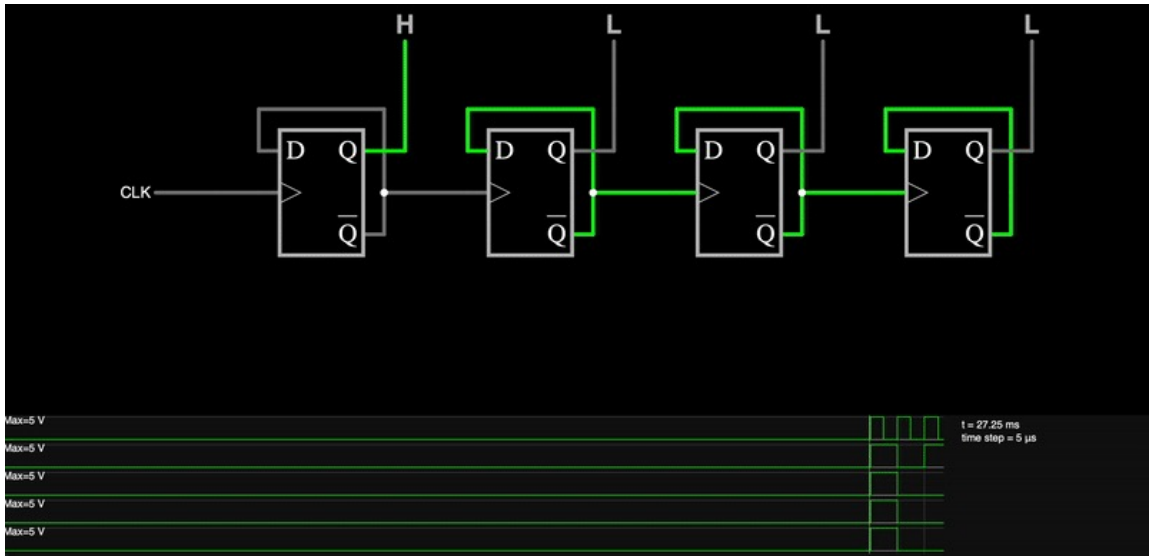
$$f_{cl_CNT} = \frac{\mathrm{f}_{cl_\mathrm{PSC}}}{PSC + 1}$$

If you are curious … Clock Divider

How could you implement a clock divider?

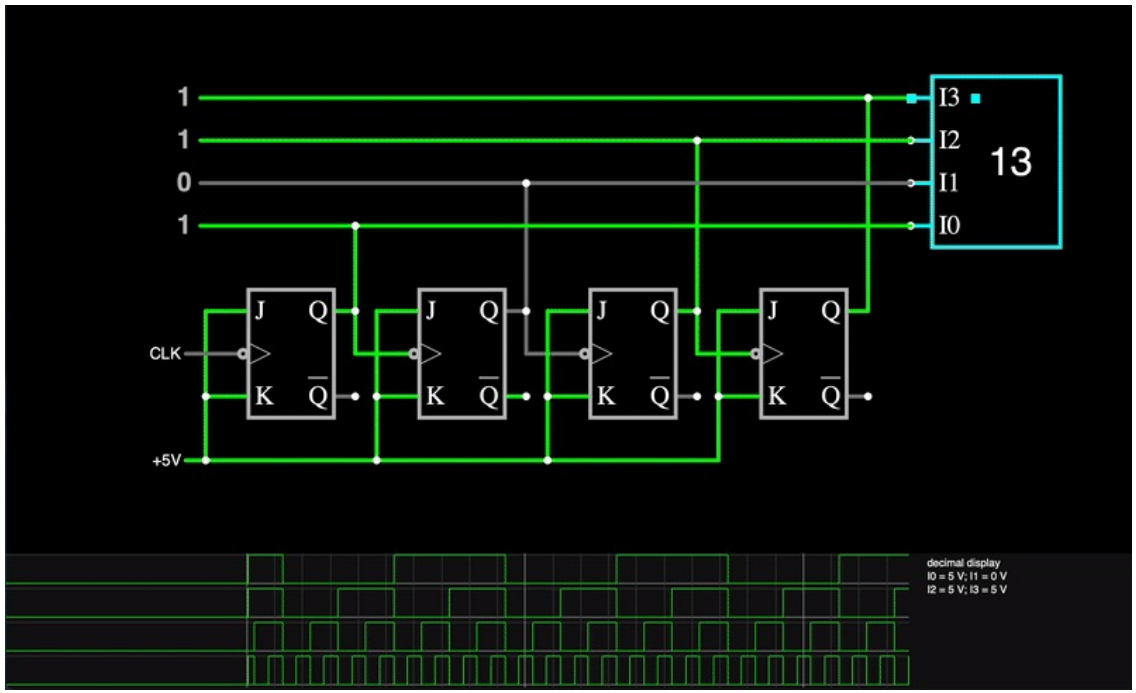
Use DFFs

[Falstad Clock Divider Example (LINK)]



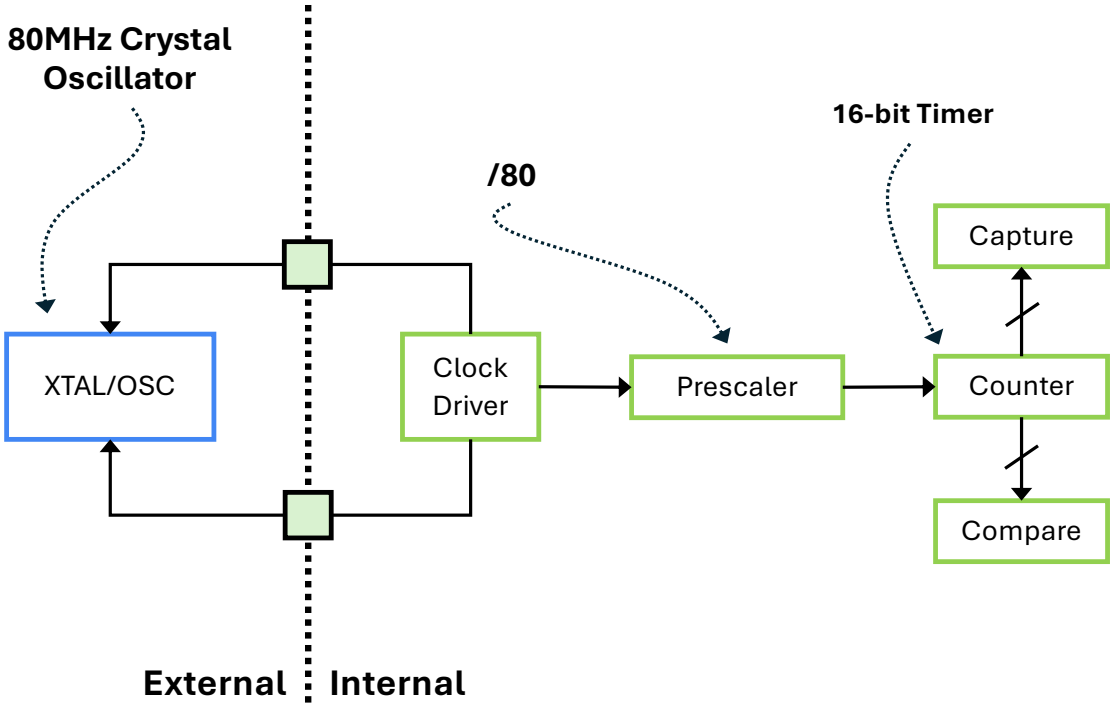
If you are curious … Counter Example

Falstad Counter Example (LINK)



- Count up
- Count down
- Hold

Group Exercise



What is the output clock frequency after the prescaler?

1MHz

What is the longest duration of event that could be measured?

16-bit Timer?
Can count to $2^{16} = 65{,}535$ before rolling over

$$\frac{1}{f} = \frac{1}{1MHz} = 0.001\ ms \times 65535 = 65.535\ ms$$

Or measure events no longer than 65.5ms!

Oscillating Frequency Tradeoffs

There is a tradeoff between **resolution** and **range**

80MHz OSC Resolution:

$$\frac{1}{80e6} = 12.5\ ns$$

1MHz OSC Resolution:

$$\frac{1}{1e6} = 1000\ ns$$

16-bit Timer

80MHz OSC Range:

$$\frac{1}{80e6} \times 2^{16} = 0.8192\ ms$$

1MHz OSC Range:

$$\frac{1}{1e6} \times 2^{16} = 65.536\ ms$$

Key Takeaways

Higher oscillating frequency = higher resolution

Higher oscillating frequency = Timer overflows quicker

2-minute Group Exercise

Given an input clock of 100MHz what is the maximum timeout period for:

- 16 Bit Timer
- 32 Bit Timer
- 64 Bit Timer

2-minute Group Exercise

Given an input clock of 100MHz what is the maximum timeout period for:

- 16 Bit Timer $\frac{1}{f} = \frac{1}{100MHz} = 0.001\ ms \times 2^{16} = 0.655\ ms$
- 32 Bit Timer $\frac{1}{f} = \frac{1}{100MHz} = 0.001\ ms \times 2^{32} = 42.95\ s$
- 64 Bit Timer $\frac{1}{f} = \frac{1}{100MHz} = 0.001\ ms \times 2^{64} = 5849.42\ years$

Capture and Compare

Capture and Compare

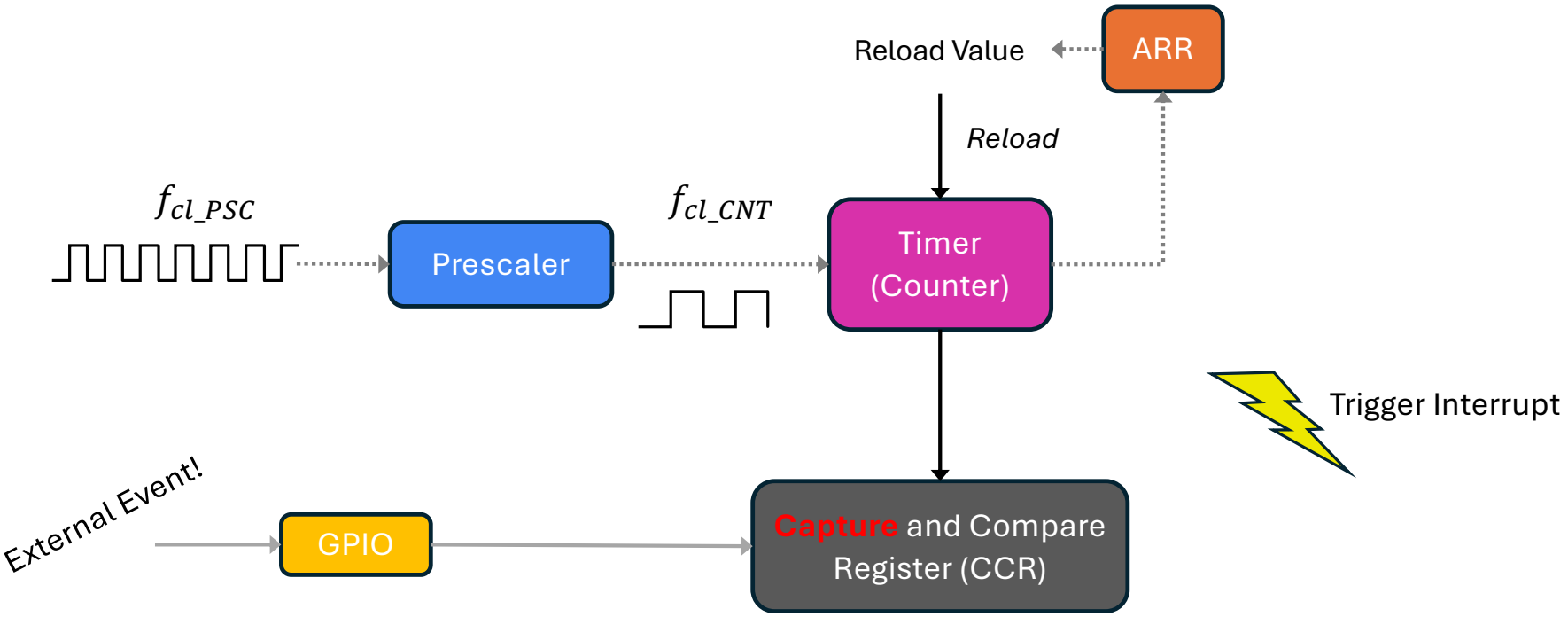
Two primary timer modes:

1. **Measure how long something takes**
   capture how long something takes to happen

2. **Have something happen once or every N time periods**
   compare a clock source to desired time and act

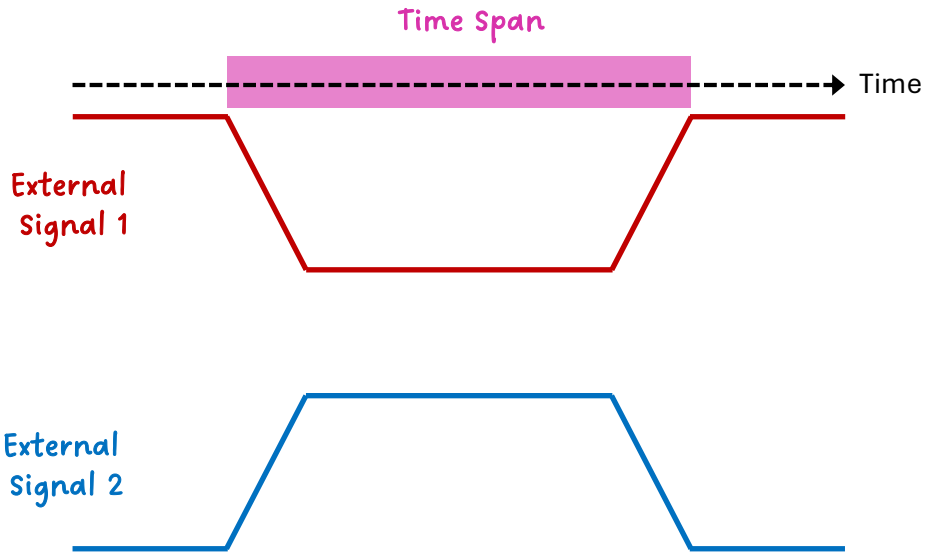
Capture

Auto Reload Register (ARR)
Interrupt Service Routine (ISR)
Prescaler (PSC)

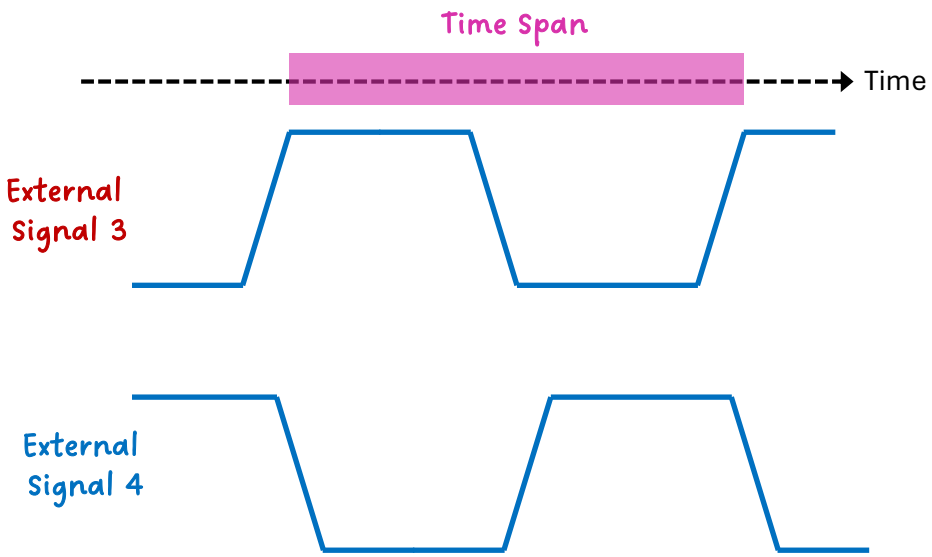


Input Capture

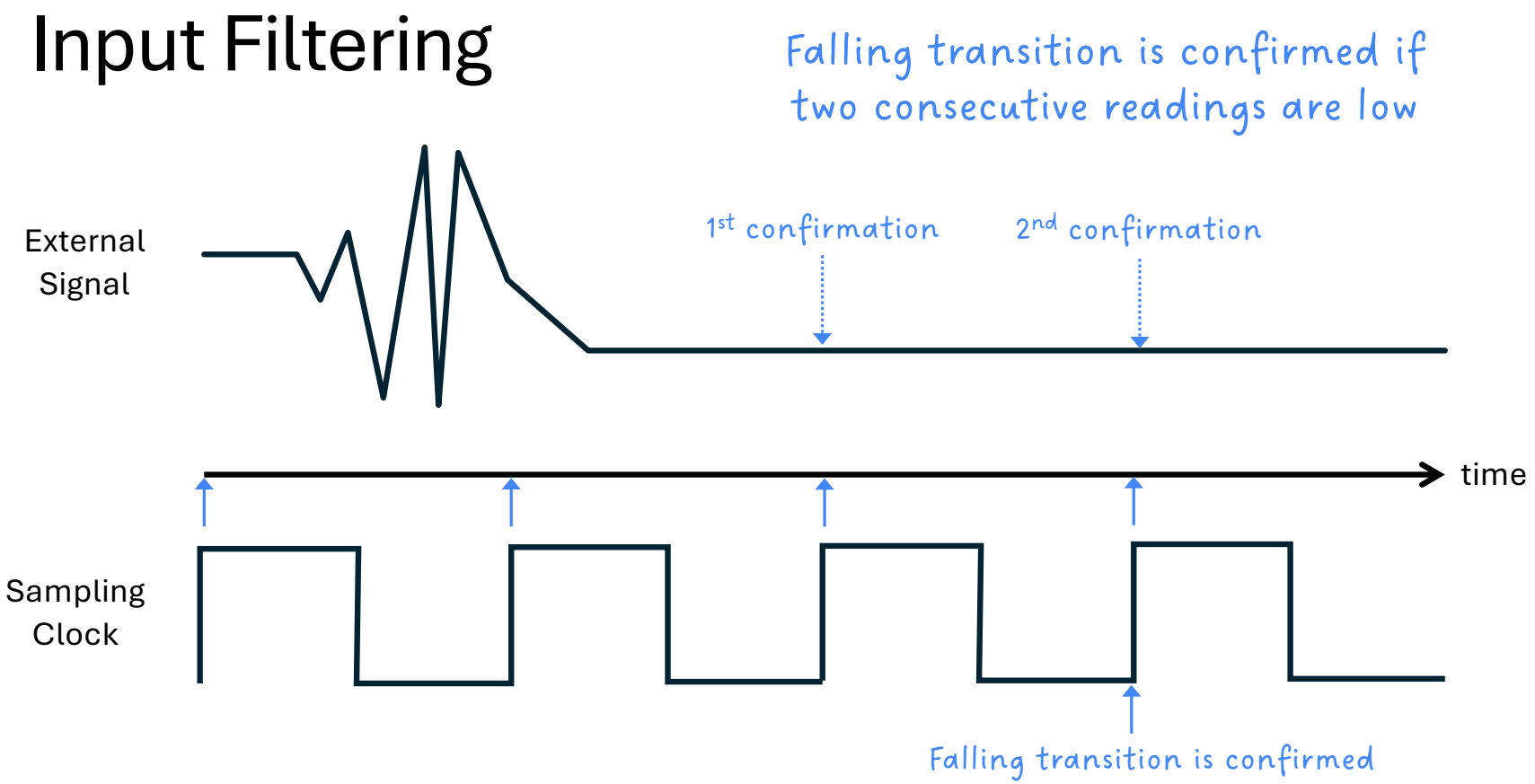
Monitor **Both** Rising & Falling Edge



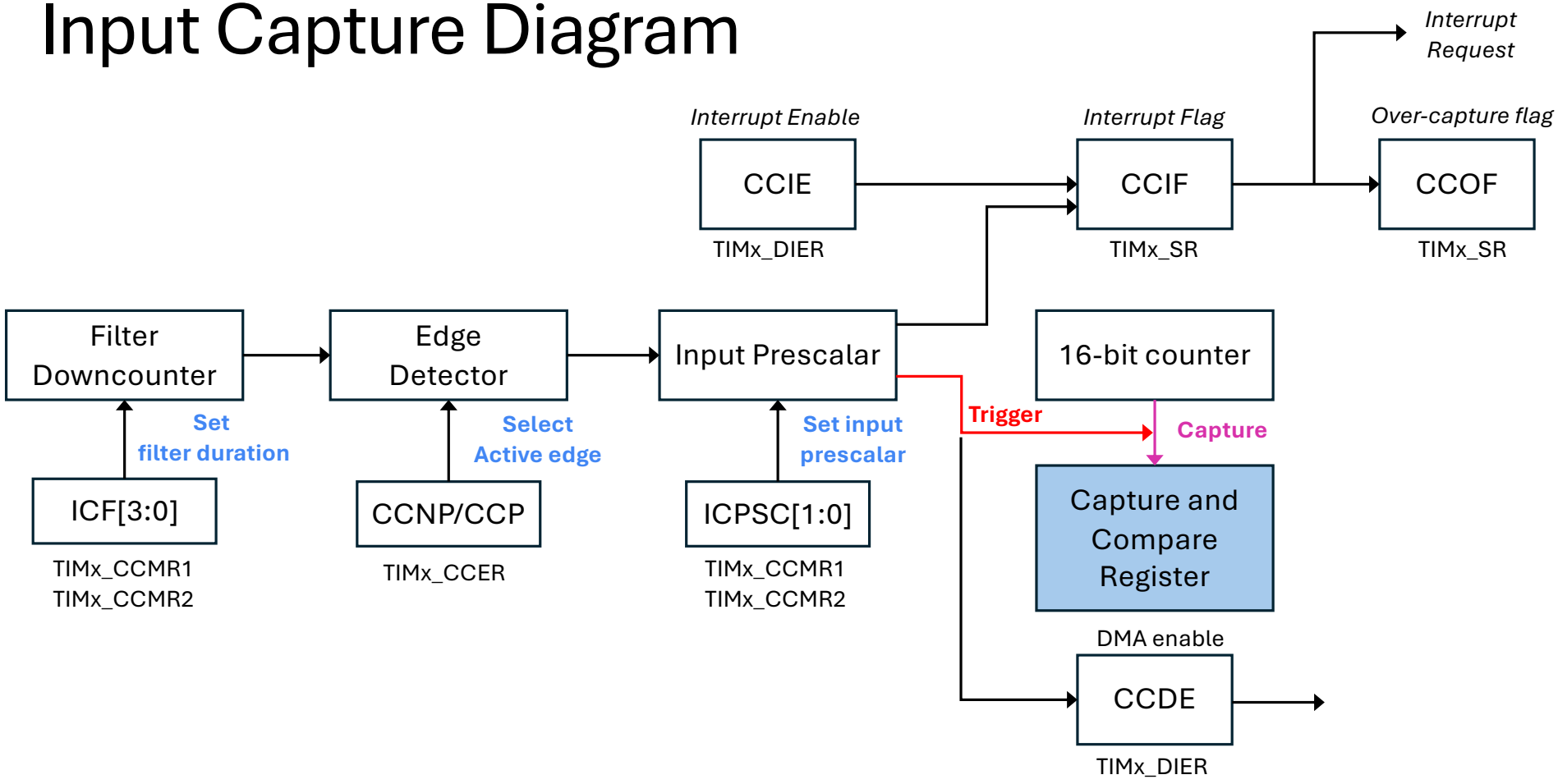
Monitor **Only** Rising or Falling Edge



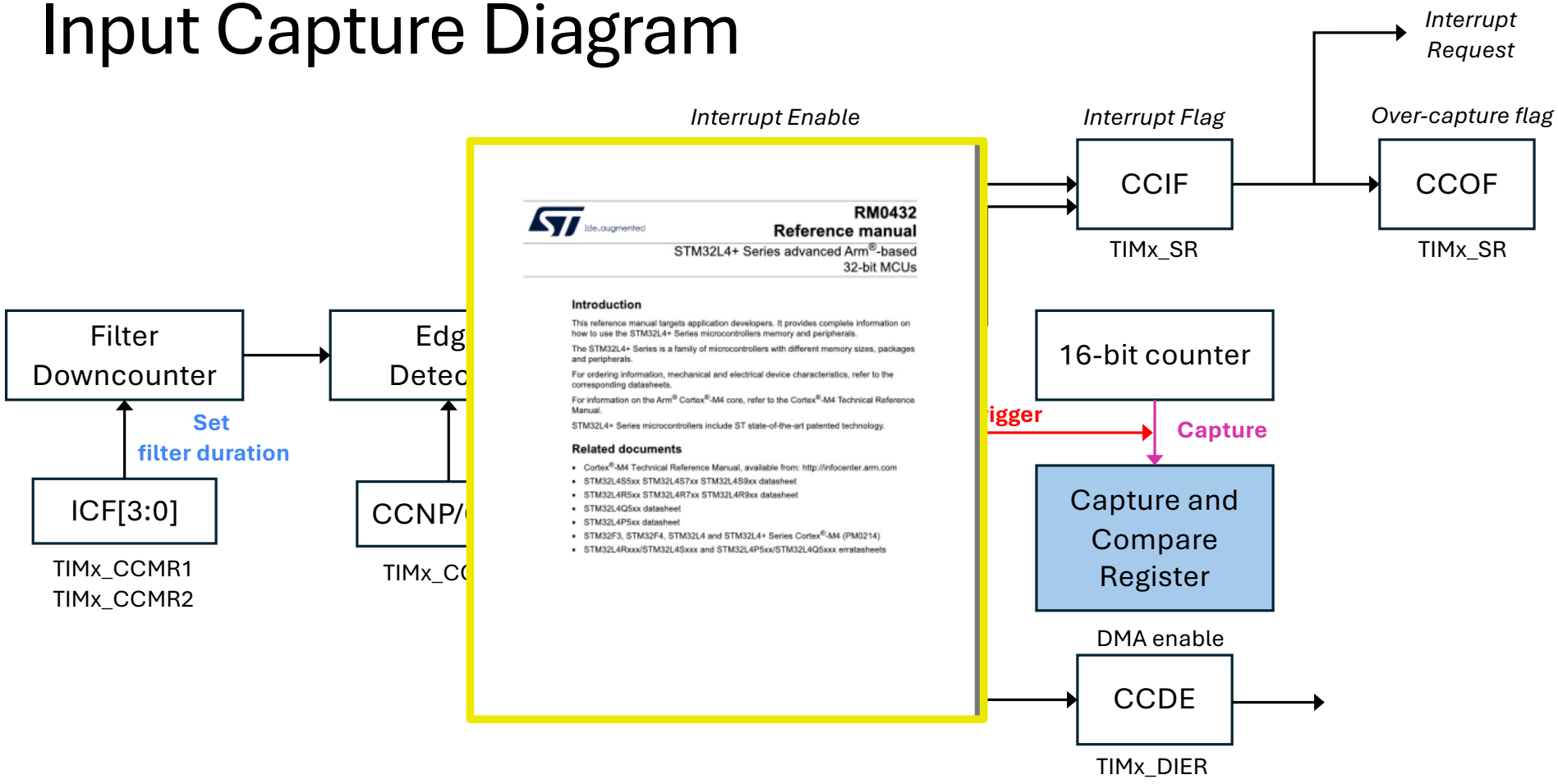
Input Filtering



Input Capture Diagram

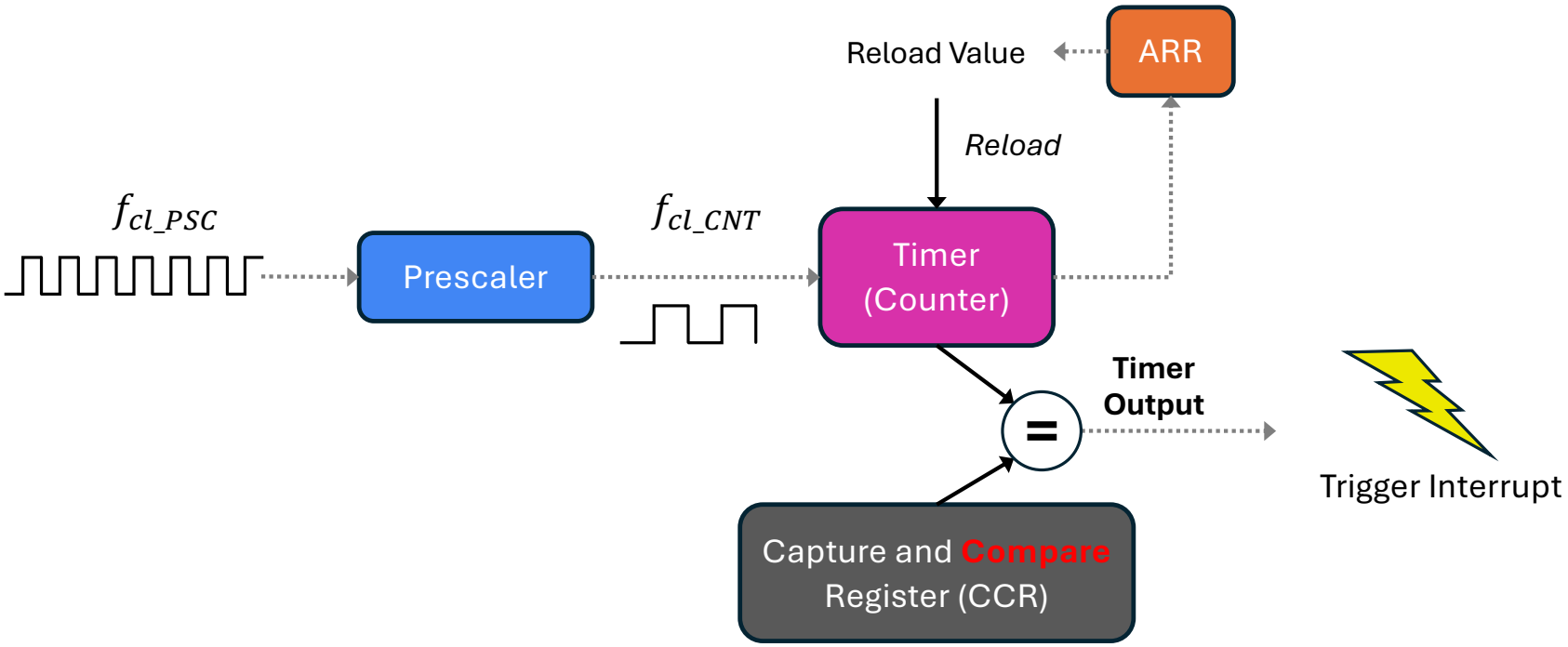


Input Capture Diagram



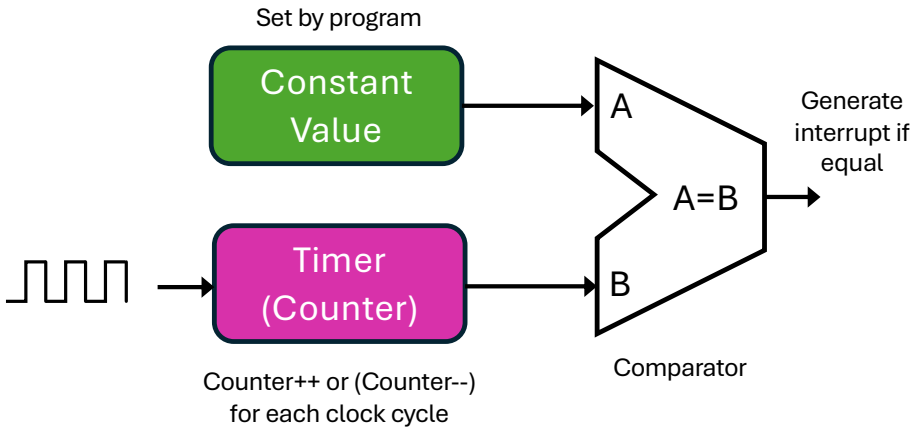
Compare

Auto Reload Register (ARR)
Interrupt Service Routine (ISR)
Prescaler (PSC)



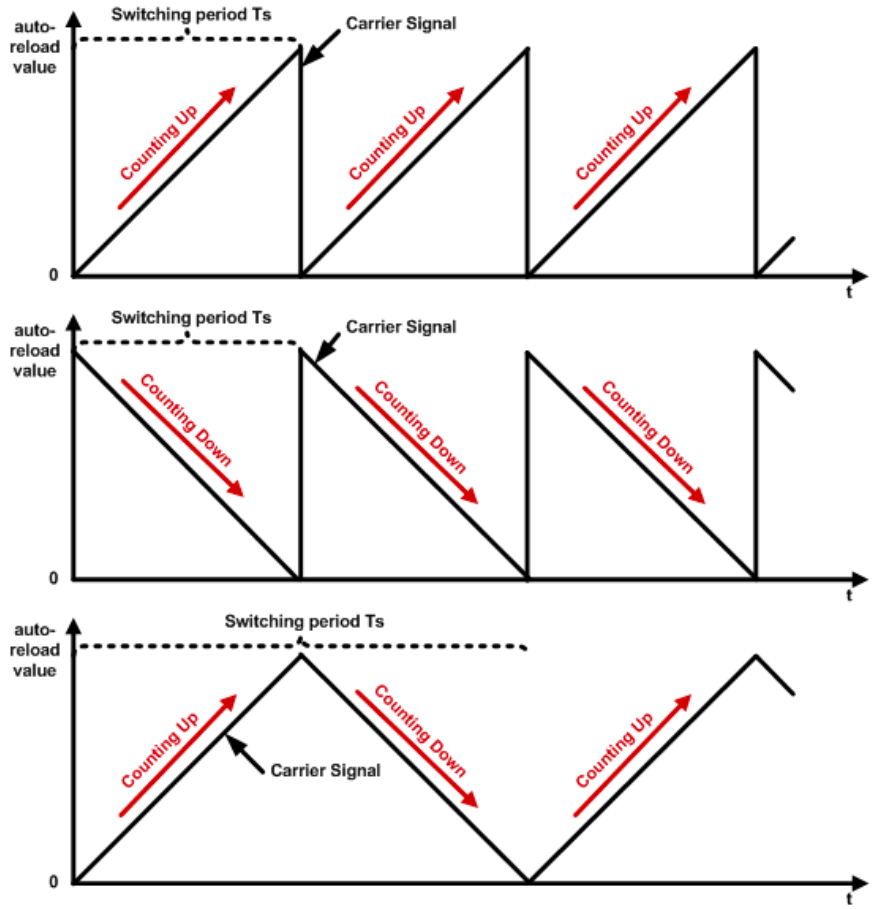
Output Compare Mode (OMC)

Output Compare Mode (OMC)	Timer Output (OCREF)
000	Frozen
001	High if CNT == CCR
010	Low if CNT = CCR
011	Toggle if CNT == CCR
100	Forced low (always low)
101	Forced high (always high)

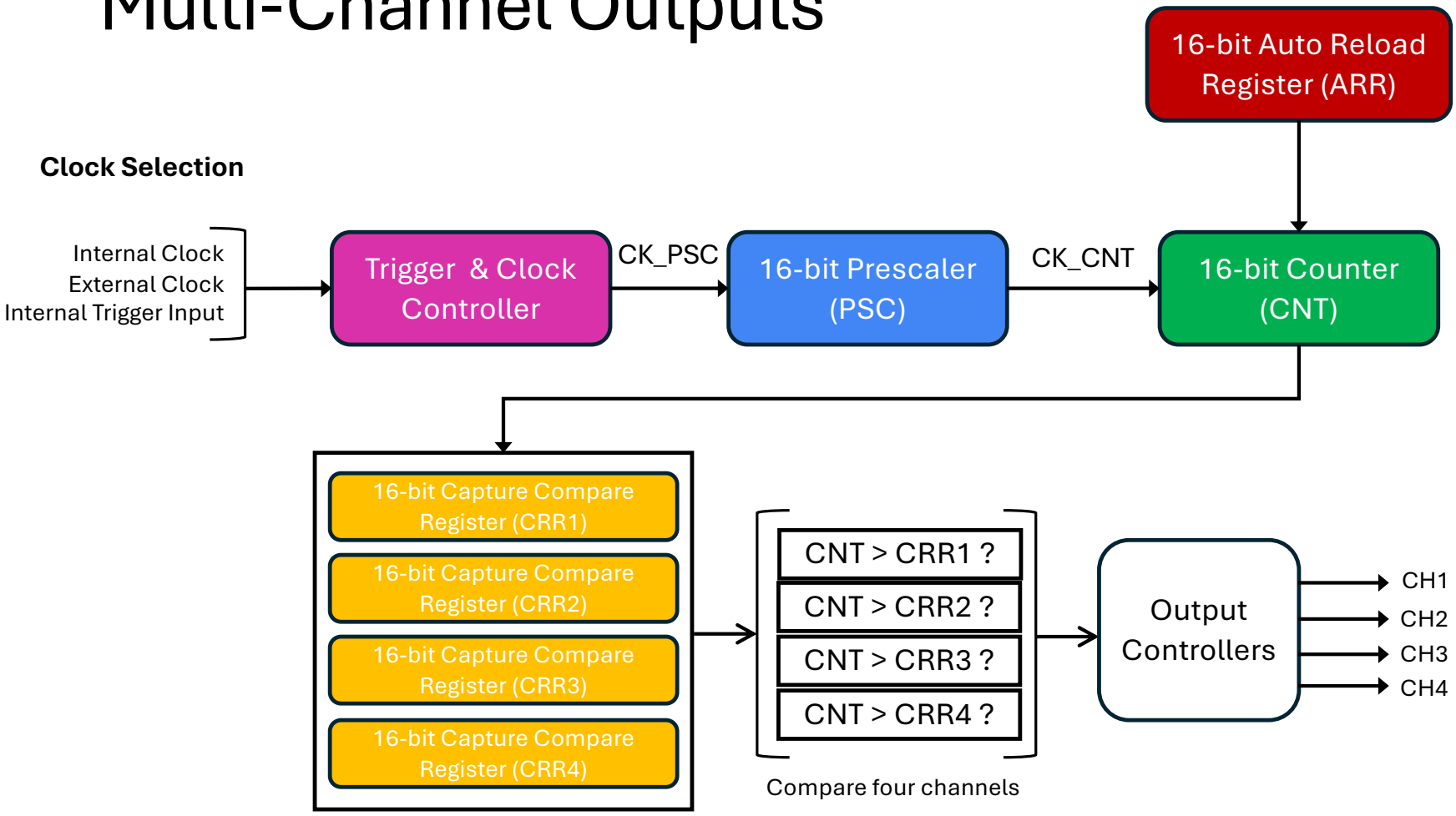


Counting Configurations

- Count up and reload
- Count down and reload
- Count up and then count down



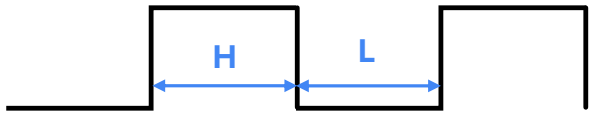
Multi-Channel Outputs

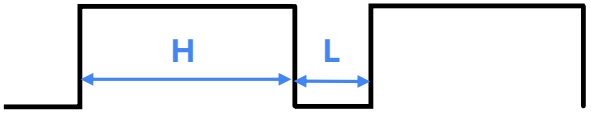


Pulse Width Modulation (PWM)

Pulse Width Modulation (PWM)

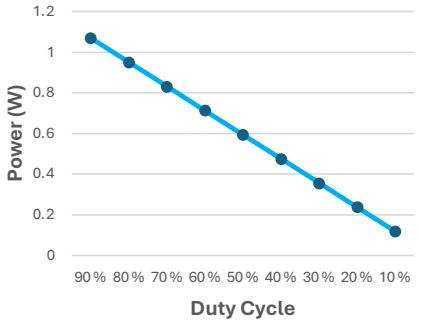
Basic Idea: Create a digital output wave of fixed frequency, but allow the microcontroller to vary its duty cycle





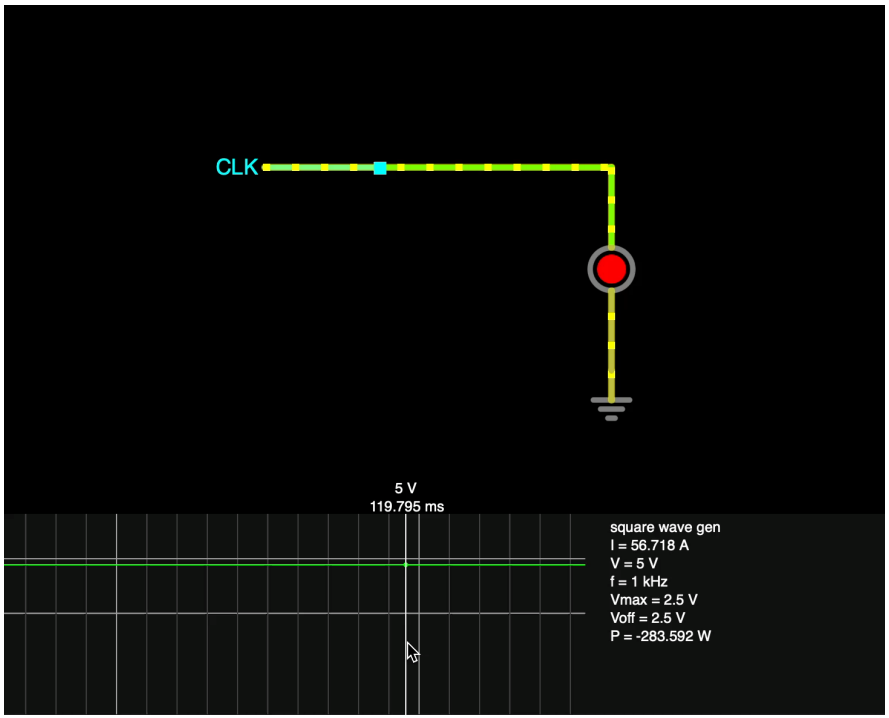
$$duty\ cycle = \frac{H}{H+L} \qquad P = V \times I$$

$$Delivered\ Power = P * (duty\ cycle) = \frac{H}{H+L} \times P$$



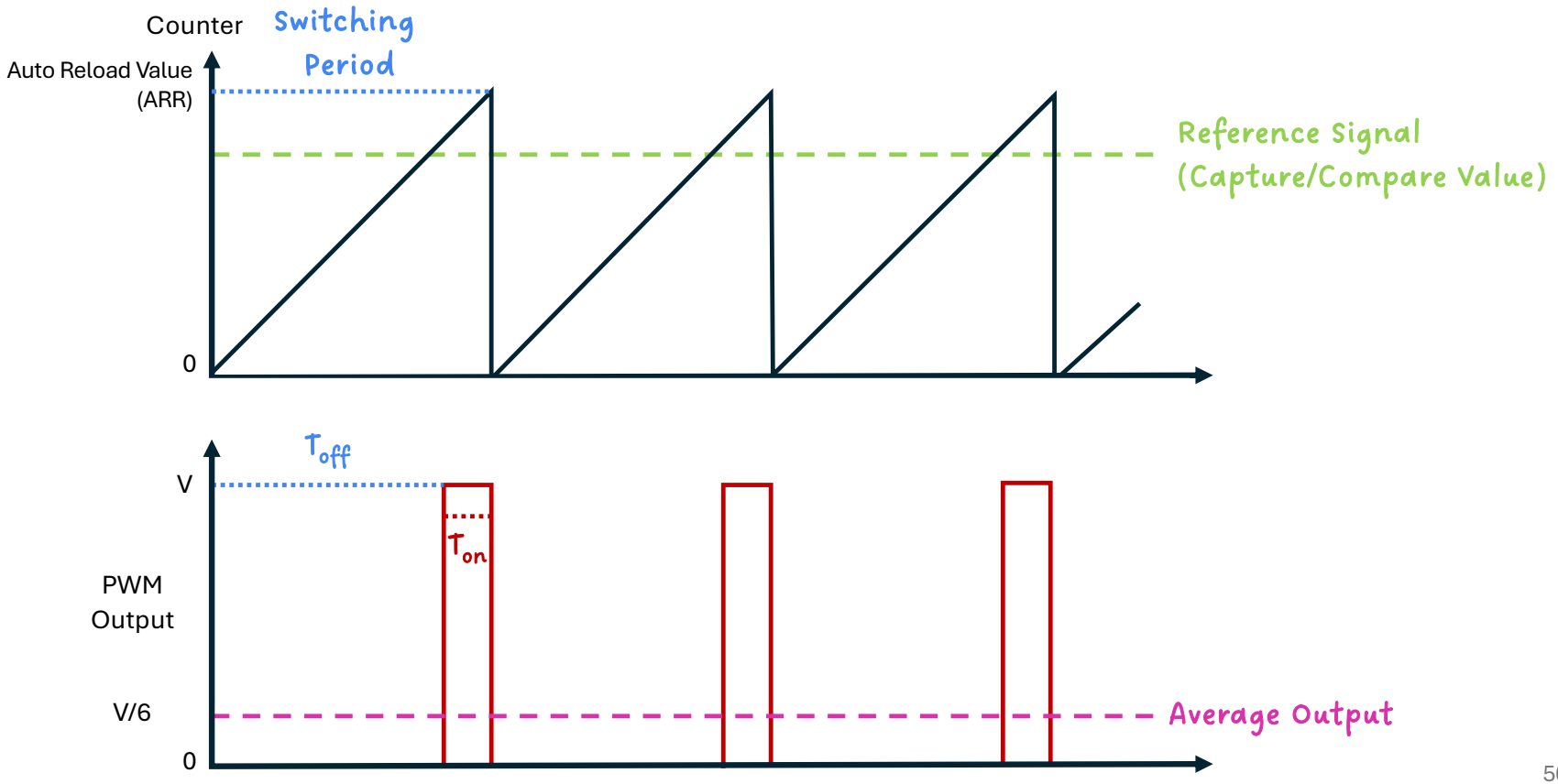
Pulse Width Modulation

High level example



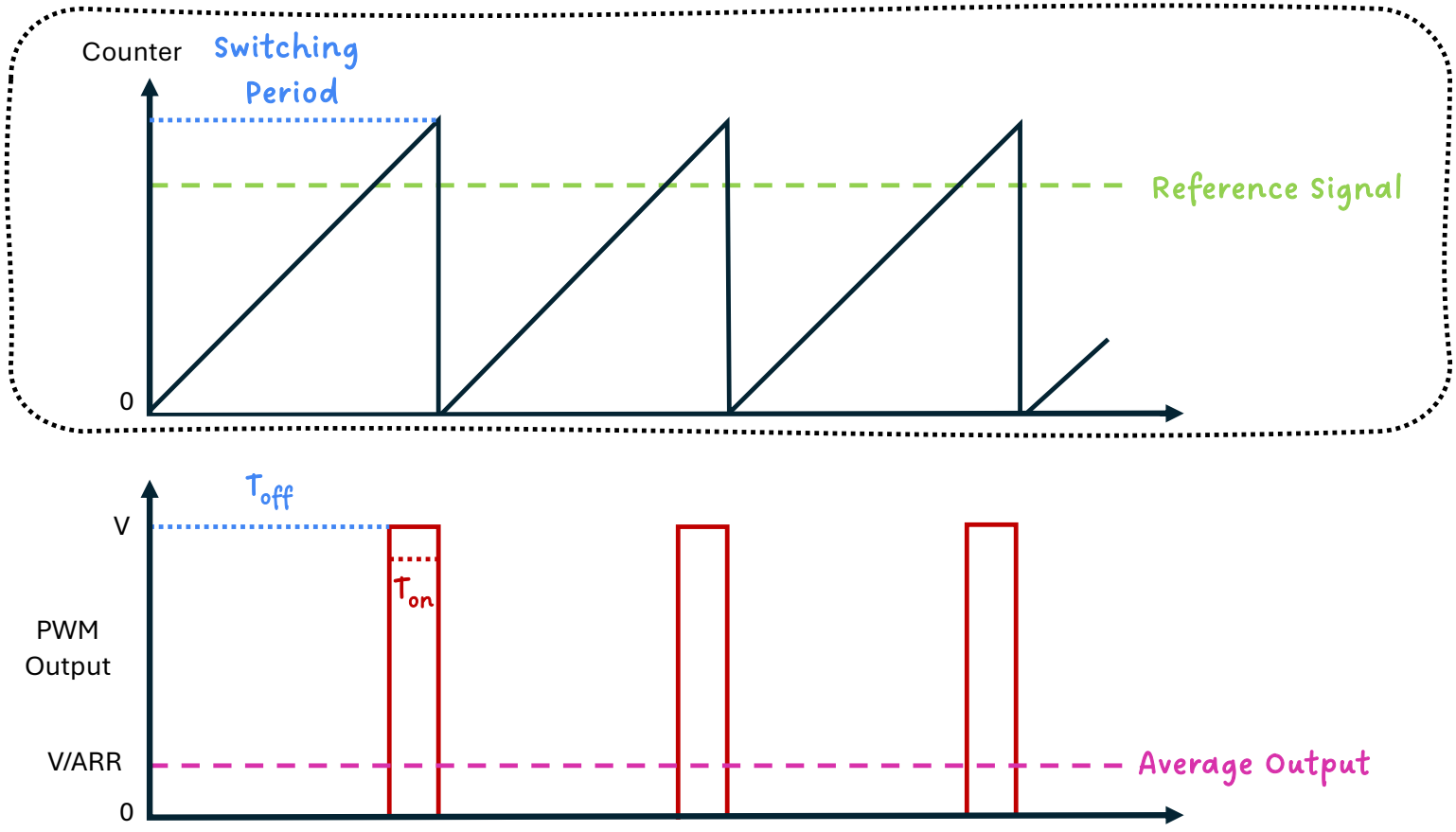


PWM Mode

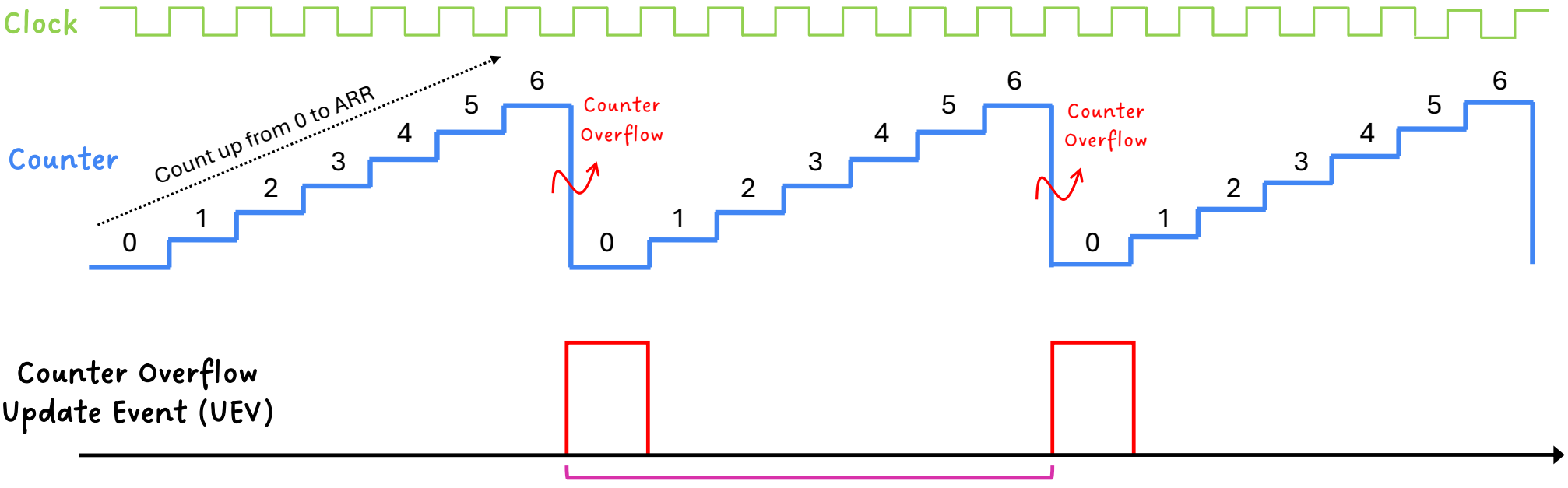

Counter
Switching Period
Auto Reload Value (ARR)
Reference Signal (Capture/Compare Value)
0
T_{off}
V
T_{on}
PWM Output
V/6
Average Output
0

PWM Mode

How do we setup the counter?

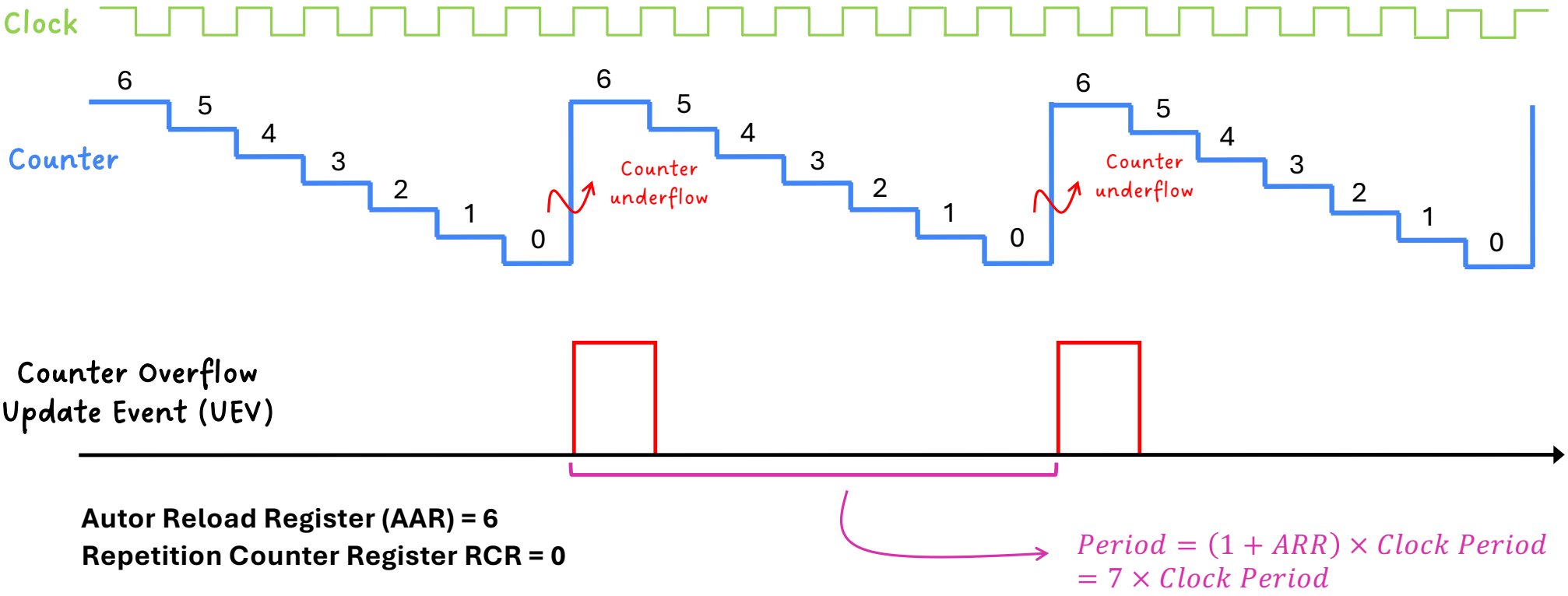


Edge-aligned Mode (Up-counting)



Autor Reload Register (AAR) = 6
Repetition Counter Register RCR = 0
0 = Repeat forever. Set to a non-zero value, the timer counts the number of occurrences of an update event before generating an interrupt

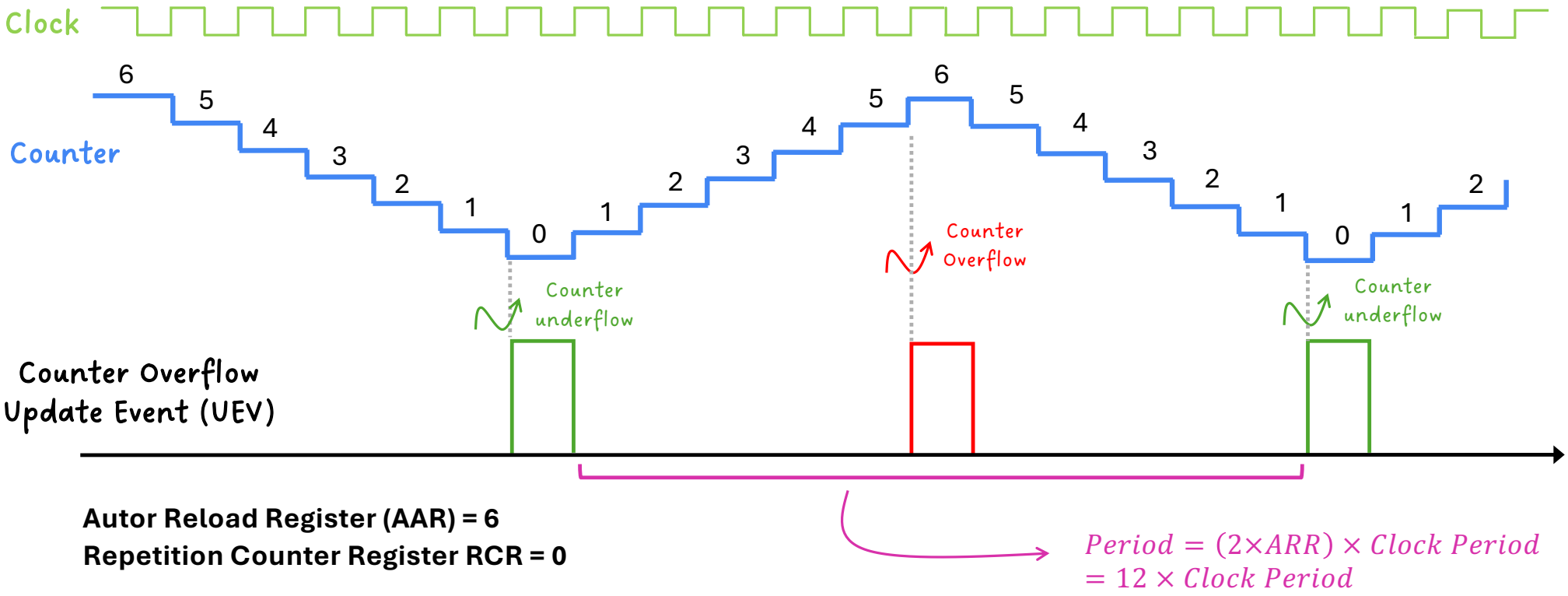
Edge-aligned Mode (Down-counting)



Autor Reload Register (AAR) = 6
Repetition Counter Register RCR = 0

$$Period = (1 + ARR) \times Clock\ Period$$
$$= 7 \times Clock\ Period$$

Center-aligned Mode

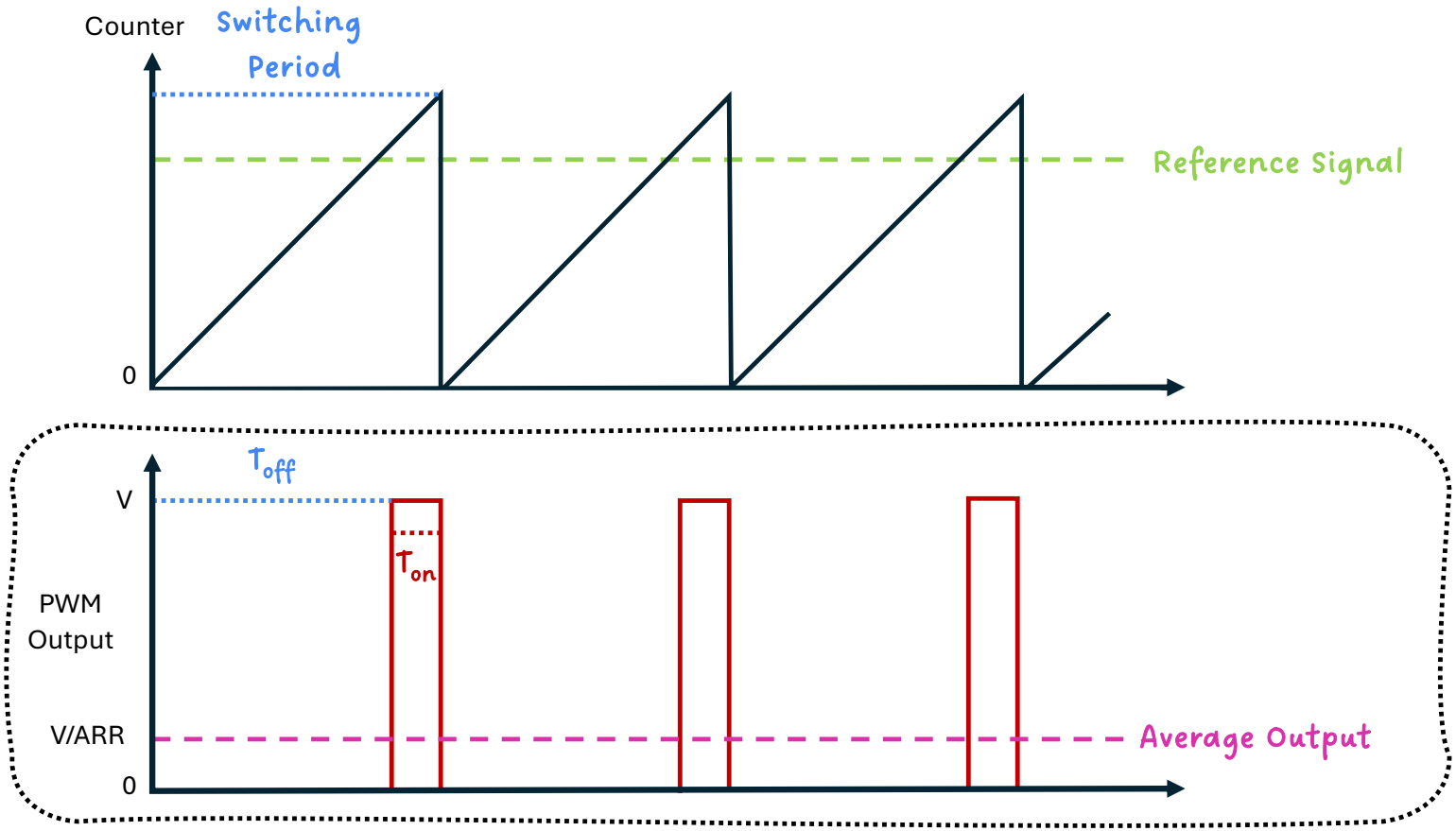


Autor Reload Register (AAR) = 6
Repetition Counter Register RCR = 0

$$Period = (2 \times ARR) \times Clock\ Period$$
$$= 12 \times Clock\ Period$$

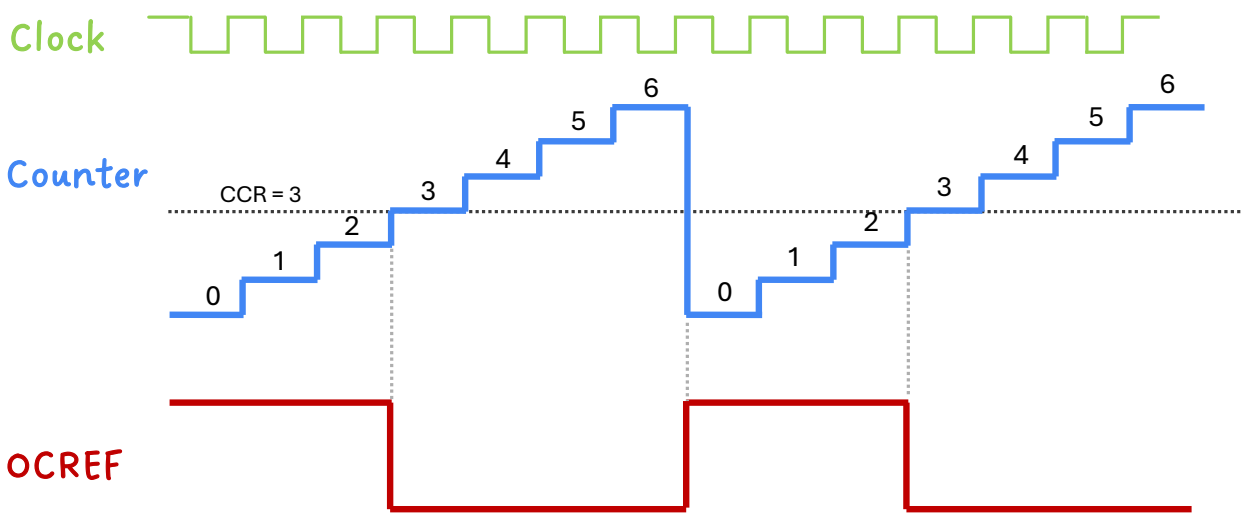
PWM Mode

How do we generate the PWM output signal?



PWM Mode 1 (Low True)

Up-counting Mode, ARR = 6, **CCR = 3**, RCR = 0



Timer Output

High if counter < CCR

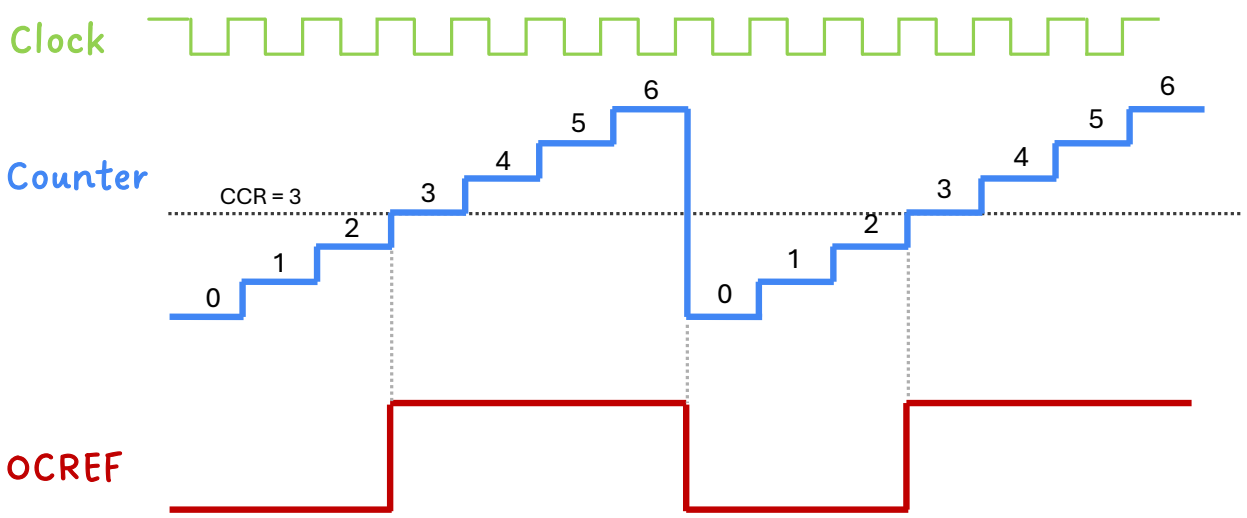
Low if counter ≥ CCR

$$Duty\ Cycle = \frac{CCR}{ARR + 1} = \frac{3}{7}$$

$$Period = (1 + ARR) \times Clock\ Period = 7 \times Clock\ Period$$

PWM Mode 2 (High True)

Up-counting Mode, ARR = 6, CCR = 3, RCR = 0



Timer Output

Low if counter < CCR

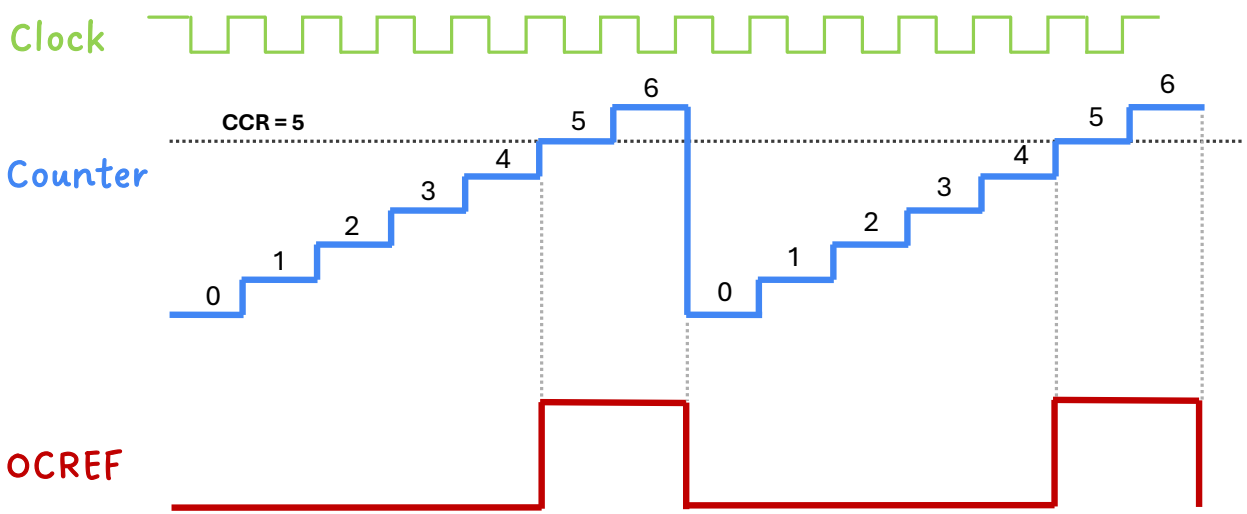
High if counter ≥ CCR

$$Duty\ Cycle = 1 - \frac{CCR}{ARR + 1} = \frac{4}{7}$$

$$Period = (1 + ARR) \times Clock\ Period = 7 \times Clock\ Period$$

PWM Mode 2 (High True)

Up-counting Mode, ARR = 6, **CCR = 5**, RCR = 0



Timer Output

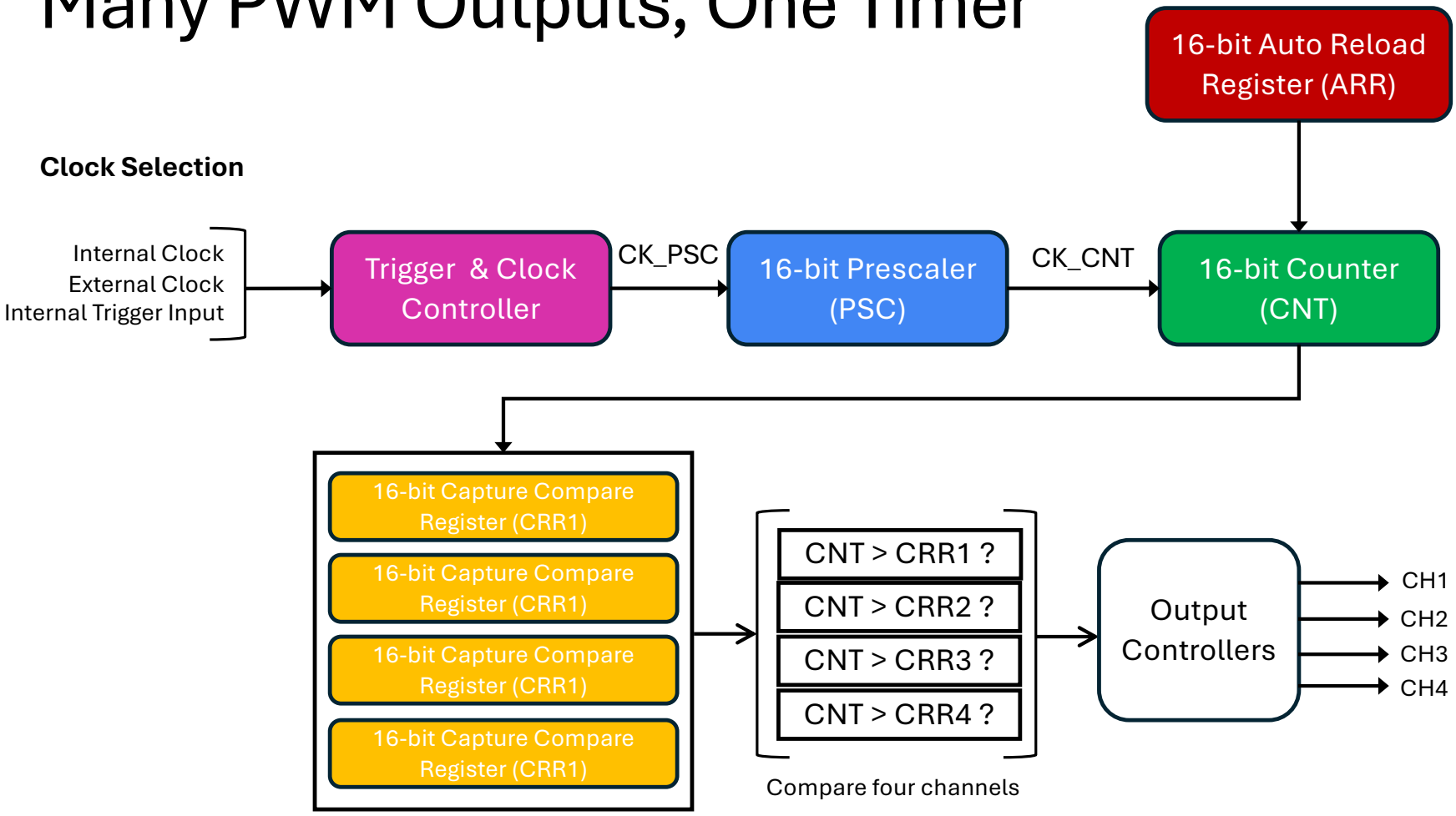
Low if counter < CCR

High if counter ≥ CCR

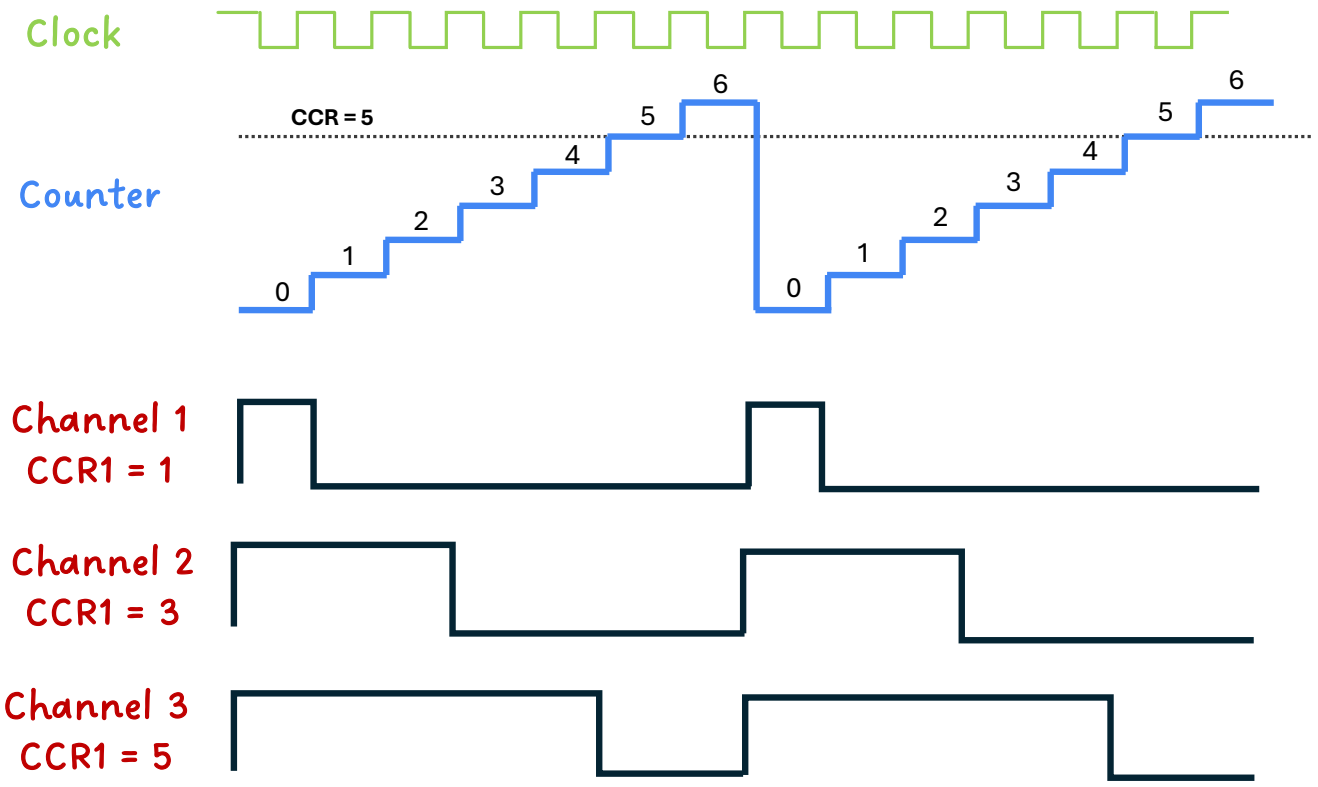
$$Duty\ Cycle = 1 - \frac{CCR}{ARR + 1} = \frac{2}{7}$$

$$Period = (1 + ARR) \times Clock\ Period = 7 \times Clock\ Period$$

Many PWM Outputs, One Timer



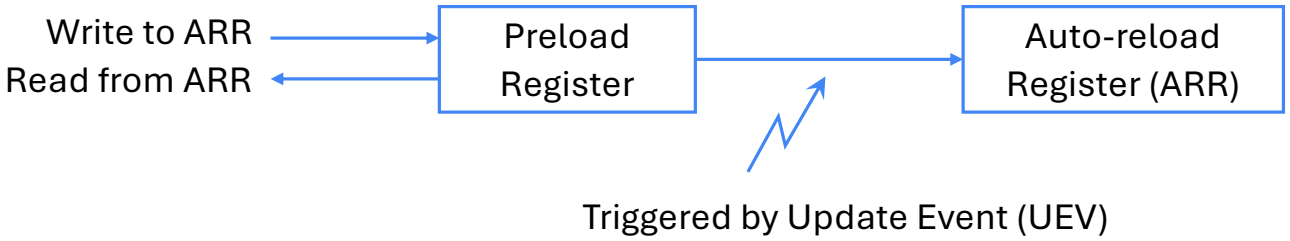
Multiple PWM Outputs



Auto-Reload Preload Enable (ARPE)

Auto-Reload Preload Enable (ARPE) bit in TIMx_CR1

ARPE = 1 (synchronous update)



Read the Datasheet

We covered many things, but not everything!

Table 278. Output control bits for complementary OCx and OCxN channels with break feature

Control bits					Output states[1]	
MOE bit	OSSI bit	OSSR bit	CCxE bit	CCxNE bit	OCx output state	OCxN output state
1	X	X	0	0	Output disabled (not driven by the timer: Hi-Z) OCx=0, OCxN=0	
		0	0	1	Output disabled (not driven by the timer: Hi-Z) OCx=0	OCxREF + Polarity OCxN = OCxREF xor CCxNP
		0	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP	Output Disabled (not driven by the timer: Hi-Z) OCxN=0
		X	1	1	OCREF + Polarity + dead-time	Complementary to OCREF (not OCREF) + Polarity + dead-time
		1	0	1	Off-State (output enabled with inactive state) OCx=CCxP	OCxREF + Polarity OCxN = OCxREF x or CCxNP
		1	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP	Off-State (output enabled with inactive state) OCxN=CCxNP
0	0	X	X	X	Output disabled (not driven by the timer: Hi-Z).	
			0	0		
	1		0	1	Off-State (output enabled with inactive state) Asynchronously: OCx=CCxP, OCxN=CCxNP (if BRK or BRK2 is triggered).	
			1	0		
			1	1	Then (this is valid only if BRK is triggered), if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state (may cause a short circuit when driving switches in half-bridge configuration). **Note**: BRK2 can only be used if OSSI = OSSR = 1.	

Announcements

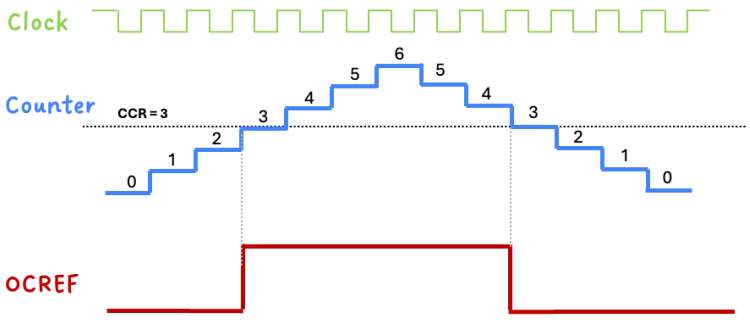
- Lab 2 due tomorrow (Friday 10/16) at 10 PM
- HW 3 due date pushed to to next Thursday (10/22) and Lab 3 pushed to Tuesday (10/27)
- Sign up for Project Proposal advising next week (will post link on Ed & Canvas announcement)

Group Exercise

PWM Timer Setup

Given the following assumptions, find the timer's PSC, ARR, CCR to achieve a PWM output with a duty cycle of 10% and a period of 10ms

- A 16 MHz clock is selected to drive the timer
- The timer is setup as a repeatedly up-counting and down-counting, similar to the figure shown below
- PWM Mode 2 is selected. In this mode, the timer output is high if the counter is larger than or equal to CCR. Otherwise, the output is low.



$$\textbf{Timer Clock Frequency} = \frac{\text{Input Clock Frequency}}{1+\text{PSC}}$$

PWM Period = 2 × ARR × Timer Clock Period (center aligned mode)

$$\textbf{PWM Duty Cycle} = \frac{1 - \text{CCR}}{\text{ARR}}$$

Group Exercise

PWM Timer Setup

$$\textbf{Timer Clock Frequency} = \frac{\text{Input Clock Frequency}}{1+\text{PSC}} = \frac{16\ \text{MHz}}{1+15} = 1\ \text{MHz}$$

PWM Period = 2 × ARR × Timer Clock Period (up then down count mode)

PWM Duty Cycle = 1 - CCR / ARR

$$\textbf{ARR} = \frac{\text{PWM Period}}{2 \times \text{Timer Clock Period}} = \frac{10}{2 \times (1/f)} = \frac{10}{0.002} = 5000$$

CCR = (1 – Duty Cycle) × ARR = (1 – 10%) × 5000 = 4500

System Timer

System Timer

A simple 24-bit down counter

Software uses SysTick to create time delays, to generate periodic interrupts, to execute tasks repeatedly

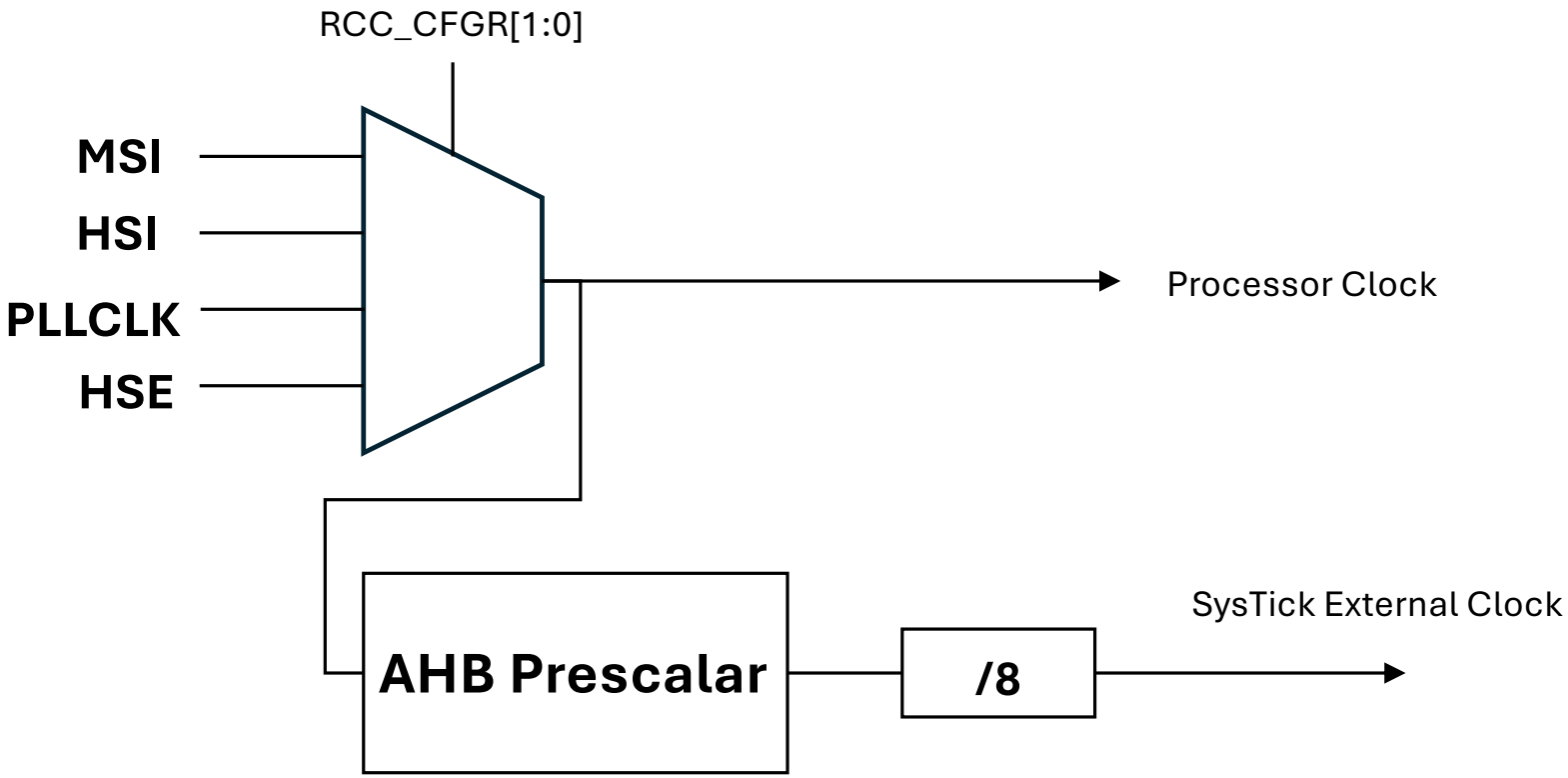
- Counts down from N-1 to 0 and the processor generates a SysTick interrupt once the counter reaches 0
- After reaching zero, the SysTick counter loads the value held in a special register (SysTick Reload Register) and counts down again

Configuring System Timers

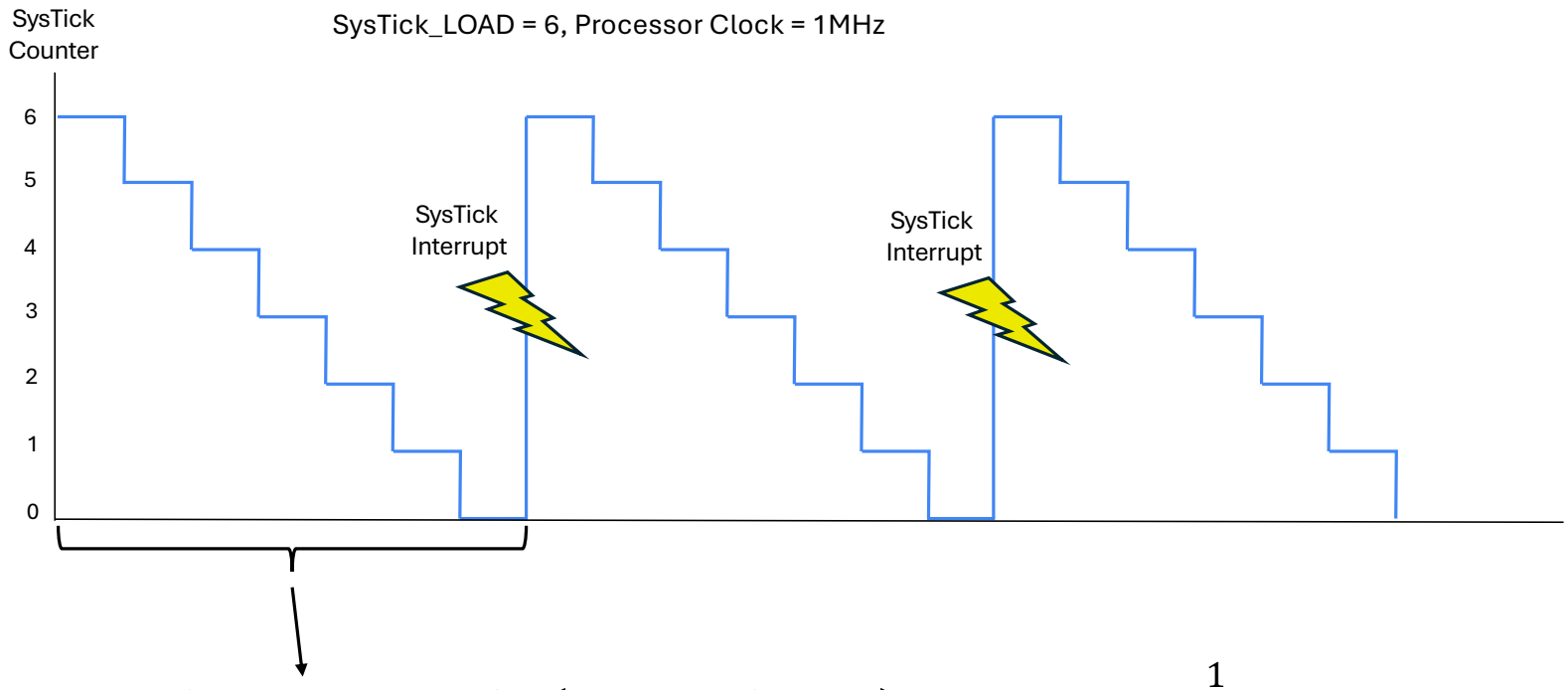
Four 32-bit registers for configuring system timers

Register	Memory Address	Description
SysTick_CTRL	0xE000E010	SysTick control and status register
SysTick_LOAD	0xE000E014	SysTick reload value register
SysTick_VAL	0xE000E018	SysTick current value register
SysTick_CALIB	0xE000E01C	SysTick calibration register

System Timer



SysTick Timer Interrupt



$$SysTick\ Interrupt\ Period = (1 + SysTick_LOAD) \times \frac{1}{SysTick\ Counter\ Clock\ Frequncy}$$

$$= (1 + 6) \times \frac{1}{1MHz} = 7\mu s$$

Recap

- Oscillators
  - Internal and external
  - PLL
- Clock distribution and prescalars
- Timers
  - Capture and compare
- Pulse Width Modulation (PWM) Generation and Timer Modes
- SysTick Timer

Acknowledgements

- These lecture slides were adapted from the University of Michigan's EECS 373 course (many thanks to Alanson Sample!)

- These lecture slides also leverage materials from the following books:
  - Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C
    - Yifeng Zhu
  - The Definitive Guide to ARM Cortex – M3 and Cortex – M4 Processors
    - Joseph Yiu
  - Introduction to ARM Cortex – M Microcontrollers, Embedded Systems
    - Jonathan W. Valvano

Virtual Timers

You may find that you need more timers!

How can we use software to deal with this?

>> 60 second brainstorm <<

Simple Idea

- Let's say we have several events that we want to generate
- Create a list and set the timer to go off for the *first* one
- Do that first task, change the timer to interrupt for the next task

