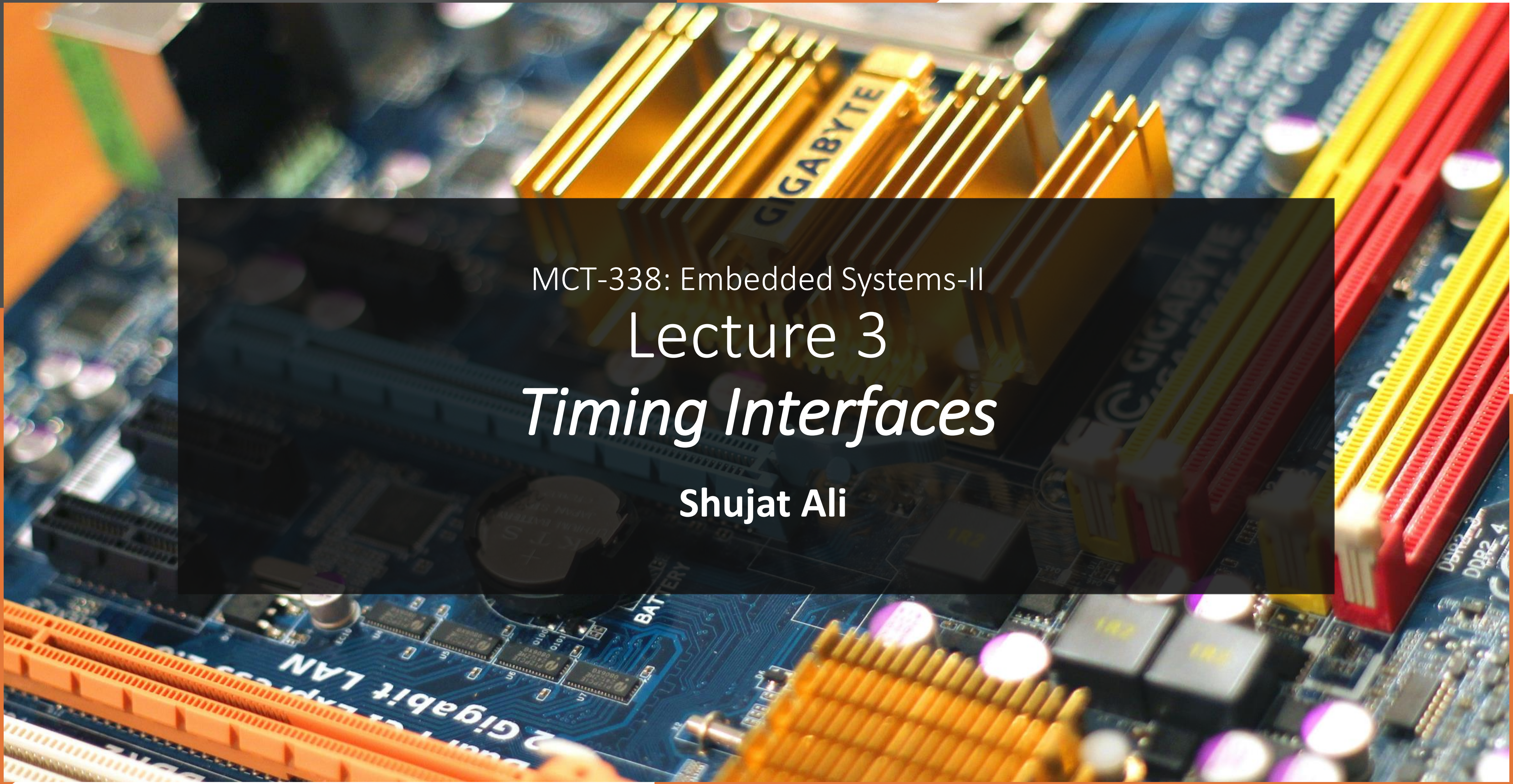
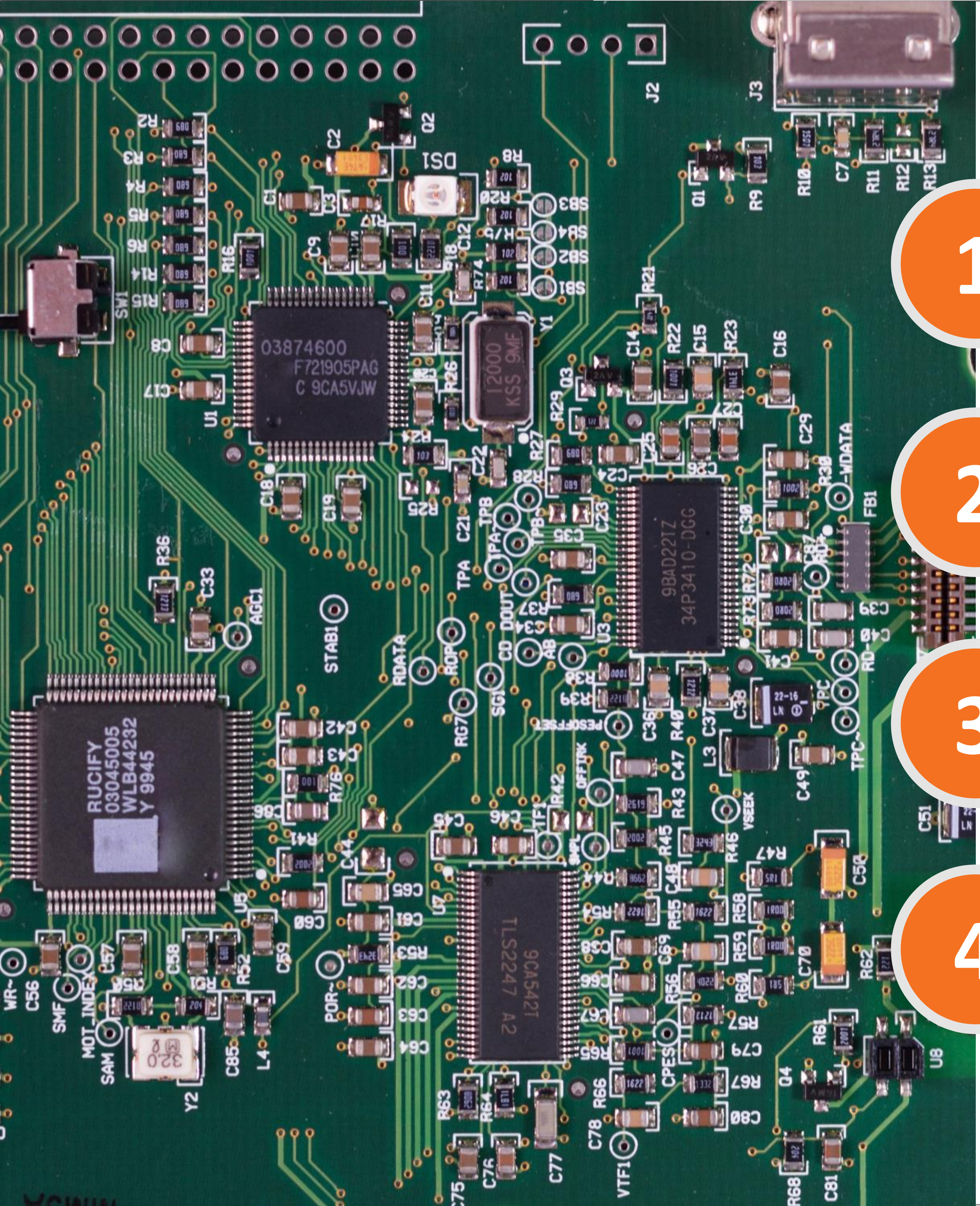


MCT-338: Embedded Systems-II

Lecture 3

Timing Interfaces

Shujat Ali



1 BASICS OF TIMING INTERFACES

2 CLOCKING A MICROCONTROLLER

3 TM4C123 CLOCK CONFIGURATION

4 TIMER FUNDAMENTALS

BASICS OF TIMING INTERFACES

Types of timing interfaces and their applications

Types of Timing Interfaces

BASICS OF TIMING INTERFACES

Each microcontroller has two types of timing interfaces.

1. Timing interface is used to generate the system clock to operate the microcontroller at desired frequency.
2. Timing interface based on a timer, which is primarily a peripheral module.
    - One primary use of a timer module is to generate timing intervals of desired value.
    - In addition, the timer modules can be configured to operate either as an input or an output device.
    - The timer module can be configured to time or count external events (used as input device)
    - The timer module can generate signals of varying duty cycle as well as frequency (used as output device)

Applications of Timing Interfaces

BASICS OF TIMING INTERFACES

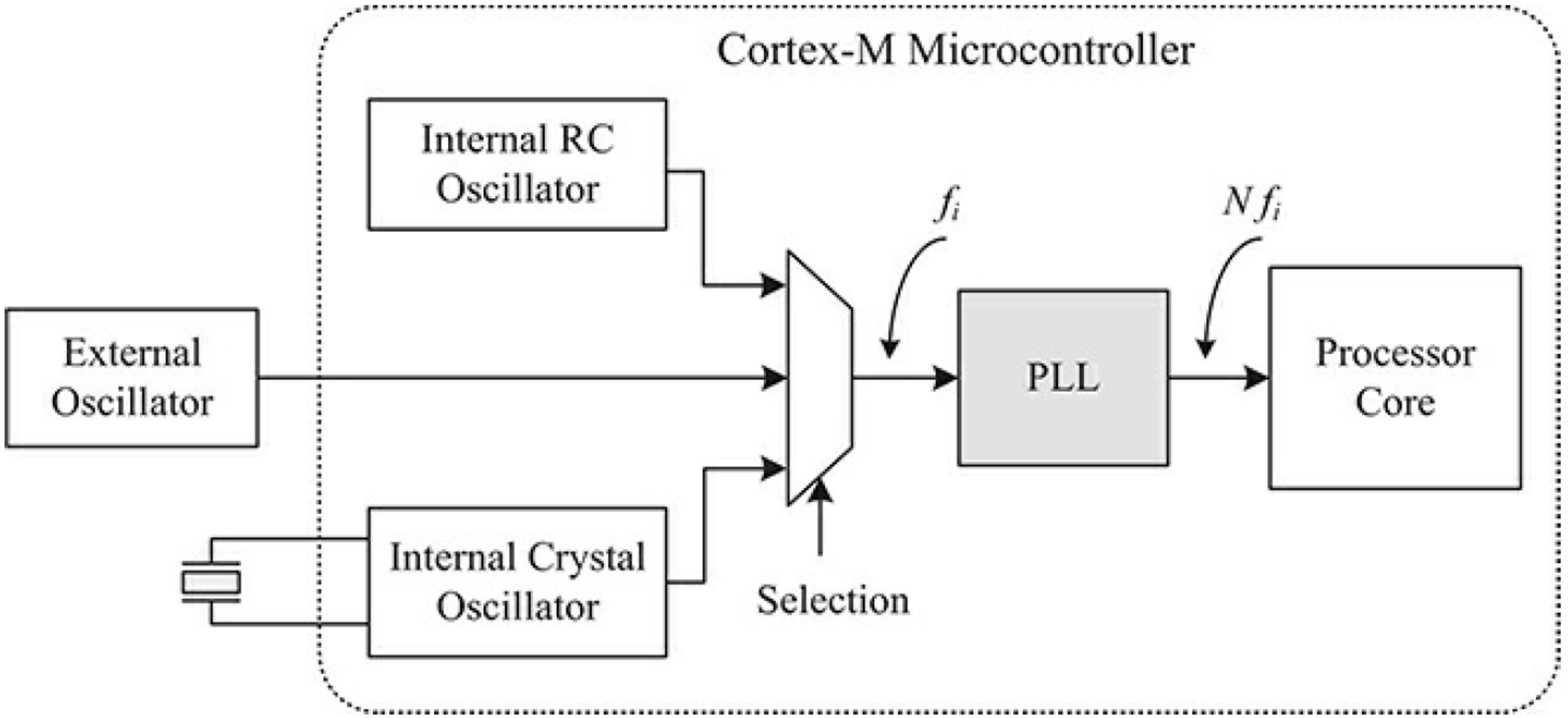
- At the basic level, the timer can be used to generate accurate timing signals, to measure time intervals, to generate interrupts at specific time intervals as well as to count events of interest.
- However, many complex applications are build using these basic timer as fundamental building blocks. Some key application areas are listed below:
    - Periodic data sampling
    - Scheduling of different tasks
    - Data communication rate control
    - Motion control
    - Navigation

CLOCKING A MICROCONTROLLER

Oscillators, their types, and PLL

Cortex-M Clock Sources

CLOCKING A MICROCONTROLLER



Cortex-M Clock Sources

CLOCKING A MICROCONTROLLER

An oscillator is used to generate clock signal for the microcontroller. Following are two most commonly used types of oscillator:

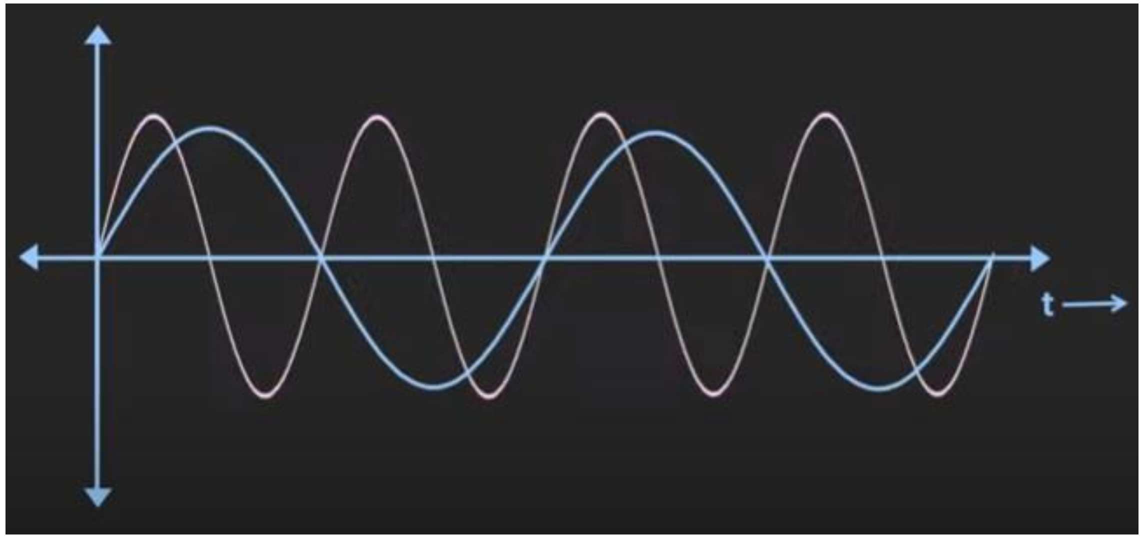
1. **Crystal Oscillator**: provide high frequency accuracy and low frequency variation with respect to temperature.
2. **RC Oscillators**: provide fast startup and are relatively cheaper, but they suffer from poor frequency accuracy in the presence of variations in supply voltage as well as temperature.

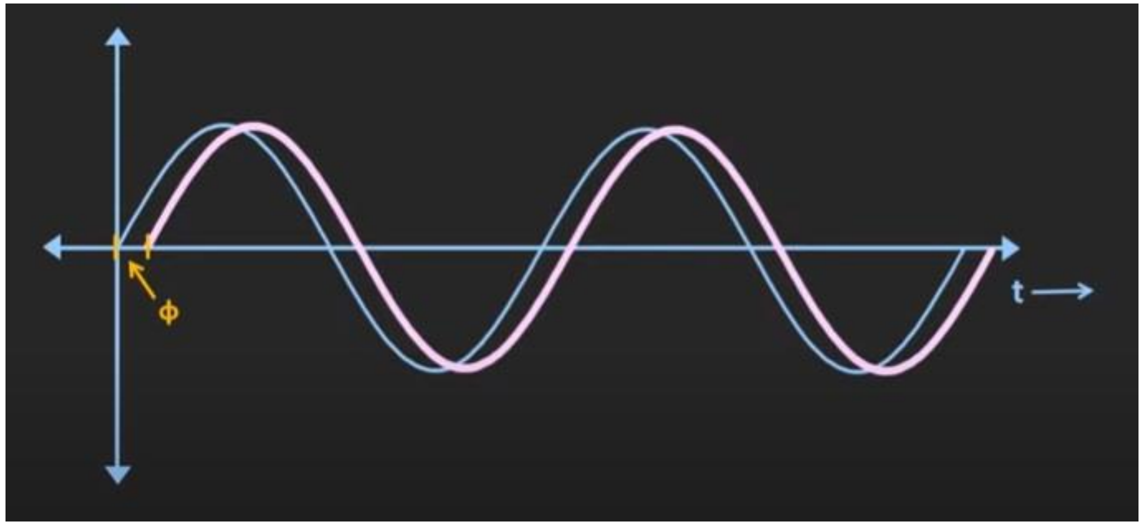
- Typically, microcontrollers have one or more internal integrated oscillators of fixed frequencies
- When a microcontroller is required to operate on different range of frequencies, an external crystal or RC oscillator can be used to select the desired frequency
- An external oscillator can also be used to obtain higher frequency accuracy by making the clock source independent of any temperature variations of the microcontroller chip.

Phase Lock Loop (PLL)

CLOCKING A MICROCONTROLLER

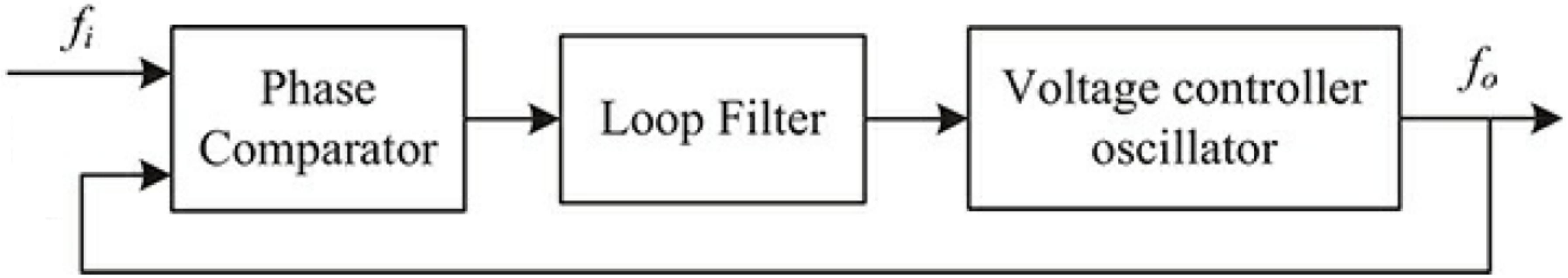
- PLL is a device used to track the phase of an input signal for a range of frequencies.
- By tracking of phase we mean that the phase difference of PLL output and incoming signal is constant, provided the frequency of incoming signal does not change.
- When the PLL output tracks input signal perfectly, the corresponding frequency difference between incoming signal and the PLL output is zero.





Phase Lock Loop (PLL)

CLOCKING A MICROCONTROLLER



In general, PLL comprises of the following three basic building blocks.

1. **Phase Detector** (PD): generates an error signal proportional to the phase difference by comparing the phases of two inputs (i.e., f_i and f_o)
2. **Low Pass Filter** (LPF): removes the high frequency noise in the output of PD and controls the dynamic characteristics such as bandwidth, transient response, and capture and lock ranges (frequency range in which PLL acquires phase lock)
3. **Voltage Controlled Oscillator** (VCO): generates an output frequency that is proportional to it's input voltage

Phase Lock Loop (PLL)

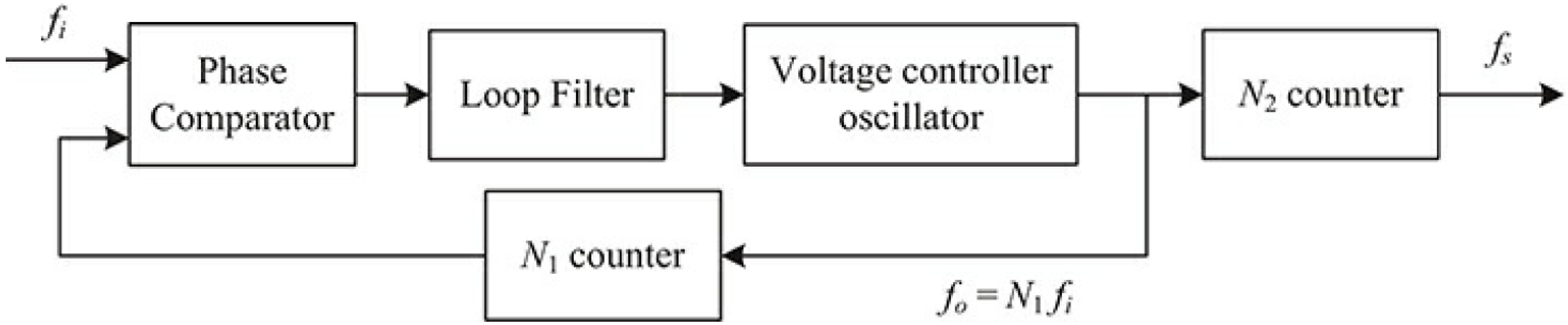
CLOCKING A MICROCONTROLLER

- When PLL is tracking the phase (by keeping the phase difference constant), then PLL is considered in lock condition.
- Under lock condition, the PLL tracks the incoming signal and any variations in the input signal frequency are followed by the PLL output (i.e., VCO output).
- In addition, the signal frequencies at the two inputs of phase detector are exactly same under lock condition.

PLL as a Frequency Multiplier

CLOCKING A MICROCONTROLLER

- If we introduce a simple N1 counter in the feedback path of VCO to PD the signal from VCO is divided in its frequency by some integer N_1 before being compared at PD
- For this scenario, the PLL under lock condition will ensure that the frequencies of the two PD input signals are exactly same and the VCO output frequency is N_1 times higher than the input signal frequency.
- This will only produce an integer frequency multiplication factor equal to N_1
- A non-integer frequency multiplication factor can be implemented by using 2nd integer counter N_2 at the output of VCO. The final output frequency will be given by $\boldsymbol{f_s = N_1/N_2\, f_i}$



TM4C123 CLOCK CONFIGURATION

Clock Source Selection & Frequency Value by using RCC & RCC2 Configuration Registers

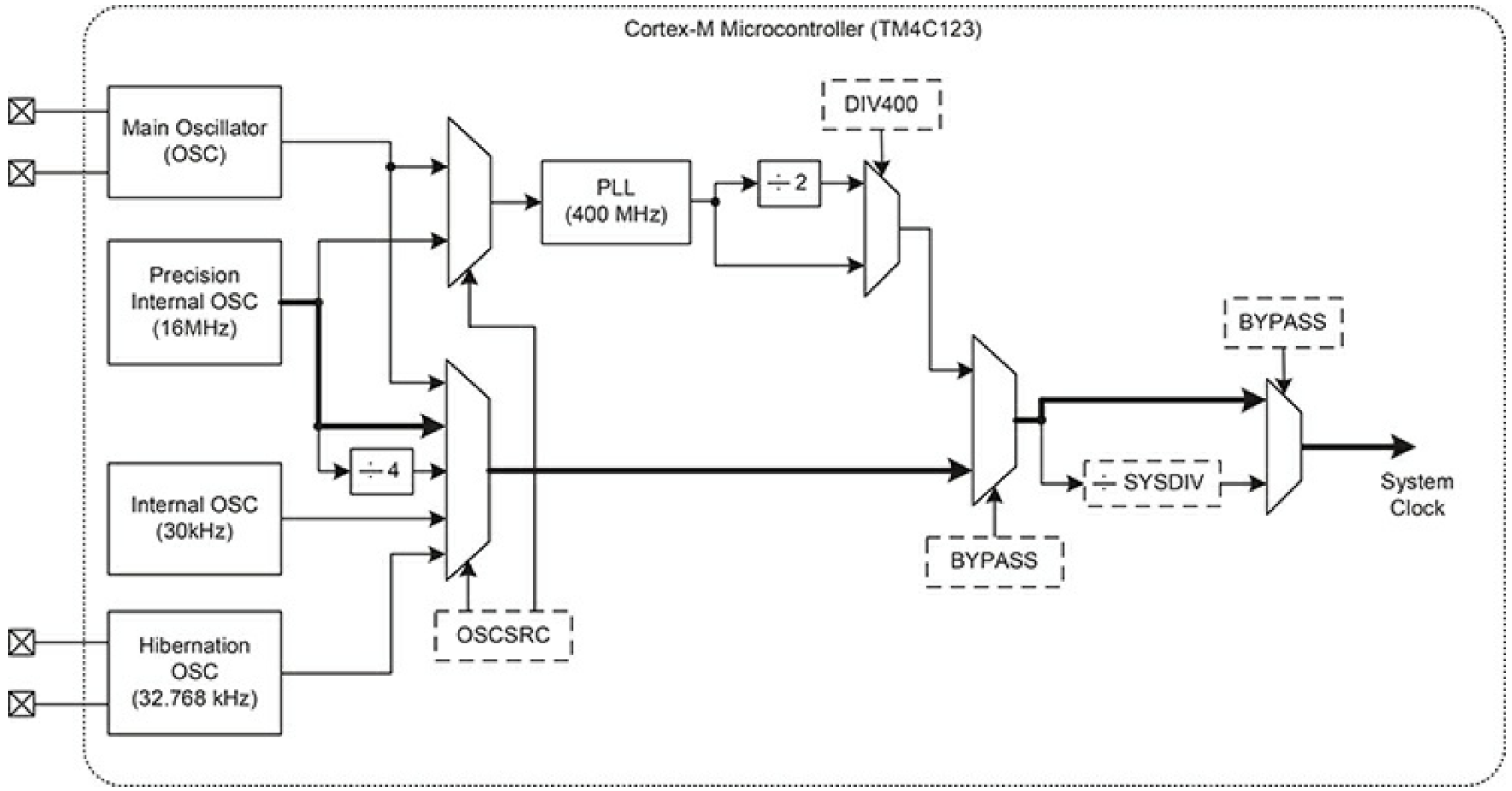
Why do we need Clock Configuration?

TM4C123 CLOCK CONFIGURATION

- The clock configuration for Cortex-M microcontroller has two associated components:
    1. Clock source selection
    2. Clock frequency value
- Every microcontroller when powered, starts with the default clock source and frequency.
- If default configuration is not sufficient then clock configuration is performed to run the processor at the desired operating frequency.
- Furthermore, there are situations where the processor operating frequency is dependent on the current operating state and is reconfigured for power conservation as well as heat dissipation management.

Default Clock Source and Frequency Configuration

TM4C123 CLOCK CONFIGURATION



The dashed line boxes represent the configurable bit-fields in RCC and RCC2 registers. The default values of these configurable bit-fields, after reset, are responsible for selecting the clock source and default system clock frequency.

RCC and RCC2 Configuration Registers

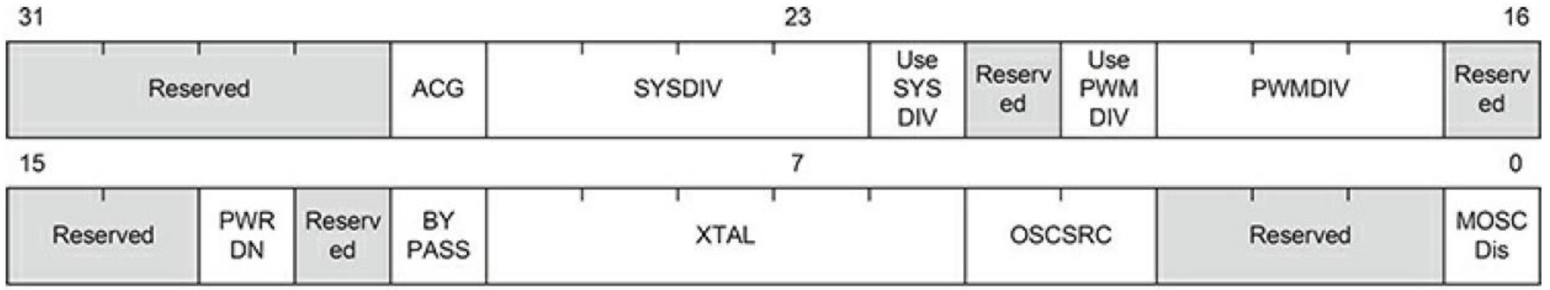
TM4C123 CLOCK CONFIGURATION

The processor clock source and frequency configuration can be performed using following clock configuration registers.

Run-mode Clock Configuration (RCC): Configure system clock and oscillators

Register address : 0x400FE060

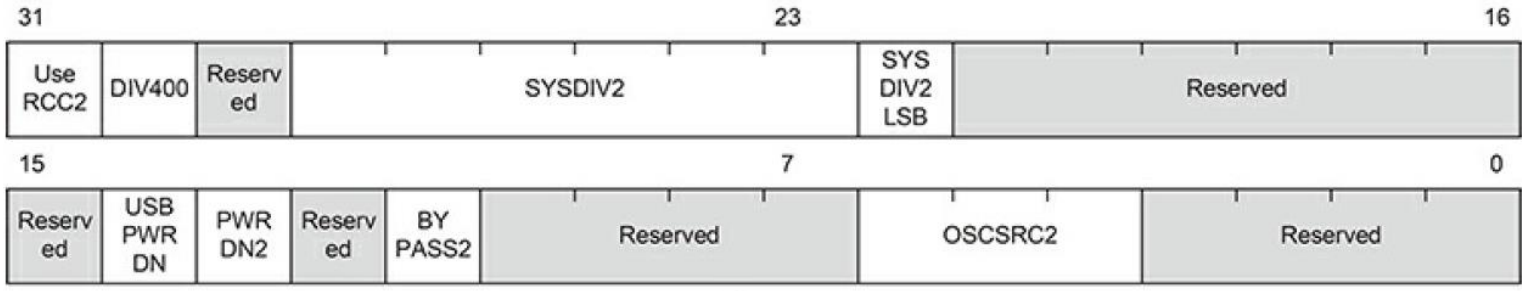
Reset value: 0x078E3AD1



Run-mode Clock Configuration 2 (RCC2): Configure clock and overrides RCC equivalent fields

Register address : 0x400FE070

Reset value: 0x07C06810



RCC and RCC2 Configuration Registers

TM4C123 CLOCK CONFIGURATION

Bit Field	Description
OSCSRC (RCC) OSCSRC2 (RCC2)	Oscillator source selection, with default value of 0x1 that selects PIOSC. Setting to 0x0 selects main oscillator, while setting to 0x3 selects low frequency internal oscillator. OSCSRC2 overrides the OSCSRC when USERCC2 bit is set in RCC2 configuration register. For OSCSRC2, it is possible to configure a value of 0x7 to select a 32.768 k Hz external oscillator.
SYSDIV (RCC) SYSDIV2 (RCC2)	System clock divisor, used to specify divisor value to generate system clock. SYSDIV2 overrides SYSDIV when both USESYSDIV bit in RCC register and USERCC2 bit in RCC2 register is set to 1.
USESYSDIV (RCC)	This bit when set to 1, enables the use of system clock divider. The divider is forced to be used when PLL is selected as the source. If USERCC2 bit in RCC2 register is set, then the corresponding SYSDIV2 field in RCC2 register is used as the system clock divider rather than the SYSDIV field in this register.
MOSCDIS	This bit when set to 1, disables the main oscillator.
USERCC2	When this bit is set, the RCC2 bit fields override corresponding the RCC register bit fields.

RCC and RCC2 Configuration Registers

TM4C123 CLOCK CONFIGURATION

Bit Field	Description
PWRDN (RCC) PWRDN2 (RCC2)	When this bit is 1, the PLL is powered down. PWRDN2 overrides the PWRDN when USERCC2 bit in RCC2 register is set. Clearing this bit turn on the PLL.
BYPASS (RCC) BYPASS2 (RCC2)	When this bit set to 1, the system clock is derived from oscillator directly by bypassing the PLL. BYPASS2 overrides the BYPASS when USERCC2 bit in RCC2 register is set.
DIV400	This bit filed selects the output of PLL 400 MHz (when set) or 200 MHz (when clear). When this bit is 0, SYSDIV2 is used as clock divisor and 200 MHz pre-divided PLL output is used.
XTAL	Used to specify crystal value, attached to the main oscillator, from 0x05 to 0x1A with corresponding crystal frequencies ranging from 4 MHz to 25 MHz.
SYSDIV2LSB	This bit becomes the LSB of SYSDIV2 when DIV400 is set to 1. By appending SYSDIV2LSB bit to the SYSDIV2 field provides a 7 bit divisor and uses 400 MHz PLL output.

Default Clock Configuration

TM4C123 CLOCK CONFIGURATION

- Run-mode Clock Configuration (RCC): Reset value: **0x078E3AD1**
- Run-mode Clock Configuration 2 (RCC2): Reset value: **0x07C06810**

- Based on the default values of RCC and RCC2 registers, we can observe that
  - The USERCC2 bit field is cleared and as a result only register RCC is involved in the clock configuration
  - From register RCC, default value of bit field OSCSRC is 0x1, which means PIOSC is selected as clock source
  - The default value of BYPASS field in RCC register is 0x1, which bypasses the use of PLL as well as the system clock divisor (i.e., use of SYSDIV)
- Based on these default values the 16 MHz clock from the precision internal oscillator becomes the system clock frequency and is the default clock configuration

Clock Frequency Configuration – 80 MHz

TM4C123 CLOCK CONFIGURATION

- Many different clock frequencies can be configured for system clock, depending on the application requirements.
- Maximum permissible operating frequency for TM4C123 microcontroller is 80 MHz which can be achieved as follows.
  - Select 400 MHz as the PLL output by setting DIV400 bit
  - Use SYSDIV2 equal to 5 to get desired clock frequency (i.e., 400/5 = 80 MHz)
- It should be remembered that in this configuration, SYSDIV2LSB is appended to SYSDIV2 as LSB to construct a 7 bit divisor.

Clock Frequency Configuration – 80 MHz

TM4C123 CLOCK CONFIGURATION

Following are the key steps performed for successful configuration of system clock.

- Bypass the PLL and system clock divider by setting the BYPASS bit and clearing the USESYSDIV bit in the RCC register. If USERCC2 bit is set in register RCC2, then BYPASS2 bit should also be set. It is important to write the RCC register prior to writing the RCC2 register. Doing so will allow us to configure the PLL and validate its operation, while running the microcontroller from an oscillator directly before switching it to PLL.
- Select the oscillator source by configuring OSCSRC (OSCSRC2), the crystal value using bit field XTAL and clear the PWRDN (PWRDN2) bit in RCC (RCC2). Setting the XTAL field automatically pulls a valid PLL configuration data for the selected crystal, while clearing the PWRDN (PWRDN2) bit powers the PLL and enables its output.
- Select desired system divider SYSDIV (SYSDIV2 and SYSDIV2LSB) in RCC (RCC2) and set USESYSDIV bit in RCC. The SYSDIV (SYSDIV2 and SYSDIV2LSB) field determines system clock frequency for microcontroller.
- Wait for the PLL to lock by polling the PLLLRIS bit in the Raw Interrupt Status (RIS) register.
- Enable use of PLL by clearing the BYPASS (BYPASS2) bit in RCC (RCC2).

TIMER FUNDAMENTALS

Counter, Clock Signal, Overflow, Reload, Interrupt, Pre-scaler

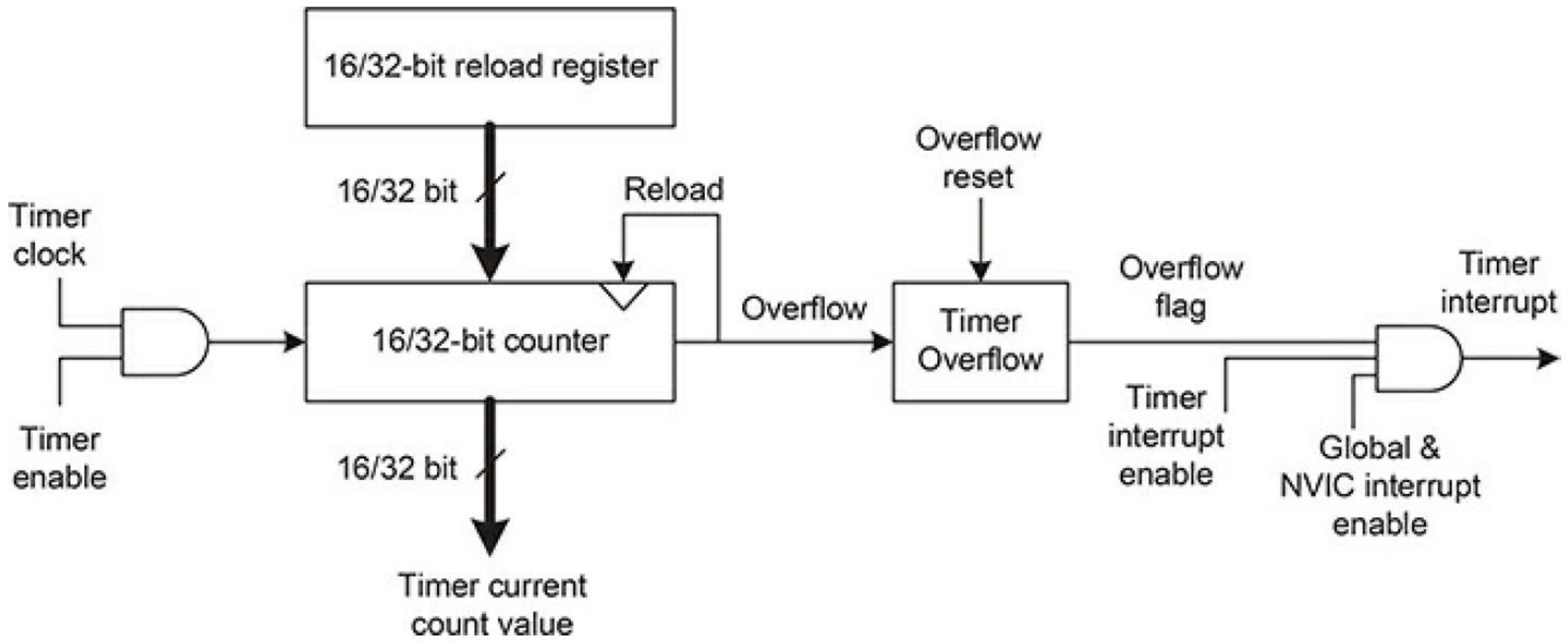
Basic Concepts

TIMER FUNDAMENTALS

- The basic timer module comprise of an up (down) **counter** with an associated counter overflow (underflow) detection capability
- When enabled, an input **clock signal** is applied to the counter, which increments (decrements) counter value.
- The **timer size** or **timer width** is the number of bits allocated for its counter
- When an up-counter timer reaches its maximum possible value, timer **overflow** occurs, and it also sets a timer **overflow flag**
- After timer overflow counter value rolls over and is loaded either with 0 or with a user specified value from the **timer reload register** to start count again
- The user can either poll this flag bit to determine the occurrence of an overflow or a **timer interrupt** (when configured) can be generated by this overflow flag.
- The user can also measure time elapsed since the occurrence of previous overflow, by reading **current timer value**.

Basic Timer Block Diagram

TIMER FUNDAMENTALS



Timer Interval Evaluation

TIMER FUNDAMENTALS

Consider a 16-bit count-down timer with 0xFFFF reload value (***x***). If the input clock frequency is 4 MHz (***f***), then timer overflow interval (**Δt**) will be

$$\Delta t = \text{Timer overflow interval} = \frac{\text{Counter reload value}}{\text{Input signal clock frequency}} = \frac{x}{f} = \frac{2^{16}}{4\times 10^{6}} = 16.384\ ms$$

What if we want to generate a delay of 10 ms, using the same timer module? There are two possible choices:

- <u>Option 1</u>: Change the frequency of input clock signal to 6.5536 MHz
- <u>Option 2</u>: Change the counter reload value to 0x9C40 rather than 0xFFFF

Timer Reload Value

TIMER FUNDAMENTALS

Consider the scenario, where we are interested in generating a timer interrupt every ***Δt*** seconds. If the timer width is **w**-bit and it is operating at a frequency, ***f*** Hz with count down configuration, then the reload value $\boldsymbol{x_d}$, when the timer underflows after reaching 0x0000, is obtained as

$$x_d = f \times \Delta t$$

However, when the timer is counting up, it overflows after reaching 0xFFFF and the corresponding reload value $\boldsymbol{x_u}$ is obtained using the following modified expression

$$x_u = 2^w - f \times \Delta t$$

For the above expressions to be valid, both $\boldsymbol{x_d}$ and $\boldsymbol{x_u}$ are upper bounded by 2^w.

Timer Resolution and Range

TIMER FUNDAMENTALS

- **Timer resolution** is the smallest time interval that can be measured by the timer.
- For a given resolution, **timer range** determines the maximum interval, a timer can measure without overflowing.

- For example, a 16-bit timer can accumulate 2^{16} number of clock cycles, which also determines the timer range for a given timer clock frequency.
- Since counter value changes by +1 or -1, when time elapsed is equal to one clock cycle, hence timer resolution is simply equal to time period of input clock given by *j*, where *f* is the frequency of timer input clock.

Need of Timer Prescaler

TIMER FUNDAMENTALS

Timer range and resolution are inter-connected, e.g., increasing timer range will require decreasing timer resolution.

- For a 16-bit timer with 1 µs resolution (i.e., timer clock is 1MHz), the maximum time interval (i.e., timer range) that can be measured is 65.536 ms.
- If 16-bit timer range is required to be 1 s the corresponding resolution will be approximately 16 us.

This tradeoff, between timer resolution and range, can be achieved using timer prescaler.

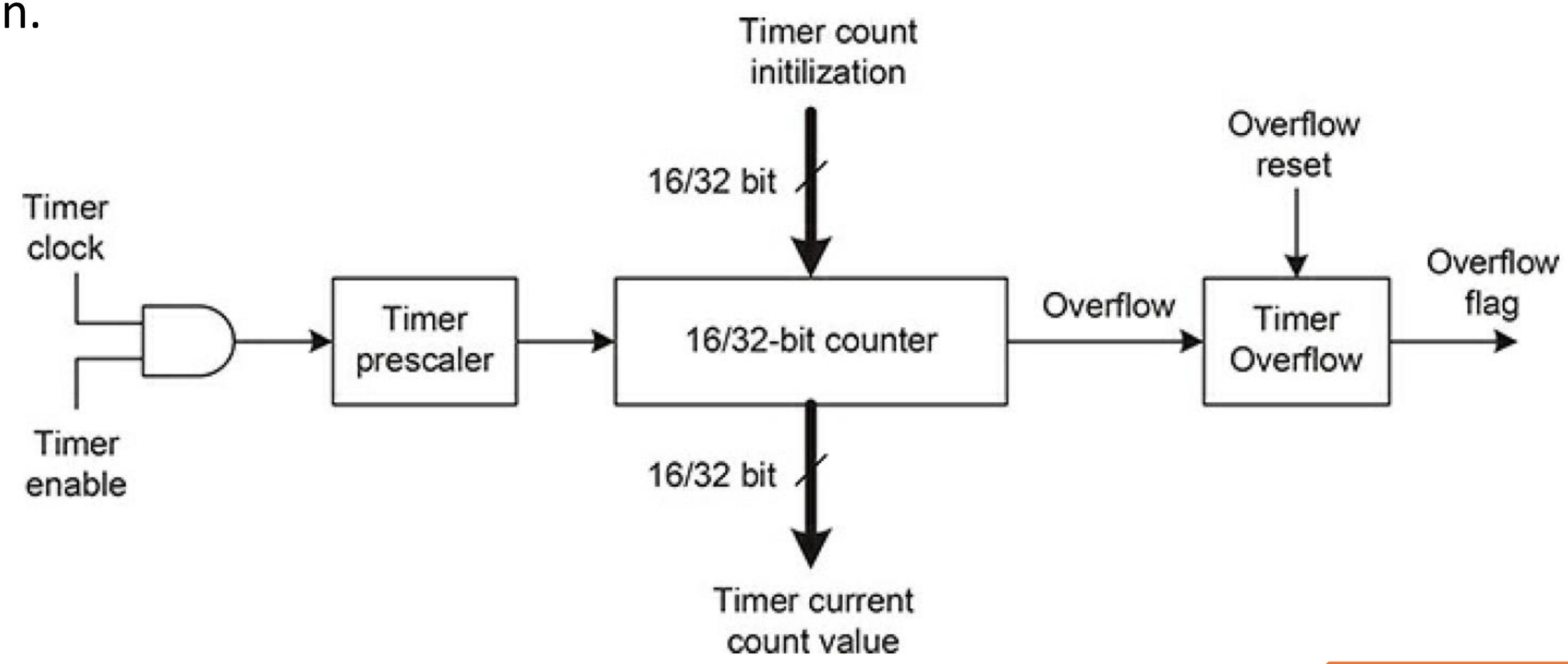
To extend timer range, while maintaining its resolution, a possible solution is to extend the counter size.

- For instance, using a 32-bit timer, provides a range of 4295 s at 1 µs resolution, or alternatively 42.95 s at 0.01 µs resolution.

Timer Prescaler

TIMER FUNDAMENTALS

A **prescaler** divides the frequency of timer input clock signal before feeding it to the counter. This scaling increases the timer operating range or the maximum time interval that it can generate using timer overflow. However, using a prescaler reduces the timer resolution.



THANK YOU

Any Questions???