



AMERICAN INTERNATIONAL UNIVERSITY – BANGLADESH (AIUB)

Where leaders are created



Introduction to Timers

What is a Timer?

- A **timer/counter** is like a **clock**, and can be used to measure time events.
- The timer can be programmed by some special registers.
- The controller of the Arduino is the ATmega328, which has **3 timers**, called **Timer0**, **Timer1** and **Timer2**.

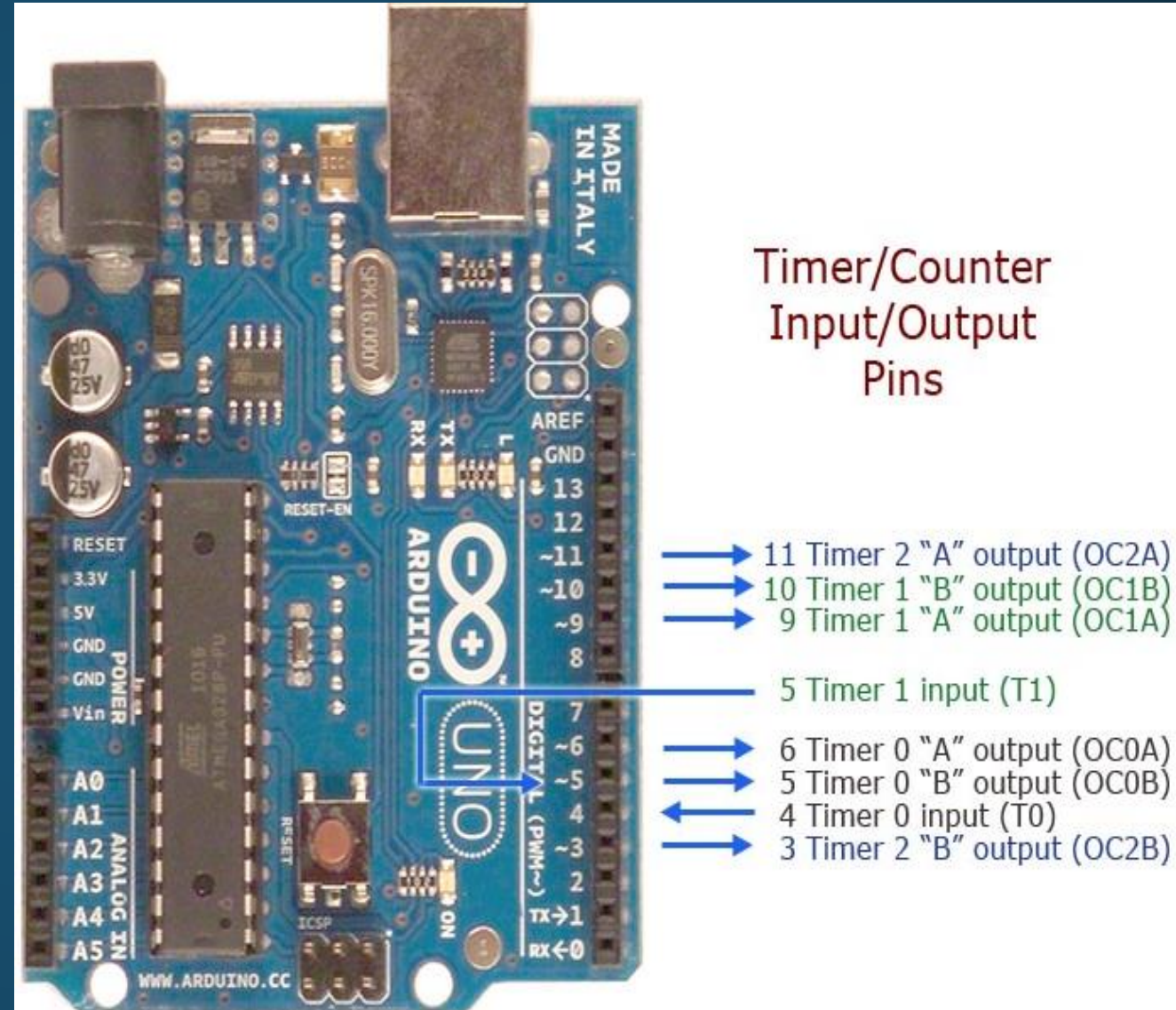


Timer/Counter Input/Output Pins

- 11 Timer 2 "A" output (OC2A)
- 10 Timer 1 "B" output (OC1B)
- 9 Timer 1 "A" output (OC1A)
- 5 Timer 1 input (T1)
- 6 Timer 0 "A" output (OC0A)
- 5 Timer 0 "B" output (OC0B)
- 4 Timer 0 input (T0)
- 3 Timer 2 "B" output (OC2B)

What is a Timer?

- Timer0 and Timer2 are 8-bit timers, whereas Timer1 is a 16-bit timer. The most important difference between 8-bit and 16-bit timers is the **timer resolution**.
- 8-bit timer is capable of counting $2^8 = 256$ steps from 0 to 255. 16-bit timer is capable of counting $2^{16} = 65536$ steps from 0 to 65535.



Usefulness of Timer :

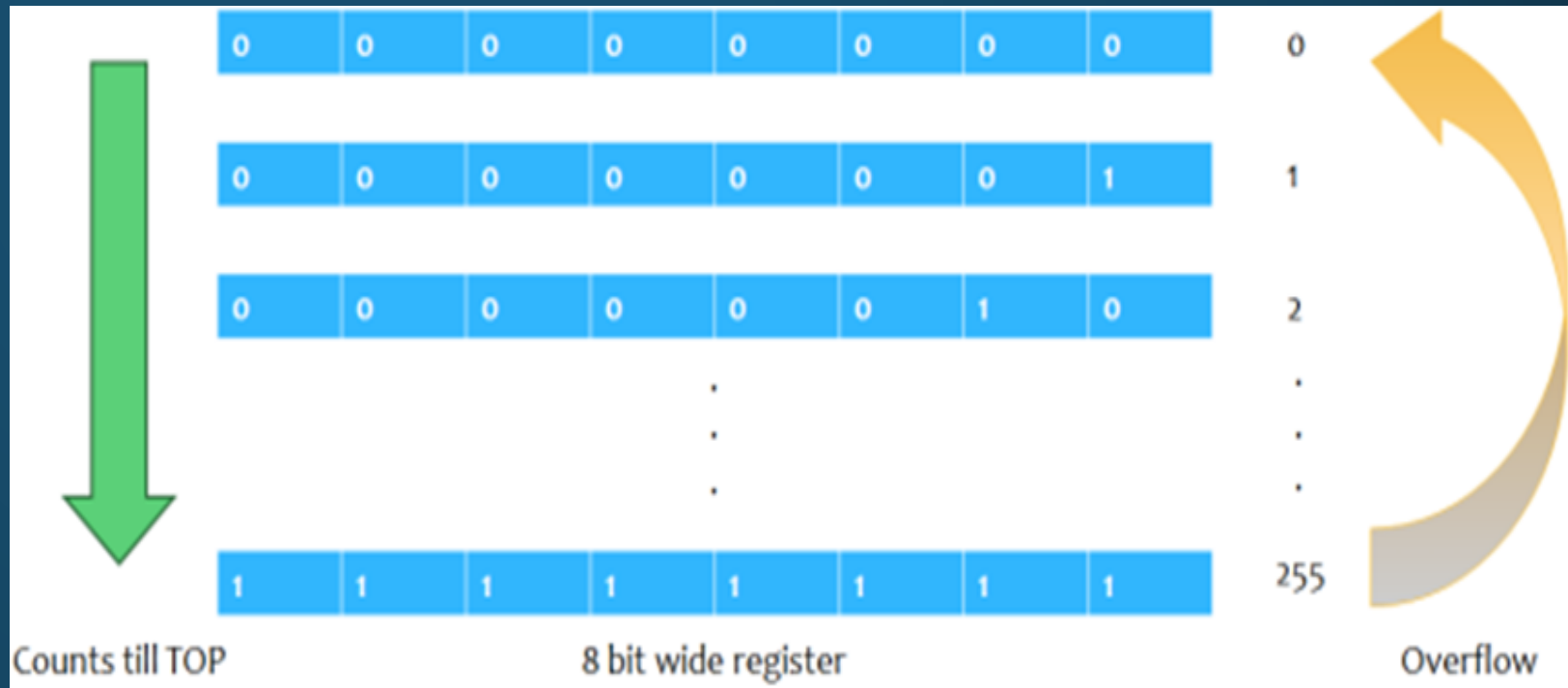
- Timer is an important concept in the field of electronics. Every electronic component of a sequential logic circuit works on a time base to keep all the work synchronized.
- An **advantage** of the **timer** is that it is totally **independent of the CPU**. Thus, it runs parallel to the CPU and there is no CPU's intervention, which makes the timer quite accurate.
- This is why **using a timer is preferred for long delays** instead of simply using the **delay() function**. The **drawback of delay()** is that your loop gets halted, and functions above and below the delay() are not being executed during this interval.

Usefulness of Timer :

- A timer approach is a little **harder to implement** but the **main loop keeps executing** and only **excludes the code and functions which a programmer wants to exclude**.
- If the programmer needs **multiple tasks to occur** at the same time or if the programmers' application requires that you constantly read/save data from inputs, **use of the delay() function should be avoided**.

Timer Basics: Overflow

- Due to this counting feature, **timers are also known as counters**. Once they reach their maximum possible value, the program does not stop executing and the timer count simply returns to its initial value of zero. In such a situation, we say that the timer/counter **overflows**.
- For example, Timer0 is an 8-bit timer, meaning that it can count up from zero to 2^8-1 , or 255. Once that number is reached, the count resets back to zero and starts counting again.



How Timer Works?

- Timers on the Arduino count up from zero, **incrementing with every clock cycle** of the oscillating crystal that drives Arduino.
- If **smaller frequencies are necessary**, it is possible to “divide” the clock through an approach called **pre-scaling**.
- How quickly timer reaches the target count depends on the **clock divider**. With no divider, the clock would go through 16 million cycles per second (16 MHz), and would overflow and reset this counter many times per second.
- The three timers (called Timer0, Timer1, and Timer2) of Atmega328 can either be operated in one of these three modes, called normal mode, CTC mode, or PWM mode.
- For lab experiment 3, we will operate it in normal mode (counter) using **Timer0**.

Timer Basics: Registers (Timer0)

- The timer can be programmed by some special registers, where we can **configure the pre-scaler** for the timer, or the **mode of operation** and many other things necessary for proper operation.
- The registers of interest for our purposes are:
 - Timer/Counter Register – **TCNT0**: to store timer count
 - Timer/Counter Control Register – **TCCR0A** and **TCCR0B**: to define operation mode and pre-scaler
 - Timer/Counter Interrupt Flag Register– **TIFR0**: to observe if there is any overflow
 - Output Compare Register - **OCR0A** and **OCR0B**: to match the timer count with some custom value (for CTC or PWM mode, not needed for normal mode)

Timer Basics: Registers (Timer0)

- **Timer/Counter Control Register A for Timer0 – TCCR0A:**
The bits **WGM02** (from TCCRB), **WGM01** and **WGM00** decide which mode the timer will run on.

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Timer/Counter Control Register B for Timer0 – TCCR0B:**
The three **Clock Select bits- CS02, CS01, CS00** select the clock source and pre-scalar value to be used by the Timer/Counter

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer Basics: Registers (Timer0)

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{IO}}/(\text{No prescaling})$
0	1	0	$\text{clk}_{\text{IO}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{IO}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{IO}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{IO}}/1024$ (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Please note that if you do not initialize this register, all the bits will remain as zero and the timer/counter will remain stopped.

Timer Basics: Registers (Timer0)

- **Timer/Counter Register – TCNT0:**

- ✓ This is where the 8-bit counter of the timer resides.
- ✓ The value of the counter is stored here and increases or decreases automatically.
- ✓ Data can be both read or written from this register.
- ✓ The register resets to zero after each overflow.

Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer Basics: Registers (Timer0)

- **Timer/Counter Interrupt Flag Register– TIFR0:**
 - **TOV0** bit is set (one) whenever TIMERO0 overflows.

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Output Compare Register - OCR0A and OCR0B:**
 - Both Output Compare Registers A and B contain 8-bit values that are continuously compared with the counter value (TCNT0).
 - **The OCF0A or OCF0B bit in the TIFR register is set (one) whenever the count matches the stored value.**

Bit	7	6	5	4	3	2	1	0	
0x28 (0x48)	OCR0B[7:0]								OCR0B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer Basics: Registers (Timer1)

- Most of the registers are very similar to Timer0.
- **Timer/Counter Register – TCNT1H and TCNT1L (TCNT1):**
 - ✓ The main difference between Timer0 and Timer1 is in the timer/counter register.
 - ✓ The two Timer/Counter I/O locations (**TCNT1H and TCNT1L, combined TCNT1 of 16-bit**) give direct access, both for read and write operations, to the Timer/Counter unit **16-bit counter**.
 - ✓ The register resets to zero after each overflow.

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer Basics: Registers (Timer1)

- Most of the registers are very similar to Timer0.
- **Timer/Counter Register – TCNT1H and TCNT1L (TCNT1):**
 - The other differences include-
 - ✓ The output compare registers are also 16 bit and made up from two 8-bit registers.
 - ✓ There is an extra Input Capture Register.
 - ✓ There are several extra modes Timer1 can run on.

Timer Basics: Terminology

- Definitions of some commonly used terms in Timer:
- **BOTTOM**: The counter reaches the BOTTOM when it becomes 0x00.
- **MAX**: The counter reaches its maximum when it becomes 0xFF (decimal 255) in Timer0 and 0xFFFF (decimal 65535) in Timer1.
- **TOP**: The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value (MAX) or the value stored in the OCRnA or OCRnB Register. The assignment is dependent on the mode of operation.

Timer Modes: Normal Mode

- The simplest mode of operation is the Normal mode. In this mode, the counting direction is always **up (incrementing)**, and no counter clear is performed.
- The counter simply overflows when it passes its maximum value and then restarts from zero.
- In normal mode of operation, the Timer/Counter Overflow Flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero.

Timer Modes: Clear Timer on Compare, CTC Mode

- In **Clear Timer on Compare** or **CTC** mode, the counter is cleared to zero when the counter value (TCNT0) matches either a value stored in OCR0A register.
- When the desired value is reached, a flag in a status register is set to '1'.
- This method can be more efficient than **comparing bytes to detect a match** as **checking a single flag is simpler**.

Counter Unit

Red: signals entering/exiting a block
Green: description of a block

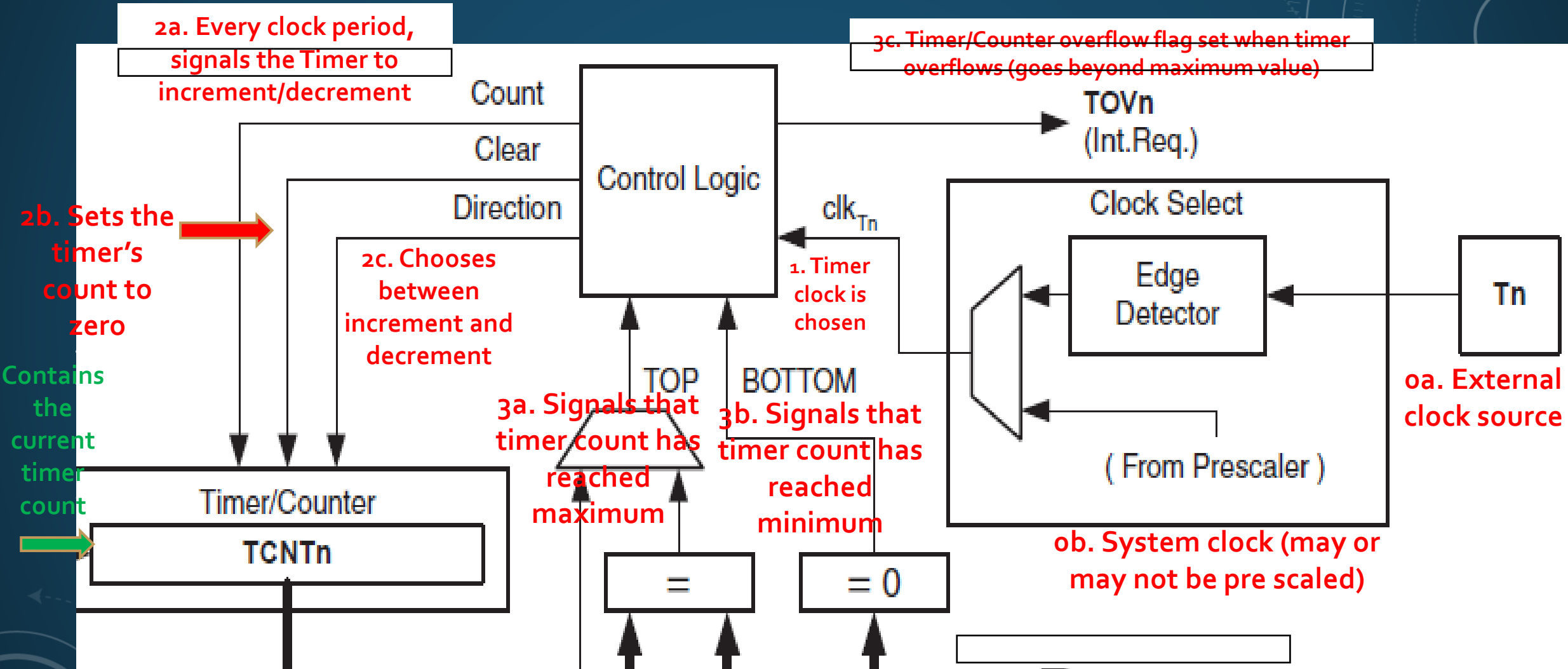
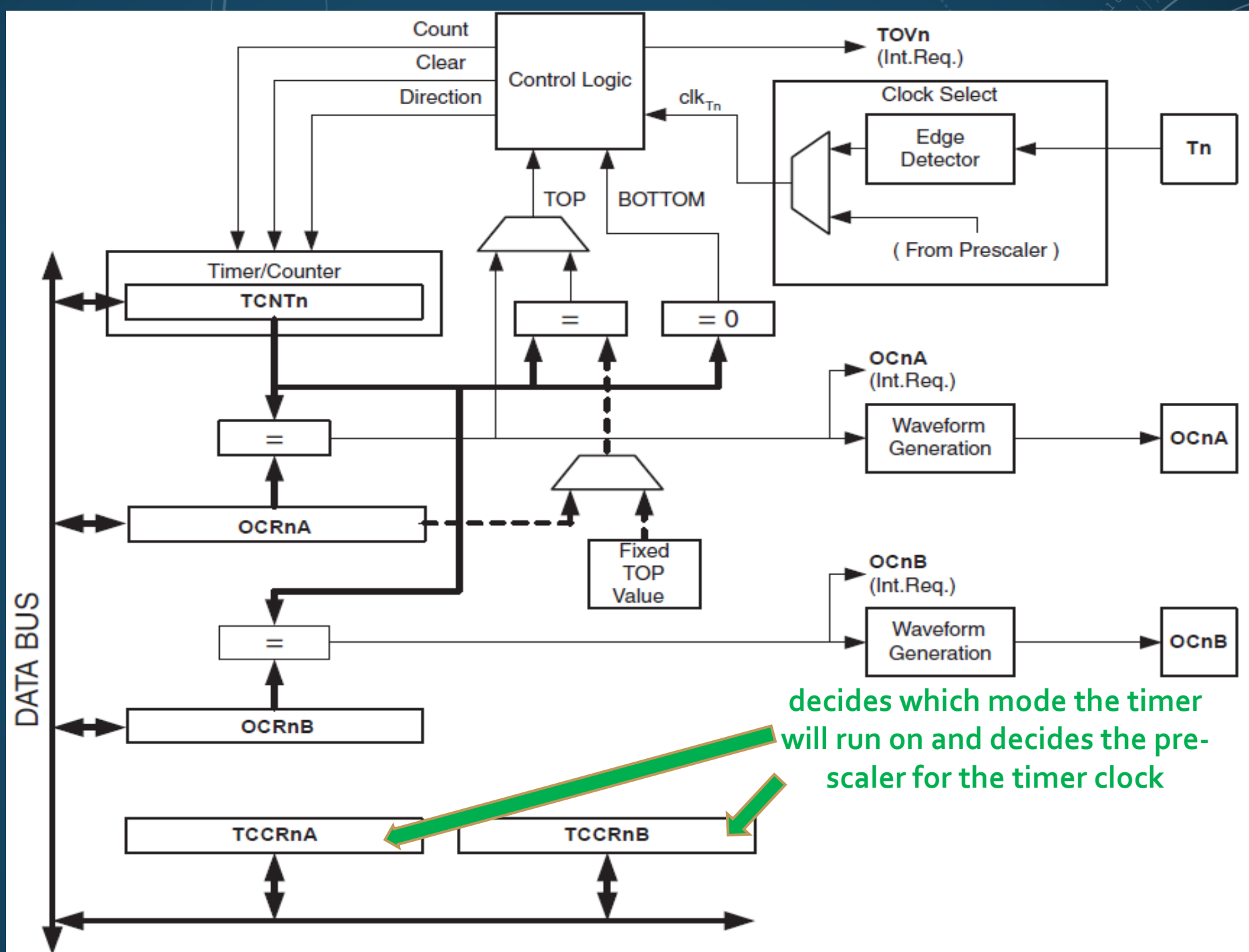


Diagram illustrating the flag set logic for Tn:

- Input: **Tn**
- Intermediate Output: **OCnA** (Flag set, enabled)
- Final Output: **OCnB** (Flag set, enabled if enabled)



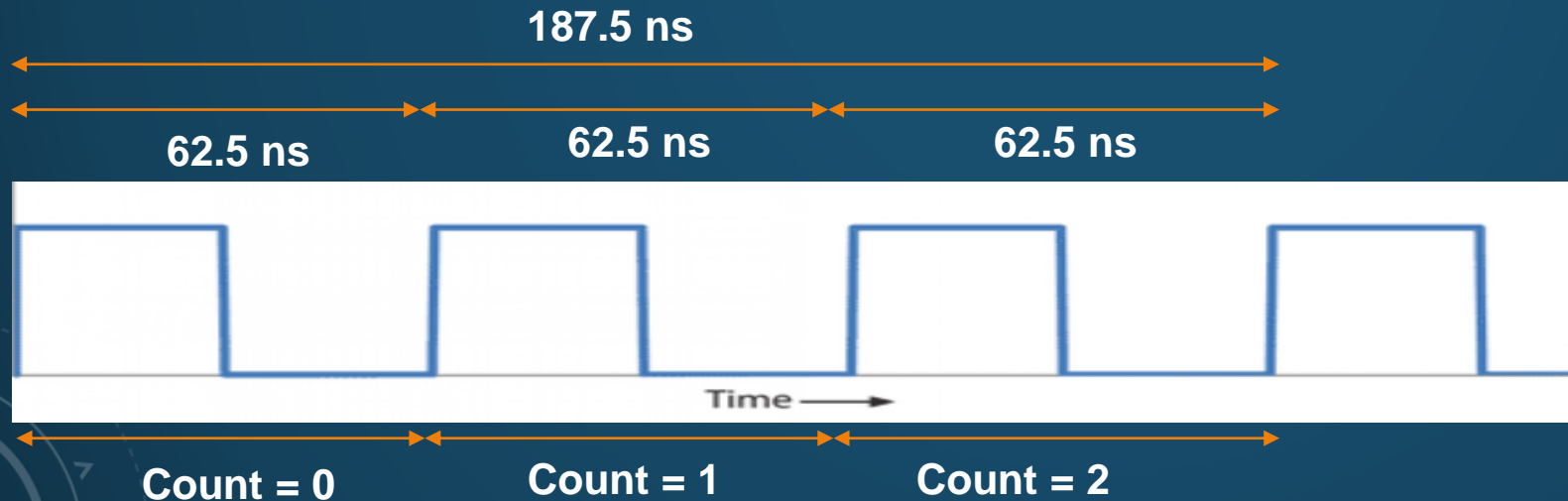
- Red: signals entering/exiting a block (Normal mode)
- Blue: signals entering/exiting a block (CTC/PWM mode)
- Green: description of a block



decides which mode the timer will run on and decides the pre-scaler for the timer clock

Timer for Delay: Calculating Count

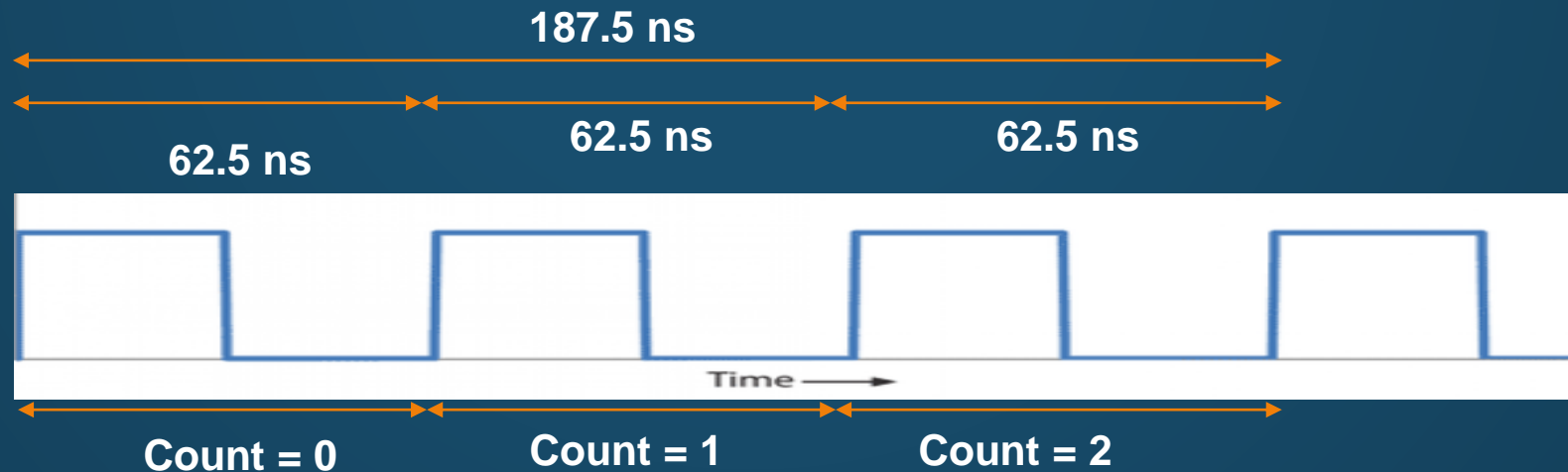
- Timer needs a clock pulse for each transition from one number to the next, so we want to establish a formula relating the necessary timer count for a specific delay with the timer clock period/frequency.
- For $F_{\text{CPU}} = 16 \text{ MHz}$, the system clock period = $1/16\text{MHz} = 62.5 \text{ ns}^*$. So, if the timer clock is the same as the system clock, it takes only **62.5 ns** for every transition (0 to 1, 1 to 2, etc.).



$$* T = \frac{1}{f}$$

Timer for Delay: Calculating Count

- We can see that 3 time periods are needed to count to 2, so timer count = (number of periods needed to reach the count) - 1.
- Also, we can see that to get a delay of 187.5 ns, we need 3 periods lasting 62.5 ns, so the number of periods needed = Required delay/Timer clock period.



Timer for Delay: Calculating Count

- Suppose, we need a delay of 1 ms. To get an idea of how long it takes, let's calculate the timer count from the following formula:

$$\text{Timer count} = \frac{\text{Required delay}}{\text{Timer clock period}} - 1$$

Here, required delay = 1 ms and timer clock period = 62.5 ns, so Timer Count = $(1\text{m}/62.5\text{n}) - 1 = 15,999$.

- So, the clock needs to **tick 15,999 times** to give a delay of only 1 ms.

Timer for Delay: Calculating Count

- Maximum possible delay for **Timer0** (which is an 8-bit timer) at 16 MHz : $62.5 \text{ ns} \times 256 = \mathbf{16 \text{ }\mu\text{s}}$
- Maximum possible delay for **Timer1** (which is a 16-bit timer) at 16 MHz: $62.5 \text{ ns} \times 65,536 = \mathbf{4.096 \text{ ms}}$
- If we plan to get delays simply by directly counting at system frequency, it is difficult to use an 8-bit timer (as it has an upper limit of 255, after which it overflows, and resets count to zero). Even with a 16-bit timer (which can count to 65,535), **it is not possible to get longer delays.**

Timer for Delay: Calculating Count

- To stay in the safe side, we use the highest available **pre-scalar** and reduce timer clock frequency to $16\text{MHz}/1024 = 15625\text{ Hz}$, with a **new timer clock period** (= System clock period \times pre-scalar) = $62.5\text{ns} \times 1024 = 64\text{ }\mu\text{s}$. Now, the needed timer count = $(1\text{ms}/64\text{ }\mu\text{s}) - 1 = 14.625$. Now that **Timer clock frequency = System clock frequency/pre-scalar**, we can update the equation

$$\text{Timer count} = \frac{\text{Required delay}}{\text{Timer clock period}} - 1$$

to

$$T = \frac{1}{\frac{f}{ps}} = \frac{ps}{f}$$

$$\text{Timer count} = \frac{\text{Required delay} \times \text{System clock frequency}}{\text{prescalar (ps)}} - 1$$

Timer for Delay: Calculating Count

- Maximum possible delay for **Timer0** at 15625 Hz: $64\ \mu\text{s} \times 256 = 16.384\ \text{ms}$
- Maximum possible delay for **Timer1** at 15625 Hz: $64\ \mu\text{s} \times 65536 = 4.194304\ \text{s}$
- To get longer delays, we will use Timer1, and for delays longer than 4 s, nested if statements can be used.

Problem Statement 1

Make an LED blink every **2 milliseconds** while using Arduino system frequency (F_CPU) 16 MHz, using timer to generate the delay without any application of `delay()` function. Delay = 2 ms, so Timer0 can be used with pre-scalar 1024. Number of count needed to reach 2 ms = $(2,000 \mu\text{s}/64 \mu\text{s}) - 1 = 30.25 \approx 31$.

```
#define PIN_USED 8 //define name of pins used

int milisec = 2; //define delay length in milliseconds
int prescalar = 1024; //define prescalar
int clock_freq = 16000000/prescalar; //calc timer clock freq
float clock_period = 1/(float)clock_freq; //calc timer period
int count = ((milisec*.001/clock_period)-1); //calc count for required delay

void setup() {
    //define pins connected to LEDs as outputs
    pinMode(PIN_USED, OUTPUT);

    //set up timer
    TCCR0A = 0b00000000;
    TCCR0B = 0b00000101; //setting prescaler for timer clock
    TCNT0=0;

}

void loop() {
    if(TCNT0 >= count) // Checking if delay time has passed
    {
        TCNT0=0;
        digitalWrite(PIN_USED, !digitalRead(PIN_USED)); //toggle pin output
    }
}
```

Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0			0	0	

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	WGM02	CS02	CS01	CS00	-	-	-	-	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0		0	0	

WGM01 and WGM00 set to zero for normal mode

WGM02 set to zero for normal mode, CS02, CS01, CS00 set to 1,0,1 for pre-scalar 1024

Make LED blink

Problem Statement 2

- Make an LED to blink every **2 seconds** while using Arduino system frequency (F_CPU) 16 MHz, using a timer to generate the delay without any application of delay() function.
- Delay = 2 s, so Timer1 (remember that it is a 16-bit timer) can be used with the pre-scalar value of 1024.
- Number of count required to get a delay of 2 s = $(2,000,000 \mu\text{s} / 64 \mu\text{s}) - 1 = 31,249$.

Code:

Same code, only register names have 1 instead of 0 and delay length changed!

```
#define PIN_USED 8 //define name of pins used

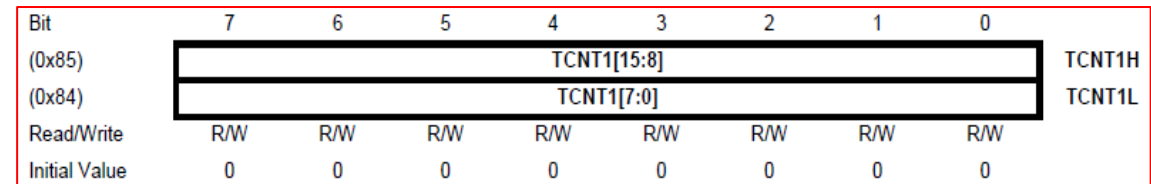
int milisec = 2000; //define delay length in miliseconds
int prescalar = 1024; //define prescalar
int clock_freq = 16000000/prescalar; //calc timer clock freq
float clock_period = 1/(float)clock_freq; //calc timer period
int count = ((milisec*.001/clock_period)-1); //calc count for required delay

void setup() {
    //define pins connected to LEDs as outputs
    pinMode(PIN_USED, OUTPUT);

    //set up timer
    TCCR1A = 0b00000000;
    TCCR1B = 0b00000101; //setting prescaler for timer clock
    TCNT1=0;
}

void loop() {
    if(TCNT1 >= count) // Checking if delay time has passed
    {
        TCNT1=0;
        digitalWrite(PIN_USED, !digitalRead(PIN_USED)); //toggle pin output
    }
}
```

Set-up registers are 8-bits even the Timer is 16-bit



References

- ATMega328 manual
- <https://www.avrfreaks.net/forum/tut-c-newbies-guide-avr-timers>
- <http://maxembedded.com/2011/06/avr-timers-timer0/>

THANKS FOR ATTENDING....

