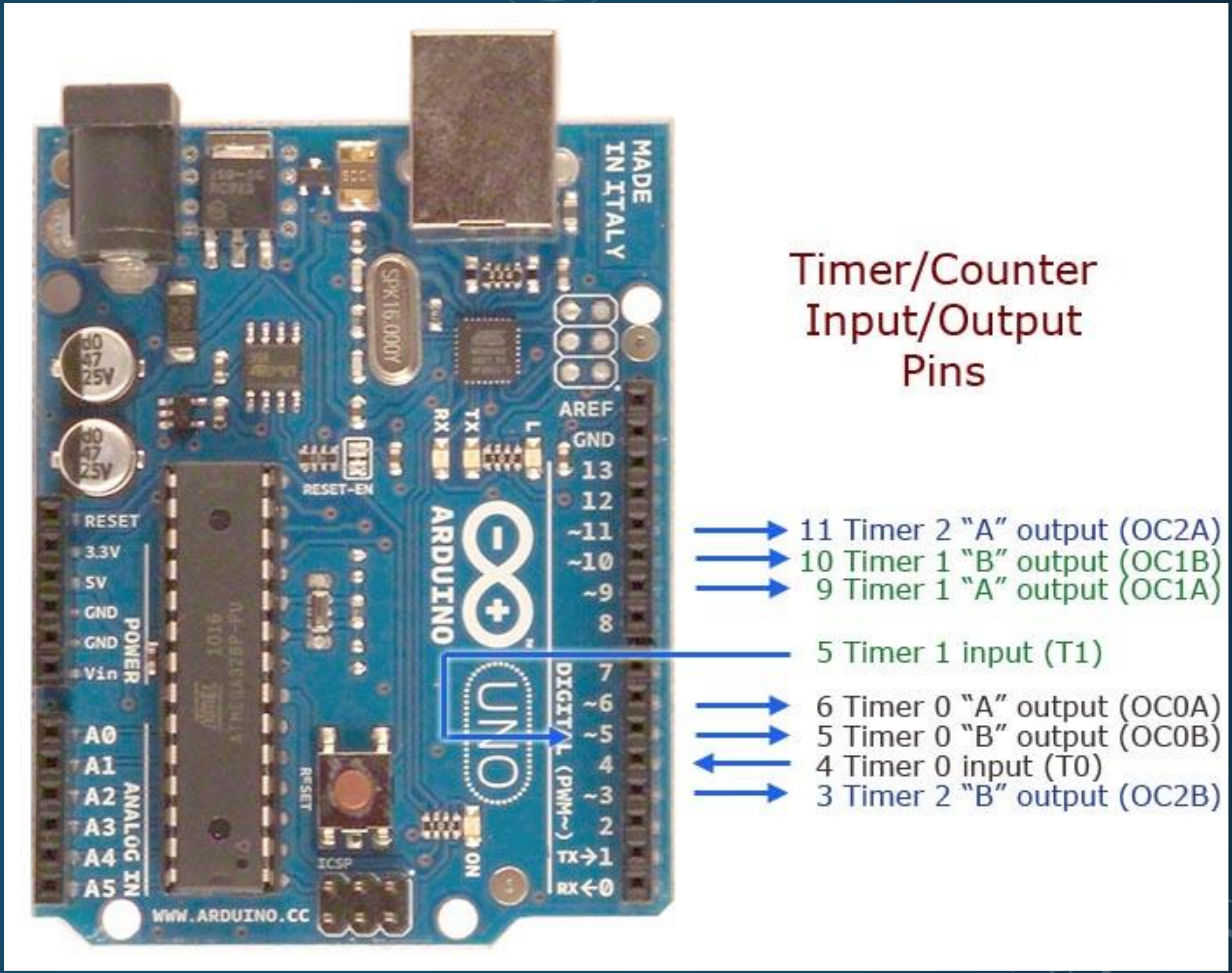
AMERICAN INTERNATIONAL UNIVERSITY – BANGLADESH(AIUB)

Where leaders are created

INTRODUCTION TO TIMERS

BY NIRJHOR ROUF

- 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.
- Timer0 and timer2 are 8bit timers, whereas timer1 is a 16bit timer. The most important difference between 8bit and 16bit timer 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.

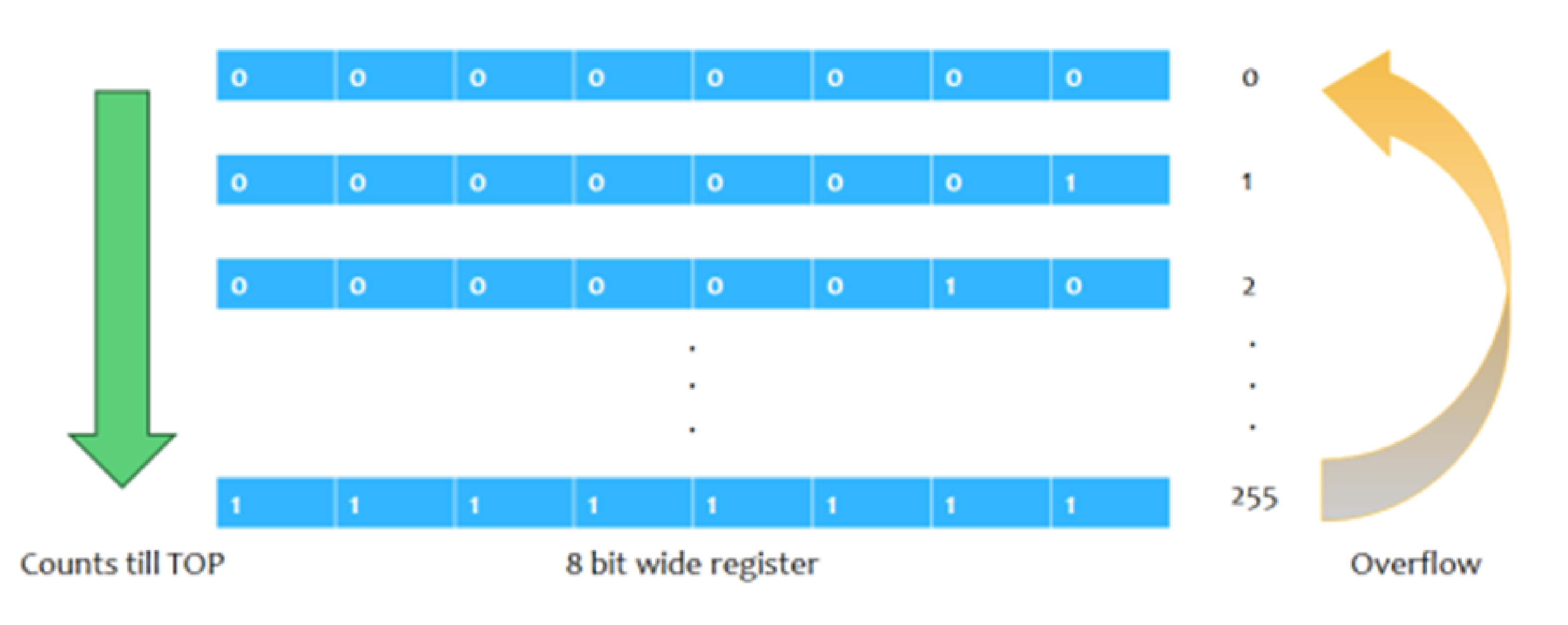


USEFULNESS OF TIMER :

- 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.
- 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, using 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 the 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 (16MHz), and would overflow and reset this counter many times per second.
- The three timers (called timer0, timer1 and timer2) of Atmega328 can be either operated in normal mode, CTC mode or PWM mode.
- For lab experiment 3 we will operate 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 – **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 – **TCCR0B** : The three Clock Select bits CS02, CS01, CS00 select the clock source and prescalar 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	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/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/Counter Register – **TCNT0** : This is where the 8-bit counter of the timer resides. The value of the counter is stored here and increases/decreases automatically. Data can be both read/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 TIMER0 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 TIFR register is set (one) whenever the count matches the stored value.

Bit	7	6	5	4	3	2	1	0	
0x27 (0x47)	OCR0A[7:0]								OCR0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

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) give direct access, both for read and for 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	

- 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.

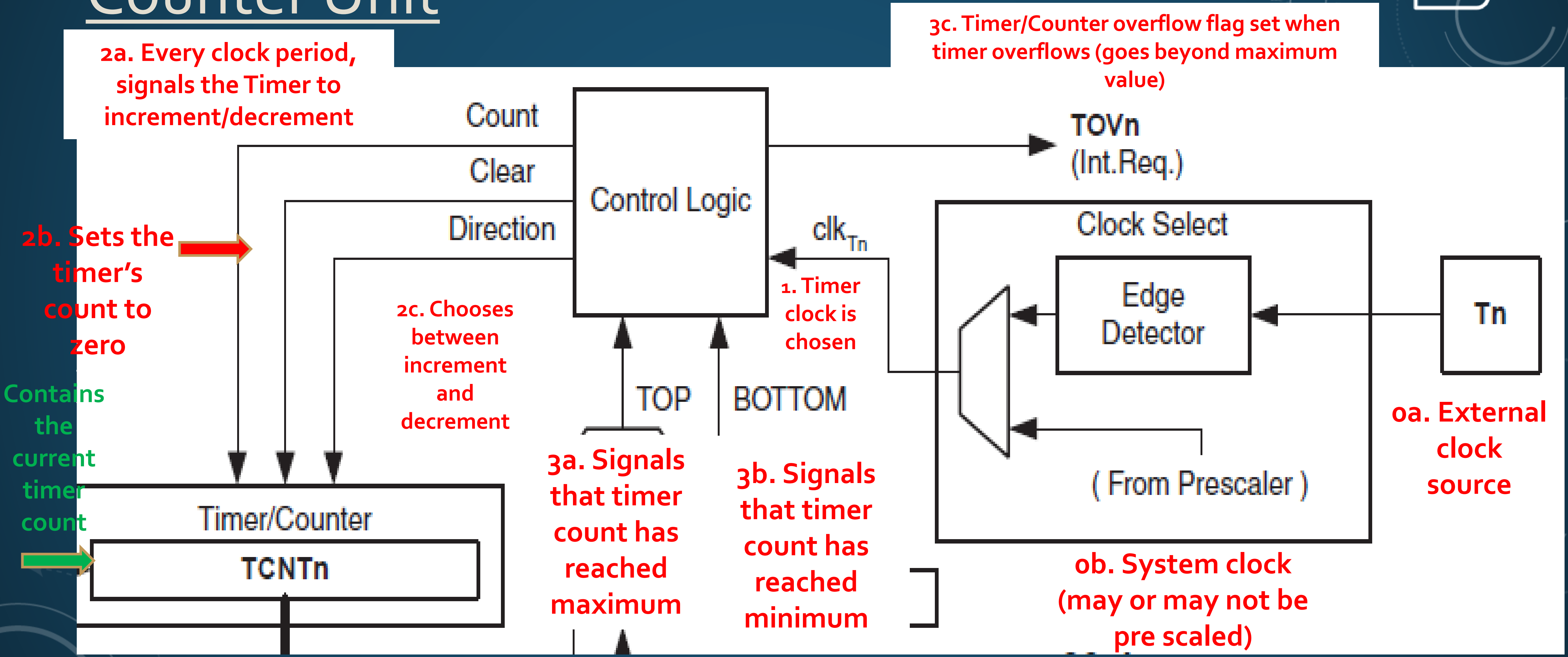
TIMER MODES: NORMAL MODE

- 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 operation the Timer/Counter Overflow Flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero.

TIMER MODES: CTC MODE

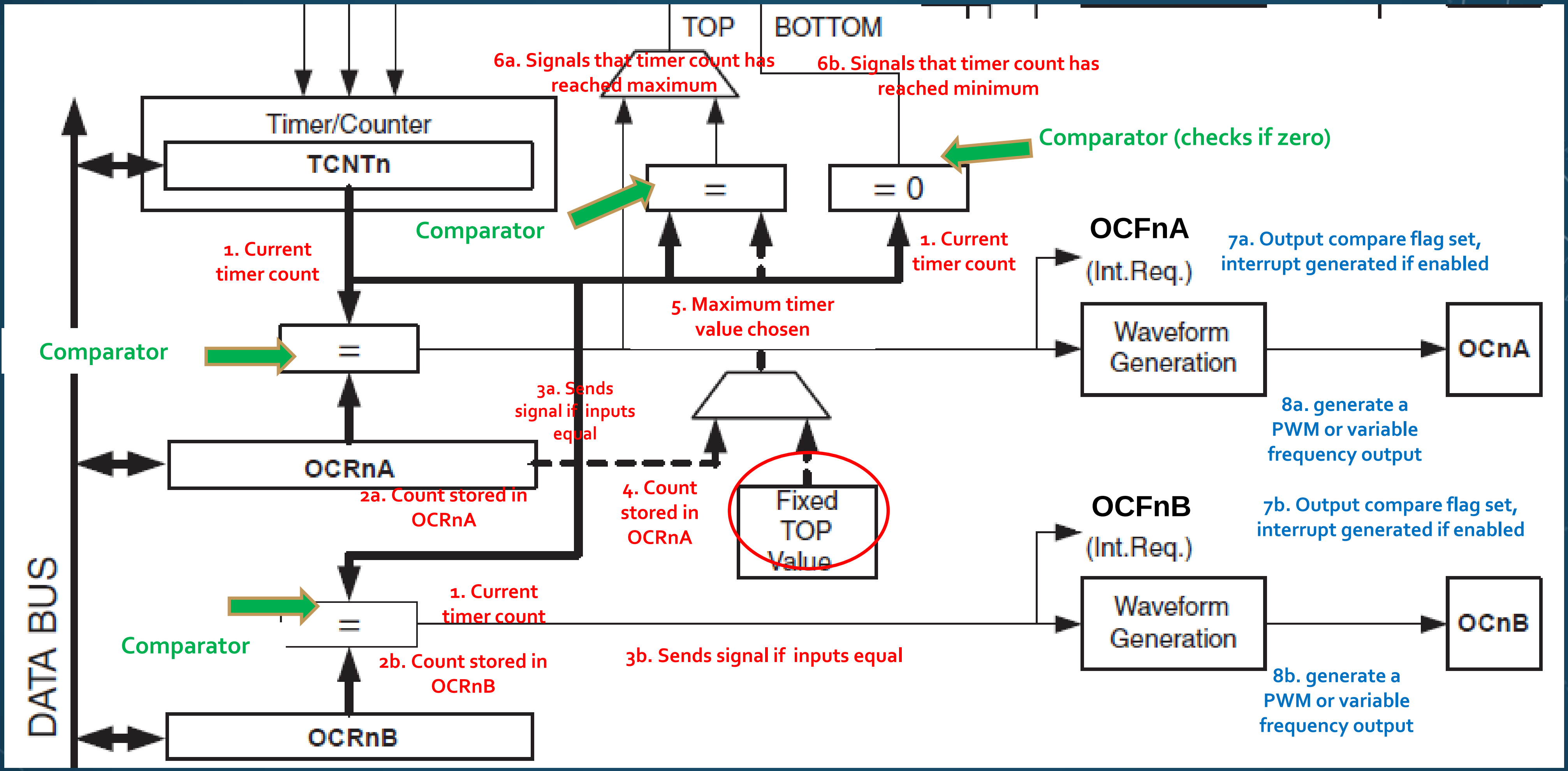
- In Clear Timer on Compare or CTC mode, the counter is cleared to zero when the counter value (TCNT0) matches either a value stored in OCR0A register.
- When the wanted value is reached a flag in a status register is set to ‘1’.
- This method can be more efficient than comparing bytes to detect a match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

Counter Unit

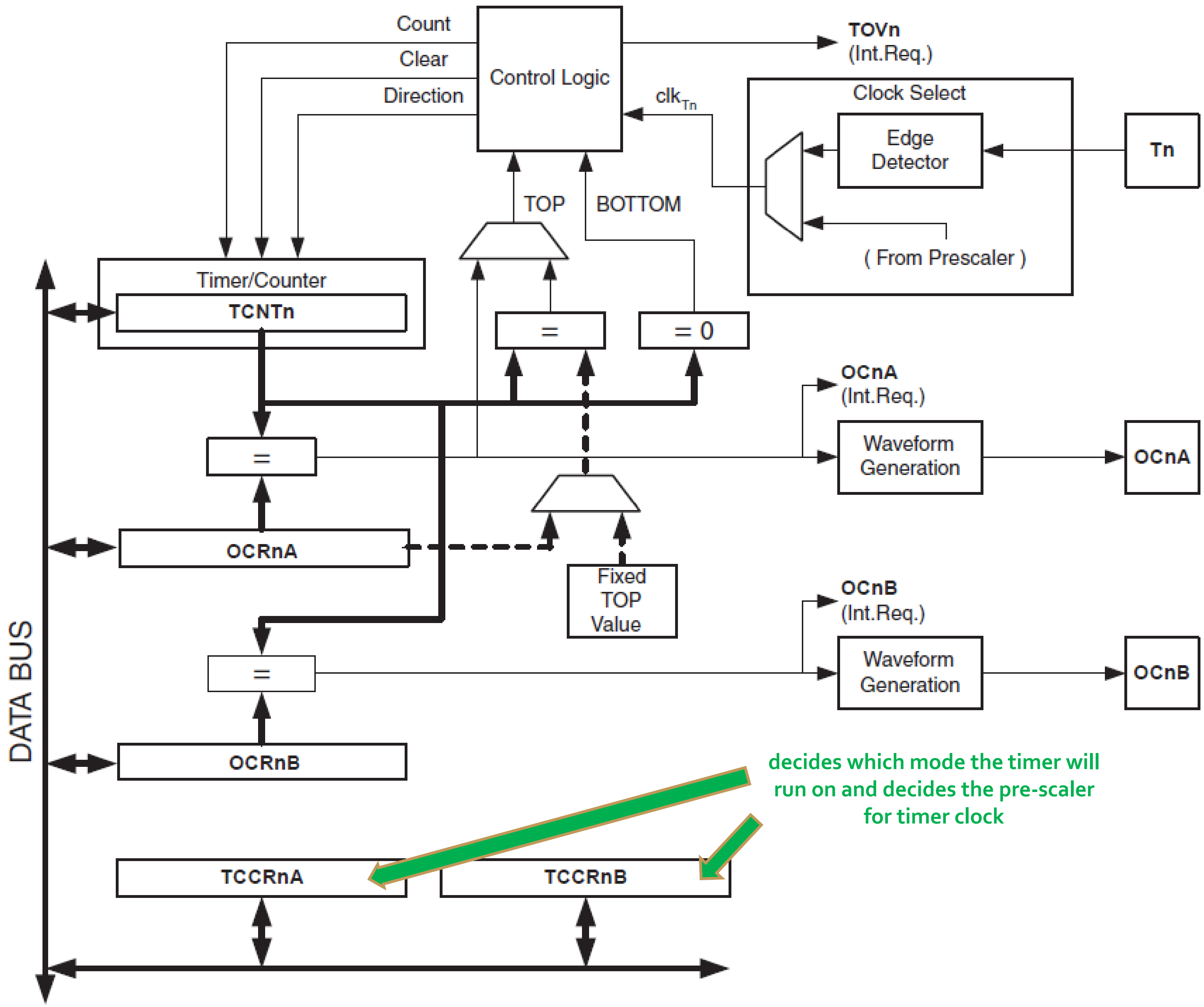


Red: signals entering/exiting a block
Green: description of a block

Output Compare Unit

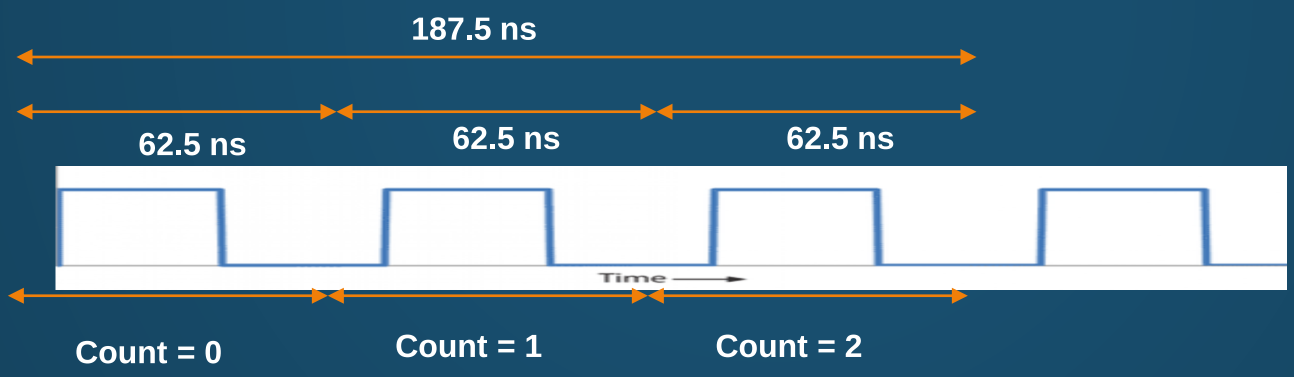


- Red: signals entering/exiting a block (Normal mode)
- Blue: signals entering/exiting a block (CTC/PWM mode)
- Green: description of a block


Count
Clear
Direction
Control Logic
TOVn
(Int.Req.)
clkTn
Clock Select
Edge
Detector
Tn
(From Prescaler)
TOP
BOTTOM
Timer/Counter
TCNTn
=
= 0
OCnA
(Int.Req.)
Waveform
Generation
OCnA
OCRnA
Fixed
TOP
Value
OCnB
(Int.Req.)
Waveform
Generation
OCnB
OCRnB
DATA BUS
TCCRnA
TCCRnB
decides which mode the timer will run on and decides the pre-scaler for 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_CPU = 16 MHz, system clock period = 1/16M = 62.5 ns. So **if timer clock is the same as system clock**, it takes only **62.5 ns** for every transition (0 to 1, 1 to 2, etc).



- 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:

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

Here required delay = 1 ms and timer clock period = 62.5 ns, so Timer Count = (1m/62.5n)-1= 15999.

- So the clock needs to tick **15999 times** to give a delay of only 1 ms.
- Maximum possible delay for **timer0** at 16MHz: 62.5 ns * 256 = **16µs**
- Maximum possible delay for **timer1** at 16MHz: 62.5 ns * 65536 = **4.096 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 is capable of counting up to 65535), it is not possible get longer delays.

TIMER FOR DELAY: CALCULATING COUNT

- To stay in the safe side, we use the highest available **prescalar** and reduce timer clock frequency to 16M/1024 = 15625Hz, with a new **timer clock period (= System clock period * prescalar)= 62.5ns*1024= 64 µs**. Now the needed timer count = (1m/ 64 µ) -1 = 15.6249.Now that **Timer clock frequency = System clock frequency/ prescalar**, we can update the equation to:

$$\boxed{Timer\ count = \frac{Required\ delay \times System\ clock\ frequency}{prescalar} - 1}$$

- Maximum possible delay for **timer0** at 15625Hz: 64µs * 256 = **16.384ms**
- Maximum possible delay for **timer1** at 15625Hz: 64µs * 65536 = **4.194304 s**
- To get longer delays, we will use timer1, and for delays longer than 4s, 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 2ms = (2m/64 μ)-1=30.25 ≈ 31

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

int milisec = 2;    //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
    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
      }
}
```

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 prescalar 1024

Make LED blink

PROBLEM STATEMENT 2

- **Make an LED blink every 2 seconds while using Arduino system frequency (F_CPU) 16 MHz, using timer to generate the delay without any application of delay() function.**
- Delay = 2 s, so Timer1 can be used with prescalar 1024.
- Number of count needed to reach 2s = (2/64 μ)-1=31249

CODE:

```
#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
      }
}
```

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

REFERENCES

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