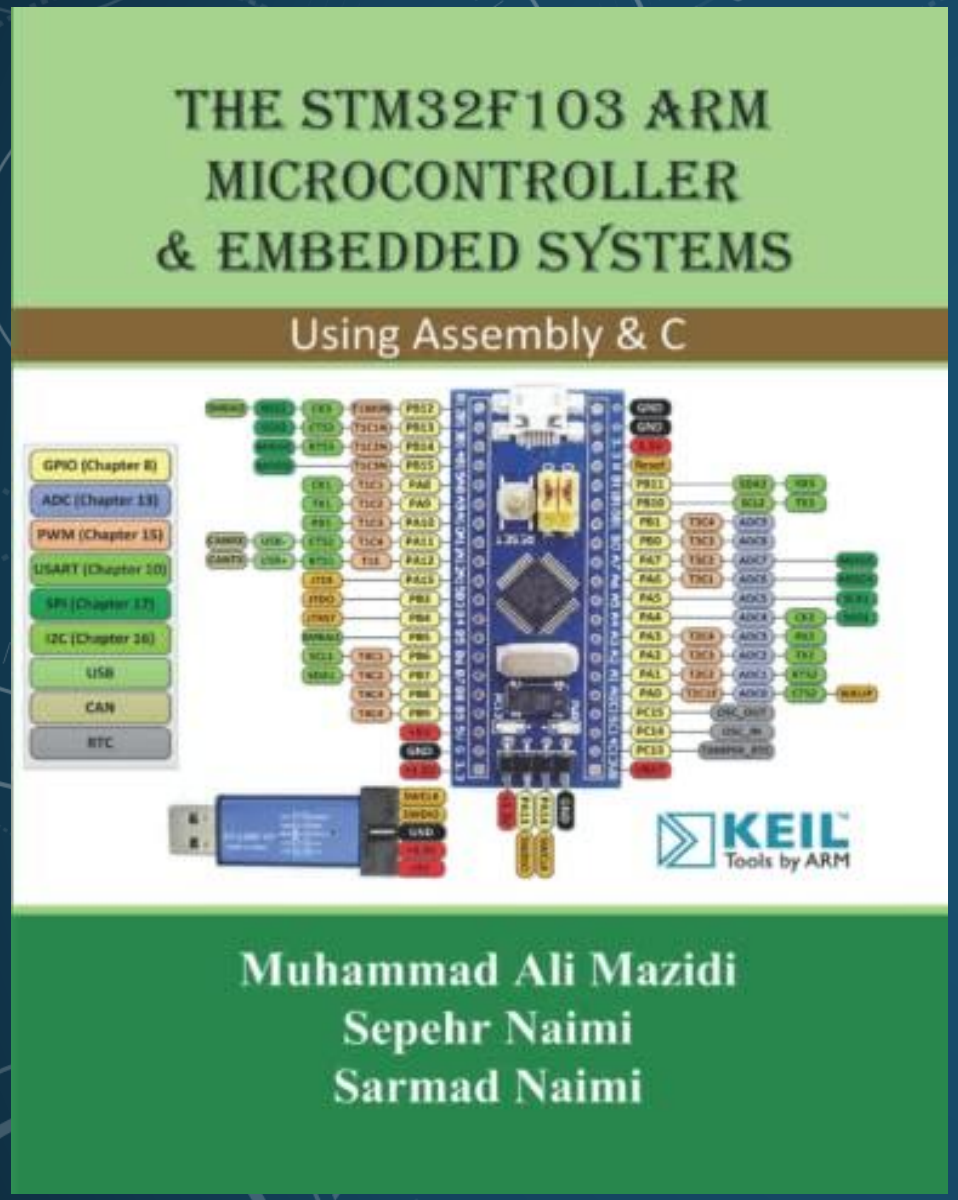


LECTURE # 02M

Prof. Dr. Engr. Muhibul Haque Bhuyan
Professor, Department of EEE
American International University-Bangladesh (AIUB)
Dhaka, Bangladesh

AMERICAN INTERNATIONAL UNIVERSITY – BANGLADESH (AIUB)
Where leaders are created

Introduction to Timers

BY NOWSHIN ALAM

Course Teacher: **Prof. Dr. Engr. Muhibul Haque Bhuyan**

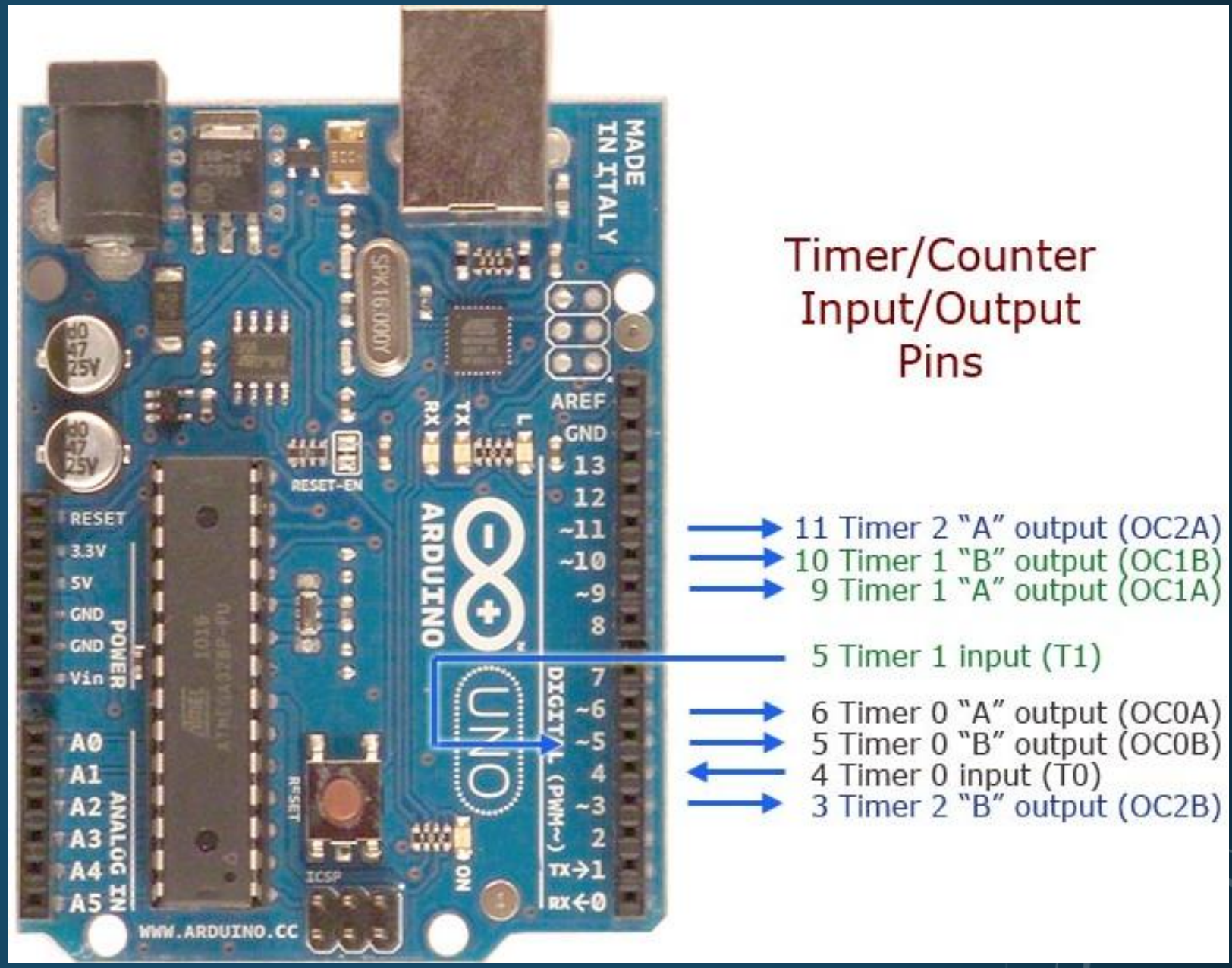
What is a Timer?

- A **timer/counter** is like a **clock** and can be used to measure time events.

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
.	.	.	.	.	.	.	.
1	1	1	1	1	1	1	1

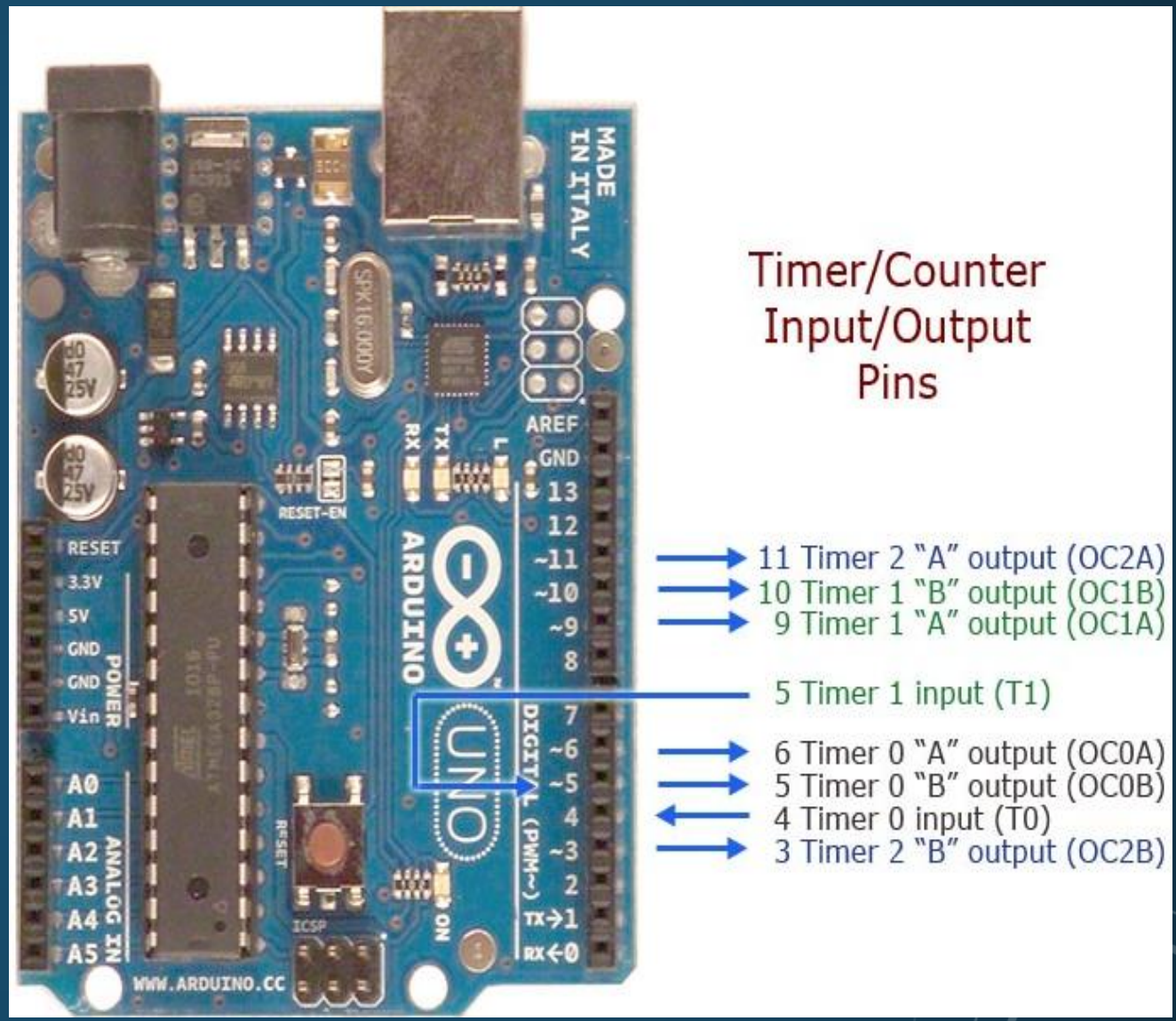
What is a Timer?

- 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**.



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 :

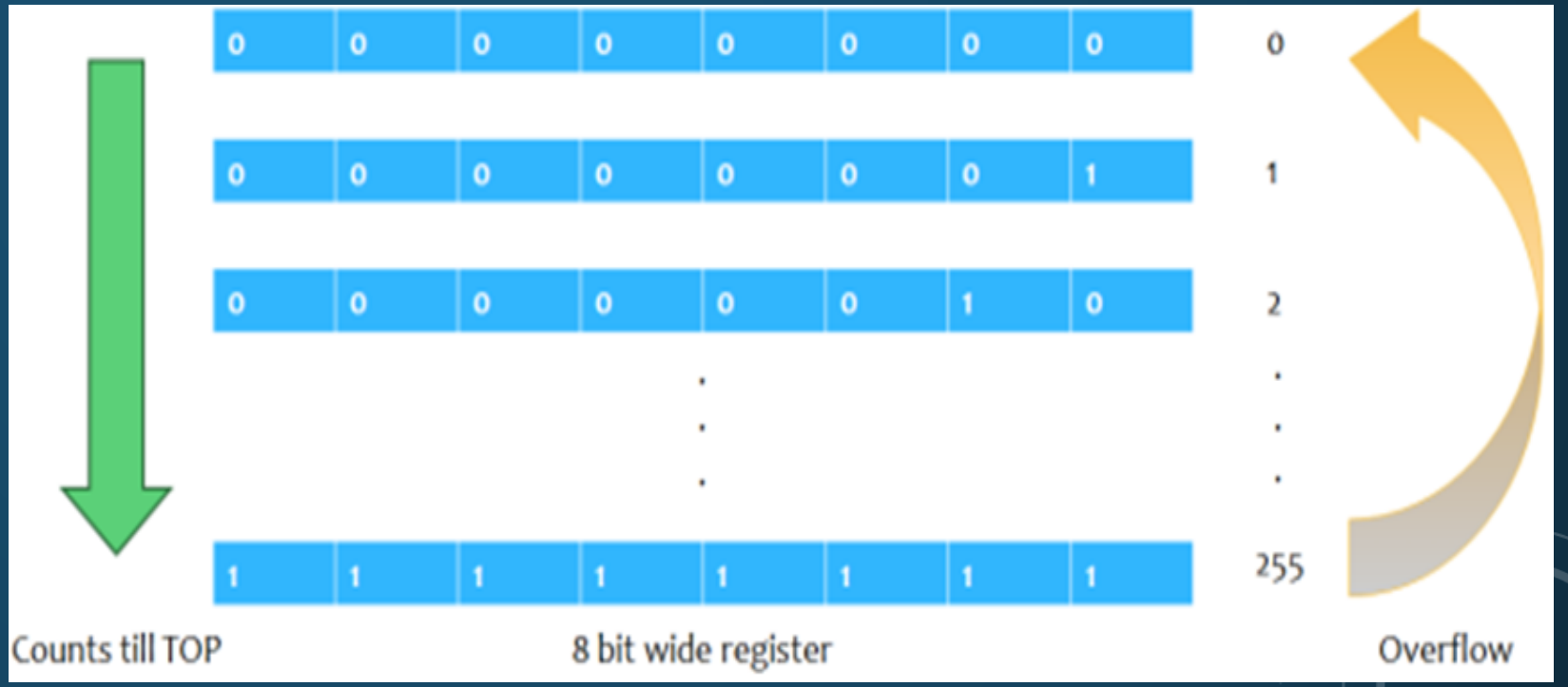
- Timer is an important concept in the field of electronics. Every electronic component of a sequential logic circuit works on a time base to keep all the work synchronized.
- An **advantage** of the **timer** is that it is totally **independent of the CPU**. Thus, it runs parallel to the CPU and there 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	$clk_{I/O}$/(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 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 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 the TIFR register is set (one) whenever the count matches the stored value in OCR0A/B.**

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

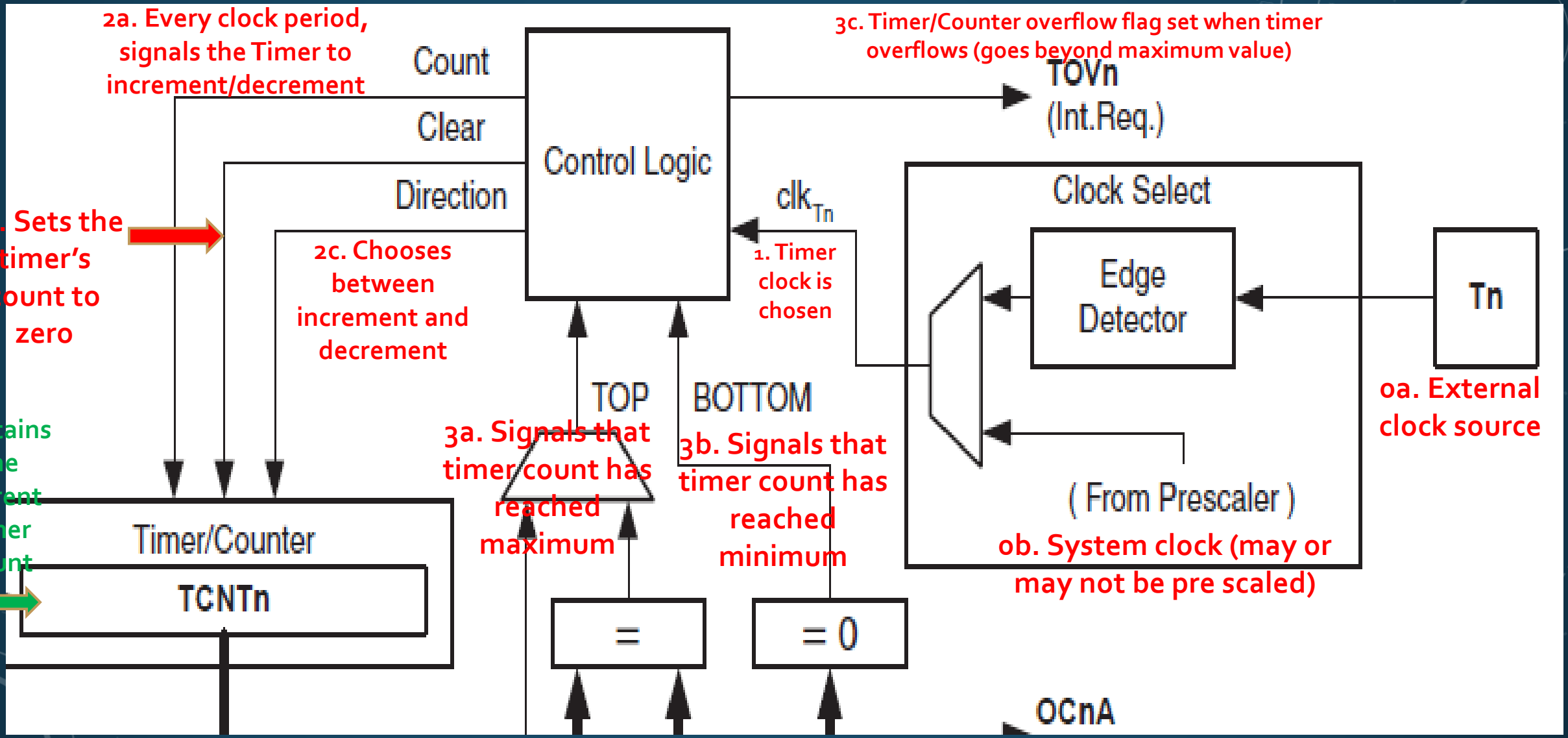
- The simplest mode of operation is the Normal mode. In this mode, the counting direction is always **up (incrementing)**, and no counter clear is performed.
- The counter simply overflows when it passes its maximum value and then restarts from zero.
- In normal mode of operation, the Timer/Counter Overflow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

Timer Modes: Clear Timer on Compare, CTC Mode

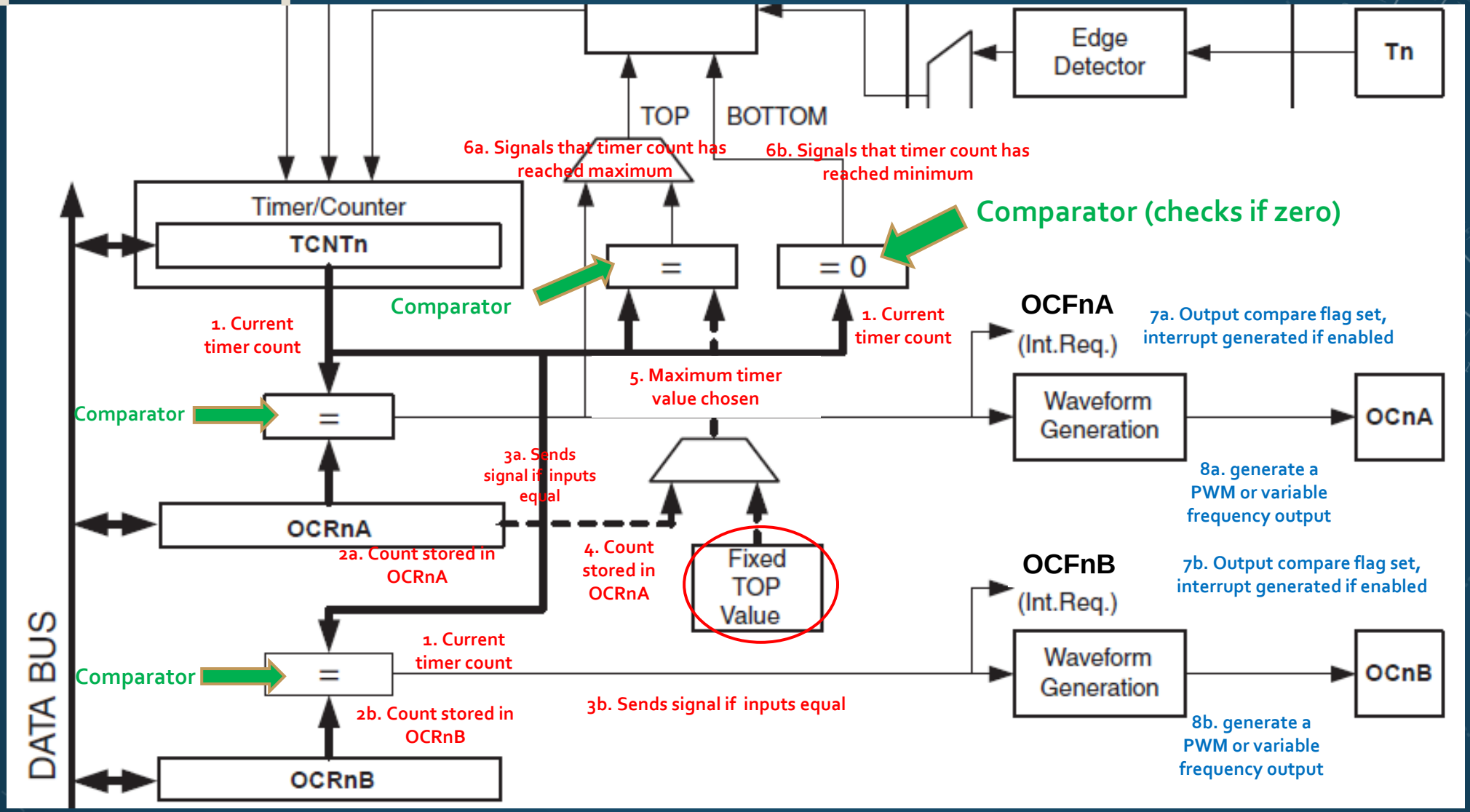
- In **Clear Timer on Compare or CTC mode**, the counter is cleared to zero when the counter value (TCNT0) matches either a value stored in OCR0A register.
- When the desired value is reached, a flag in a status register is set to ‘1’.
- This method can be more efficient than comparing bytes to detect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

Counter Unit

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

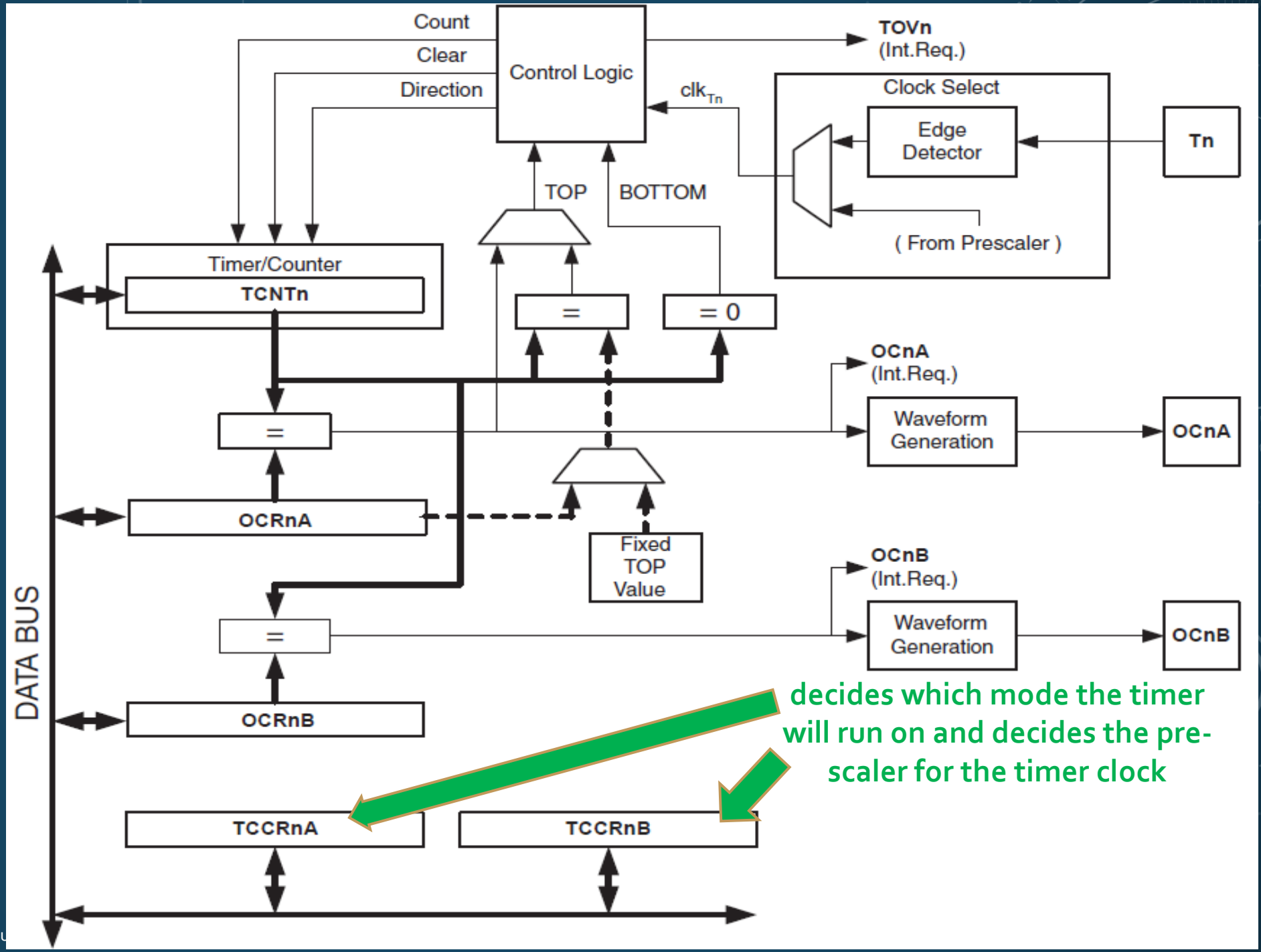


Output Compare Unit



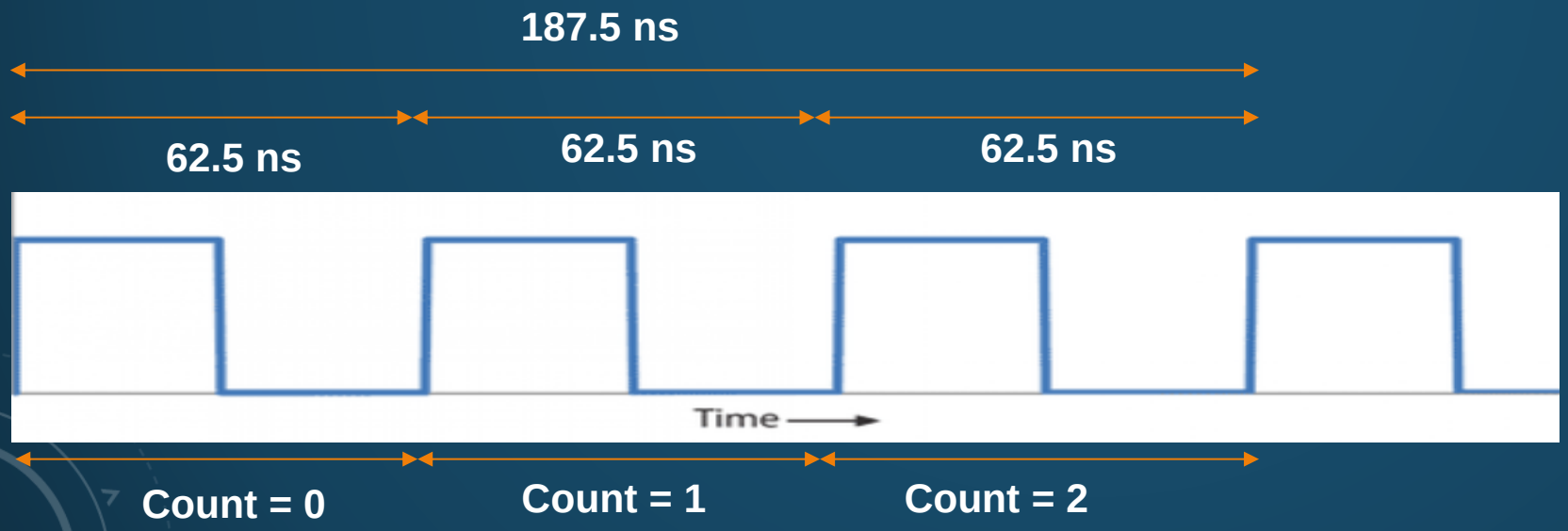
- Red: signals entering/exiting a block (Normal mode)
- Blue: signals entering/exiting a block (CTC/PWM mode)
- Green: description of a block

Control Unit



Timer for Delay: Calculating Count

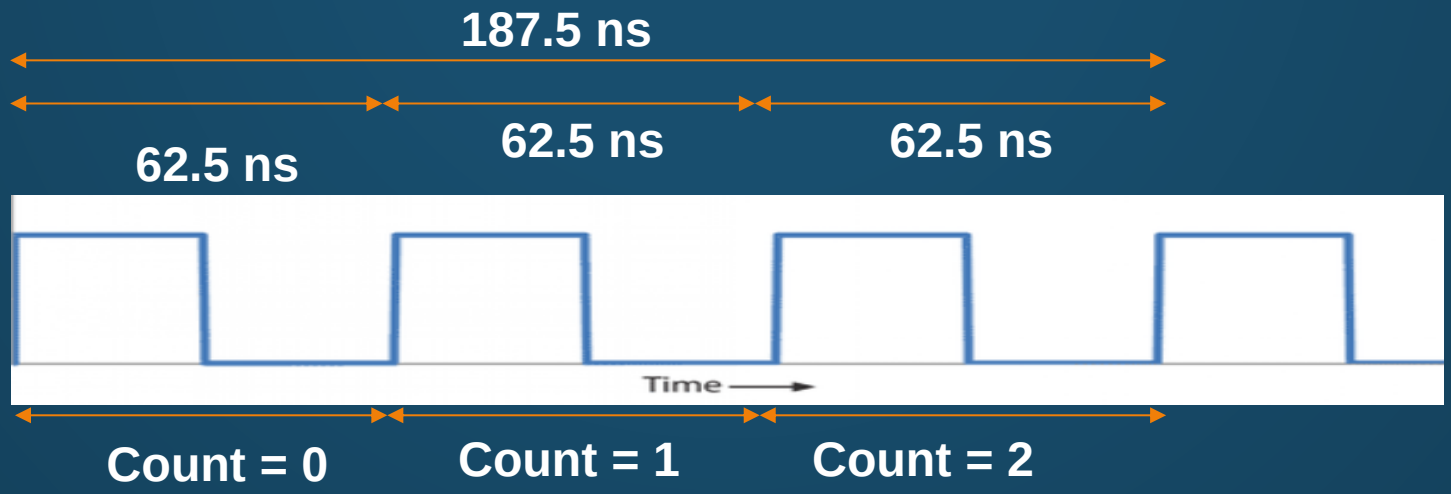
- Timer needs a clock pulse for each transition from one number to the next, so we want to establish a formula relating the necessary timer counts for a specific delay with the timer clock period/frequency.
- For F_CPU, f = 16 MHz, the system clock period, T = 1/16MHz = 62.5 ns*. So, if the timer clock 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 from 0 to 2, so timer count = (number of time periods needed to reach the count) -1.
- That is, to get a delay of 187.5 ns, we need 3 time periods lasting 62.5 ns, i.e., 62.5 ns x 3 = 187.5 ns
- So, $\text{Number of periods needed} = \dfrac{\text{Required delay}}{\text{Timer clock period}}$.



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 = (1000000ns/62.5ns) – 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 ns × 256 = **16 µs**
- Maximum possible delay for **Timer1** (which is a 16-bit timer) at 16 MHz: 62.5 ns × 65,536 = **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 the 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 16MHz/1024 = 15625 Hz, with a **new timer clock period** **(= System clock period × pre-scalar) = 62.5ns×1024 = 64 µs**. Now, the needed timer count = (1ms/64µs) – 1 = 14.625. Now that **Timer clock frequency, f_t = System clock frequency, *f*/pre-scalar, *ps***, we can update the equation

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

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

to

$$Timer\ count = \frac{Required\ delay}{prescalar\ (ps) \times T} - 1$$

Timer for Delay: Calculating Count

- Maximum possible delay for **Timer0** at 15625 Hz: 64 µs × 256 = **16.384 ms**
- Maximum possible delay for **Timer1** at 15625 Hz: 64 µs × 65536 = **4.194304 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 μs/64 μs) – 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
      }
}
```

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				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 μs/64 μs) – 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

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	

Code for the Timer0 Function

```
// Declaring the timer function to count 1 ms of delay instead of calling the delay function
int delay_timer (int milliseconds){
  int count = 0;
  while(1)
  {
   if(TCNT0 >= 16)  // Checking if 1 millisecond has passed
     {
      TCNT0 = 0;
      count++;
      if (count == milliseconds) //checking if required milliseconds delay has passed
      {
         count = 0;
         break; // exits the loop
      }
     }
  }
  return 0;
}
```

Delay = 1 ms, so Timer0 can be used with pre-scalar 1024. The number of counts needed to reach 1 ms = (1,000 μs/64 μs) – 1 = 15.625 -1 ≈ 15.

Modified Code using Timer0 for the Traffic Control

```
void setup() {
  //Define pins connected to LEDs as outputs
  pinMode(RED_PIN, OUTPUT);
  pinMode(YELLOW_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);

  //Set up the Timer0
  TCCR0A = 0b00000000;
  TCCR0B = 0b00000101; //setting pre-scaler for timer clock
  TCNT0 = 0;
```

Modified Code using Timer0 for the Traffic Control

```
void loop() {
  //to make the green LED turned on
  digitalWrite(GREEN_PIN, HIGH);
  delay_timer(green_on);
  //to make the green LED turned off
  digitalWrite(GREEN_PIN, LOW);

  //for turning the yellow LED on and off for 4 times
  for(int i = 0; i < 4; i = i+1)
  {
  digitalWrite(YELLOW_PIN, HIGH);
  delay_timer(yellow_blink);
  digitalWrite(YELLOW_PIN, LOW);
  delay_timer(yellow_blink);
  }

  //to make the red LED turned on
  digitalWrite(RED_PIN, HIGH);
  delay_timer(red_on);
  //to make the red LED turned off
  digitalWrite(RED_PIN, LOW);
  }
```

Change the marked lines with the new function, delay_timer() inside the lop.

Old Code using delay() Function for the Traffic Control

```
void loop() {
// Turning on the voltage at the output pin 8 (for green LED)
digitalWrite(8, HIGH);
delay(3000);    // green LED is on for 3 seconds
// Turning off the voltage at output pin 8 (for green LED)
digitalWrite(8, LOW);   // green LED is off

// Turning the yellow LED on and off for 4 times
for (int i = 0; i < 4; i = i+1)
{
digitalWrite(10, HIGH);
delay(500);   // yellow LED is on for 0.5 seconds
digitalWrite(10, LOW);
delay(500);   // yellow LED is off for 0.5 seconds
}

// Turning on the voltage at the output pin 12 (for red LED)
digitalWrite(12, HIGH);
delay(6000);      // red LED is on for 6 seconds
```

To-do list:

- Initialize timer registers
- Write a new delay function using a timer
- Change the marked lines

Example 1:

If you want to make an LED of blue color blink every 2 seconds while using an Arduino system frequency of 16 MHz using a timer to generate delay without any application of the built-in delay() function, select a suitable timer of the Arduino Uno for your application. Use a pre-scaler value of 256. Determine the necessary register set-up required for the program. A table of clock-select bits and register structures is given at the end of this question paper.

Answer:

Required delay = 2 s = 2000000 μs; Given system frequency = 16 MHz

So, clock period = 1/frequency = 1/16 MHz = 0.0625 μs

As such,

$$Timer\ count = \frac{Required\ delay}{prescalar\ (ps) \times T} - 1 = \frac{2000000}{256 \times 0.0625} - 1 = 1{,}24{,}999$$

Example 1:

Timer 0 can count up to 256 and Timer 1 can count up to 65,536; so, neither Timer0 nor Timer1 of Arduino is suitable for this application, but with the use of a Timer Function Sub-Program that can generate 1 ms of delay upon each call of the function, any one timer may be used for 10 s time calculation. The timer will run in normal mode as such all values of the timer counter control registers (TCCRnx, here n = 0/1 and x = A/B) will be zero except for the clock select bits (CSn2:0) of the pre-scaler value of 256 in the TCCR0B or TCCR1B will be 100. If we use Timer0 then the register setup will be as follows:

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	
	0	0	0	0	0	0	0	0	

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	
	0	0	0	0	0	1	0	0	

Example 2:

Determine the outputs of the program and correct the program if there are any errors. Assume that a green color LED is connected to pin 5 and the Arduino frequency is 16 MHz. Compute the number of counts required for a 3 ms delay in each cycle for the given delay. Compute the number of times the while_true loop will run to execute the delays of LED ON/OFF times. Use a pre-scaler value of 1024.

```
#define YELLOW_PIN 10
int green_on = 5000; // delay is given in ms
int green_off = 4000; // delay is given in ms

int delay_timer (int milliseconds){
  int count = 0;
  while(1) {
    if(TCNT1 >= _____) {
       TCNT1=0;
       count++;
       if (count == milliseconds) {
          count=0;
          break; }
    }
  }
  return 0;
}
```

```
void loop() {
   pinMode(GREEN_PIN, INPUT);
   TCCR0A = 0b00000000;
   TCCR0B = 0b00000100;
   TCNT1 = 0x0000;
}

void loop() {
   digitalWrite(BLUE_PIN, HIGH);
   delay(green_on);

   digitalRead(GREEN_PIN, LOW);
   delay_timer(blue_off);
```

Prescaler values and corresponding register contents.

CSx2	CSx1	CSx0	Prescaler
0	0	1	1
0	1	0	8
0	1	1	64
1	0	0	256
1	0	1	1024

Example 2:

Answer:

Corrected Codes

```
#define YELLOW_PIN 10 GREEN_PIN 5
int blue_on = 5000; // delay is given in ms green_on
int red_off = 4000; // delay is given in ms green_off

int delay_timer (int milliseconds){
  int count = 0;
  while(1) {
    if(TCNT1 >= 46) {
        TCNT1=0;
        count++;
        if (count == milliseconds) {
            count=0;
            break; }
      }
  }
  return 0;
}
```

```
void loop() {   setup
    pinMode(GREEN_PIN, INPUT);  OUTPUT
    TCCR0A = 0b00000000;  TCCR1A
    TCCR0B = 0b00000100;  TCCR1B
    TCNT1 = 0x0000;
}

void loop() {
    digitalWrite(BLUE_PIN, HIGH);  GREEN_PIN
    delay(green_on);    delay_timer

    digitalRead(GREEN_PIN, LOW);  digitalWrite
    delay_timer(blue_off);    green_off
}
}
```

Example 2:

The output of the program is: A green LED connected to pin 5 will be turned ON for 5 s and OFF for 4 s and this ON-OFF status will be repeated indefinitely after correcting the program (shown in the program above with the wrong parts stroked through in red color) until the microcontroller's power is turned OFF.

Given system frequency = 16 MHz; So, clock period = 1/frequency = 1/16 MHz = 0.0625 μs
Required delay = 3 ms = 3000 μs

$$Timer\ count = \frac{Required\ delay}{prescalar\ (ps) \times T} - 1 = \frac{3000}{1024 \times 0.0625} - 1 = 46$$

The number of counts required for a 5 s (5000 ms) delay in each cycle for the LED ON time = 5000 ms/3 ms = 1667. The number of counts required for a 4 s (4000 ms) delay in each cycle for the LED OFF time = 4000 ms/3 ms = 1334.
Timer0 can count up to 255 and Timer1 can count up to 65,535; Timer1 is appropriate for this application.

Example 3:

If you want to make an LED of red color blink every 4 seconds while using an Arduino system frequency of 16 MHz using a timer to generate the delay without any application of the delay() function, which timer of the Arduino Uno is suitable for your application? You may use a pre-scaler value of 256 1024. Write the necessary register set-up required for the program.

Given system frequency = 16 MHz; So, clock period = 1/frequency = 1/16 MHz = 0.0625 µs; Required delay = 4 s = 4000000 µs

$$Timer\ count = \frac{Required\ delay}{prescalar\ (ps) \times T} - 1 = \frac{4000000}{1024 \times 0.0625} - 1 = 62499$$

But Timer 0 can count up to 255 and Timer 1 can count up to 65,535; so Timer1 is suitable for this application.

Example 3:

```
void setup() {

   pinMode(REDLED_PIN, OUTPUT);

   TCCR1A = 0x00;  // Since Timer1 is used, so control registers of Timer1
   TCCR1B = 0x05;  // are used in this program instead of registers of Timer0.
   TCNT1 = 0x0000;
}
```

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....

?