Timers and Counters

Lecture Outline

- Counters and Timer Basics
- Timers in the AVR Device Family

Timer

A **timer** is a specialized type of clock that is used to measure time intervals.
It is a device that counts down from a specified time interval and is used to generate a time delay, for example, an hourglass is a timer.
The register is incremented for every machine cycle.

Counter

A **counter** is a device that stores (and sometimes displays) the number of times a particular event or process occurred, with respect to a clock signal.

It is used to count the events (e.g., events: rising edge or falling edge) happening outside the microcontroller.

Timer and Counter

In electronics, Timer functions are created by devices known as oscillators.

The Atmega has a 16 MHz crystal oscillator.

A counter segments a timer in a finite set of values. Each of this segments are referred as ticks.

Timers and Counters in Embedded Systems

- Embedded systems make extensive use of timers and counters
  - Measure duration of events
  - Count the number of events
  - Signal / clock generation

Counters

- Counters, uh, count
- Typical use case
  - A sensor detects an event and generates a pulse
  - The counter increments its count
  - Output is placed on bus or sent to a display

An Example Counter

- Basic Operation Overview
  - Signal to be counted is sent to the CPC pin the count is incremented
  - When the CPR pin is set to high, the counter value is transferred to the storage register
  - Assuming the output is enabled, the register value appears on the output pins Q0-Q7

74HC590 Counter

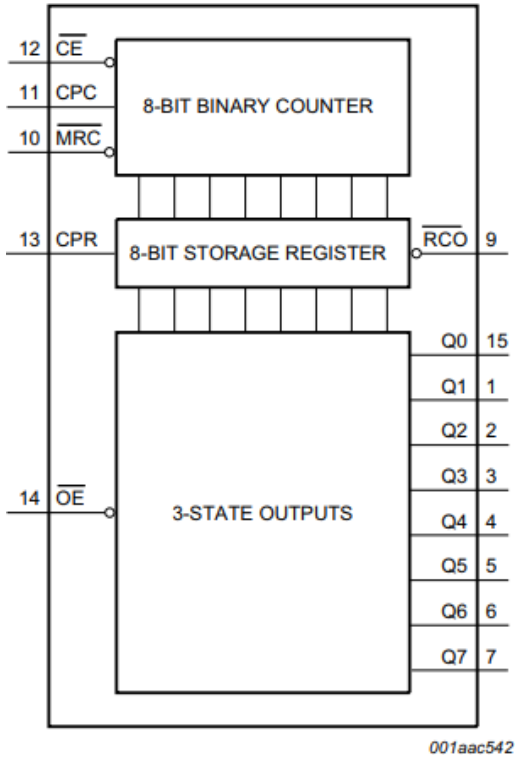


Table 2. Pin description

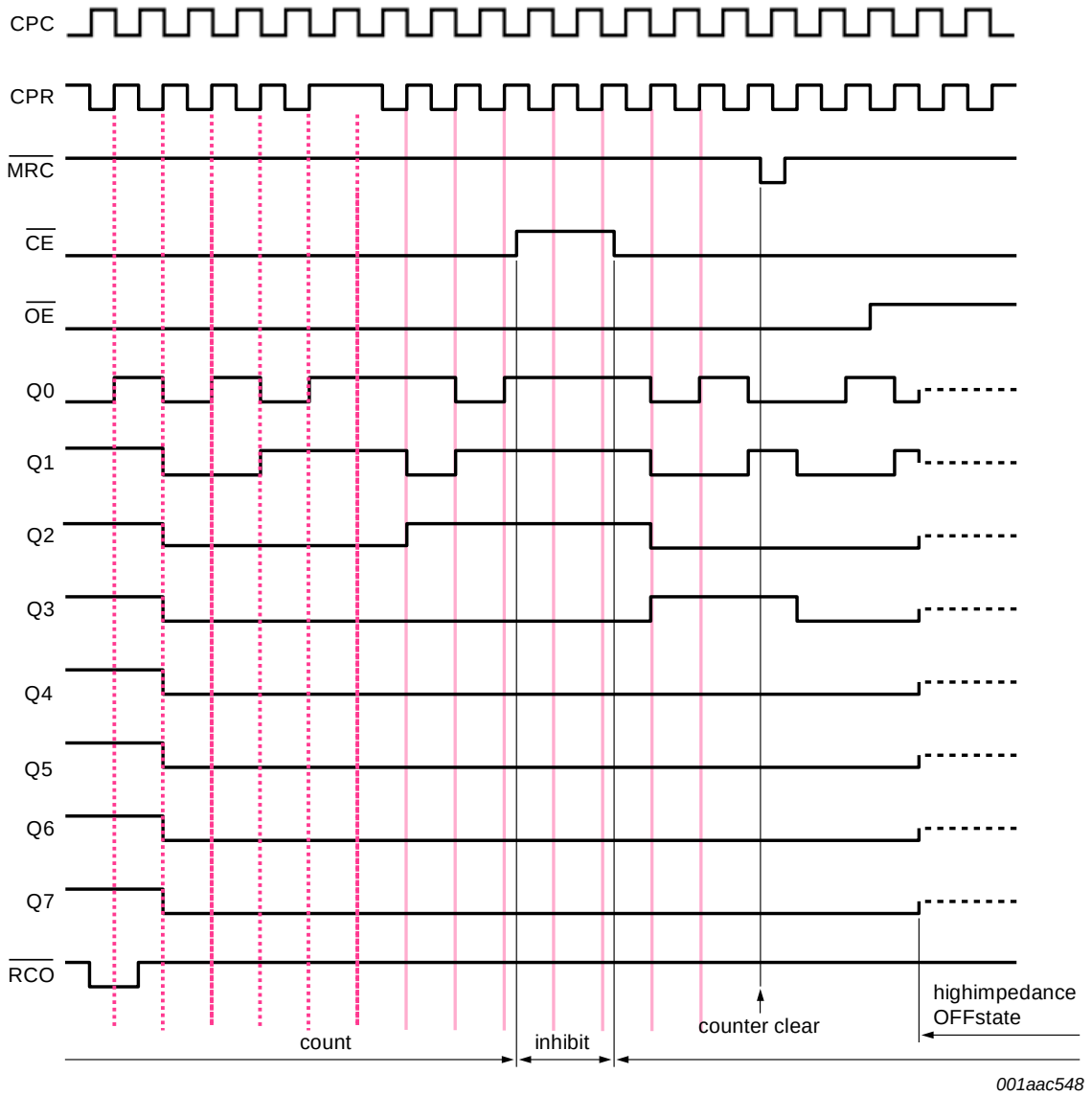
Symbol	Pin	Description
Q0 to Q7	15, 1, 2, 3, 4, 5, 6, 7	parallel data output
GND	8	ground (0 V)
$\overline{RCO}$	9	ripple carry output (active LOW)
$\overline{MRC}$	10	master reset counter input (active LOW)
CPC	11	counter clock input (active HIGH)
$\overline{CE}$	12	count enable input (active LOW)
CPR	13	register clock input (active HIGH)
$\overline{OE}$	14	output enable input (active LOW)
V_{CC}	16	supply voltage

Example Counter Timing Diagram

- Typical timing diagram for the 74HC590 IC
- Sequence starts with RCO going low, indicating that the counter is starting over from 0
- NOTE: Clocks are positive edge triggered

Table 2. Pin description

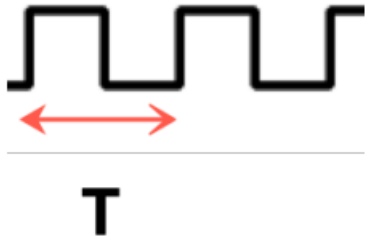
Symbol	Pin	Description
Q0 to Q7	15, 1, 2, 3, 4, 5, 6, 7	parallel data output
GND	8	ground (0 V)
$\overline{RCO}$	9	ripple carry output (active LOW)
$\overline{MRC}$	10	master reset counter input (active LOW)
CPC	11	counter clock input (active HIGH)
$\overline{CE}$	12	count enable input (active LOW)
CPR	13	register clock input (active HIGH)
$\overline{OE}$	14	output enable input (active LOW)
V_{CC}	16	supply voltage



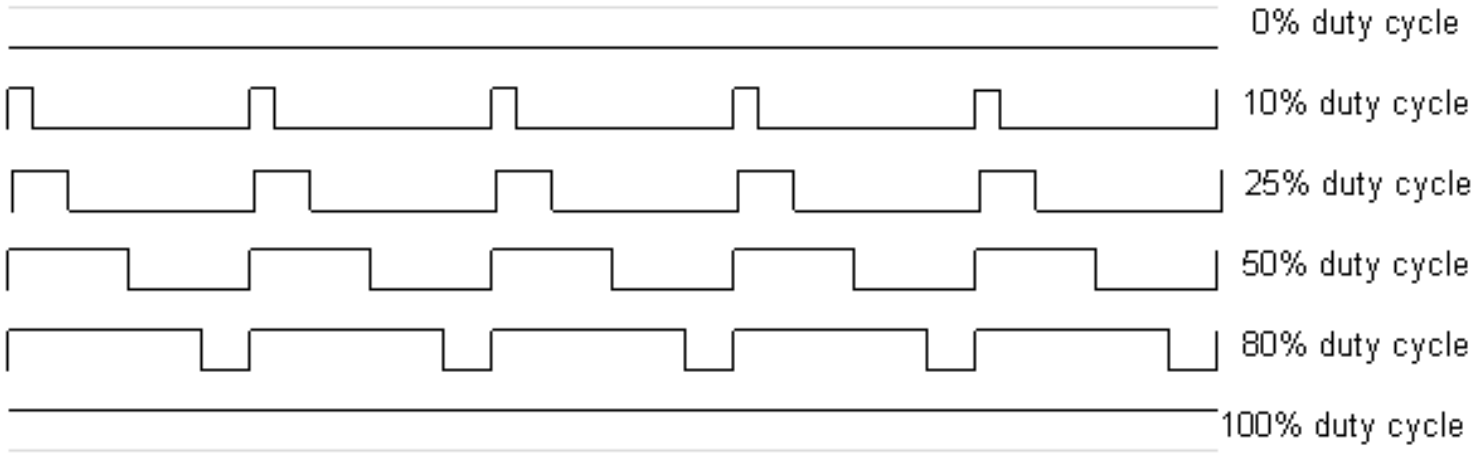
Source: https://assets.nexperia.com/documents/data-sheet/74HC590.pdf

Timers

- Timers are just counters where the counted input is a clock
- Since we know the frequency of the clock, the elapsed time for a given number of ticks can be determined
- Example:
  - fc = clock frequency = 1 MHz
  - Period T = 1/fc = 1 μs
  - If we start our timer at 123 and an event happens at count 231, the time elapsed would be (231-123)* 1 μs = 108 μs



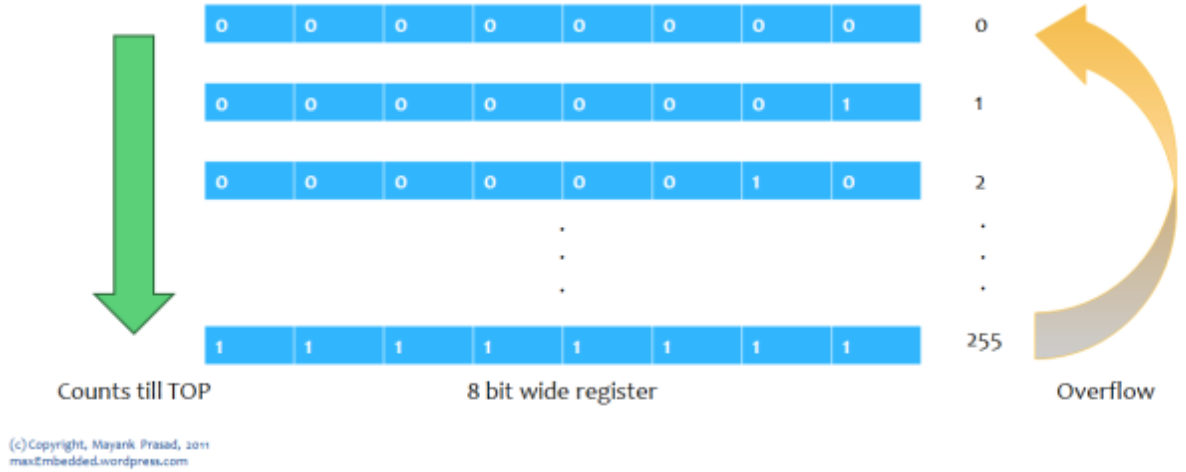
Duty Cycle – the signal's percentage of being in a high state.



The AVR Timers and Counters

- The timers are actually a hardware peripheral that is embedded in the microcontroller
  - In a typical computer, the timers would be in separate chips/systems
- The AVR family has from 1 to 6 timers
- In the 2560, there are 2 8-bit and 4 16-bit counters labelled 0 through 5
- Timers 0 and 2 are 8-bit, timers 1, and 3-5 are 16 bit

In an 8-bit timer, the register used is 8-bit wide whereas in 16-bit timer, the register width is of 16 bits. This means that the 8-bit timer is capable of counting 2^8=256 steps from 0 to 255 as demonstrated below.



Similarly, a 16-bit timer is capable of counting 2^16=65536 steps from 0 to 65535. Due to this feature, **timers are also known as counters**.

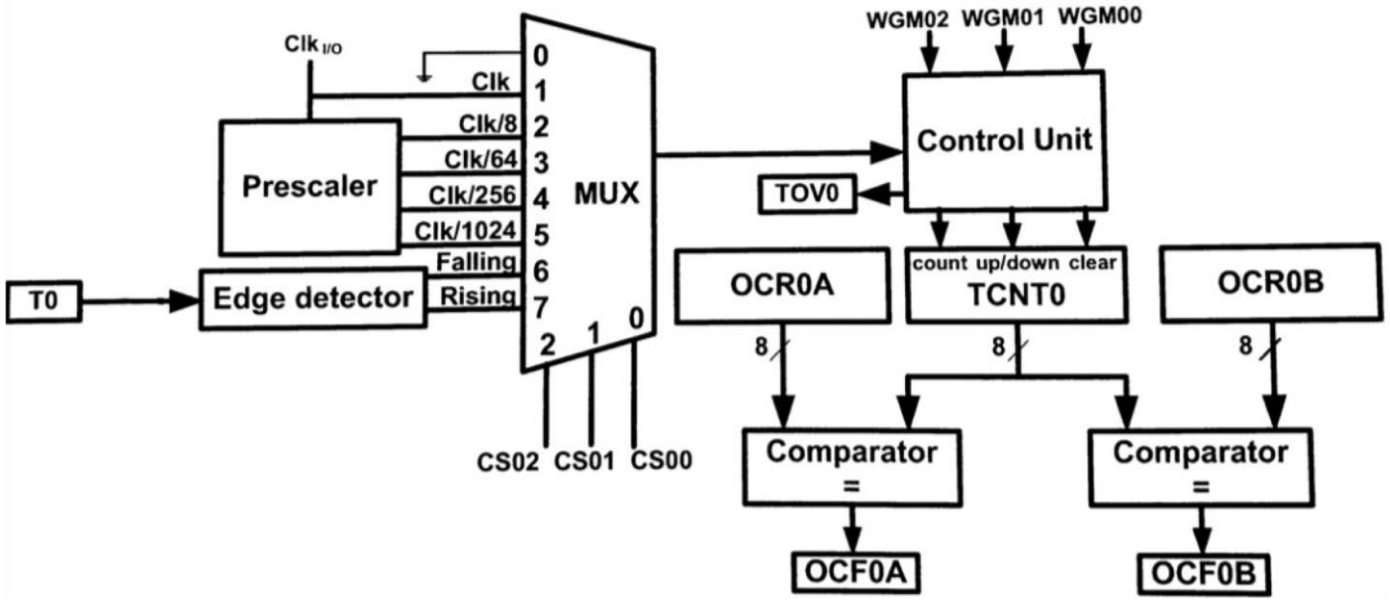
Now, what happens once they reach their MAX? Does the program stop executing? Well, the answer is quite simple. It returns to its initial value of zero. We say that the timer/counter **overflows**.

AVR Timer Details

- Each timer has two *channels*: A and B
- The output of these channels are at the same frequency but can have different pulse widths
- Timers have multiple *modes*
- Modes determine the behavior of the counter/timer, affecting the pulse width, frequency, and relationship to the system clock

Timer 0 in Detail

- TOV0: Timer 0 overflow flag
- OCR0n: Output compare registers
- TCNT0: Timer 0 counter
- OCF0n: Output comparator flags
- WGM0n: Waveform generator mode bits



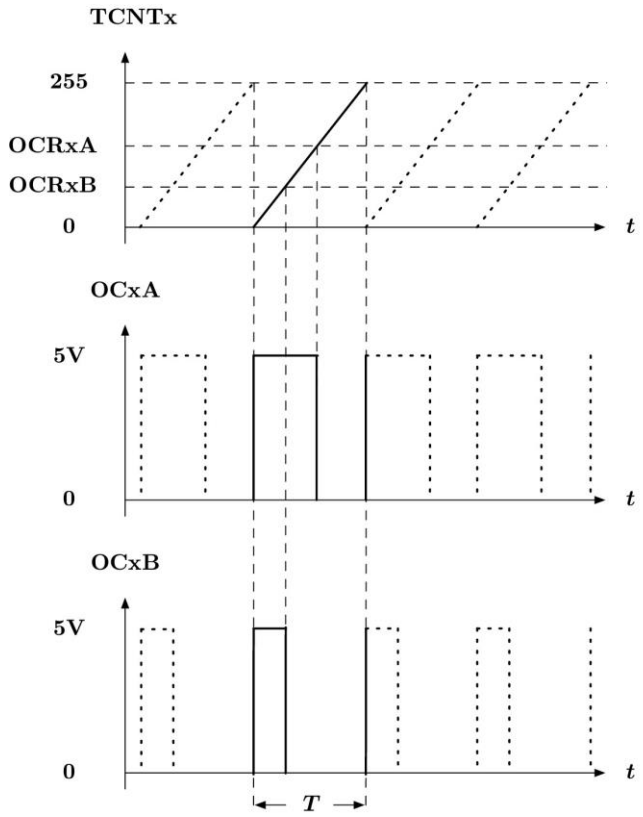
The AVR Microcontroller and Embedded Systems Using Assembly and C, Mazadi et al.

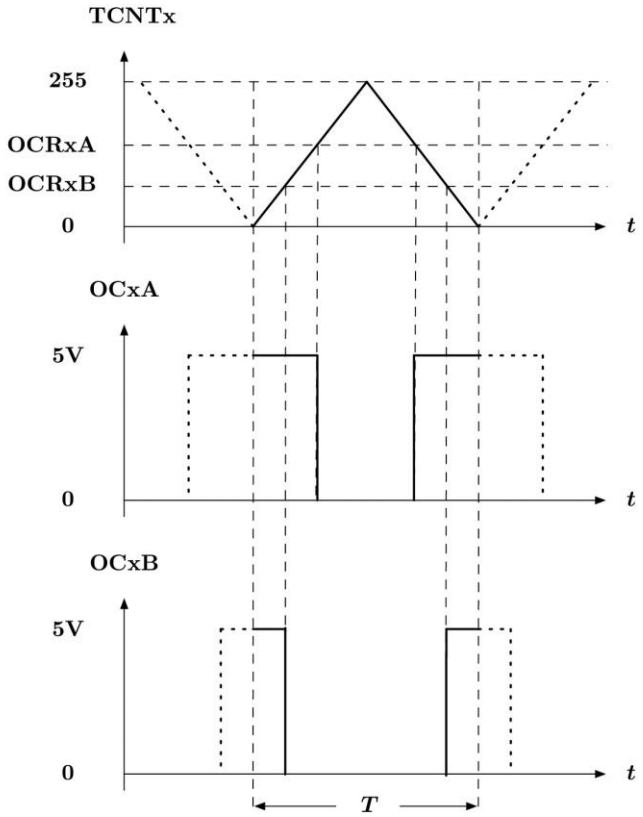
Timer Modes

- Normal mode
  - The timer always counts to its maximum value and then starts over
- Fast PWM Mode
  - Used to generate fast PWM signals
  - The timer counts to its maximum value or to a predefined value based on a register settings
    - These settings effect both the frequency and the pulse width
- Phase-Correct PWM
  - The counter counts up and then back down.
  - Register settings control the pulse width
  - The generated pulses are centered on the timer counter
    - In fast PWM modes, the generated pulses are based on the leading edge of the clock

Timer Modes –TOP = $FF

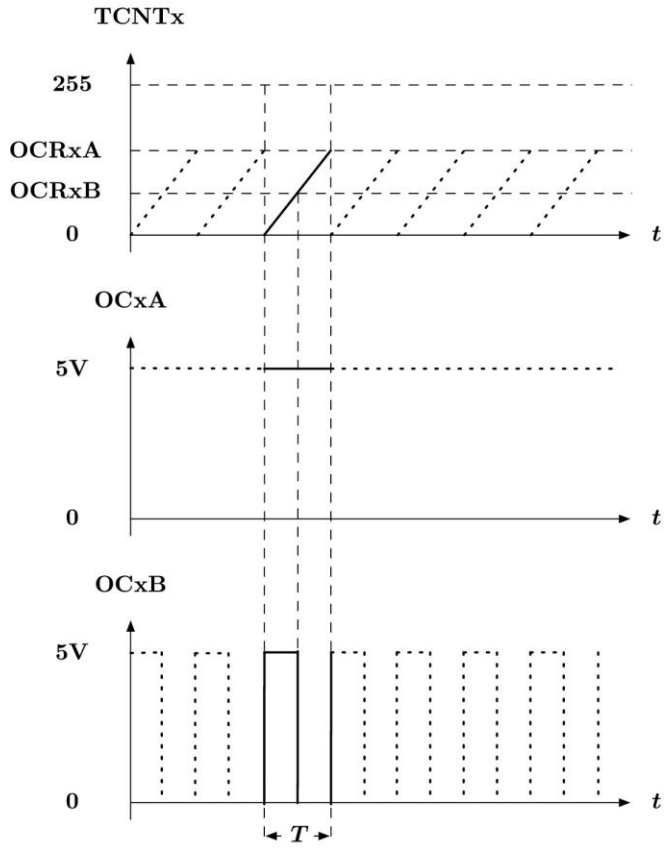
- Setting the appropriate bits in the timer/counter control register (TCCR) causes the counter to its max value (called TOP)
- The output compare registers (OCRnA and B) control the width of the pulse

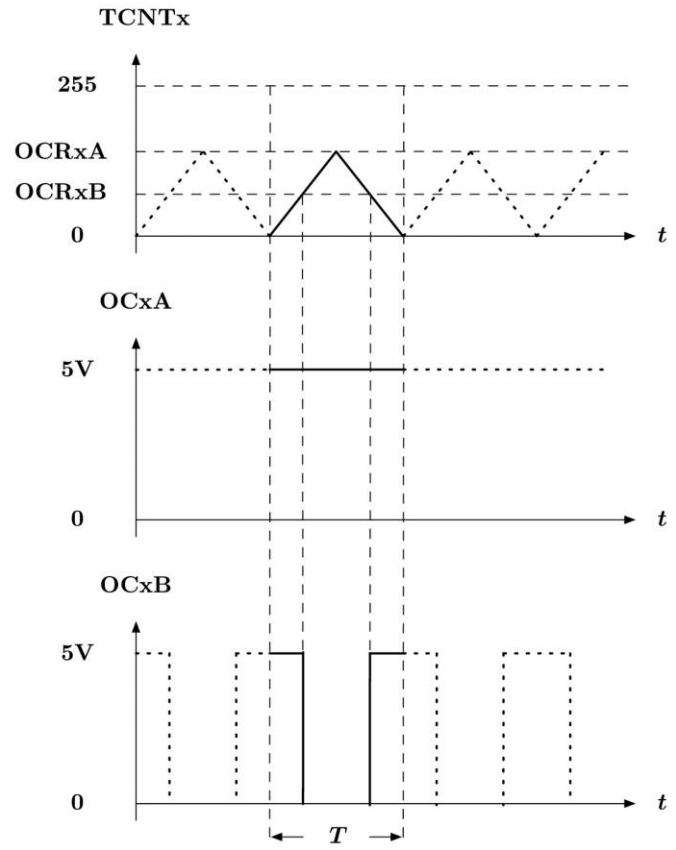




Timer Modes –TOP as OCRA Register

- Setting the appropriate bits in the timer/counter control register (TCCR) causes the counter to reset at the value stored in the output compare register A (OCRnA)
- Output compare register B (OCRnB)controls the pulse width





Controlling the Output Frequency

- The Arduino Mega 2560 clock is 10 MHz
- By default, the system clock is applied directly to the timers
- To gain better control of the clock frequencies applied to the timers, a *prescaler* can be used
- The scaling is enabled by setting the appropriate bits in the lower 3 bits of the TCCRnB register

CS02-0	Description
000	No clock (timer is stopped)
001	No pre-scaling
010	Clock / 8
011	Clock / 64
100	Clock / 256
101	Clock / 1024
110	External clock on T0 pin (falling edge)
111	External clock on T0 pin (rising edge)

Next Time

- Dive into the math and code settings required to use the timers on the Arduino
- Read Chapter 7 in Russel
  - Don’t worry if you don’t understand it right away –you just need to begin getting the general idea ☺

Timers and Counters

Lecture Outline

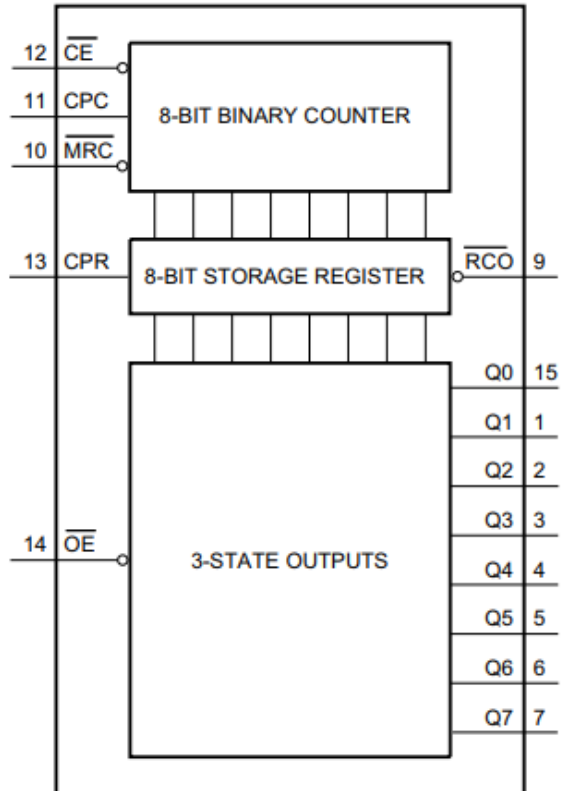
- Counters and Timer Basics
- Timers in the AVR Device Family

Timers and Counters in Embedded Systems

- **Counter:** A circuit that increments or decrements a number based on an input pulse
- **Timer:** A counter that is driven by a constant pulse so the time it takes to count from one number to the next is constant
- Embedded systems make extensive use of timers and counters
  - Measure duration of events
  - Count the number of events
  - Signal / clock generation

An Example Counter IC

74HC590 Counter



- Basic Operation Overview
  - Signal to be counted is sent to the CPC pin the count is incremented
  - When the CPR pin is set to high, the counter value is transferred to the storage register
  - Assuming the output is enabled (OE=0), the register value appears on the output pins Q0-Q7

Table 2. Pin description

Symbol	Pin	Description
Q0 to Q7	15, 1, 2, 3, 4, 5, 6, 7	parallel data output
GND	8	ground (0 V)
$\overline{RCO}$	9	ripple carry output (active LOW)
$\overline{MRC}$	10	master reset counter input (active LOW)
CPC	11	counter clock input (active HIGH)
$\overline{CE}$	12	count enable input (active LOW)
CPR	13	register clock input (active HIGH)
$\overline{OE}$	14	output enable input (active LOW)
V_{CC}	16	supply voltage

Example Counter Timing Diagram

- Typical timing diagram for the 74HC590 Chip
- Sequence starts with RCO going low, indicating that the counter is starting over from 0
- NOTE: Cycles are positive edge triggered

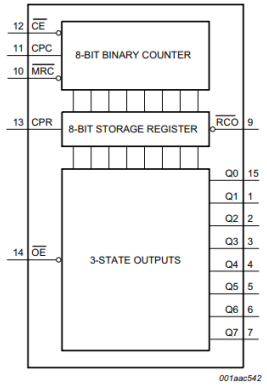
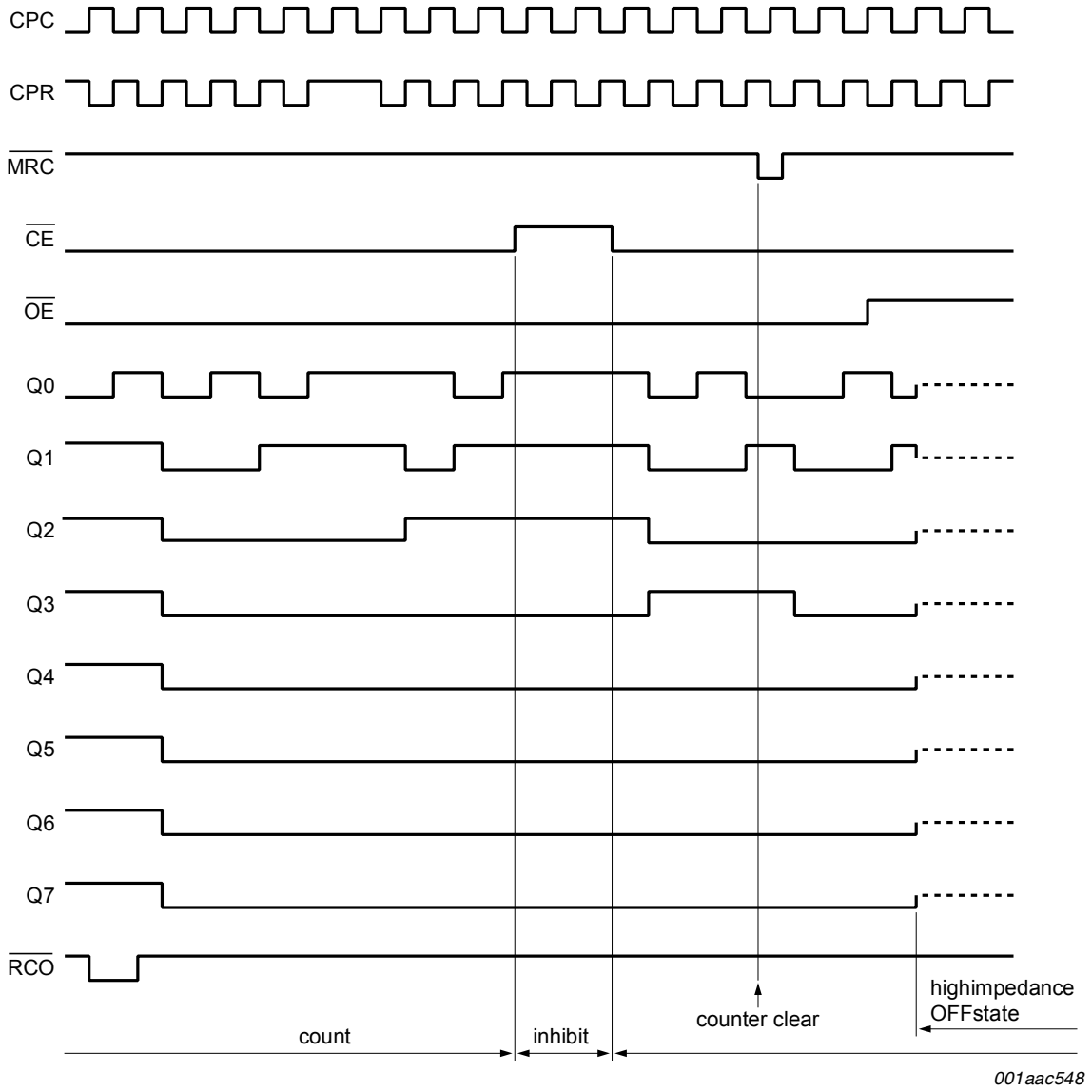


Table 2. Pin description

Symbol	Pin	Description
Q0 to Q7	15, 1, 2, 3, 4, 5, 6, 7	parallel data output
GND	8	ground (0 V)
$\overline{RCO}$	9	ripple carry output (active LOW)
$\overline{MRC}$	10	master reset counter input (active LOW)
CPC	11	counter clock input (active HIGH)
$\overline{CE}$	12	count enable input (active LOW)
CPR	13	register clock input (active HIGH)
$\overline{OE}$	14	output enable input (active LOW)
V_{CC}	16	supply voltage



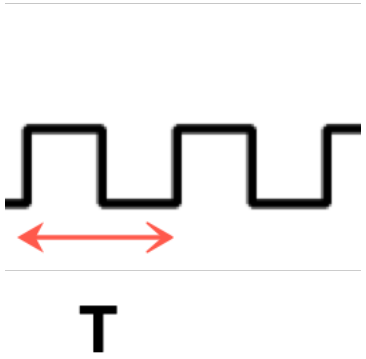
Source: https://assets.nexperia.com/documents/data-sheet/74HC590.pdf

Timers

- Timers are just counters where the counted input is a clock
- Since we know the frequency of the clock, the elapsed time for a given number of ticks can be determined
- Example:

$$f_c = 1 \text{ MHz}$$

$$T = \frac{1}{f_c} = 1\mu s$$



- If we start our timer at 123 and an event happens at count 231, the time elapsed would be (231-123)* 1 μs = 108 μs

The AVR Timers

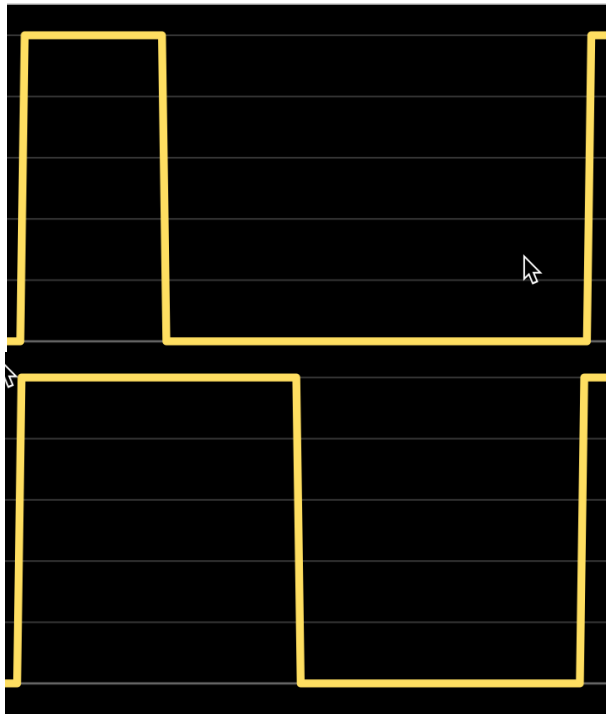
- Timers are hardware peripherals that are embedded in the microcontroller
  - In micro*processor* systems, timers would be in separate chips/systems
- The AVR family has from 1 to 6 timers
- In the 2560, there are 2 8-bit and 4 16-bit counters labelled 0 through 5
  - Timers 0 and 2 are 8-bit, timers 1 and 3-5 are 16 bit

MCU Timer Overview

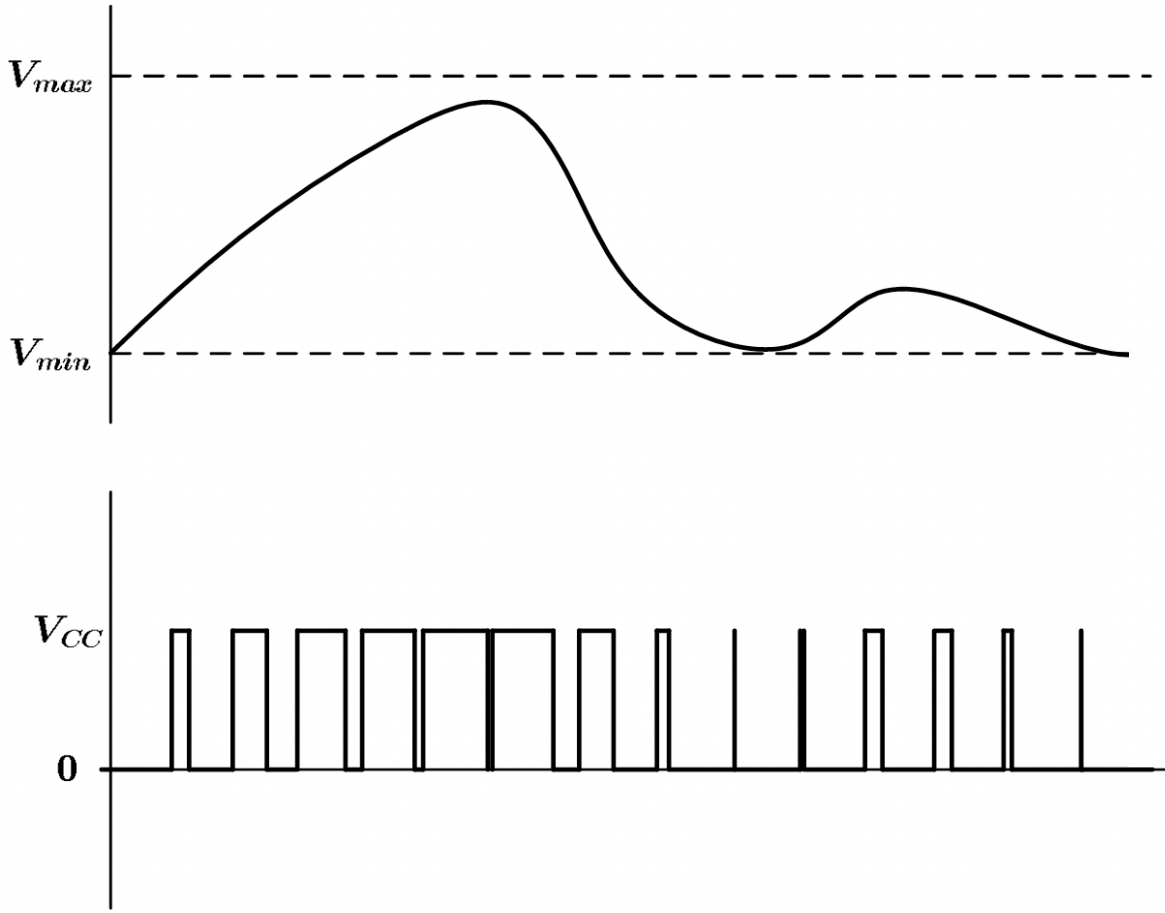
- A timer counts up (or up and down) based on a clock
- When the timer reaches its maximum value, it *rolls over*
    - When this happens, a *flag* is set
- Common approach for creating a delay:
    1. Set an initial value in the counter
    2. Start the counter
    3. Poll for the overflow flag (TOVn)
    4. Stop the counter
    5. Go to (1)

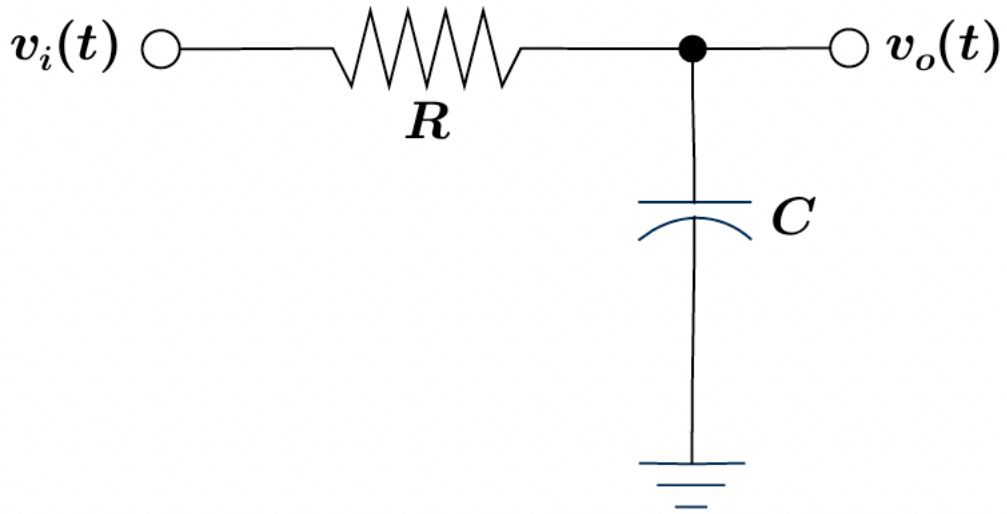
AVR Timer Details

- Each timer has two *channels*: A and B
- The output of these channels are at the same frequency but can have different pulse widths
- Timers have multiple *modes*
  - Modes determine the behavior of the counter/timer, affecting the pulse width, frequency, and relationship to the system clock



Pulse Width Modulation





AVR Registers Related to Timers

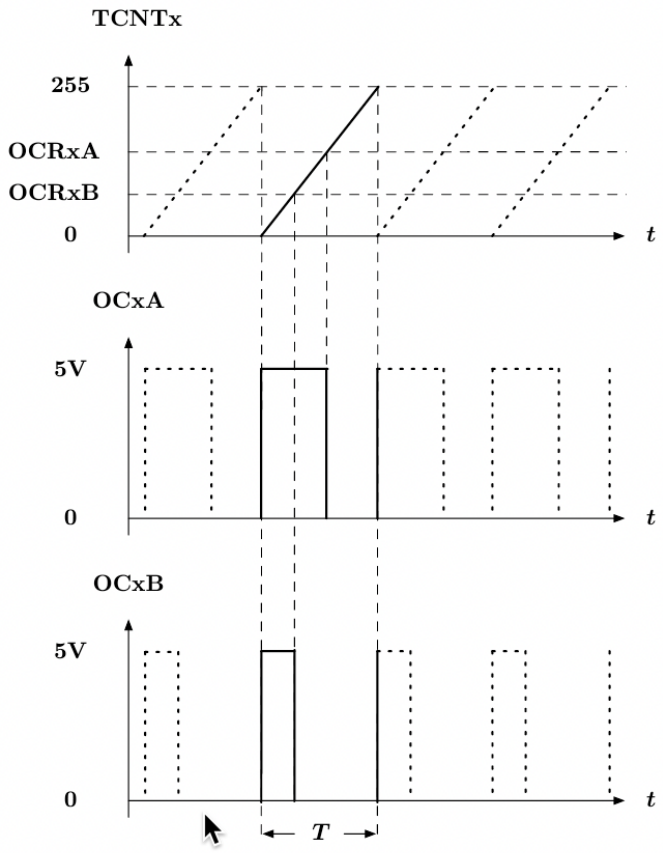
- TCNT = Timer Count Register
- TCCR= Timer/Counter Control Registers
  - Contains bits which control the trimer
  - Start/stop
  - Prescaler
- TIFR = Timer Interrupt Flag Register
- TIMSK = Timer Interrupt Mask Register

Timer Modes

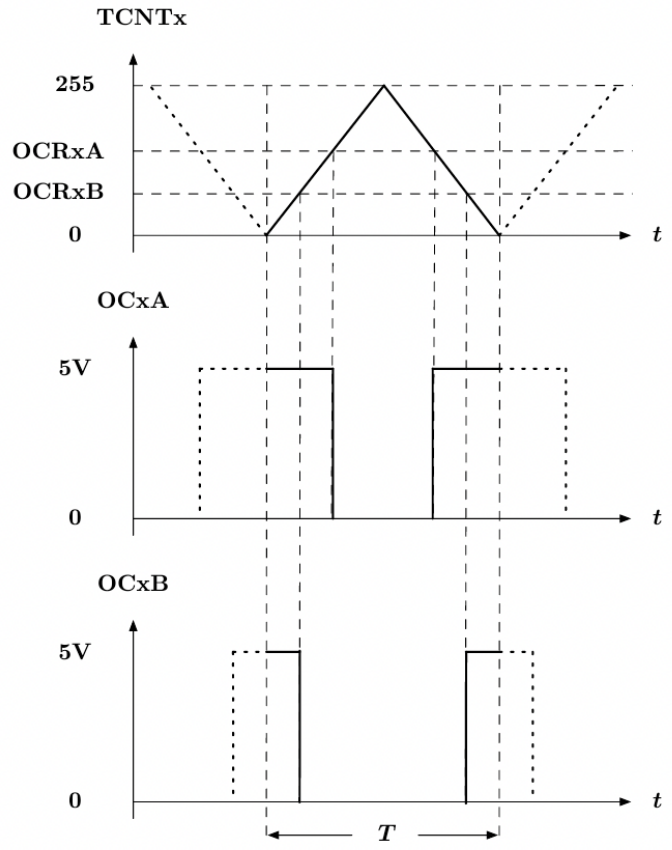
- Normal mode
  - The timer always counts to its maximum value and then starts over
- Clear Timer on Compare Match (CTC) Mode
  - The OCR0A Register is used to manipulate the counter resolution.
  - In CTC mode the counter is cleared to zero when the counter value (TCNT0) matches the OCR0A.
  - OC0A register can be used as output (toggles when reset)
- Fast PWM Mode
  - Used to generate fast PWM signals
  - The timer counts to its maximum value or to a predefined value based on a register settings
    - These settings effect both the frequency and the pulse width
- Phase-Correct PWM
  - The counter counts up and then back down.
  - Register settings control the pulse width
  - The generated pulses are centered on the timer counter
    - In fast PWM modes, the generated pulses are based on the leading edge of the clock

Timer Modes – TOP = $FF

- Setting the appropriate bits in the timer/counter control register (TCCR) causes the counter to its max value (called TOP)
- The output compare registers (OCRnA and B) control the width of the pulse



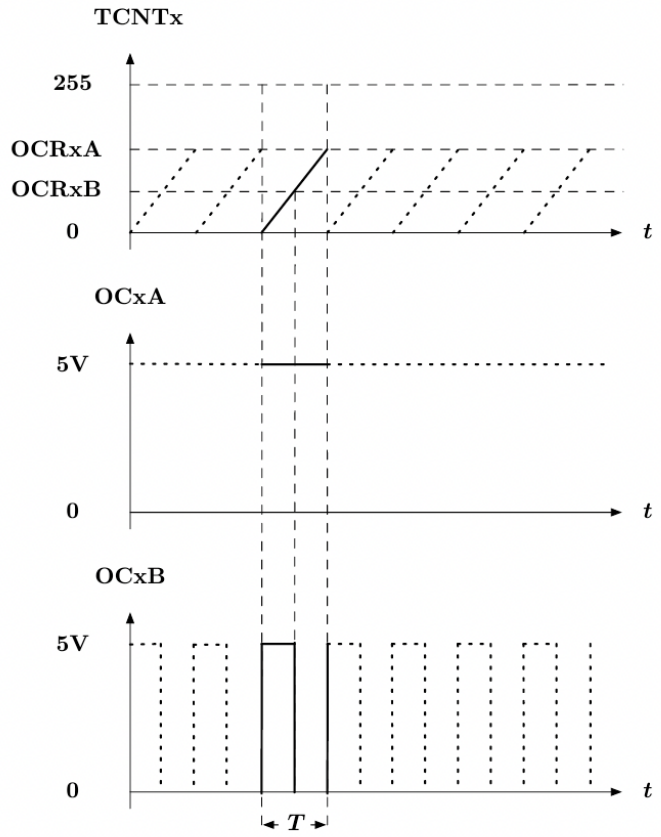
(a) Fast PWM



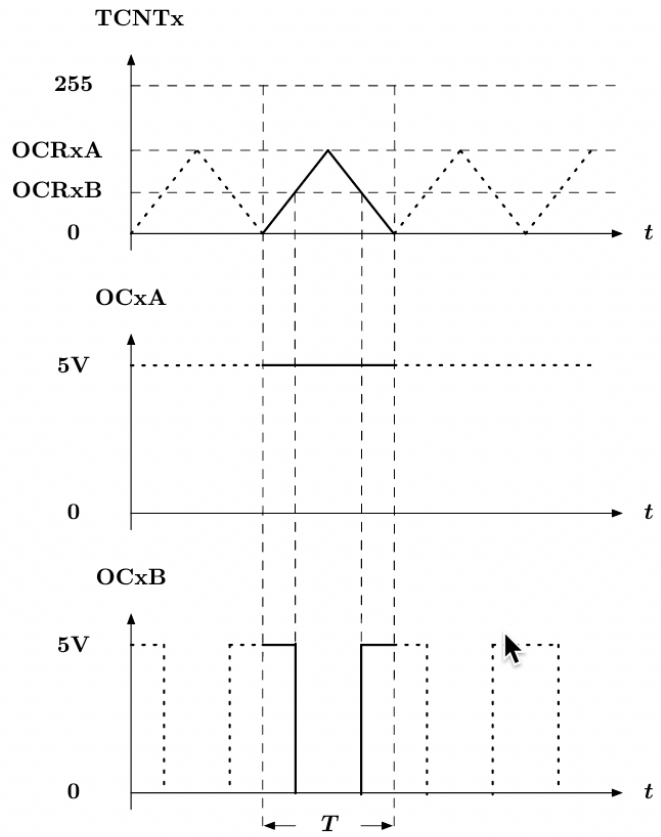
(b) Phase-Correct PWM

Timer Modes – TOP as OCRA Register

- Setting the appropriate bits in the timer/counter control register (TCCR) causes the counter to reset at the value stored in the output compare register A (OCRnA)
- Output compare register B (OCRnB)controls the pulse width



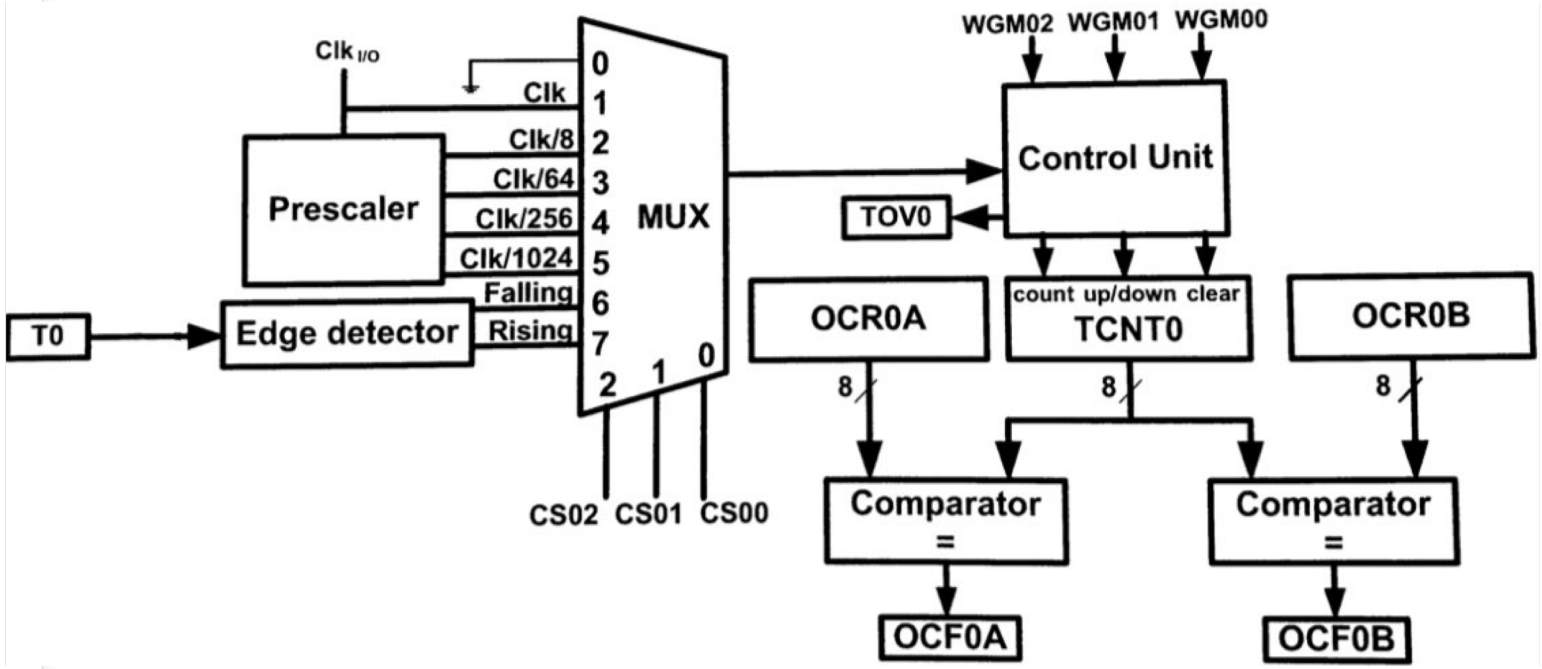
(a) Fast PWM



(b) Phase-Correct PWM

Timer 0 in Detail

- TOV0: Timer 0 overflow flag
- OCR0n: Output compare registers
- TCNT0: Timer 0 counter
- OCF0n: Output comparator flags
- WGM0n: Waveform generator mode bits



The AVR Microcontroller and Embedded Systems Using Assembly and C, Mazadi et al.

Controlling the Output Frequency

- The Arduino Mega 2560 clock is 16 MHz
- By default, the system clock is applied directly to the timers
- To gain better control of the clock frequencies applied to the timers, a *prescaler* can be used
- The scaling is enabled by setting the appropriate bits in the lower 3 bits of the TCCRnB register

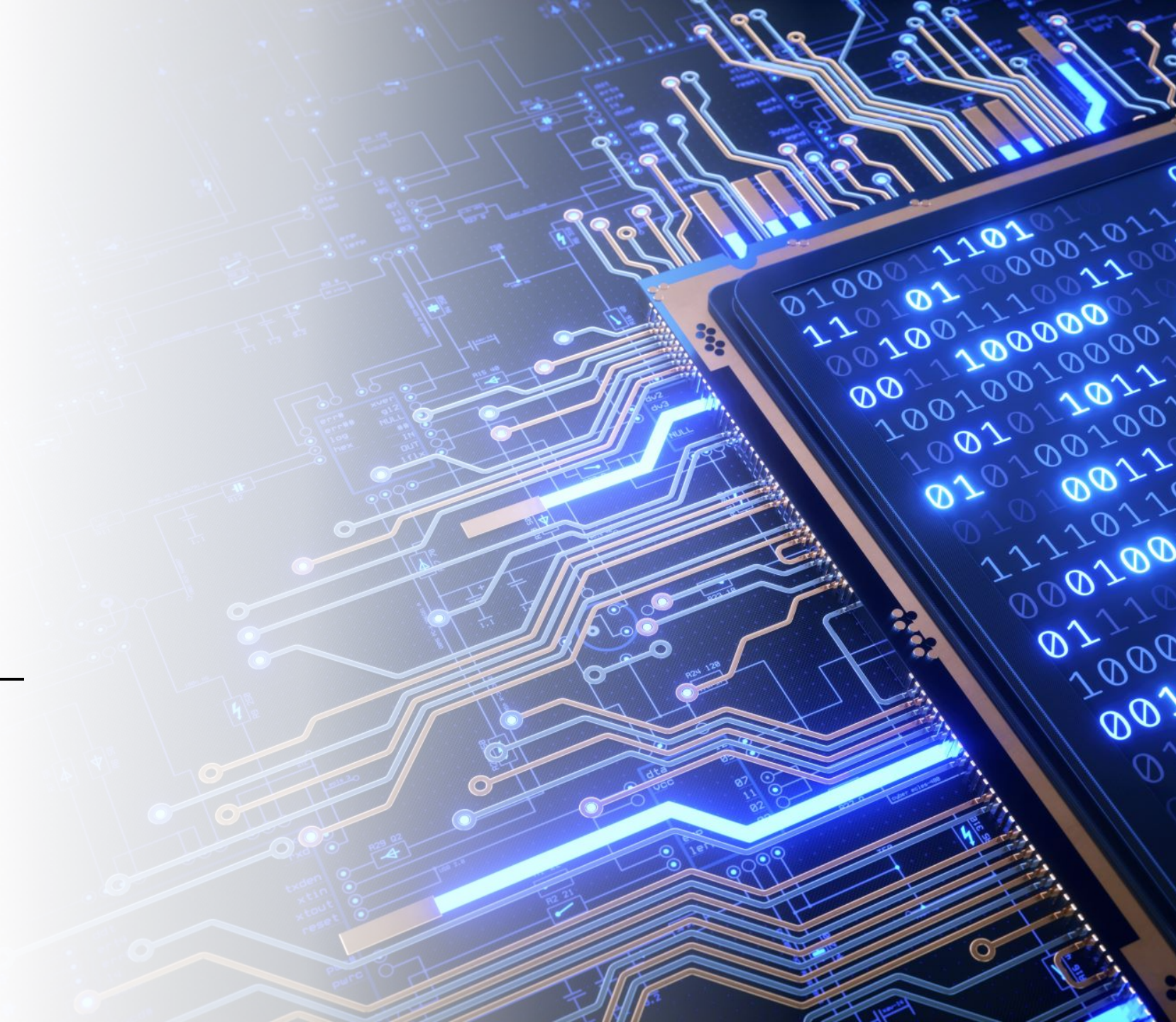
CS02-0	Description
000	No clock (timer is stopped)
001	No pre-scaling
010	Clock / 8
011	Clock / 64
100	Clock / 256
101	Clock / 1024
110	External clock on T0 pin (falling edge)
111	External clock on T0 pin (rising edge)

Next Time

- Dive into the math and code settings required to use the timers on the Arduino
- Read Chapter 9 and 15 in Mazidi, with focus on the C code
- Read Chapter 7 sections 7.4 through 7.4.3 in Jimenez

Timers and Counters (2)

More theory and coding



Lecture Outline

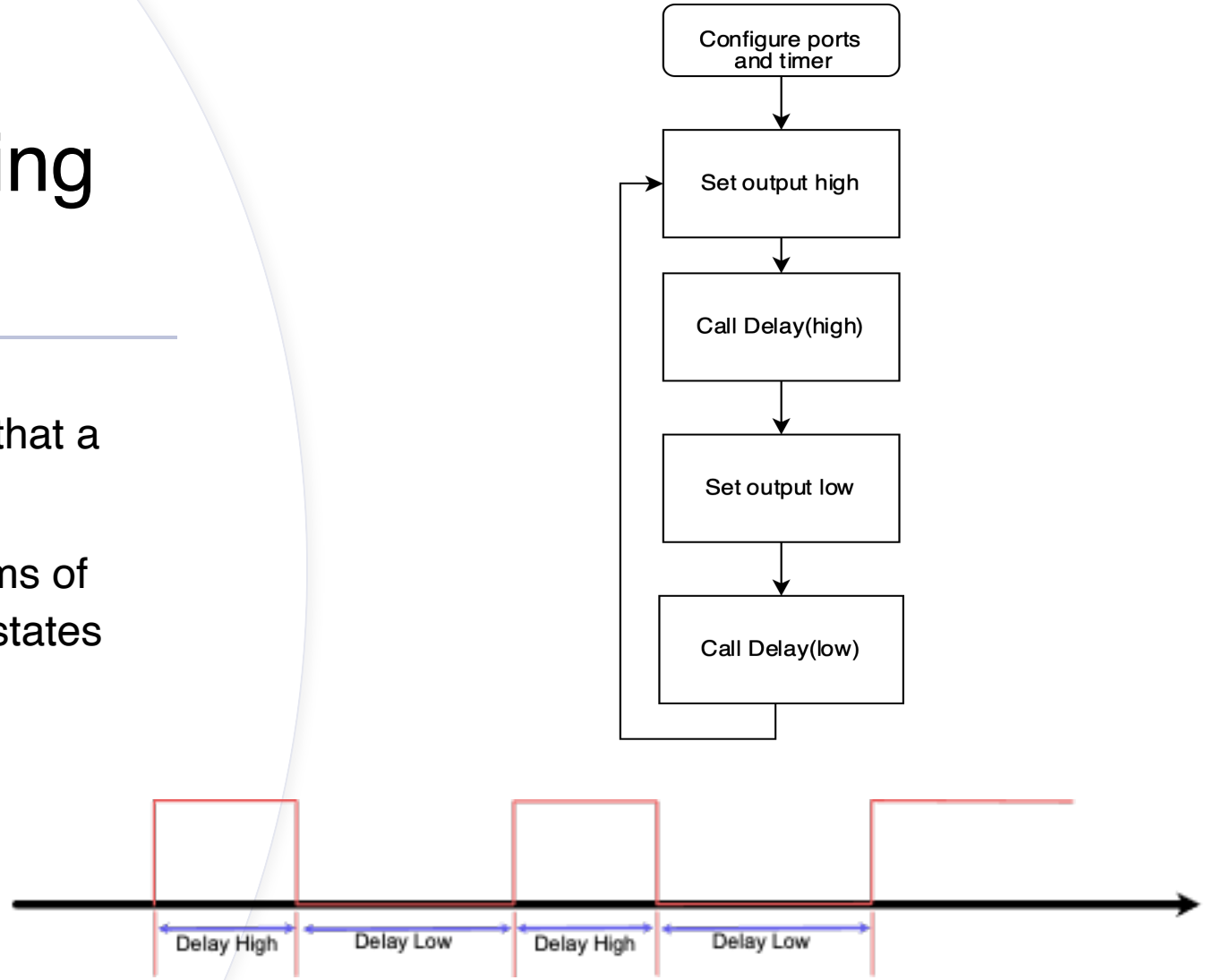
- Calculations for Generating Waveforms in Normal Mode
- Coding the Timers

Signal Generation in Normal Mode

- Recall that in normal mode, the counter (8 or 16 bit) counts to its maximum value (255, or 65535, respectively)
- We use the pre-scaler and the counter *start* value to determine the output frequency and the duty cycle
- Finer control is available in the other modes, where the output compare registers (OCRs) are used set top values, etc.

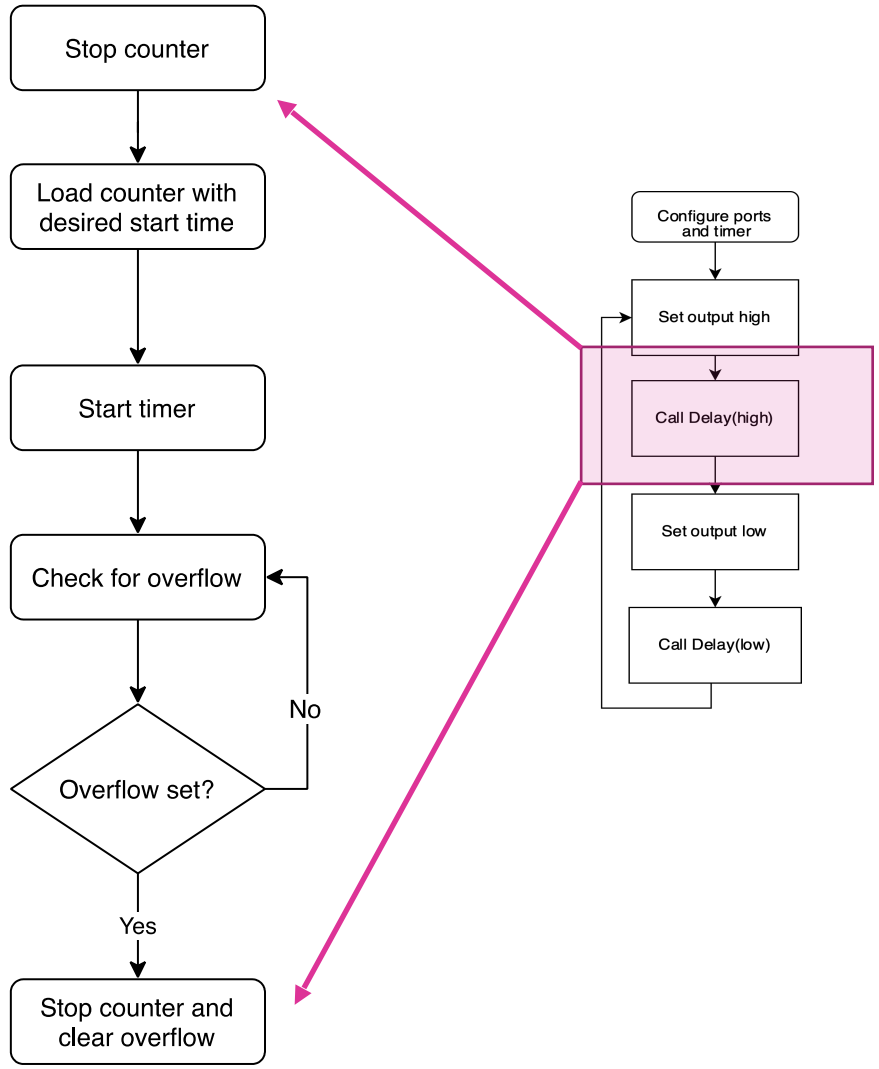
Strategy for Generating a Waveform

- Think in terms of the amount of time that a signal should be high and low
- Calculate the required settings in terms of *delays* required for the high and low states



Creating the Delay

- Determine the period T of the clock
- Determine how many ticks of the clock are required for the desired delay
- Calculate the *start* point for the clock in so that the counter counts up the correct number of ticks before overflowing
- Load the counter with the start point you calculated
- Start the counter and wait for overflow to occur



Calculating the Delay Times

1. f_{wave} = desired output frequency
2. T_{wave} = period = $1/f_{wave}$
3. t_{high} = T * (desired duty cycle)
4. t_{low} = T – t_{high}

Example:

f_{wave} = 100kHz
T_{wave} = 1/100000 = 10μs

For 75% duty cycle,

t_{high} = $10*10^{-6}*0.75$ = 7.5 μs
t_{low} = 2.5 μs

From Time Delay to Counter Ticks

- Goal: 500Hz wave with 50% duty cycle

$$f_{wave} = 500Hz \rightarrow T_{wave} = 0.002\ Sec$$

$$50\%\ Duty\ Cycle \rightarrow T_{wave/2} = 0.001\ Sec$$

$$F_{clk} = 16MHz \rightarrow T_{clk} = 0.0000000625\ Sec$$

$$\frac{T_{wave/2}}{T_{clk}} = \frac{0.001}{0.0000000625} = 16{,}000\ ticks$$

$$16 - bit\ Timer \rightarrow 2^{16} Counts = 65536$$

$$Timer\ Counts\ UP! \rightarrow Load\ Value_{High} = 65536 - 16{,}000 = 49{,}536$$

$$Timer\ Counts\ UP! \rightarrow Load\ Value_{Low} = 65536 - 16{,}000 = 49{,}536$$

From Time Delay to Counter Ticks (2)

- Goal: 2kHz wave with 75% duty cycle

$$F_{wave} = 2kHz \rightarrow T_{wave} = 0.0005\ Sec$$

$$75\%\ Duty\ Cycle \rightarrow T_{wave/4} = 0.00125\ Sec$$
$$F_{clk} = 16MHz \rightarrow T_{clk} = 0.0000000625\ Sec$$

$$\frac{T_{wave/4}}{T_{clk}} = \frac{0.00125}{0.0000000625} = 2{,}000$$

$$16 - bit\ Timer \rightarrow 2^{16} Counts = 65536$$

$$Timer\ Counts\ UP! \rightarrow Load\ Value_{High} = 65{,}536 - (2000 * 3) = 59536$$

$$Timer\ Counts\ UP! \rightarrow Load\ Value_{Low} = 65{,}536 - 2000 = 64536$$

From Time Delay to Counter Ticks (3)

- Goal: 10 Hz wave with 50% duty cycle

$$F_{wave} = 10\ Hz \rightarrow T_{wave} = 0.1\ Sec$$

$$50\%\ Duty\ Cycle \rightarrow T_{wave/2} = 0.05\ Sec$$
$$F_{clk} = 16MHz \rightarrow T_{clk} = 0.0000000625\ Sec$$

$$\frac{T_{wave/2}}{T_{clk}} = \frac{0.05}{0.0000000625} = 800{,}000$$

$$16 - bit\ Timer \rightarrow 2^{16} Counts = 65536$$

$$Timer\ Counts\ UP! \rightarrow Load\ Value_{High} = 65{,}536 - 800{,}000 = Negative$$

$$Timer\ Counts\ UP! \rightarrow Load\ Value_{Low} = 65{,}536 - 800000 = Negative$$

Prescaler is required

Trying Again with Prescaler = 8

- Goal: 10 Hz wave with 50% duty cycle

$$F_{wave} = 10\ Hz \rightarrow T_{wave} = 0.1\ Sec$$

$$50\%\ Duty\ Cycle \rightarrow T_{wave/2} = 0.05\ Sec$$

$$F_{clk} = \frac{16MHz}{8} \rightarrow T_{clk} = 0.0000005\ Sec$$

$$\frac{T_{wave/2}}{T_{clk}} = \frac{0.05}{0.0000005} = 100{,}000$$

Must be < 65535!

Trying Again with Prescaler = 64

- Goal: 10 Hz wave with 50% duty cycle

$$F_{wave} = 10\ Hz \rightarrow T_{wave} = 0.1\ Sec$$

$$50\%\ Duty\ Cycle \rightarrow T_{wave/2} = 0.05\ Sec$$

$$F_{clk} = \frac{16MHz}{\mathbf{64}} \rightarrow T_{clk} = 0.000004\ Sec$$

$$\frac{T_{wave/2}}{T_{clk}} = \frac{0.05}{0.000004} = 12{,}500$$

$$16 - bit\ Timer \rightarrow 2^{16} Counts = 65536$$

$$Timer\ Counts\ UP! \rightarrow Load\ Value_{High} = 65{,}536 - 12{,}500 = 53{,}036$$

$$Timer\ Counts\ UP! \rightarrow Load\ Value_{Low} = 65{,}536 - 12{,}500 = 53{,}036$$

Sanity check: (12,500 * 2) * 0.000004 = 0.1

Clocks/cycle Seconds/clock Seconds/cycle

Prescaling with 16 MHz Clock (16 Bit Timer)

Prescale	Clock F Scaled	Period T (s)	Maximum Delay (s)	Minimum Frequency (Hz)
1	16000000	6.25E-08	0.004096	244.140625
8	2000000	0.0000005	0.032768	30.51757813
64	250000	0.000004	0.262144	3.814697266
256	62500	0.000016	1.048576	0.953674316
1024	15625	0.000064	4.194304	0.238418579

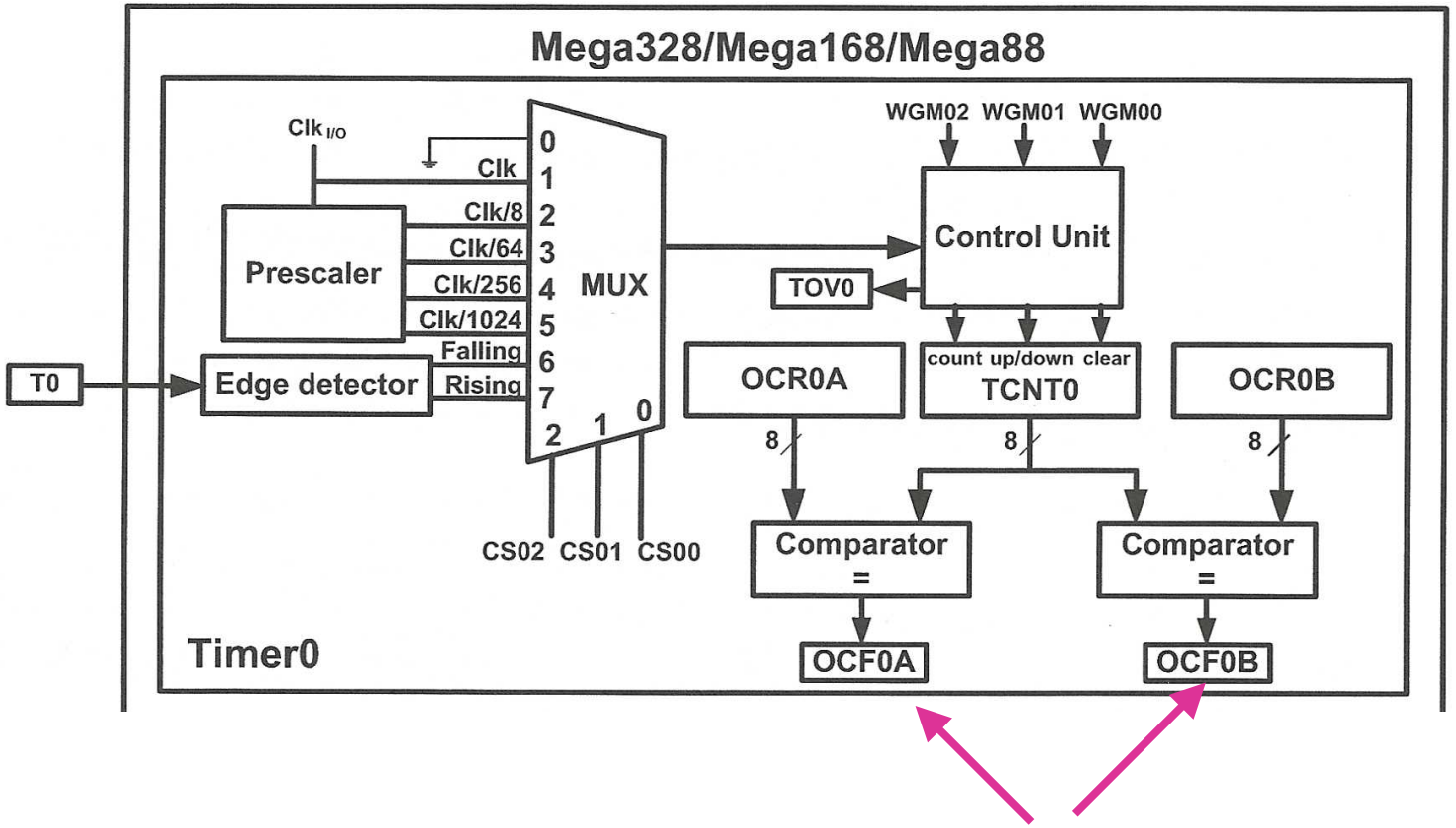
$$T = \frac{1}{F_{scaled}}$$

$$Delay_{max} = T * 2^{16}$$

$$F_{min} = \frac{1}{Delay_{max}}$$

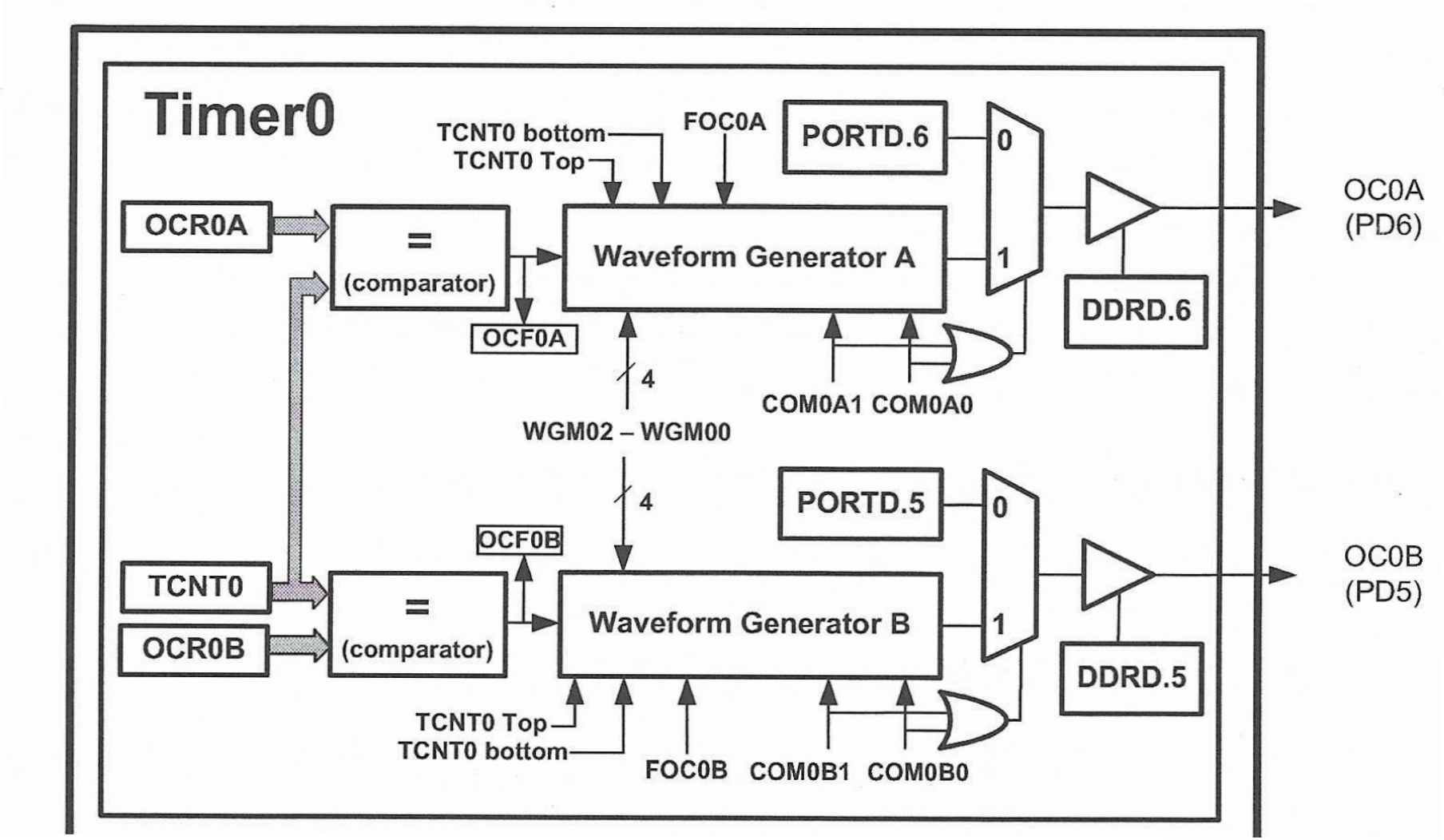
AVR Timer0 Architecture

- TCNT = Timer Count Register
- TCCR= Timer/Counter Control Registers
- OCR = Output Compare Register
- TIFR = Timer Interrupt Flag Register
- TIMSK = Timer Interrupt Mask Register



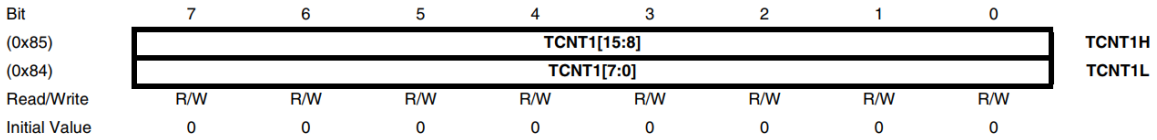
These are flags in the TIFR0 register

Waveform Generator System



The Timer/Counter (TCNT) Registers

- For 8 bit timers, single register
- For 16 bits, two registers required:
  - TCNT1H (high byte for timer 1)
  - TCNT1L (low byte for timer 1)
- Used to read or write the value of the counter

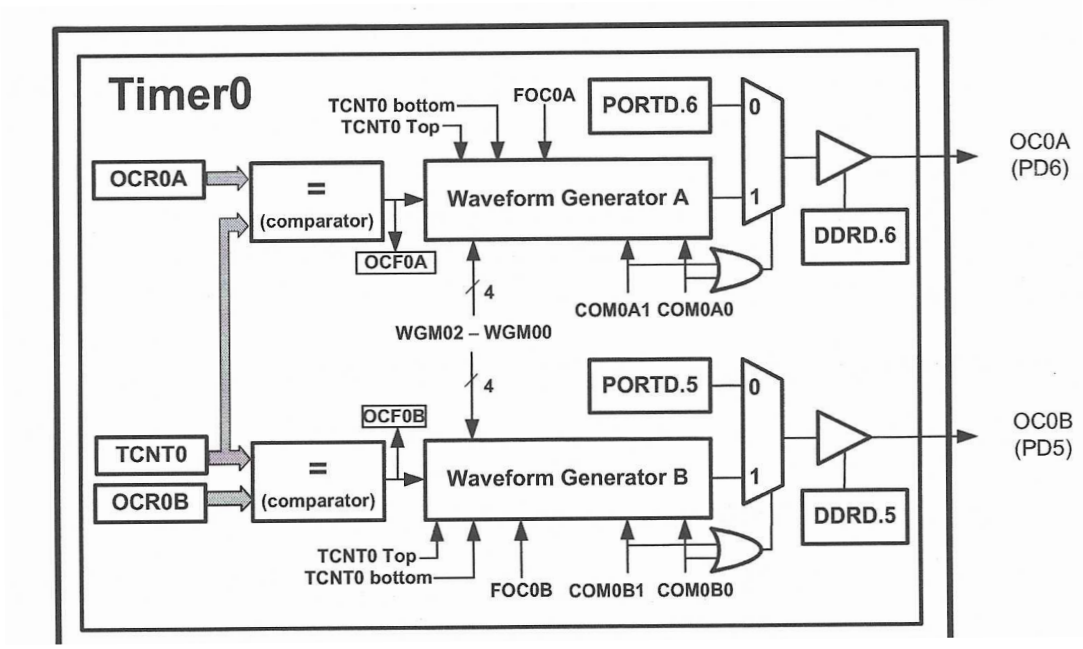


Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

How Do You Load a 16 Bit Register???

- We can only load 8 bits at a time from memory
- Loading only one register at a time means that the high and low bytes might not be part of the same value!
- Solution: When you write into the high byte, the value is stored in a TEMP register. When you write into the low byte, the value in the TEMP register is written into the high byte at the same time
- *Always write the high byte first, then the low*

Timer/Counter Control Register A (TCCRnA)



Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
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 – COMnA1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**
- **Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C**
- **Bit 1:0 – WGMn1:0: Waveform Generation Mode**

Table 17-2. Waveform Generation Mode Bit Description[1]

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM

Table 17-3. Compare Output Mode, non-PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match

Timer/Counter Control Register B TCCRnB

- TCCRB sets prescaler and waveform generation mode

Bit	7	6	5	4	3	2	1	0	
(0x121)	ICNC5	ICES5	–	WGM53	WGM52	CS52	CS51	CS50	TCCR5B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7 - ICNC5 –Input Capture Noise Canceller
- Bit 6 - ICES5 – Input Capture Edge Select
- Bit 4:3 – WGM53:2 – Waveform Generation (see TCCRNA)
- **Bit 2:0 – CS52 – Clock Select**

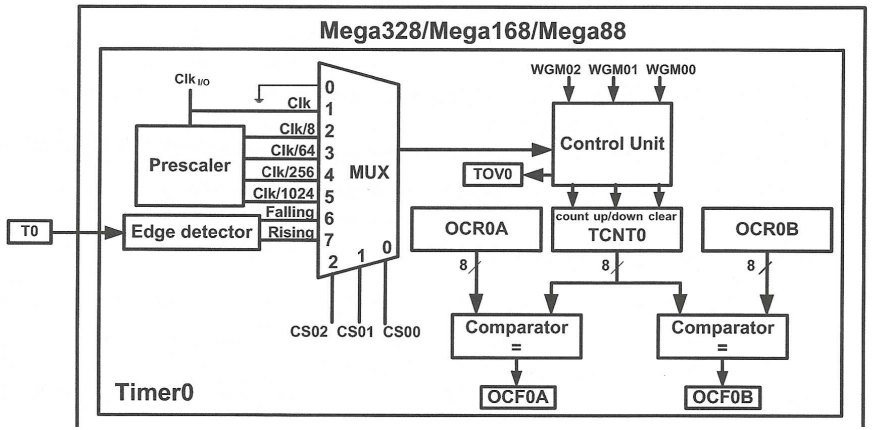


Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (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 Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Timer/Counter Control Register C (TCCRnC)

- TCCRnC only found in 16 bit registers
- For our purposes, set bits 5-7 to 0 for normal mode

Bit	7	6	5	4	3	2	1	0	
(0x122)	FOC5A	FOC5B	FOC3C	–	–	–	–	–	TCCR5C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7 – FOC5A:C – Force Output Compare
  - Set to 0

Timer Interrupt Flag Register (TIFRn)

- The Timer Overflow Flag (TOV) is set when the timer overflows
- For normal mode operation, other bits are not relevant

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	OCF1C	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 5 – ICF1 – Input Capture Flag
- Bit 3:1 – OCF1C:A – Output Compare Match Flag
- **Bit 0 – TOV1 – Timer Overflow 1**

- **Bit 0 – TOVn: Timer/Countern, Overflow Flag**

The setting of this flag is dependent of the WGMn3:0 bits setting. In Normal and CTC modes, the TOVn Flag is set when the timer overflows. Refer to Table 17-2 on page 145 for the TOVn Flag behavior when using another WGMn3:0 bit setting.

TOVn is automatically cleared when the Timer/Countern Overflow Interrupt Vector is executed. Alternatively, TOVn can be cleared by writing a logic one to its bit location.

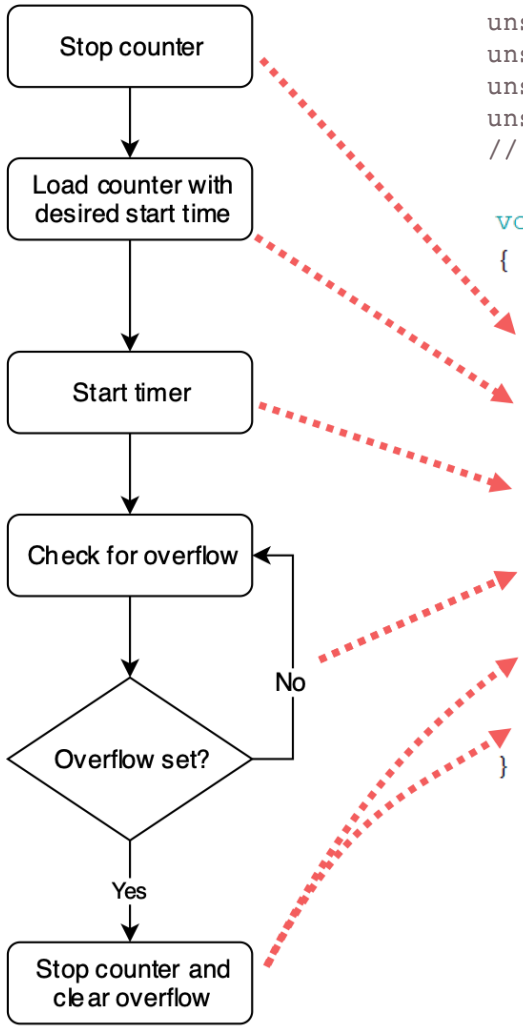
Timer Interrupt Mask Register (TIMSKn)

- The TIMSK register is used to enable/disable interrupts
- Setting bit 0, TOIE0, enables the overflow interrupt (TOV from before)

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 2:1 – OCIE0B:A – Output Compare Interrupt Enable
- **Bit 0 – TOIE0 – Timer Overflow Interrupt Enable**

The Delay Subroutine




```
// initialize pointers to registers
unsigned char *myTCCR1A = (unsigned char *) 0x80;
unsigned char *myTCCR1B = (unsigned char *) 0x81;
unsigned char *myTCCR1C = (unsigned char *) 0x82;
unsigned char *myTIMSK1 = (unsigned char *) 0x6F;
unsigned int  *myTCNT1  = (unsigned  int *) 0x84;
unsigned char *myTIFR1 =  (unsigned char *) 0x36;
unsigned char *portDDRB = (unsigned char *) 0x24;
unsigned char *portB =    (unsigned char *) 0x25;
// setup() and loop() not shown

void my_delay(unsigned int ticks)
{
  // stop the timer (set prescaler to 000)
  *myTCCR1B &= 0xF8; // 0b1111 1000
  // set the counts (16 bits -> 2 addresses!)
  *myTCNT1 = (unsigned int) (65536 - ticks);
  // start the timer (set prescaler to 001)
  * myTCCR1B |= 0x01; //0b0000 0001
  // wait for overflow
  while((*myTIFR1 & 0x01)==0); // 0b0000 0001
  // stop the timer (set prescaler to 000)
  *myTCCR1B &= 0xF8; // 0b1111 1000
  // reset TOV (write a 1 to reset to 0)
  *myTIFR1 |= 0x01; // 0b0000 0001
}
```