EE-222: Microprocessor Systems

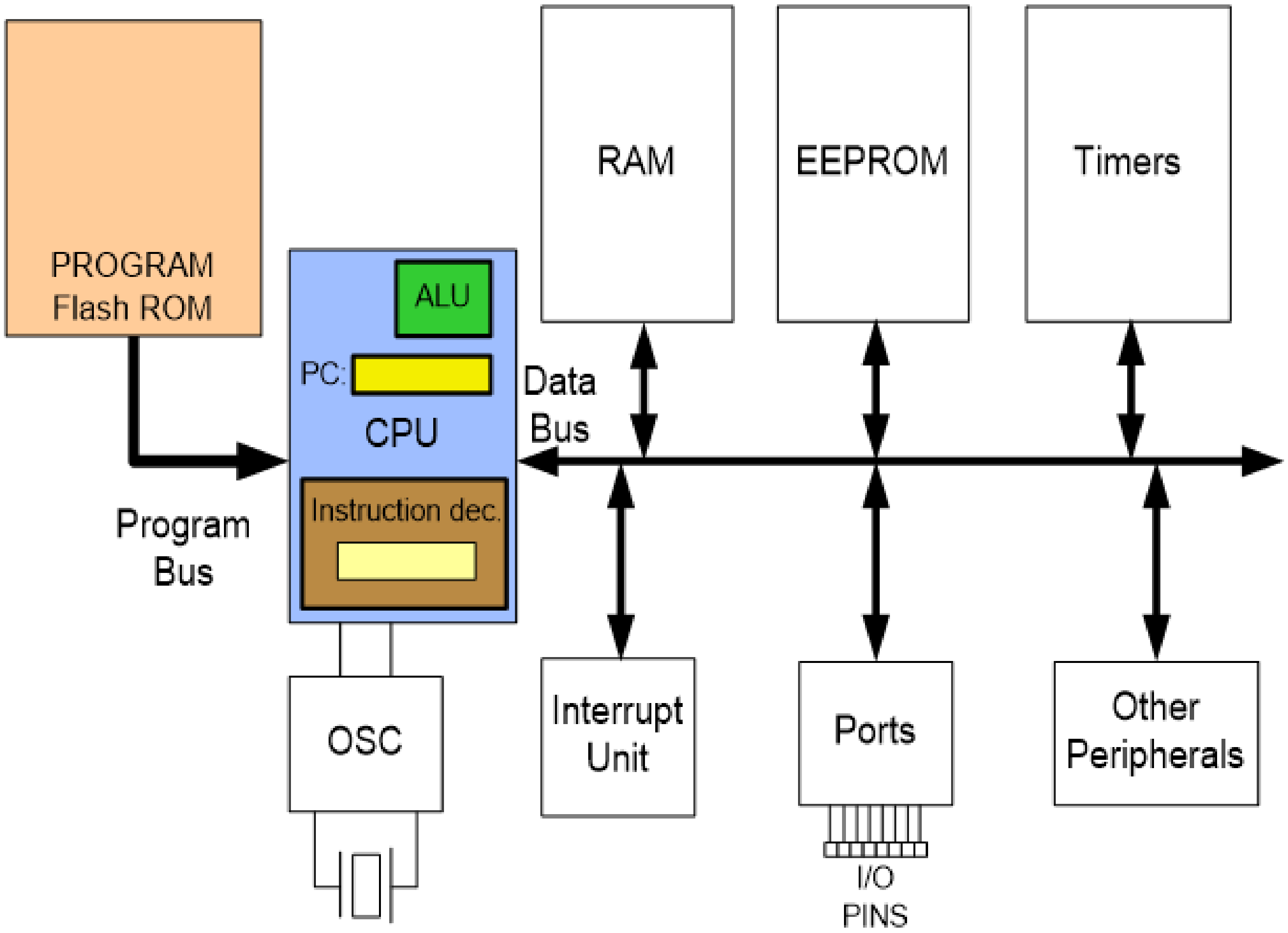
AVR Timers

Instructor: Dr. Arbab Latif

NUST

SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

Timers

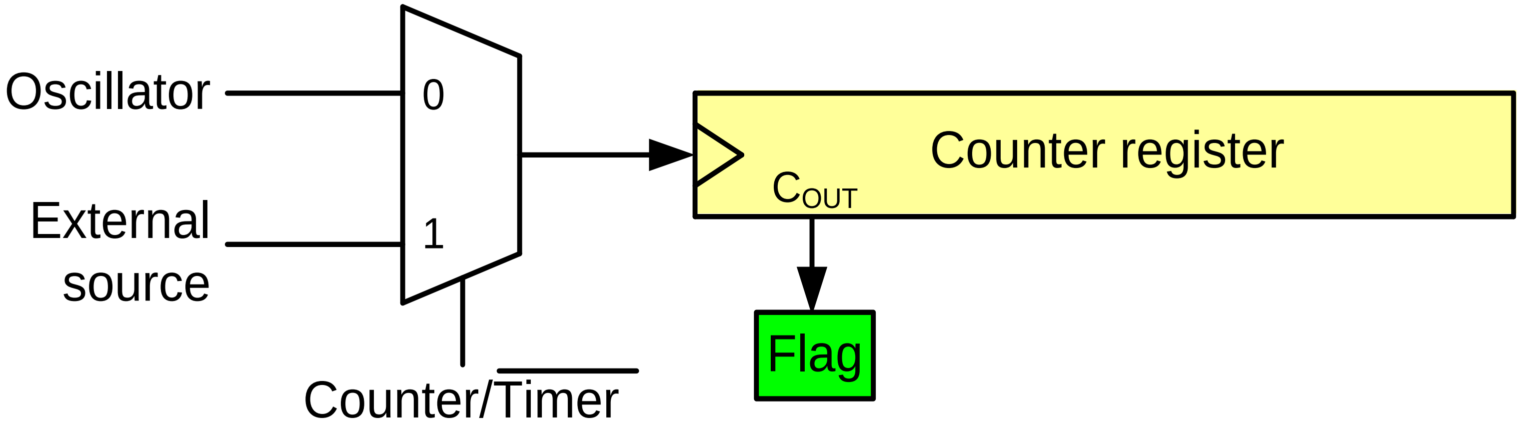

PROGRAM
Flash ROM
ALU
PC:
CPU
Instruction dec.
Program
Bus
Data
Bus
RAM
EEPROM
Timers
OSC
Interrupt
Unit
Ports
I/O
PINS
Other
Peripherals

Timers: Why do we need them?

- Provide accurately timed delays or actions independent of code execution time
- How are Timers used?
  - Accurate delay
    - Read the timer, store value as K. Loop until timer reaches the delay value.
  - Schedule important events
    - Setup an *Output Compare* to trigger an interrupt at a precise time
  - Measure time between events
    - When event#1 happens, store timer value as K
    - When event#2 happens, read timer value and subtract K
    - The difference is the time elapsed between the two events

A Generic Timer/Counter

- Delay generating
- Counting
- Wave-form generating



- Counting Event: Connect the external source to the clock pin of the counter register

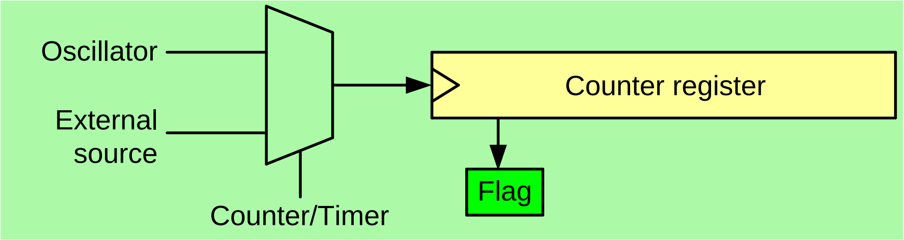
- Generate time delays: Connect oscillator to the clock pin of the counter

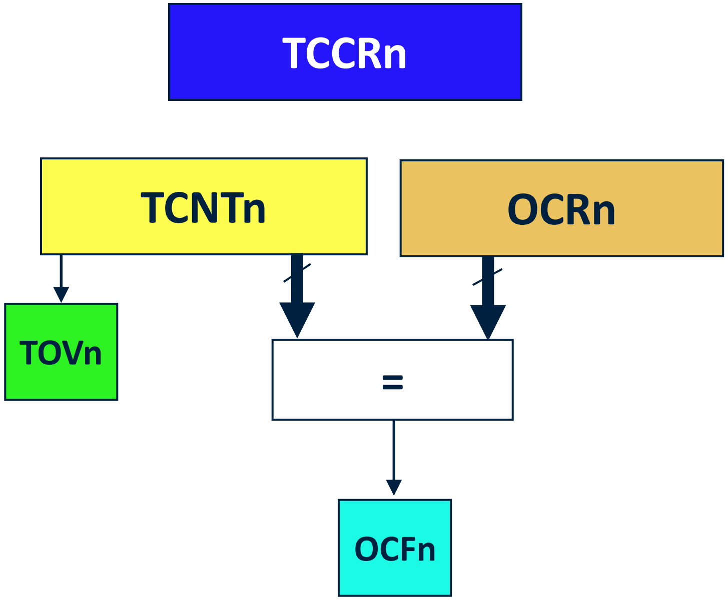
Overview of Atmega16 Timers

	Timer 0	Timer 1	Timer 2
Overall	- 8-bit counter - 10-bit prescaler	- 16-bit counter - 10-bit prescaler	- 8-bit counter - 10-bit prescaler
Functions	- PWM - Frequency generation - Event counter - Output compare	- PWM - Frequency generation - Event counter - Output compare 2 channels - Input capture	- PWM - Frequency generation - Event counter - Output compare
Operation modes	- Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM	- Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM	- Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM

Timer Registers

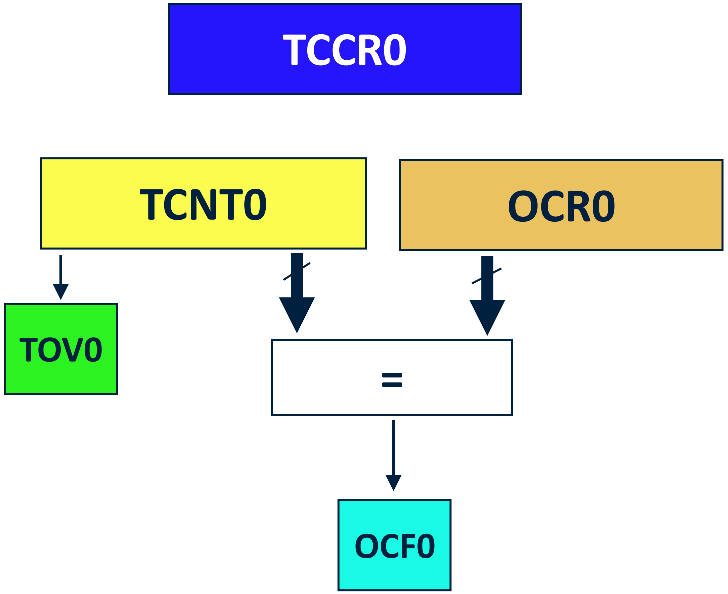
- TCNTn (Timer/Counter register)
- TOVn (Timer Overflow flag)
- TCCRn (Timer Counter control register)
- OCRn (output compare register)
- OCFn (output compare match flag)





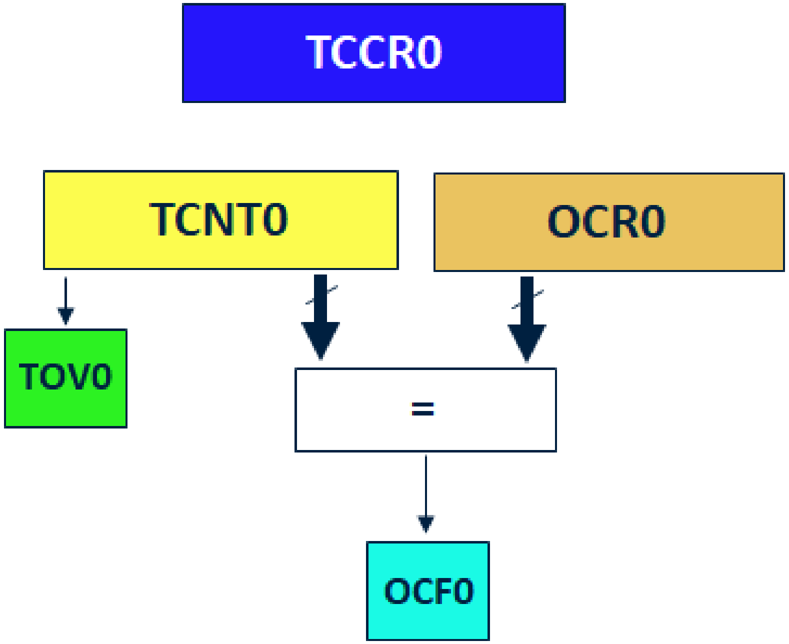
Programming Timer 0

Basic Registers: AVR Timer/Counter 0



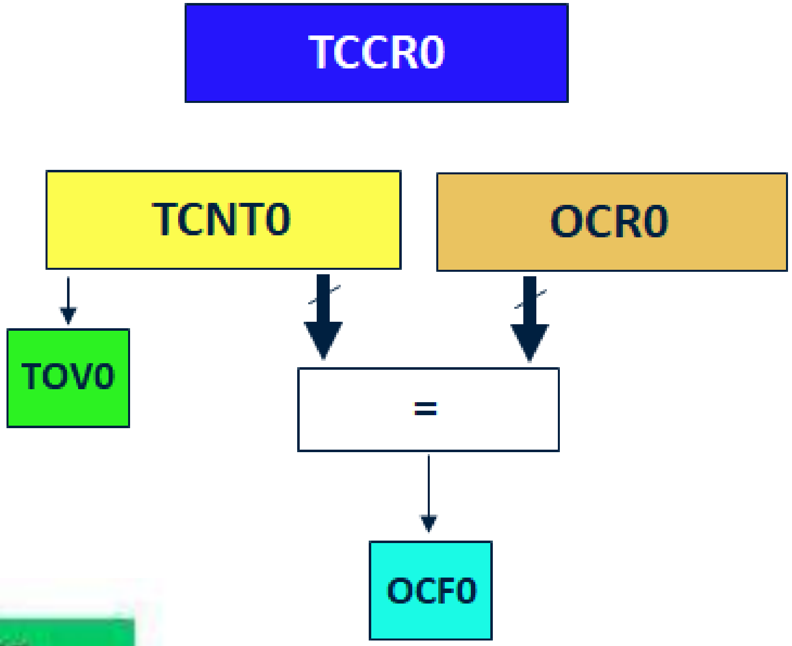
TCNT0 Register

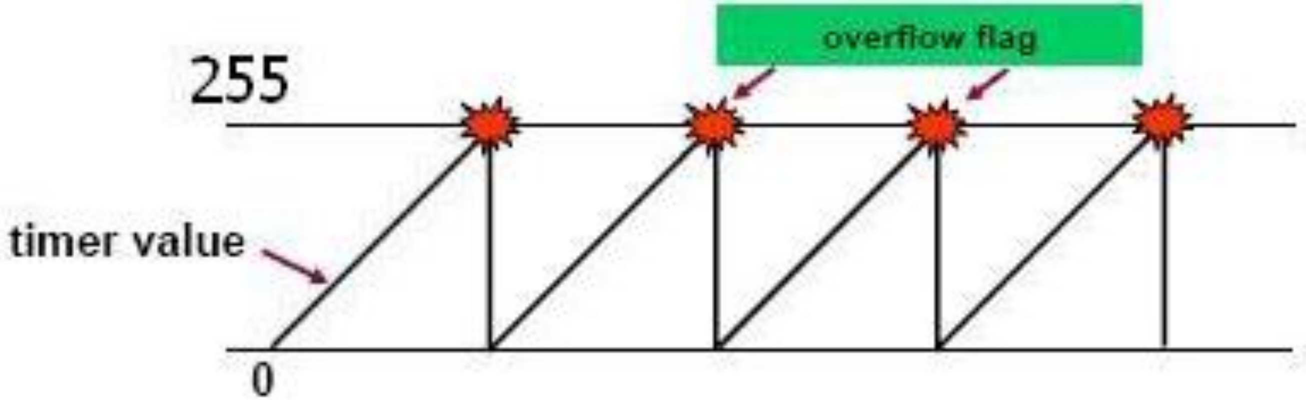
- TCNT0 [Timer/Counter] Register:
  - R/W
  - ZERO upon RESET
  - Contents of timer/counter can be accessed through this register.



TOV0 Flag

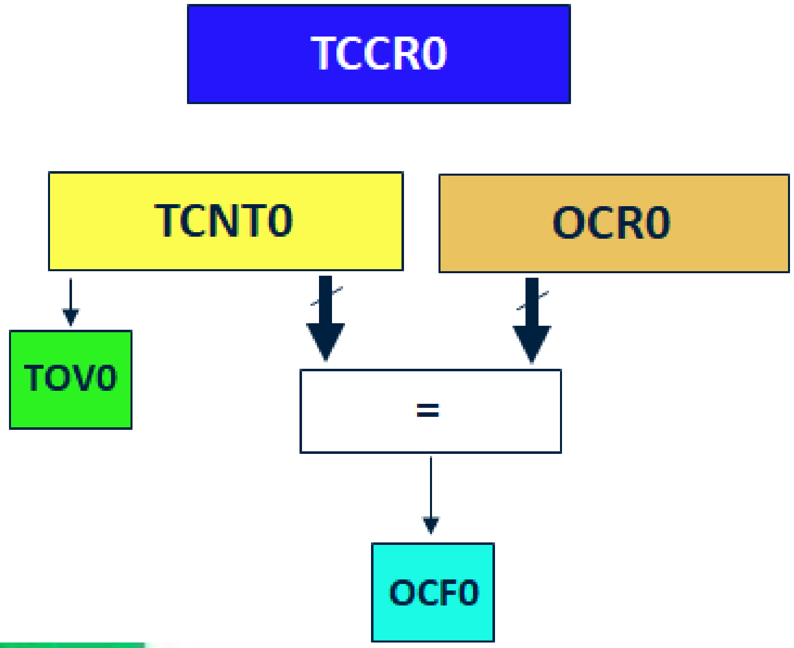
- TOV0 [Timer Overflow] Flag Register:
  - TOV0 sets, when a timer overflows

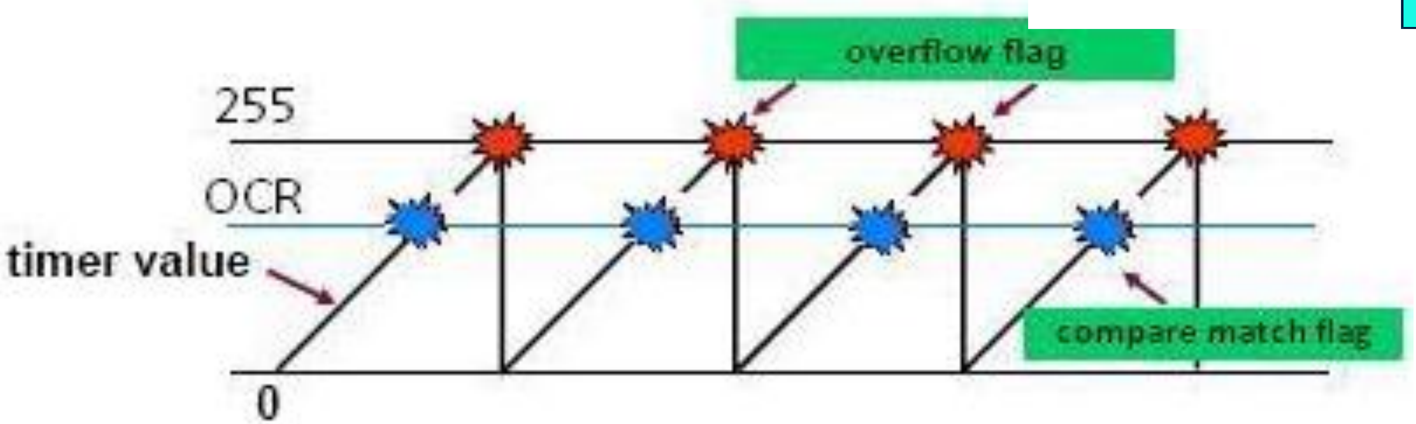




OCR0 Flag

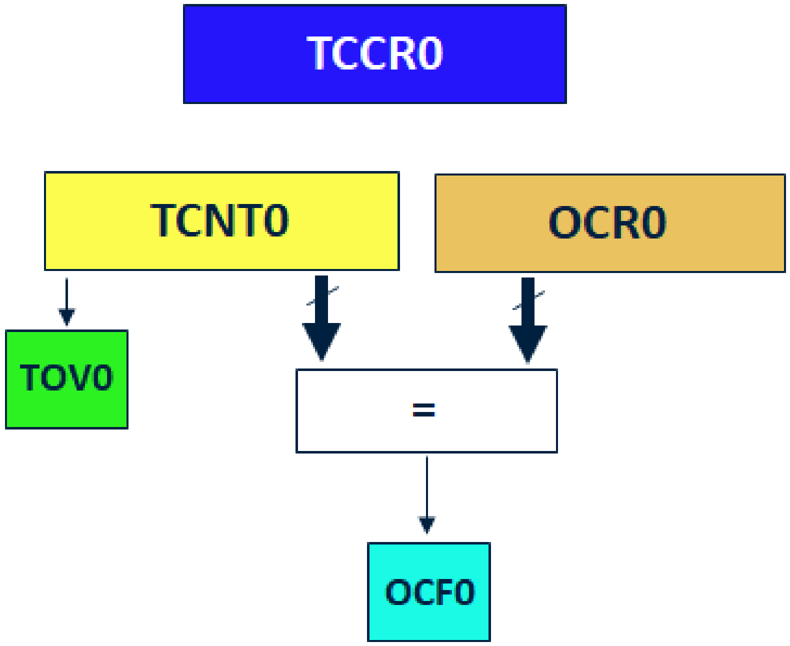
- OCR0 [Output Compare] Flag:
  - The content of the OCR0 is compared with the contents of the TCNT0
    - OCR0 flag is set when both are equal.





TCCR0 Register

- TCCR0 [Timer/Counter Control] Register:
  - Used for various settings (see next)



FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0

TCCR0: Clock Selector

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

Clock Selector (CS)

CS02	CS01	CS00	Comment
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 64
1	0	0	clk / 256
1	0	1	clk / 1024
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

TCCR0: Mode Selector

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0

Timer Mode (WGM)

WGM00	WGM01	Comment
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	Phase correct PWM
1	1	Fast PWM

Timer Registers in the I/O Register Memory

Address Mem.	Address I/O	Name
$20	$00	TWBR
$21	$01	TWSR
$22	$02	TWAR
$23	$03	TWDR
$24	$04	ADCL
$25	$05	ADCH
$26		
$27		
$28		
$29		
$2A		
$2B		
$2C	$0C	UDR
$2D	$0D	SPCR
$2E	$0E	SPSR
$2F	$0F	SPDR
$30	$10	PIND
$31	$11	DDRD
$32	$12	PORTD
$33	$13	PINC
$34	$14	DDRC
$35	$15	PORTC

Address Mem.	Address I/O	Name
$36	$16	PINB
$37	$17	DDRB
$38	$18	PORTB
$39	$19	PINA
$3A	$1A	DDRA
$3B	$1B	PORTA
$41	$21	WDTCR
$42	$22	ASSR
$43	$23	OCR2
$44	$24	TCNT2
$45	$25	TCCR2
$46	$26	ICR1L
$47	$27	ICR1H
$48	$28	OCR1BL
$49	$29	OCR1BH
$4A	$2A	OCR1AL

Address Mem.	Address I/O	Name
$4B	$2B	OCR1AH
$4C	$2C	TCNT1L
$4D	$2D	TCNT1H
$4E	$2E	TCCR1B
$4F	$2F	TCCR1A
$50	$30	SFIOR
$51	$31	OCDR / OSCCAL
$52	$32	TCNT0
$53	$33	TCCR0
$54	$34	MCUCSR
$55	$35	MCUCR
$56	$36	TWCR
$57	$37	SPMCR
$58	$38	TIFR
$59	$39	TIMSK
$5A	$3A	GIFR
$5B	$3B	GICR
$5C	$3C	OCR0
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

Accessing Timer Registers using IN and OUT instructions i.e

```
LDI R20, 25
OUT TCNT0, R20
```

Note: Although memory address $20-$5F is set aside for I /O registers (SFR) we can access them as I /O locations with addresses starting at $00.

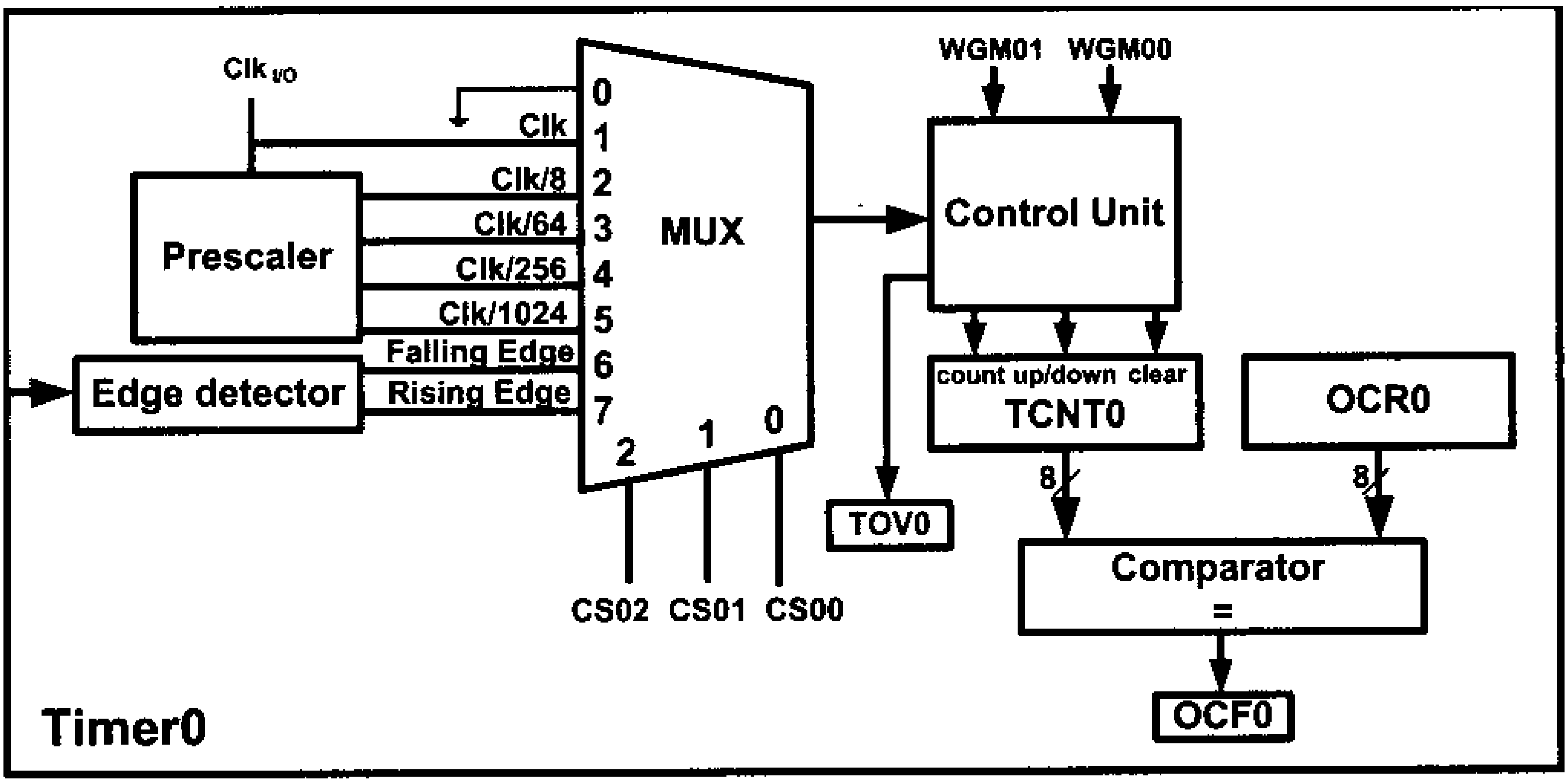
TIFR (Timer/Counter Interrupt Flag Register)

- TOV0 and OCF0 are part of TIFR register

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
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

TOV0	D0	Timer0 overflow flag bit
		0 = Timer0 did not overflow.
		1 = Timer0 has overflowed (going from \$FF to \$00).
OCF0	D1	Timer0 output compare flag bit
		0 = compare match did not occur.
		1 = compare match occurred.
TOV1	D2	Timer1 overflow flag bit
OCF1B	D3	Timer1 output compare B match flag
OCF1A	D4	Timer1 output compare A match flag
ICF1	D5	Input Capture flag
TOV2	D6	Timer2 overflow flag
OCF2	D7	Timer2 output compare match flag

Overall: Timer 0 Hardware Organization



Class Activity

- Find the value of TCCR0 if we want to program Timer0 in:
  - Normal Mode
  - No prescaler
  - Use AVR's Crystal Oscillator for the clock source

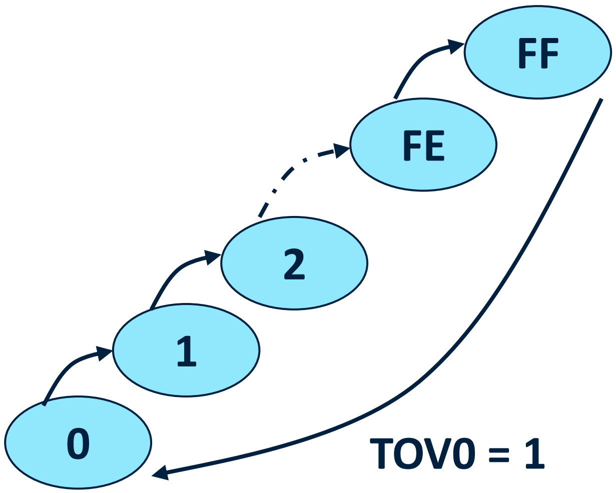
- Sol:

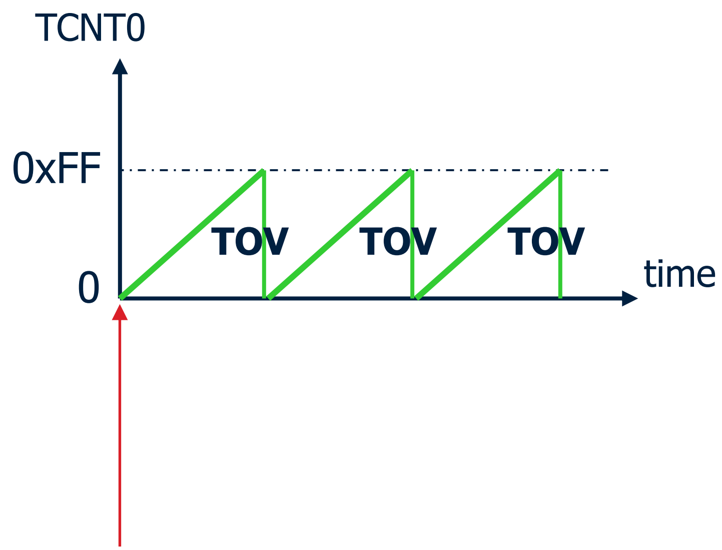
TCCR0 =

0	0	0	0	0	0	0	1
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

Steps to Program Timer 0 in Normal Mode

Normal mode





TOV0: 1

Steps to Program Timer0 in Normal Mode

1. Load the TCNT0 with the initial count value.

2. Configure timer/counter mode through TCCR0 register.

3. Keep monitoring the timer overflow flag (TOV0):
   - Get out of the loop when TOV0 becomes high

4. Stop the timer by disconnecting the clock source:
   - LDI R20, 0x00
   - TCCR0,R20

5. Clear the TOV0 flag for the next round.

6. Go back to Step 1 to load TCNT0 again.

Timer 0 Demo

1. Load the TCNT0
2. Configure TCCR0 register
3. Monitor TOV0
4. Stop the timer
5. Clear the TOV0

```
#include <avr/io.h>
int main()
{
 TCNT0 = 0xF2;
 TCCR0 = 0x01;    //WGM=0000 (Normal)
 while ((TIFR & (1 << TOV0)) ==0)
//wait for TOV0 to roll over
 TCNT0 = 0;
 TIFR = 0x01;
}
```

```
LDI R20, 0xF2
OUT TCNT0, R20

LDI R20, 0x01
OUT TCCR0, R20

AGAIN: IN R20,TIFR
       SBRS R20,TOV0
       RJMP AGAIN

LDI R20,0x0
OUT TCCR0,R20

LDI R20,0x01
OUT TIFR, R20
```

In example 1 calculate the delay. XTAL = 10 MHz.

Solution 1 (inaccurate)**:**

1) **Calculating T:**

T = 1/f = 1/10M = 0.1µs

2) **Calculating num of machine cycles:**

$$\begin{array}{r} \$100 \\ -\$F2 \\ \hline \$0E = 14 \end{array}$$

3) **Calculating delay**

14 * 0.1µs = 1.4 0µs

```
        LDI     R16,0x20
        SBI     DDRB,5   ;PB5 as an output
        LDI     R17,0
        OUT     PORTB,R17
BEGIN:  LDI     R20,0xF2
        OUT     TCNT0,R20          ;load timer0
        LDI     R20,0x0
        OUT     TCCR0A,R20
        LDI     R20,0x01
        OUT     TCCR0B,R20 ;Normal mode, inter. clk
AGAIN:  SBIS    TIFR0,TOV0 ;if TOV0 is set skip next
        RJMP    AGAIN
        LDI     R20,0x0
        OUT     TCCR0B,R20         ;stop Timer0
        LDI     R20,(1<<TOV0)      ;R20 = 0x01
        OUT     TIFR0,R20          ;clear TOV0 flag

        EOR     R17,R16            ;toggle D5 of R17
        OUT     PORTB,R17          ;toggle PB5
        RJMP    BEGIN
```

Accurate calculating

Other than timer, executing the instructions consumes time; so if we want to calculate the accurate delay a program causes we should add the delay caused by instructions to the delay caused by the timer

	LDI	**R16,0x20**		
	SBI	**DDRB,5**		
	LDI	**R17,0**		
	OUT	**PORTB,R17**		
BEGIN:	**LDI**	**R20,0xF2**		**1**
	OUT	**TCNT0,R20**		**1**
	LDI	**R20,0x00**		**1**
	OUT	**TCCR0A,R20**	**1**	
	LDI	**R20,0x01**		**1**
	OUT	**TCCR0B,R20**	**1**	
AGAIN:	**SBIS**	**TIFR0,TOV0**		**1 / 2**
	RJMP	**AGAIN**		**2**
	LDI	**R20,0x0**		**1**
	OUT	**TCCR0B,R20**	**1**	
	LDI	**R20,0x01**		**1**
	OUT	**TIFR0,R20**		**1**
	EOR	**R17,R16**		**1**
	OUT	**PORTB,R17**		**1**
	RJMP	**BEGIN**		**2**
				18

Delay caused by timer = 14 * 0.1μs = 1.4 μs **Delay caused by instructions = 18 * 0.1μs = 1.8**

Total delay = 3.2 μs ➔ wave period = 2*3.2 μs = 6.4 μs ➔ wave frequency = 156.25 KHz

Finding values to be loaded into the Timer

1. Calculate the period of clock source.
   - Period = 1 / Frequency
     - E.g. For XTAL = 16 MHz ➔ T = 1/16MHz
2. Divide the desired time delay by period of clock.
3. Perform 256 - n, where n is the decimal value we got in Step 2.
4. Set TCNT0 = 256 - n

Example

- Assuming XTAL = 8 Mhz, write a program to generate a square wave with a period of 12.5 us on PIN PORTB.3.

Solution

For a square wave with T = 12.5 µs we must have a time delay of 6.25 µs. Because XTAL = 8 MHz, the counter counts up every 0.125 µs. This means that we need 6.25 µs / 0.125 µs = 50 clocks. 256 – 50 = 206 = 0xCE. Therefore, we have TCNT0 = 0xCE.

```
.INCLUDE "M32DEF.INC"
      INITSTACK          ;add its definition from Example 9-3
      LDI   R16,0x08
      SBI   DDRB,3       ;PB3 as an output
      LDI   R17,0
      OUT   PORTB,R17
BEGIN:RCALL DELAY
      EOR   R17,R16      ;toggle D3 of R17
      OUT   PORTB,R17    ;toggle PB3
      RJMP BEGIN
;---------------- Timer0 Delay
DELAY:LDI   R20,0xCE
      OUT   TCNT0,R20    ;load Timer0
      LDI   R20,0x01
      OUT   TCCR0,R20    ;Timer0, Normal mode, int clk, no prescaler
AGAIN:IN    R20,TIFR     ;read TIFR
      SBRS  R20,TOV0     ;if TOV0 is set skip next instruction
      RJMP  AGAIN
      LDI   R20,0x00
      OUT   TCCR0,R20    ;stop Timer0
      LDI   R20,(1<<TOV0)
      OUT   TIFR,R20     ;clear TOV0 flag
      RET
```

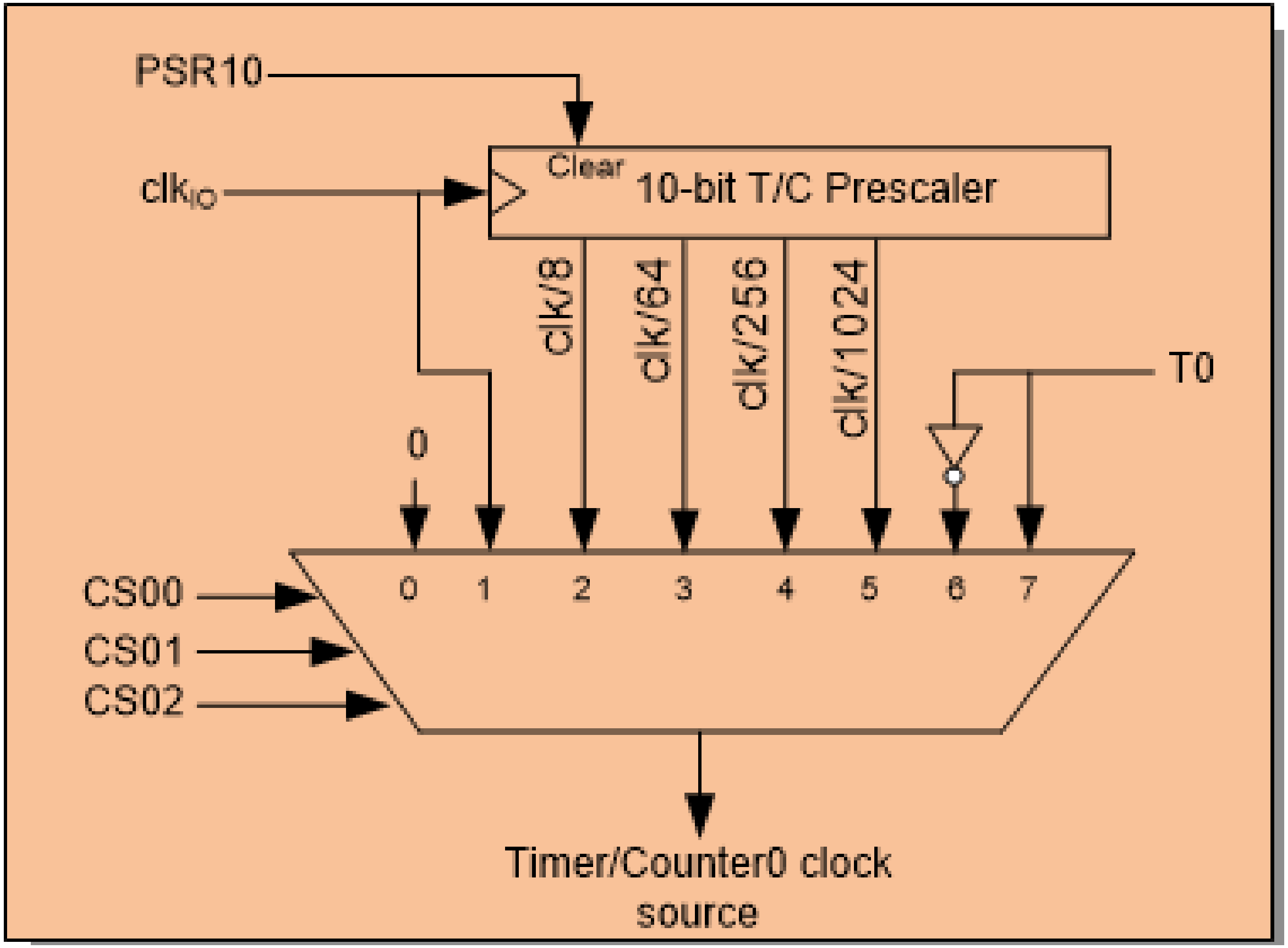
Prescalar and Generating a Large Time Delay

- Time delay depends on:
  - Crystal Frequency
  - Timer’s 8-bit register

- Both are fixed

- How to generate large time delay?
  - Use prescalar to increase the delay by reducing the clock time period

Prescalar and Generating a Large Time Delay

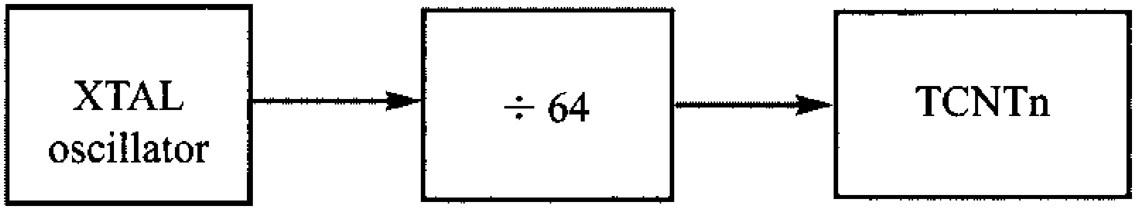


Example

Find the timer's clock frequency and its period for various AVR-based systems, with the following crystal frequencies. Assume that a prescaler of 1:64 is used.

(a) `8 MHz` (b) `16 MHz` (c) `10 MHz`

Solution:



(a) $1/64 \times 8$ MHz = 125 kHz due to 1:64 prescaler and T = 1/125 kHz = 8 μs
(b) $1/64 \times 16$ MHz = 250 kHz due to prescaler and T = 1/250 kHz = 4 μs
(c) $1/64 \times 10$ MHz = 156.2 kHz due to prescaler and T = 1/156 kHz = 6.4 μs

Example

Find the value for TCCR0 if we want to program Timer0 in Normal mode with a prescaler of 64 using internal clock for the clock source.

Solution:

From Figure 9-5 we have TCCR0 = 0000 0011; XTAL clock source, prescaler of 64.

TCCR0 =

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
0	0	0	0	0	0	1	1

Recommended Reading

- The AVR Microcontroller and Embedded Systems: Using Assembly and C by Mazidi et al., Prentice Hall
  - Chapter-9: 9.1

- Make sure you attempt and understand all the examples in the book.

THANK YOU