ATmega103 Timer0 and Interrupts

Costas Foudas, Imperial College,
Rm: 508, x47590

The Need for Processor Interrupts

Up to now if you wanted to do something in a program as soon as a bit was set (key pressed, bit set in a register, voltage exceeded a given threshold,...) you had to keep reading the bit till it changed !

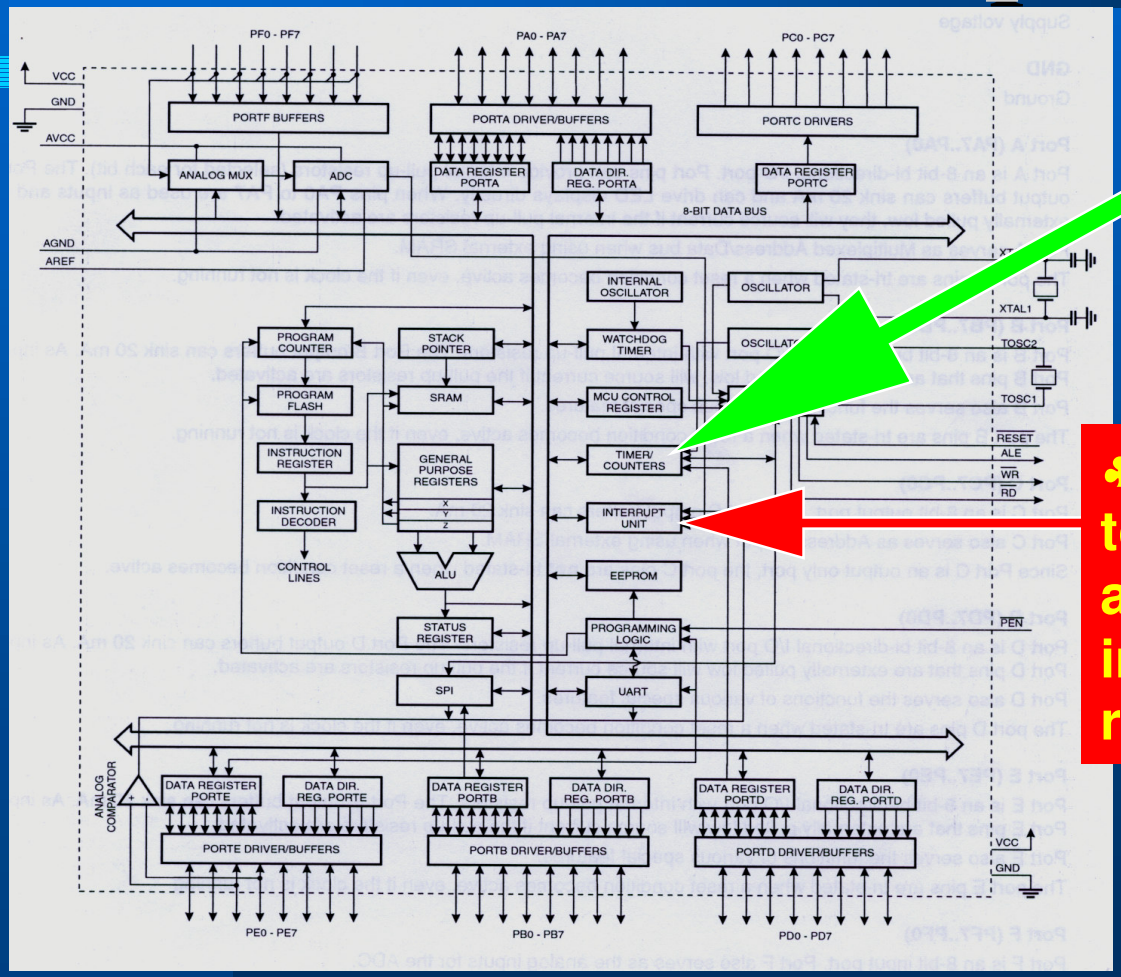
This is clearly not an efficient way of doing things.....

This is why the interrupts were introduced and are associated with bits that users tend to use frequently

Processor Interrupts

Interrupts are subroutine calls initiated not by an *rcall* command but by hardware signals. These hardware calls direct your program to the interrupt service routine which is executed. At the end your program returns where it was when the interrupt occurred.

The ATmega103 Timers and Interrupts



♣ A device that counts time (Timer) could be one of the reasons for interrupt (internal)

♣24 different reasons to interrupt the CPU and execute an interrupt service routine.

Seven External Interrupt Sources

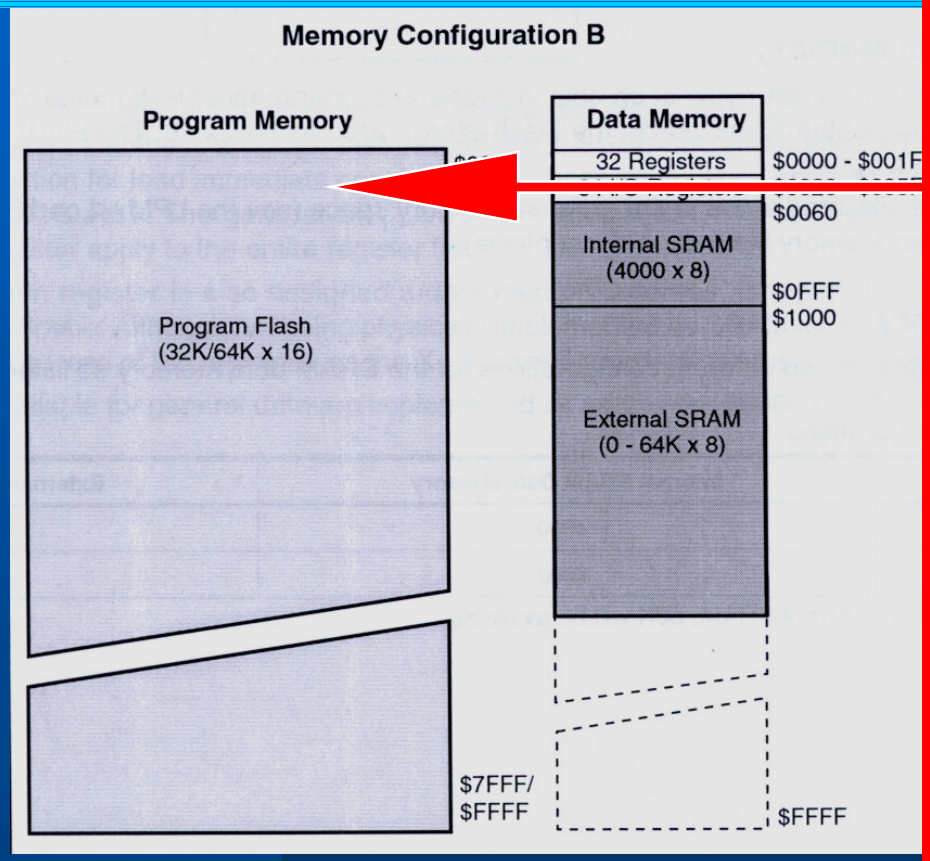
Status Register Interrupt Bit:

D7	D6	D5	D4	D3	D2	D1	D0
I	T	H	S	V	N	Z	C
★		SREG		?	?	?	?

By now you should know what The V,N,Z,C bits are.

In order to use <u>any</u> interrupts on ATmega103 you must set the ★ bit using the command *sei*

The ATmega103 Memory Map



The first 24 2 byte addresses in the flash program memory are dedicated to the interrupts : That is where does your program jump if the cause of a given interrupt occurs.

Typical Program with Interrupts (1)

```
;Address;Labels;Code; Comments
$0000 jmp RESET          ; Reset Handler
$0002 jmp EXT_INT0       ; IRQ0 Handler
$0004 jmp EXT_INT1       ; IRQ1 Handler
$0006 jmp EXT_INT2       ; IRQ2 Handler
$0008 jmp EXT_INT3       ; IRQ3 Handler
$000A jmp EXT_INT4       ; IRQ4 Handler
$000C jmp EXT_INT5       ; IRQ5 Handler
$000E jmp EXT_INT6       ; IRQ6 Handler
$0010 jmp EXT_INT7       ; IRQ7 Handler
$0012 jmp TIM2_COMP      ; Timer2 Compare Handler
$0014 jmp TIM2_OVF       ; Timer2 Overflow Handler
$0016 jmp TIM1_CAPT      ; Timer1 Capture Handler
```

Typical Program with Interrupts (2)

```
$0018 jmp TIM1_COMPA    ; Timer1 CompareA Handler
$001A jmp TIM1_COMPB    ; Timer1 CompareB Handler
$001C jmp TIM1_OVF      ; Timer1 Overflow Handler
$001E jmp TIM0_COMP     ; Timer0 Compare Handler
$0020 jmp TIM0_OVF      ; Timer0 Overflow Handler
$0022 jmp SPI_STC       ; SPI Transfer Complete Handler
$0024 jmp UART_RXC      ; UART RX Complete Handler
$0026 jmp UART_DRE      ; UDR Empty Handler
$0028 jmp UART_TXC      ; UART TX Complete Handler
$002A jmp ADC           ; ADC Conversion Complete Handler
$002C jmp EE_RDY        ; EEPROM Ready Handler
$002E jmp ANA_COMP      ; Analog Comparator Handler
```

Typical Program with Interrupts (3)

```
$0030 MAIN: ldi r16, high(RAMEND); Main
program start

$0031 out SPH,r16

$0032 ldi r16, low(RAMEND)

$0033 out SPL,r16

$0034 <instr> xxx

... ... ... ...
```

The Timer/Counter Mask Register TIMSK

D7	D6	D5	D4	D3	D2	D1	D0
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

OCIE2: Timer/Counter2 Output Compare Interrupt Enable
TOIE2:Timer/Counter2 Overflow Interrupt Enable
TICIE1: Timer/Counter1 Input Capture Interrupt Enable
OCIEA1, OCIEA2:Timer/Counter1 Output CompareA,B Match Interrupt Enable
TOIE1: Timer/Counter1 Overflow Interrupt Enable
OCIE0: Timer/Counter0 Output Compare Interrupt Enable
TOIE0: Timer/Counter0 Overflow Interrupt Enable

Timer/Counter0 Control Register

TCCR0:

D7	D6	D5	D4	D3	D2	D1	D0
Res.	PWM0	COM01	COM00	CTC0	*CS02*	*CS01*	*CS00*

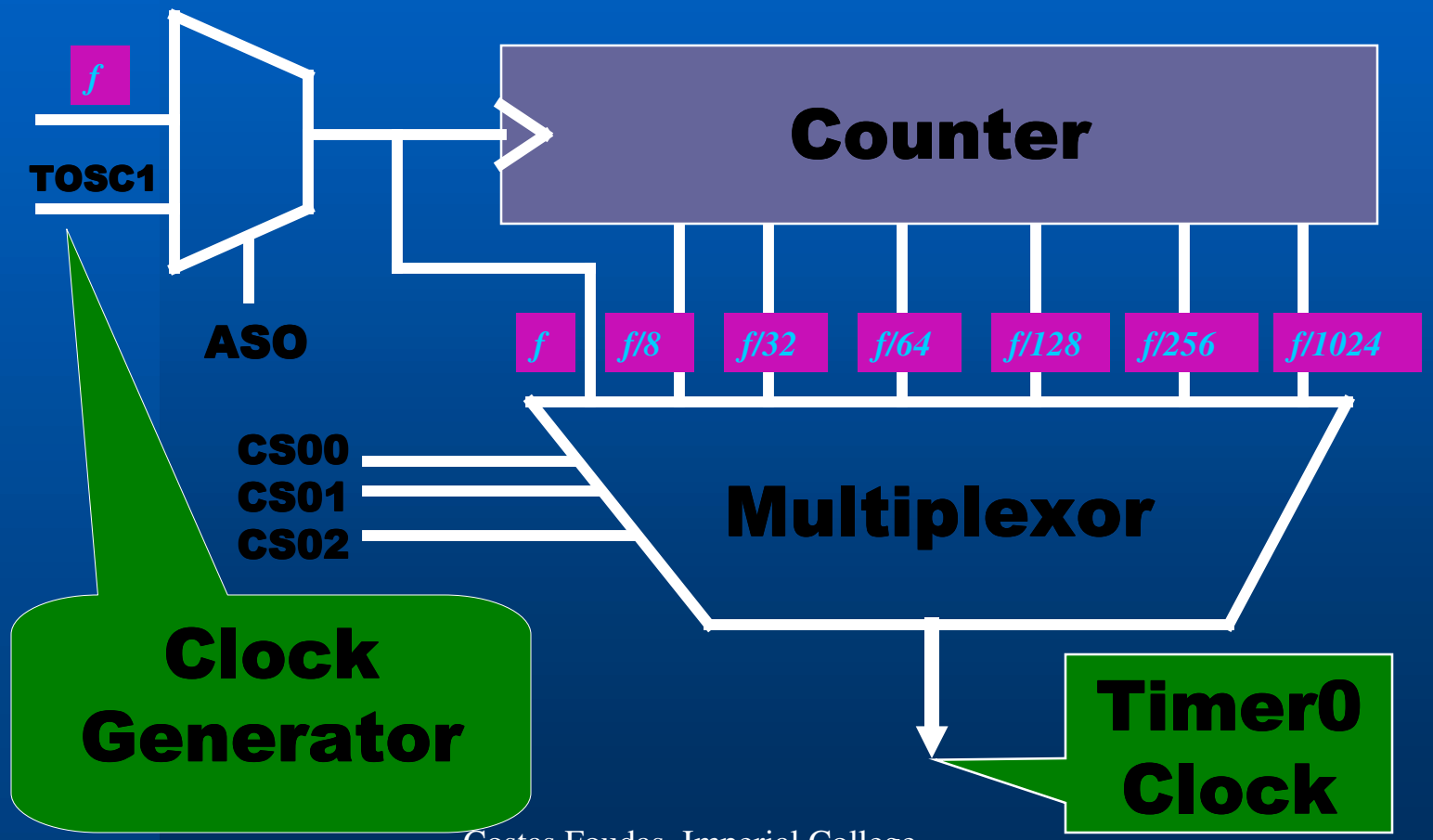
The timer pre-scale factor : CS02; CS01; CS00

CTC0: Clear Timer/Counter on Compare Match

COM00 / COMM01:Compare Output Mode, Bits 1 and 0

PWM0: Pulse Width Modulator Enable

Pre-scaling the Timer via TCCR0



Pre-scaling the Timer via TCCR0

CS02	CS01	CS00	Frequency (PCK0 = 4 MHz)
0	0	0	Timer/Counter0 is stopped
0	0	1	PCK0
0	1	0	PCK0/8
0	1	1	PCK0/32
1	0	0	PCK0/64
1	0	1	PCK0/128
1	0	0	PCK0/256
1	1	1	PCK0/1024

Timer/Counter0 TCNT0

D7	D6	D5	D4	D3	D2	D1	D0

This is the counter preset

…….These 8-bit registers contain the value of the Timer/Counters.
Both Timer/Counters are realized as up or up/down (in PWM mode) counters with read and write access. If the Timer/Counter is written to and a clock source is selected, it continues counting in the timer clock cycle after it is preset with the written value……..

Example Program

In the example program the 4 Mhz clock of the ATmega103 is pre-scaled by 256 and the timer zero is loaded with $B2. So the counter overflows ($00) every 250 x 256 x ($FF-$B2 + 1) nsec (approx every 5 msec) and causes an interrupt. Every 200 Interrupts a counter is incre-mented and the results are displayed on the PORTB LEDs.

Interrupt Section

```
jmp Init   ; jmp is 2 word instruction to set correct vector
nop
reti
.
.
.
jmp  TIM0_OVF   ; Timer 0 Overflow interrupt  Vector
nop             ; Vector Addresses are 2 words apart
reti
.
.
.
;*****************************************************************
```

At Program address 0 your program jumps to the initialization

Here it comes under hardware control when Timer0 overflows

Interrupt initialization

```
;
; **** Timer0 Setup Code ****
;
        ldi  r16,$06          ; Timer 0 Setup
        out TCCR0, r16        ; Timer - PRESCALE TCK0 BY 256
                              ; (devide the 4 Mhz clock by 256)
        ldi r16,$b2           ;  load timer with n=178
        out TCNT0,r16         ;  The counter will go off
                              ;   in (256 - n)*256*250 nsec
;
; **** Interrupts Setup Code ****
;
        ldi r16, $01          ; Timer Interrupt Enables
        out TIMSK, r16        ; T0: Overflow
;
```

Interrupt Service Routine

```
;*****************************************
TIM0_OVF:
        in R4,SREG        ;save SREG
        inc r17           ; increment cycle
        nop
        cpi r17,$C8       ;compare cycle with 200
        brne again        ;if <> jump to again
        out PORTB, r18    ; send stuff to PORTB
        inc r18           ; Increment the portB number
        clr r17           ;clear cycle and start counting
                          ; again 200 interrupts
again:
        ldi r16,$B2       ;reload timer value
        out TCNT0,r16
        out SREG,r4       ;restore sreg
        reti              ;*****************
```

When an interrupt occurs : D7 of SREG is set to '0' and the program jumps to the interrupt service routine

It is a good idea to save the information stored on the status register and restore it at the end of the program

'reti' sets the interrupt bit, in the SREG, back to '1' so the next interrupt can be serviced

Main Program

```
Main:                ;  waisting time
;
            nop
            nop
            nop
            nop
;
           rjmp main
```

But the timer is counting in the background and when it overflows it causes an interrupt forcing the program to execute the interrupt service routine. After that it comes back in the loop where it was when the interrupt occurred.....see STUDIO demo.