Programación de Sistemas Embebidos

Clase 7 – 2020

Timers e Interrupciones

Programa analítico de la asignatura

UNIDAD 3: **E/S de bajo nivel:**
Timers: relojes, contadores, PWM, interrupciones

UNIDAD 4: **Programación de bajo nivel:**
Lenguaje C. Programación sobre hardware sin sistema (baremetal).

Temario

- AVR: Interrupciones
- Modelo de las Interrupciones
- Programación de las rutinas de atención de interrupciones con avr-gcc y avr-libc
- Timers/Contadores
- Timer0: Configuraciones posibles (modos)
- Aplicaciones (real-time clock, pwm, contadores, eventos de tiempo-real)
- Ejemplo de driver

AVR: Interrupciones

- Libera al programa de tener que esperar (polling) por eventos
- Permite al programa responder a eventos cuando ocurren

- Eventos externos (ejemplos):
  - Señal que cambia en un pin. La acción depende del contexto
  - # pulsos que llegaron a un pin

- Eventos internos (ejemplos):
  - Falla en la energía
  - Excepción aritmética
  - Timer "tick"

VectorNo.	Program Address[2]	Source	Interrupt Definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B

AVR: Modelo de las Interrupciones

Cuando ocurre una interrupción:

- El procesador llama automáticamente a la RUTINA DE ATENCIÓN de INTERRUPCIONES
  - Cada evento tiene su propia dirección de interrupción
  - El bit enable global (en SREG) es automáticamente desactivad

    Push current PC, Jump to interrupt address

    por lo que no están permitidas las interrupciones anidadas
  - El SREG bit puede ser activado para habilitarlas

VectorNo.	Program Address(2)	Source	Interrupt Definition
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

- Interrupt procedure, aka “interrupt handler”
  - Realiza lo que sea necesario, entonces, termina con RETI
  - El bit enable global se activa automáticamente con RETI
  - Una instrucción del programa se ejecuta siempre luego de RETI

AVR: Modelo de las Interrupciones (Inicialización de la Tabla de Vectores de Interrupción)

- Tipicamente cada vector de interrupciones contiene una instrucción de salto (JMP o RJMP) que salta a la primer instrucción de la rutina de atención de interrupciones.
- O simplemente RETI (return from interrupt) si no se usa la interrupción.
- El reset se trata como una interrupción no enmascarable.
- Cada interrupción tiene un vector de 2 bytes, que contiene una instrucción a ser ejecutada cuando la CPU acepta la interrupción.
- Cada vector de interrupción tiene un número de vector, un entero de 1 a n (máximo número de interupciones)
- La prioridad de cada interrupción está determinada por su número de vector.
  - EL número de vector más bajo, tiene la mayor prioridad.
- Todos los vectores de interrupción, llamada Tabla de Vectores de Interrupción, se guardan en una sección contigua en la memoria flash.
  - Comienza desde 0 (por defecto).
  - Puede ser reubicada.

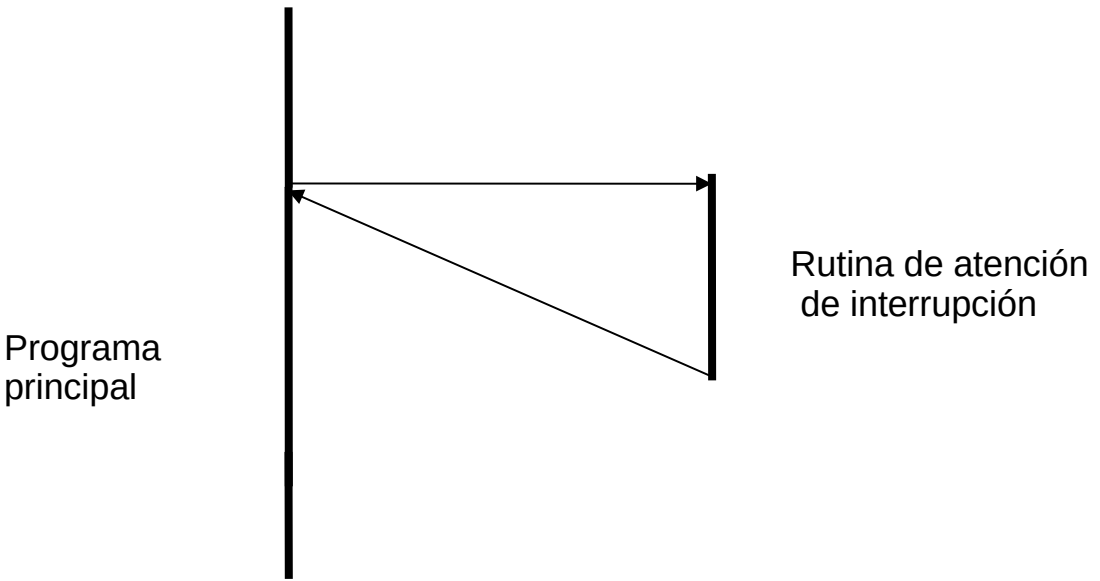
AVR: Modelo de las Interrupciones

- La rutina de atención de interrupciones es invisible al programa
  - Excepto cuando hay efectos laterales (flags o variables compartidas)
  - El tiempo de ejecución para el programa cambia
    - No se puede confiar en “dead-reckoning” para calcular un delay

- La rutina de atención de interrupciones debe ser desarrollada como si fuese invisible
  - No debe modificar el estado del programa
  - Se debe salvaguardar el contexto del programa (save y restore registros usados en la rutina)
    - Incluyendo SREG

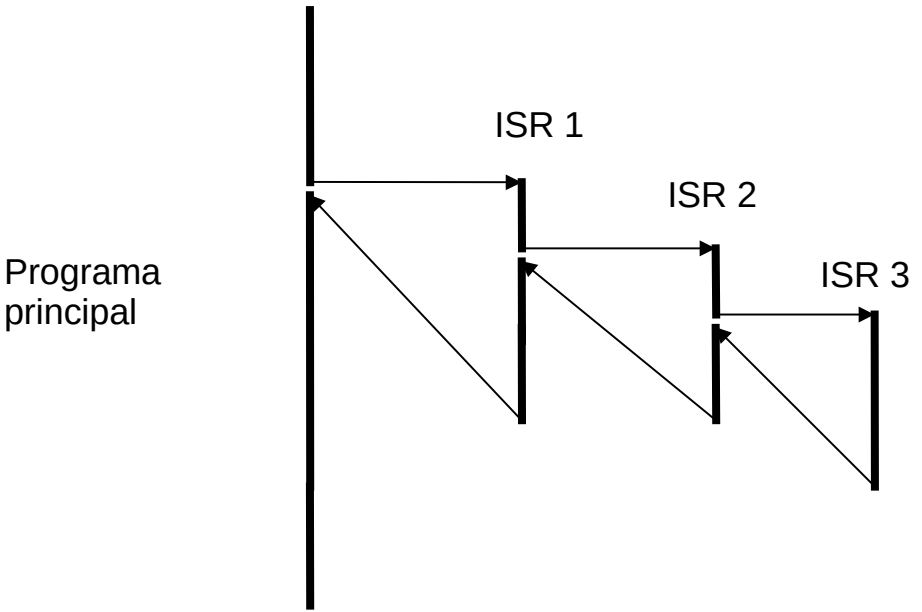
Interrrupciones no anidadas

- **Las rutinas de atención de interrupción no pueden ser interrumpidas por otras interrupciones.**



Interrrupciones anidadas

- **Las rutinas de atención de interrupción pueden ser interrumpidas por otras interrupciones.**



Cuando utilizar Interrrupciones

- Cuando el procesamiento de la aplicación no puede realizar “en tiempo” ENTRADA o SALIDA.
- Ejemplo:
  - El sistema obtiene 1 dato (ENTRADA) cada un milisegundo
  - La aplicación debe procesar 10 datos de entrada y emitir una salida
  - El procesamiento de la información de 10 datos toma 5 milisegundos
- ¿Es posible realizar este trabajo con polling?
- Solución: INTERRUPCIONES

AVR: Vectores de interrupción

- La tabla de vectores contiene la primera instrucción de cada rutina de atención de interrupciones.
- Tipicamente, la tabla está en la dirección cero (puede ser reubicada)

```
Address  Labels Code                 Comments
0x0000          jmp    RESET        ; Reset Handler
0x0002          jmp    EXT_INT0     ; IRQ0 Handler
0x0004          jmp    EXT_INT1     ; IRQ1 Handler
0x0006          jmp    PCINT0       ; PCINT0 Handler
0x0008          jmp    PCINT1       ; PCINT1 Handler
0x000A          jmp    PCINT2       ; PCINT2 Handler
0x000C          jmp    WDT          ; Watchdog Timer Handler
0x000E          jmp    TIM2_COMPA   ; Timer2 Compare A Handler
0x0010          jmp    TIM2_COMPB   ; Timer2 Compare B Handler
0x0012          jmp    TIM2_OVF     ; Timer2 Overflow Handler
0x0014          jmp    TIM1_CAPT    ; Timer1 Capture Handler
0x0016          jmp    TIM1_COMPA   ; Timer1 Compare A Handler
0x0018          jmp    TIM1_COMPB   ; Timer1 Compare B Handler
0x001A          jmp    TIM1_OVF     ; Timer1 Overflow Handler
0x001C          jmp    TIM0_COMPA   ; Timer0 Compare A Handler
0x001E          jmp    TIM0_COMPB   ; Timer0 Compare B Handler
```

AVR: Vectores de interrupción

- Si las interrupciones no se utilizan, entonces esta parte de la memoria puede ser utilizada como parte del programa
  - Por lo que esta sección de memoria no tiene nada de especial
- Ejemplo de rutina de interrupción
  - RESET: Sets up the stack pointer

```
0x0033RESET:   ldi    r16, high(RAMEND); Main program start
0x0034         out    SPH,r16        ; Set Stack Pointer to top of RAM
0x0035         ldi    r16, low(RAMEND)
0x0036         out    SPL,r16
0x0037         sei                   ; Enable interrupts
0x0038         <instr>  xxx
```

AVR: Interrupciones definidas en avr-gcc / avr-libc

```
#define INT0_vect          _VECTOR(1)   /* External Interrupt Request 0 */
#define INT1_vect          _VECTOR(2)   /* External Interrupt Request 1 */
#define PCINT0_vect        _VECTOR(3)   /* Pin Change Interrupt Request 0 */
#define PCINT1_vect        _VECTOR(4)   /* Pin Change Interrupt Request 0 */
#define PCINT2_vect        _VECTOR(5)   /* Pin Change Interrupt Request 1 */
#define WDT_vect           _VECTOR(6)   /* Watchdog Time-out Interrupt */
#define TIMER2_COMPA_vect  _VECTOR(7)   /* Timer/Counter2 Compare Match A */
#define TIMER2_COMPB_vect  _VECTOR(8)   /* Timer/Counter2 Compare Match A */
#define TIMER2_OVF_vect    _VECTOR(9)   /* Timer/Counter2 Overflow */
#define TIMER1_CAPT_vect   _VECTOR(10)  /* Timer/Counter1 Capture Event */
#define TIMER1_COMPA_vect  _VECTOR(11)  /* Timer/Counter1 Compare Match A */
#define TIMER1_COMPB_vect  _VECTOR(12)  /* Timer/Counter1 Compare Match B */
#define TIMER1_OVF_vect    _VECTOR(13)  /* Timer/Counter1 Overflow */
#define TIMER0_COMPA_vect  _VECTOR(14)  /* TimerCounter0 Compare Match A */
#define TIMER0_COMPB_vect  _VECTOR(15)  /* TimerCounter0 Compare Match B */
#define TIMER0_OVF_vect    _VECTOR(16)  /* Timer/Couner0 Overflow */
#define SPI_STC_vect       _VECTOR(17)  /* SPI Serial Transfer Complete */
#define USART_RX_vect      _VECTOR(18)  /* USART Rx Complete */
#define USART_UDRE_vect    _VECTOR(19)  /* USART, Data Register Empty */
#define USART_TX_vect      _VECTOR(20)  /* USART Tx Complete */
#define ADC_vect           _VECTOR(21)  /* ADC Conversion Complete */
#define EE_READY_vect      _VECTOR(22)  /* EEPROM Ready */
#define ANALOG_COMP_vect   _VECTOR(23)  /* Analog Comparator */
#define TWI_vect           _VECTOR(24)  /* Two-wire Serial Interface */
#define SPM_READY_vect     _VECTOR(25)  /* Store Program Memory Read */
```

AVR: Interrupciones en avr-gcc / avr-libc

- Habilitar las interrupciones a nivel global
  - Bit en SREG
  - Permite deshabilitar todas las interrupciones con un único bit
  - sei() – activa el bit
  - cli() – desactiva el bit
- La prioridad de la interrupción es determinada por el orden en la tabla
  - Direcciones mas bajas a prioridades mas altas
- ISR(vector) define el vector de interrupción
  - Ejemplo: ISR(TIMER1_OVF_vect)
- reti() – (return) vuelve de la rutina

La buena noticia del día: la definición del vector, la modificación de la tabla de vectores, el resguardo del contexto en la ISR, y la recuperación y vuelta de la ISR, es generado automáticamente por la macro ISR()

Handling the Interrupt

- The command `ISR(TIMER1_COMPA_vect)` creates a "vector" pointing to the program memory location of the piece that is meant to service the interrupt
  - near beginning of assembly code listing:

```
2c:   0c 94 80 00     jmp     0x100   ; 0x100 <__vector_11>
```

  - vector 11 is specially defined in ATMega 328 to correspond to a comparison match to OCR1A on timer 1
  - when this particular sort of interrupt is encountered, it'll jump to program location 0x100, where:
    - various working registers are PUSHed onto the STACK
      - so the service function can use those registers for itself
    - the interrupt service functions are performed
    - the STACK contents are POPped back into registers
    - the program counter is reloaded with the pre-interruption value
- The vector approach allows use of multiple interrupts

AVR: Interrupciones externas

- Sirven para monitorear cambios en las señales de los pines
- QUE causa una interrupción puede ser configurado

- Pines:
  - INT0 e INT1 – permiten un rango de opciones
    - INT0 – PORT D [2]
    - INT1 – PORT D [3]
  - PCINT[23:0] – cualquier cambio en la señal (toggle)
    - PCINT[7:0] – PORT B [7:0]
    - PCINT[14:8] – PORT C [6:0]
    - PCINT[23:16] – PORT D [7:0]

- Los pulsos de entrada deben ser mas lentos que la frecuencia del reloj de E/S

```
#define INT0_vect            _VECTOR(1)   /* External Interrupt Request 0 */
#define INT1_vect            _VECTOR(2)   /* External Interrupt Request 1 */
#define PCINT0_vect          _VECTOR(3)   /* Pin Change Interrupt Request 0 */
#define PCINT1_vect          _VECTOR(4)   /* Pin Change Interrupt Request 0 */
#define PCINT2_vect          _VECTOR(5)   /* Pin Change Interrupt Request 1 */
#define WDT_vect             _VECTOR(6)   /* Watchdog Time-out Interrupt */
#define TIMER2_COMPA_vect    _VECTOR(7)   /* Timer/Counter2 Compare Match A */
#define TIMER2_COMPB_vect    _VECTOR(8)   /* Timer/Counter2 Compare Match A */
#define TIMER2_OVF_vect      _VECTOR(9)   /* Timer/Counter2 Overflow */
#define TIMER1_CAPT_vect     _VECTOR(10)  /* Timer/Counter1 Capture Event */
#define TIMER1_COMPA_vect    _VECTOR(11)  /* Timer/Counter1 Compare Match A */
#define TIMER1_COMPB_vect    _VECTOR(12)  /* Timer/Counter1 Compare Match B */
#define TIMER1_OVF_vect      _VECTOR(13)  /* Timer/Counter1 Overflow */
#define TIMER0_COMPA_vect    _VECTOR(14)  /* TimerCounter0 Compare Match A */
#define TIMER0_COMPB_vect    _VECTOR(15)  /* TimerCounter0 Compare Match B */
#define TIMER0_OVF_vect      _VECTOR(16)  /* Timer/Couner0 Overflow */
#define SPI_STC_vect         _VECTOR(17)  /* SPI Serial Transfer Complete */
#define USART_RX_vect        _VECTOR(18)  /* USART Rx Complete */
#define USART_UDRE_vect      _VECTOR(19)  /* USART, Data Register Empty */
#define USART_TX_vect        _VECTOR(20)  /* USART Tx Complete */
#define ADC_vect             _VECTOR(21)  /* ADC Conversion Complete */
#define EE_READY_vect        _VECTOR(22)  /* EEPROM Ready */
#define ANALOG_COMP_vect     _VECTOR(23)  /* Analog Comparator */
#define TWI_vect             _VECTOR(24)  /* Two-wire Serial Interface */
#define SPM_READY_vect       _VECTOR(25)  /* Store Program Memory Read */
```

INT0 and INT1

- External Interrupt Control Register:

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Sense Control (INT0 is the same)

Table 12-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

INT0 and INT1

- External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - If INT# bit is set (and the SREG I-bit is set), then interrupts are enabled on pin INT#

- External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	–	–	–	–	–	–	INTF1	INTF0	EIFR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - Interrupt flag bit is set when a change triggers an interrupt request
  - Flag is cleared automatically when interrupt routine is executed
  - Flag can be cleared by writing a 1 to it

PCINT[23:0]

- Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - PCIE2 enables interrupts for PCINT[23:16]
  - PCIE1 enables interrupts for PCINT[14:8]
  - PCIE0 enables interrupts for PCINT[7:0]
- Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	–	–	–	–	–	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - PCIF# set if corresponding pins generate an interrupt request
  - Cleared automatically when interrupt routine is executed

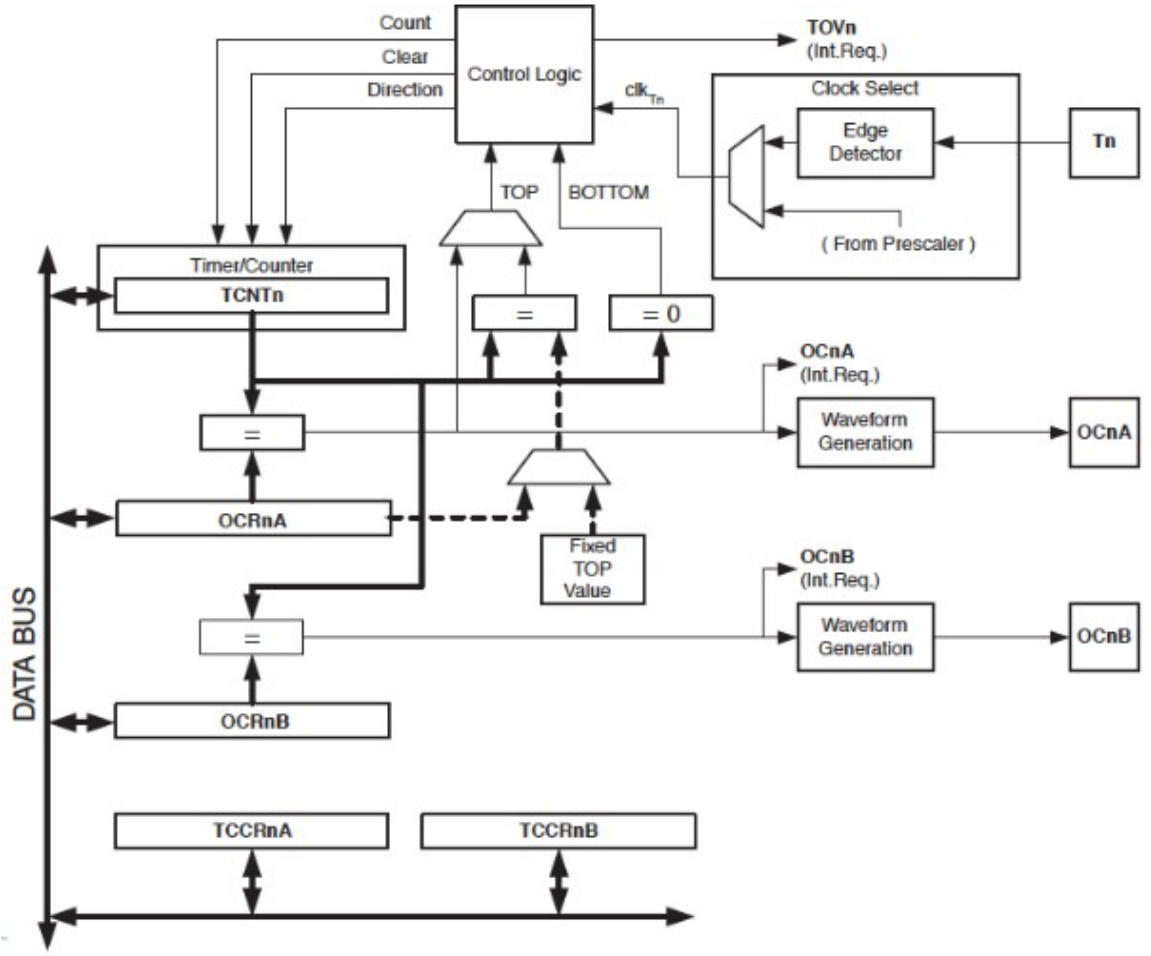
PCINT[23:0]

- Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
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	

- Each bit controls whether interrupts are enabled for the corresponding pin
- Change on any enabled pin causes an interrupt
- (Mask registers 1 and 0 are similar)

AVR: Timer/Counter 0 (8-bit)

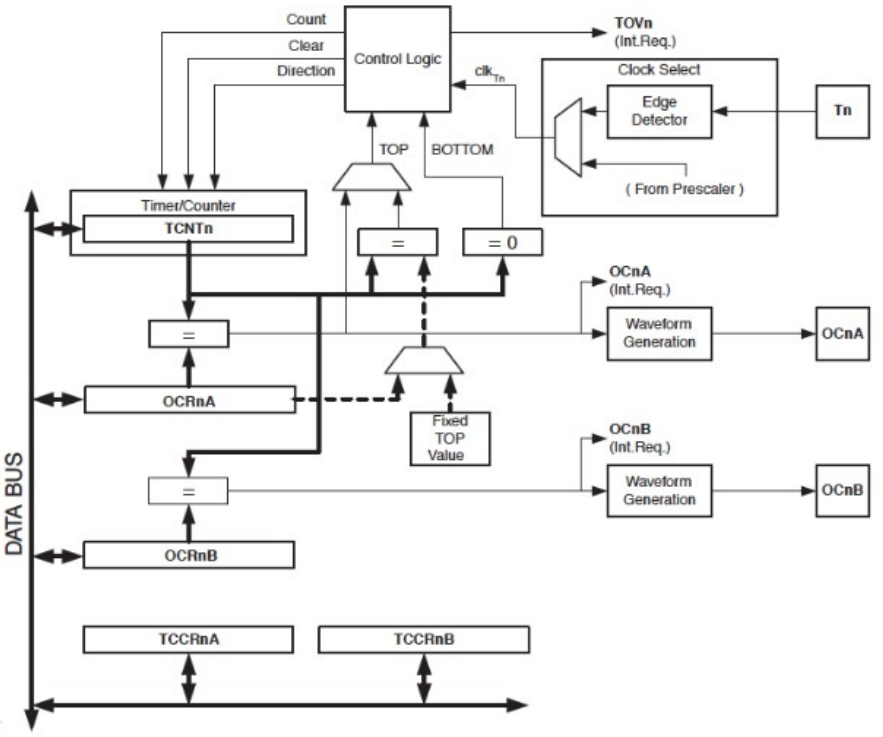


AVR: Timer/Counter 0 (8-bit) – seleccionar el origen del reloj

- El timer0 está "apagado" si CS02:CS01:CS00 = 0:0:0

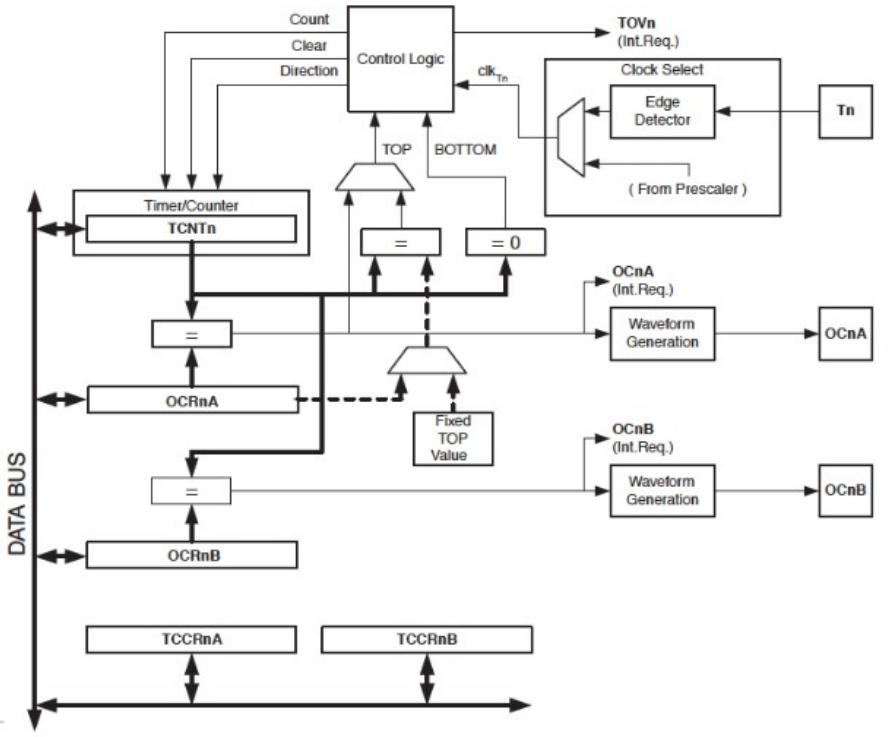
CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

- T0 pin – PORT D[4]
- T1 pin – PORT D[5]
- Pin can be configured as output pin
  - Program can generate clock



AVR: Timer/Counter 0 (8-bit) – REGISTROS

- TCNT0 – Timer/Counter Register (8-bit)
- OCR0A – Output Compare Register A
- OCR0B – Output Compare Register B
- TCCR0A/B – Timer/Counter Control Registers
- TIMSK0 – Timer/Counter Interrupt Mask Register
  - TOV interrupt
  - Compare A&B interrupts
- TIFR0 – Timer/Counter Interrupt Flag Register
  - TOV interrupt
  - Compare A&B interrupts



AVR: Timer/Counter 0 (8-bit) – Usos del timer

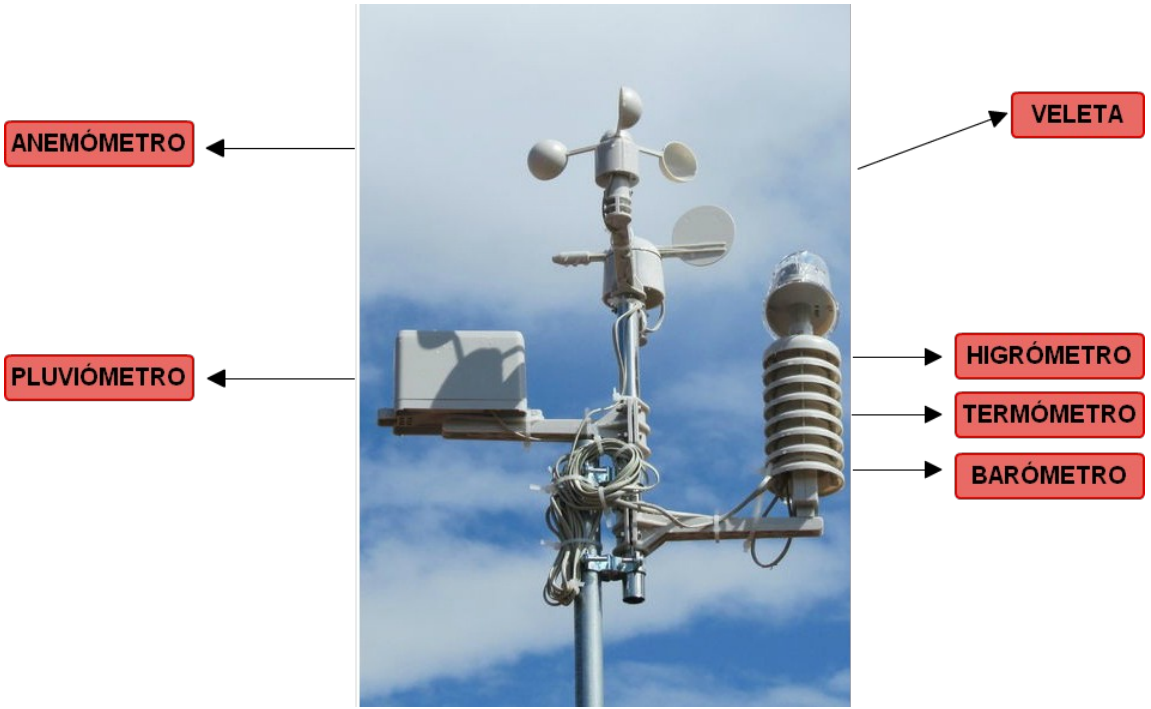
- "Contar" eventos externos (y generar, si se desea, una interrupción en cierta cantidad)
- Real-time clock
- "Medir" tiempo transcurrido con precisión (por ej.: desde que se inicia el timer hasta que se detiene, por software)
- Para realizar tareas en momentos precisos (diferentes tareas en diferentes momentos)
- Para generar una interrupción luego de un período
- Para generar señales PWM (~ a una salida analógica si se coloca un filtro)

Timed Interrupts

- Really handy to have timed action, despite whatever `loop()` is doing
  - could check for serial or other input on a regular basis
  - could read analog signal for regular sampling
  - could produce custom signal at specific frequency
- Idea is to set up timer so when it reaches specified count, it creates an interrupt
  - and also resets counter to zero so cycle begins anew
- Interrupt Service Routine (ISR) should be short and sweet
  - performs whatever periodic task you want

AVR: Timer/Counter 0 (8-bit) – Usos del timer

- "Contar" eventos externos (y generar, si se desea, una interrupción en cierta cantidad)



AVR: Timer/Counter 0 (8-bit) – Usos del timer

- Para generar señales PWM : control de servos, control de velocidad , emular una señal analógica

AVR: Timer/Counter 0 (8-bit) – Usos del timer

- Para generar una interrupción luego de un período: preemptive scheduler RTOS, tareas en momentos precisos, contar momentos exactos

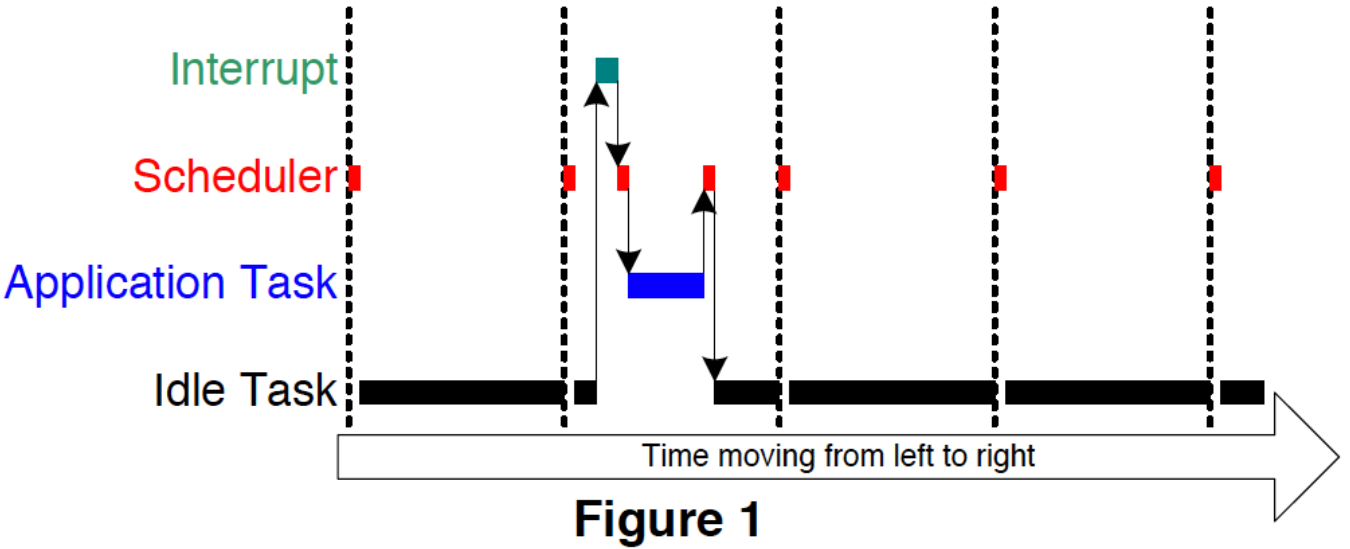


Figure 1

AVR: Timer/Counter 0 (8-bit) – Modo normal

- max frequency of each is 16 MHz, (as assembled)
- TIMER0 is an 8-bit timer, with 1, 8, 64, 256, 1024 prescaler options
- TIMER1 is a 16-bit timer, with 1, 8, 64, 256, 1024 prescaler options
- TIMER2 is an 8-bit timer with 1, 8, 32, 64, 128, 256, 1024 prescaler options

Modo NORMAL:

- El timer/contador se incrementa
- Cuenta hasta TOP = 0xFF
- Luego empieza en cero nuevamente
- Se activa la interrupción (TOV0 interrupt flag) cuando el contadorTCNT0 se resetea a 0
- Útil para generar interrupciones cada N unidades de tiempo

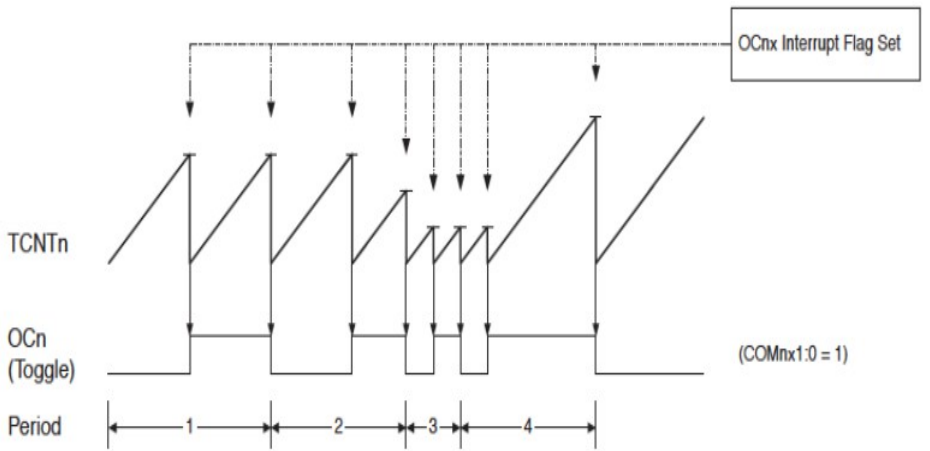
Wrap Times

- TIMER0 is 8-bit (0–255)
  - when prescaler = 1, reaches full count in 16 µs
  - when prescaler = 1024, full count in 16.384 ms
- TIMER1 is 16-bit (0–65536)
  - when prescaler = 1, reaches full count in 4.096 ms
  - when prescaler = 1024, full count in 4.194 seconds
- TIMER2 is 8-bit (0–255)
  - when prescaler = 1, reaches full count in 16 µs
  - when prescaler = 1024, full count in 16.384 ms
- These wrap times set limits on timed interrupts
  - makes TIMER1 attractive, for its 16 bits

AVR: Timer/Counter 0 (8-bit) – Modo 2 CTC

Modo 2 CTC (Clear Timer on Compare):

- **El timer/contador se incrementa**
- **Hasta llegar a OCR0A**
- **OCR0A define el valor máximo del contador y luego reinicia a cero 0**



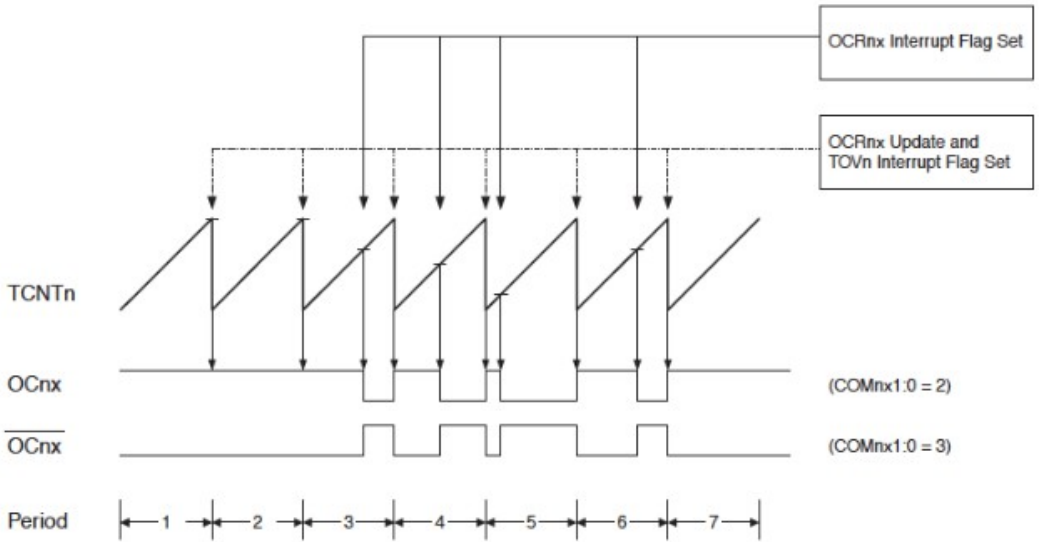
- **Se activa una interrupción (OCF0A interrupt flag) cuando el contador TCNT0 se resetea a 0**
- **El pin OC0A puede cambiar (toggle) cuando el contador se reseta**

(Generate output waveform)

AVR: Timer/Counter 0 (8-bit) – Modo Fast PWM

Modo Fast PWM:

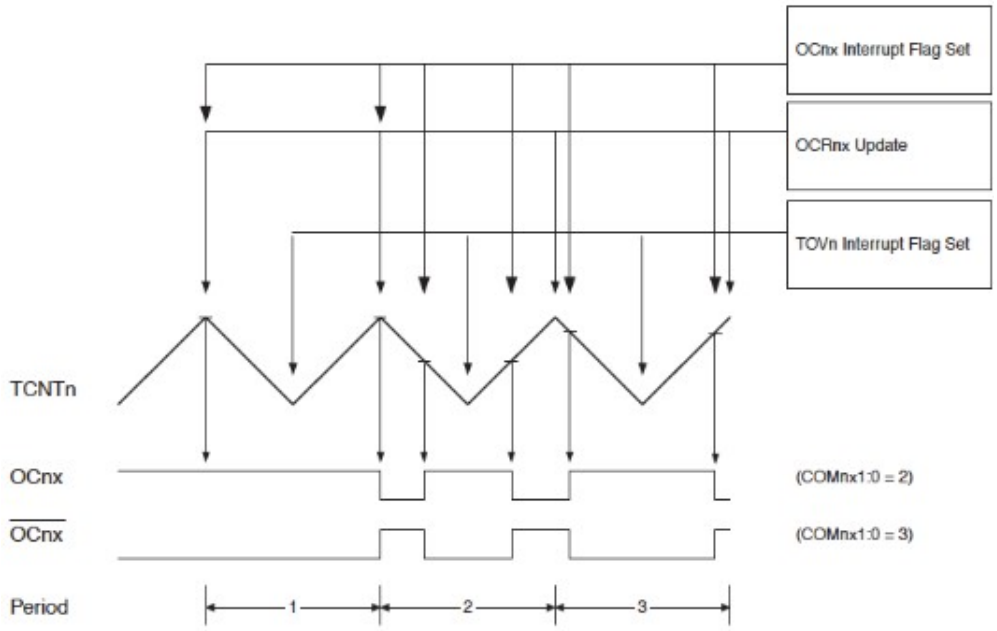
- El timer/contador se incrementa
- Hasta llegar a 0xFF (3) o OCR0A (7)
- Luego comienza en cero nuevamente
- Pin OC0x puede generar una onda
  - Set (o reset) cuando timer=0
  - Reset (o set) cuando timer=OCR0x



AVR: Timer/Counter 0 (8-bit) – Modo PWM fase-corregida

Modo PWM fase-corregida:

- El timer/contador se incrementa, luego decrementa
- Se incrementa desde cero
- Cuando llega a 0xFF (1) o OCR0A (5)
- comienza a decrementar para llegar a cero
- Pin OC0x puede generar una onda
  - Set cuando timer=OCR0x mientras se incrementa
  - Reset cuando timer=OCR0x mientras decrementa



AVR: Timer/Counter 0 (8-bit) – REGISTROS

–

Timer/Counter Control Register A

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

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	

AVR: Timer/Counter 0 (8-bit) – REGISTROS

- Seleccionar el modo con los bits WGM02, WGM01 y WGM00

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[1][2]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

AVR: Timer/Counter 0 (8-bit) – REGISTROS

Table 14-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

Table 14-3. Compare Output Mode, Fast PWM Mode(1)

0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected. WGM02 = 1: Toggle OC0A on Compare Match.
1	0	Clear OC0A on Compare Match, set OC0A at BOTTOM, (non-inverting mode).
1	1	Set OC0A on Compare Match, clear OC0A at BOTTOM, (inverting mode).

Table 14-4. Compare Output Mode, Phase Correct PWM Mode(1)

0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected. WGM02 = 1: Toggle OC0A on Compare Match.
1	0	Clear OC0A on Compare Match when up-counting. Set OC0A on Compare Match when down-counting.
1	1	Set OC0A on Compare Match when up-counting. Clear OC0A on Compare Match when down-counting.

AVR: Timer/Counter 0 (8-bit) – REGISTROS

_El registro

Timer/Counter 0 Interrupt Mask

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	

- TOIE0 – Timer Overflow interrupt
- OCIE0A/B – Compare A/B interrupt

Timer/Counter 1 Interrupt Flags

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	

- TOV0 – Timer Overflow flag
- OCF0A/B – Compare A/B interrupt flag

AVR: Timer/Counter 0 (8-bit) – REGISTROS

- El registro timsk0 (Interrupt Mask) permite indicar QUE interrupcciones se desea generar (por ejemplo, cuando el contador es igual al comparador OCR0A)

▸ Timer/Counter 0 Interrupt Mask

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	

- TOIE0 – Timer Overflow interrupt
- OCIE0A/B – Compare A/B interrupt

▸ Timer/Counter 1 Interrupt Flags

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	

- TOV0 – Timer Overflow flag
- OCF0A/B – Compare A/B interrupt flag

AVR: Timer/Counter 0 (8-bit) – Ejemplo

-

```c
/* Genera 1000 interrupciones por segundo, utilizando el Timer0 de 8bits.
 *
 * 1 segundo 1000 milisegundos
 * 1 milisegundo 1000 microsegundos
 *
 * Utilizamos modo CTC, el cual compara con TOP. TOP es definido por OCR0A.
 * Cuando CONTADOR == TOP el CONTADOR se pone a cero.
 *
 * reloj del sistema: 16000000 de ticks/s  /  64 (prescalar) = 250000 ticks/s
 *
 * 250000 ticks/s  /  250 = 1000 ticks/s =>
 */

#include <avr/io.h>
#include <avr/interrupt.h>

volatile int ticks = 0;

typedef struct
{
    uint8_t tccr0a; /* Registro de control A del timer0 */
    uint8_t tccr0b; /* Registro de control B del timer0 */
    uint8_t tcnt0;  /* Registro timer/contador */
    uint8_t ocr0a;  /* Registro comparador A */
    uint8_t ocr0b;  /* Registro comparador B */

} volatile timer0_t;

volatile timer0_t *timer0 = (timer0_t *)(0x44);

/* registro mascara de interrupciones timer 0 */
volatile unsigned char *timer0_timsk0 = (unsigned char *)(0x6E);

#define TIMER0_CS 0x03 /* prescalar=64 CS02|CS01|CS00 = 0b011 */
#define TIMER0_CTC 0x02 /* CTC = WGM = 0b010 : modo CTC es comparar y volver a cero*/

#define SYSTEM_TICKS 16000000
#define PRESCALAR 64
#define TICKS_PER_SECOND 1000

#define TIMER0_OCR0A (SYSTEM_TICKS/PRESCALAR/TICKS_PER_SECOND)
```

AVR: Timer/Counter 0 (8-bit) – Ejemplo

```
void timer0_init( void )
{
        timer0->tccr0a |= TIMER0_CTC;
        timer0->tccr0b |= TIMER0_CS;
        timer0->ocr0a = TIMER0_OCR0A; /* valor contra el cual comparar */
        (*timer0_timsk0) |= 0x02;  /* 0x02: compara contra registro OCR0A
                                      y genera interrupcion si hay un match */
}


/* rutina de atención de interrupciones */
ISR(TIMER0_COMPA_vect)
{
    ticks ++;
    if (ticks == 1000) {
        /* pasó un segundo */
   }
}
```

AVR: Timer/Counter 0 (8-bit) – Fun is coming

_ Próximo TP habrá que _ :

- Medir cantidad máxima de llamadas adc y serial por segundo
- Desarrollar un real-time clock
- Ejecutar tareas diferentes en momentos exactos diferentes
  - ¿Se puede hacer con un bucle infinito?
- Construir una placa de sonido 8bit 11khz para la PC (aplicación de tiemp-real duro). Desarrollar driver
- Señal NTSC (extreme)

AVR: Interrupciones. Notas finales

- El manejo de interrupciones trae complejidad al diseño
- Es necesario el manejo de Buffers (más software: mas diseño, desarrollo, testing)
- Race conditions

- Puede ser útil una biblioteca de buffers circulares que pueda ser utilizada por cada driver, y de manera atómica
- Cada driver mantiene un puntero a su buffer
- Llama a rutinas de la biblioteca de buffers utilizando como argumento sus punteros de buffers (get/put).

AVR: Interrupciones. Notas finales

¿Encuentra algún problema con esta rutina de atención de interrupciones?

```
int gIndex = 0;
interrupt void serialReceiveIsr(void)
{
    /* Store receive character in memory buffer. */
    gIndex++;
}

int main(void)
{
    while (1) {
        if (gIndex) {
            /* Process receive character in memory buffer. */
            gIndex--;
        }
    }
}
```

¿Y con este código (AVR)?

```
unsigned int Data;

void main()
{
      unsigned int LocalData;
      --
      while(1)
      {
            LocalData = Data;
            --
      }
}



void I2C_ISR(void)
{
      Data = I2C_BUF[0];
      Data =
      (Data<<8)|(I2C_BUF[1]);
}
```