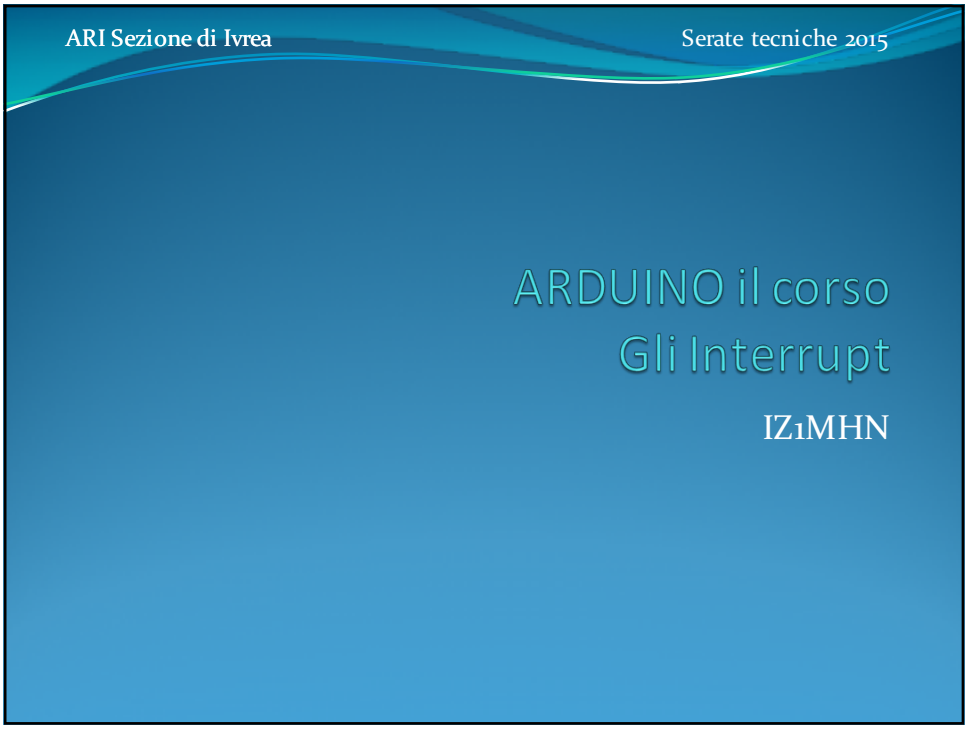
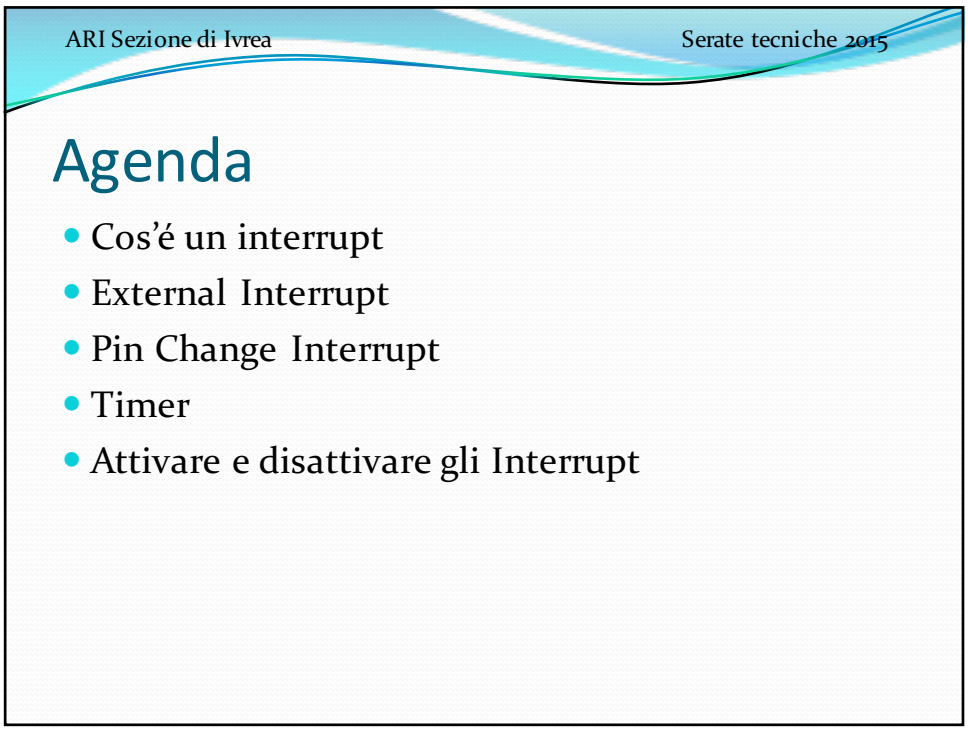
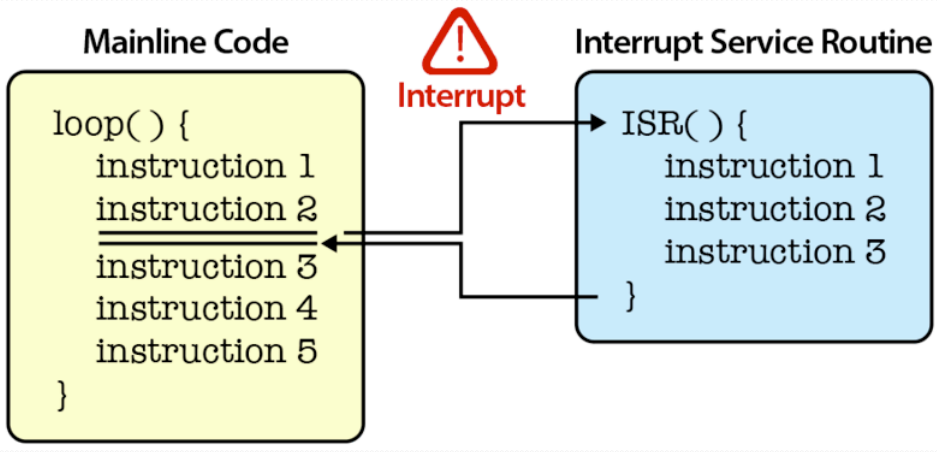

ARI Sezione di Ivrea
Serate tecniche 2015
ARDUINO il corso
Gli Interrupt
IZ1MHN


ARI Sezione di Ivrea
Serate tecniche 2015
Agenda
Cos'é un interrupt
External Interrupt
Pin Change Interrupt
Timer
Attivare e disattivare gli Interrupt

Cos'è un interrupt

- … è un'interruzione!
- È un segnale asincrono che interrompe l'esecuzione del loop per gestire un evento.
- È generato dalla variazione dello stato di un Pin (esterno o interno)

Cos'è un Interrupt



12.4 Interrupt Vectors in ATmega328 and ATmega328P

Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

VectorNo.	Program Address(1)	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
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

Notes: 1. When the BOOTRST Fuse is programmed, the device will jump to the Boot Loader address at reset, see "Boot Loader Support – Read-While-Write Self-Programming" on page 269.

External Interrupt

- Possiamo immaginarli come gli Interrupt di base!
- Sono direttamente esposti sui Pin di Arduino
- … dipendono dall'Arduino per numero e nome!

Scheda	Int 0	Int 1	Int 2	Int 3	Int 4	Int 5
UNO	2	3				
MEGA	2	3	21	20	19	18
Leonardo	3	2	0 (RX)	1 (TX)	7	
Micro	3	3	0 (RX)	1 (TX)		
DUE	Qualsiasi Pin!!!					
Nota	gli interrupt 2 e 3 sono disponibili su Leonardo e Micro solo se si disattiva la seriale					

External Interrupt

Uno			
attachInterrupt	Name	Pin on chip (PDIP)	Pin on board
0	INT0	4	D2
1	INT1	5	D3

Mega2560			
attachInterrupt	Name	Pin on chip (TQFP)	Pin on board
0	INT4	6	D2
1	INT5	7	D3
2	INT0	43	D21
3	INT1	44	D20
4	INT2	45	D19
5	INT3	46	D18

Leonardo			
attachInterrupt	Name	Pin on chip (TQFP)	Pin on board
0	INT0	18	D3
1	INT1	19	D2
2	INT2	20	D0
3	INT3	21	D1
4	INT6	1	D7

ARI Sezione di Ivrea Serate tecniche 2015

External Interrupt

- La gestione avviene con due comandi:
  - `attachInterrupt(interruptID, function, mode)`
  - `detachInterrupt(interrupt)`
- La prima "assegna" una funzione di interrupt (Interrupt Service Routine o ISR) a un Pin
- La seconda "sgancia" la funziona dal Pin

attachInterrupt

`attachInterrupt(interruptID, function, mode)`

- interruptID: specifica l'interrupt in base ai valori della tabella
- Function: il nome esatto della ISR che chiamiamo
- Mode: il cambiamento che scatena la ISR
  - LOW
  - CHANGE
  - RISING
  - FALLING

Esempio 1

```
volatile int Variabile = 10;
void setup() {
pinMode(2, INPUT);
  pinMode(13, OUTPUT);
  attachInterrupt(0, Incrementa, FALLING);
  attachInterrupt(1, Decrementa, FALLING);
  Serial.begin(9600);
}
void loop() {
  Serial.println(Variabile);
}
void Incrementa () {
  Variabile++;
}
void Decrementa () {
  Variabile--;
}
```

Esempio 2

```
int pulsante = 0;
int led = 13;
volatile int stato = LOW;

void setup(){
  pinMode(led, OUTPUT);
  attachInterrupt(pulsante, cambia, CHANGE);
}

void loop(){
  for (int i = 0; i < 100; i++)
  {
    delay(10);
  }
}

void cambia(){
  stato = !stato;
  digitalWrite(ledOut, stato);
}
```


volatile

volàtile agg. e s. m. [dal lat. *volatĭlis*, der. di *volare* «volare²» ... In senso fig., letter., labile, evanescente, effimero... In informatica, in opposizione a permanente, detto di unità di memoria il cui contenuto di informazioni è perduto in caso di mancanza di alimentazione elettrica; sono di questo tipo le memorie RAM a semiconduttore.

- In arduinese... forza il compilatore a usare memoria RAM invece che uno Storage Register, in modo da rendere "persistente" la variabile.
- Il valore dei registri, nel salto verso e dalla ISR potrebbe variare.
- Per sicurezza tutte le variabili modificate da ISR e da loop vanno dichiarate come `volatile`

Le ISR

- Non possono avere parametri!
- Non possono ritornare valori!
- Durante la loro esecuzione gli altri interrupt sono disabilitati: la `delay()` e la `millis()` … non funzionano!
- Le ISR vanno mantenute il più compatte e veloci possibile

Piccolo problema…

- Vi ricordate come avevamo risolto il debounce?
- Via SW:
  - Leggi il valore del pin (HIGH/LOW)
  - Aspetta x ms (con la delay())
  - Rifai la lettura: se i due valori coincidono… tutto ok
- Ma… non si può più usare la delay()…
- Il debounce va fatto in hw! ☹

Esempio di debounce HW

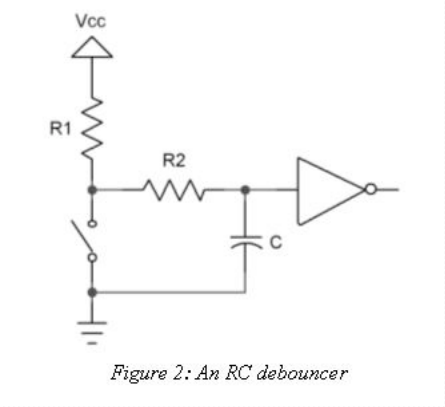


Figure 2: An RC debouncer

Pin Change Interrupt

- Se non mi bastano gli External Interrupt (e non voglio usare un Arduino Due o Zero!).
- Attenzione: dipendono pesantemente dal micro e dalla scheda!
- Per questa volta ci basiamo su Arduino UNO R3.
- Possono usare quasi tutti Pin
- Bisogna fare riferimento direttamente al Datasheet.

Pin Change Interrupt

Bisogna far riferimento direttamente al Datasheet: ATmega48A/PA/88A/PA/168A/PA/328/P 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTES IN-SYSTEM PROGRAMMABLE FLASH DATASHEET (in particolare a pag 65 tabella 12.4).

12.4 Interrupt Vectors in ATmega328 and ATmega328P

Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

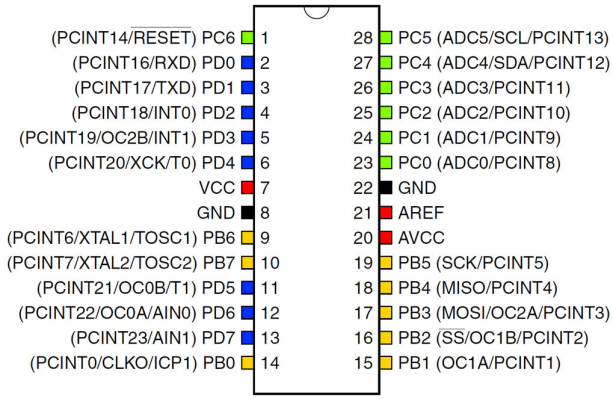
VectorNo.	Program Address(2)	Source	Interrupt Definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2

Pin Change Interrupt: Porte

- ATMEGA328 divide i pin di I/O in 3 porte: PA, PB e PC.
- Ogni porta ha associato un registro che contiene la direzione (INPUT/OUTPUT) e la condizione (HIGH o LOW).
- Ogni porta ha associato un vettore di Pin Change Interrupt (PC0, PC1 e PC2).
- Ogni PC ha associato un altro vettore per richiamare la ISR e una Mask (PCMSK0, PCMSK1 e PCMSK2)

ARI Sezione di Ivrea Serate tecniche 2015

Pin Change Interrupt: Porte



ARI Sezione di Ivrea Serate tecniche 2015

Pin Change Interrupt: Uso

- Più facile a usarsi che a spiegarsi…
- Tre passi:
  - Attivare la porta
  - Attivare i pin di interrupt
  - Scrivere una funzione ISR per ciascun interrupt

Attivare la porta

- Va impostato ad uno il valore corrispondente alla porta nel registro PCICR
- Usiamo l'operatore |= (compound bitwise OR) lavorando direttamente in binario.
  - `PCICR |= 0b00000001;  // Attiva la porta B`
  - `PCICR |= 0b00000010;  // Attiva la porta C`
  - `PCICR |= 0b00000100;  // Attiva la porta D`
- Si possono combinare:
  - `PCICR |= 0b00000111;`

PCICR

Facciamo sempre riferimento al Datasheet do ve troviamo a pag 73: "When the PCIE2 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 2 is enabled. Any change on any enabled PCINT[23:16] pin will cause an interrupt. The corresponding interrupt of Pin Change Interrupt Request is executed from the PCI2 Interrupt Vector. PCINT[23:16] pins are enabled individually by the PCMSK2 Register."

PCICR – 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	

Attivare i pin di interrupt

- Vanno usati tre registri: PCMSK0, PCMSK1 e PCMSK2
- Anche in questo caso andiamo direttamente in binario:
  - `PCMSK0 |= 0b00000001;  //abilita sulla Porta //B PCINT0 (pin 8)`
  - `PCMSK2 |= 0b10010001;  //abilita PCINT16, 19 // e23 (pin 0, 3 e 7)`

PCMSK

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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	

PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK2 – 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	

Scrivere le funzioni ISR

- Le ISR usate nei Pin Change Interrupt hanno nomi fissi. In particolare i tre nomi sono:
  - `ISR(PCINT0_vect){}` per la Porta B,
  - `ISR(PCINT1_vect){}` per la Porta C,
  - `ISR(PCINT2_vect){}` per la Porta D.
- NB: prestare molta attenzione alla sintassi dei nomi delle ISR: se si verifica un interrupt e non viene trovata la ISR corrispondente, viene eseguito il default interrupt, cioè il RESET.

Esempio 3

```
volatile int Variabile = 10;
void setup()
{
  noInterrupts();           // Disabilito gli interrupt x configurarli
  PCICR |= 0b00000011;      // Abiilito le porte B e C
  PCMSK0 |= 0b00000001;     // PCINT0 -> Arduino pin D8
  PCMSK1 |= 0b00001000;     // PCINT11 -> Arduino pin A3
  interrupts();             // Riabilito gli interrupt
  Serial.begin(9600);
}

void loop() {
  Serial.println(Variabile);
}

ISR(PCINT0_vect) {
  Variabile++;
}

ISR(PCINT1_vect) {
  Variabile--;
}
```

Piccolo problema…

- Ma… come faccio a capire su quale Pin si è effettivamente verificato l'interrupt??
- Non c'è un sistema automatico.
- Il trucco è di salvare lo stato delle tre porte in tre vettori in modo da confrontarli in l'istante t-o.

Discriminare il Pin Change

Piccolo problema… 2

- Il numero di Pin disponibili per il Pin Change… varia da Arduino ad Arduino.
- Bisogna tenerlo presente se si vuole sviluppare uno sketch portabile
- Arduino UNO: qualsiasi Pin
- Arduino Leonardo e Micro: Pin 8,9,10 e 11.
- Arduino Mega: Pin 10,11,12,13,14,15 e i Pin analogici da 6 a 15.

Timer

- Se prima vi sembrava complesso… aspettate adesso!
- Comunque… un Timer è un… timer!
- Semplicemente, invece che far suonare una sveglia… attiva un Interrupt!
- Esattamente come un External Interrupt, un Timer è un evento asincrono.

Timer: il funzionamento

- I Timer funzionano incrementando una variabile dedicata (Counter Register).
- Quando raggiunge il limite superiore e va in overflow si resetta e riparte.
- Il Counter imposta un flag per segnalare che è andato in overflow.
- È possibile controllare il flag a mano o associargli una Interrupt e ovviamente una ISR.

Timer: il funzionamento

- Per incrementare il valore a intervalli regolari, il Timer deve aver accesso a una clock source.
- La Clock Source genera un segnale consistente e ripetitivo.
- Ogni volta che il clock rileva un segnale, incrementa il contatore di 1.
- La Clock Source può essere pari al clock della CPU o può essere rallentata con un prescaler.

Quanti Timer?

- Arduino UNO espone tre Timer
  - Timer0:
    - 8 bit
    - Usato per le funzioni di tempo: delay(), millis() e micros().
    - Attenzione se lo si usa!
  - Timer1:
    - 16 bit
    - Usato dalla funzione servo()
  - Timer2:
    - 8 bit
    - Usato dalla funzione tone()
- Arduno Mega aggiunge i Timer3, Timer4 e Timer5, tutti a 16 bit
- E il DUE/ZERO? Ovviamente tutto diverso!

Configurare e attivare i Timer

- La configurazione dei Timer si basa sui registri.
- In particolare:
  - TCCRx - Timer/Counter Control Register.
  - TCNTx - Timer/Counter Register.
  - OCRx - Output Compare Register
  - ICRx - Input Capture Register (Solo nei registri a 16 bit)
  - TIMSKx - Timer/Counter Interrupt Mask Register.
    TIFRx - Timer/Counter Interrupt Flag Register

TCCRx - Timer/Counter Control Register

- È usato per impostare la modalità di funzionamento del timer, il , prescaler e altre opzioni.
- Due registri da 8 bit nel Timer1, uno solo in Timer0 e Timer2.
  - **TCCRxA** – Timer/Counter Control Register A
  - **TCCRxB** – Timer/Counter Control Register B

TCCRxA e TCCRxB

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
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	

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
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	

ARI Sezione di Ivrea Serate tecniche 2015

COM1X1:0: Compare Output Mode for Channel X

Table 16-1. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on Compare Match.
1	0	Clear OC1A/OC1B on Compare Match (Set output to low level).
1	1	Set OC1A/OC1B on Compare Match (Set output to high level).

Table 16-3. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM[1]

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 9 or 11: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match when up-counting. Set OC1A/OC1B on Compare Match when downcounting.
1	1	Set OC1A/OC1B on Compare Match when up-counting. Clear OC1A/OC1B on Compare Match when downcounting.

Table 16-2. Compare Output Mode, Fast PWM[1]

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM (non-inverting mode)
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at BOTTOM (inverting mode)

ARI Sezione di Ivrea Serate tecniche 2015

WGM11:0: Waveform Generation Mode

Table 16-4. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 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
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

PWM (Pulse width modulation) mode – gli outputs di Ocxy possono essere usati per generare segnali PWM.
CTC (Clear timer on compare match) mode – quando il Timer Counter dagiunge il registro di compare match, viene resettato.
NB: WGM12 arriva dal registro TCCR1B

CS12:0: Clock Select

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	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 T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Calcolo della frequenza del Timer

- Che poi va caricata in TCNTx
- Ciascun Timer può avere una frequenza diversa.
- Per determinare la frequenza cui impostare il timer:
  - Considerare la frequenza di Arduino -> 16 MHz
  - Considerare il valore massimo del timer -> 256 (8 bit) o 65536 (16 bit)
  - Considerare il valore del prescaler -> 8, 64, 256, 1024
  - Dividere la frequenza di Arduino per il prescaler
  - Dividere il risultato per la frequenza desiderata
  - Se il valore è maggiore del valore massimo del registro, aumentare il prescaler

Calcolo della frequenza del Timer

- Esempio: ipotizziamo di voler avere una frequenza di 20 Hz, usando il Timer1 (16 bit) con il prescaler a 128
  - 16.000.000/128 = 125000
  - 125000/20 = 6250
  - 6250 < 65536 OK!!!
- Esempio 2: portiamo il prescaler a 8!
  - 16000000/8 = 2000000
  - 250000 / 20 = 100000
  - 100.000 > 65536 KO!!! Cambio prescaler

TCNT, OCR e ICR

- TCNT1: Timer/Counter Register. Conta i Clock ticks, direttamente o attraverso un prescaler o sfrutta un clock da un pin.
- OCR1: Output Compare register è usato per generare un Interrupt dopo che il numero di clock ticks che contiene è raggiunto.
  - È confrontato costantemente con TNCT1.
  - Quando il due valori corrispondono, viene triggerato l'Interrupt.
  - Se si vuole ripetere , il bit di CTC (in TCCR1B) va impostato.
- ICR1: Input Capture viene usato per misurare il tempo tra due impulsi su Pin esterni o external ICP (Input Capture Pin) Pin
  - Il come avviene la connessione tra ICP e ICR viene specificata con i bit ICNC e ICES di TCCR1A.

TIMSK and TIFR

- Il Timer Interrupt Mask (TIMSK) Register e il Timer Interrupt Flag (TIFR) Register si usano per controllare quali interrupt sono attivi (TIMSK) e quali di questi sono triggerati (TIFR)
- TIMSK
  - ICIE1: Timer/Counter1, Input Capture Interrupt Enable.
  - OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable
  - OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable
  - TOIE1: Timer/Counter1, Overflow Interrupt Enable
- TIFR
  - ICF1: Timer/Counter1, Input Capture Flag
  - OCF1B: Timer/Counter1, Output Compare B Match Flag
  - OCF1A: Timer/Counter1, Output Compare A Match Flag
  - TOV1: Timer/Counter1, Overflow Flag

TIMSK and TIFR

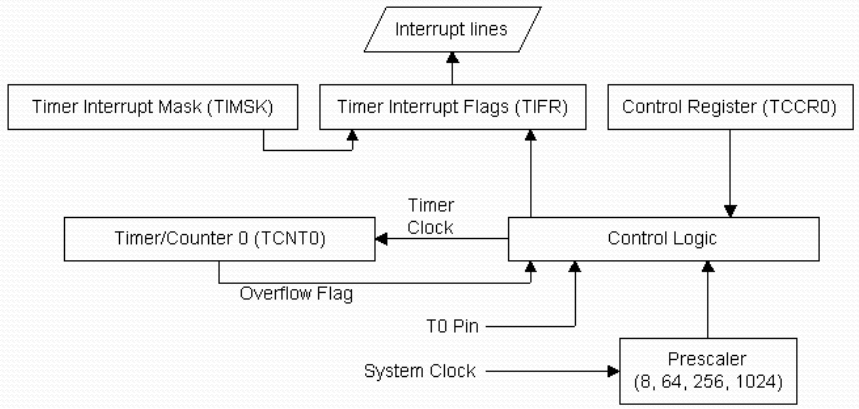
16.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

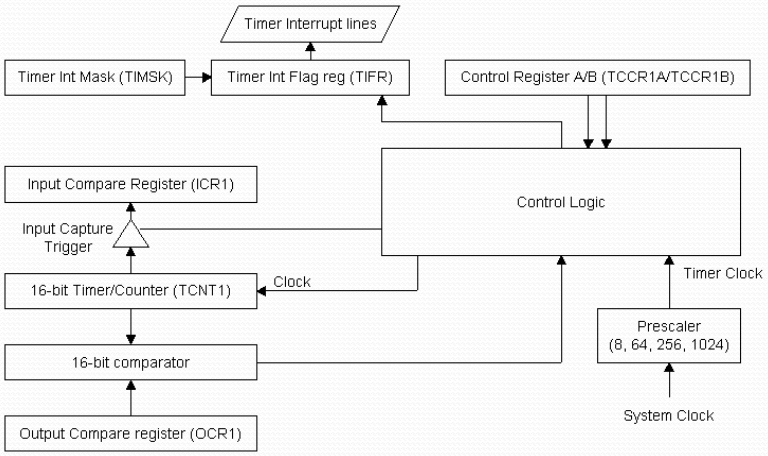
16.11.9 TIFR1 – Timer/Counter1 Interrupt Flag Register

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

8 bit Timer



16 bit Timer



Timer Interrupt

- Un timer può generare diversi tipi di Interrupt.
- Il dettaglio dei registri e dei bit si trovano in due posti:
  - nel datasheet
  - nel file di header delle definizioni di I/O(ad esempio iomx8.h per Arduino UNO nella cartella hardware/tools/avr/include/avr).
- Il suffisso x indica il numero del Timer (0-2 o 0-5 a seconda del modello), mentre il suffisso y indica il numero di registro (A,B,C):
  - TIMSK1 (Timer1 interrupt mask register)
  - OCR2A (Timer2 output compare register A).

Timer Interrupt

Si possono generare 4 tipi di Timer Interrupt:

- Timer Overflow:
- Output Compare Match:
- Timer Input Capture:
- PWM

Timer Overflow (o modo normale)

- Quando si verifica, il timer overflow bit TOVx viene impostato nel interrupt flag register TIFRx.
- Se il timer overflow interrupt enable bit TOIEx nel interrupt mask register TIMSKx è impostato, viene richiamata la ISR(TIMERx_OVF_vect).

Timer Overflow

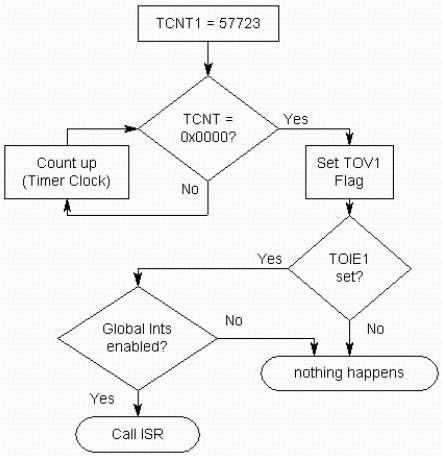
- Nel funzionamento normale, TCNT1 si incrementa e triggera un interrupt quando passa da oxFFFF a 0x0000
  - Non basta caricare il numero di ticks in TCNT1 e aspettare.
  - Va caricato la differenza tra oxFFFF e il numero di ticks.
- Facciamo un esempio:
  - Frequenza di esempio 8MHz
  - vogliamo un timer a 1s
  - vogliamo un prescaler da 1024
  - 8.000.000/1024 = 7812,5 -> 7812
  - oxFFFF - 7812 = 57723 -> caricato in TCNT1

Timer Overflow

In pratica i passi da fare sono:

- impostare il prescaler (CS12 e CS10 in TCCR1B)
- caricare oxFFFF-ticks in TCNT1
- abilitare TOIE1 in TIMSK
- abilitare in SREG i global interrupt
- … aspettare!

Timer Overflow



Esempio Timer Overflow

```
#define LEDPIN 2

void setup()
{
    pinMode(LEDPIN, OUTPUT);

    // initialize Timer1
    cli();             // disable global interrupts
    TCCR1A = 0;        // set entire TCCR1A register to 0
    TCCR1B = 0;

    TIMSK1 = (1 << TOIE1);    // enable Timer1 overflow interrupt
    TCCR1B |= (1 << CS10);    // Set CS10 bit so timer runs at
                              // clock speed:
    sei();                    // enable global interrupts:

}
```


Output compare match

- Quando si verifica, viene impostato il flag OCFxy nel interrupt flag register TIFRx .
- Se il bit Output Compare Interrupt Enable OCIExy nel interrupt mask register TIMSKx è impostato, viene richiamata la ISR(TIMERx_COMPy_vect)

Output Compare Match

- Il modo Output Compare viene usato per timing ripetitivi.
- Il valore di TNCT1 viene costantemente confrontato con OCR1A.
- Quando questi valori sono uguali il flag Output Compare Interrupt (OCF in TIFR) viene impostato e viene chiamata la ISR associata.
- Se si imposta in TCCR1B il bit CTC1 il timer viene riavviato quando si verifica il Compare Match.

Output Compare Match

Facciamo un esempio.

- Supponiamo di voler triggerare un interrupt ogni 10ms
- Supponiamo sempre di avere un clock di 8MHz.
- Ci serve un prescaler da 8
- Carichiamo 10.000 in OCR (NB non va caricato 0xFFFF - 10.000)
- Impostiamo CTC1
- Impostiamo il bit OCIE1A in TIMSK per specificare che vogliamo che si attivi un interrupt.

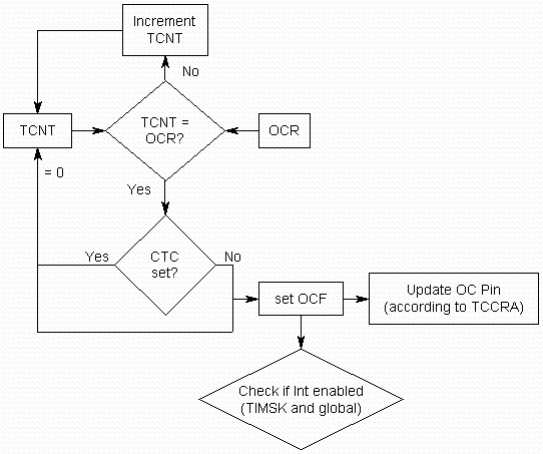
Output Compare Match

- Nota 1: se non si imposta CTC1:
  - raggiunto il valore in OCR, TCNT1 continua ad aumentare
  - raggiunge oxFFFF e riparte.
  - Quindi l'interrupt successivo si verifica non a 10 ms ma a (oxFFFF - 10.000) + 10.000, cioè a 0.065536 s.

Output Compare Match

- Nota 2: se vogliamo usare OC1 (Output Compare 1), dobbiamo specificare in TCCR1A coma va usato.
- Abbiamo tre possibilità: set, clear o toggle.
- Attenzione se si sceglie il toggle... la frequenza va dimezzata: 5ms -> toggle on -> 5ms -> toggle off per avere un impulso ogni 10 ms.

Output Compare Match



Esempio di Output compare match

```
#define ledPin 13
void setup(){
  pinMode(ledPin, OUTPUT);
  // initialize timer1
  noInterrupts();           // disable all interrupts
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0;
  //compare match register 16MHz/256/2Hz
  OCR1A = 31250;
  TCCR1B |= (1 << WGM12);         // CTC mode
  TCCR1B |= (1 << CS12);          // 256 prescaler
  TIMSK1 |= (1 << OCIE1A);        // enable timer compare
                                  // interrupt
  interrupts();                   // enable all interrupts
}
```

Esempio di Timer

```
//timer compare interrupt service routine
// toggle LED pin
ISR(TIMER1_COMPA_vect){
  digitalWrite(ledPin, digitalRead(ledPin) ^ 1);
}
void loop(){
  // your program here.
}
```


Timer Input Capture

- Quando si verifica, viene impostato il flag ICFx nel interrupt flag register TIFRx.
- Se il input capture interrupt enable bit ICIEx nel interrupt mask register TIMSKx is set, è impostato, viene richiamata la ISR(TIMERx_CAPT_vect).

Timer Input Capture

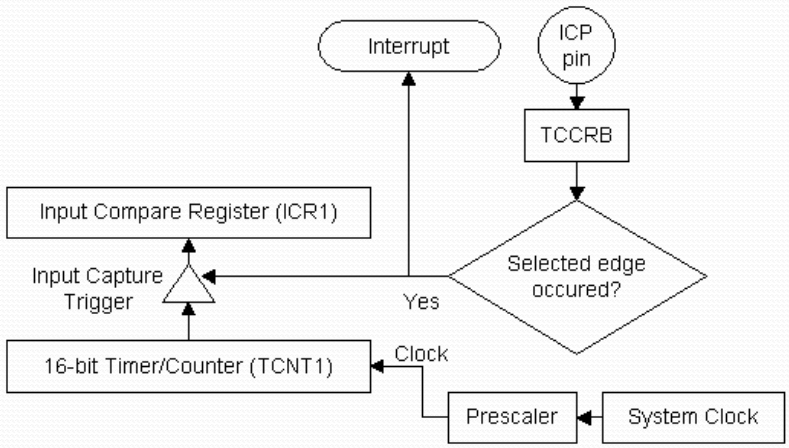
- Il modo Input Capture, può essere usato per misurare il tempo tra due edge che si verificano sul pin ICP (Input Capture Pin).
  - Si può usare dei circuiti per generare impulsi da misurare.
  - Si possono misurare i giri di un motore.
  - Si può misurare la durata di un impulso

Timer Input Capture

Per misurare la durata di un impulso:

- Impostiamo il bit ICES (Input Capture Edge Select) a 1 (detect rising edge)
- Quando viene triggerato l'interrupt e il flag va Alto, nella ISR va cambiato ICES a 0 per rilevare l'edge negativo e va azzerato TCNT1.
- Quando si verifica il nuovo trigger, ci troviamo in TCNT1 il numero di ticks in cui il segnale è stato alto e il flag va basso.
- Se all'interno della ISTR cambiamo di nuovo ICES e riazzeriamo TCNT1, quando il segnale triggera di nuovo abbiamo il numero di ticks in cui è stato basso.
- In questo modo abbiamo la durata alta, la durata bassa e possiamo calcolare il duty cycle e la durata del segnale.

Timer Input Capture



PWM

... 'naltra volta!

Se interessa merita una serata a parte!

The easy way… TimerOne

- Ok ma è l'unico modo di usare i timer?
  - Se vogliamo un controllo preciso… si
  - Esiste tuttavia una (due) comoda libreria (in evoluzione continua) che ci semplifica la vita (e di tanto!!!)
- TimerI.h

TimerOne: metodi

```
void initialize(long microseconds=1000000);
void start();
void stop();
void restart();
unsigned long read();
void setPeriod(long microseconds);
void pwm(char pin, int duty, long microseconds=-1);
void setPwmDuty(char pin, int duty);
void disablePwm(char pin);
void attachInterrupt(void (*isr)(), long
microseconds=-1);
void detachInterrupt();
```

Esempio

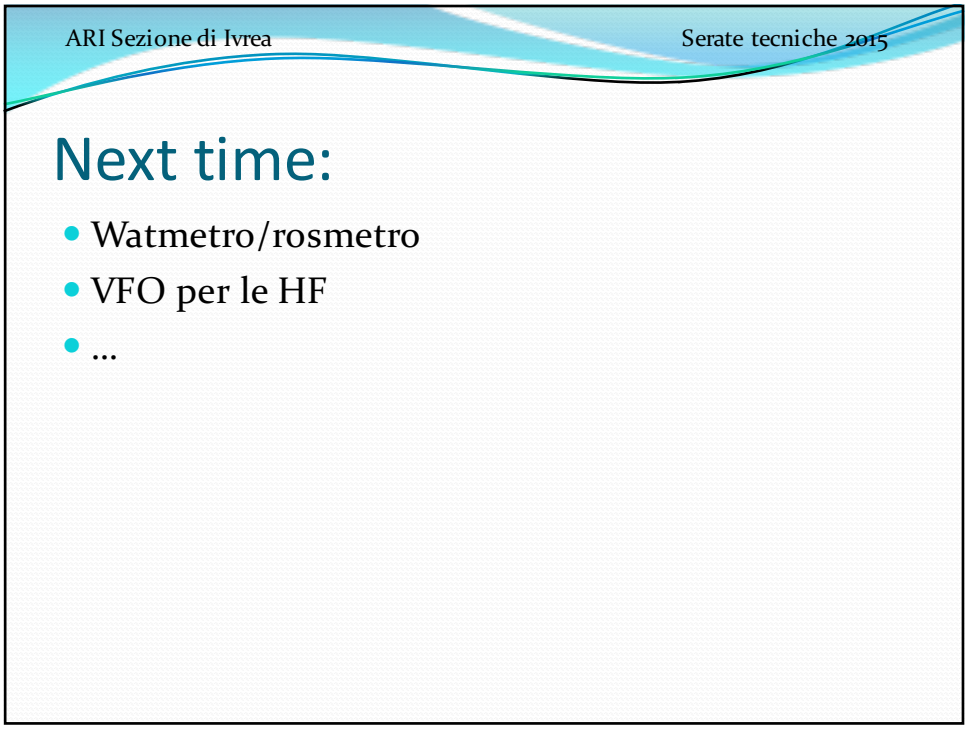
```
#include "TimerOne.h"
void setup()
{
  pinMode(10, OUTPUT);
  Timer1.initialize(500000);          // initialize timer1, and set a 1/2 second period
  Timer1.pwm(9, 512);                 // setup pwm on pin 9, 50% duty cycle
  Timer1.attachInterrupt(callback);  // attaches callback() as a timer overflow
                                      // interrupt
}

void callback()
{
  digitalWrite(10, digitalRead(10) ^ 1);
}

void loop()
{
  // your program here...
}
```


Attivare e disattivare gli Interrupt

- `noInterrupts()` - `cli()`
  - Disattivo i global interrupt
- `interrupts()` - `sei()`
  - Attivo i global interrupt


ARI Sezione di Ivrea
Serate tecniche 2015
Next time:
Watmetro/rosmetro
VFO per le HF
…