ilguru.eu

Interrupt esterni – Blog del Guru

9-12 minuti

In questo articolo mostreremo come operare con gli interrup relativi al cambiamento di stato logico dei pin delle porte di I/O digitali. Effettueremo tutti gli esempi utilizzando una mcu Atmega328p con clock a 16 MHz.

Alcuni pin delle mcu avr sono contrassegnati con le label **INTn** e **PCINTn**. Questi pin possono essere impostati come sorgenti di interrupt sul cambiamento dello stato logico, sia che siano configurati come ingressi sia che siano configurati come uscite e possono essere utilizzati per risvegliare la mcu dagli stati sleep ed idle

Descrizione dei registri coinvolti

SREG: AVR Status Register

SREG							
I	T	H	S	V	N	Z	C

L'unico bit che ci interessa ai fini del controllo delle interrupt è il settimo bit denominato I (Global Interrupt Enable). Questo bit permette di abilitare o disabilitare globalmente tutte le interrupt e può essere impostato ad 1 tramite la macro C **sei()** che si traduce nell'istruzione assembler **sei** (Set Global Interrupt Flag)

Analogamente, può essere resettato a 0 tramite la macro C **cli()** che si traduce nell'istruzione assembler **cli** (Clear Global Interrupt Flag)

EICRA: External Interrupt Control Register A
Questo registro di cui è possibile impostare solo il nibble basso controlla quale stato logico dei pin INT0 ed INT1 generano le richieste External Interrupt

EICRA							
–	–	–	–	ISC11	ISC10	ISC01	ISC00

ISC11 ed ISC10 controllano INT1 mentre ISC01 ed ISC00 controllano INT0.
Le due coppie di bit possono assumere 4 configurazioni, ciascuna delle quali seleziona lo stato del pin INTn che genera la richiesta di interrupt

ISCn1	ISCn0	Attivazione interrupt
0	0	Stato logico 0
0	1	Ogni stato logico
1	0	Fronte di discesa della transazione da stato logico 1 a 0
1	1	Fronte di salita della transazione da stato logico 0 ad 1

EIMSK: External Interrupt Mask Register
Questo registro è composto dai soli due bit meno significativi

EIMSK							
–	–	–	–	–	–	INT1	INT0

La funzione di ciascun bit è quella di abilitare le richieste di interrupt sul pin omonimo

EIFR: External Interrupt Flag Register
Come EIMSK è composto dai soli due bit meno significativi

EIMSK							
–	–	–	–	–	–	INTF1	INTF0

Questi due bit sono i flag che vengono impostati ad 1 quando viene generata la relativa richiesta di interrupt in modo da inibire l'esecuzione di una ulteriore richiesta di interrupt identica.
Quando il bit I del registro SREG ed il bit INTn del registro EIMSK sono impostati ad 1, viene eseguita la ISR relativa al vettore di interrupt INTn.
Il flag INTFn rimane settato ad 1 fino alla conclusione della ISR dopo di che viene resettato a 0. Il flag può essere resettato all'interno della ISR scrivendoci dentro un 1

PCICR: Pin Change Interrupt Control Register
Questo registro è composto dai tre bit meno significativi

PCICR							
–	–	–	–	–	PCIE2	PCIE1	PCIE0

La funzione di ciascun bit è quella di abilitare le richieste interrupt change. Quando SREG.I è impostato ad 1 e PCIEn è impostato ad 1, la interrupt change n è abilitata. Ognuna delle interrupt change n esegue la ISR relativa al vettore di interrupt PCIn e fa capo ad un insieme di pin denominati con le label PCINTn che vengono abilitati individualmente tramite i registri PCMSKn

- PCIE0: Fa riferimento ai pin da PCINT0 a PCINT7
- PCIE1: Fa riferimento ai pin da PCINT8 a PCINT14
- PCIE2: Fa riferimento ai pin da PCINT16 a PCINT23

PCIFR: Pin Change Interrupt Flag Register
Questo registro è composto dai tre bit meno significativi

PCIFR							
–	–	–	–	–	PCIF2	PCIF1	PCIF0

Quando cambia lo stato logico di uno dei pin PCINTn, il bit SREG.I e il bit PCICR.PCIEn relativo sono settati ad 1, la mcu esegue la ISR relativa al vettore di interrupt PCIn ed il bit PCIFn viene settato ad 1. Rimane in questo stato durante l'esecuzione della ISR dopo di che viene resettato a 0.
Il flag può essere resettato all'interno della ISR scrivendoci dentro un 1

PCMSK[0:2]: Pin Change Mask Register [0:2]
Questo registro è composto dai tre bit meno significativi

PCMSK0

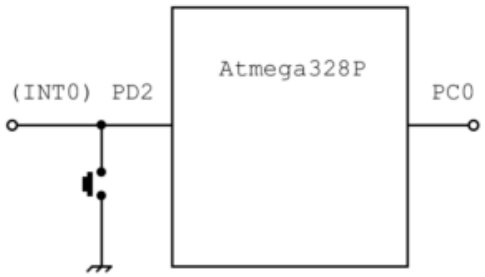
PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
PCMSK1							
PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8
PCMSK2							
PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16

Ogni bit abilita interrupt change sul pin corrispondente

Schema riepilogativo

Interrupt	Pin	Global Enable	Control Register	Enable Mask	Flag Register	Interrupt Vector
INT0	INT0	SREG.I	EICRA.ISC00 EICRA.ISC01	EIMSK.INT0	EIFR.INTF0	INT0_vect
INT1	INT1	SREG.I	EICRA.ISC10 EICRA.ISC11	EIMSK.INT1	EIFR.INTF1	INT1_vect
PCI0	PCINT[7:0]	SREG.I	PCICR.PCIE0	PCMSK0.PCINT[7:0]	PCIFR.PCIF0	PCINT0_vect
PCI1	PCINT[15:8]	SREG.I	PCICR.PCIE1	PCMSK1.PCINT[15:8]	PCIFR.PCIF1	PCINT1_vect
PCI2	PCINT[23:16]	SREG.I	PCICR.PCIE2	PCMSK2.PCINT[23:16]	PCIFR.PCIF2	PCINT2_vect

Esempio: INT0



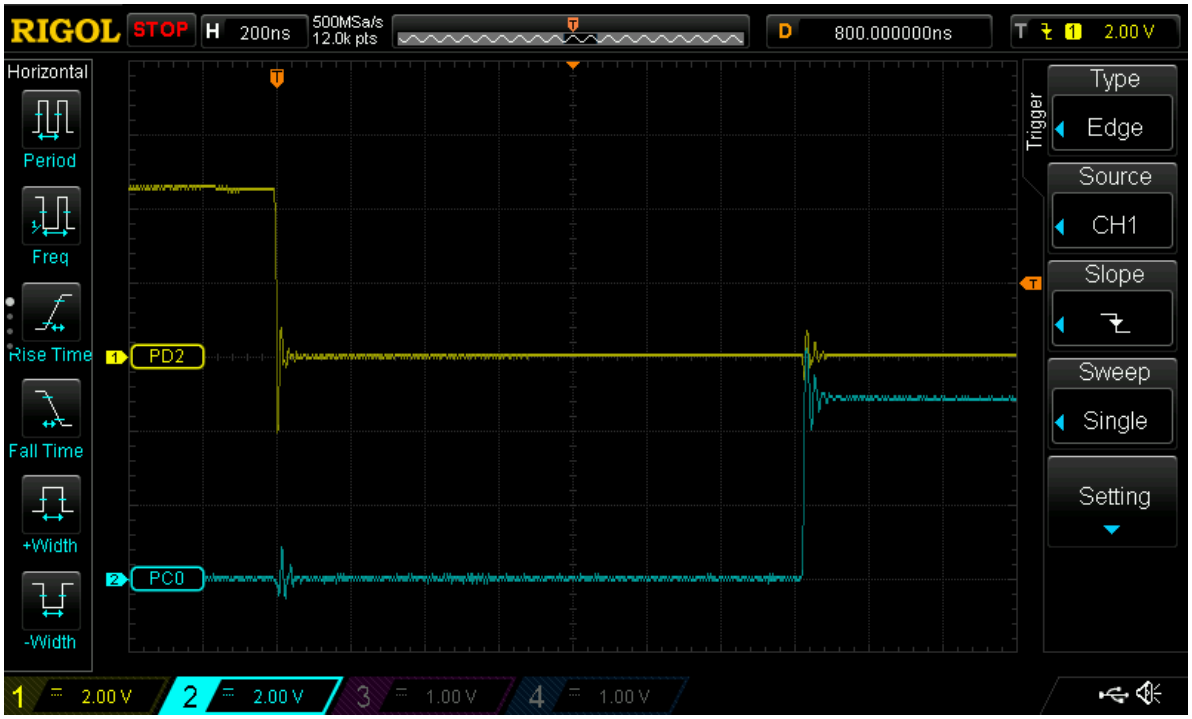
In questo esempio abiliteremo l'interrupt INT0 che viene generato dal pin 2 della porta D della mcu. Imposteremo questo pin come ingresso con la pull-up attiva pertanto sarà presente lo stato logico 1 che verrà portato a 0 con pressione di un pulsante. Imposteremo il registro di controllo per generare l'interrupt sul fronte di discesa del pin PD2/INT0. La ISR invertirà lo stato logico del pin PC0 che viene impostato a 0 all'inizio del programma

```
1  #include <avr/io.h>
2  #include <avr/interrupt.h>
3  ISR( INT0_vect ){
4    if ( bit_is_set( PORTC, PORTC0 ) ){
5      PORTC &= ~(_BV( PORTC0 ));
6    } else {
7      PORTC |=  _BV( PORTC0 );
```

```
8     }
9   }
10  int main( void ){
11    //    PORTA C Pin0 Output
12    DDRC |=   _BV( DDC0 );
13    //    PORTA D Pin2 Input
14    DDRD &= ~(_BV( DDD2 ));
15    //    PORTA C Pin0 LO
16    PORTC &= ~(_BV( PORTC0 ));
17    //    PORTA D Pin2 Pull-Ups attiva
18    PORTD |=   _BV( PORTD2 );
19    //    Attendo che si stabilizzi lo stato di PD2
20    loop_until_bit_is_set( PIND, PIND2 );
21    //    Control Register: Interrupt su fronte di discesa
22    EICRA &= ~(_BV( ISC00 ));
23    EICRA |=   _BV( ISC01 );
24    //    Enable Mask
25    EIMSK |=   _BV( INT0 );
26    //    Attivo globalmente interrupt
27    sei();
28    //    Loop
29    while( 1 ){
30      ;
31    }
32    return 0;
33  }
34
35
36
37
38
39
40
41
```

```
42
43
```

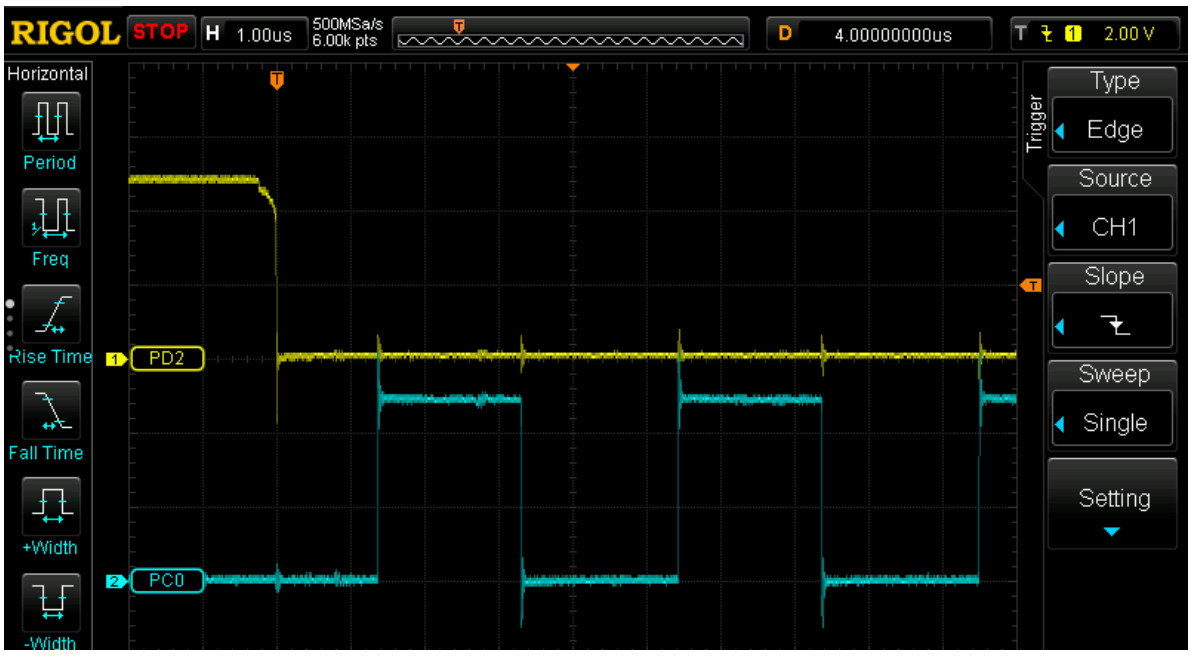
Diagramma temporale



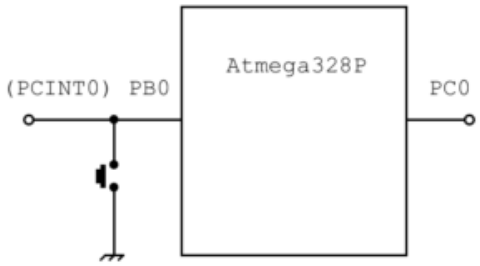
Cosa sarebbe accaduto se l' interrupt INT0 fosse stato attivo sullo stato logico 0 di PD2?

```
..
  //    Control Register: Interrupt su stato logico 0
  EICRA &= ~(_BV( ISC00 ));
  EICRA &= ~(_BV( ISC01 ));
..
```

La ISR verrebbe richiamata per tutto il tempo in cui il pulsante rimane premuto continuando ad alternare lo stato logico di PC0



Esempio: PCINT0

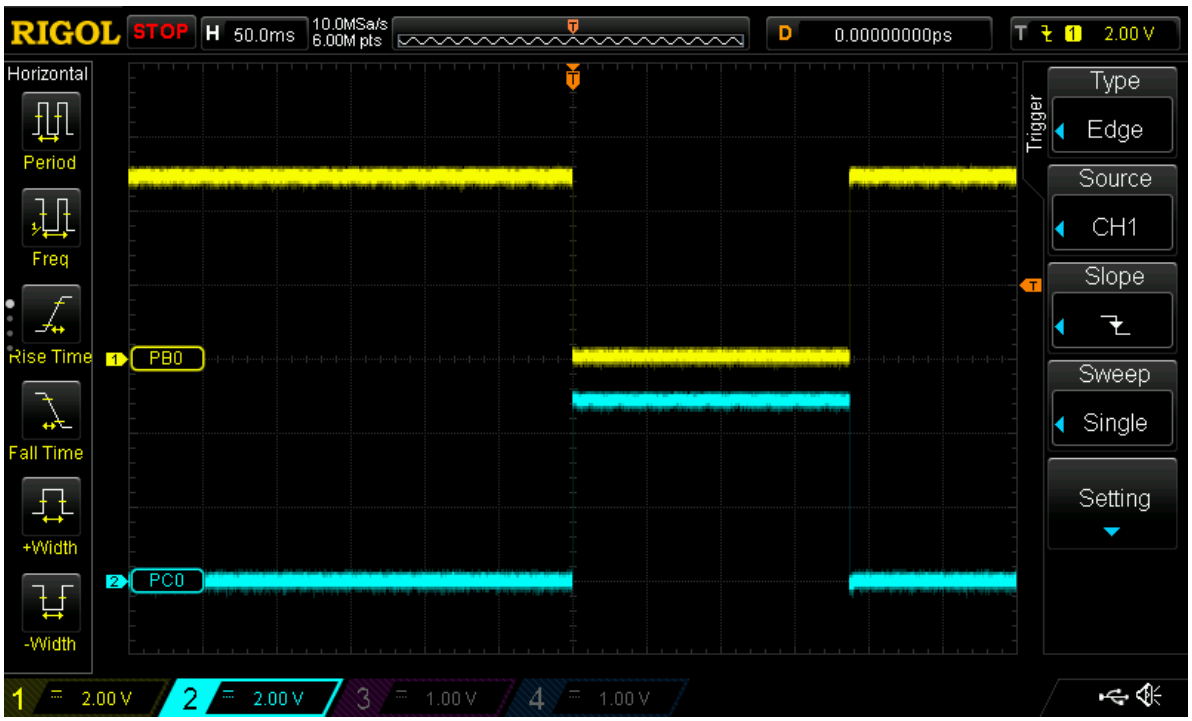


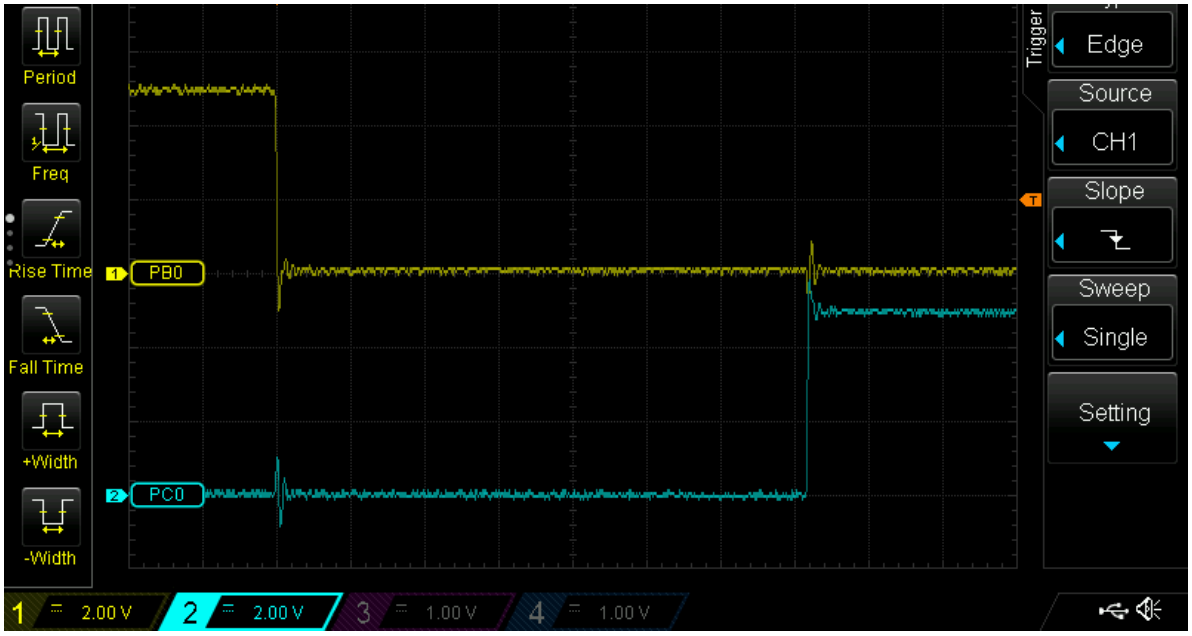
In questo esempio abiliteremo l'interrupt PCINT0 che viene generato sul pin 0 della porta B su ogni transizione di stato pertanto dovremo testare il valore del pin. Lo stato logico è mantenuto normalmente ad 1 tramite la pull-up interna della mcu, viene portato a 0 tramite la pressione del pulsante e riportato ad 1 al suo rilascio. La ISR alterna il valore di uscita del pic 0 della porta C

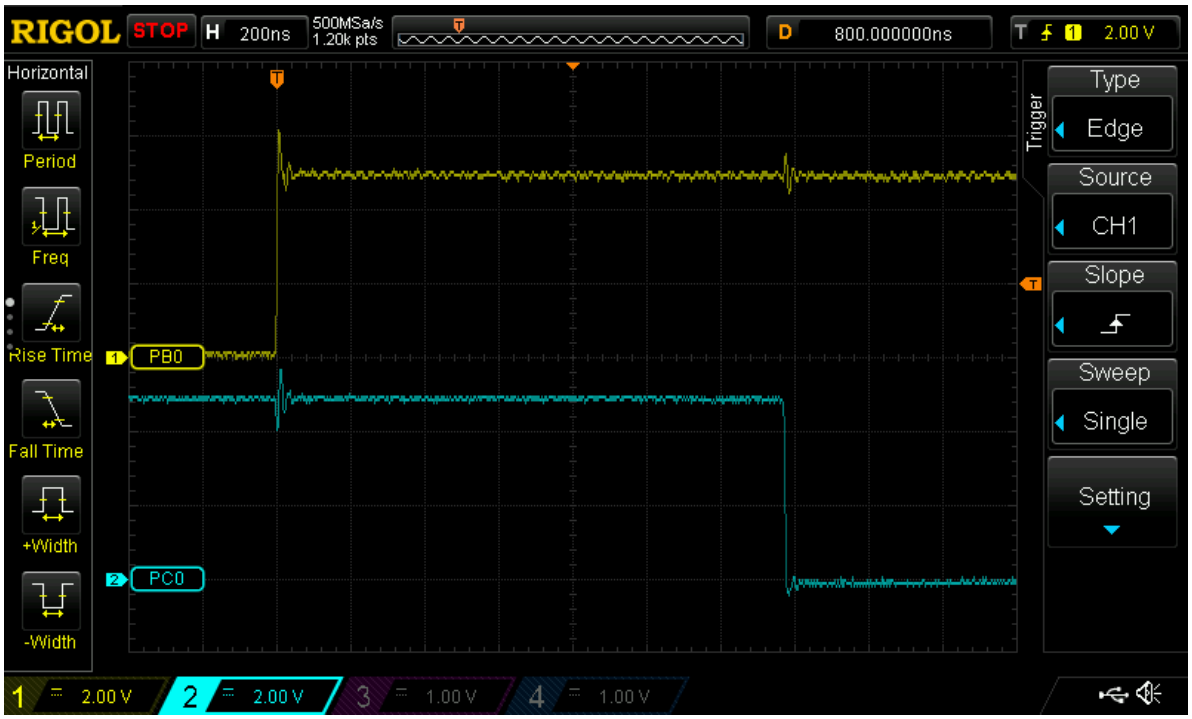
```
1  #include <avr/io.h>
2  #include <avr/interrupt.h>
3  ISR( PCINT0_vect ){
4    if ( bit_is_set( PORTC, PORTC0 ) ){
5      PORTC &= ~(_BV( PORTC0 ));
6    } else {
7      PORTC |=  _BV( PORTC0 );
8    }
9  }
10 int main( void ){
11   //   PORTA C Pin0 Output
12   DDRC |=  _BV( DDC0 );
13   //   PORTA B Pin0 Input
14   DDRB &= ~(_BV( DDB0 ));
15   //   PORTA C Pin0 LO
16   PORTC &= ~(_BV( PORTC0 ));
17   //   PORTA B Pin0 Pull-Ups attiva
18   PORTB |=  _BV( PORTB0 );
19   //   Attendo che si stabilizzi lo stato di PD2
20   loop_until_bit_is_set( PINB, PINB0 );
21   //   Control Register
22   PCICR |=  _BV( PCIE0 );
23   //   Enable Mask
```

```
24    PCMSK0|=   _BV( PCINT0 );
25    //    Attivo globalmente interrupt
26    sei();
27    //    Loop
28    while( 1 ){
29      ;
30    }
31    return 0;
32  }
33
34
35
36
37
38
39
40
41
42
```

In questo caso non testiamo il valore di PB0 all'interno della ISR quindi il valore PC0 viene modificato sia quando il pulsante viene premuto sia quando viene rilasciato





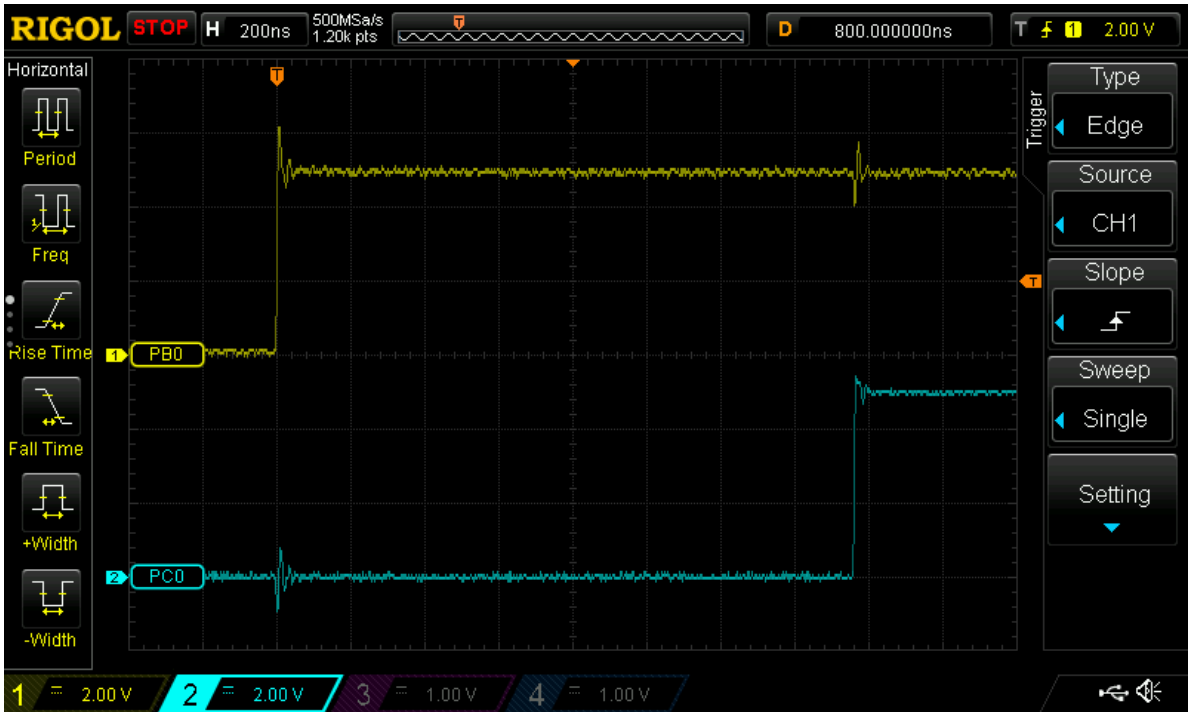


Testiamo lo stato 1 di PB0 all'interno della ISR

```
ISR( PCINT0_vect ){
  if ( bit_is_set( PINB, PINB0 ) ){
    if ( bit_is_set( PORTC, PORTC0 ) ){
      PORTC &= ~(_BV( PORTC0 ));
    } else {
      PORTC |=  _BV( PORTC0 );
    }
  }
}
```

PC0 viene modificato solo quando PB0 passa da 0 ad 1 e la ISR lo trova set

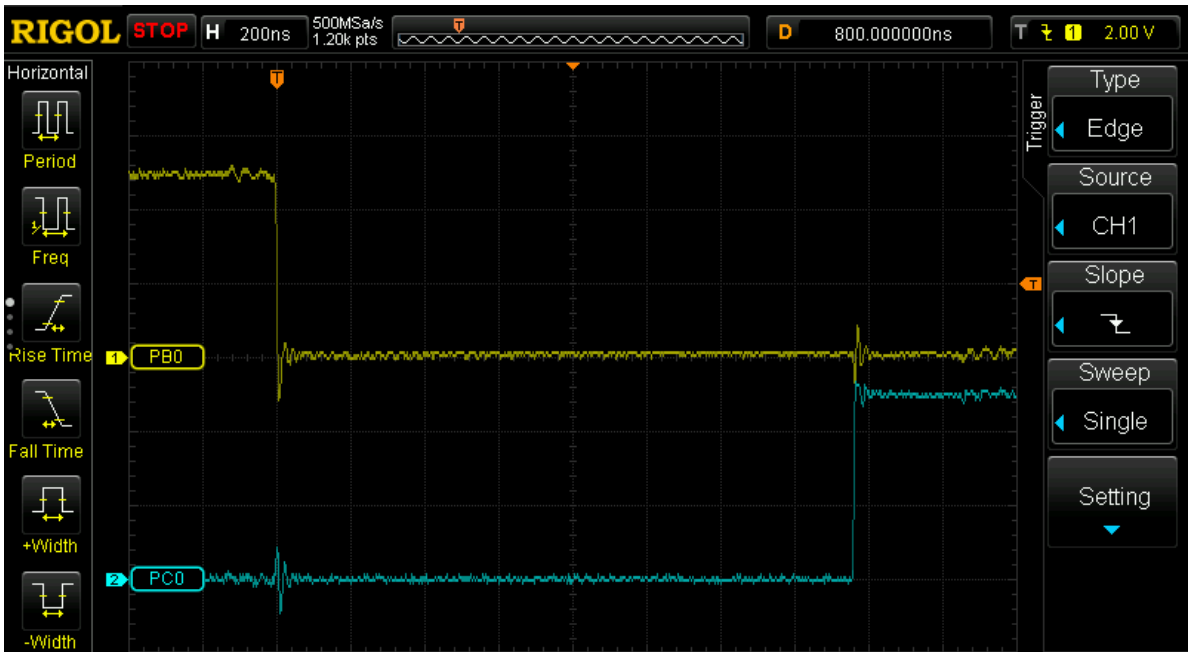
---



Testiamo lo stato 0 di PB0 all'interno della ISR

```
ISR( PCINT0_vect ){
  if ( bit_is_clear( PINB, PINB0 ) ){
    if ( bit_is_set( PORTC, PORTC0 ) ){
      PORTC &= ~(_BV( PORTC0 ));
    } else {
      PORTC |=  _BV( PORTC0 );
    }
  }
}
```

PC0 viene modificato solo quando PB0 passa da 1 a 0 e la ISR lo trova clear



Esempio: PCINT0 software

In questo esempio non utilizziamo nessun pulsante, vale tutto come nell' esempio precedente soltanto che PB0 lo impostiamo come uscita ed alterandone il valore generiamo la richiesta PCINT0. Utilizziamo la macro _delay_us per mantenere PC0 allo stato logico 0 per 80 micro secondi

```
1  #include <avr/io.h>
2  #include <avr/interrupt.h>
3  #include <util/delay.h>
4  ISR( PCINT0_vect ){
5    if ( bit_is_set( PORTC, PORTC0 ) ){
6      PORTC &= ~(_BV( PORTC0 ));
7    } else {
8      PORTC |=   _BV( PORTC0 );
9    }
10 }
11 int main( void ){
12   //    PORTA C Pin0 Output
13   DDRC |=   _BV( DDC0 );
14   //    PORTA B Pin0 Output
15   DDRB |=   _BV( DDB0 );
16   //    PORTA C Pin0 LO
17   PORTC &= ~(_BV( PORTC0 ));
18   //    PORTA B Pin0 HIGH
19   PORTB |=   _BV( PORTB0 );
20   //    Attendo che si stabilizzi lo stato di PD2
21   loop_until_bit_is_set( PINB, PINB0 );
22   //    Control Register
23   PCICR |=   _BV( PCIE0 );
24   //    Enable Mask
25   PCMSK0|=   _BV( PCINT0 );
26   //    Attivo globalmente interrupt
27   sei();
28   PORTB &= ~(_BV( PORTB0 ));
29   _delay_us( 80 );
```

```
30    PORTB |=  _BV( PORTB0 );
31    //   Loop
32    while( 1 ){
33      ;
34    }
35    return 0;
36  }
37
38
39
40
41
42
43
44
45
46
47
```

Come nell'esempio precedente la ISR viene eseguita ad ogni commutazione di PB0 che essendo configurato come uscita viene modificato dal software