

Progetto di un Frequenzimetro con Arduino UNO

© Politecnico di Torino

Questo materiale è distribuito gratuitamente ad esclusivo uso degli allievi del Politecnico di Torino per la preparazione all'esame. Ogni altro uso sia commerciale sia divulgativo è espressamente vietato senza il consenso scritto dell'autore

Frequenzimetro

Obiettivo:

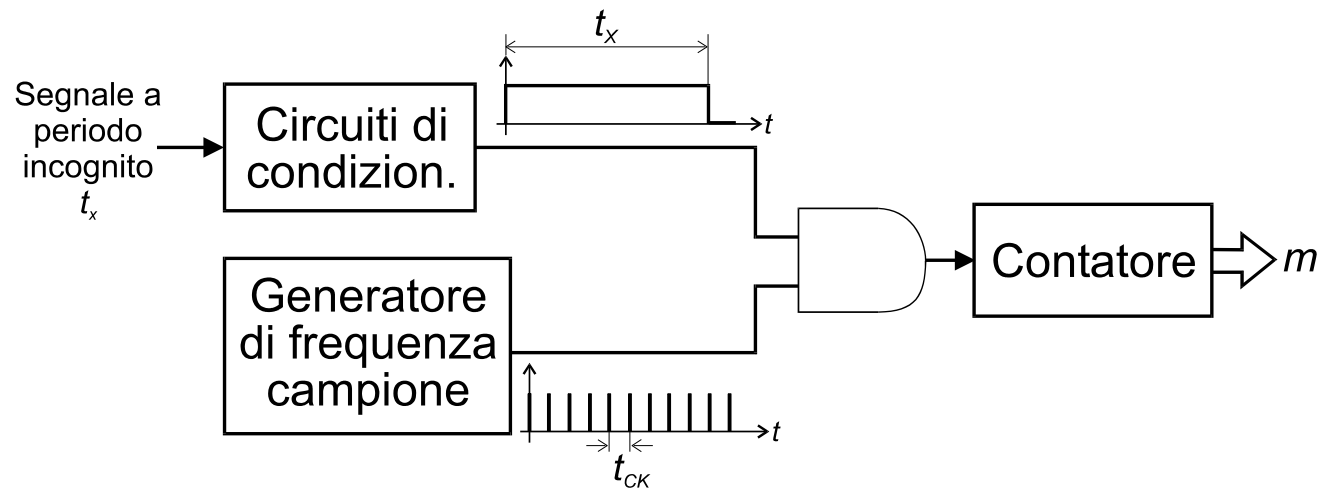
Caratterizzare il clock di Arduino e realizzare un frequenzimetro con le seguenti specifiche:

- Per segnali digitali
- Campo di misura: 1 Hz – 100 kHz
- Misura diretta di periodo
- Risoluzione: almeno 10^{-7} con tempo di misura di 1 s
- Incertezza migliore consentita dal clock di Arduino
- Tempo di gate impostabile (in software/da terminale)

Frequenzimetro

A misura diretta di periodo

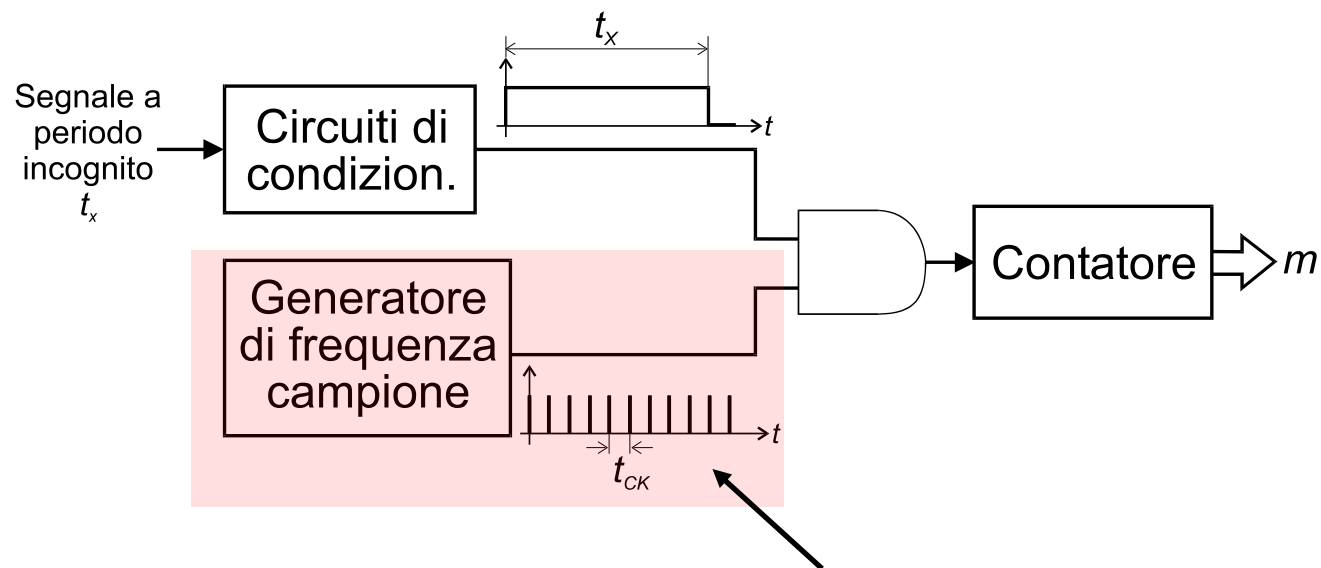
Schema di principio: si contano i periodi t_{CK} contenuti in un periodo di t_x



Frequenzimetro

A misura diretta di periodo

Schema di principio: si contano i periodi t_{CK} contenuti in un periodo di t_x

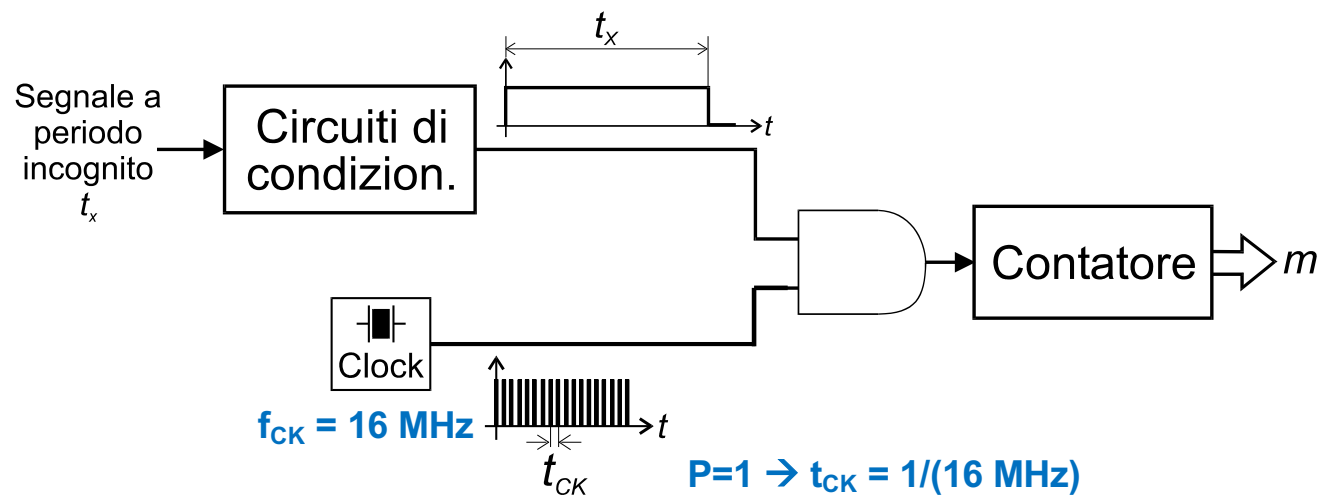


Possiamo contare i periodi del clock di sistema con prescaler 1:1

Frequenzimetro

A misura diretta di periodo

Schema di principio: si contano i periodi t_{CK} contenuti in un periodo di t_x

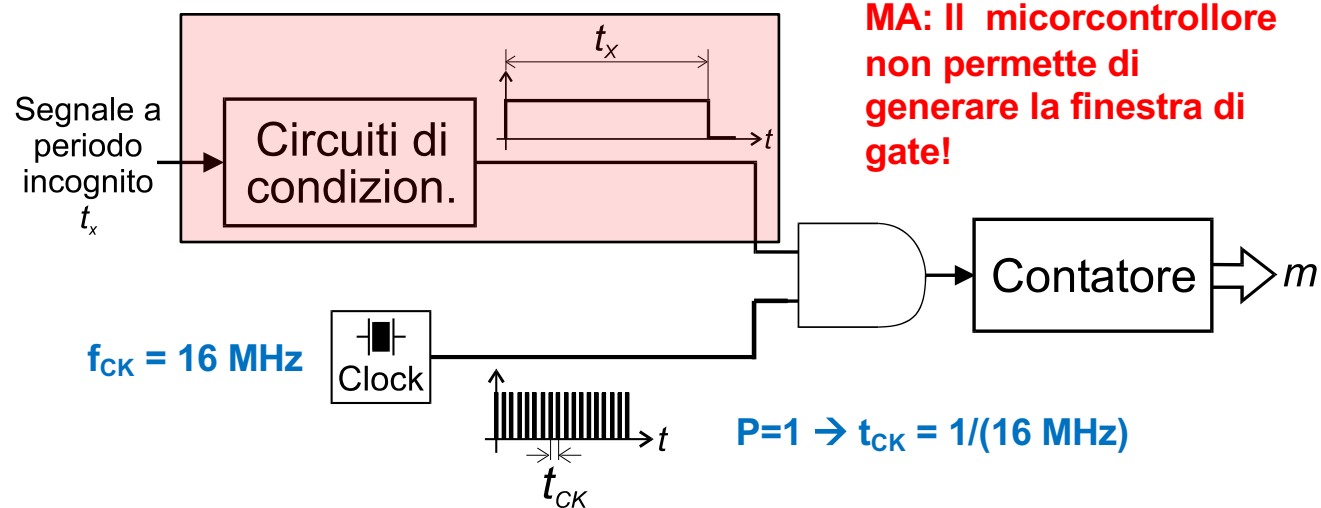


✓ Ok misura diretta di periodo! ($f_{x,MAX} = 100 \text{ kHz}$)

Frequenzimetro

A misura diretta di periodo

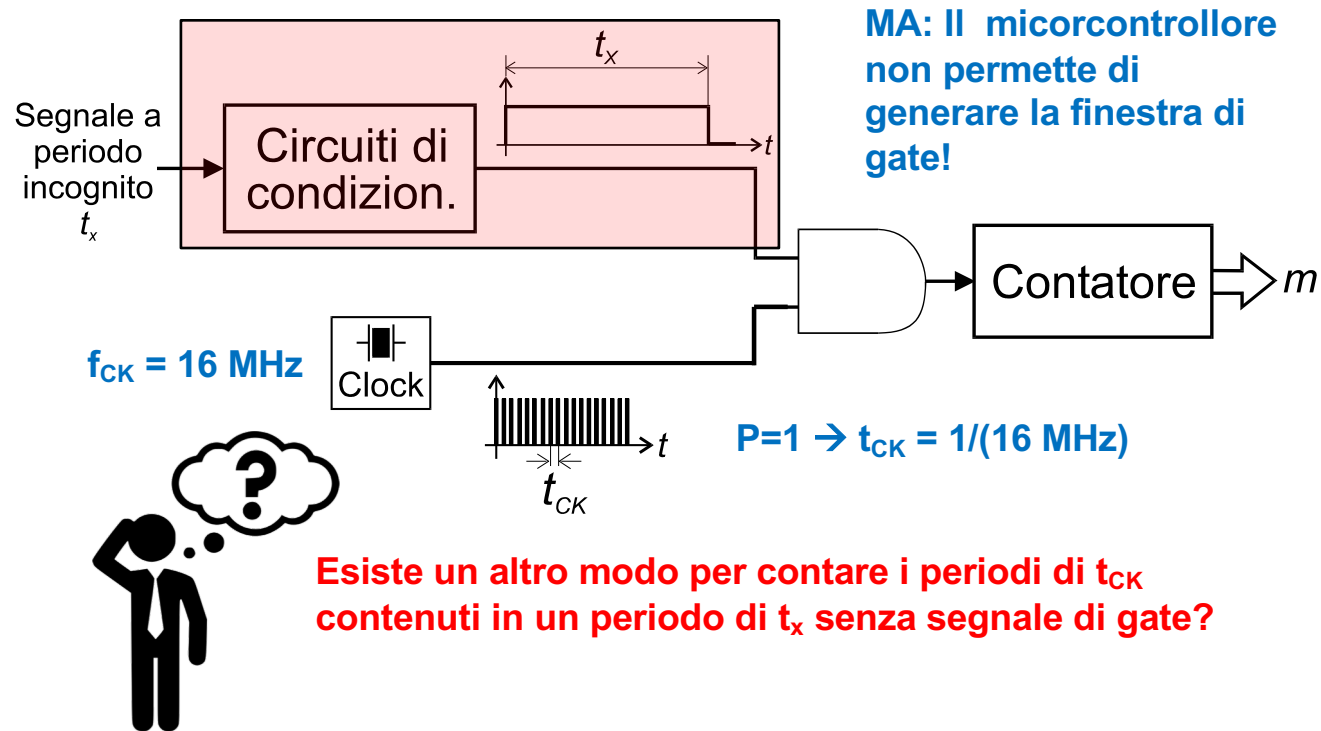
Schema di principio: si contano i periodi t_{CK} contenuti in un periodo di t_x



Frequenzimetro

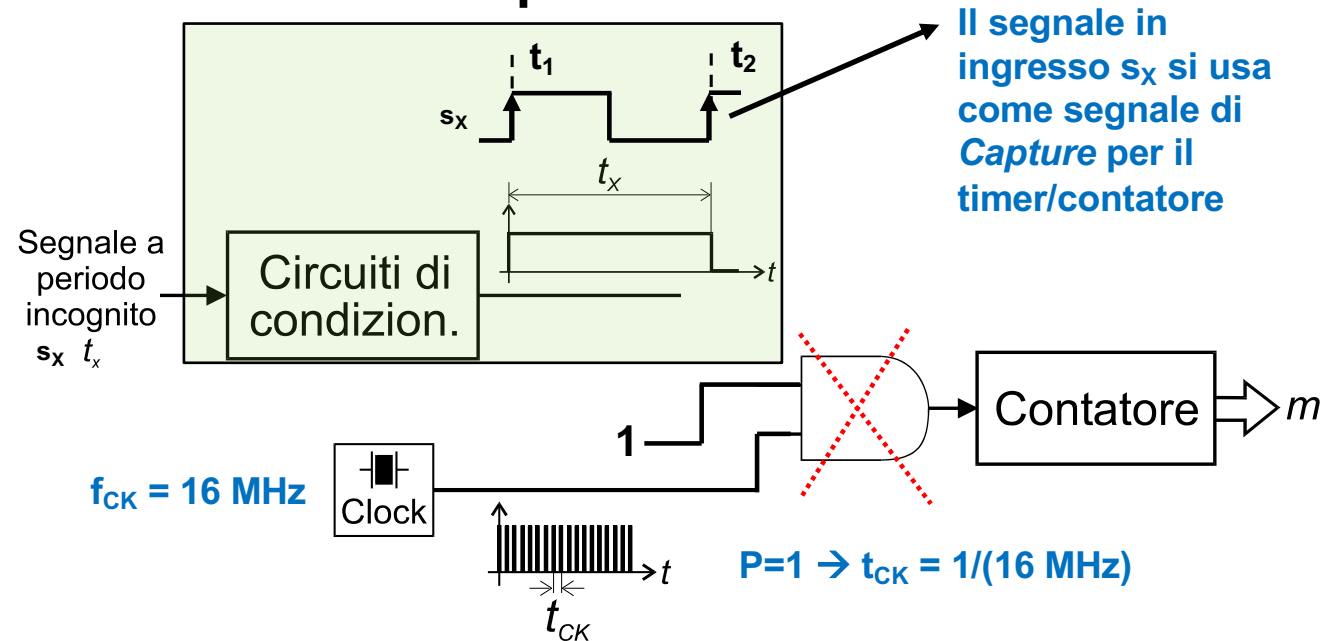
A misura diretta di periodo

Schema di principio: si contano i periodi t_{CK} contenuti in un periodo di t_x



Frequenzimetro

A misura diretta di periodo

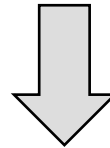


Si conta continuamente e si calcola $m = m_2 - m_1$ catturati agli istanti t_1 e t_2

Frequenzimetro

Attenzione alla risoluzione

Per ottenere la risoluzione di 10^{-7} occorre contare $m \geq 10^7$, ma il Timer 1 ha un contatore a 16 bit (conteggio massimo 65535).

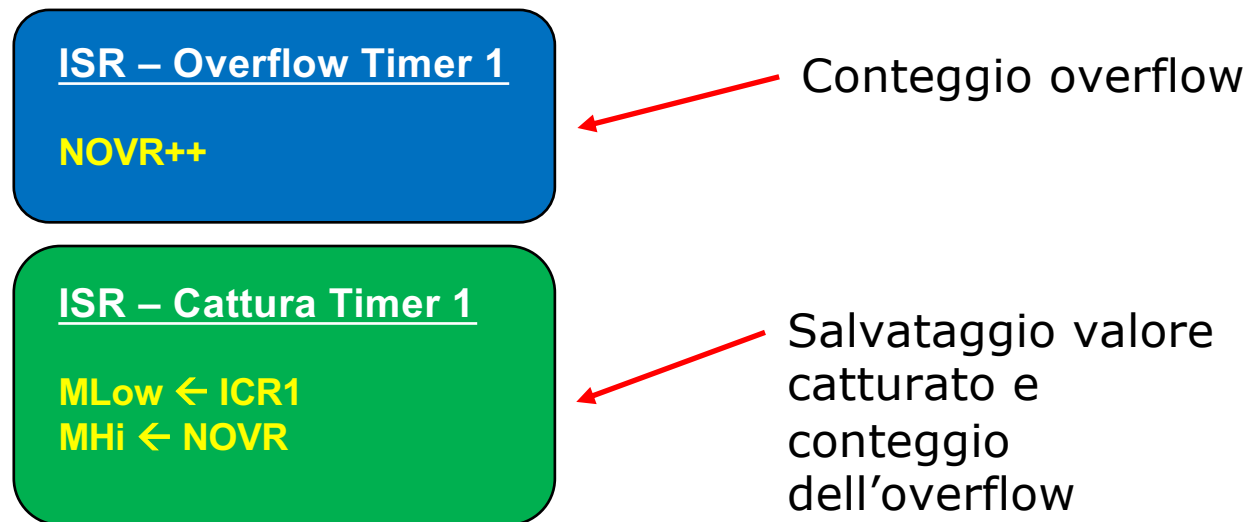


Il solo contenuto del Timer 1 non basta; **occorre contare anche gli overflow del Timer 1 (in software)**.

→ Con tempo di gate di 1 s basta un contatore complessivo da 24 bit (max 16.7 M), ma per supportare tempi più lunghi meglio usare un contatore complessivo di 32 bit

Frequenzimetro

Architettura del programma – Conteggio a 32 bit



ICR1: registro a 16 bit

NOVR: variabile globale tipo *unsigned short* (16 bit)

Frequenzimetro

Attenzione ai problemi di lettura non coerente

- Parte del contatore complessivo è gestito in hardware
 - ❖ i 16 bit meno significativi sono catturati dal timer
- Parte del contatore complessivo va gestito in software
 - ❖ i 16 bit più significativi sono il conteggio software degli overflow

La lettura delle due parti non può essere simultanea: le due parti possono essere cambiate dalla ISR durante la lettura.
→ Bisogna garantire la coerenza!

Frequenzimetro

Architettura del programma – Coerenza

ISR – Overflow Timer 1

NOVR++

ISR – Cattura Timer 1

```
MLow ← ICR1  
MHi ← NOVR  
If (REQ)  
{  
  ResH = MHi;  
  ResL = MLow;  
  REQ = 0;  
}
```

✓ Chiediamo alla ISR di comunicarci una lettura coerente del contatore:

- si pone 1 → REQ
- si attende: REQ == 0
- si legge ResL e ResH

Frequenzimetro

Architettura del programma – Coerenza

ISR – Overflow Timer 1

NOVR++

ISR – Cattura Timer 1

```
MLow ← ICR1  
MHi ← NOVR  
If (REQ)  
{  
  ResH = MHi;  
  ResL = MLow;  
  REQ = 0;  
}
```

✓ Chiediamo alla ISR di comunicarci una lettura coerente del contatore:

- si pone 1 → REQ
- si attende: REQ == 0
- si legge ResL e ResH


Ancora un piccolo problema:

se la cattura avviene tra l'overflow e l'esecuzione della sua ISR → **incoerenza NOVR**

Introduzione ad Arduino

Elenco interrupt supportati

Table 9-6. Reset and Interrupt Vectors in ATmega328P



Priorità crescente

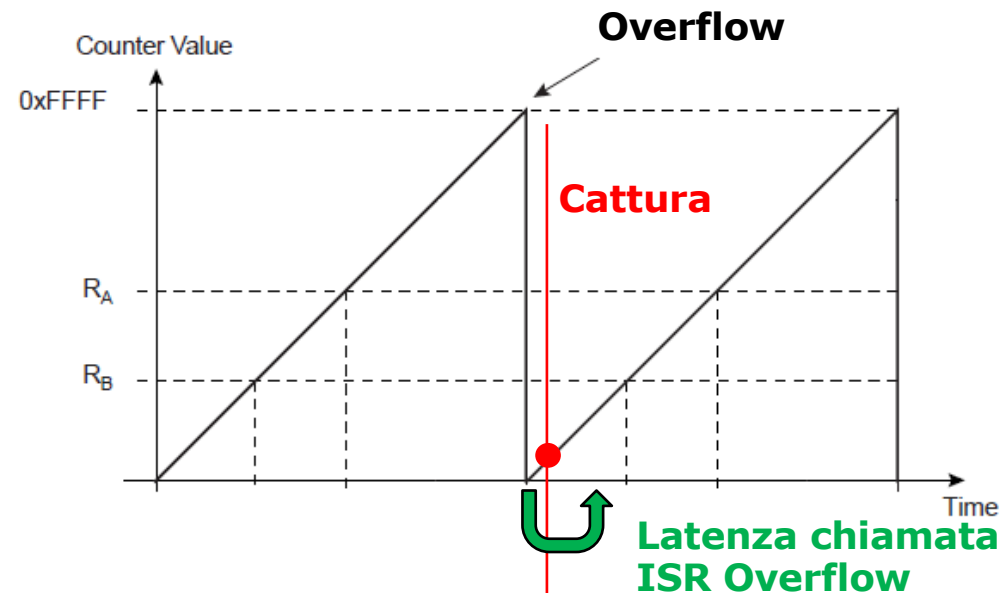
VectorNo.	Program Address ⁽²⁾	Source	Interrupt Definition
1	0x0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin 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/Counter1 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

La cattura è
prioritaria sull'
Overflow

Frequenzimetro

Architettura del programma – Coerenza

Esempio:

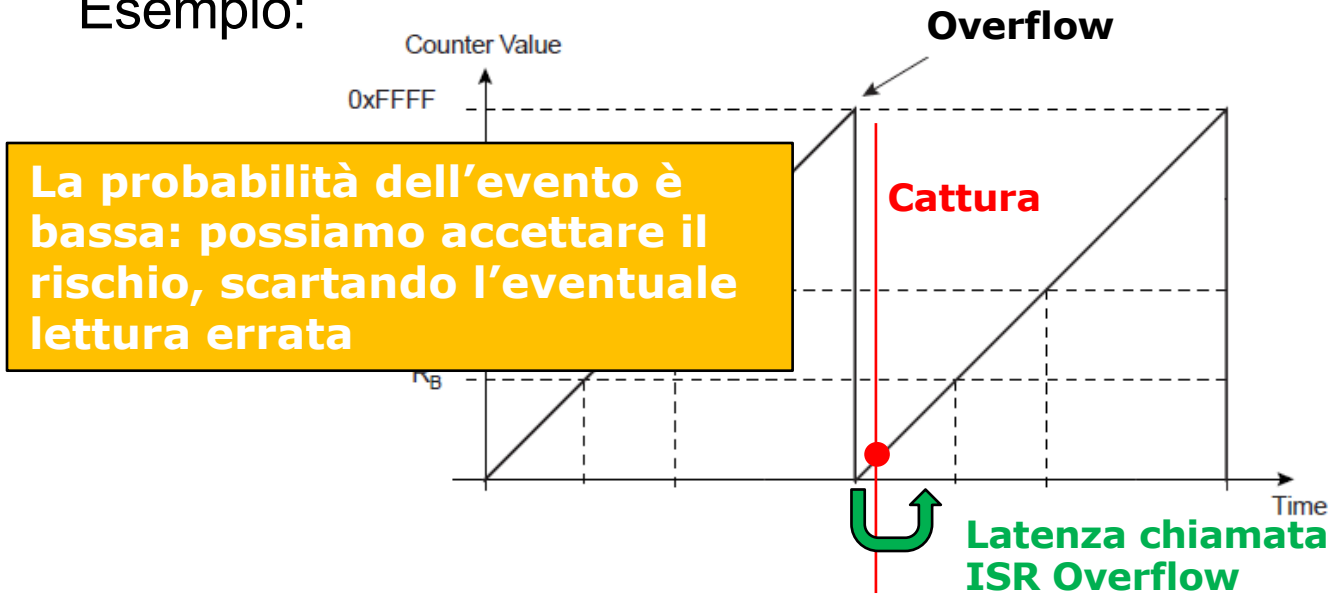


A seguito della latenza e delle priorità, partirà prima la ISR della cattura, che userà il contatore NOVR di Overflow errato

Frequenzimetro

Architettura del programma – Coerenza

Esempio:



A seguito della latenza e delle priorità, partirà prima la ISR della cattura, che userà il contatore NOVR di Overflow errato

Frequenzimetro

Architettura del programma – Coerenza

ISR – Overflow Timer 1

NOVR++

ISR – Cattura Timer 1

```
MLow ← ICR1  
MHi ← NOVR  
If (REQ)  
{  
  ResH = MHi;  
  ResL = MLow;  
  REQ = 0;  
}
```

✓ Chiediamo alla ISR di comunicarci una lettura coerente del contatore:

- si pone 1 → REQ
- si attende: REQ == 0
- si legge ResL e ResH

Ancora un piccolo problema:

se la cattura avviene tra overflow e l'esecuzione della sua ISR → **incoerenza NOVR**

Altrimenti si scarta il risultato se MLow è prossimo a zero (< ~100)

Frequenzimetro

Architettura del programma – Coerenza

Attenzione a comporre correttamente le due parti del contatore in una variabile a 32 bit:

```
unsigned long m1;  
m1 = (unsigned long) ResL + ((unsigned long) ResH << 16);
```

Frequenzimetro

Architettura del programma

ISR – Overflow Timer 1

NOVR++

ISR – Cattura Timer 1

```

MLow ← ICR1
MHi ← NOVR
If (REQ)
{
  ResH = MHi;
  ResL = MLow;
  REQ = 0;
}
    
```

Funzione loop()

```

1 → REQ
Attesa: REQ == 0
Composizione m1
    
```

```

1 → REQ
Attesa: REQ == 0
Composizione m2
    
```

Calcolo $m = m2 - m1$

Frequenzimetro

Architettura del programma – Variabili Volatile

Il compilatore cerca di ottimizzare il codice:

`while (REQ == 1) {...}`  `if (REQ == 1) while(1) {...}`

Nel caso in cui nessuna istruzione cambi la variabile REQ nel ciclo while, il compilatore ottimizza il programma leggendo REQ solo per decidere se entrare nel ciclo.

Nel nostro caso, modifichiamo REQ nella ISR, quindi dobbiamo informare il compilatore, definendo la variabile con la proprietà **volatile**, es:

```
volatile char REQ;
```

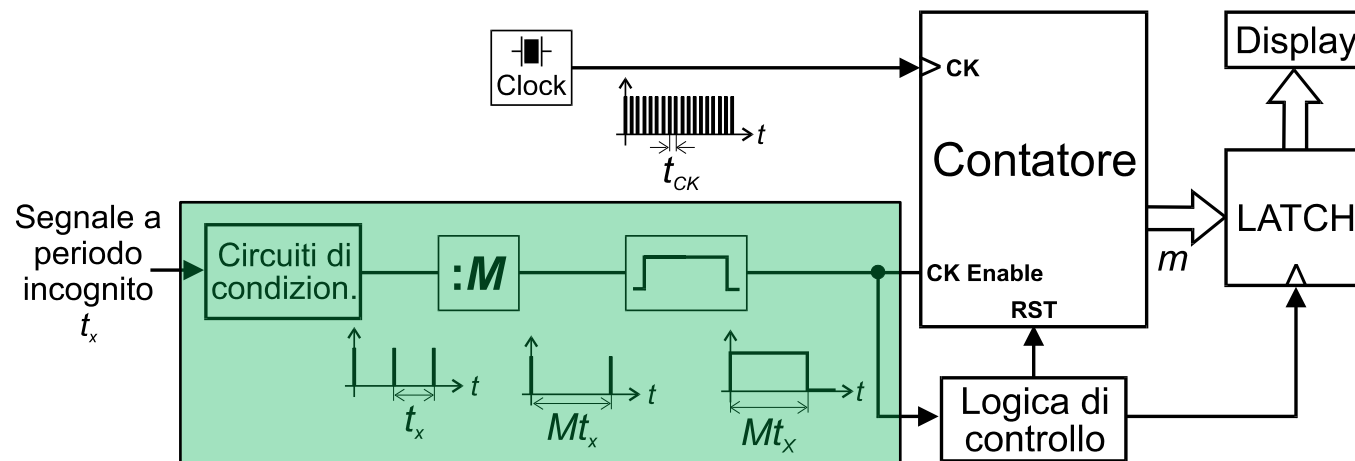
Frequenzimetro

Come realizzare l'intervallo di Gate (misura)?



Frequenzimetro

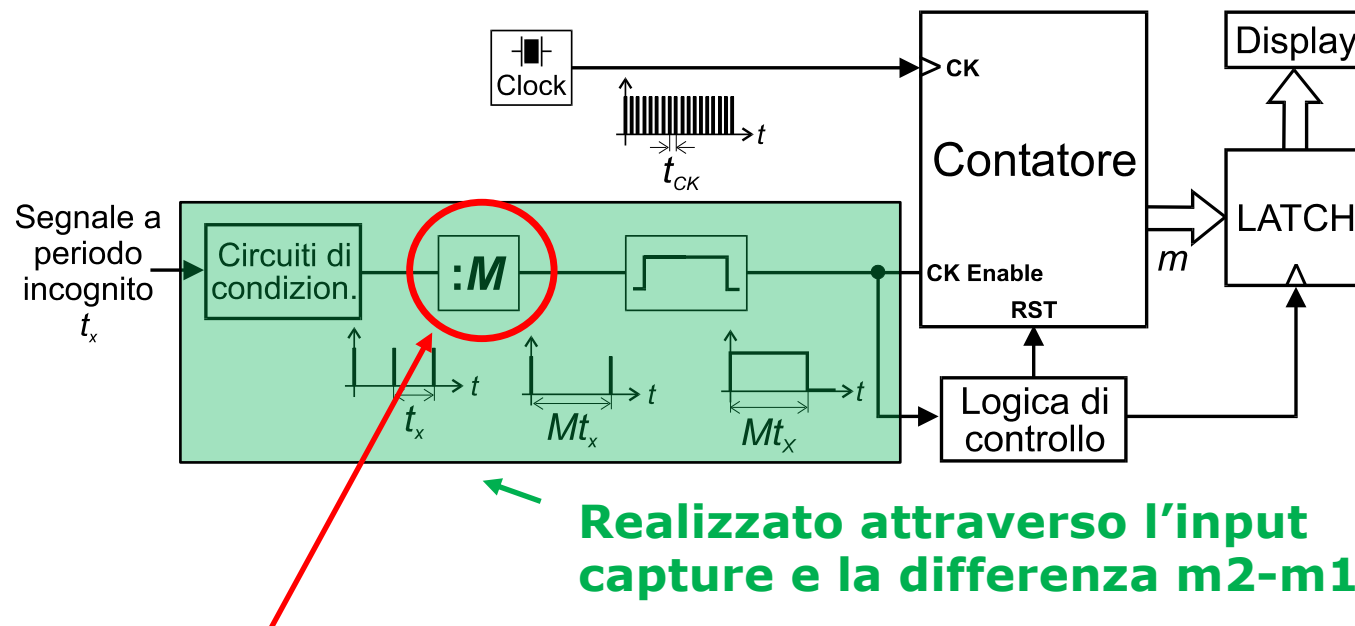
Schema operativo visto in teoria



Realizzato attraverso l'input capture e la differenza $m2-m1$

Frequenzimetro

Schema operativo visto in teoria

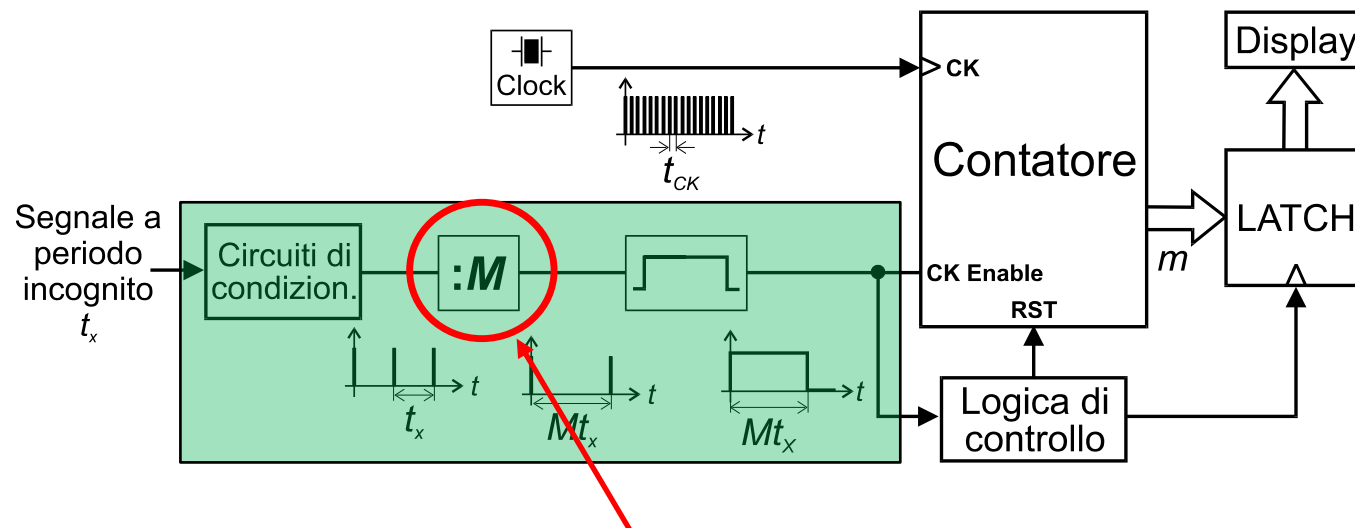


Non ancora realizzato!

$M = 1 \rightarrow$ conto solo su 1 periodo (t_x)

Frequenzimetro

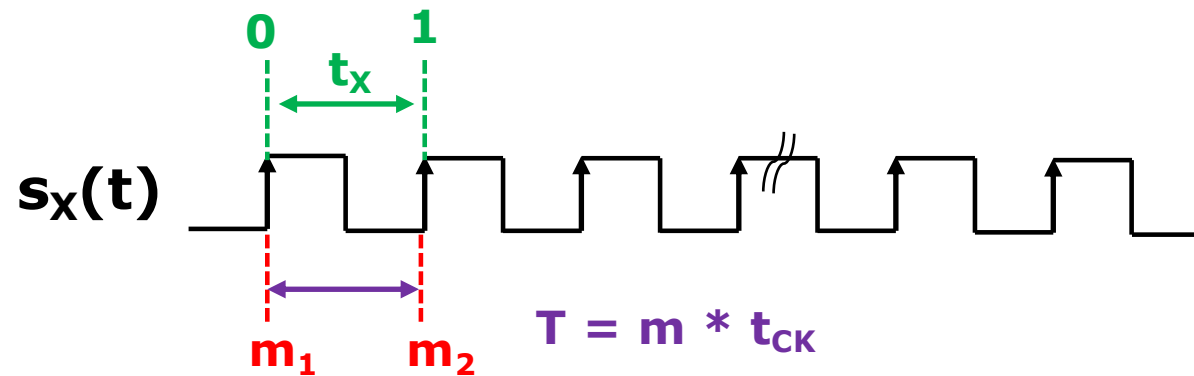
Schema operativo visto in teoria



Dalla teoria sappiamo che M serve per aumentare il tempo di Gate (misura) a piacimento in modo da contare un numero di impulsi m elevato e migliorare l'incertezza relativa di conteggio o risoluzione (conto su M periodi t_x)

Frequenzimetro

Realizzazione (senza tempo di Gate)



$$m = m_2 - m_1$$

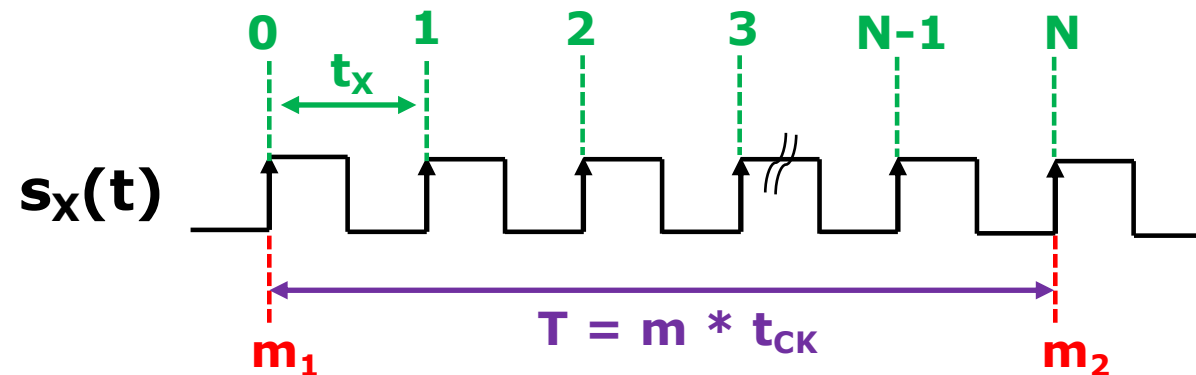
$$t_{ck} = 1/(16 \text{ MHz})$$



$$t_x = T$$

Frequenzimetro

Realizzazione



$$m = m_2 - m_1$$

$$t_{ck} = 1/(16 \text{ MHz})$$



$$t_x = T / N$$

Occorre contare i periodi, N , e fare in modo che T sia circa T_{GATE} (scelto dall'utente)

Frequenzimetro

Operazioni in virgola mobile

Arduino per risparmiare risorse utilizza sempre le operazioni tra interi, se non è obbligato a fare diversamente:

❖ Risultato inaccurato o addirittura nullo!

Occorre forzare il type *casting* con float/double delle variabili intere coinvolte:

```
double freq = (fck/ (double)m) * ( (double)N) ;
```

Frequenzimetro

Architettura del programma

ISR – Overflow Timer 1

NOVR++

ISR – Cattura Timer 1

MLow \leftarrow ICR1

MHi \leftarrow NOVR

NPCount++;

If (REQ)

{

ResH = MHi;

ResL = MLow;

ResN = NPCount;

REQ = 0;

}

Funzione loop()

1 \rightarrow REQ

Attesa: REQ == 0

Composizione m1

N1 \leftarrow ResN

Attesa Tempo di gate

1 \rightarrow REQ

Attesa: REQ == 0

Composizione m2

N2 \leftarrow ResN

Calcolo m = m2-m1

Calcolo NP = N2 – N1

Frequenzimetro

```
volatile char REQ;  
volatile unsigned short NOVR, MLow, MHi, ResH, ResL;  
volatile long NPCount, ResN;  
double fck = 16e6;  
long Tgate_ms = 1000;  
  
void setup() {  
  
    Serial.begin(57600);  
    Serial.println("Ready");  
    pinMode(8, INPUT);  
  
    TCCR1A = 0b00000000;    // Nessuna azione sui pin  
    TCCR1B = 0b00000001;    // Normal Mode, Prescaler 1:1  
  
    TIFR1 |= 32+1;    // Cancella IF cattura e IF Overflow  
    TIMSK1 |= 32+1;    // Abilita IE cattura e IE Overflow  
  
}
```

Frequenzimetro

ISR – Overflow Timer 1

NOVR++

```
ISR(TIMER1_OVF_vect)
{
    NOVR++;
}
```

ISR – Cattura Timer 1

```
MLow ← ICR1
MHi ← NOVR
NPCount++;
If (REQ)
{
    ResH = MHi;
    ResL = MLow;
    ResN = NPCount;
    REQ = 0;
}
```

```
ISR(TIMER1_CAPT_vect)
{
    MLow = ICR1;
    MHi = NOVR;
    NPCount++;
    if (REQ)
    {
        ResH = MHi;
        ResL = MLow;
        ResN = NPCount;
        REQ = 0;
    }
}
```