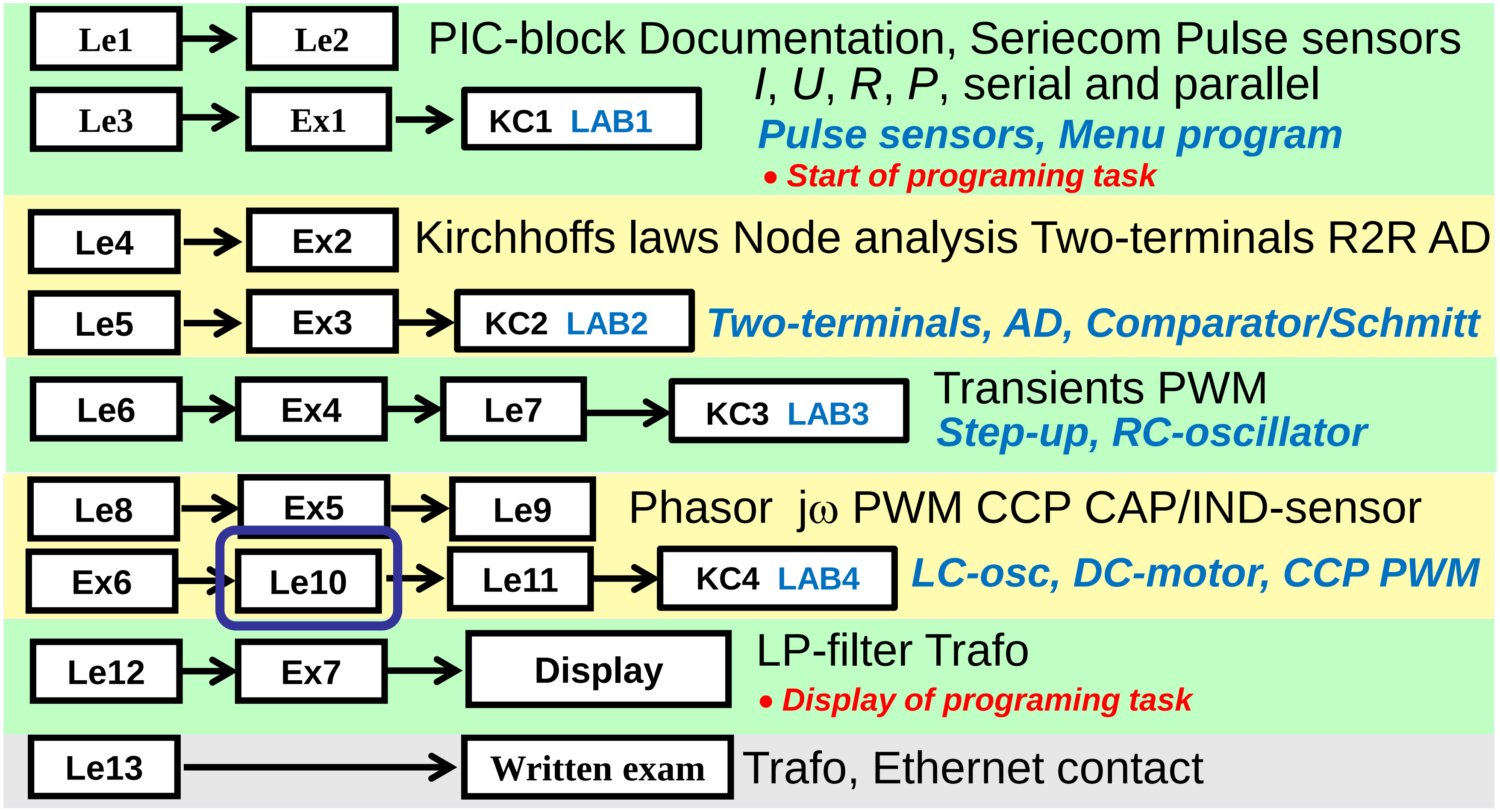
IE1206 Embedded Electronics



PIC-block Documentation, Seriecom Pulse sensors
I, *U*, *R*, *P*, serial and parallel
Pulse sensors, Menu program
• ***Start of programing task***

Kirchhoffs laws Node analysis Two-terminals R2R AD
Two-terminals, AD, Comparator/Schmitt

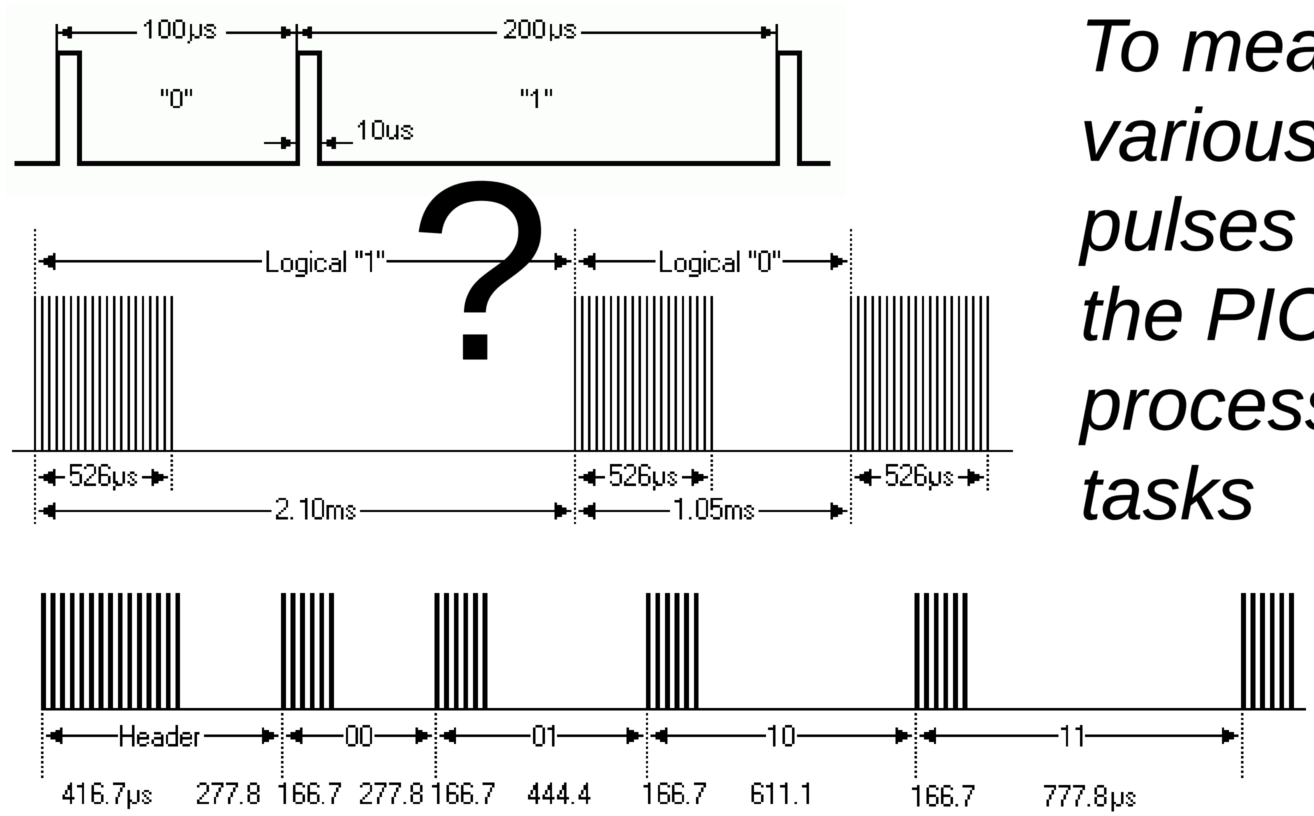
Transients PWM
Step-up, RC-oscillator

Phasor $j\omega$ PWM CCP CAP/IND-sensor
LC-osc, DC-motor, CCP PWM

LP-filter Trafo
• ***Display of programing task***

Trafo, Ethernet contact

How to measure pulses?



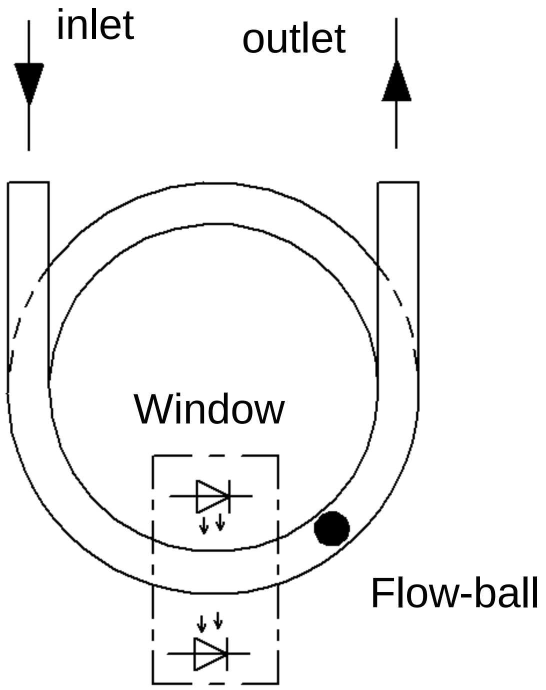
To measure various digital pulses is one of the PIC processor main tasks

• *Pulses from numerous sensors*

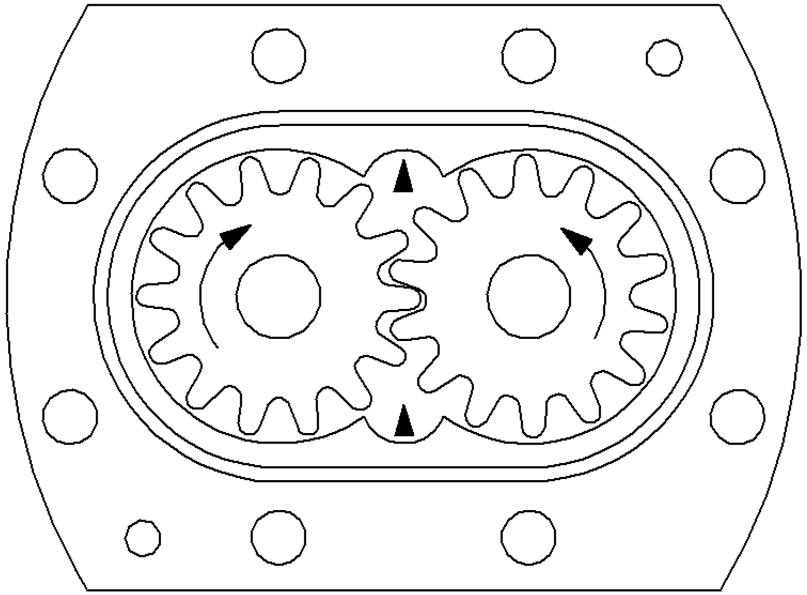
Numerous sensors have their output in the form of digital pulses: number, time, period time, frequency, duty cycle …   *Here are some examples* :

With the stream flow meter. The flow-ball followes the fluid and pass the photodiode each lap.

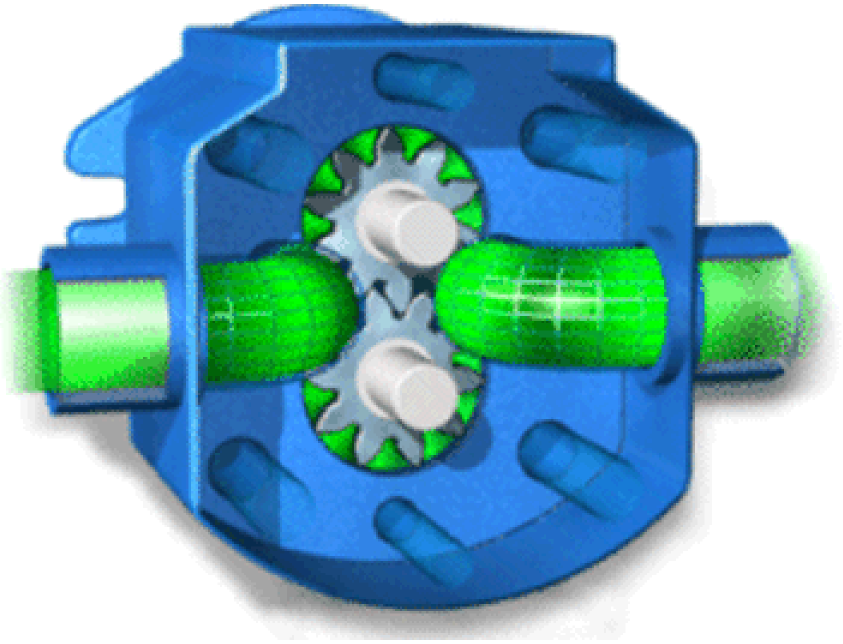
The sensor is used as fuel gauge, the number of pulses from the photo-diode are summarized as fuel consumed.



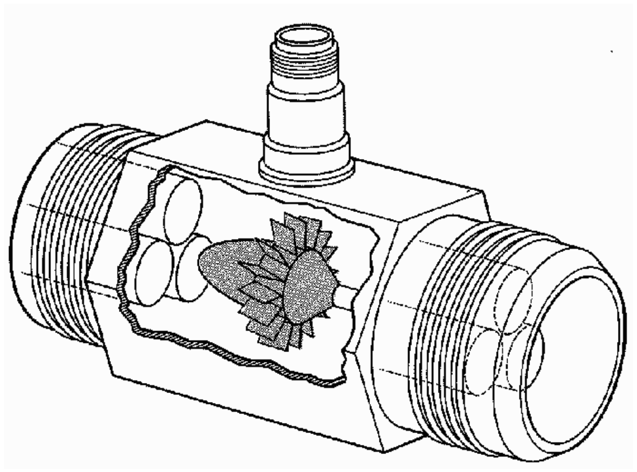
eg. Number

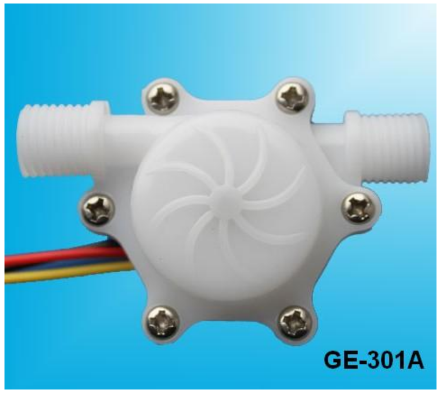


Gear meter. Fluid moves in "tooth gaps". No leaks, can measure very small amounts of liquid (the resolution is the volume of a tooth gap). Used as a fuel gauge on gasoline stations. The number of turns is a measure of liquid quantity.



Propeller and turbine Meter



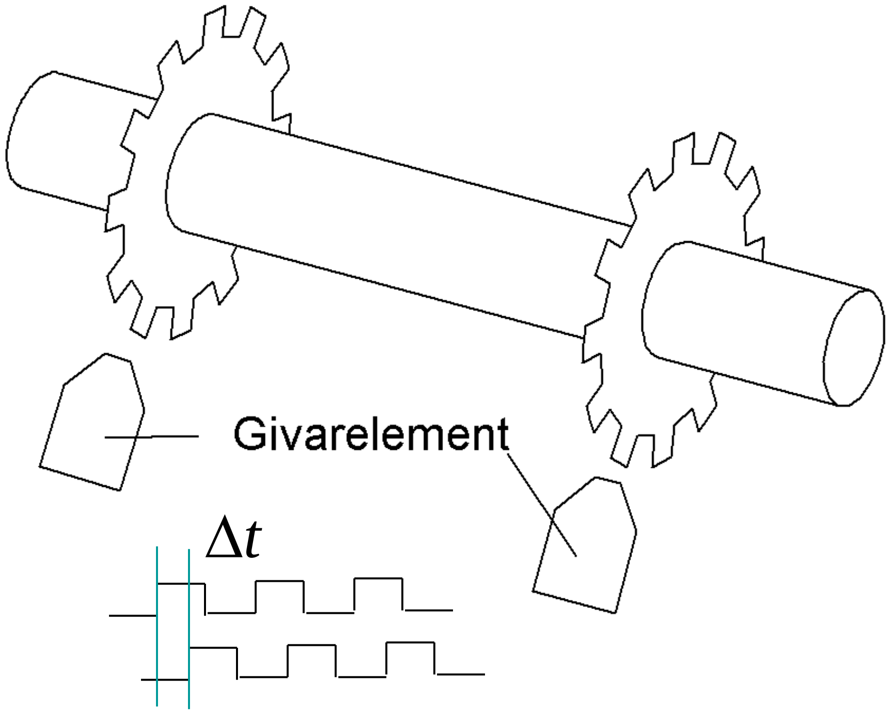


Pulse frequency is proportional to the flow rate.

eg. Pulse time

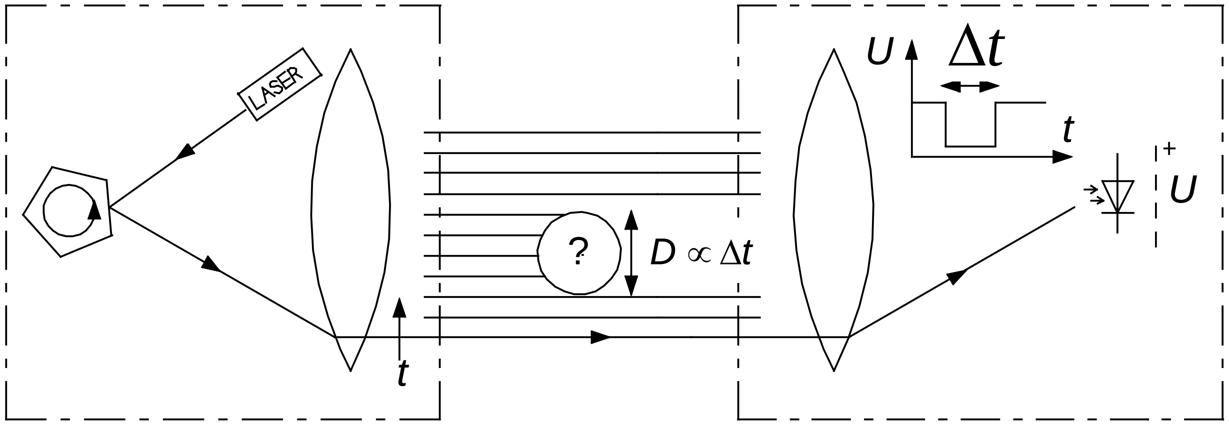
Torque meter.

When a torque is transferred with a rotating shaft, it will be sheared so that the gear wheels rotate relative to each other.
It will be an a measurable time difference between the pulses from the sensor elements, which detects teeth peaks passage.



The torque can be calculated from this time difference with knowledge of the shaft torsional stiffness.

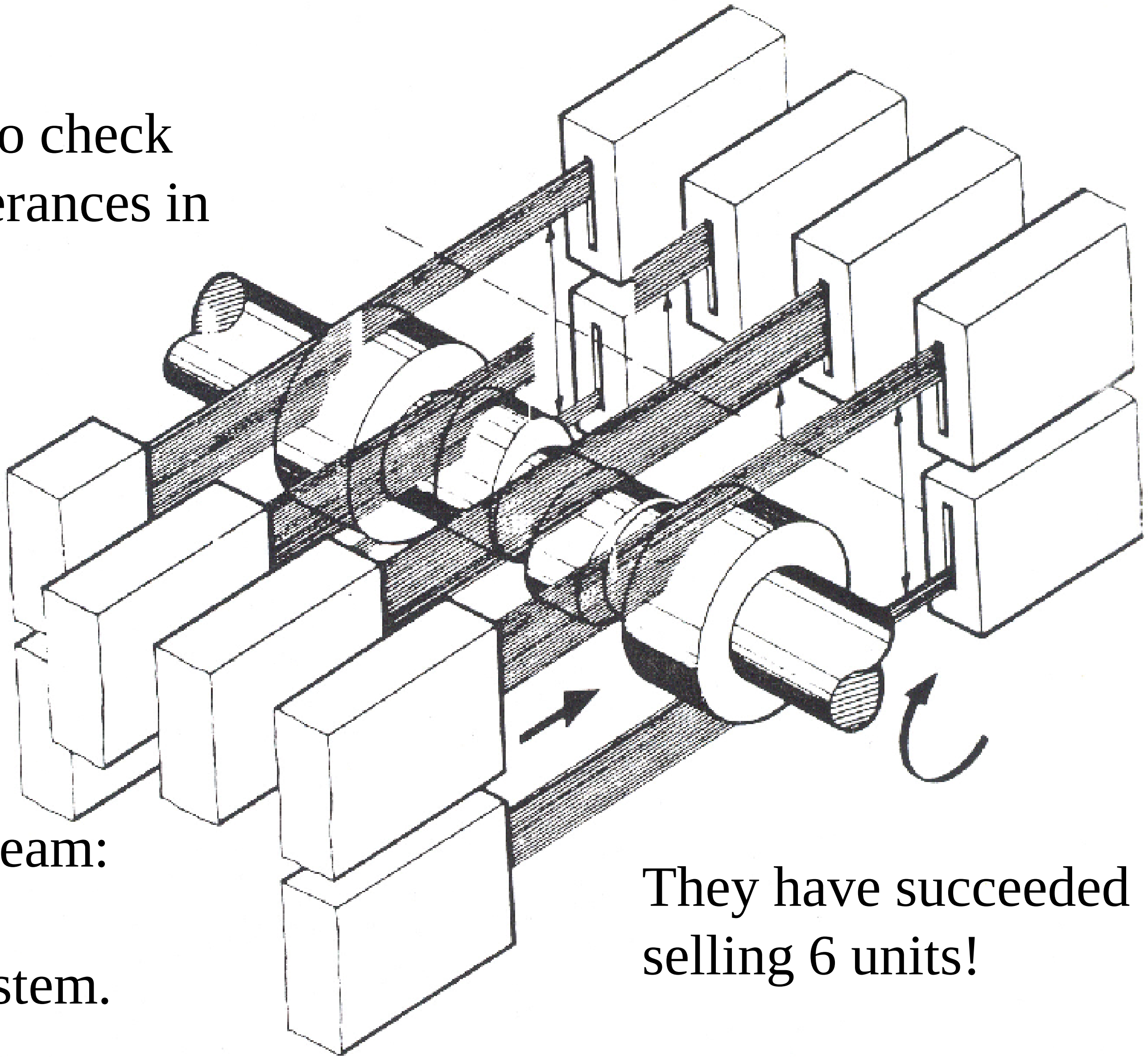
eg. Pulse time



Laser Scan Micrometer. Measured object diameter shades the laser light. A resolution of 1 μm is possible.

$$D \propto \Delta t$$

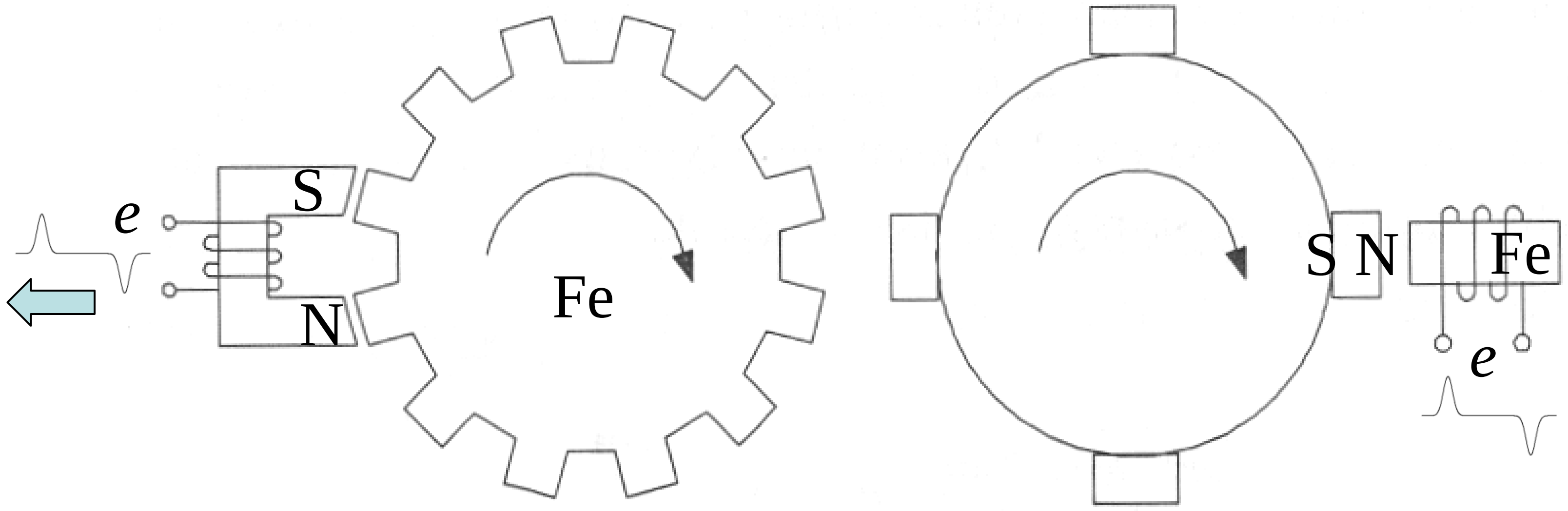
This is how to check camshaft tolerances in one turn!



Sales man's dream: Computerized Measuring System.

They have succeeded selling 6 units!

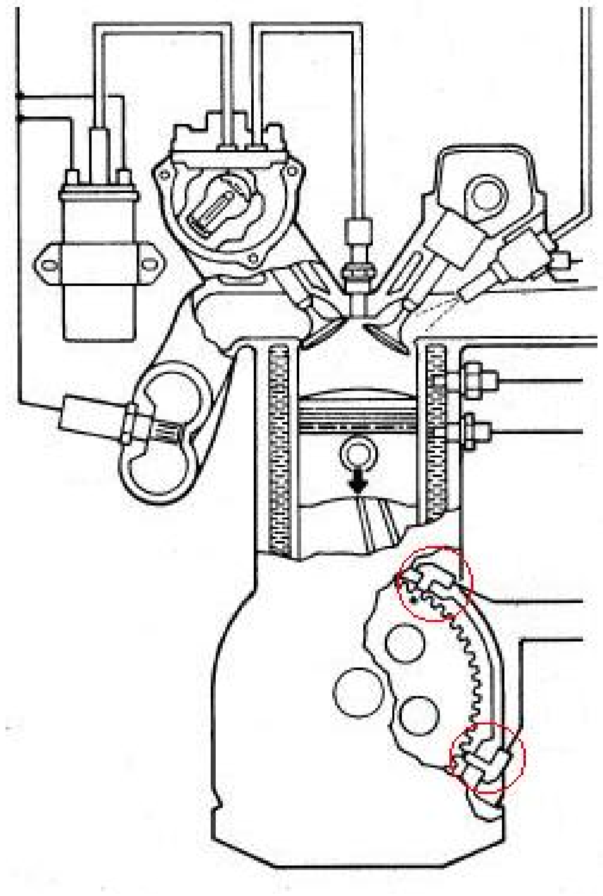
Inductive pulse sensor



There are some requirements on the magnetic properties.

$$e \propto \frac{\Delta\Phi}{\Delta t}$$

Control of the internal combustion engine

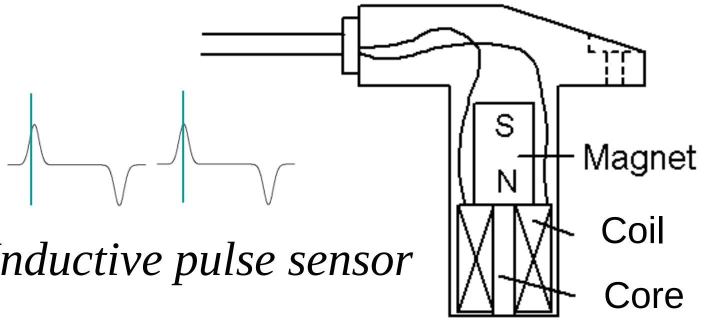


Inductive pulse sensor

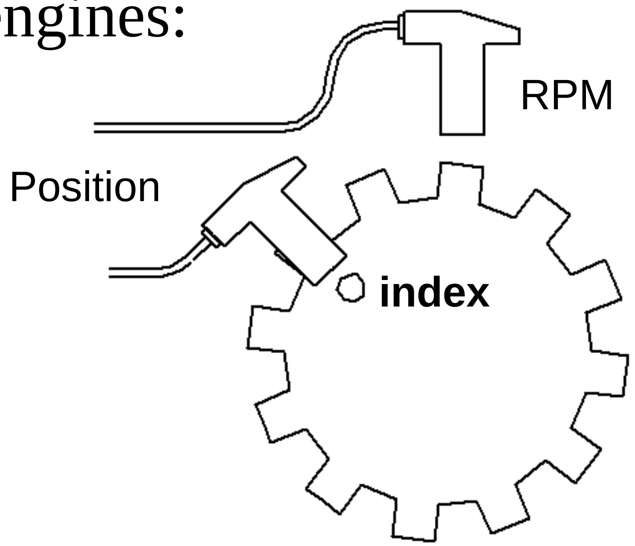
Inductive pulse sensor

eg. Pulse time, number

Passenger cars combustion engines:



Inductive pulse sensor

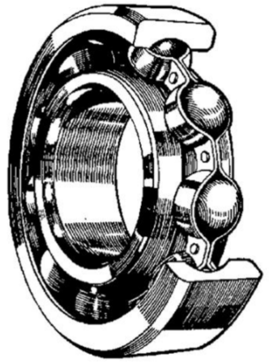


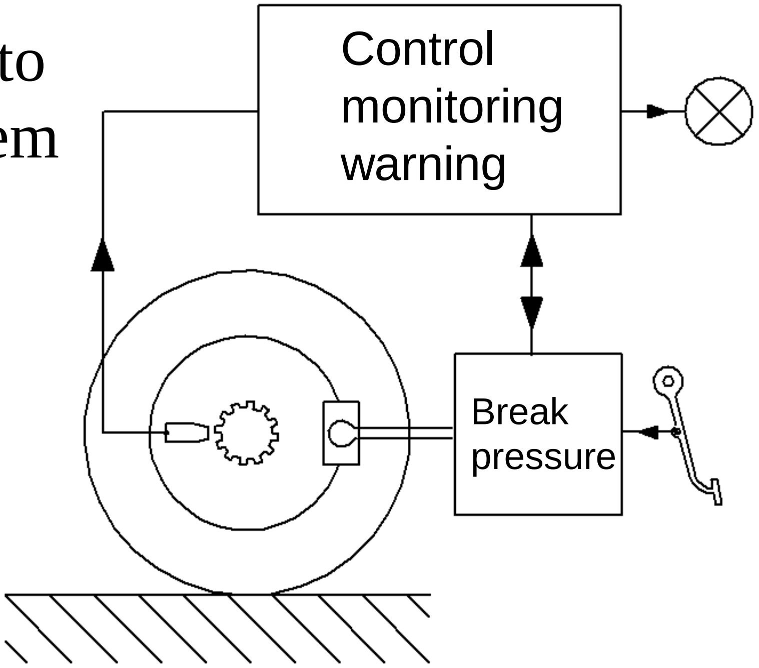
Speed and angle are measured against a gear ("starting ring gear") with an inductive pick up. The sensor produces a pulse for each tooth top. The speed. RPM, is calculated from the pulse duration between two peaks.

An "index mark" denotes the angle 0°.
(Alternatively, a cog can be "missing" at 0°).

eg. ***Low*** *pulse frequency*

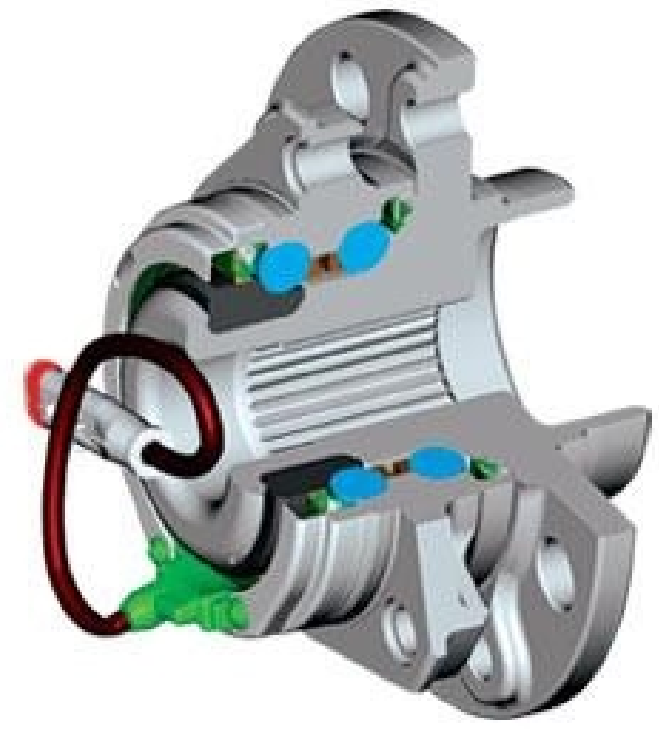
ABS brakes. When the wheel "locks up", it releases the grip to the ground. This the ABS system detects and then "reduces" the brake pressure.





An pulse sensor is integrated in the wheel bearing and gives a pulse frequency proportional to the wheel speed. "Locked" wheel is signified by low pulse rate.

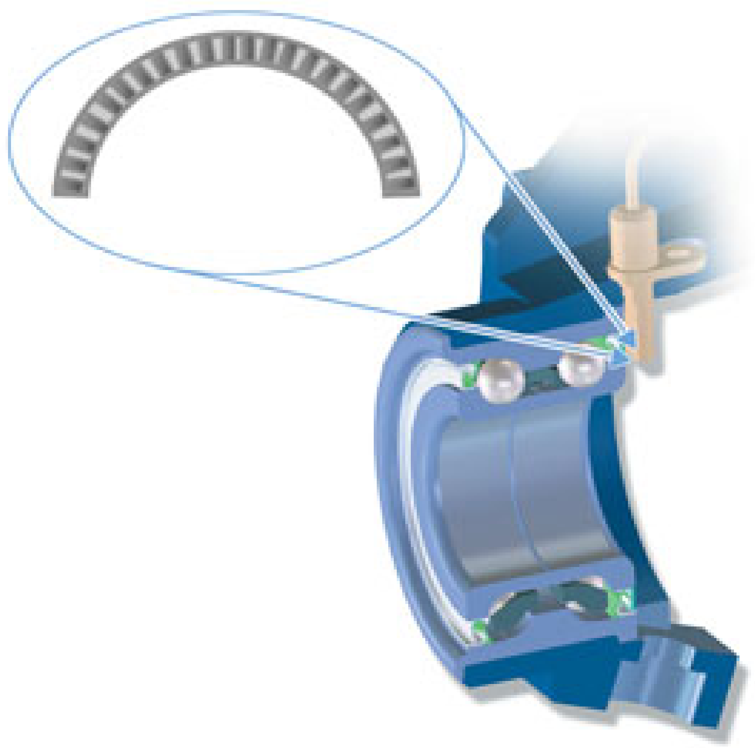
Sensors are nowadays often integrated in pure machine products

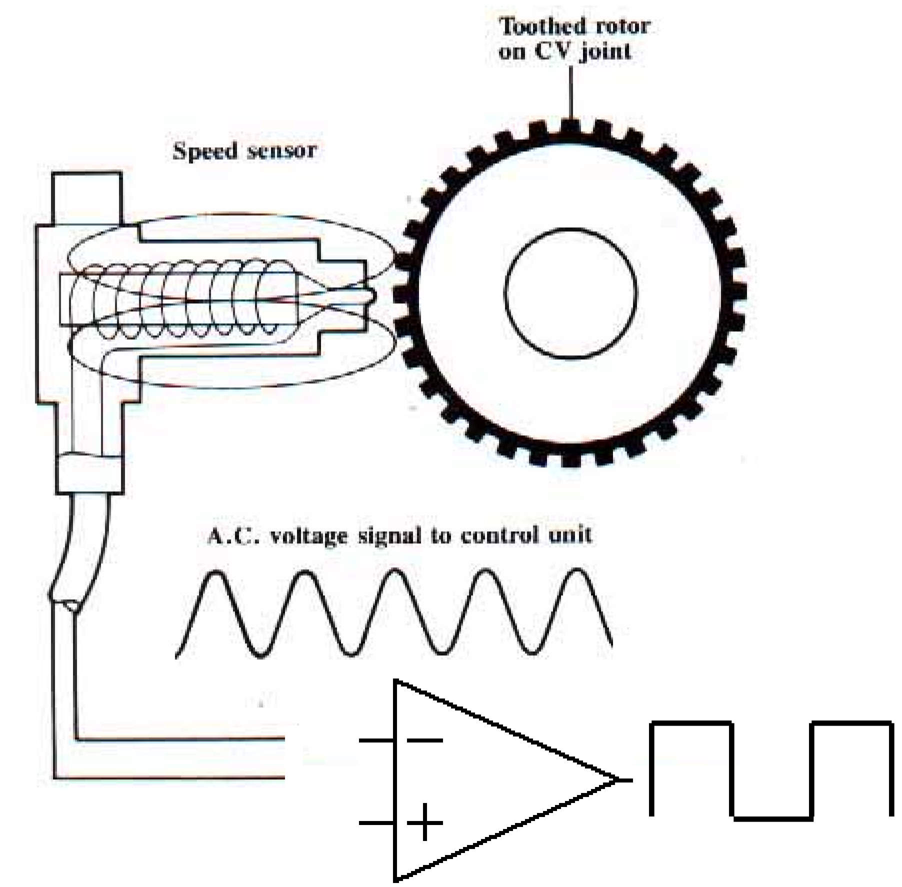


Hub bearing unit with integrated ABS sensor. SKF.

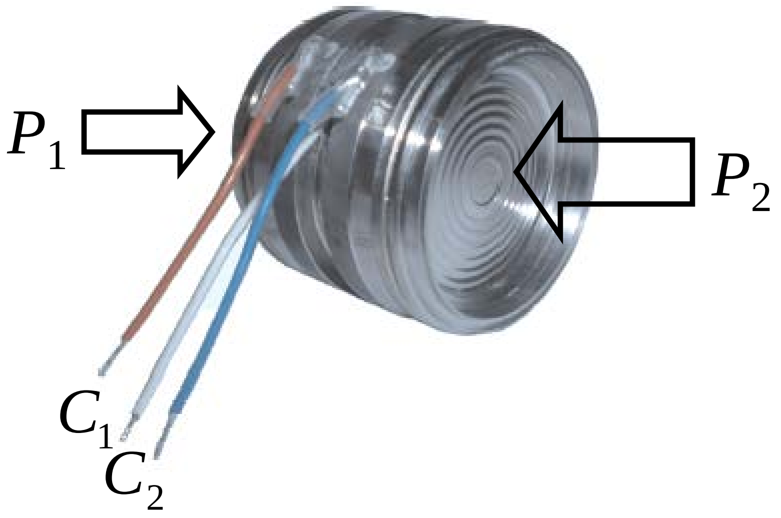
Inductive ABS-sensor (coil)

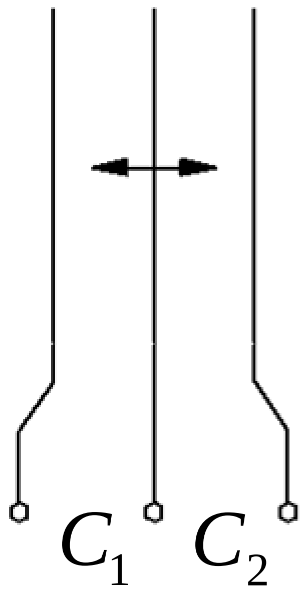
The toothed metal wheel is embodied in the ball bearing plastic seal! (eg. SKF)




Toothed rotor on CV joint
Speed sensor
A.C. voltage signal to control unit
−
+

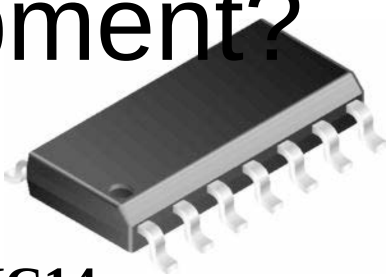
ƒ Capacitive pressure sensor





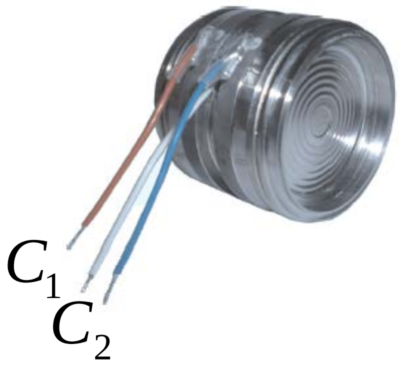
Differential capacitor for pressure difference

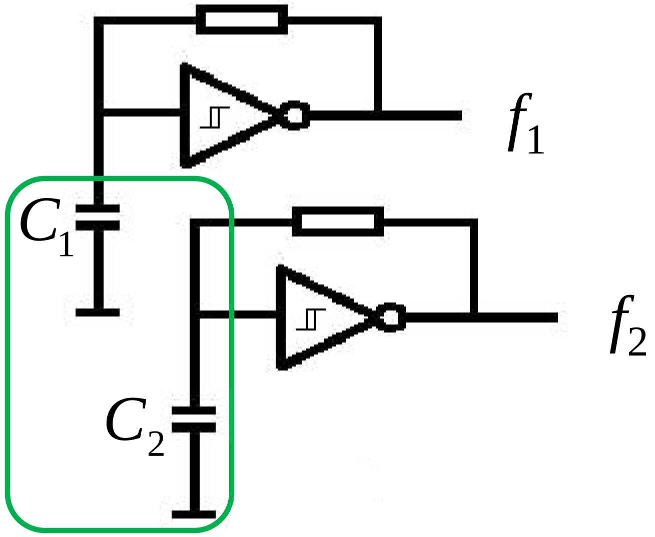
Simple measurement equipment?



74HC14

Six CMOS Schmitt-trigger inverter





$$\frac{f_1}{f_2}$$

Two oscillators are constructed close to the differential capacitor. The frequencies f_1 and f_2 are measured. By forming the ratio between the frequencies then everything that affected both frequencies equally is suppressed (= can be shortend away).

Accurate measurement of *f*

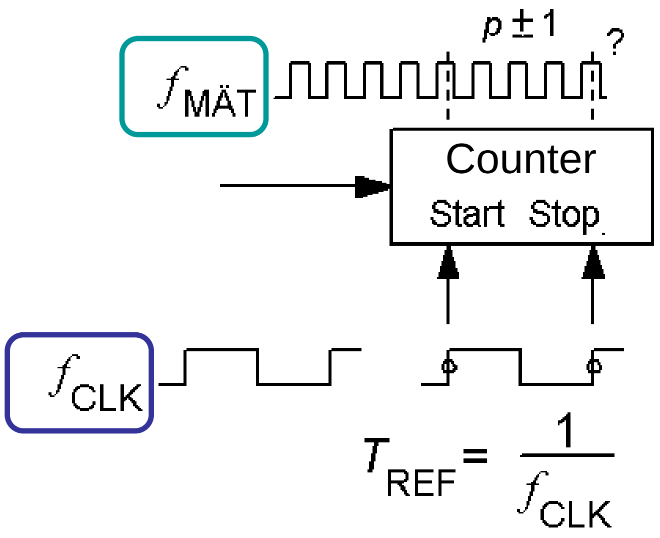
Measurement of frequency can be done very accurate. More accurate than other measurements.

The pulse sensors emit pulses of highly variable appearance and frequencies - there is not a single measurement method that can cover all the measuring case.

PIC processor has three different Timer's and a CCP device for this. The processor clock can be generated with eight different methods.

Frequency Measurement

High frequency
$f_{\text{MÄT}} > f_{\text{CLK}}$



Quantization.
The counter onlu counts complete pulses.

$$f = \frac{p \pm 1}{T_{\text{REF}}}$$

$$f = (p \pm 1) \cdot f_{\text{CLK}}$$

- **Direct frequency measurement**

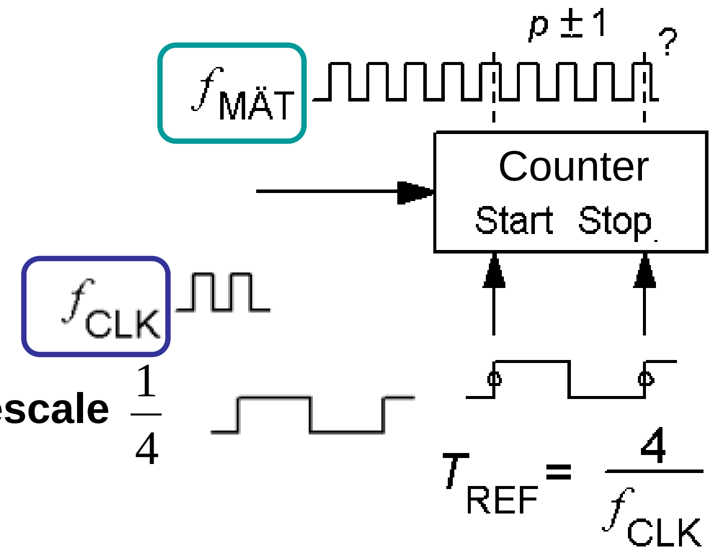
the Number of positive edges *p* under *one* period of T_{REF} is counted ($T_{\text{REF}}=1/f_{\text{CLK}}$).

High measured frequency $f_{\text{MÄT}}$ together with *long* measure time T_{REF} minimizes the impact of the quantization error.

Frequency Measurement

Lower frequency

$f_{\text{MÄT}} > \frac{1}{4} f_{\text{CLK}}$



- **Prescale** $\frac{1}{4}$

$$T_{\text{REF}} = \frac{1}{\frac{1}{4} \cdot f_{CLK}} = \frac{4}{f_{CLK}}$$

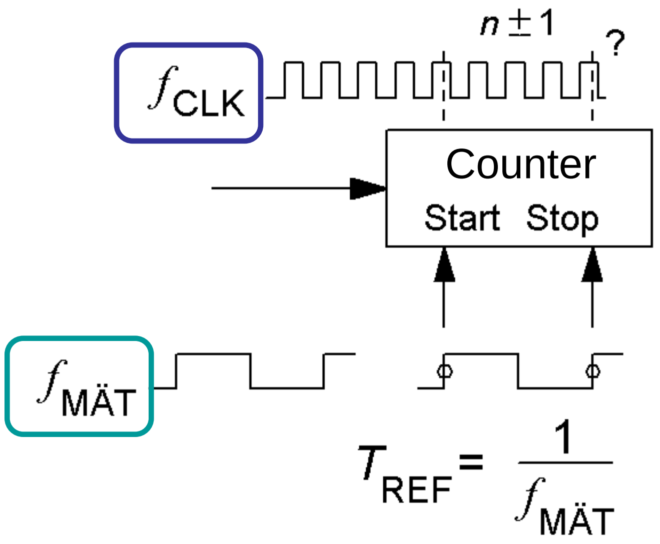
$$f = \frac{p \pm 1}{T_{\text{REF}}} = \frac{(p \pm 1) \cdot f_{\text{CLK}}}{4}$$

To measure lower frequencies requires that the measurement time is extended by dividing down the reference frequency f_{CLK} with a **prescaler**.

Period time measurement

Low frequency

$f_{\text{MÄT}} < f_{\text{CLK}}$



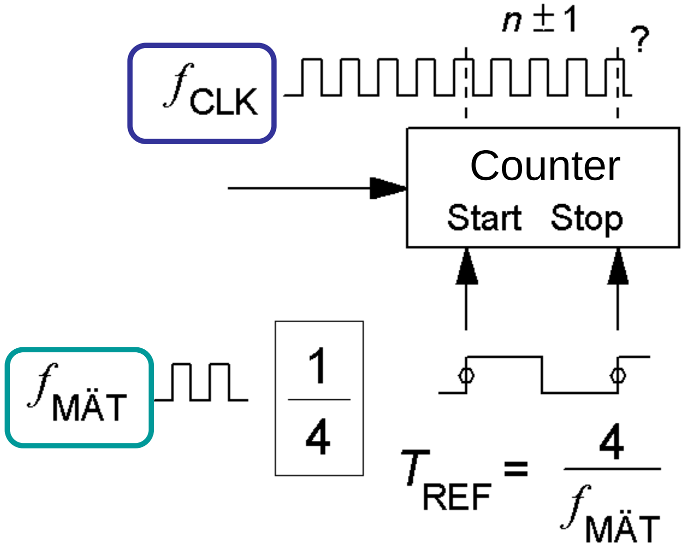
$$f = \frac{f_{\text{CLK}}}{(n \pm 1)}$$

Alternatively, when measuring low frequencies one can do this indirectly by **measuring the period time**. The measurement frequency is obtained by mathematically invert the count.

During a period of the signal *n* clock pulses are counted.

Multiperiod time measurement

Higher frequency
$\frac{1}{4} f_{\text{MÄT}} < f_{\text{CLK}}$



$$f = k \cdot \frac{f_{\text{CLK}}}{(n \pm 1)}$$

Higher frequencies can be measured with **multiperiod time measurement**. The measured signal frequency is then divided down by a factor *k* before measurement (register only every 4 or every 16 of the edges).

- PIC processor is prepared for all these different measurement methods. (And many more …)

Clock frequency accuracy

In addition to quantization, ie counting only the whole pulses, one will always have a relative error which is equal to the reference frequency error.

Eg. Wrist watch requires crystal. Crystals have typical error Δf $\pm$ 20 ppM (parts per million). $f = 4$ MHz $\pm$ 80 Hz.

Wishes: clock may not lose more than 10 sek/month.
10s/(30[days]·24[hr]·60[min]·60[sec]) = 25 ppM.

Clock frequency accuracy

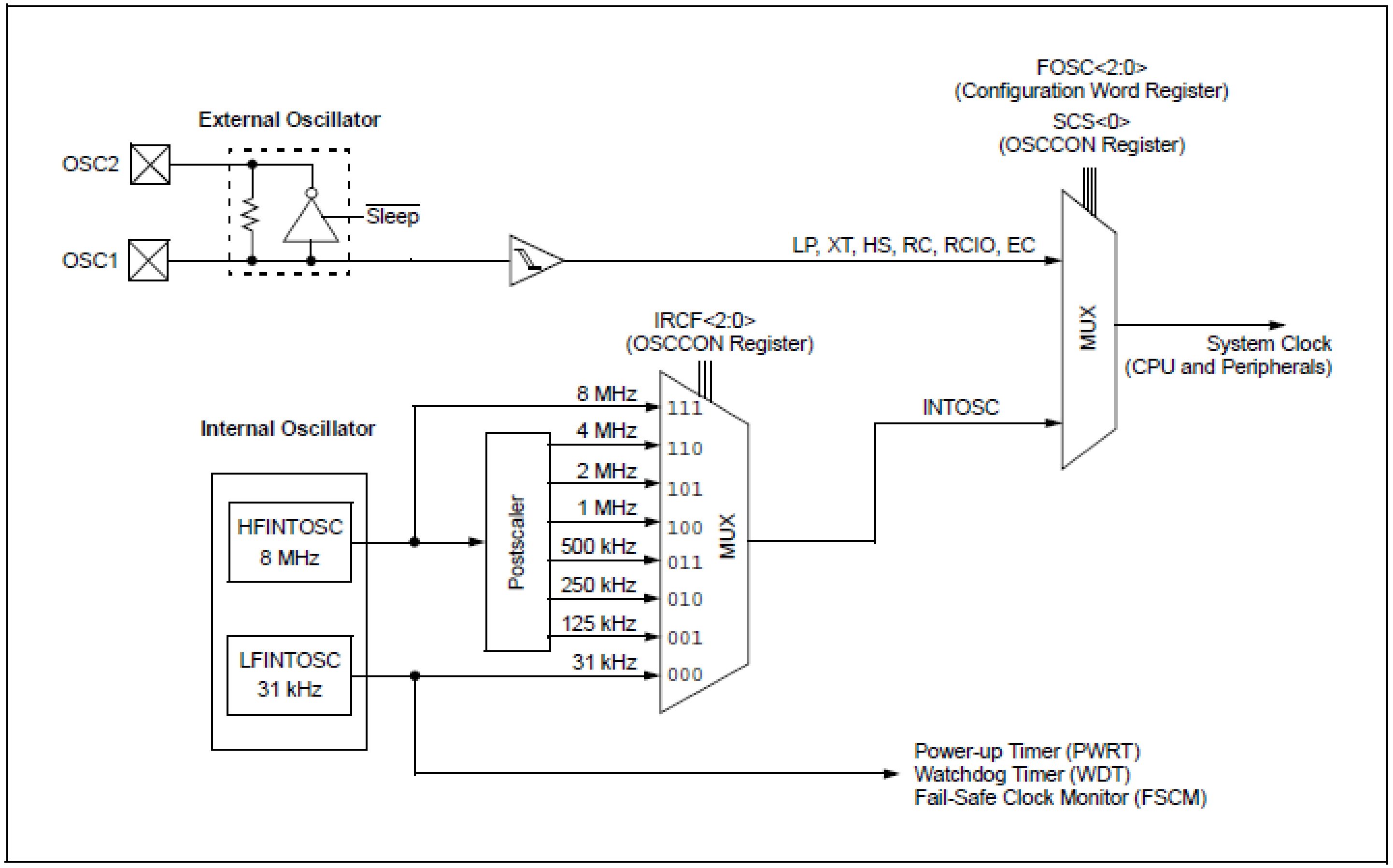
Eg. Stopwatch to use at a 800m race.
(2 minutes total measurement time is probably enough)
Wishes: resolution 0.01 sec.
1/(2[min]·60[sek]·100) = 1 ‰.

A RC-oscillator has typical a 5% error, if untrimmed.
(R 1%, but C seldom better than 5%)

PIC16F690-processor internal RC-oscillator is **factory trimmed** to ±1%.
Dthis is not enough … but perhaps we can finetune!

PIC-processor clock module

FIGURE 3-1: SIMPLIFIED PIC® MCU CLOCK SOURCE BLOCK DIAGRAM

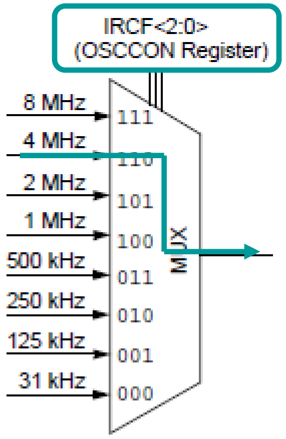


PIC-processor clock module

REGISTER 3-1: OSCCON: OSCILLATOR CONTROL REGISTER

U-0	R/W-1	R/W-1	R/W-0	R-1	R-0	R-0	R/W-0
—	IRCF2	IRCF1	IRCF0	OSTS[1]	HTS	LTS	SCS
bit 7							bit 0

- At lab we use the default setting, 4 MHz – that makes it easy to calculate the execution time.



REGISTER 3-2: OSCTUNE: OSCILLATOR TUNING REGISTER

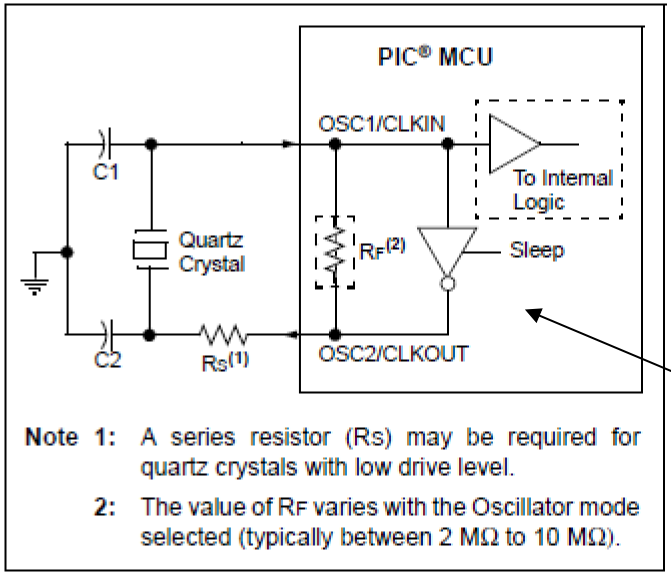
U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	TUN4	TUN3	TUN2	TUN1	TUN0
bit 7							bit 0

10000(min) – **0**0000 (fabrikstrim) – **0**1111(max)

- If you are able to "tune yourself" so can the factory tuned frequency be adjusted in ±16 small steps to ≈ ±0,5‰ . *Now enough for the stopwatch!*

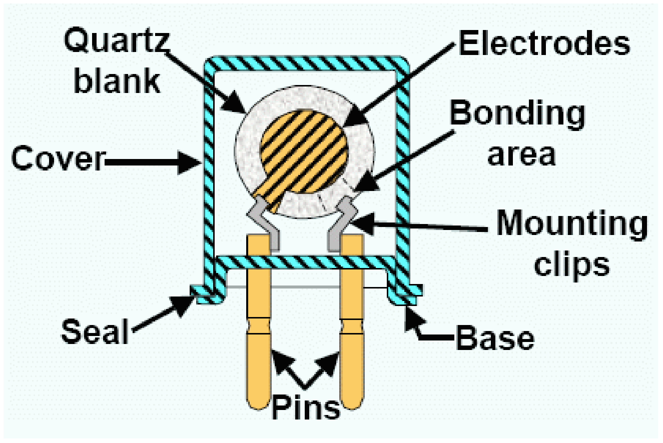
External crystal

FIGURE 3-3: QUARTZ CRYSTAL OPERATION (LP, XT OR HS MODE)



Note 1: A series resistor (RS) may be required for quartz crystals with low drive level.

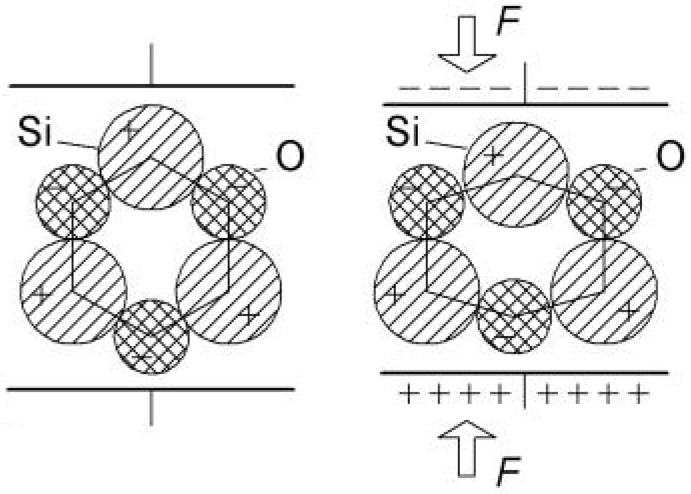
2: The value of RF varies with the Oscillator mode selected (typically between 2 MΩ to 10 MΩ).

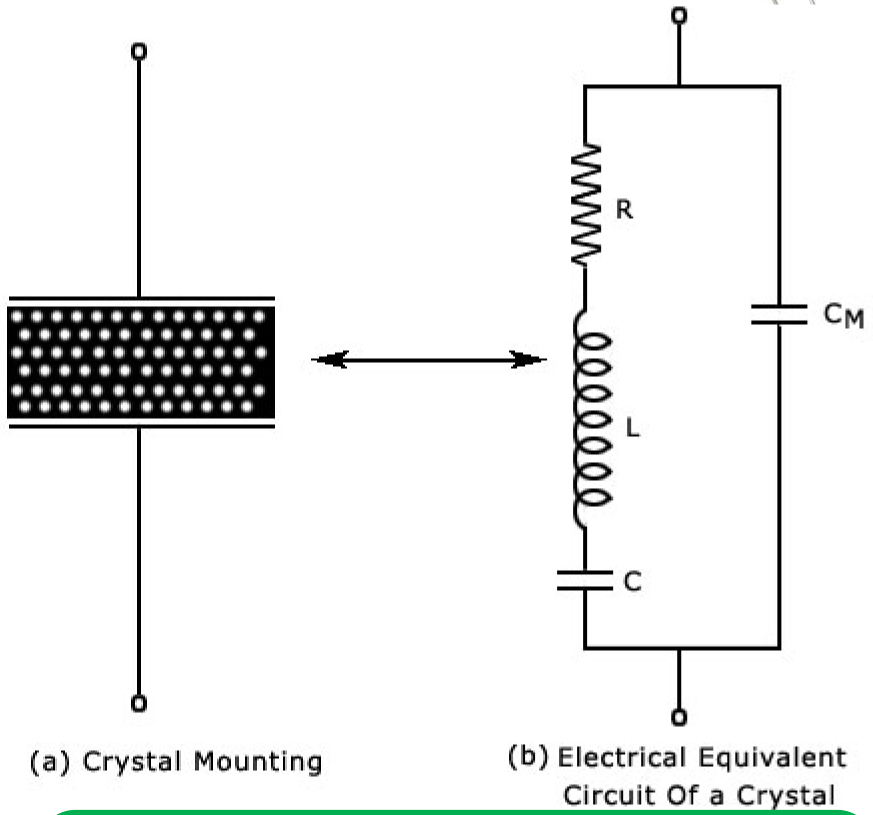


Same kind of circuit as in the course LC-oscillator lab.

PIC processors can use external crystal. C1 and C2 can be omitted on the breadboard, but they are necessary on a PCB.

Piezoelectric crystal

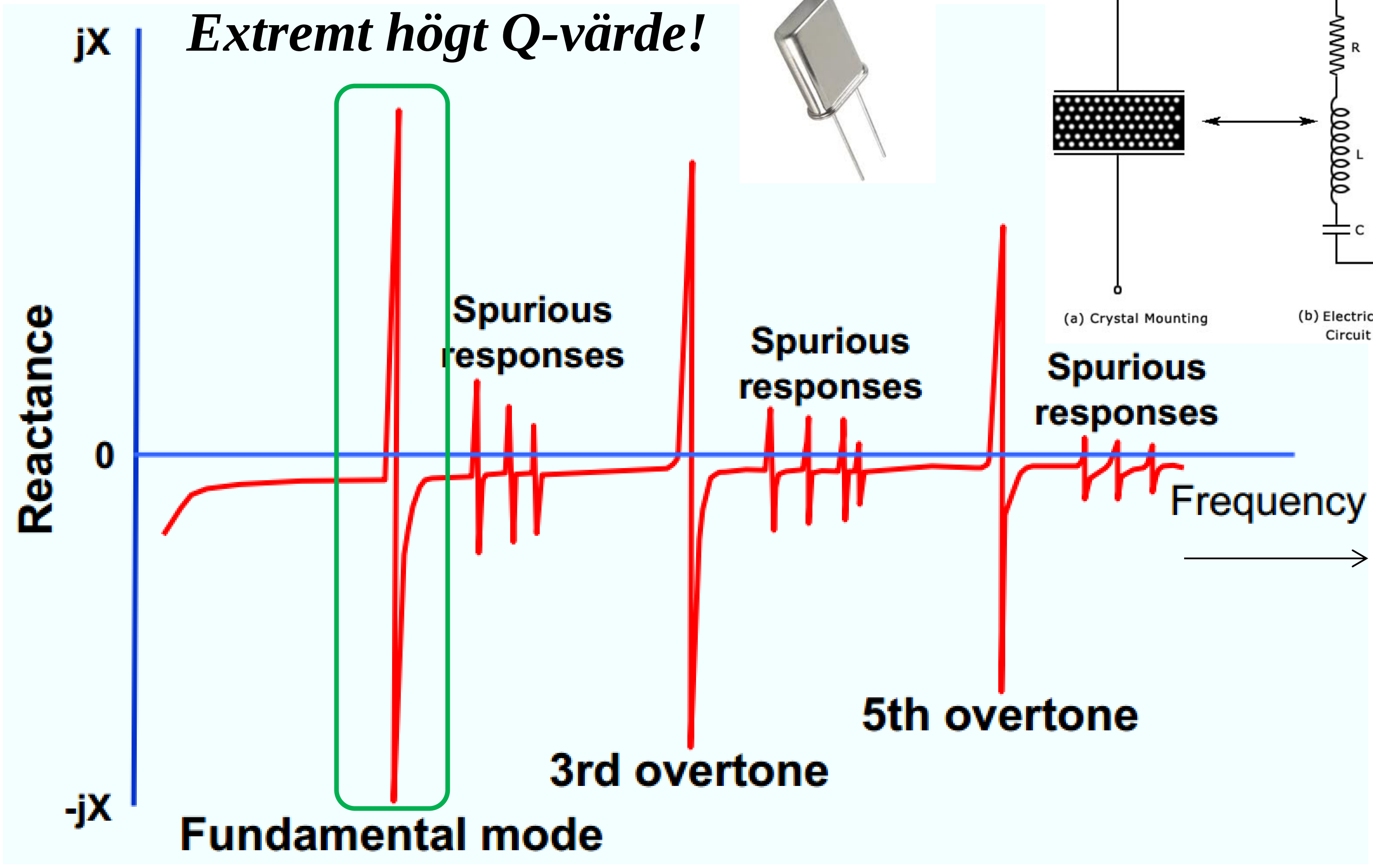


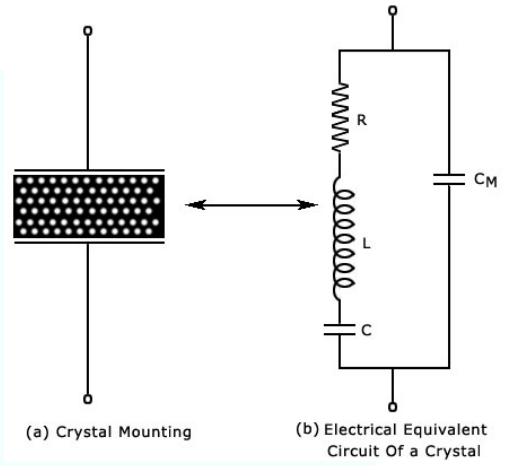


Add current (charge) to a "quartz crystal" and it is compressed, then when it "springs" back it will suply the current.

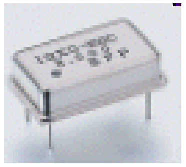
- Electric, a crystal can be compared to a resonant circuit - with extremely high *Q* value.

Piezoelectric crystal





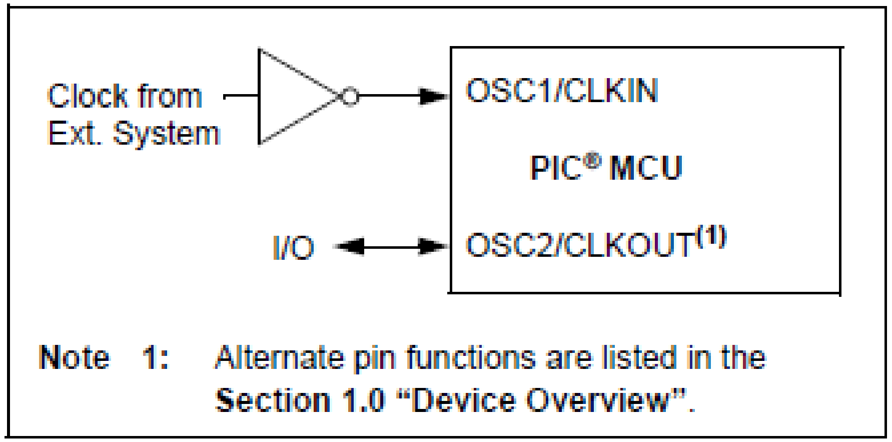
External clock signal



PIC processors can use the external clock frequency signal.

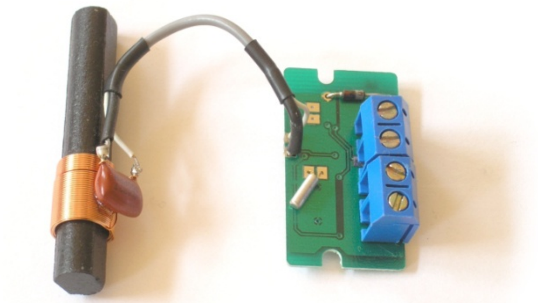
If you have access to an exact frequency then the PIC processor to can be as accurate. (The picture shows such an external clock module, oscillator and crystal "all in one").

FIGURE 3-2: EXTERNAL CLOCK (EC) MODE OPERATION


Clock from Ext. System
OSC1/CLKIN
PIC® MCU
I/O
OSC2/CLKOUT(1)
Note 1: Alternate pin functions are listed in the Section 1.0 "Device Overview".

Atomic clock?

Radio Controlled Watches, from eg. Claes Ohlsson & co, are locked to an atomic standard in germany.
So it can actually be possible to get extremely accurate reference frequency to low price!



Such a clock module gives a pulse per second (excluding sec No 60). A so-called PPS signal.

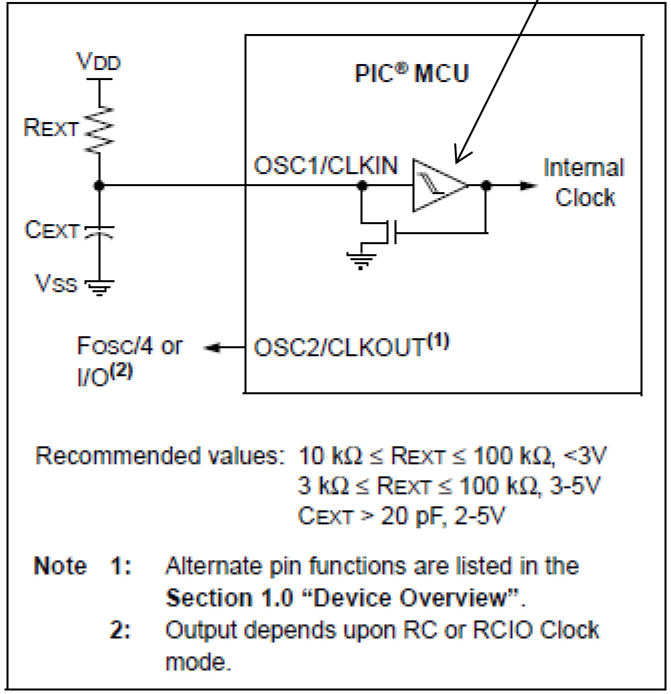
Low clock frequency *RC*

When the frequency accuracy is not that important
– external RC-circuit.

Data acquisition of one measurement per day does not require high clock frequencies. You can then change/increase the clock frequency of the program when the processor will report back!

Schmitt-trigger

FIGURE 3-5: EXTERNAL RC MODES



Recommended values: 10 kΩ ≤ REXT ≤ 100 kΩ, <3V
3 kΩ ≤ REXT ≤ 100 kΩ, 3-5V
CEXT > 20 pF, 2-5V

Note 1: Alternate pin functions are listed in the **Section 1.0 "Device Overview"**.
2: Output depends upon RC or RCIO Clock mode.

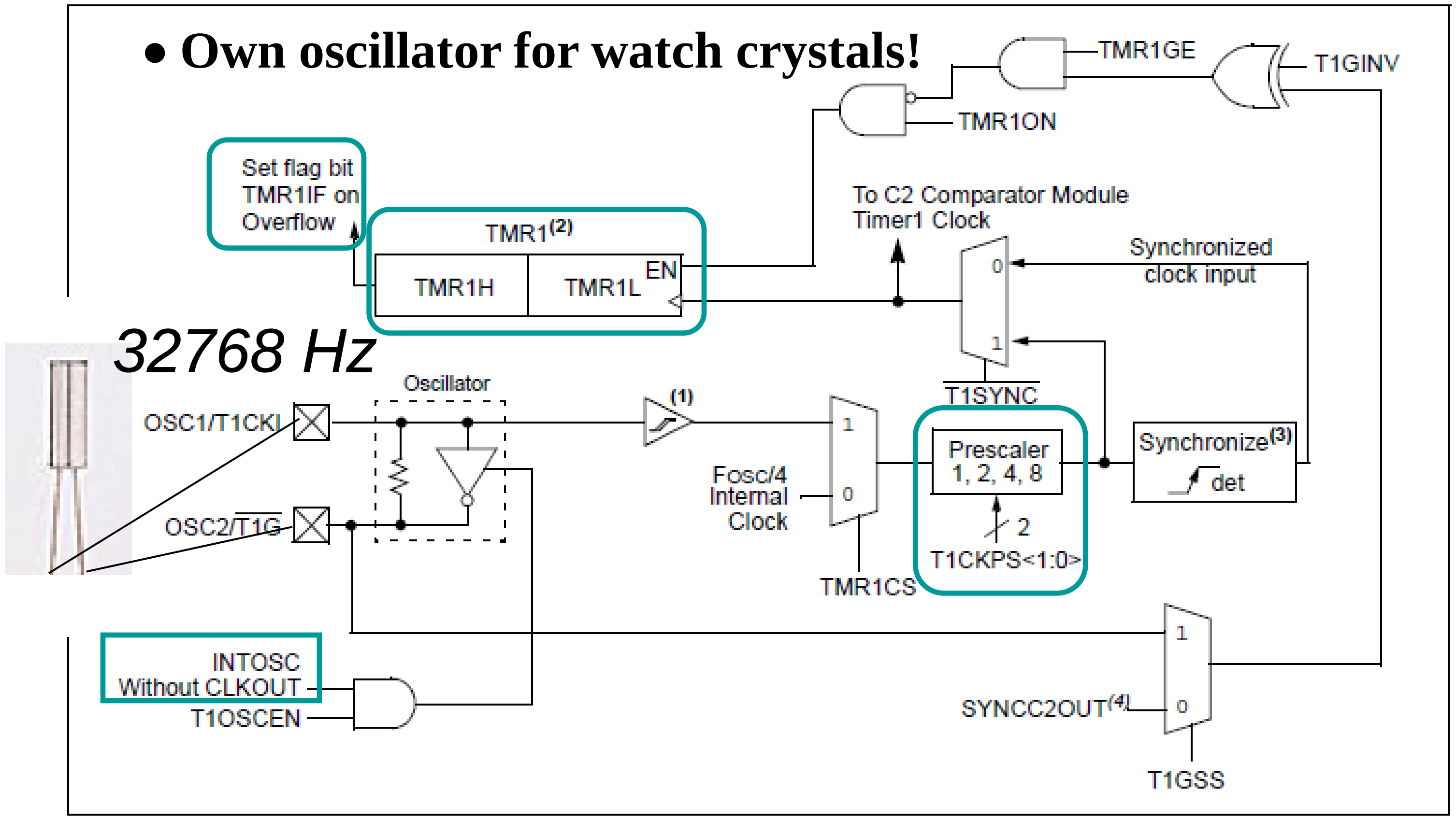
- The lower the clock speed, the lower current consumption, and less risk that the PIC processor emits interferences.

PIC 16F690 Timer1

FIGURE 6-1: TIMER1 BLOCK DIAGRAM

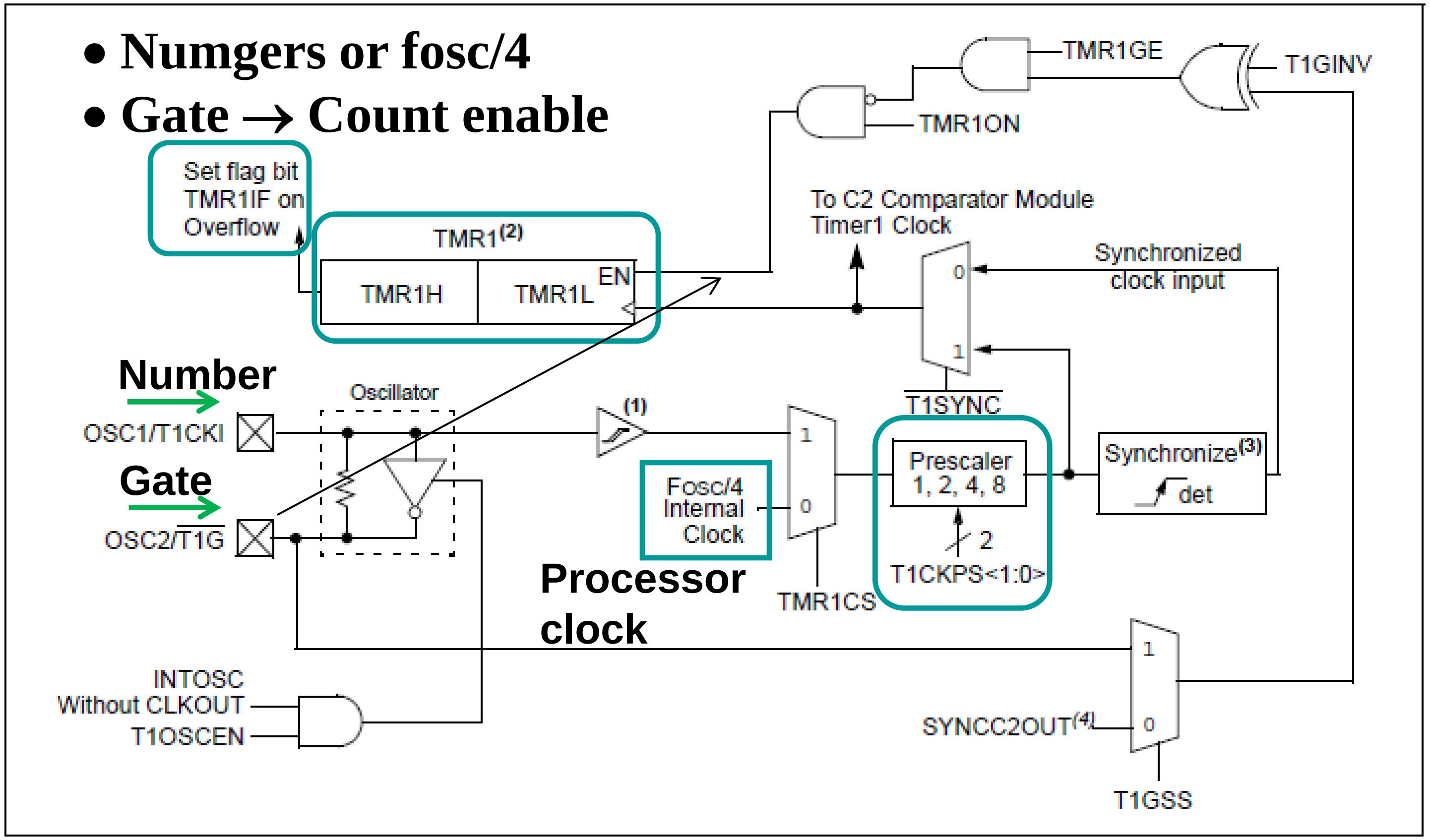
- **Own oscillator for watch crystals!**

32768 Hz



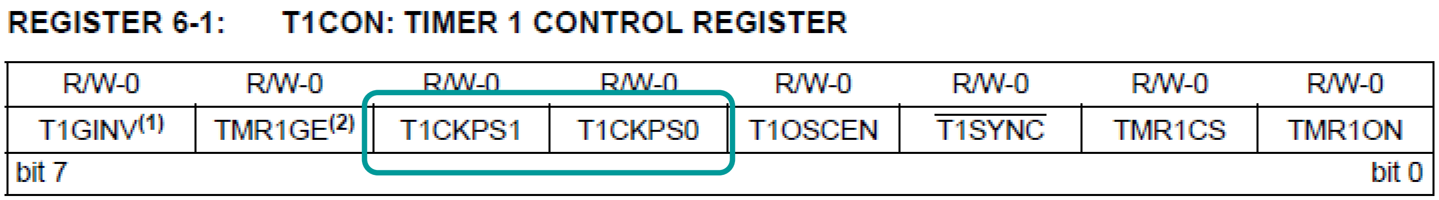
PIC 16F690 Timer1

FIGURE 6-1: TIMER1 BLOCK DIAGRAM



PIC 16F690 Timer1

Timer 1 is a 16-bit timer/counter. You reach it through two 8-bit registers **`TMR1H`** and **`TMR1L`**. A flag **`TMR1IF`** will be set if the timer overflows. Must be reset if you want to know if this happens again.
Timer1 can use its own oscillator – for a 32768 Hz watch crystal, or it could use the processor cloch. Timer 1 has then a **Prescaler** for {1:1,1:2,1:4,1:8}.



REGISTER 6-1: T1CON: TIMER 1 CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
T1GINV(1)	TMR1GE(2)	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

0	**0**	prescaler	**0**	**1**	**0**	**1**

Settings at our frequency measurment lab.

How to read from a 16-bit "Freerunning" Timer1?

Timer 1 is a 16-bit counter. It must be read as two 8-bit numbers, the 8 most significant bits **TMR1H** and the 8 last significant bits **TMR1L**. This can be a problem because the timer can "turn around" between the readings of 8-bit numbers. The following code shows the safe way:

```
long unsigned int time; char TEMPH; char TEMPL;
TEMPH = TMR1H; TEMPL = TMR1L;
if (TEMPH == TMR1H) // Timer1 not rolled over = good value
 {
   time = TEMPH*256;
   time += TEMPL;
 }
else // Timer1 rolled over - no new rollover for some time
     // lots of time to read new good values
 {
   time = TMR1H*256;
   time += TMR1L;
 }
```

OK direct

OK now

How to write to a 16-bit "Freerunning" Timer1?

It can also be problematic to write to a 16-bit counter as it must be done as two 8-bit number.
This is the safe way :

```
TMR1L = 0; // clear low byte = no rollover for some time
TMR1H = 12345/256; // high byte of constant 12345
TMR1L = 12345%256; // low byte of constant 12345
```

The number **12345** fits in 16 bits. With integer division / and the och modulo operator **%** a constant can be split into two 8-bit parts 8at compilation time). One other way is to use hexadecimal constants:

$12345_{10} = 3039_{16}$ `TMR1H=0x30` `TMR1L=0x39`

CCP **synchronized registers**

ECCP-unit, Enhenced Capture/Compare/(PWM)

• One can avoid writing to and reading from Timer1 registers - there is *synchronized registers* in the ECCP unit for this!

FIGURE 11-1: CAPTURE MODE OPERATION BLOCK DIAGRAM

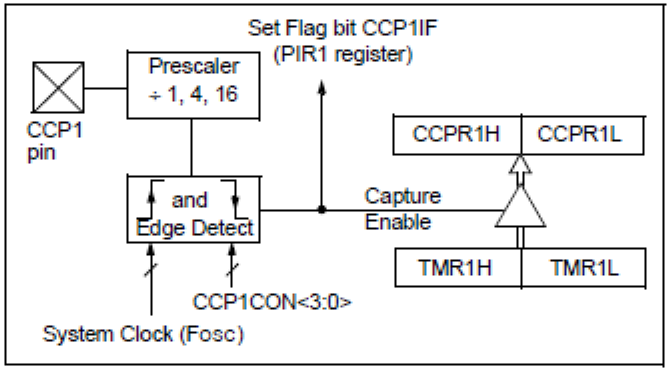
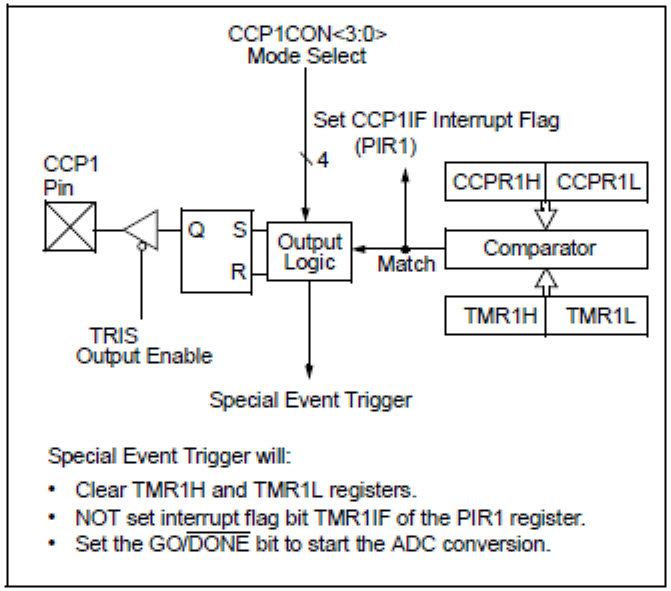


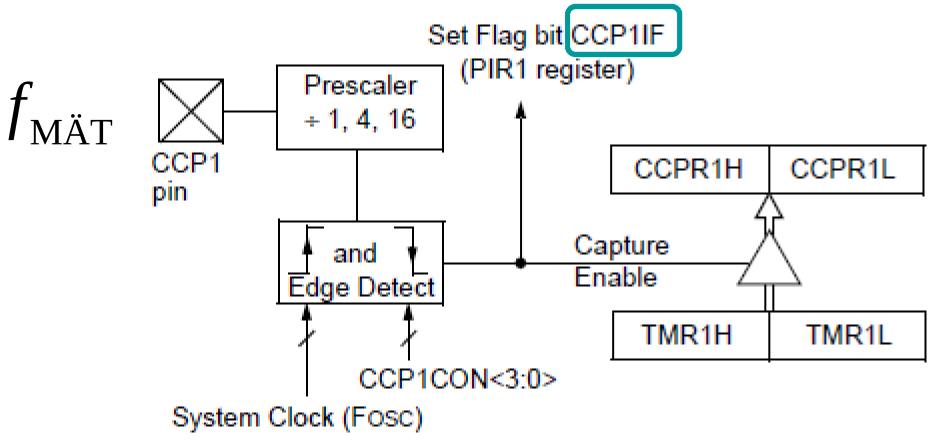
FIGURE 11-2: COMPARE MODE OPERATION BLOCK DIAGRAM



Special Event Trigger will:
- Clear TMR1H and TMR1L registers.
- NOT set interrupt flag bit TMR1IF of the PIR1 register.
- Set the GO/$\overline{\text{DONE}}$ bit to start the ADC conversion.

• **`CCPR1H`** and **`CCPR1L`**

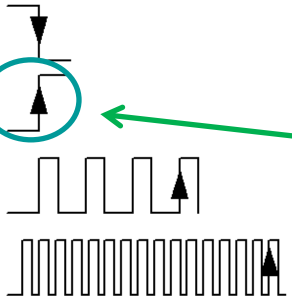
ECCP Capture modes



When the Capture event occurs Timer1 is directly copied to the **CCPR1H** and **CCPR1L** registers where they can be read where they can be read in "peace ". Bit **CCP1IF** signals when this happens. We must then reset this bit

REGISTER 11-1: CCP1CON: ENHANCED CCP1 CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0
bit 7							bit 0
-	-	-	-	0	1	0	1



Periodtime measurement

Multiperiodtime measurement

CCP1M<3:0>: ECCP Mode Select bits

0000 =	Capture/Compare/PWM off (resets ECCP module)
0100 =	Capture mode, every falling edge
0101 =	Capture mode, every rising edge
0110 =	Capture mode, every 4th rising edge
0111 =	Capture mode, every 16th rising edge

Setup Timer1

Timer1, as fast as possible:

```
// Setup TIMER1

/*
  0.x.xx.x.x.x.x TMR1 gate not invert
  x.0.xx.x.x.x.x TMR1 gate not enable
  x.x.00.x.x.x.x Prescale 1:1
  x.x.xx.0.x.x.x TMR1-oscillator is shut off
  x.x.xx.x.1.x.x no input clock-synchronization
  x.x.xx.x.x.0.x Use internal clock f_osc/4
  x.x.xx.x.x.x.1 TIMER1 is ON
*/

T1CON = 0b0.0.00.0.1.0.1 ;
```

Clear comment that shows how the **`T1CON`** value is developed.

Setup ECCP

CCP1, capture time for positive edges :

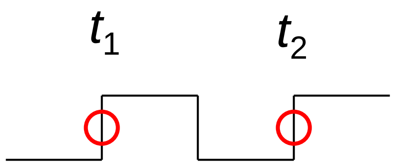
```
// Setup CCP1
/*
  00.00.xxxx -- --
  xx.xx.0101 Capture each positive edge
*/

CCP1CON = 0b00.00.0101 ;
```

Wait for the edges

```
unsigned long T, f, t1, t2;
```
16-bit numbers

```
CCP1IF = 0 ;               // reset the flag
while (CCP1IF == 0 ) ; // wait for capture
t1 = CCPR1H*256;
t1 += CCPR1L;
CCP1IF = 0 ;               // reset the flag
while (CCP1IF == 0 ) ; // wait for next capture
t2 = CCPR1H*256;
t2 += CCPR1L;
```



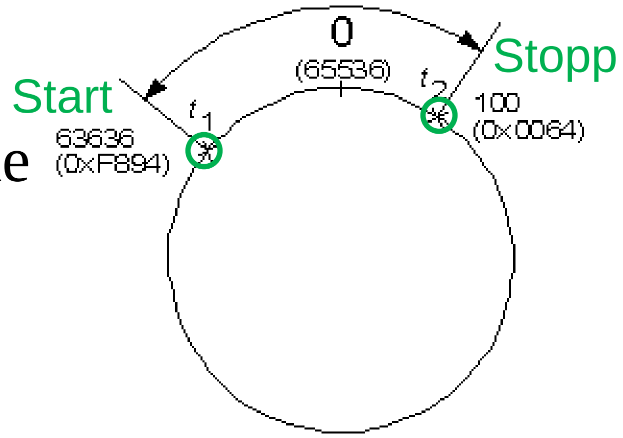
$$T = t_2 - t_1 \qquad f = \frac{1}{T}$$

```
T = t2 - t1;               // calculate period
f = 1000000U/T;            // calculate frequency
```

t2 – t1

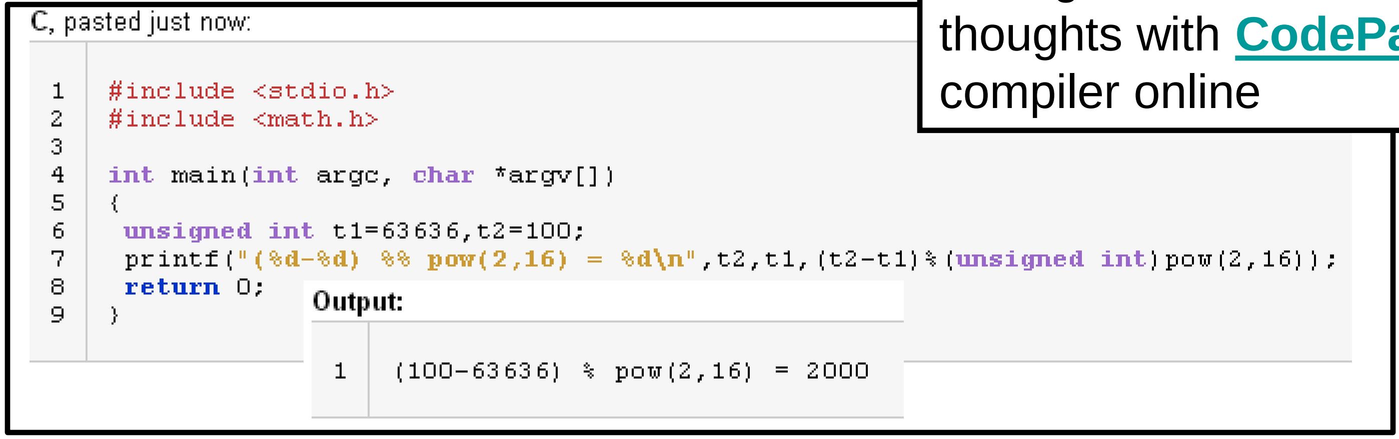
```
unsigned long T, f, t1, t2;
```

- What will happen if $\boxed{t1 > t2}$ (100-65636)?



The difference t2-t1 is taken **modulo 2^{16}** so the number of counts between t1 and t2 will always be the correct value ”around the circle”!

$$(100 - 63636) \bmod (2^{16}) = 2000$$


C, pasted just now:

```c
#include <stdio.h>
#include <math.h>

int main(int argc, char *argv[])
{
 unsigned int t1=63636,t2=100;
 printf("(%d-%d) %% pow(2,16) = %d\n",t2,t1,(t2-t1)%(unsigned int)pow(2,16));
 return 0;
}
```

Output:

```
(100-63636) % pow(2,16) = 2000
```

It’s a good idea to check your thoughts with **CodePad** C compiler online

(**Codepad** Online C-compilator)

It is convenient to try your formulas with a standard C compiler. One must then take into account that the PIC processor has different variable sizes than what is the usually standard. You must print the results with the "module" the PIC processor uses.

```
// int in Codepad is 32 bit
// int in Cc5x PIC is 8 bit
// long int in Cc5x PIC is 16 bit
int a= -25;
printf(”PIC int a=%d”, a%256);
printf(”PIC long int a=%d”, a%pow(2,16));
```

Cc5x-compiler does not follow the C-standard (this of performance reasons). You need to read the manual, and you need to "test drive" computational part of your program with the hardware to make sure that you understood everything correctly.

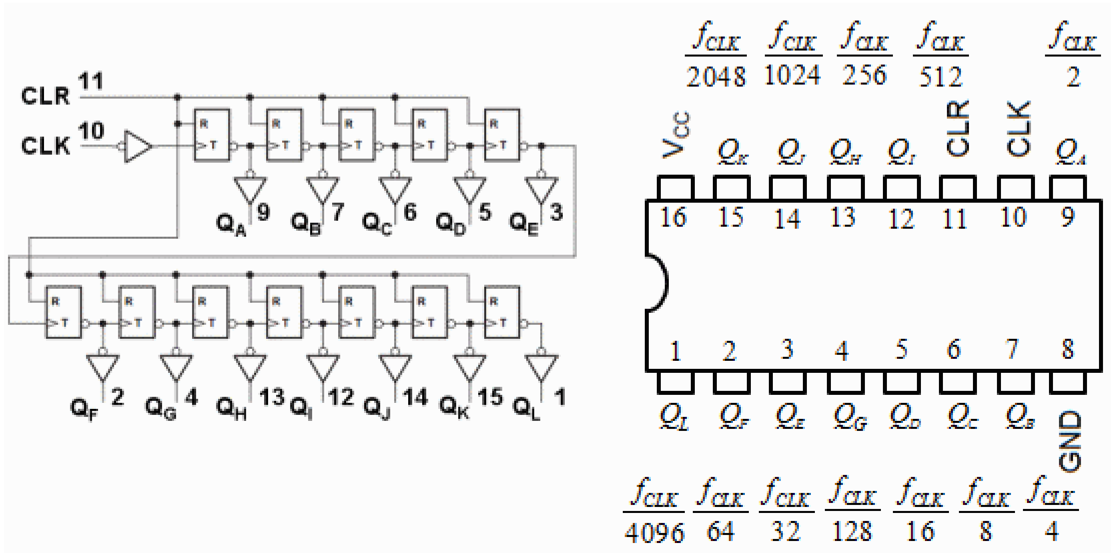
f=1000000U/T

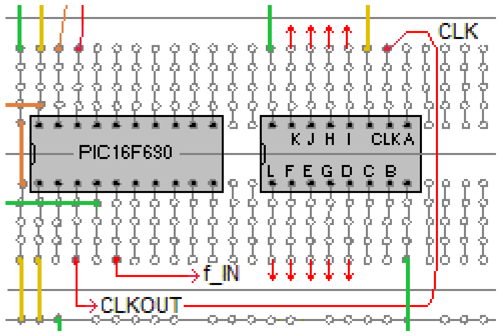
```
unsigned long T, f, t1, t2; /* long is max 65535 */
f = 1000000U/T;
```

- **Scalefactor** between *f* and *T* is 1000000. Timer1 is clocked with 1 MHz.
- If *T*=1 (T=1±1) the measured frquency is 1 MHz. *f* > 65535, to big for 16 bit.
- If *T*=10 (T=10±1) the measured frequency is 100 kHz. *f* > 65535, to big.
- If *T*=**100** (T=100±1) the measured frequency is **10 kHz**. *f* < 65535, ok.
- If *T*=**1000** (T=1000±1) the measured frequency is **1 kHz**. *f* < 65535, ok.
- If *T*=**10000** (T=10000±1) measured frequency is **100 Hz**. *f* < 65535, ok.
- If *T* > 65535 TMR1 overflows can be anything *f* = ?

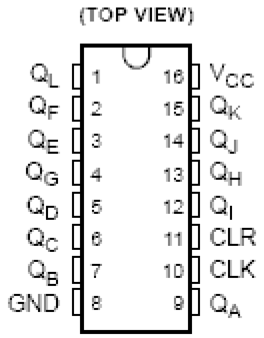
Frequence measurement lab

PIC16F690 can distribute the processor clock f_{OSC} **/4** = 1 MHz to the pin **CLKOUT**. Wit the cheap frequency divider chip 74HC4040 we will get 12 different frequencies for for measuring purposes!

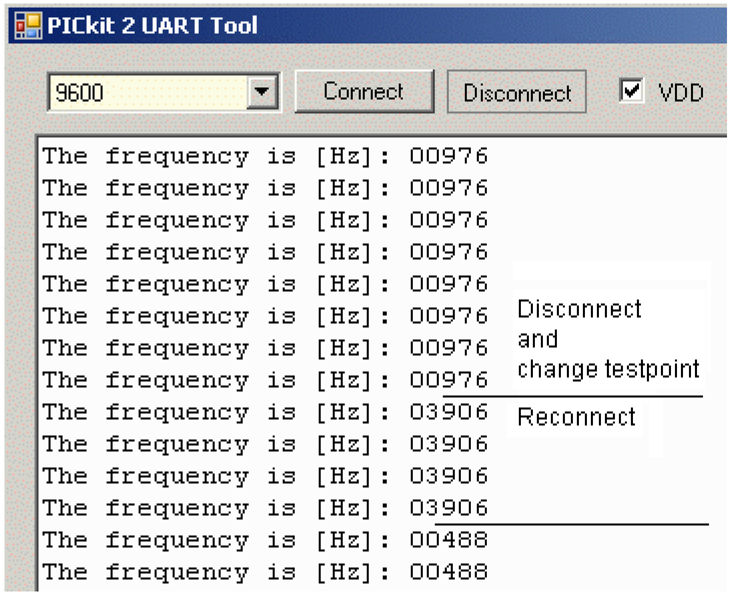




Frequence measurement lab



Why is the readings so incredibly precise ? Have you got hold of a ***super*** PIC16F690?



```
The frequency is [Hz]: 00976
The frequency is [Hz]: 00976
The frequency is [Hz]: 00976
The frequency is [Hz]: 00976
The frequency is [Hz]: 00976
The frequency is [Hz]: 00976
The frequency is [Hz]: 00976
The frequency is [Hz]: 00976
The frequency is [Hz]: 03906
The frequency is [Hz]: 03906
The frequency is [Hz]: 03906
The frequency is [Hz]: 03906
The frequency is [Hz]: 00488
The frequency is [Hz]: 00488
```

Disconnect
and
change testpoint

Reconnect

74HC4040 $f_{CLK} = 1$ MHz

PIN	freq [Hz]	Uppmätt [Hz]	Kommentar
1	$10^6/2^{12} = 244,1$		
2	$10^6/2^6 = 15625$		
3	$10^6/2^5 = 31250$		
4	$10^6/2^7 = 7812,5$		
5	$10^6/2^4 = 62500$		
6	$10^6/2^3 = 125000$		
7	$10^6/2^2 = 250000$		
9	$10^6/2 = 500000$		
12	$10^6/2^9 = 1953,1$		
13	$10^6/2^8 = 3906,3$	3906	
14	$10^6/2^{10} = 976,6$	976	
15	$10^6/2^{11} = 488,3$	488	

?

Something seems fishy …