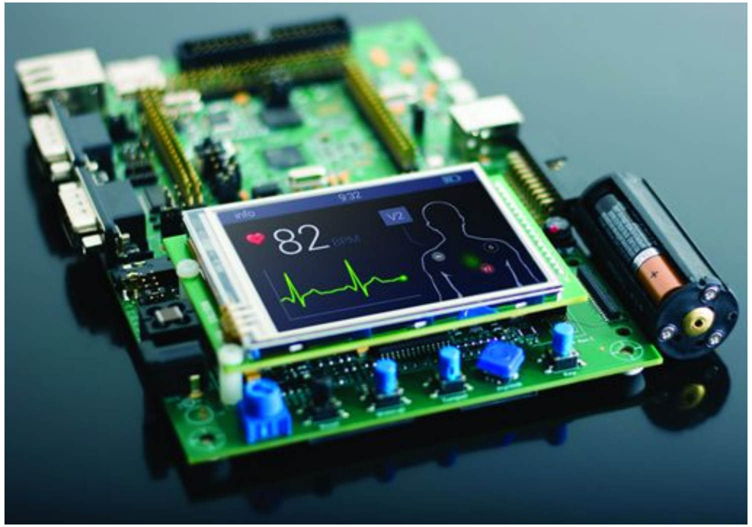
Embedded Software

CS 145/145L



Caio Batista de Melo

Announcements (2022-04-07)

- Homework #1 is due tomorrow!
- You should have started the project 1 already!
    - Due date for partner formation is tomorrow as well

Recap

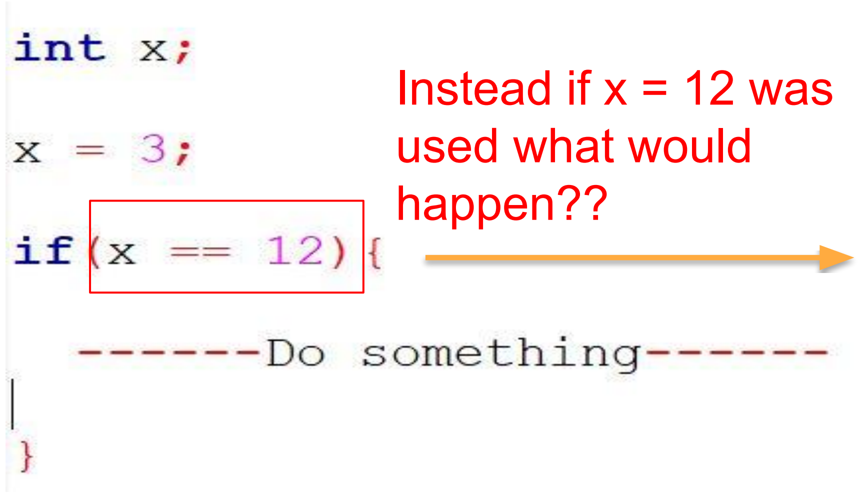
- Cross Compilation
- GPIO => State(s) – ON and OFF

  1 and 0

  +5V and GND
- SFRs have a type and store data like variables.
- SFR => More than variables

  PORT *, PIN *, DDR *   * = A / B / C / D
- Bit manipulation

Common Bug

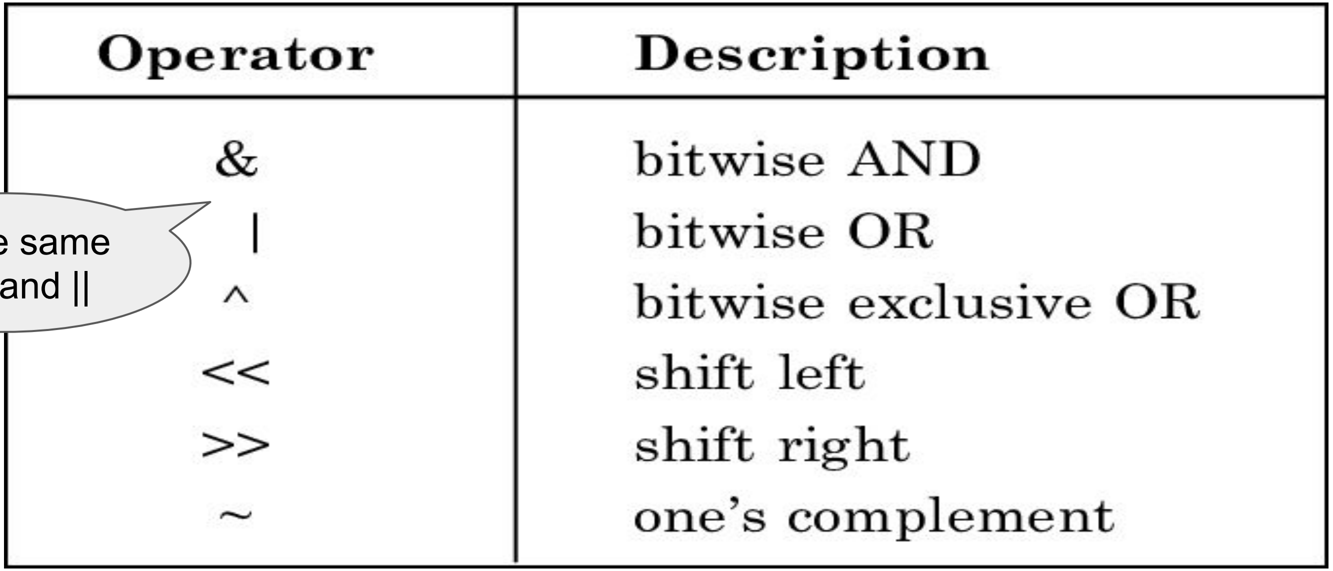


```
int x;

x = 3;

if (x == 12) {

   -------Do something-------

}
```

Instead if x = 12 was used what would happen??

if (12 == x)

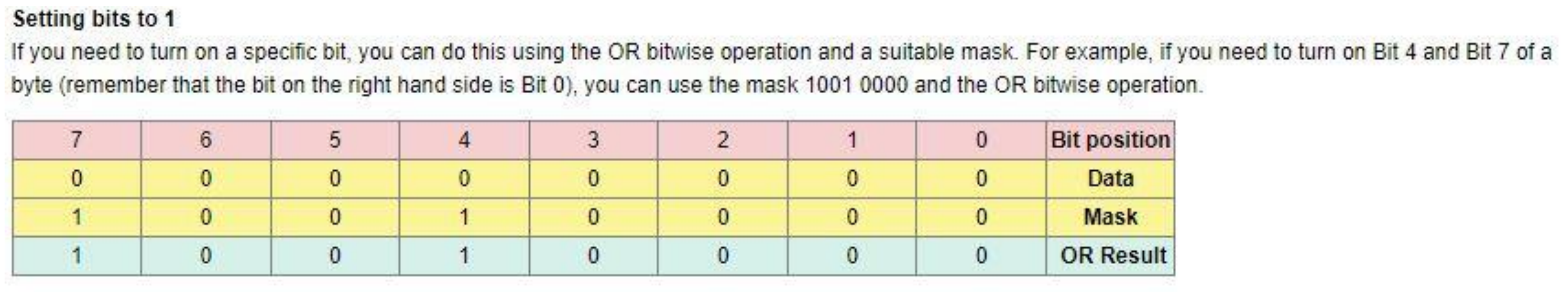
Bit Operators



Operator	Description
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right
~	one's complement

Not the same as && and ||

Masking Bits – Operation Set Bit



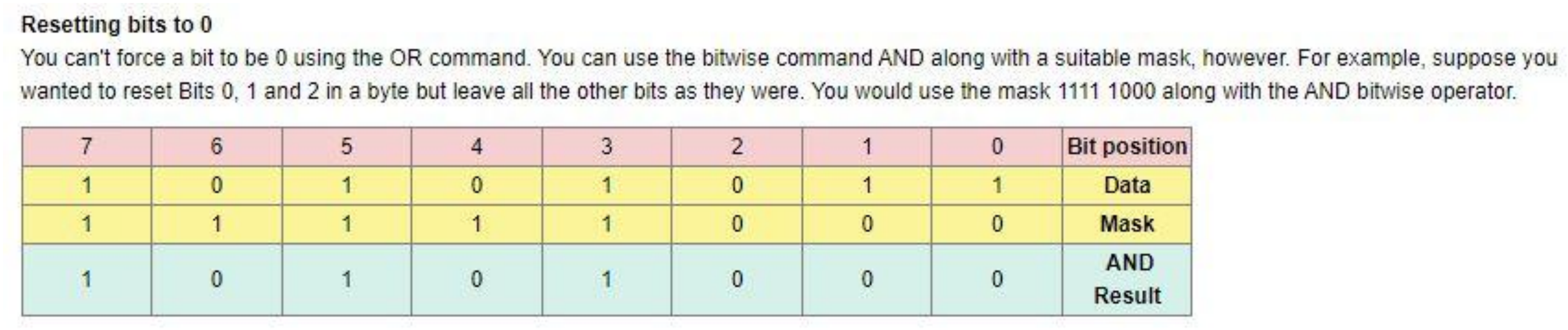
Setting bits to 1

If you need to turn on a specific bit, you can do this using the OR bitwise operation and a suitable mask. For example, if you need to turn on Bit 4 and Bit 7 of a byte (remember that the bit on the right hand side is Bit 0), you can use the mask 1001 0000 and the OR bitwise operation.

7	6	5	4	3	2	1	0	Bit position
0	0	0	0	0	0	0	0	Data
1	0	0	1	0	0	0	0	Mask
1	0	0	1	0	0	0	0	OR Result

You need bit-wise <u>OR</u> (“|”) operation to <u>SET</u> a bit

Masking Bits – Reset / Clear Operation



Resetting bits to 0

You can't force a bit to be 0 using the OR command. You can use the bitwise command AND along with a suitable mask, however. For example, suppose you wanted to reset Bits 0, 1 and 2 in a byte but leave all the other bits as they were. You would use the mask 1111 1000 along with the AND bitwise operator.

7	6	5	4	3	2	1	0	Bit position
1	0	1	0	1	0	1	1	Data
1	1	1	1	1	0	0	0	Mask
1	0	1	0	1	0	0	0	AND Result

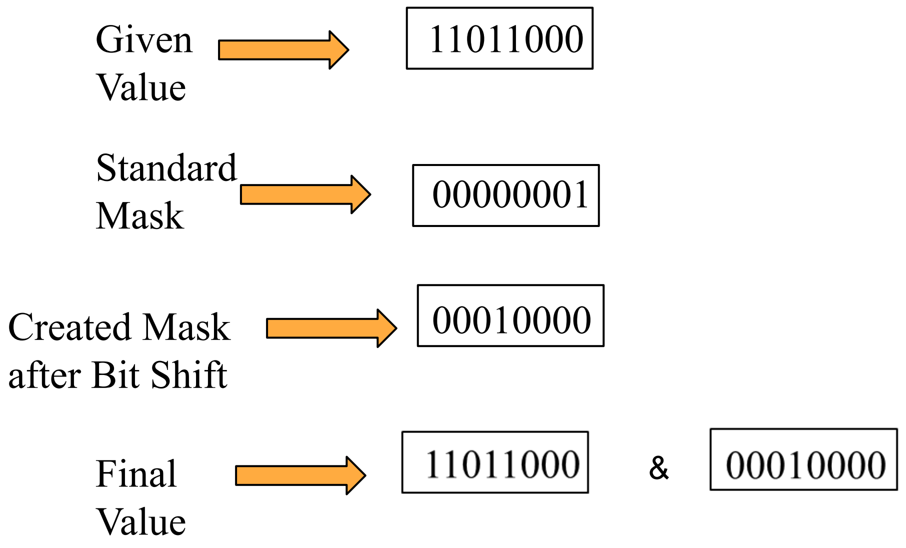
You need bit-wise <u>AND</u> ("&") operation to <u>CLEAR</u> a bit

Creating Masks

Consider you want to clear every bit except the bit in the 5th position

Solution:

1. Create a standard mask
   (mask = 1)
2. Left shift it by four spaces
   (1 =>00010000)
3. AND it with the current value
   (value & 00010000)

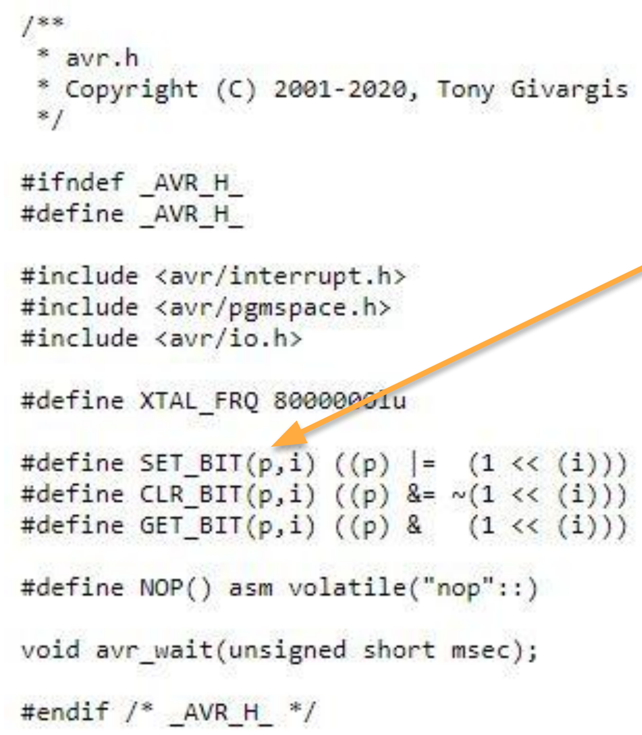


Clearing a Specific Bit

Consider you want to clear the bit in the 5th position

Given Value	11011000		
Standard Mask	00000001		
Created Mask after Bit Shift	00010000		
Invert the mask and Bitwise AND	11101111	&	11011000

avr.h



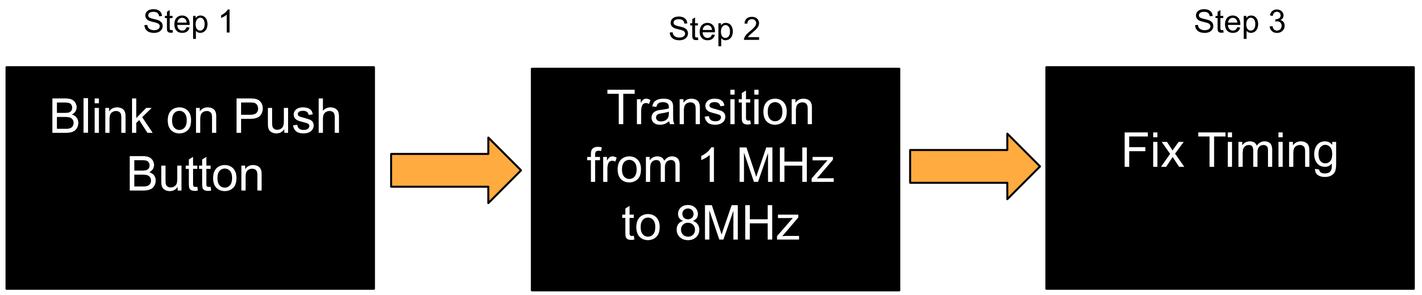
```
/**
 * avr.h
 * Copyright (C) 2001-2020, Tony Givargis
 */

#ifndef _AVR_H_
#define _AVR_H_

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

#define XTAL_FRQ 8000000lu

#define SET_BIT(p,i) ((p) |=  (1 << (i)))
#define CLR_BIT(p,i) ((p) &= ~(1 << (i)))
#define GET_BIT(p,i) ((p) &   (1 << (i)))

#define NOP() asm volatile("nop"::)

void avr_wait(unsigned short msec);

#endif /* _AVR_H_ */
```

The i_{th} bit of variable *p*

Bit Operators

Test this out

Project 1 Roadmap



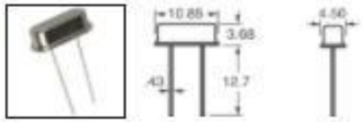
Blink according to instruction delay loops

What is the problem after this step?

Use built-in Timers

8 MHz Crystal

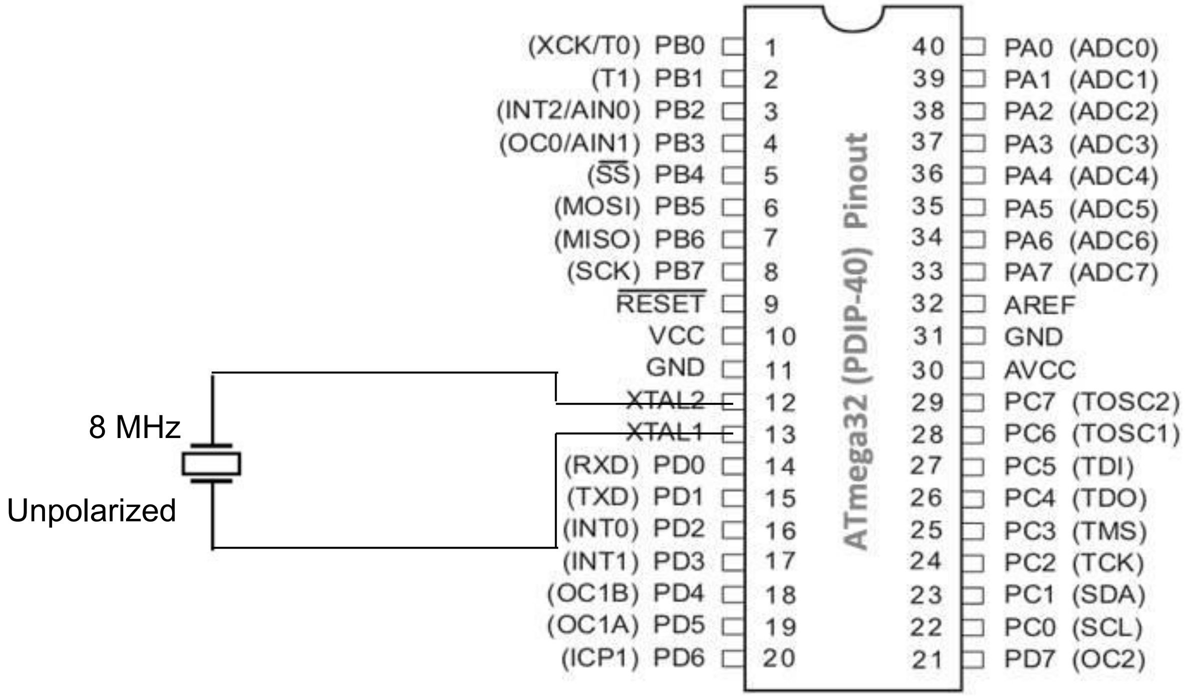
Image shown is a representation only. Exact specifications should be obtained from the product data sheet.



ATS08A

Digi-Key Part Number	CTX406-ND
Manufacturer	CTS-Frequency Controls
Manufacturer Product Number	ATS08A
Supplier	CTS-Frequency Controls
Description	CRYSTAL 8.0000MHZ 20PF TH
Manufacturer Standard Lead Time	24 Weeks
Detailed Description	8 MHz ±30ppm Crystal 20pF 60 Ohms HC-49/US
Customer Reference	Customer Reference
Datasheet	Datasheet

Adding 8 MHz Crystal (Hardware Method)



Fuse in Software

If you're using Microchip Studio:

1. You need to access the Fuse settings in Device Programming menu;
2. Last item in the list -> LOW.SUT_CKSEL;
3. Last choice -> "Ext. Crystal/Resonator High Freq.; Start-up time: 16K CK + 64 ms";
4. <u>DON'T CHANGE ANYTHING ELSE!</u>
5. Click program and close!

Ref: https://microchipdeveloper.com/8avr:avrfuses

Fuse in Software

If you're using MPLAB X:

1. Window -> Target Memory Views -> Configuration Bits;
2. Click Read Configuration Bits;
3. Second item in the list -> FIELD == LOW.SUT_CKSEL;
4. Choose "Ext. Crystal/Resonator High Freq.; Start-up time: 16K CK + 64 ms";
5. <u>DON'T CHANGE ANYTHING ELSE!</u>
6. Click Program Configuration Bits and close this window.

Ref: https://microchipdeveloper.com/mplabx:view-and-set-configuration-bits

Fuse in Software

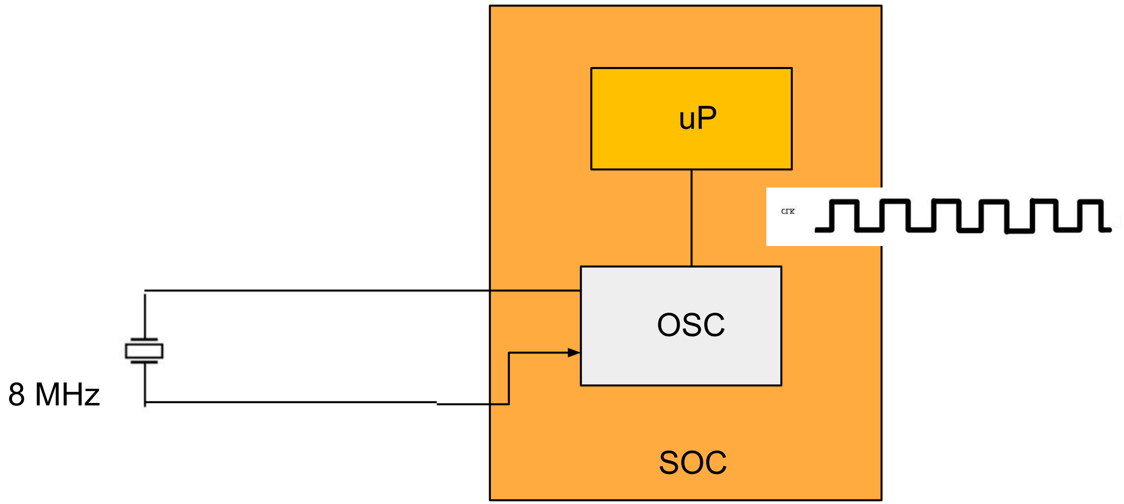
If you're using something other than those (platformio?)…

GOOD LUCK :)

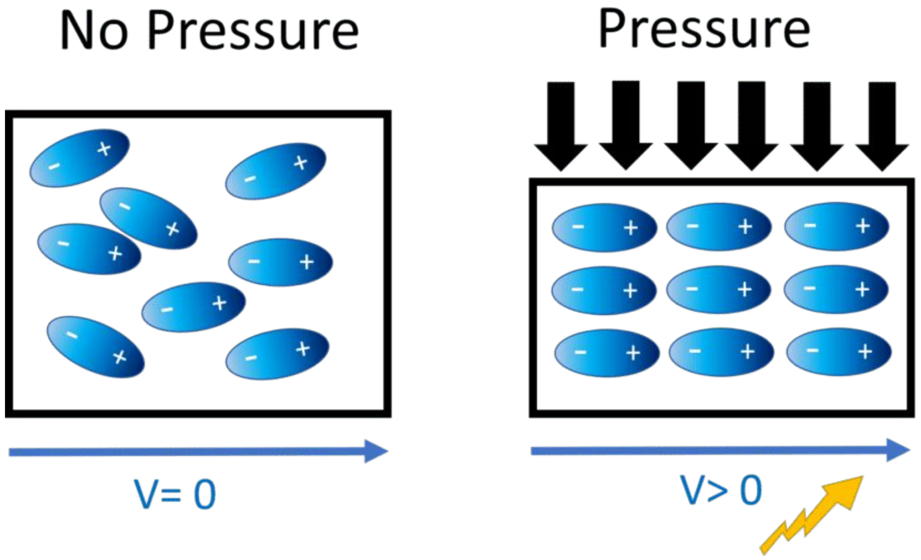
This might help though:

https://caiobatista.com/uploads/courses/uci/s22/cs145/avrdude-examples.pdf

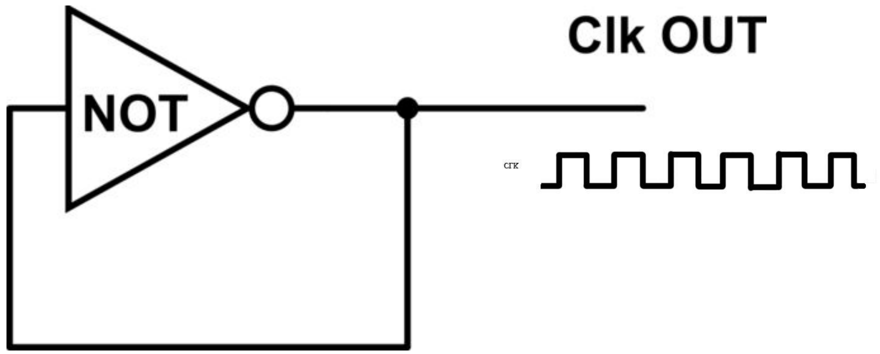
Layout with Crystal



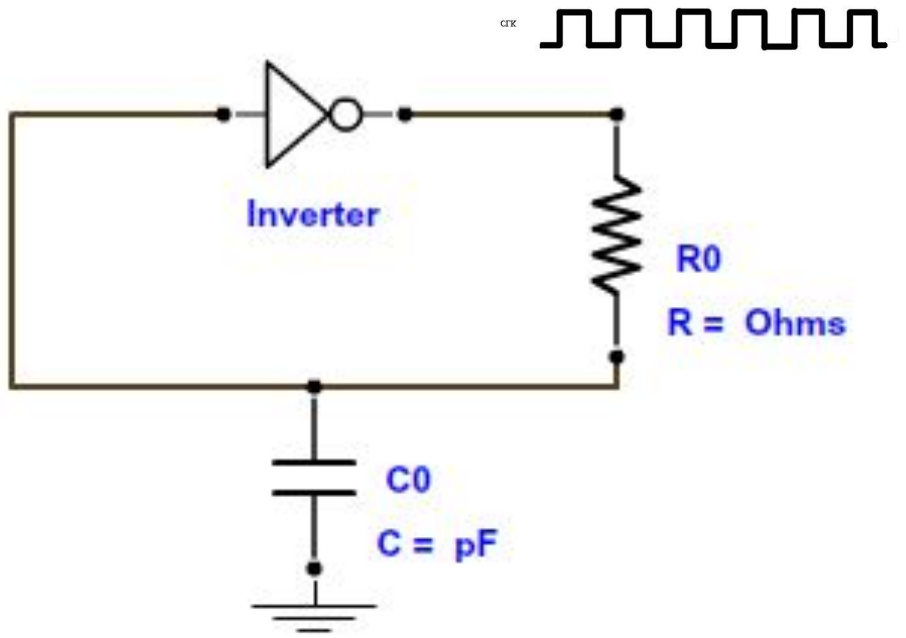
Piezoelectric



Work of an Oscillator



Using R and C to control the Switching



Timers - Bucket Analogy

- Capacity
- Rate
- Overflow
- Seed

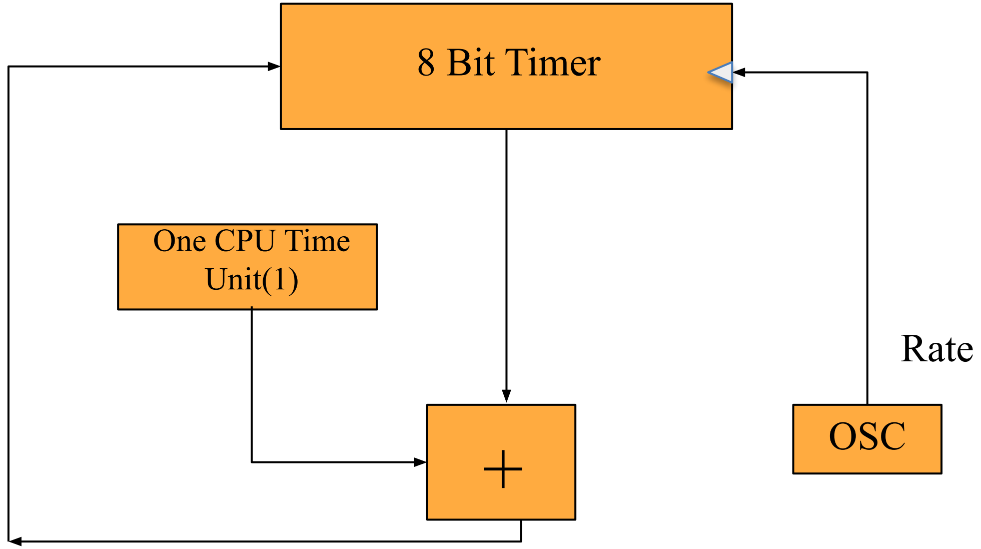
Calculate Seed based on T

$$\begin{aligned}\text{Seed} &= \text{Capacity} - (T * \text{Rate})\\ &= 5_{L} - (3_{min} * 1_{L/min})\\ &= 2_{L}\end{aligned}$$

Timer Block Diagram

Your AVR has 3 physical internal timers.
Let's use Timer0.

Count from 0 to 255 and set overflow flag and reset



Software to Work with a Timer

```c
/**
 * avr.h
 * Copyright (C) 2001-2020, Tony Givargis
 */

#ifndef _AVR_H_
#define _AVR_H_

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

#define XTAL_FRQ 8000000lu

#define SET_BIT(p,i) ((p) |=  (1 << (i)))
#define CLR_BIT(p,i) ((p) &= ~(1 << (i)))
#define GET_BIT(p,i) ((p) &   (1 << (i)))

#define NOP() asm volatile("nop"::)

void avr_wait(unsigned short msec);

#endif /* _AVR_H_ */
```

Time in *ms*

```c
/**
 * avr.c
 * Copyright (C) 2001-2020, Tony Givargis
 */

#include "avr.h"

void
avr_wait(unsigned short msec)
{
        TCCR0 = 3;
        while (msec--) {
                TCNT0 = (unsigned char)(256 - (XTAL_FRQ / 64) * 0.001);
                SET_BIT(TIFR, TOV0);
                while (!GET_BIT(TIFR, TOV0));
        }
        TCCR0 = 0;
}
```

Timer ON with Xtal Freq

Timer OFF

While loop calculates 1 ms

Turning the Timer On/Off

```
TCCR0 = 3;
.
.
.
TCCR0 = 0;
```

Timer/Counter Control Register – TCCR0 (Refer to page 80~82 on ATmega32 Manual)

- **Bit 2:0 – CS02:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter.

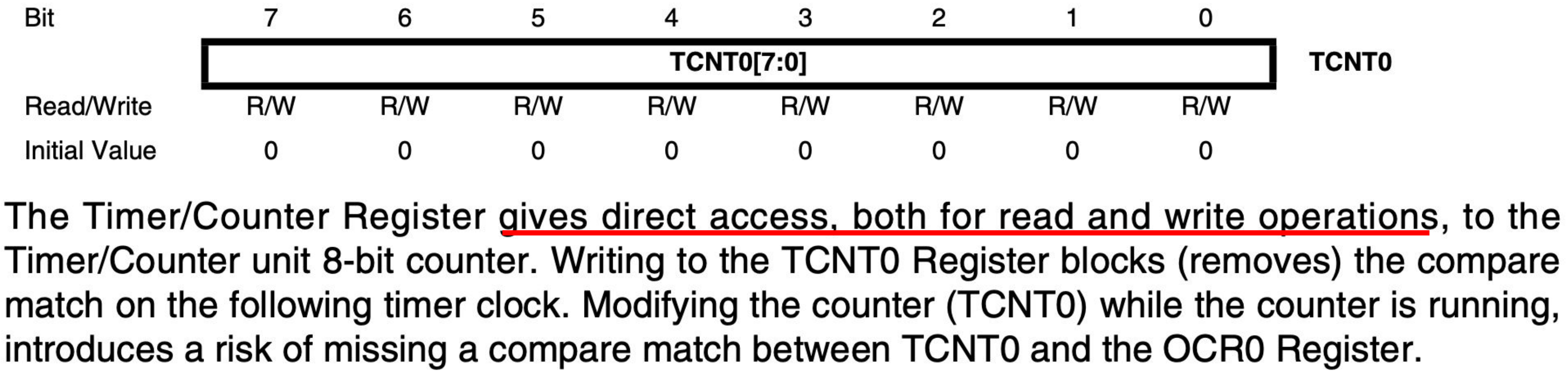
Table 42. Clock Select Bit Description

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.

Setting a Seed

```
TCNT0 = (unsigned char)(256 - (XTAL_FRQ / 64) * 0.001);
```

Timer/Counter Register – TCNT0 (Refer to page 82 on ATmega32 Manual)



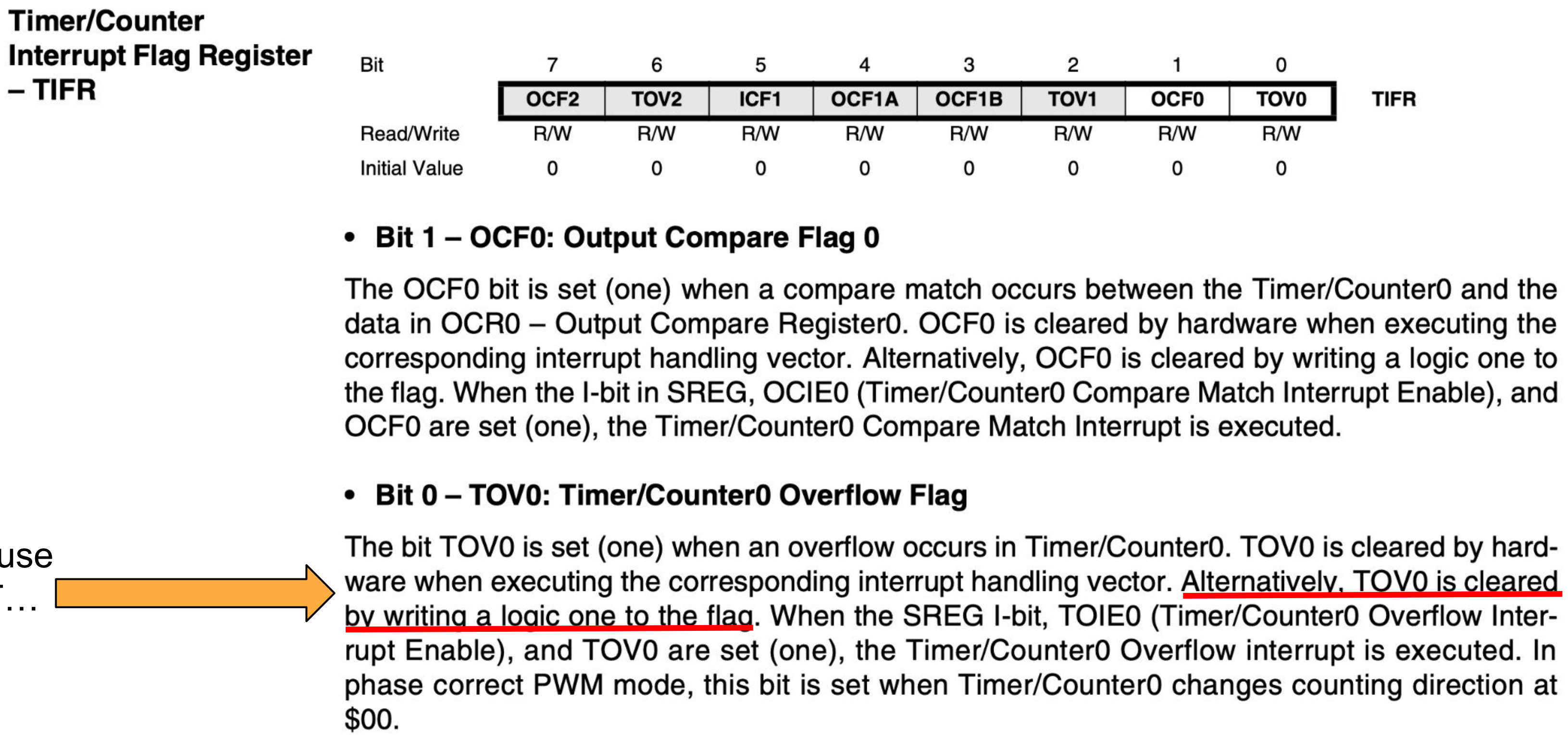
Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT0 Register blocks (removes) the compare match on the following timer clock. Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a compare match between TCNT0 and the OCR0 Register.

Waiting for Overflow

```
SET_BIT(TIFR, TOV0);
while (!GET_BIT(TIFR, TOV0));
```

Timer/Counter Interrupt Flag Register– TIFR (Refer to page 83 on ATmega32 Manual)



Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
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	

- **Bit 1 – OCF0: Output Compare Flag 0**

The OCF0 bit is set (one) when a compare match occurs between the Timer/Counter0 and the data in OCR0 – Output Compare Register0. OCF0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0 is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0 (Timer/Counter0 Compare Match Interrupt Enable), and OCF0 are set (one), the Timer/Counter0 Compare Match Interrupt is executed.

- **Bit 0 – TOV0: Timer/Counter0 Overflow Flag**

The bit TOV0 is set (one) when an overflow occurs in Timer/Counter0. TOV0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV0 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE0 (Timer/Counter0 Overflow Interrupt Enable), and TOV0 are set (one), the Timer/Counter0 Overflow interrupt is executed. In phase correct PWM mode, this bit is set when Timer/Counter0 changes counting direction at $00.

Why we use SET_BIT…

See you next time :)

Q & A