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

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

```java
int x;
x = 3;
if (x == 12) {
    ----Do something-----
}
```
Bit Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>bitwise AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>bitwise exclusive OR</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>shift left</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>shift right</td>
</tr>
<tr>
<td>~</td>
<td>one’s complement</td>
</tr>
</tbody>
</table>

Not the same as && and ||
You need bit-wise OR ("|") operation to SET a bit
Masking Bits – Reset / Clear Operation

You need bit-wise **AND** ("&") operation to **CLEAR** 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)

Given Value: 11011000
Standard Mask: 00000001
Created Mask after Bit Shift: 00010000
Final Value: 11011000 & 00010000
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 & 1101000
Bit Operators

Test this out

The \(i_{th}\) bit of variable \(p\)
Project 1 Roadmap

Step 1: Blink on Push Button
Blink according to instruction delay loops

Step 2: Transition from 1 MHz to 8 MHz
What is the problem after this step?

Step 3: Fix Timing
Use built-in Timers
# 8 MHz Crystal

## ATS08A

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digi-Key Part Number</td>
<td>CTX406-ND</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>CTS-Frequency Controls</td>
</tr>
<tr>
<td>Manufacturer Product Number</td>
<td>ATS08A</td>
</tr>
<tr>
<td>Supplier</td>
<td>CTS-Frequency Controls</td>
</tr>
<tr>
<td>Description</td>
<td>CRYSTAL 8.0000MHZ 20PF TH</td>
</tr>
<tr>
<td>Manufacturer Standard Lead Time</td>
<td>24 Weeks</td>
</tr>
<tr>
<td>Detailed Description</td>
<td>8 MHz ±30ppm Crystal 20pF 60 Ohms HC-49/US</td>
</tr>
</tbody>
</table>

Image shown is a representation only. Exact specifications should be obtained from the product data sheet.
Adding 8 MHz Crystal (Hardware Method)
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. **DON’T CHANGE ANYTHING ELSE!**
5. Click program and close!

Ref: [https://microchipdeveloper.com/8avr:avrfuses](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. DON’T CHANGE ANYTHING ELSE!
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:

Layout with Crystal

8 MHz

uP

OSC

SOC
Piezoelectric

No Pressure

\[ V = 0 \]

Pressure

\[ V > 0 \]
Work of an Oscillator
Using R and C to control the Switching
Timers - Bucket Analogy

- Capacity
- Rate
- Overflow
- Seed

Calculate Seed based on T

\[
\text{Seed} = \text{Capacity} - (T \times \text{Rate})
\]

\[
= 5_L - (3_{\text{min}} \times 1_{L/\text{min}})
\]

\[
= 2_L
\]
Your AVR has 3 physical internal timers. Let's use Timer0.

8 Bit Timer

Count from 0 to 255 and set overflow flag and reset

One CPU Time Unit(1)

Rate

OSC
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_ */

/**
 * 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;
}
```

**Software to Work with a Timer**

Time in ms

Timer ON with Xtal Freq

While loop calculates 1 ms

Timer OFF
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>
<thead>
<tr>
<th>CS02</th>
<th>CS01</th>
<th>CS00</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No clock source (Timer/Counter stopped).</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>clk_I/O (No prescaling)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>clk_I/O/8 (From prescaler)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>clk_I/O/64 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>clk_I/O/256 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>clk_I/O/1024 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>External clock source on T0 pin. Clock on falling edge.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>External clock source on T0 pin. Clock on rising edge.</td>
</tr>
</tbody>
</table>
TCNT0 = (unsigned char)(256 - (XTAL_FRQ / 64) * 0.001);
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));
Why we use SET_BIT...

Timer/Counter Interrupt Flag Register– TIFR

(RRefer to page 83 on ATmega32 Manual)

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OCF2</td>
<td>TOV2</td>
<td>ICF1</td>
<td>OCF1A</td>
<td>OCF1B</td>
<td>TOV1</td>
<td>OCF0</td>
<td>TOV0</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **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 $00$. 

CS145 - Spring '22
See you next time :)