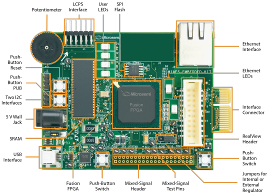
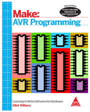


Bit Twiddling

Kizito NKURIKIYEYEZU, Ph.D.

Reading material

1. Bit manipulation (AKA "Programming 101")[1]
2. Chap 4 of Williamson, E. (2014). Make: Avr programming. Maker Media[2].
3. AVR Bit Manipulation in C[3]
4. Bitwise Operations in Embedded Programming[4]



[1] https://www.avrfreaks.net/forum/tut-c-bit-manipulation-aka-programming-101?page=all
[2] https://apprize.best/hardware/avr/5.html
[3] http://www.rjhcoding.com/avrc-bit-manip.php
[4] https://binaryupdates.com/bitwise-operations-in-embedded-programming/

Why bit twiddling?

- When setting PORTs and DDRs, one needs to be careful not to disturb the state of other bits of the register.
- For example, the following code attempts to set pin 2 of PORTD

```
    DDRD  |= 0b00000100;
```

- Unfortunately, this code also clears all other bits of PORTD
- Bit twiddling allows not to set all 8 bits in register PORT without regard for the directions of each individual pin, i.e. all the bits stored in DDR
- For example, the above example could be best solved as follows

```
  DDRD = DDRD | (1<<2);
  /*which can also be written as*/
  DDRD   |= (1<<2);
```

- Please read “Programming 101 - By Eric Weddington”[5] for more details.

[5] https://www.avrfreaks.net/forum/tut-c-bit-manipulation-aka-programming-101?page=all

Bit Shifting

- Bit shifting—a bitwise operator that allows to move (to the left or right) the order of one or several bits
- Bit-shifting is very fast and required fewer CPU operations compared to arithmetic (e.g., multiplication and division) operations.
- Bit shifting uses Bitwise Operators[6]

Operator	Name	Example	Result
&	Bitwise AND	6 & 3	2
|	Bitwise OR	10 | 10	10
^	Bitwise XOR	2^2	0
~	Bitwise 1’s complement	~9	-10
<<	Left-Shift	10<<2	40
>>	Right-Shift	10>>2	2

FIG 1. Example of Bitwise operations

[6] https://en.wikipedia.org/wiki/Bitwise_operation

Bit Shifting

There are three main types of shifts:

- Left Shifts—When shifting left, the most-significant bit is lost, and a 0 bit is inserted on the other end.
  - The left shift operator is usually written as <<

```
    (0010 << 1)=0100          /*(2<<1)=4*/
    (0010 << 2)=1000         /*(2<<2)=8*/
```

- Right Shifts—When shifting right with an arithmetic right shift, the least-significant bit is lost and the most-significant bit is copied.
  - The right shift operator is usually written as >>

```
    (1011 >> 1)=1101            /*(11>>1)=5 */
    (1011 >> 3)=0001           /*(11>>3)=1 */
```

- Logical Right Shifts—When shifting right with a logical right shift, the least-significant bit is lost and a 00 is inserted on the other end.

```
    (1011 >>> 1)=0101
    (1011 >>> 3)=0001
```


Controlling Memory-Mapped I/O Registers Using Bit Operations

Setting Bits with the OR operator

Consider the diodes in Figure 4 and Figure 5[7]

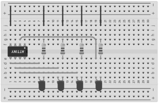


FIG 2

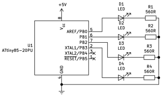


FIG 3

- How would you turn on LED1 while other LEDs are turned off?

```
 /*set the pin as an
    output*/
DDRB |= (1<<PBO);
/*set the bit PBO as
   high*/
PORTB |=(1<<PBO);
```

- How would you turn on only LED2 and LED3 and leave out other LEDs in their previous state?

```
PORTB |= (1<< PB1) |
    (1<< PB2);
```

[7] https://www.xibeoding.com/avrs-bit-manip.php

Clearing a bit with AND and NOT operators

- How to turn OFF LED 1 only

```
    /*Set PBO to low*/
    PORTB &=~(1<<PBO);
```

- How would you turn OFF only LED2 and LED3 and leave out other LEDs in their previous state?

```
PORTB &=~((1<<PB1) |(1<<
    PB2));
```

NOTE: There is a NOT outside the parentheses in order to have two zeros

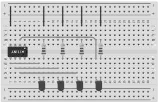


FIG 4

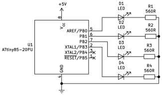


FIG 5

Toggling Bits with XOR operator

- How to toggle OFF LED 1 only

```
PORTB ^=(1<<PBO);
```

- How to toggle only LED2 and LED3 and leave out other LEDs in their previous state?

```
PORTB ^=((1<<PB1)|(1<<
    PB2));
```

Noted:

- Don't forget to set direction of pins first! else, the pin will not be set
- Remember if pins are configured as inputs (DDRBn bit is 0) then the corresponding

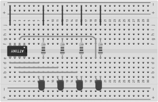


FIG 6

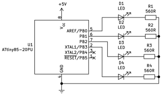


FIG 7

Testing a Bit

- Suppose we need to know if the switch *S*1 is pressed
- We use the PIN register to know the content of the PORT

```
int status=(PINB &(1<<PB));
if(status){
  // If the switch is pressed
}
```

- You can also check multiple switches

```
int status=PINB&((1<<PB4)|(1<<
    PB5))
if(status){
//If any of the switches is
    pressed
}
```

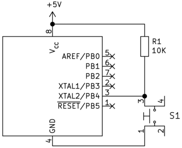


FIG 8

AVR bit twiddling

TAB 1. Important bit-twiddling operations.

Operation	Implementation in C	Implication
Set a bit	`PORTB |= (1<<PB1)`	Bit PB1 is set to 1 (other pins are left unchanged)
Clear bit	`PORTB &= ~(1<<PB1)`	Bit PB1 is set to 0 (other pins are left unchanged)
Toggle a bit	`PORTB ^= (1<<PB1)`	If Bit PB1 was 1, it is toggled to 0. Otherwise, it is set to 1 (other pins are left unchanged)
Read a value bit	`uint8_t bit = PORTB & (1<< PB1)`	Read and put the value of bit PB1 of PORTB into the variable bit. This is used to read switches.

Important readings:

- Please read the document—which is uploaded on the course website—entitled 'AVR Bit Twiddling' to better understand this important topic.
- You should also read "Bit manipulation" by By Eric Weddington [8]

[8] https://www.avrfreaks.net/forum/tut-c-bit-manipulation-aka-programming-101?page=all

Special bit twiddling AVR functions

One can use the **_BV(x)** macro defined in avr/sfr _defs.h which is included through avr/io.h as #define _BV(x) (1«x)

```
DDRD &= ~_BV(0); //set PORTD pin0 to zero as
    input
PORTD |= _BV(0); //Enable pull up;
DDRD |= _BV(1); //set PORTD pin1 to one as output
PORTD |= _BV(1); //led ON
while (1) {
  if (bit_is_clear(PIND, 0)) {
    //if button is pressed
    while (1) {
      PORTD &= ~_BV(1); //turn the led OFF
      //LED OFF while Button is pressed
      loop_until_bit_is_set(PIND, 0);
      PORTD |= _BV(1); //turn the led ON
    }
  }
}
```

Software Delay Functions

AVR GCC compiler's util/delay.h defines the _delay_ms(double ms) function

- Requires # include <util/delay.h >
- F_CPU preprocessor symbol should be defined as MCPU frequency in Hz using #define or passed through the -D compiler option
  - In code: #define F_CPU 8000000UL //8 MHz clock
  - Command line option: -D F_CPU=8000000UL
- The maximum delay is calculated as

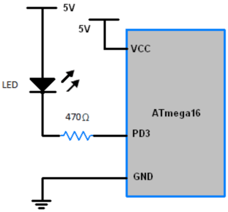
$$delay = \frac{4294967.295 \cdot 10^6}{F\_CPU} \tag{1}$$

- Thus, for an 8MHz clock, the maximum delay would be

$$delay = \frac{4294967.295 \cdot 10^6}{8 \cdot 10^6} = 536871 ms \tag{2}$$

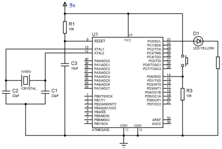
Example: blink an LED

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
    //Set all pins of DDR3 as output
    DDRD = DDRD | (1<<3);
    while(1)
    {
        //Turn on the LED by making pin PD3 high
        PORTD = PORTD | (1<<3);
        // Wait one second
        _delay_ms(1000);
        // Turn of the LED by making pin PD3 low
        PORTD = PORTD & (~(1<<3));
        _delay_ms(1000);
    }
        return 0;
}
```



Example: Reading switch

```
#include <avr/io.h>
#include <util/delay.h>
int main(void) {
  //Set PC0 as Output
  DDRC |= (1 << PC0);
  //Set PD0 as an input
  DDRD &= ~(1 << PD0);
  while (1) {
    //Turns OFF LED
    PORTC &= ~(1 << PC0);
    //If switch is pressed
    if (PIND & (1 << PD0) == 1) {
      //Turns ON LED for one second
      PORTC |= (1 << PC0);
      _delay_ms(1000);
    }
  }
}
```



The end