Unit 4

Microcontrollers (Arduino) Overview
Digital I/O

Using software to perform logic on individual (or groups) of bits

BIT FIDDLING

Numbers in Other Bases in C/C++

- Suppose we want to place the binary value 00111010 into a char variable, \( v \) [i.e. `char v;`]
  - We could convert to decimal on our own (58\(_{10}\))
  - \( v = 58; \)
  - All compilers support hexadecimal using the ___ prefix
    - \( v = 0x3a; \)
  - Our Arduino compiler supports binary using the ___ prefix
    - \( v = 0b00111010; \)
- Important note: Compilers convert EVERYTHING to equivalent _______. The 3 alternatives above are equivalent because the compiler will take all 3 and place 00111010 in memory.
  - Use whichever base makes the most sense in any given situation
  - *It is your (the programmer's)______________...the compiler will end up converting to binary once it is compiled*

Modifying Individual Bits

- Suppose we want to change only a single bit (or a few bits) in a variable [i.e. `char v;`] without changing the other bits
  - Set the LSB of \( v \) to 1 w/o affecting other bits
    - Would this work? \( v = 1; \)
  - Set the upper 4 bits of \( v \) to 1111 w/o affecting other bits
    - Would this work? \( v = 0xf0; \)
  - Clear the lower 2 bits of \( v \) to 00 w/o affecting other bits
    - Would this work? \( v = 0; \)
  - ____!!! Assignment changes ____ bits in a variable
- Because the smallest unit of data in computers is usually a byte, manipulating individual bits requires us to use BITWISE LOGICAL OPERATIONS.
  - Use _____ operations to clear individual bits to 0
  - Use _____ operations to set individual bits to 1
  - Use _____ operations to invert bits
  - Use AND to check a bit(s) value in a register
Bitwise Logical Operations

- ANDs can be used to control whether a bit passes unchanged or results in a __
- ORs can be used to control whether a bit passes unchanged or results in a __
- XORs can be used to control whether a bit passes unchanged or is inverted

<table>
<thead>
<tr>
<th>Ctrl</th>
<th>Bit</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

0 AND x = __
1 AND x = __
x AND x = __

T1 X + 0 = X
T1' X • 1 = X
T2 X + 1 = 1
T2' X • 0 = 0
T3 X + X = X
T3' X • X = X

Logical Operations

- Bitwise logic operations are often used for "bit fiddling"
  - Change the value of a bit in a register w/o affecting other bits
  - C operators: & = AND, | = OR, ^ = XOR, ~ = NOT

Logical Operations & Masking

- Logical operations on numbers means performing the operation on each pair of bits

<table>
<thead>
<tr>
<th>Ctrl</th>
<th>Bit</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

0 OR x = __
1 OR x = __
x OR x = __

0 XOR x = __
1 XOR x = __
x XOR x = __

AND 0xF0 | 1111 0000
AND 0x3C | 0011 1100

OR 0xF0 | 1111 0000
OR 0x3C | 0011 1100

XOR 0xF0 | 1111 0000
XOR 0x3C | 0011 1100

NOT 0xF0 | NOT 1111 0000

Examples (Assume an 8-bit variable, v)

- Change the value of a bit in a register w/o affecting other bits
  - OR 0x3C = 0011 1100
  - Set the MSB to '1' w/o affecting other bits
  - v = v | __; or equivalently
  - v = v & ___;
  - flip the LS 4-bits w/o affecting other bits
  - v = v ^ ___;
  - v = v ~ ___;
  - v = v ^ ___;

You already knew the above ideas. It is just T1-T3.
4.9 Changing Register Bits

- Bitwise logic operations can be used to change the values of individual bits in registers without affecting the other bits in the register.
  - Set bit 0 of v to a ‘1’
    \[ v = v | ____; \]
  - Clear the 4 upper bits in v to ‘0’s
    \[ v = v & ____; \]
  - Flip bits 4 and 5 in v
    \[ v = v ^ {_______}; \]

Note: It is the programmer’s choice of writing the “mask” constant in binary, hex, or decimal. However, hex is usually preferable (avoids mistakes of missing a bit in binary and easier than converting to decimal).

4.10 Checking Register Bits

- To check for a given set of bits we use a bitwise-AND to isolate just those bits
  - The result will then have 0’s in the bit locations not of interest
  - The result will keep the bit values of interest

Examples
- Check if bit 7 of v = ‘1’
  \[ \text{if } ( (v \& 0x80) == 0x80) \{ \text{code} \} \text{ or } \]
  \[ \text{if } ____ \{ \text{code} \} \]
- Check if bit 2 of v = ‘0’
  \[ \text{if } ( (v \& 0x04) == 0x00) \{ \text{code} \} \text{ or } \]
  \[ \text{if } ____ \{ \text{code} \} \]
- Check if bit 2:0 of v = "101"
  \[ \text{if } ( (v \& 0b00000111) == 0b00000101) \{ \text{code} \} \]
- Check if bit 5-4 of v = "01"
  \[ \text{if } ( (v \& 0x30) == 0x10) \{ \text{code} \} \]

4.11 Short Notation for Operations

- In C, assignment statements of the form
  \[ x = x \text{ op } y; \]
  Can be shortened to
  \[ x \text{ op}= y; \]
- Example:
  \[ x = x + 1; \text{ can be written as } x += 1; \]
- The preceding operations can be written as
  \[ v| = 0x01; \]
  \[ v &= 0x0f; \]
  \[ v ^= 0b00110000; \]
• The Arduino Uno is a microcomputer development board based on the Atmel ATmega328P 8-bit processor.

• Most microcomputer manufacturers (Atmel, Freescale, etc.) produce small PC boards with their chips on them for engineers to experiment with and hopefully generate sales of the product.

• Arduino
  – An Italian company
  – They make numerous boards with different processors
  – Hardware and software are open source.
  – Very popular with hobbyists, due in a large part to their low cost.

• Arduino Unos can be stacked with "shield" boards to add additional capabilities (Ethernet, wireless, D/A, LCDs, sensors, motor control, etc.)
**ARDUINO PORTS AND PINS**

Flashback to Week 1

- Recall the computer interacts with any input or output (I/O) device by simply doing reads/writes to the memory locations (often called registers) in the I/O interfaces...
- The Arduino has many of these I/O interfaces all connected via the data bus.

Atmel ATmega328P

- The Arduino Uno is based on an Atmel ATmega328P 8-bit microcontroller
  - 32kb of FLASH ROM
  - ____ bytes of RAM
  - ____ I/O lines
  - 3 timer/counters
  - Serial/SPI/I\(^2\)C interfaces
  - A/D converter

Arduino Digital I/O

- ATmega328P has 23 pins on the chip that can be connected to other devices (switches, LEDs, motors, etc.)
  - Other members of the ATmega family may have more or less lines.
  - The Arduino Uno can make use of only 20 of these lines.
- Each pin can be used as a digital input or a digital output
  - **For output pins:** Your code determines what value ('1' or '0') appears
  - **For input pins:** Your code senses/reads what value another device is putting on the pin.

Main Point: Individual pins on the Arduino can be used as inputs OR outputs.
## Arduino Port/Pin Mapping

- Since computers usually deal with groups of 8-bits (a.k.a. a byte), all of the 20 I/O pins are split into **three 8-bit I/O ports (B, C and D)**
  - The avr-gcc software (SW) and the Arduino hardware use different names to refer to the bits within each port

<table>
<thead>
<tr>
<th>SW</th>
<th>Arduino</th>
<th>SW</th>
<th>Arduino</th>
<th>SW</th>
<th>Arduino</th>
</tr>
</thead>
<tbody>
<tr>
<td>PortB, bit0</td>
<td>DIG8</td>
<td>PortB, bit1</td>
<td>AN0</td>
<td>PortD, bit0</td>
<td>DIG0</td>
</tr>
<tr>
<td>PortB, bit1</td>
<td>DIG9</td>
<td>PortB, bit2</td>
<td>AN1</td>
<td>PortD, bit1</td>
<td>DIG1</td>
</tr>
<tr>
<td>PortB, bit2</td>
<td>DIG10</td>
<td>PortB, bit3</td>
<td>AN2</td>
<td>PortD, bit2</td>
<td>DIG2</td>
</tr>
<tr>
<td>PortB, bit3</td>
<td>DIG11</td>
<td>PortB, bit4</td>
<td>AN3</td>
<td>PortD, bit3</td>
<td>DIG3</td>
</tr>
<tr>
<td>PortB, bit4</td>
<td>DIG12</td>
<td>PortB, bit5</td>
<td>AN4</td>
<td>PortD, bit4</td>
<td>DIG4</td>
</tr>
<tr>
<td>PortB, bit5</td>
<td>DIG13</td>
<td>PortB, bit6</td>
<td>AN5</td>
<td>PortD, bit5</td>
<td>DIG5</td>
</tr>
<tr>
<td>PortB, bit6</td>
<td>Clock1 (don't use)</td>
<td>PortB, bit7</td>
<td>Clock2 (don't use)</td>
<td>PortD, bit6</td>
<td>DIG6</td>
</tr>
</tbody>
</table>

**Main Point:** Each pin has a name the software uses (Portx) and a name used on the Arduino circuit board (Anx or DIGx)

## Arduino Digital I/O

- The I/O ports (i.e. groups of pins) are the ____________ between your software program and the physical devices connected to the chip.
  - Your program is responsible for managing these ports (groups of I/O pins) in order to make things happen on the outside
- Most I/O pins in a port can be directly controlled by your software for "__________" OR be used for other specific HW functionality integrated on chip
  - PORTC0 can be used as a digital I/O OR as the Analog-to-Digital Conversion input: ADC0
  - PORTD0 can be used as digital I/O OR the serial communication receive input: RXD
- **We will discuss these other HW functions later...focus on digital I/O**
Controlling the pins of the Arduino to be digital inputs and outputs

**ARDUINO DIGITAL I/O**

### Overview

- In the next few slides you will learn
  - What your software needs to do to setup the pins for use as digital inputs and/or outputs
  - To set bits (to 1) and clear bits (to 0) using bitwise operations (AND, OR, NOT) to control individual I/O pins
  - How to do it in a readable syntax using shift operators (<<, >>)

- Don't be worried if it doesn't make sense the first time...listen, try to make sense of it, and ask a lot of questions.

### Controlling I/O Ports

- Each port (B, C, and D) has 3 registers in the µC associated with it that control the operation
  - Each bit in the register controls something about the corresponding I/O bit.
  - **DDRx** (Data direction register) [x=B,C,D...DDRB, DDRC, DDRD]  
  - Controls whether pins on the chip act as inputs or outputs.
  - Example: If DDRB[5] = __ -> PB5 (Port B bit 5 = DIG13 pin) will be used as __________
  - Example: If DDRB[5] = __ -> PB5 (Port B bit 5) will be used as __________
  - All I/O lines start out as inputs when the µC is reset or powered up.

- You'll write a program that sets these bits to 1's or 0's as necessary

### Register 1: Data Direction Register

- **DDRx** (Data direction register) [x=B,C,D...DDRB, DDRC, DDRD]  
  - Controls whether pins on the chip act as inputs or outputs.
  - Example: If DDRB[5] = __ -> PB5 (Port B bit 5 = DIG13 pin) will be used as __________
  - Example: If DDRB[5] = __ -> PB5 (Port B bit 5) will be used as __________
  - All I/O lines start out as inputs when the µC is reset or powered up.

Consider a leaf BLOWER / VACCUM. There must be a switch to select if you want it to blow (output) or produce suction (input)...DDR register is that "switch"
Register 2: PORT (Pin-Output) Register

- PORTx (Primarily used if port X is configured as an output)
  - When a pin is used as an output (DDRx[n] = 1), the corresponding bit in PORTx[n] determines the value/voltage of that pin.
  - E.g. By placing a ‘1’ in port B bit 5, pin PB5 will output a ________ voltage

<table>
<thead>
<tr>
<th>DDRD</th>
<th>DDBR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0011</td>
</tr>
</tbody>
</table>

PORTD

- PORTD7 PORTD6 PORTD5 PORTD4
- PORTD3 PORTD2 PORTD0

Main Point: For pins configured as outputs, the values you put in the PORT register will be the output voltages

Register 3: PIN (Pin-Input) Register

- PINx[n] (Used if PORT is configured as an input)
  - When a bit is an input (DDxn=___), getting the bit from PINxn reflects the current value at the corresponding input pin
  - The program doesn’t have to do anything special to read the digital signals into the PIN register, just use the register name
  - The action of referencing PINx causes all the signals to be acquired.
  - if(PIND == 0x00) // check if all the signals coming into port D are 0's
  - char val = PINB; // read and save all 8 signals coming into port B in a variable 'val'.
  - Programs must read the ________ bits in the PIN register, but can then use bitwise logical operations to check individual bits
  - If a port is an input but has no signal connected to it, it will “float” and could be read as either zero or one.

Review of Accessing Control Registers in C

- Control registers have names and act just like variables in a C program
- To put values into a control register you can assign to them like any C variable or perform bitwise operations
  - DDRD = 0xff; // 0b11111111 or 255
  - DDBR = 255;
  - PORTD |= 0x0c; // 0b11000000 or 192
  - PORTD |= 0b01110000;
- To read the value of a control register you can write expressions with them
  - unsigned char myvar = PIND; // grabs all 8-inputs on the port D
  - myvar = PINB & 0x0f; // you will see this grabs just the lower 4 inputs

Practice: Changing Register Bits

- Use your knowledge of the bitwise logic operations to change the values of individual bits in registers without affecting the other bits in the register.
  - Set DDRB, bit 3 to a '1'
    DDRB |= 0b10000000; // DDRB |= 0x80;
  - Clear the 2 upper bits in PORTC to ‘0’s
    PORTC &= 0x0f; // PORTC &= ~(0b11000000)
  - Flip bits 7 and 1 in DDRC
    DDRC ^= 0b10010000; // DDRC ^= 0x88;
  - Check if PIND, bit 4 = '1'
    if (PIND & 0x10) { code }
EXAMPLES

4.33 LED Outputs Review

- Recall we can connect LEDs to the outputs of a digital signal
  - The digital output value that will turn the LED on varies based on how we wire the LED
- Be sure to use a current-limiting resistor (few hundred ohms ~330-470 ohm)

Option 1
- LED is on when gate outputs '1'
Option 2
- LED is on when gate outputs '0'

Can be discrete gate or Arduino output pin

4.34 Switch & Button Input Review

- Recall: Switches/buttons alone do not produce 1's and 0's; they must be connected to voltage sources
- Preferred connection:
  - Connect one side of switch to GND (ground)
  - Connect other side of switch to digital input AND to a pull-up resistor (around 10Kohms) whose other side is connected to Vdd
- Switch/button will produce a:
  - 0 when pressed
  - 1 when open (not-pressed)

Preferred: Use a pullup resistor

Main Point: Buttons & switches should have GND connected to one side & a pull-up resistor on the other

4.35 Blinking an LED

- Hardware and software to make an LED connected to D7 blink

```c
#include<avr/io.h>
#include<util/delay.h>
int main()
{
    // Init. D7 to output
    DDRD |= 0x____;
    // Repeat forever while(1){
        // PD7 = 1 (LED on)
        PORTD |= 0x____;
        _delay_ms(500); // PD7 = 0 (LED off)
        PORTD &= ______;
        _delay_ms(500); // never reached
    return 0;
}
```

DDRD

PORTD
Turning an LED on/off with PB

- Hardware to turn an LED connected to D7 on/off when pressing a pushbutton connected to D4

Turning on an LED from a Button

- Note: When the button is pressed a __ is produced at the PD4 input

```c
#include<avr/io.h>

int main()
{
    // Init. D7 to output
    DDRD |= 0x80;
    // All pins start as input // on reset, so no need to // clear DDRO bit 4
    // Repeat forever while(1){ // Is PD4 pressed?
        if( (PIND _______) == 0){ // PD7 = 1 (LED on)
            PORTD |= 0x80;
        } else { // PD7 = 0 (LED off)
            PORTD &= ~(0x80);
        }
    } // Never reached
    return 0;
}
```

Pull Up Resistors

- Adding and wiring pull-up resistors for input buttons can be time consuming...
- Thankfully, each Arduino input bit has an optional ________ “pull-up resistor” associated with it.
  - If the pull-up is enabled, in the absence of an input signal, the input bit will be “pulled” up to a logical one.
  - The pull-up has no effect on the input if an active signal is attached.

Enabling Pull Up Resistors

- When DDRx bit n is '0' (i.e. a pin is used as input), the value in the ________ registers determines whether the internal pull-up is enabled
  - Remember, the PORT register is normally used when a pin is an output, but here its value helps enable the internal pull-up resistor
Using Internal Pull-up Resistors

- Let's simplify our wiring and use the internal pull-up resistors

Turning on an LED from a Button

- Note: When the button is pressed a '0' is produced at the PD4 input

#include<avr/io.h>
int main()
{
    // Init. D7 to output
    DDRD |= 0x80;
    // Enable pull-up on PD4
    PORTD |= 0x10;
    // Repeat forever while(1){
        // Is PD4 pressed?
        if((PIND & 0x10) == 0){
            // PD7 = 1 (LED on)
            PORTD |= 0x80;
        } else {
            // PD7 = 0 (LED off)
            PORTD &= ~(0x80);
        }
    }
    // Never reached
    return 0;
}

Code Read-ability Tip #1

- Try to replace hex and binary constants with shifted constants

FIDDLING WITH STYLE!
Shift Operations

- In C, operators '<<' and '>>' are the shift operators
  - << = ______ shift
  - >> = ______ shift
- Format: data << bit_places_to_shift_by
- Bits shifted out and dropped on one side
- Usually (but not always) 0's are shifted in on the other side

Another Example

- To get a 1 in a particular bit location it is easier to shift the constant 1 some number of places than try to think of the hex or binary constant

Putting it All Together

- Values for working with bits can be made using the ‘<<’ shift operator
  - OK: PORTB |= 0x20; Better: PORTB |= (1 << 5);
  - OK: DDRD |= 0x04; Better: DDRD |= (1 << 2);
- This makes the code more readable and your intention easier to understand...
- More examples
  - Set DDRC, bit 5: DDRC |= (1 << 5)
  - Invert PORTB, bit 2: PORTB ^= (1 << 2)
  - Clear PORTD, bit 3: PORTD &= (1 << 3)
    - Wrong! Why?
  - Clear PORTD, bit 3: PORTD &= (0 << 3)
    - Wrong! Why?
  - Clear PORTD, bit 3: PORTD &= ~(1 << 3)
    - Right! Why?

Clearing Bits...A Common Mistake

- When using the ‘&=’ operation to clear bits, remember to invert the bits.
- This won’t work to clear 3 to ‘0’
  - PORTD &= (1 << 3);
  - is the same as
  - PORTD &= 0b0001000;
  - which clears ___________ but bit 3
- Use the ‘~’ operator to complement the bits
  - PORTD &= ~(1 << 3);
  - is the same as
  - PORTD &= 0b11110111;
  - and now 3 gets cleared.
- And __________ use a mask of all 0’s
  - PORTD &= (0 << 3); // 0 shifted by any amount is 0 in all bit places
Setting/Clearing Multiple bits

- Can combine multiple bits into one defined value
  - `PORTB |= ((1 << 3) | (1 << 4) | (1 << 5));`
  - is the same as `PORTB |= 0b00111000`
  - `PORTB &= ~((1 << 3) | (1 << 4) | (1 << 5));`
  - is the same as `PORTB &= 0b11000111`;

### DEBOUNCING SWITCHES

#### Counting Presses

- Consider trying to build a system that counted button presses on PC2 (increment once per button press)
- We can write code to check if the button is pressed (==0) and then increment 'cnt'
- But remember, your code executes extremely ______...what will happen?

```
#include<avr/io.h>
int main() {
  PORTC |= (1 << PC2); int cnt = 0;
  while(1){
    char pressed = (PINC & 0x04);
    if( pressed == 0 ){
      cnt++;
      return 0;
    }
  }
  return 0;
}
```

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>cnt</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Waiting Through a Press

- Consider trying to build a system that counted button presses on PC2 (increment once per button press)
- We can write code to check if the button is pressed (==0) and then increment 'cnt'
- But remember, your code executes extremely fast...what will happen?

```
#include<avr/io.h>
int main() {
  PORTC |= (1 << PC2); int cnt = 0;
  while(1){
    char pressed = (PINC & 0x04);
    if( pressed == 0 ){
      while( (PINC & 0x04) == 0 ){
        cnt++;
      }
    }
    return 0;
  }
  return 0;
}
```

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>cnt</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Interfacing Mechanical Switches/Buttons

- Mechanical switches and buttons do not make solid, steady contact immediately after being pressed/changed
- For a short (few ms) time, “__________” will ensue and can cause spurious SW operation (one press of a button may look like multiple presses)
- Need to “debounce” switches with your software
  - Usually waiting around ________ from the first detection of a press will get you past the bouncing and into the stable period

Waiting Through a Press

- Consider trying to build a system that counted button presses on PC2 (increment once per button press)
- We can write code to check if the button is pressed (==0) and then increment 'cnt'
- But remember, your code executes extremely fast...what will happen?

What's Your Function

- Because there is a fair amount of work to do just to recognize a button press, you may want to extract those to functions you can call over and over again