Introduction to Microcontrollers

Registers

Goals

- Become familiar with boolean and bitwise logic
- Become familiar with other ways of representing numbers
- Learn about C structs and how they are used with the AVR IO library
- Know how to use GPIO with AVRs

Pop-Quiz

- Where can we find microcontrollers?
- What are the benefits of using microcontrollers?
- What are some of the problems with using microcontrollers?

Boolean Logic

- Logic operations
- True and False
  - Often true and false are represented by numbers instead
  - ‘0’ is false
  - Any number not equal to ‘0’ is true
- C/C++
  - AND
    - &&
    - If (a && b)
      - If a is true and b is true
  - OR
    - ||
    - If (a || b)
      - If a is true or b is true
  - NOT
    - !
    - Switch truth

AND

A	B	=
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	=
0	0	0
0	1	1
1	0	1
1	1	1

Pop-Quiz

Get a pen and paper then solve:

- 1 && 0 = ?
- 1 || 0 = ?
- 1 && !1 = ?
- !1 || 0 = ?

AND

A	B	=
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	=
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise Operators

- Operations are per bit
- C/C++
  - AND &
    - ‘1’ & ‘1’ = ‘1’
  - OR |
    - ‘1’ | ‘0’ = ‘1’
  - XOR ^
    - ‘1’ ^ ‘1’ = ‘0’
  - Left Shift <<
    - “01” << 1 = “10”
  - Right Shift >>
  - NOT ~
    - ~“01” = “10”

Truth Table

AND

A	B	=
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	=
0	0	0
0	1	1
1	0	1
1	1	1

XOR

A	B	=
0	0	0
0	1	1
1	0	1
1	0	0

4-bit operations

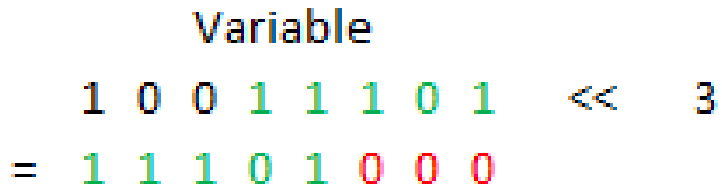
```
   1 0 0 1
 & 1 0 1 0
 = 1 0 0 0
```

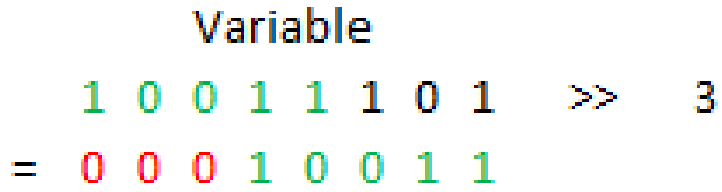
```
   1 0 0 1
 | 1 0 1 0
 = 1 0 1 1
```

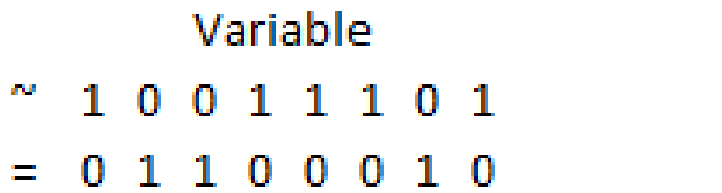
```
   1 0 0 1
 ^ 1 0 1 0
 = 0 0 1 1
```

Bitwise Operators

- Right and left shift
  - Pushes bits left or right by an amount
  - New values are replaced by ‘0’
    - In C/C++ types may affect behavior!
    - signed types may replace with ‘1’ on right shift if the number is negative
- Bitwise NOT
  - Simply flip the value of each bit







Pop-Quiz

- “0011” | “1010” = ?
- “0011” & “1010” = ?
- “0011” ^ “1010” = ?
- ~”0011” | “1010” = ?
- “0001” << 2 = ?
- “1000” >> 2 = ?
- (“0010” << 2) >> 4 = ?

AND

A	B	=
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	=
0	0	0
0	1	1
1	0	1
1	1	1

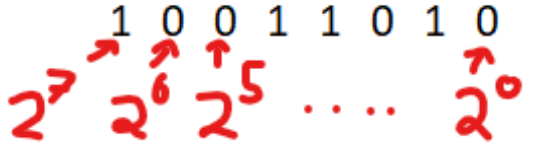
XOR

A	B	=
0	0	0
0	1	1
1	0	1
1	1	0

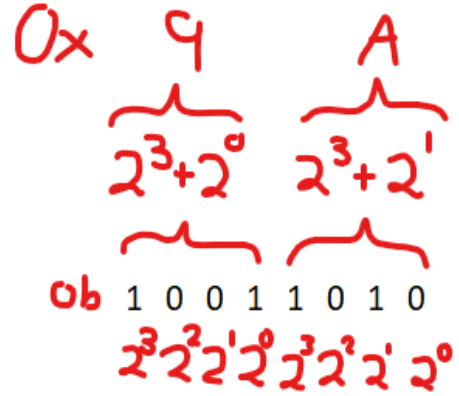
Representation of (unsigned) Numbers

- Decimal
  - C/C++ Prefix : None
  - Example value: 128
- Binary
  - C/C++ Prefix : 0b
  - Example value: 0b10000000
- Hexadecimal
  - C/C++ Prefix : 0x
  - Example value: 0x80
- We usually treat registers as unsigned values
  - Some registers use signed values!
    - Make sure you know what type of sign is used
    - Two's complement is by far the most used, but not always

Binary to decimal



Binary to hexadecimal



Dec	Bin	Hex
3	0b00000011	0x03
127	0b01111111	0x7F
170	0b10101010	0xAA

Pop-Quiz

- 0b0011 in decimal = ?
- 0b0011 in hexadecimal = ?
- 255 in hexadecimal = ?
- 255 in binary = ?

C/C++ Types

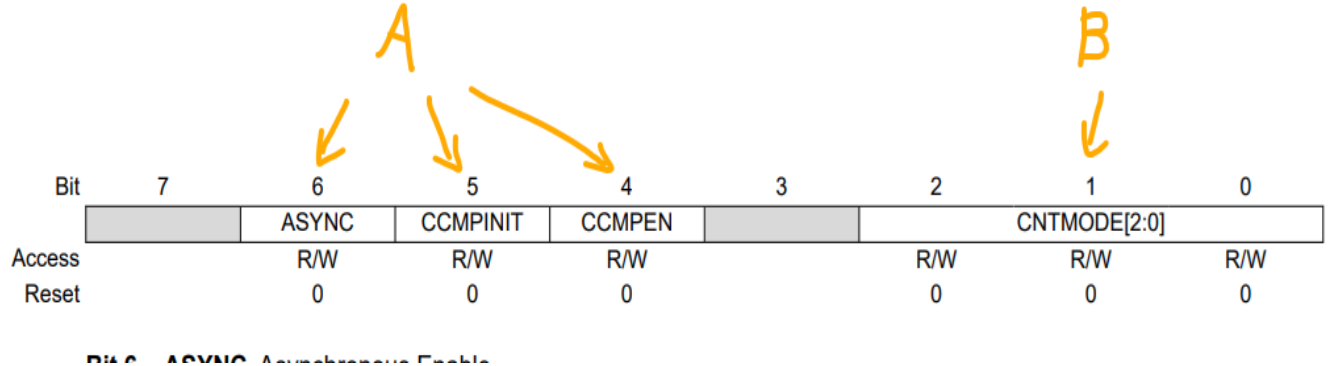
- C/C++ use types
- Types may have different sizes
  - Type sizes is dependent on what platform the code is compiled for
  - Int on a regular computer is 4 bytes long, in an AVR it is 2 bytes long
  - In an AVR float and double is the same size of 4 bytes
- There is a need to specify precise types!
  - int8_t - Signed integer of 8 bits
  - uint8_t - Unsigned integer of 8 bits
  - uint16_t - Unsigned integer of 16 bits
  - int32_t - Signed integer of 32 bits
  - Etc
- In AVRs, the databus is 8 bits wide
  - Therefore 8-bit variables are most efficient

Bitmasks

- The avr/io.h library uses bitmasks
  - (1 << 6) = 0x40 = 0b01000000 = PIN6_bm
- Many analogies
  - Filter
  - Stencil
  - Mask
- Bitmasks lets us affect only certain bits with our bitwise operators
  - 0b0000 | PIN1_bm = 0b0000 | 0b0010 = 0b0010

Group Configurations

- avr/io.h also introduces group configurations
  - Do not work like bitmasks as these may include zeros as information
- A: Bitmasks are used to manipulate single bits
- B: Group configurations are used to modify a group of bits
  - This includes writing both ‘1’s and ‘0’s
- Example
  - TCB_CNTMODE_INT_gc
- Cannot be used with SET/CLR/TGL logic



Bits 2:0 – CNTMODE[2:0] Timer Mode

Writing to this bit field selects the Timer mode.

Value	Name	Description
0x0	INT	Periodic Interrupt mode
0x1	TIMEOUT	Time-out Check mode
0x2	CAPT	Input Capture on Event m
0x3	FRQ	Input Capture Frequency
0x4	PW	Input Capture Pulse-Width
0x5	FRQPW	Input Capture Frequency
0x6	SINGLE	Single-Shot mode
0x7	PWM8	8-Bit PWM mode

Struct

- C/C++ type
- Same as class, but intended to be used for data structures
- Understanding structs properly requires a good understanding of pointers
  - Beyond the scope of this course

```
struct MY_STRUCT {
    uint8_t inductors;
    int16_t capacitors;
    float resistors;
};

MY_STRUCT my_parts;


my_parts.capacitors = 10;
my_parts.inductors++;
```

AVR Naming Convention

- Registers
  - PERIPHERAL_INSTANCE.REGISTER
- Bitmask
  - PERIPHERAL_BITNAME_bm
- Group configurations
  - PERIPHERAL_GROUPNAME_VALUE_gc

```
ADC0.CTRLC  = ADC_SAMPCAP_bm
            | ADC_PRESC_DIV4_gc;
```

30.5.3 Control C

Name: CTRLC
Offset: 0x02
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
		SAMPCAP	REFSEL[1:0]			PRESC[2:0]		
Access	R	R/W	R/W	R/W	R	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 6 – SAMPCAP Sample Capacitance Selection
This bit selects the sample capacitance, and hence, the input impedance. The best value is dependent on the reference voltage and the application's electrical properties.

Value	Description
0	Recommended for reference voltage values below 1V.
1	Reduced size of sampling capacitance. Recommended for higher reference voltages.

Bits 5:4 – REFSEL[1:0] Reference Selection
These bits select the voltage reference for the ADC.

Value	Name	Description
0x0	INTERNAL	Internal reference
0x1	VDD	V_{DD}
Other	-	Reserved.

Bits 2:0 – PRESC[2:0] Prescaler
These bits define the division factor from the peripheral clock (CLK_PER) to the ADC clock (CLK_ADC).

Value	Name	Description
0x0	DIV2	CLK_PER divided by 2
0x1	DIV4	CLK_PER divided by 4
0x2	DIV8	CLK_PER divided by 8
0x3	DIV16	CLK_PER divided by 16
0x4	DIV32	CLK_PER divided by 32
0x5	DIV64	CLK_PER divided by 64
0x6	DIV128	CLK_PER divided by 128
0x7	DIV256	CLK_PER divided by 256

Register Manipulation

- Most registers are “dumb”
  - You must perform bitwise operations to update their content correctly
- Some registers perform bitwise operations on other registers
  - xxxSET : Performs |= on register xxx
  - xxxCLR : Performs &=~ on register xxx
  - xxxTGL : Performs ^= on register xxx
- Using SET, CLR, TGL registers saves clock cycles!
  - No arithmetic is done by the Arithmetic Logic Unit (ALU)
  - Sometimes the compiler optimization is just as efficient

Register Manipulation

- The content or a register is simply a number
  - It is the context of what the specific bits does that is important
- These values are all equivalent
  - PORTC.OUTTGL = 0x40;
  - PORTC.OUTTGL = 0b01000000;
  - PORTC.OUTTGL = 64;
  - PORTC.OUTTGL = PIN6_bm;
- These operations are equivalent
  - PORTC.OUTTGL = PIN6_bm;
  - PORTC.OUT ^= PIN6_bm;

Common register operations

- X &= ~Y : CLR – Clear bits, other bits remain unaffected
  - 0b0101 &= ~0b0001 = 0b0100
- X |= Y: SET – Set bits, other bits remain unaffected
  - 0b0101 |= 0b0011 = 0b0111
- X ^= Y : TGL – Toggle bits, other bits remain unaffected
  - 0b0101 ^= 0b0011 = 0b0110
- X = Y : Write – All bits are set or cleared according to the value of Y
- X & Y : Check if bits in Y are ‘1’ in X
  - Usually to see if a specific bit is set in a register
    - If (X & (1<<1)) returns true if bit 1 in X is ‘1’
    - If (!(X & (1<<1)) returns true if bit 1 in X Is ‘0’
- X = A | B : Add bitmasks/groups to form a new update value
  - This is how we use multiple bitmasks/groups in a single register update
  - Important! All bitmasks which are not set are updated to ‘0’s
  - 0b1000 = 0b0001 | 0b0100 = 0b0101

Task

- The PORTx.IN register can be used to read a logical value on a GPIO
- PC7 is connected to the push button
  - When the button is pressed, PC7 is connected to GND
  - Enabling the internal pull-up resistor gives a default reference
    - Button not pushed : Voltage on PC7 = VDD – Logical ‘1’
    - Button is pushed : Voltage on PC7 = GND – Logical ‘0’

- Make a program that turns ON the LED on PC6 when the button on PC7 is pressed
  - When button is not pressed the LED should be OFF
  - The LED is ON when PC6 is logical ‘0’

- Hints
  - Refer to previous slide (Slide on github)
  - PIN7CTRL is used to enable the internal pull-up resistor

```
while (1)
{
    if ( <BUTTON_PRESSED_LOGIC> )
    {
        <TURN_LED_ON>
    }
    else
    {
        <TURN_LED_OFF>
    }
}
```

Next time

- UART
  - Transmit data to PC
- ADC
  - Convert analog voltages to digital values

- Any questions?