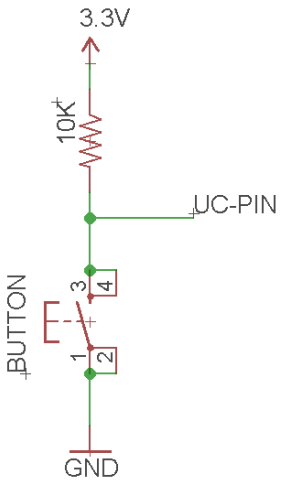
Lab 3 - Digital Input - Conditional Statements

Part 1 - Pull-ups, Inputs and Resets

How to connect a switch to a microcontroller input

When a pin on the microcontroller is configured as an input using the pinMode(pin,INPUT) command it puts the pin into a high impedance and is said to be floating if the pin is not connected to any other part of the circuit. A floating pin is due to the nature of the insulated gate of the FET not having any current path. If there is no current path then a static charge can build up on it. If there is no charge on a floating pin it will be interpreted as a logic 0 but with a little stray charge the pin will be interpreted as a logic 1. If we wish to hook up a simple SPST (Single Pole Single Throw) switch to the microcontroller we need to force the insolated gate to a known charge state. That is done by connecting the pin to ground or Vdd with a resistor. (Vdd is the voltage that the microcontroller runs at and typically will be either +3.3VDC or +5VDC but other voltages are possible). When a resistor ties an input pin to a voltage level it is called either a pull-down (when connected to ground) or pull-up (when connected to Vdd). In the case of a pull up a SPST (single pole single throw) switch can then be connected to the uC pin to ground. This way when the switch is open the uC pin sees the voltage at Vdd (logic 1) and when the switch is closed the uC pin sees 0V (ground). The circuit to the right is an example of a typical pull-up with a N.O. (Normally Open) SPST switch to ground. This circuit results in a logic 1 on the uC pin when the button is not pressed and a logic 0 when the button is pressed. Reversing the place of the resistor and switch in this circuit will create a pull-down equivalent and the logic will be reversed.



How to choose the resistor value for the pull-up or pull-down

Essentially any resistor value that is significantly greater than zero will work, however, thanks to ohms law and the RC time constants practical values are usually between 2K Ohms - 100K Ohms. Values too close to zero will cause excessive power draw since our power supply voltage (V) is fixed and the inverse relationship of resistance (R) to power (P):

$$P = \frac{V^2}{R}$$

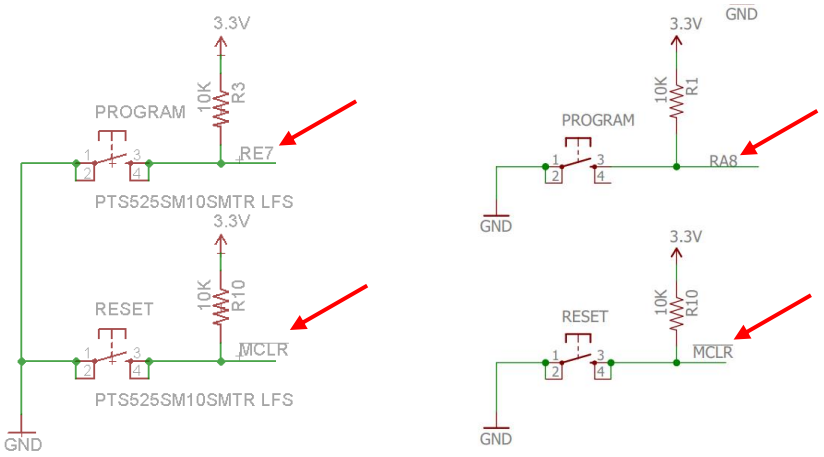
Since the insolating gate of the transistor acts like a charging a very small capacitor (gate capacitance C) when the switch is open excessive values of R will cause the time (approximately 5τ) that it takes the charge to build up on the FET gate to be slow and limit our frequency response (which for a human pushing a button can typically be ignored).

$$\tau = RC$$

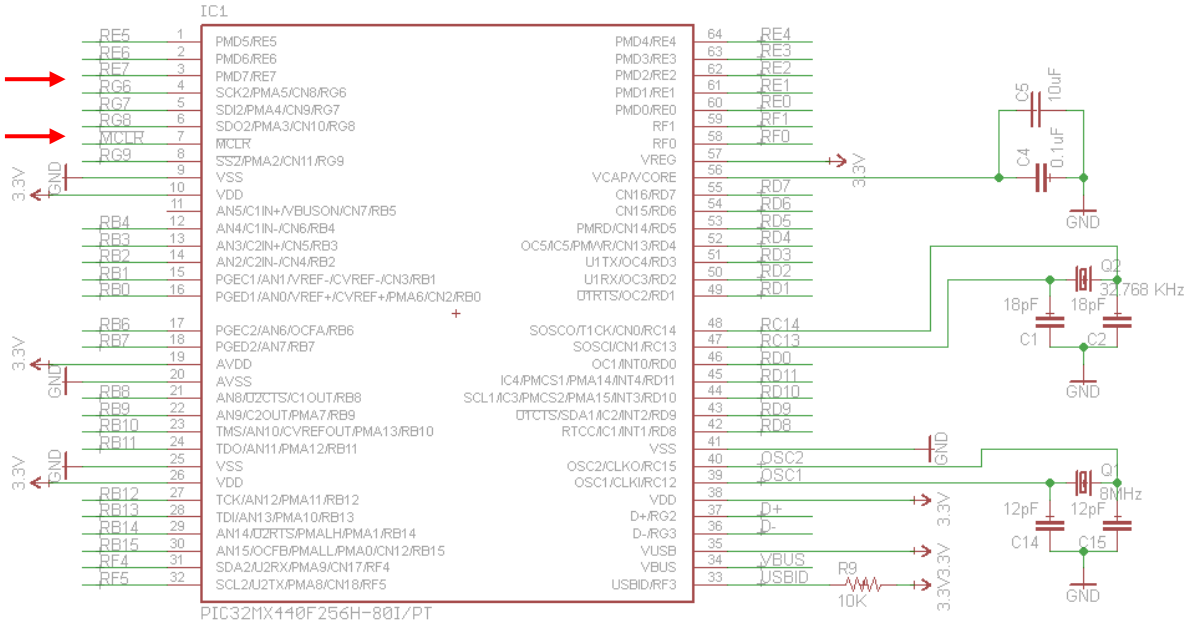
If values increase further then the resistors ability to provide a charge path begins to compete with the resistance to air and other charge sources could influence the input causing the logic state to be unstable when the button is not pressed.

RESET ($\overline{MCLR}$)

The two buttons on the Fubarino are examples of input pins of the uC being pulled-up with a SPST switch to ground. The RESET button on the Fubarino is tied to the $\overline{MCLR}$ pin of the PIC32 and the PRG (program) button is tied to PIC32 I/O port (pin RE7 of the Fubarino SD and pin RA8 of the Fubarino Mini). On the Fubarino SD PIC32 pin RE7 has been abstracted to pin 23. On the Fubarino Mini PIC32 pin RA8 has been abstracted to pin 16.



Fubarino SD Fubarino Mini



Fubarino SD (refer to the Fubarino Mini schematic in Lab 2 button connections)

When set to active low this causes the PIC32 to go into reset and restart the uC running at the beginning of memory. On all boards compatible with Arduino IDE this code at this location is called a bootloader. The Fubarino bootloader reads the state of the PRG button and if it is low causes the board to create a USB serial port and wait for a program to be uploaded from Arduino IDE. If the PRG button is high the bootloader code will just move on and run the user code.

To test the reset behavior an expanded blink sketch can be used to help us identify when we come out of reset. Modify the example blink sketch to that the led blinks three times one second on and one second off in the setup() and then in the loop() turns the led on for 0.25 seconds then off for 0.25 seconds. Your resultant program should look something like this:

```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);   // set the LED on
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // set the LED off
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, HIGH);   // set the LED on
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // set the LED off
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, HIGH);   // set the LED on
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // set the LED off
  delay(1000);                       // wait for a second
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // set the LED on
  delay(250);                        // wait for a 1/4 second
  digitalWrite(LED_BUILTIN, LOW);    // set the LED off
  delay(250);                        // wait for a 1/4 second
}
```

Compile your program and verify its operation. Test the program by pushing the reset button by itself and then again while holding down the PRG button. Note that the LED blinks slowly three times then goes into the loop when the PRG button is not pressed.

Check off

Call the instructor over to demonstrate your completed lab.

Be prepared to answer the following questions:

What usage other than getting to the bootloader might the RESET button be useful for?

Part 2: Digital input – digitalRead()

Up until this point we have only been using pins of the microcontroller as output. To use a microcontroller pin as an input we must first use the pinMode() to configure the pin. Next the pin state can be read and stored in a variable using the digitalRead(). The following code reads the button pin on the using the digitalRead() and saves the value read in the variable 'value'. Next it writes out the value stored in the 'value' variable to the PIN_OUT.

Copy the following code into the Arduino IDE then upload and test. Be sure to open the serial terminal and view the serial output from the program.

```
uint16_t PIN_OUT = 2; // pin  1 for Fubarino Mini, pin 21 for Fubarino SD
uint16_t PIN_IN  = 3; // pin 16 for Fubarino Mini, pin 23 for Fubarino SD

void setup() {
  Serial.begin(115200);
  pinMode(PIN_OUT, OUTPUT);  // Configure PIN_OUT as OUTPUT
  pinMode(PIN_IN, INPUT);    // Configure PIN_IN as INPUT
}

void loop() {
  uint8_t value;

  value = digitalRead(PIN_IN);  // read PIN_IN and store it in value
  digitalWrite(PIN_OUT, value); // write value out to PIN_OUT
  Serial.println(value, BIN);
  delay(100);
}
```

Check off

Call the instructor over and demonstrate the running program.

Be prepared to answer the following questions:

Why does LED go off when pushing the button and come on when it is not pressed?

Part 3: Logical and Bitwise Inversion

Change the following line in the previous program from this:

```
value = digitalRead(PIN_IN);  // read PIN_IN and store it in value
```

To this:

```
value = !digitalRead(PIN_IN);  // read PIN_IN and store it in value
```

This change introduces the logical not operator (!). This (!) operator logically inverts the value read with the digitalRead() and inverts it. So if a true (1) value is read a false (0) value is stored in the 'value' variable and vice versa.

Upload and test the program.

Next change this:

```
value = !digitalRead(PIN_IN);  // read PIN_IN and store it in value
```

To this:

```
value = ~digitalRead(PIN_IN);  // read PIN_IN and store it in value
```

Upload and look at the output on the serial terminal.

Check off

Call the instructor over and demonstrate the running program.

Be prepared to answer the following questions:

Why do you think the logical operator works and the bitwise operator does not?

Do the digitalRead() and digitalWrite() use bitwise or logical values?

Part 4: Blink Decision

In the first lab we saw that the on-board LED of the Fubarino is connected to an abstracted pin (21 on FubSD / 1 on FubMini) and that we were able to make it blink by setting it to an output. The PROGRAM button that is used to put the Fubarino into the bootloader can be used as an input in our applications as well.

The following program reads the state of the PROGRAM button and changes the blink rate depending on weather the button is pressed or not.

```
/*
  Blink with a Decision
 */
uint16_t PIN_OUT = 2; // pin  1 for Fubarino Mini, pin 21 for Fubarino SD
uint16_t PIN_IN  = 3; // pin 16 for Fubarino Mini, pin 23 for Fubarino SD

void setup() {
  pinMode(PIN_OUT, OUTPUT);
  pinMode(PIN_IN, INPUT);
}

void loop() {
  digitalWrite(PIN_OUT, !digitalRead(PIN_OUT)); // toggle the LED
  if( digitalRead(PIN_IN) == LOW ) {            // Made decision
    delay(200);                                 // True condition
  }
  else {
    delay(1000);                                // False condition
  }
}
```

Analysis

Two global variables called PIN_OUT and PIN_IN have been created to make it simple to change the input and output pins used in this program. The newly introduced function digitalRead allows us to check the logic state of a pin. It can be used to detect both INPUT and OUTPUT pins. If an OUTPUT pin is read the last value written to it is will be returned. If an INPUT pin is read the currently presented voltage on the pin will be read. The following line of code now incorporates the digitalRead with the digitalWrite to which allows us to toggle the pin state by use of the ! (logical not) operator.

```
  digitalWrite(PIN_OUT, !digitalRead(PIN_OUT)); // toggle the LED
```

The next line of code is very different from anything we have seen up until now and uses an if statement to check the result of the digitalRead() to make a decision.

```
  if( digitalRead(PIN_IN) == LOW ) {            // Made decision
```

The statement between the parenthesis in the above statement if true will run the first statement and if false will run the statement after the else clause. The double equals (==) will evaluate the whether the result of the digitalRead function is equivalent to the constant LOW.

Check off

Call the instructor over for a check off.

Be prepared to answer the following questions:

What is the purpose of the ! operator?

Part 5: Blink Decision with pull-ups

Change the input and output pin to use an external LED and an external switch with a pull-up resistor.

Check off

Call the instructor over for a check off.

Be prepared to change the pull up to a pull a pull down and re-demonstrate.

Part 6: Blink Decision sequence

Wire up four LED's to pins 0-4 of the Fubarino. Modify the program so that it does nothing until a button is pressed. When the button is pressed, successively light up one LED until all four are lit with a one second delay between each light. Then wait one more second and shut off all LED's. If the button is pressed again it will repeat the behavior described above.

Check off

When the circuit and program are working, call the instructor over for a check off.

Part 7: Shift Right Operator

The following functions are useful for setting reading and writing of four pins in a group (a nibble). The next remaining parts of this lab will use these functions, you can add them to the top of your program.

```
void nibbleInit(uint8_t pin_b3, uint8_t pin_b2, uint8_t pin_b1, uint8_t
pin_b0, uint8_t mode)
{
  pinMode(pin_b3, mode);
  pinMode(pin_b2, mode);
  pinMode(pin_b1, mode);
  pinMode(pin_b0, mode);
}
uint8_t pinsToNibble(uint8_t pin_b3, uint8_t pin_b2, uint8_t pin_b1, uint8_t
pin_b0)
{
  uint8_t nibble;
  nibble = digitalRead(pin_b3) << 3 |
           digitalRead(pin_b2) << 2 |
           digitalRead(pin_b1) << 1 |
           digitalRead(pin_b0) << 0;
  return nibble;
}

void nibbleToPins(uint8_t in, uint8_t pin_b3, uint8_t pin_b2, uint8_t pin_b1,
uint8_t pin_b0) {
  // use bitwise & operator to create logic condition that is
  // used to light bits as if in the same nibble.
  digitalWrite(pin_b0, in & 0b00000001);
  digitalWrite(pin_b1, in & 0b00000010);
  digitalWrite(pin_b2, in & 0b00000100);
  digitalWrite(pin_b3, in & 0b00001000);
}
```

This sketch demonstrates the shift operator. With LED's still hooked up to outputs 0-3. Test the following sketch:

```
uint8_t s; // shift bits

void setup() {
  Serial.begin(115200);
  nibbleInit(0,1,2,3, OUTPUT);
}

void loop() {
  Serial.println("looping");
  s = 0x80;
  while( s != 0x00 )
  {
    nibbleToPins(s,0,1,2,3);
    s = s >> 1;
    delay(1000);
  }
}
```

Check off

Note that the time that the loop runs is about eight seconds, and that LED's are only lit for four out of the eight seconds of the loop. What can be inferred from this?

Call the instructor over for check off when you get it working.

Part 8: Shift Left Operator

Change the direction of the shifting from left to right and modify the while loop logical condition so that **while loop** only executes four times instead of eight as in the previous sections.

Check off

Call the instructor over for check off when you get it working.

Part 9: Bitwise operators

This lab demonstrates bitwise logic operators by reading two groups of four switches and preforming a logical operation on them. The result of the operation is displayed on LED's and in the serial terminal.

If you look at the setup() you can see that pins 0-3 are configured as output. This is where we are going to hook up the LED's. Pins 4-7 and 8-11 are configured as inputs. These will be the switch inputs for our program.

```
void setup() {
  Serial.begin(115200);
  nibbleInit(0,  1,  2,  3, OUTPUT);
  nibbleInit(4,  5,  6,  7, INPUT);
  nibbleInit(8,  9, 10, 11, INPUT);
  delay(3000);
  Serial.println("starting...");
}

void loop() {
  uint8_t a, b;
  uint8_t accumulator;

  a = pinsToNibble(4,  5,  6,  7);
  b = pinsToNibble(8,  9, 10, 11);
  accumulator = a & b;
  nibbleToPins(accumulator, 0, 1, 2, 3);

  Serial.print(a, BIN);
  Serial.print(" & ");
  Serial.print(b, BIN);
  Serial.print(" = ");
  Serial.print(accumulator, BIN);
  Serial.println(" ");
  delay(100);
}
```

Test the program by setting inputs 4-11 to binary values and verify the output on the LED's.

Check off

Modify the program to use a bitwise logic or operator. Verify it works then call the instructor over to demonstrate the program.

Part 10: Logical Operators

The following code demonstrates logical operators.

```
void setup() {
  Serial.begin(115200);
  nibbleInit(0,  1,  2,  3, OUTPUT);
  nibbleInit(4,  5,  6,  7, INPUT);
  nibbleInit(8,  9, 10, 11, INPUT);
  delay(3000);
  Serial.println("starting...");
}

void loop() {
  uint8_t a, b;
  uint8_t accumulator;

  a = digitalRead(4);
  b = digitalRead(8);
  Serial.print("a=");
  Serial.print(a, BIN);
  Serial.print(" b=");
  Serial.print(b, BIN);

  Serial.print (" a&&b=");
  if( a && b ) {
    digitalWrite(0, HIGH);
    Serial.print ("HIGH");
  }
  else {
    digitalWrite(0, LOW);
    Serial.print("LOW ");
  }
  Serial.println(".");
  delay(100);
}
```

Check off

Verify the program works then call the instructor over to demonstrate the program.

Part 10: Logical Expanded

Write a program that has two inputs A and B and four outputs W, X, Y and Z. Make the outputs have the following behavior:

W = NOT A

X = A NAND B

Y = A OR B

Z = A XOR B

Call the instructor over for check off when you get it working.

Homework

Create a new function called byteToPins() that will output a byte in binary on 8 pins of the Fubarino. Use this function to display a count from 0x00 to 0xFF in binary on the LEDs connected to the Fubarino as well as print out the value in hexadecimal in the serial terminal. As usual this homework is not collected, but if you can get this program to work then you have a good understanding of the material, if you cannot seek help.

Advanced: Find the gate capacitance of a typical GPIO input on a PIC32 uC by looking it up in the datasheet. This is challenging.