Arduino Application: Buttons for user input

ME 120
Mechanical and Materials Engineering
Portland State University
http://web.cecs.pdx.edu/~me120
User input features of the fan

- Potentiometer for speed control
  - Continually variable input makes sense for speed control
  - Previously discussed
- Start/stop
  - Could use a conventional power switch
  - Push button (momentary) switch
- Lock or limit rotation angle
  - Button click to hold/release fan in one position
  - Potentiometer to set range limit
Conventional on/off switch

• Basic light switch or rocker switch
  - Makes or breaks connection to power
  - Switch stays in position: On or Off
  - Toggle position indicates the state
  - NOT in the Arduino Inventors Kit
How does a button work?

• Simple switch schematic
• Use DMM to measure open/closed circuit
• Map the pin states
Measure Open and Closed Circuits

<table>
<thead>
<tr>
<th>Connect Pins</th>
<th>Measured Resistance (Ω)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>When not pressed</td>
</tr>
<tr>
<td>1 and 2</td>
<td></td>
</tr>
<tr>
<td>1 and 3</td>
<td></td>
</tr>
<tr>
<td>1 and 4</td>
<td></td>
</tr>
<tr>
<td>2 and 3</td>
<td></td>
</tr>
</tbody>
</table>
# Measure Open and Closed Circuits

<table>
<thead>
<tr>
<th>Connect Pins</th>
<th>Measured Resistance (Ω)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>When not pressed</td>
<td>When pressed</td>
</tr>
<tr>
<td>1 and 2</td>
<td>∞</td>
<td>0</td>
</tr>
<tr>
<td>1 and 3</td>
<td>∞</td>
<td>0</td>
</tr>
<tr>
<td>1 and 4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 and 3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

![Top View](image1.png) ![Bottom View](image2.png)

- **Side 1** Two connected pins
- **Side 2** Two connected pins when pressed

---

**ME 120: Button inputs**
Push Button Switches

• A momentary button is a “Biased Switch”
• Pushing the button changes state
• State is reversed (return to biased position) when button is released
• Two types
  • NO: normally open
  • NC: normally closed

Normally Open          Normally Closed
Momentary or push-button switches

- Normally open
  - Electrical contact is made when button is pressed
- Normally closed
  - Electrical contact is broken when button is pressed
- Internal spring returns button to its un-pressed state

Image from sparkfun.com
Putting buttons into action

1. Build the circuit: same one is used for all examples
   a. Test with LED on/off
   b. LED is only controlled by the button, not by Arduino code

2. Create a “wait to start” button
   a. Simplest button implementation
   b. Execution is blocked while waiting for a button click

3. Use an interrupt handler
   a. Most sophisticated: Doesn’t block execution while waiting for button input
   b. Most sophisticated: Requires good understanding of coding
   c. Requires “de-bouncing”
   d. Not too hard to use as a black box
Momentary Button and LED Circuit

• Digital input with a *pull-down resistor*
  
  ❖ When switch is open (button not pressed):
    ‣ Digital input pin is tied to ground
    ‣ No current flows, so there is no voltage difference from input pin to ground
    ‣ Reading on digital input is LOW
  
  ❖ When switch is closed (button is pressed):
    ‣ Current flows from 5V to ground, causing LED to light up.
    ‣ The 330Ω resistor limits the current draw by the input pin.
    ‣ The 10k resistor causes a large voltage drop between 5V and ground, which causes the digital input pin to be closer to 5V.
    ‣ Reading on digital input is HIGH
Usually we do not include an LED directly in the button circuit. The following diagrams show plan button circuits with pull-up and pull-down resistors. In these applications, the pull-up or pull-down resistors should be 10k. Refer to Lady Ada Tutorial #5:

- http://www.ladyada.net/learn/arduino/lesson5.html
Programs for the LED/Button Circuit

• Continuous monitor of button state
  ❖ Program is completely occupied by monitoring the button
  ❖ Used as a demonstration — not practically useful

• Wait for button input
• Interrupt Handler

All three programs use the same electrical circuit
Momentary Button and LED Circuit
Momentary Button and LED Circuit

```
5V

LED

Push-button switch

330 Ω

10 kΩ

Digital input pin
```
Continuous monitor of button state

```cpp
int button_pin = 2;  // pin used to read the button

void setup() {
    pinMode(button_pin, INPUT);
    Serial.begin(9600);  // Button state is sent to host
}

void loop() {
    int button;
    button = digitalRead(button_pin);
    
    if (button == HIGH) {
        Serial.println("on");
    } else {
        Serial.println("off");
    }
    
Serial monitor shows a continuous stream of “on” or “off”
```

This program does not control the LED
Programs for the LED/Button Circuit

• Continuous monitor of button state
  ❖ Program is completely occupied by monitoring the button
  ❖ Used as a demonstration — not practically useful

• Wait for button input
  ❖ Blocks execution while waiting
  ❖ May be useful as a start button

• Interrupt Handler

All three programs use the same electrical circuit
While loop

Syntax: 

```java
while (expression) {
    statement block
}
```

While loops will loop continuously, and infinitely, until the expression inside the parenthesis becomes false.

- Something must change the tested variable (= variable in the expression), or the while loop will never exit.
- This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.
int button_pin = 2; // pin used to read the button

void setup() {
    int start_click = LOW; // Initial state: no click yet
    pinMode(button_pin, INPUT);
    Serial.begin(9600);

    while (start_click == LOW) {
        start_click = digitalRead(button_pin);
        Serial.println("Waiting for button press");
    }
}

void loop() {
    int button;

    button = digitalRead(button_pin);
    if (button == HIGH) {
        Serial.println("on");
    } else {
        Serial.println("off");
    }
}
Wait for button input

int button_pin = 2;  // pin used to read the button

void setup() {
    int start_click = LOW;  // Initial state: no click yet
    pinMode( button_pin, INPUT);
    Serial.begin(9600);
    while ( !start_click ) {
        start_click = digitalRead( button_pin );
        Serial.println("Waiting for button press");
    }
}

void loop() {
    int button;

    button = digitalRead( button_pin );
    if ( button == HIGH ) {
        Serial.println("on");
    } else {
        Serial.println("off");
    }
}
# Digital - Review

<table>
<thead>
<tr>
<th>Value</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>Pin level</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>Voltage</td>
<td>5 V</td>
<td>0 V</td>
</tr>
</tbody>
</table>
Programs for the LED/Button Circuit

• Continuous monitor of button state
  ❖ Program is completely occupied by monitoring the button
  ❖ Used as a demonstration — not practically useful

• Wait for button input
  ❖ Blocks execution while waiting
  ❖ May be useful as a start button

• Interrupt Handler
  ❖ Most versatile
  ❖ Does not block execution
  ❖ Interrupt is used to change a flag that indicates state
  ❖ Regular code in loop function checks the state of the flag

All three programs use the same electrical circuit
Use push button as a switch

Now we would like to use the push button as a switch. We push once: it would show on. We push once more, it would show off. Etc…

→ Use of interrupts
**Interrupt**

`attachInterrupt(interrupt, ISR, mode)`

**Interrupt: either 0 or 1.**
- If digital pin 2 is used, the interrupt number is 0
- If digital pin 3 is used, the interrupt number is 1

**ISR: Interrupt Service Routine.**
- It is a user-defined function that is called when the interrupt occurs.
- It needs to be short.
- It cannot contain the functions `delay()`.
- It cannot have any parameters and cannot return any value.
- Therefore it is of the form `void name_of_ISR (){ block of code }`

**Mode: defines when interrupt should be triggered**
- Values can be LOW, CHANGE, RISING, FALLING
Interrupt handler for button input

int button_interrupt = 0;    // Interrupt 0 is on pin 2 !!
int toggle_on = false;       // Button click switches state

void setup() {
    Serial.begin(9600);
    attachInterrupt(button_interrupt, handle_click, RISING);  // Register handler
}

void loop() {
    if (toggle_on) {
        Serial.println("on");
    } else {
        Serial.println("off");
    }
}

void handle_click() {

    static unsigned long last_interrupt_time = 0;       // Zero only at start
    unsigned long interrupt_time = millis();             // Read the clock

    if (interrupt_time - last_interrupt_time > 200) {    // Ignore when < 200 msec
        toggle_on = !toggle_on;
    }

    last_interrupt_time = interrupt_time;
}
Interrupt handler for button input

```c
int button_interrupt = 0;  // Interrupt 0 is on pin 2 !!
int toggle_on = false;       // Button click switches state

void setup() {
    Serial.begin(9600);
    attachInterrupt(button_interrupt, handle_click, RISING);  // Register handler
}

void loop() {
    if (toggle_on) {
        Serial.println("on");
    } else {
        Serial.println("off");
    }
}

void handle_click() {
    static unsigned long last_interrupt_time = 0;       // Zero only at start
    unsigned long interrupt_time = millis();            // Read the clock

    if (interrupt_time - last_interrupt_time > 200) {   // Ignore when < 200 msec
        toggle_on = !toggle_on;
    }

    last_interrupt_time = interrupt_time;
}
```

button_interrupt is the ID or number of the interrupt. It must be 0 or 1

Interrupt handler must be registered when program starts

A RISING interrupt occurs when the pin changes from LOW to HIGH

The interrupt handler, handle_click, is a user-written function that is called when an interrupt is detected
Interrupt handler for button input

```c
int button_interrupt = 0;    //  Interrupt 0 is on pin 2 !!
int toggle_on = false;       //  Button click switches state

void setup() {    
  Serial.begin(9600);
  attachInterrupt( button_interrupt, handle_click, RISING);  // Register handler
}

void loop() { 
  if ( toggle_on ) {   
    Serial.println("on");
  } else {    
    Serial.println("off");
  }
}

void handle_click() { 
  static unsigned long last_interrupt_time = 0;       //  Zero only at start
  unsigned long interrupt_time = millis();            //  Read the clock

  if ( interrupt_time - last_interrupt_time > 200 ) {  // Ignore when < 200 msec
    toggle_on = !toggle_on;
  }
  last_interrupt_time = interrupt_time;
}
```

**toggle_on** is a global variable that remembers the “state”. It is either true or false (1 or 0).

The **loop()** function only checks the state of **toggle_on**. The value of **toggle_on** is set in the interrupt handler, **handle_click**.

The value of **toggle_on** is flipped only when a **true** interrupt even occurs. De-bouncing is described in the next slide.
Interrupt handler for button input

```c
int button_interrupt = 0; // Interrupt 0 is on pin 2 !
int toggle_on = false;    // Button click switches state

void setup() {
  Serial.begin(9600);
  attachInterrupt( button_interrupt, handle_click, RISING); // Register handler
}

void loop() {
  if (toggle_on) {
    Serial.println("on");
  } else {
    Serial.println("off");
  }
}

void handle_click() {
  static unsigned long last_interrupt_time = 0; // Zero only at start
  unsigned long interrupt_time = millis();       // Clock time when current interrupt occurs

  if (interrupt_time - last_interrupt_time > 200) { // Ignore when <$ 200 msec
    toggle_on = !toggle_on;
  }
  last_interrupt_time = interrupt_time;           // Save current time as the new “last” time
}
```

Value of a static variable is always retained

Use `long`: the time value in milliseconds can become large

Ignore events that occur in less than 200 milliseconds from each other. These are likely to be mechanical bounces.
Other references

• Ladyada tutorial
  ❖ Excellent and detailed
  ❖ http://www.ladyada.net/learn/arduino/lesson5.html

• Arduino reference
  ❖ Minimal explanation
  ❖ Using interrupts
    ‣ http://www.arduino.cc/en/Reference/AttachInterrupt