Button Input: On/off state change

Desktop fan project
EAS 199A, Fall 2011

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

Image from sparkfun.com
Image from lowes.com
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>

Sketch Connections:

Data from Measurements:

<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>
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

Putting buttons into action

1. Build the circuit: same one is used for all examples
   - Test with LED on/off
   - LED is only controlled by the button, not by Arduino code
2. Create a “wait to start” button
   - Simplest button implementation
   - Execution is blocked while waiting for a button click
3. Use an interrupt handler
   - Most sophisticated: Don’t block execution while waiting for button input
   - Most sophisticated: Requires good understanding of coding
   - Requires “de-bouncing”
   - 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 10k resistor limits the current draw by the input pin.
  - The 330Ω 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

Programs for the LED/Button Circuit

1. **Continuous monitor of button state**
   - Program is completely occupied by monitoring the button
   - Used as a demonstration — not practically useful

2. **Wait for button input**

3. **Interrupt Handler**

All three programs use the same electrical circuit

```c
int button_pin = 4; // 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");
  }
}
```

This program does not control the LED
1. Continuous monitor of button state
   - Program is completely occupied by monitoring the button
   - Used as a demonstration — not practically useful

2. Wait for button input
   - Blocks execution while waiting
   - May be useful as a start button

3. 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

---

```
# Programs for the LED/Button Circuit

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

2. Wait for button input
   - Blocks execution while waiting
   - May be useful as a start button

3. 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

---

```

```
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

```cpp
data int button_interrupt = 0; // Interrupt 0 is on pin 2!!
data 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;
}
```

- Value of a static variable is always retained
- Use long: the time value in milliseconds can become large
- Clock time when current interrupt occurs
- Save current time as the new “last” time
- Ignore events that occur in less than 200 msec 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](http://www.ladyada.net/learn/arduino/lesson5.html)

**Arduino reference**
- Minimal explanation
- Using interrupts