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

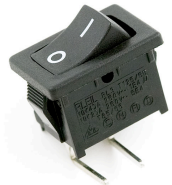


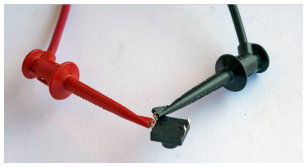
Image from sparkfun.com

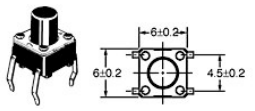
Image from lowes.com

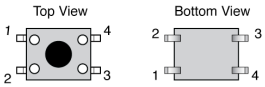
How does a button work?

- Simple switch schematic
- Use DMM to measure open/closed circuit
- Map the pin states

Measure Open and Closed Circuits







	Measured Resistance (Ω)	
Connect Pins	When not pressed	When pressed
1 and 2		
1 and 3		
1 and 4		
2 and 3		

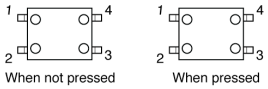
Measure Open and Closed Circuits

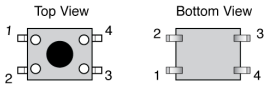
Data from Measurements:

	Measured Resistance (Ω)	
Connect Pins	When not pressed	When pressed
1 and 2		
1 and 3		
1 and 4		
2 and 3		

Sketch Connections:

Draw lines between connectors





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

Open

Closed

Image from sparkfun.com

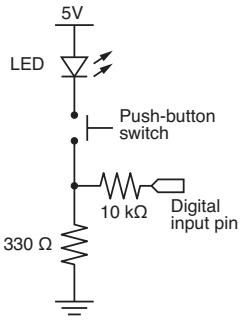
Putting buttons into action

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

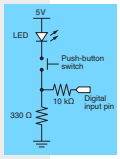
1. Continuous monitor of button state
   - Program is completely occupied by monitoring the button
   - Used as a demonstration — not practically useful
2. Wait for button input
3. Interrupt Handler

All three programs use the same electrical circuit

Continuous monitor of button state

```
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");
  }
}
```

Serial monitor shows a continuous stream of "on" or "off"



This program *does not* control the LED

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

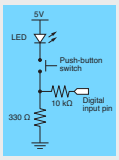
All three programs use the same electrical circuit

Wait for button input

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

while loop continues as long as start_click is FALSE

Same loop() function as before



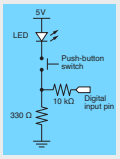
Programs for the LED/Button Circuit

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 sate of the flag

All three programs use the same electrical circuit

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

```
int  button_interrupt = 0;
int  toggle_on = false;

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

    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 must be registered when program starts

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

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

```
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

```
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;

  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

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

Save current time as the new "last" time

Other references

Ladyada tutorial

- Excellent and detailed
- http://www.ladyada.net/learn/arduino/lesson5.html

Arduino reference

- Minimal explanation
  - http://www.arduino.cc/en/Tutorial/Button
- Using interrupts
  - http://www.uchobby.com/index.php/2007/11/24/arduino-interrupts/
  - http://www.arduino.cc/en/Reference/AttachInterrupt