Home Downloads Search Forum

Home | Users | Search | FAQ

Gammon Forum

Username:

Register forum user name

Password:

Forgotten password?

Log on

See www.mushclient.com/spam for dealing with forum spam. Please read the MUSHclient FAQ!

Entire forum
Electronics
Microprocessors
Switches tutorial

Switches tutorial

Postings by administrators only.

Refresh page

Posted by **Nick Gammon** Australia (20,402 posts) bio *Forum Administrator*

Date Wed 27 Feb 2013 05:17 AM (UTC)

Amended on Sun 28 Jun 2015 04:13 AM (UTC) by Nick Gammon

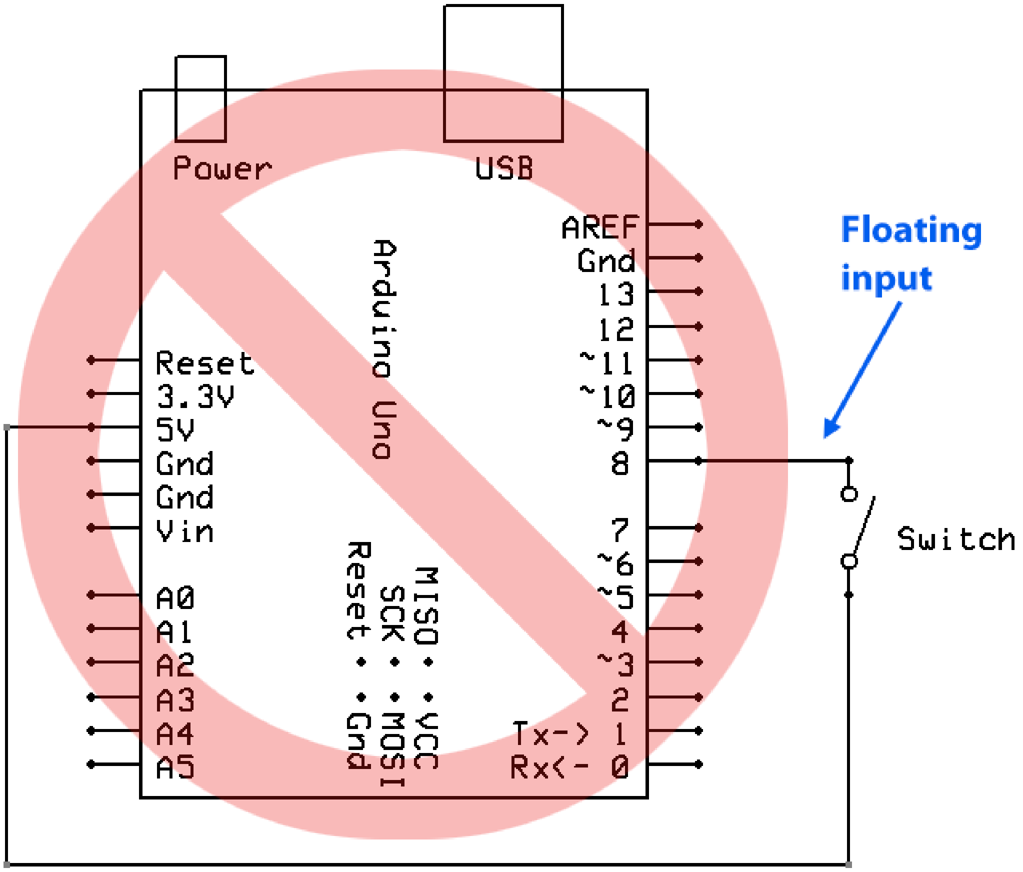
Message This post introduces the basics of managing switches on a microprocessor, such as the Arduino.

> This page can be quickly reached from the link: http://www.gammon.com.au/switches

By "switch" I am referring to mechanical switches like push-buttons, or toggle switches. We assume that you want to find out in the program code if the switch is pressed, or not.

Wiring the switch

Beginners tend to wire their switch like this (not recommended):



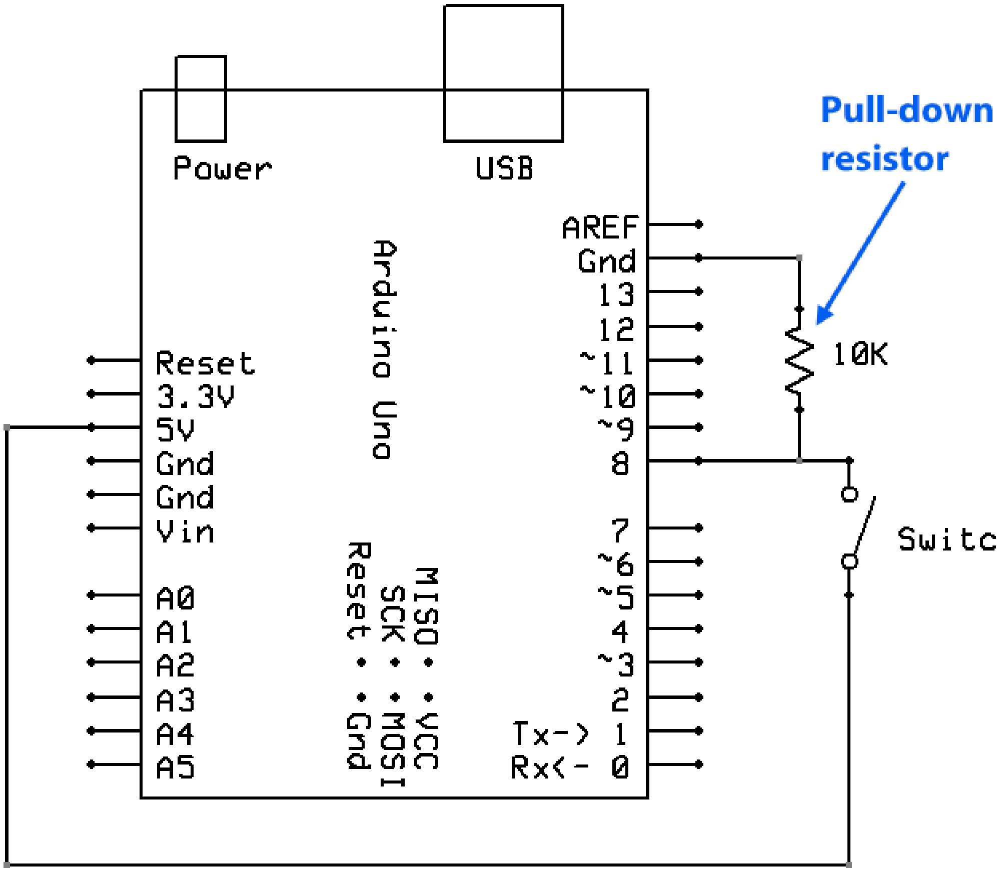
This will not work properly because the switch input (digital pin 8 in this example) is "floating" if the switch is open.

The switch is definitely +5V when closed (and thus returns HIGH when tested with digitalRead) but is **not necessarily** at 0V when open. In fact, waving your hand over the processor board is likely to generate enough stray voltages to make it look like the switch is being opened and closed.

There are three main methods for getting reliable operation, described below ...

Pull-down resistor

The circuit below adds a 10K "pull-down" resistor (the exact value doesn't matter, as long as it is not too low).



This resistor "weakly" pulls the switch down to ground, so that if the switch is open, it will have 0V on it (through the resistor) and thus will register LOW if not pressed, and HIGH if pressed.

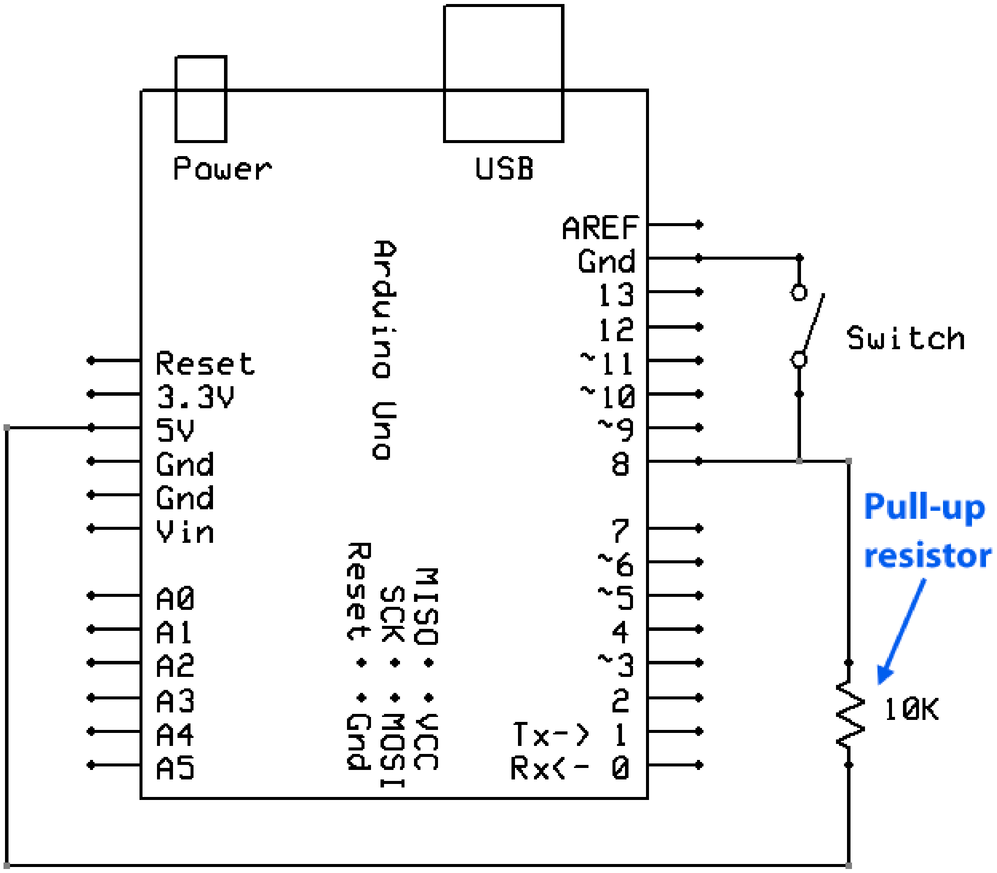
Example code:

```
const byte switchPin = 8;

void setup ()
  {
  Serial.begin (115200);
  pinMode (switchPin, INPUT);
  }  // end of setup

void loop ()
  {
  if (digitalRead (switchPin) == HIGH)
     {
     Serial.println ("Switch closed.");
     delay (1000);
     } // end if switchState is HIGH

  // other code here ...

  }  // end of loop
```

This example has a very simple "debounce" implemented via a lengthy delay of one second. More about debouncing later.

Pull-up resistor

The circuit below adds a 10K "pull-up" resistor (the exact value doesn't matter, as long as it is not too low).



This resistor "weakly" pulls the switch up to +5V, so that if the switch is open, it will have 5V on it (through the resistor) and thus will register HIGH if not pressed, and LOW if pressed.

Example code:

```
const byte switchPin = 8;

void setup ()
  {
  Serial.begin (115200);
  pinMode (switchPin, INPUT);
  }  // end of setup

void loop ()
  {
  if (digitalRead (switchPin) == LOW)
     {
     Serial.println ("Switch closed.");
     delay (1000);
     }  // end if switchState is LOW

  // other code here ...

  }  // end of loop
```

The reason for choosing a fairly high resistance is that current flows through the resistor most of the time, the amount being given by Ohm's Law:

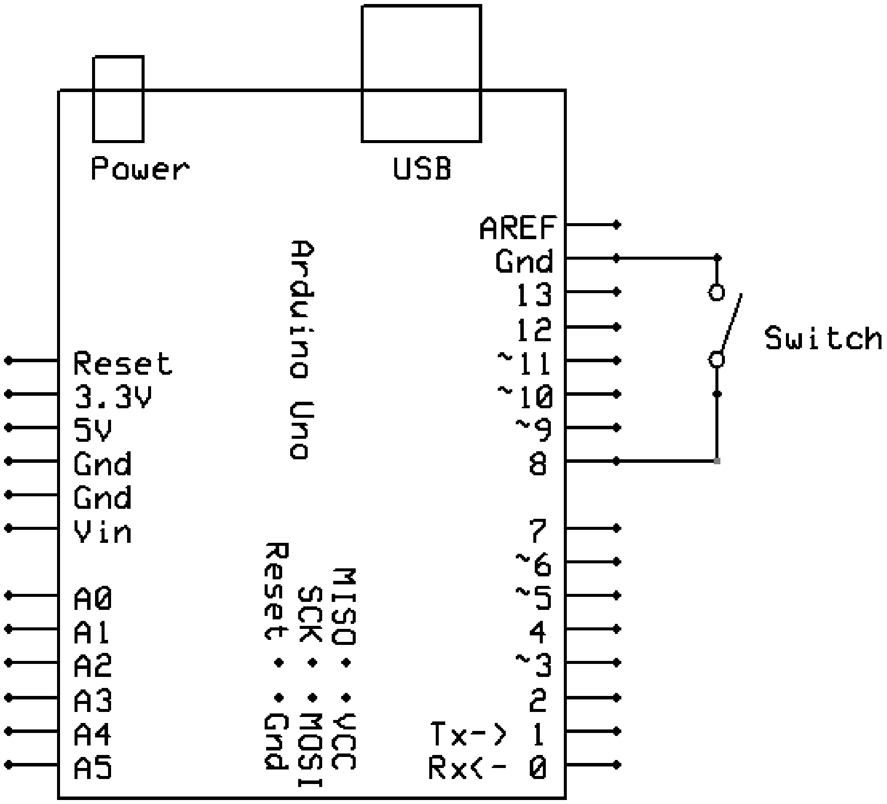
```
I = V / R

V = 5
R = 10000
```

```
Thus: I = 0.5 mA
```

A lower value could be chosen, but more current would be flowing, potentially draining your battery if the device is battery powered. For example, a 1K resistor would draw 10 times as much current, namely 5 mA.

Internal pull-up resistor

The circuit below does not have any resistor, however it relies upon the "internal pull-up" which you can turn on in your code.



This internal pull-up "weakly" pulls the switch up to +5V, so that if the switch is open, it will have 5V on it (through the resistor) and thus will register HIGH if not pressed, and LOW if pressed.

Example code:

```
const byte switchPin = 8;

void setup ()
  {
  Serial.begin (115200);
  pinMode (switchPin, INPUT_PULLUP);
  }  // end of setup

void loop ()
  {
  if (digitalRead (switchPin) == LOW)
     {
     Serial.println ("Switch closed.");
     delay (1000);
     } // end if switchState is LOW

  // other code here ...
  }  // end of loop
```

The internal pull-up is around 50K and thus does not draw much power. It is very useful because it saves a part (the resistor) and some wiring. However over long cable runs the pull-up may be too weak to counter noise being picked up on the wire.

Detecting transitions

I built in a big delay into the above examples to defer the issue of transition handling. However it can't be put off any longer. If you try them you will find that if you hold the switch closed you will see the message "Switch closed." every second, even though you have only closed it once.

What is needed is to detect a **transition**. That is, either:

- It **was** closed and is **now** open.

Or:

- It **was** open and is **now** closed.

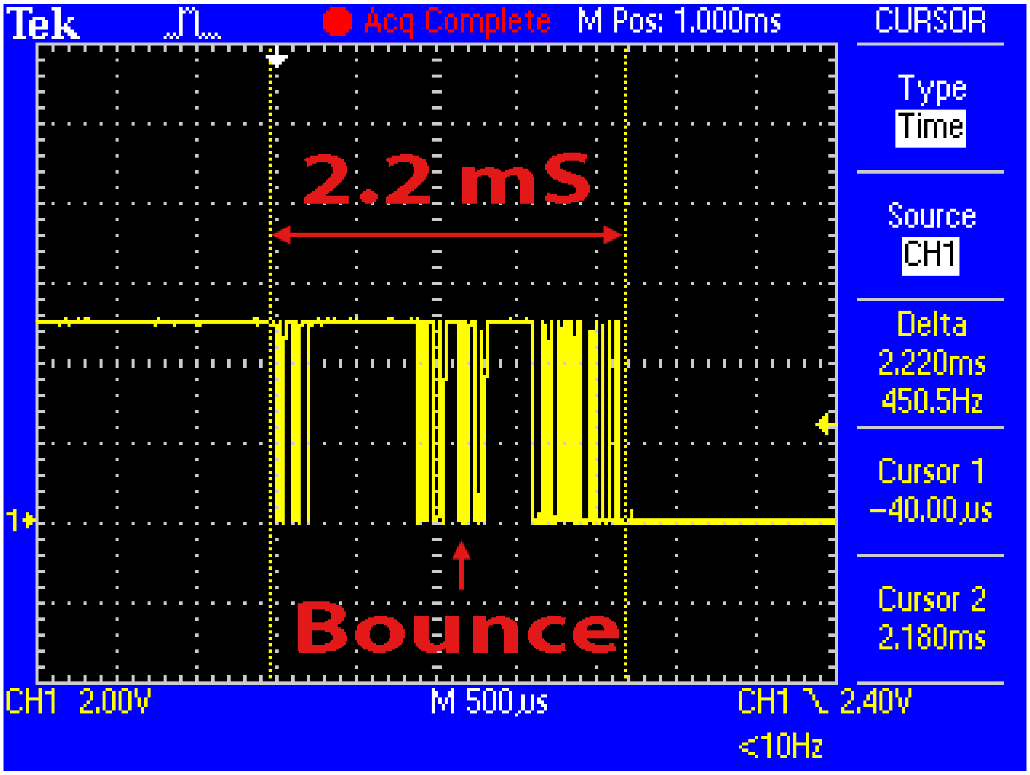
To do that we need to "remember" the previous state of the switch and detect a change, like this:

```
const byte switchPin = 8;
byte oldSwitchState = HIGH;  // assume switch open because of pull-up resistor

void setup ()
  {
  Serial.begin (115200);
  pinMode (switchPin, INPUT_PULLUP);
  }  // end of setup

void loop ()
  {
  // see if switch is open or closed
  byte switchState = digitalRead (switchPin);

  // has it changed since last time?
  if (switchState != oldSwitchState)
    {
    oldSwitchState =  switchState;  // remember for next time
    if (switchState == LOW)
       {
       Serial.println ("Switch closed.");
       }  // end if switchState is LOW
    else
       {
       Serial.println ("Switch opened.");
       }  // end if switchState is HIGH
    }  // end of state change

  // other code here ...

  }  // end of loop
```

This code looks for changes, and only displays something when the switch changes.

Debouncing

The next major issue is "switch bounce". Most switches are mechanical, they have springs to make a good contact, and these contacts bounce as they close. Example of a push-button switch being pressed, as seen on the oscilloscope:



Instead of a smooth transition from 5V (from the pull-up resistor) to 0V (Ground) we see a myriad of bounces. If we just had simple code we might think someone had jabbed the switch 20 times.

This would be very difficult to use, for example if you press the switch once to turn on a light, and again to turn it off, if the number of bounces is even, then the light will stay off.

A simple technique is just to build in a short delay, for example 10 milliseconds (mS). The graphic above shows the bouncing stopped after about 2 mS, so 10 mS should be plenty. For example:

```
const byte switchPin = 8;
byte oldSwitchState = HIGH;  // assume switch open because of pull-up resistor
const unsigned long debounceTime = 10;  // milliseconds

void setup ()
  {
  Serial.begin (115200);
  pinMode (switchPin, INPUT_PULLUP);
  }  // end of setup

void loop ()
  {
  // see if switch is open or closed
  byte switchState = digitalRead (switchPin);

  // has it changed since last time?
  if (switchState != oldSwitchState)
    {
    oldSwitchState =  switchState;  // remember for next time
    delay (debounceTime);   // debounce
    if (switchState == LOW)
       {
       Serial.println ("Switch closed.");
       }  // end if switchState is LOW
    else
       {
       Serial.println ("Switch opened.");
```

```
      }  // end if switchState is HIGH
    }  // end of state change

  // other code here ...

  }  // end of loop
```

Debouncing without delay

The above code is fine in simple applications, and if you test it, you should find that the message "Switch closed." and "Switch opened." should only occur once per switch press. But, there's a problem. If you hang around the Arduino forums for a little while you will probably see people telling you "don't use *delay*". There are various reasons for this, not the least is which that using *delay* stops your code from doing anything else useful (like testing sensors, controlling motors, flashing LEDs, etc.).

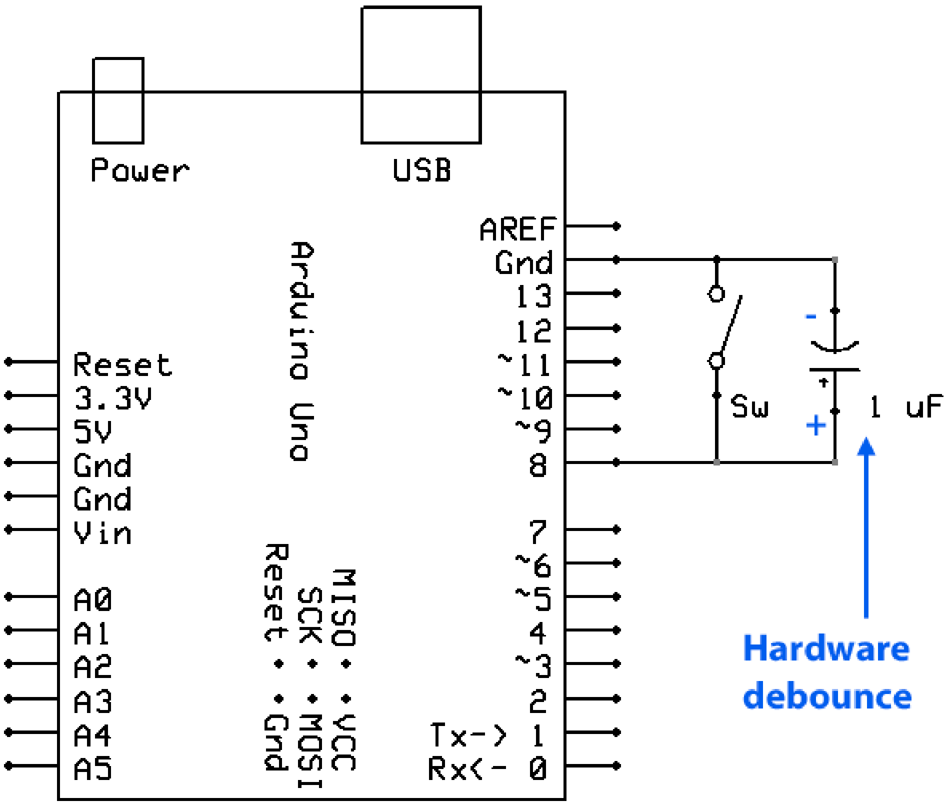
The code below does not use delay, but rather checks for the time that has elapsed after you hit the switch, and sees if the "debounceTime" (in this case 10 mS) has elapsed. If not, it ignores the transition.

```
const byte switchPin = 8;
byte oldSwitchState = HIGH;  // assume switch open because of pull-up resistor
const unsigned long debounceTime = 10;  // milliseconds
unsigned long switchPressTime;  // when the switch last changed state

void setup ()
  {
  Serial.begin (115200);
  pinMode (switchPin, INPUT_PULLUP);
  }  // end of setup

void loop ()
  {
  // see if switch is open or closed
  byte switchState = digitalRead (switchPin);

  // has it changed since last time?
  if (switchState != oldSwitchState)
    {
    // debounce
    if (millis () - switchPressTime >= debounceTime)
       {
       switchPressTime = millis ();  // when we closed the switch
       oldSwitchState =  switchState;  // remember for next time
       if (switchState == LOW)
          {
          Serial.println ("Switch closed.");
          }  // end if switchState is LOW
       else
          {
          Serial.println ("Switch opened.");
          }  // end if switchState is HIGH

       }  // end if debounce time up

    }  // end of state change

  // other code here ...

  }  // end of loop
```

More information on the Arduino site: `http://arduino.cc/en/Tutorial/Debounce`

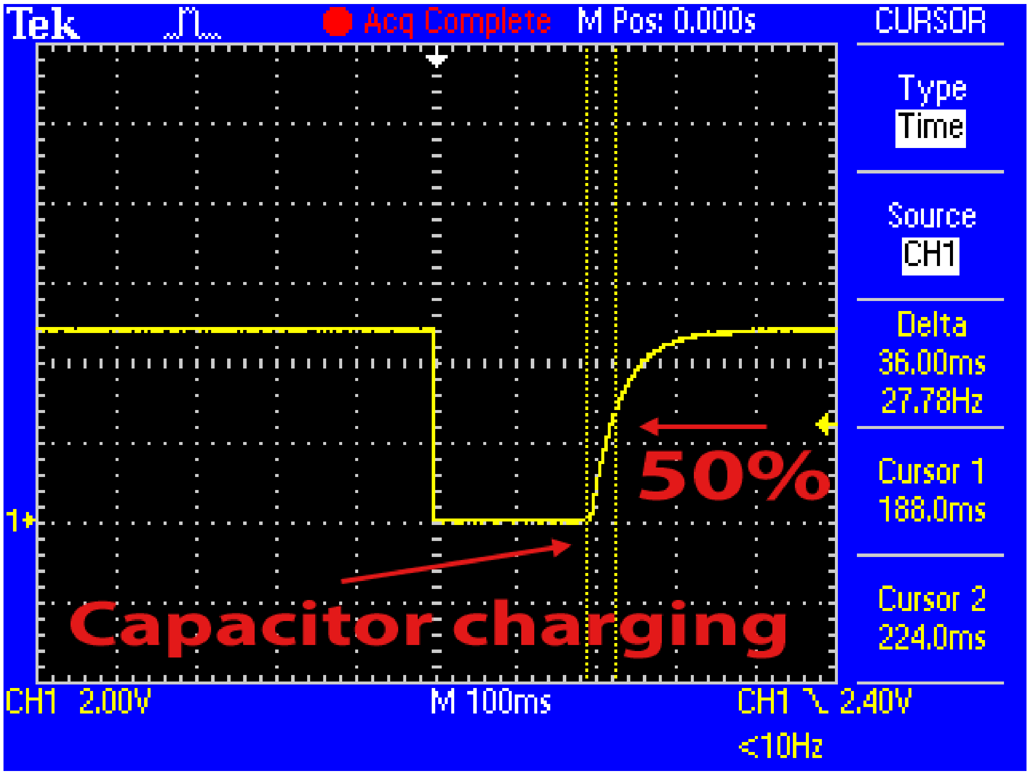
Hardware debounce

Another technique is to use hardware to debounce, like this:



In this case the capacitor is charged by the pull-up resistor. When the switch is pressed it discharges, and takes a moment to charge again. This delay effectively debounces the switch, so the simpler code above (without the debounce) could be used.

This oscilloscope graphic shows the capacitor charging once the switch is released, and since about 50% through is considered HIGH, it has given us about a 36 mS debounce. Thus a smaller capacitor could be used if faster debouncing was wanted.



Acq Complete M Pos: 0.000s

CURSOR

Type: Time

Source: CH1

Delta: 36.00ms 27.78Hz

Cursor 1: 188.0ms

Cursor 2: 224.0ms

CH1 2.00V M 100ms CH1 ╲ 2.40V

<10Hz

Hardware debounce time calculation

Why does it take 36 mS to charge to 50%? The capacitor charges to 50% in 0.69 * R * C where R is (approx) 50k, C = 1 µF (0.000001 F), thus the time is:

```
 0.69 * 50000 * 0.000001 = 0.0345 seconds (34.5 mS)
```

Given that the figure of 50k for the internal pull-up resistor is an estimate, this sounds about right.

To work that out, we look at the formula for the RC time constant:

```
V(t) = V(0) * (1 - e^(-t / RC) )
```

Assuming V(0) - the initial voltage - is 1 for simplicity we have:

```
V = 1 - e^(-t / RC)
1 - V = e^(-t / RC)
log (1 - V) = -t / RC
t = - RC.log(1 - V)
```

Where "log" is the **natural** logarithm (log to the base *e*), otherwise known as "ln".

So we can deduce from this that the time to reach half of the final voltage will be:

```
R = 50000  (50k resistor)
C = 0.000001  (1 µF capacitor)
V = 0.5    (half of initial voltage)
t = - R * C  * log(1 - V)
t = - 50000 * 0.000001 * log (1 - 0.5)
t = 0.03465 seconds (34.6 mS)
```

RC is otherwise known as the "time constant" (τ) which is the time required to charge a capacitor, through a resistor, to approximately 63.2% of its initial value. The figure of 63.2 is derived from: 1 - e^{-1} which is approximately 0.63212055882856.

```
τ = RC
```

General formula for delays

There is a RC time calculator on Ladyada - RC Delay Calculator

The general formula for the delay is:

```
t = - log ( (V - Vc) / V) * R * C
```

Where "log" is the **natural** logarithm (log to the base *e*), otherwise known as "ln".

To calculate the voltage after a specific time (t) use this formula:

```
Vc = V - (V * exp (-t / (R * C)))
```

Where "exp" is the natural exponent.

```
V  = supply voltage   (initial voltage)
Vc = output voltage   (target voltage)
R  = resistance in ohms
C  = capacitance in farads
t  = time in seconds
```

RC Time constant

Also see RC Charging Circuit.

There is discussed that τ (the Time Constant) is:

```
τ = RC
```

A capacitor will charge (or discharge) to 63.2% in τ (one time constant).

The figure 63.2% comes from:

$1 - e^{-1}$ which is approximately 0.63212055882856

A capacitor is reckoned as having fully charged (or fully discharged depending on which way you are going) in 5τ, so you could therefore calculate that for a given resistor and capacitor it will take:

```
5 * R * C
```

In fact it will charge/discharge to 99.3% in that time, where 99.3% comes from:

$1 - e^{-5}$ which is approximately 0.99326205300091

- Nick Gammon

www.gammon.com.au, www.mushclient.com

top

Posted by **Nick Gammon** Australia (20,402 posts) bio *Forum Administrator*

Date Reply #1 on Fri 20 Dec 2013 01:39 PM (UTC)

Amended on Thu 16 Jul 2015 04:17 AM (UTC) by Nick Gammon

Message As part of a recent project I found myself testing a number of switches, needing to debounce them, and act differently depending on a long or short press. Hence a small class below was written.

You can manage multiple switches by making multiple instances of the class. In your **setup** function call the **begin** function to specify the required pin number, and a switch handler.

Then, each time through **loop** call **check** to check for switch presses. When required, your callback function will be called. It is passed LOW or HIGH (which is the new switch state) plus how long has passed since it last changed state. Hence you could find out if the switch was pressed for a short or long time, or if a short or long time elapsed between two presses. The class sets the pin to INPUT_PULLUP, for ease of wiring.

Example code:

```
#include <SwitchManager.h>

// pin assignments
const byte testSwitch = 2;
const byte blueLED = 3;
const byte greenLED = 4;

SwitchManager mySwitch;

// newState will be LOW or HIGH (the is the state the switch is now in)
// interval will be how many mS between the opposite state and this one
// whichPin will be which pin caused this change (so you can share the function amongst multiple switches)

void handleSwitchPress (const byte newState, const unsigned long interval, const byte whichPin)
  {
  if (newState == LOW)
     {
     digitalWrite (blueLED, LOW);
     digitalWrite (greenLED, LOW);
     return;
     }

  // switch must be HIGH
  if (interval >= 1000)
    digitalWrite (blueLED, HIGH);
  else
    digitalWrite (greenLED, HIGH);

  }  // end of handleSwitchPress

void setup ()
  {
  mySwitch.begin (testSwitch, handleSwitchPress);
  pinMode (blueLED, OUTPUT);
  pinMode (greenLED, OUTPUT);
  }

void loop ()
  {
  mySwitch.check ();  // check for presses

  // do other stuff here
  }
```

This turns on one LED if the switch is pressed for a brief time, and the other LED if it is pressed for a longer time.

The library can be downloaded from:

http://gammon.com.au/Arduino/SwitchManager.zip

Unzip and place the folder SwitchManager inside your "libraries" folder. Then restart the IDE.

[EDIT] Modified 12th February 2015 to pass the pin number to the callback function. This lets you share the callback function between multiple switches.

- Nick Gammon

www.gammon.com.au, www.mushclient.com

top

Posted by Nick Gammon Australia (20,402 posts) bio *Forum Administrator*

Date Reply #2 on Thu 16 Jul 2015 10:24 PM (UTC)

Message

Examples of using this library

Motorcycle turn indicator with warning function

This has two switches. If you press the LH switch the LH light starts blinking, ditto for the RH switch and the RH light. If you press them together (ie. press LH switch and then press RH switch before releasing the LH switch) the both blink as a "warning" light.

Once either light is on (or the warning light) pressing either switch cancels it. However if the RH indicator is flashing and you press the LH switch it changes to the LH indicator.

```
#include <SwitchManager.h>

typedef enum {
    NONE,
    LH_DOWN,
    RH_DOWN,
    LH_LIGHT_ON,
    RH_LIGHT_ON,
    BOTH
};

const unsigned long BLINK_INTERVAL = 500; // ms

// pin assignments
const byte LH_SWITCH_PIN = 2;
const byte RH_SWITCH_PIN = 3;
const byte LH_LIGHT = A4;
const byte RH_LIGHT = A5;

SwitchManager LHswitch;
SwitchManager RHswitch;

byte state = NONE;

void handleLHPress (const byte newState, const unsigned long interval, const byte whichPin)
  {
  // switch down?
  if (newState == LOW)
     {
     switch (state)
       {
       // if other switch down, switch to warning mode
       case RH_DOWN:
         state = BOTH;
         break;

       // if already on or warning signal, turn all off
       case LH_LIGHT_ON:
       case BOTH:
         state = NONE;
         break;

       // otherwise switch is now down, but not yet released
       default:
         state = LH_DOWN;
         break;
       }  // end of switch
     return;
     }  // end of LH switch down

  // switch must be up

  if (state == LH_DOWN)  // if down, switch to down-and-released mode
    state = LH_LIGHT_ON;
  }  // end of handleLHPress

void handleRHPress (const byte newState, const unsigned long interval, const byte whichPin)
  {
  // switch down?
  if (newState == LOW)
     {
     switch (state)
       {
       // if other switch down, switch to warning mode
       case LH_DOWN:
         state = BOTH;
         break;

       // if already on or warning signal, turn all off
       case RH_LIGHT_ON:
       case BOTH:
         state = NONE;
         break;

       // otherwise switch is now down, but not yet released
       default:
         state = RH_DOWN;
         break;
       }  // end of switch
     return;
     }  // end of RH switch down

  // switch must be up

  if (state == RH_DOWN)  // if down, switch to down-and-released mode
    state = RH_LIGHT_ON;
  }  // end of handleRHPress

void setup ()
  {
  LHswitch.begin (LH_SWITCH_PIN, handleLHPress);
  RHswitch.begin (RH_SWITCH_PIN, handleRHPress);
  pinMode (LH_LIGHT, OUTPUT);
  pinMode (RH_LIGHT, OUTPUT);
  }  // end of setup

unsigned long lastBlink;
bool onCycle;

void blinkLights ()
  {
  lastBlink = millis ();
  onCycle = !onCycle;

  // default to off
  digitalWrite (LH_LIGHT, LOW);
  digitalWrite (RH_LIGHT, LOW);

  // every second time, turn them all off
  if (!onCycle)
    return;

  // blink light
  switch (state)
    {
    case NONE:
      break;

    case LH_DOWN:
    case LH_LIGHT_ON:
      digitalWrite (LH_LIGHT, HIGH);
      break;

    case RH_DOWN:
    case RH_LIGHT_ON:
      digitalWrite (RH_LIGHT, HIGH);
      break;
    case BOTH:
      digitalWrite (LH_LIGHT, HIGH);
      digitalWrite (RH_LIGHT, HIGH);
      break;

    }  // end of switch on state

  }  // end of blinkLights

void loop ()
  {
  LHswitch.check ();  // check for presses
  RHswitch.check ();  // check for presses

  if (millis () - lastBlink >= BLINK_INTERVAL)
    blinkLights ();

  // other stuff
  }  // end of loop
```

Motorcycle turn indicator with brake light

This has three switches. If you press the LH switch the LH light starts blinking, ditto for the RH switch and the RH light. If you press the "brake" switch the brake light comes on only for as long as the switch is pressed.

Once either light is on pressing either switch cancels it. However if the RH indicator is flashing and you press the LH switch it changes to the LH indicator.

```
#include <SwitchManager.h>

typedef enum {
    NONE,
    LH_DOWN,
    RH_DOWN,
    LH_LIGHT_ON,
    RH_LIGHT_ON
};

const unsigned long BLINK_INTERVAL = 500; // ms

// pin assignments
const byte LH_SWITCH_PIN = 2;
const byte RH_SWITCH_PIN = 3;
const byte BRAKE_SWITCH_PIN = 4;
const byte LH_LIGHT = A3;
const byte RH_LIGHT = A4;
const byte BRAKE_LIGHT = A5;

SwitchManager LHswitch;
SwitchManager RHswitch;

byte state = NONE;

void handleLHPress (const byte newState, const unsigned long interval, const byte whichPin)
  {
  // switch down?
  if (newState == LOW)
     {
     switch (state)
       {
       // if already on or warning signal, turn all off
       case LH_LIGHT_ON:
         state = NONE;
         break;

       // otherwise switch is now down, but not yet released
       default:
         state = LH_DOWN;
         break;
       }  // end of switch
     return;
     }  // end of LH switch down

  // switch must be up

  if (state == LH_DOWN)  // if down, switch to down-and-released mode
    state = LH_LIGHT_ON;
  }  // end of handleLHPress

void handleRHPress (const byte newState, const unsigned long interval, const byte whichPin)
  {
  // switch down?
  if (newState == LOW)
     {
     switch (state)
       {

       // if already on or warning signal, turn all off
       case RH_LIGHT_ON:
         state = NONE;
         break;

       // otherwise switch is now down, but not yet released
       default:
         state = RH_DOWN;
         break;
       }  // end of switch
     return;
     }  // end of RH switch down

  // switch must be up

  if (state == RH_DOWN)  // if down, switch to down-and-released mode
    state = RH_LIGHT_ON;
  }  // end of handleRHPress


void setup ()
  {
  LHswitch.begin (LH_SWITCH_PIN, handleLHPress);
  RHswitch.begin (RH_SWITCH_PIN, handleRHPress);
  pinMode (BRAKE_SWITCH_PIN, INPUT_PULLUP);
  pinMode (LH_LIGHT, OUTPUT);
  pinMode (RH_LIGHT, OUTPUT);
  pinMode (BRAKE_LIGHT, OUTPUT);
  }  // end of setup

unsigned long lastBlink;
bool onCycle;

void blinkLights ()
  {
  lastBlink = millis ();
  onCycle = !onCycle;

  // default to off
  digitalWrite (LH_LIGHT, LOW);
  digitalWrite (RH_LIGHT, LOW);

  // every second time, turn them all off
  if (!onCycle)
    return;

  // blink light
  switch (state)
    {
    case NONE:
      break;

    case LH_DOWN:
    case LH_LIGHT_ON:
      digitalWrite (LH_LIGHT, HIGH);
      break;

    case RH_DOWN:
    case RH_LIGHT_ON:
      digitalWrite (RH_LIGHT, HIGH);
      break;

    }  // end of switch on state
  }  // end of blinkLights


void loop ()
  {
  LHswitch.check ();  // check for presses
  RHswitch.check ();

  if (digitalRead (BRAKE_SWITCH_PIN) == LOW)
    digitalWrite (BRAKE_LIGHT, HIGH);
  else
    digitalWrite (BRAKE_LIGHT, LOW);

  if (millis () - lastBlink >= BLINK_INTERVAL)
    blinkLights ();

  // other stuff
  }  // end of loop
```

- Nick Gammon

www.gammon.com.au, www.mushclient.com

top

The dates and times for posts above are shown in Universal Co-ordinated Time (UTC).

To show them in your local time you can join the forum, and then set the 'time correction' field in your profile to the number of hours difference between your location and UTC time.

18,415 views.

Postings by administrators only.

Refresh page

Go to topic: (Choose topic) ▼ Go Search the forum

top

Quick links: **MUSHclient**. MUSHclient **help**. Forum **shortcuts**. Posting **templates**. Lua **modules**. Lua **documentation**.

Home

Designed and written by Nick Gammon

Comments to: Gammon Software support
XML Forum RSS feed (http://www.gammon.com.au/rss/forum.xml)