Arduino OneButton Library

This sketch and library shows how to use a input pin by detecting some of the typical button press events like single clicks, double clicks and long-time pressing a button. This enables you to reuse the same button for multiple functions and lowers the hardware invests.

Download the project files

- You can download the OneButton library from github by using git or svn or download the latest version as a zip file
  (right side button labeled "Download ZIP")
  https://github.com/mathertel/OneButton

Introduction

When starting with Arduino programming you surely came across the simple Button tutorial to understand how to read from push buttons and the Debounce example that shows how to get a clear signal from a pushbutton by eliminating short-time on/off sequences produced by contact jittering.

The result is a small library called OneButton that you can hook up into your sketches easily. This post shows you how to use the library, how it is implemented and gives you a brief reference sheet in the end.

This post on my web site was also written to explain some good implementation approaches so I hope that all the hyperlinks will guide you to other important samples and tutorials if you haven’t already seen them.

Reading a Button without blocking the program

One downside of the simple examples mentioned above is, that they use the loop() function and therefor make it hard to reuse the code they include.

I expect that your sketch will have to do more complicated things and that you will need the loop() function for your own purpose. So it is a common requirement for libraries and reusable code to avoid implementing this function. Instead the tick() function of the library should be called frequently. Inside this function you will find the coding for detecting all of the 3 available events.

For any of the 3 implemented events you can register a function to link your code to the library. Your functions then will be called when the corresponding situation has been detected.

Sample for using the OneButton library

Here is a simple sample that uses the OneButton library to change the default led on pin 13 when double clicking a button attached on pin A1.

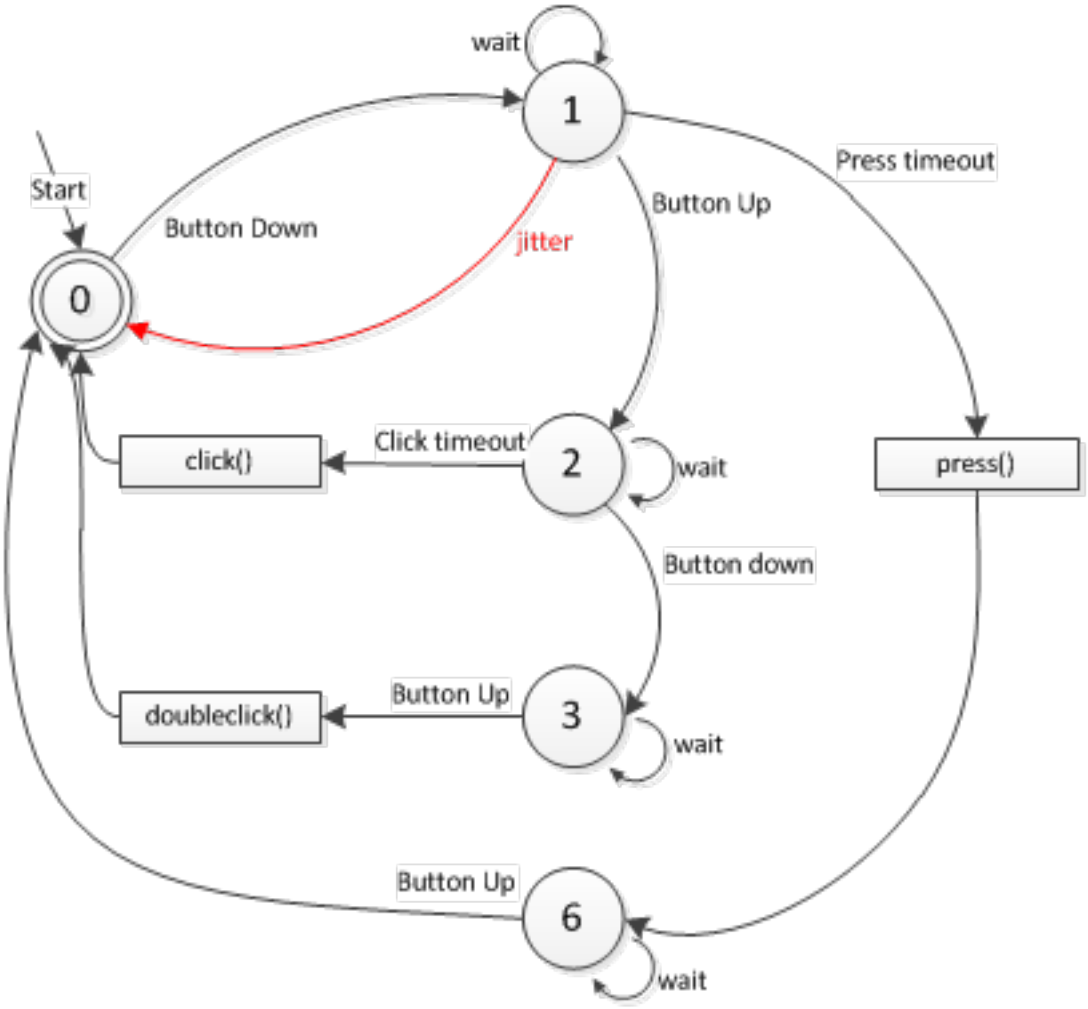
```
/*
 S01_SimpleOneButton
 Simple OneButton sketch that shows how to ???

 The circuit:
 * Connect a pushbutton to pin A1 (ButtonPin) and ground
 * and see results on pin 13 (StatusPin).
 * 03.03.2011 created by Matthias Hertel
 */

#include "OneButton.h"

// Setup a new OneButton on pin A1.
OneButton button(A1);


// setup code here, to run once:
void setup() {
  // enable the standard led on pin 13.
  pinMode(13, OUTPUT);      // sets the digital pin as output

  // link the doubleclick function to be called on a doubleclick event.
  button.attachDoubleClick(doubleclick);
} // setup


// main code here, to run repeatedly:
void loop() {
  // keep watching the push button:
  button.tick();

  // You can implement other code in here or just wait a while
  delay(10);
} // loop


// this function will be called when the button was pressed 2 times in a sho
void doubleclick() {
  static int m = LOW;
  // reverse the LED
  m = !m;
  digitalWrite(13, m);
} // doubleclick
```

Please alo read the inline comments. You can see there how the doubleclick function is attached to the situation when the button is released for the second time.

Implementation details

If you like to see how the Library works here are some explanation:

Inside the OneButton library within the tick() function you can find the implementation for checking the input pin detecting the single click, double click or a long press situation. The kind of implementation is called a finite state machine (FSM) that implements the following state diagram:



Every time the tick() function is called the current situation, regarding the current state and the input values is analyzed and when applicable an external function is called and/or the current state is changed. Especially the OneButton Library implementation is never calling delay() or similar functions and therefore will return quickly in most cases.

Here is a brief explanation of how this (software) machine works:

Input values

The first input value that is used by this implementation is the value of the digital input pin. This pin is pulled to high internally and is pulled down by the attached button when it is pressed.

Implementing this event source is done at the beginning of the state machine function:

```
int buttonLevel = digitalRead(_pin); // current button signal.
```

The other input that will be needed is the current time. The millis() gives a quiet good linear timing value that can be used perfectly for detecting that a specific amount of time has passed. The current value is copied over into the *now* variable:

```
unsigned long now = millis(); // current (relative) time in msecs.
```

State

The information that has to be remembered between one call and the following is called the state. There is especially one variable called state that will hold the information about the current situation by using the numbers in the above diagram and another one that will store the time when the button was pressed the first time.

These 2 variables are defined within the state machine function and marked with the static keyword so they will store the value for the next time the tick function is called.

```
int _state = 0; // starting with state 0: waiting for button to be pressed
unsigned long _startTime; // will be set in state 1
```

Starting situation (state 0)

The machine starts with the situation (state 0) where by waiting for the button to be pressed down. If the button is not pressed down it just returns from the function because the function will be called again shortly after.

But if the button is pressed it has to remember that another situation (state 1) is now in place. Because of later timing analysis it also has to remember the current time. That’s all.

```
if (_state == 0) { // waiting for One pin being pressed.
  if (buttonLevel == ButtonDown) {
    _state = 1; // step to state 1
    _startTime = now; // remember starting time
  } // if
```

The button is down for the first time (state 1)

The next time the function will be called (some msecs later) it’s again the starting situation (state 0) or the new situation (state 1) where the button was just pressed. In this case it just has to wait until the button is released again and switch to the next situation (state 2). When the button is still pressed nothing has to be done and because human fingers are not so quick it will likely be that this situation (state 1) will stay for some ticks.

When this situation (state 1) is lasting longer than a “long” time (currently 1 sec.) another situation (state 6) will be detected. This is done by comparing the actual time plus the defined interval to the persisted start time. In this case the external press() function will be called.

```
} else if (_state == 1) { // waiting for One pin being released.
  if (buttonLevel == ButtonUp) {
    _state = 2; // step to state 2

  } else if ((buttonLevel == ButtonDown) && (now > _startTime + _pressTicks)
    if (_pressFunc) _pressFunc();
    _state = 6; // step to state 6
  } // if
```

The button is was released from the first click (state 2)

In this situation (state 2) the button was already released and two things may happen:

If some time has passed and the button was not pressed a second time the external click() function is called and the state is reset to 0.

If the button is pressed a second time before the time has passed a second button press is detected the next situation (state 3) will carry on:

```
} else if (_state == 2) { // waiting for One pin being pressed the second ti
  if (now > _startTime + _clickTicks) {
    // this was only a single short click
    if (_clickFunc) _clickFunc();
    _state = 0; // restart.

  } else if (buttonLevel == ButtonDown) {
    _state = 3; // step to state 3
  } // if
```

Waiting for menu pin being released the second time (state 3)

The final situation (state 3) after a second click was detected in the defined timeframe is to wait for the button to be released. When this happens the external doubleClic() function is called and the state machine is reset so everything can start from scratch.

```
} else if (_state == 3) { // waiting for One pin being released finally.
  if (buttonLevel == ButtonUp) {
    // this was a 2 click sequence.
    if (_doubleClickFunc) _doubleClickFunc();
    _state = 0; // restart.
  } // if
```

Waiting for menu pin being released the second time (state 6)

This situation (state 6) is very similar to the situation after the second click exists to wait for the button to be released. When this happens the state machine is reset so everything can start from scratch.

```
} else if (_state == 6) { // waiting for One pin being release after long pr
  if (buttonLevel == ButtonUp) {
    _state = 0; // restart.
  } // if
```

Summary

Implementing a finite state machine (FSM) is easy for many situations and it can help a lot in reducing the implementation complexity and also helps in writing pseudo multitasking sketches for your Arduino.

Installation of the OneButton library

The code for the OneButton functionality is available by using the library technique and comes with 2 files. You can find information on how to build a library in the Library Tutorial.

Installing the code as a library is easy by creating a new sub-folder (called OneButton) in the libraries folder and copying the files (OneButton.h, OneButton.cpp) into it. You should restart the Arduino thereafter because the available libraries are detected on startup.

In this case you have to include the library using the #include statement with angle brackets:

```
#include <OneButton.h>
```

The alternate method is to leave the same 2 files into your sketch folder. Then these files are available in the Arduino development environment when you open your sketch the next time. This is good if you like to try-out new things or if you want to improve a library.

In this case you have to include the library using the #include statement with double quotes:

```
#include "OneButton.h"
```

Sample

There is a example sketch inside the library folder that shows how to setup the library, bind a special function to the doubleclick event and toggle a output pin.

ChangeLog

- 30.08.2011 First published version.
- 03.12.2011 Updated for compatibility with the Arduino 1.0 environment.
- 07.04.2013 Sample sketch BlinkMachine added.
  The Sketch shows how to setup the library and bind a "machine" that can blink the LED slow or fast.
