AnalogSwitches

TUTORIAL

WHAT IS AnalogSwitches?

The AnalogSwitches Library allows you to use any number of banks of up to 16 momentary switches, while using only one analog pin on the Arduino per bank.

HOW DOES IT WORK?

First we wire the switches in a specific manner (which will be explained later). Then, we tell the AnalogSwitches Library how many banks of switches we will use.

```
AnalogSwitches switchBank(2);
```

Then, for each bank we will specify:

- The ID of the bank (From 1 to n);
- The Arduino Analog pin attached to the bank;
- The number of switches in that bank;
- An index to identify the switch.

```
switchBank.addSwitches(1, A0, 7, 100);
switchBank.addSwitches(2, A1, 5, 200);
```

The switches are numbered from 1 to the actual number of switches in our setup. Zero means that no switch is pressed. When we want to know which switch is pressed (if any), we call:

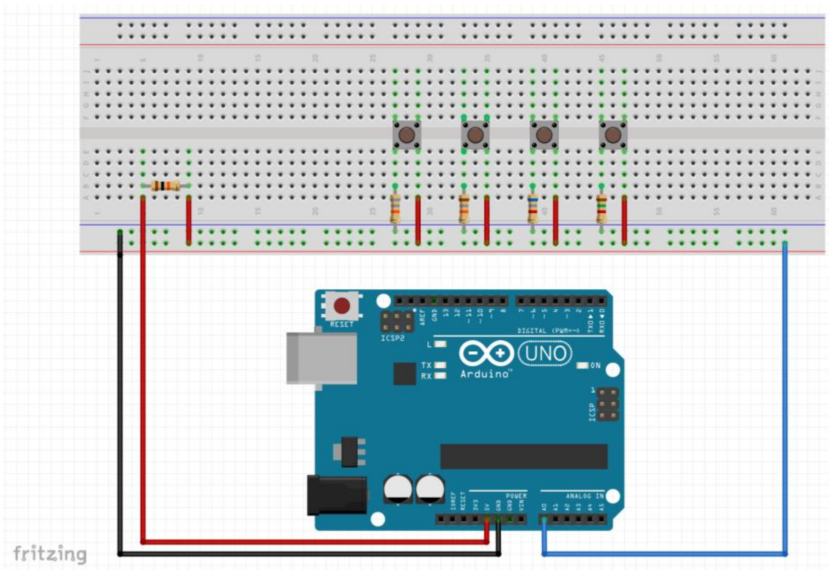
```
index = switchBank.getIndex();
```

HOW DO WE WIRE THE SWITCHES?

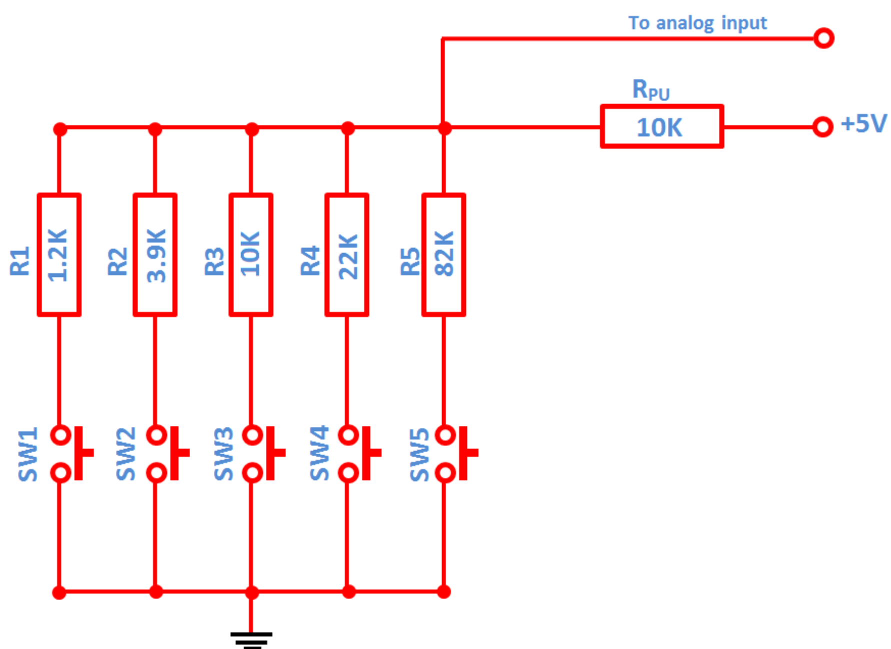
A first resistor (called Pullup) is tied to Arduino's +5V.
One of the pins of each switch is connected to the other end of the pullup. So is the Analog pin of the Arduino.
The other pin of each switch is connected to a resistor and then to Arduino's GND.

```
+5V-----Rpullup--------+-------+---   ...   ---+-------A0
                     SW0     SW1      ...     SWn
                       +       +      ...      +
                      R0      R1      ...     Rn
GND--------------------+-------+---   ...   ---+
```

Here is an example of a bank with four switches:



And here is the schematic of a bank with 5 switches



WHAT VALUES DO WE NEED FOR THE RESISTORS?

The table below shows the resistor values according to the number of switches contained in the bank.

The values in **bold** are taken from the E24 Series. For more than 10 switches, resistors with ±1% accuracy are preferred.

4	1,5K	5,6K	18,0K	68,0K												
5	1,2K	3,9K	10,0K	22,0K	82,0K											
6	820	3,3K	6,8K	15,0K	33,0K	120,0K										
7	820	2,7K	5,6K	10,0K	18,0K	39,0K	120,0K									
8	680	2,2K	4,7K	8,2K	12,0K	22,0K	47,0K	150,0K								
9	560	2,2K	3,9K	6,8K	10,0K	15,0K	27,0K	47,0K	180,0K							
10	560	1,8K	3,3K	5,6K	8,2K	12,0K	18,0K	33,0K	56,0K	180,0K						
11	470	1,5K	2,7K	4,7K	6,8K	10,0K	15,0K	22,0K	33,0K	68,0K	200,0K					
12	430	1,5K	2,7K	4,3K	6,2K	8,2K	12,0K	16,0K	24,0K	39,0K	68,0K	220,0K				
13	390	1,2K	2,2K	**3,6K**	**5,1K**	**7,5K**	10,0K	**13,0K**	18,0K	27,0K	39,0K	82,0K	270,0K			
14	330	1,2K	2,2K	3,3K	4,7K	**6,2K**	**9,1K**	**12,0K**	15,0K	22,0K	**30,0K**	47,0K	82,0K	270,0K		
15	330	1,2K	2,2K	**3,0K**	**4,3K**	5,6K	**7,5K**	10,0K	**13,0K**	18,0K	**24,0K**	33,0K	47,0K	82,0K	270,0K	
16	330	1,0K	1,8K	2,7K	3,9K	**5,1K**	6,8K	**9,1K**	**11,0K**	15,0K	**20,0K**	27,0K	33,0K	56,0K	100,0K	330,0K

WHAT ARE THE METHODS THAT THE MENU LIBRARY PROVIDES?

MANDATORY METHODS

INSTANTIATION

AnalogSwitches::AnalogSwitches(int pinCount) (MANDATORY)

```
AnalogSwitches switchBank(2);
```

The first method is called a constructor, which creates the switch banks. It has to be placed **before** setup().

PinCount is the number of banks you have in your setup.

You can have as many number of switch banks as you wish. Just give them different names.

ADDING A BANK OF SWITCHES

void AnalogSwitches::addSwitches(int pinId, int pin, int count, int index) (MANDATORY)

```
switchBank.addSwitches(1, A0, 7, 100);
```

This method will add a bank of switches to the setup.

pinId is the number assigned to the bank. If you have 3 banks, then banks will be numbered 1, 2 and 3.

pin is the analog pin of the Arduino to which the bank is tied

count is the number of switches in that bank

index is the number returned by the library if the first switch of the bank is pressed. The number of the switch in the bank is added to the index for the other switches. If you have 5 switches in a bank that has an index of 100, then the 5 switches will be identified with: 101, 102, 103, 104 and 105. Do not give another bank an index that would result in having two switches returning the same index. Giving an index of 106 to the next bank would be OK.

METHOD TO READ THE SWITCHES

int AnalogSwitches::getIndex();
You will use this method to get the index of the key that is currently pressed. If no key is pressed, you will get zero (0). This method will wait for the key to be released before returning its value.

HOW DO WE FIGURE OUT THE VALUES FOR THE RESISTORS?

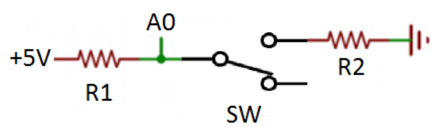
This last part is a little technical. You can stop reading here and everything will be OK.

If you are interested, read on…

The analogRead(A0) function returns a value between 0 and 1023. Those values correspond to the voltage read at the pin between 0V and 5V. What it effectively does, is to map 0V to 5V (an analog value) into an integer going from 0 to 1023 (a digital value).

We will use what is called a voltage divider to change the values of the voltage at the Analog pin.
First, we place what is called a pullup resistor between +5V and the Analog pin. The pin will read 1023 from then on, until we split this current in two with a switch and another resistor connected to ground

The basic circuit with only one switch looks like this:



When the switch is off, the whole 5V is sent to A0. When the switch is on, part of the 5V goes to A0, and another part goes to ground. How much goes where depends of the values of R1 and R2. Before doing some math, simple logic says that if R1 equals R2, half the voltage will go to A0, and the other half will go to ground.

The formula for a voltage divider is:

$V_{out} = V_{in} \frac{R2}{R1+R2}$ Or, in our case: $V_{A0} = 5V \frac{R2}{R1+R2}$

Replacing R1 and R2 with 10KΩ gives:

$$V_{A0} = 5V \frac{10000}{10000+10000} = 5V \frac{10000}{20000} = 2{,}5V$$

This is half the voltage sent by Arduino's 5V pin. So, analogRead(A0) will return 1023/2 = 512

In our setup, R1 will be the pullup resistor. R1 will be the same, regardless of the number of switches we use. But R2 will be different for each switch.

So here is our strategy: We want to use "n" switches in our setup. Since analogRead(A0) returns a value between 0 and 1023, we will divide it in "n" bins of equal size. Let's try it for four switches.

Each bin will be 1024/4 = 256 wide:

256	256	256	256
128	394	640	896

Next, we will find the center of each bin: 256/2 = 128 for the center of the first bin. To obtain the center of the other bins, we will add 256: **128**+256 = **394**, 394+256 = **640** and 640+256 = **896**

To find the resistor value for each switch to match the center of each bin, we will re-write the voltage divider equation to make it easier for us. Now, we want to find R2, for the desired “binCenter”, knowing that R1 is 10 000Ω. And to help us more, we will replace the voltage with what is actually returned by analogRead(A0) for 5V, which is 1023.

$$binCenter = 1023\ \frac{R2}{10000+R2}$$

Isolating R2 gives:

$$R2 = \frac{10000\ X\ binCenter}{(1023\ - binCenter)}$$

For the center of the first bin:

$R2 = \frac{10000\ X\ 128}{(1023\ -128)}$ = 1429Ω or 1.429KΩ

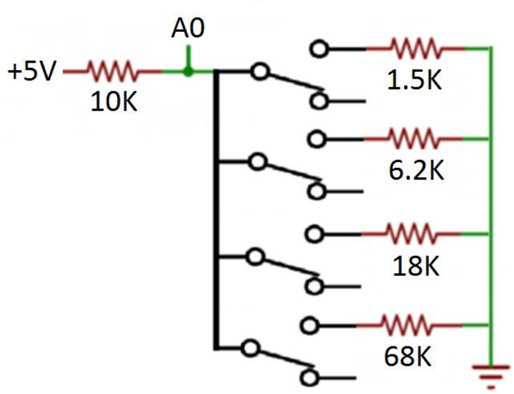
Of course, you won’t find a resistor with this exact value, but you will find one with 1,5KΩ. This is the value that we will use for the first switch. Knowing that the resistor has ±5%, tolerance, its real value will be somewhere between 1425Ω and 1575Ω.

Using the original voltage divider equation, V_{out} will be between 0.625V and 0.679V, and, mapping 0..5V to 0..1023 the value returned by analogRead(A0) will be between 128 and 139.

Doing the same math for the tree other switches yields:

Ideal (KΩ)	Real (KΩ)	Min(KΩ)	Max (KΩ)	Min Read()	Max Read()
1.429K	**1.5**	1.425	1.575	128	139
6.0	**6.2**	5.890	6.510	379	403
16.7	**16**	15.2	16.8	617	641
70	**68**	64.6	71.4	886	897

Now, our schematics look like this:



The last step to do is to use the results of analogRead(A0) to be able to return a switch number between 1 and 4 (in this case).The figure bellow shows, to scale, in red, the expected return values of analogRead(A0) for this setup. Also shown at the right end, the range that what we will use for “no key”. Notice that when R1 is close to R2, the range of possible reads gets wider. With ±5% resistors, when R1 = R2, the voltage read is ±0.062V or an analogRead(A0) difference of ±13.

If we divide the values returned by the size of our bins the results will be:

Min Read()	Max Read()	Min	Max
128	139	0.500	0.543
379	403	1.480	1.574
617	641	2.410	2.504
886	897	3.461	3.504

If we only use the integer part of those values, we will get 0, 1, 2 and 3 for the four switches. Adding one will give us the number of the switch that was pressed. So, in Arduino’s language:

```
key = int(analogRead(A0)/binSize) + 1;
```

Now, we have to take care of when there is no key pressed. We said that we would return zero. As we saw earlier, when no key is pressed, the whole 5V goes into pin A0, causing analogRead(A0) to return 1023.

Yes, but… there is some electrical noise (perturbations) which will fluctuate the voltage read by Analog pins. So, to make ourselves safe, we will consider that a value higher than 1010 will be considered as a “no key” situation.

```
if (analogRead(A0) > 1010) return 0;
```

ARE THE SWITCH DEBOUNCED?

Certainly so. Switches are switches and they will bounce.

The way to go is to read the Analog pin repeatedly, then average all the reads, and use this to return the key number.

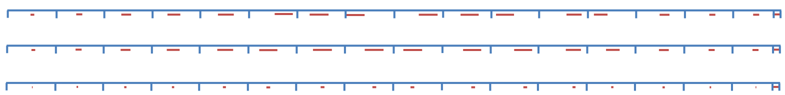
The pseudo code looks like this:

```
Read A0
if (A0 < 1010) {
  Read A0 5 times and average the readings
  Return (A0 / 1024 * SwitchCount) + BankIndex
}
```

WHY STOP AT 16 SWITCHES?

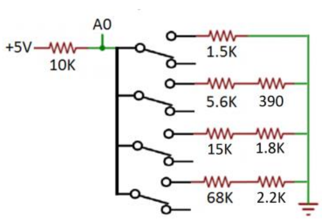
As the bins gets smaller, the margin of error caused by the ±5% tolerance on our resistors also gets smaller. To make matters worse, we have a limited set of values for our resistors. At 16 switches, the bin size is only 64 (0.3V). As you can see, even at 16 switches we are pushing our luck.

Below, you will find three graphics. The two top ones use ±5% resistors. The first one uses resistors from the E12 series. Notice how the voltages can get real close to the bins boundaries. The second one uses resistors from the E24 series. Still not perfect, but the voltages are more centered in their bins. The third one shows that using ±1% resistors helps a lot.



If we can find them and if we can afford them, the E48 series will give us two advantages: just ±1% tolerance, and a choice amongst 48 values per decade instead of 24.

To get closer to the center we could also put another small resistor in series with the real value. Looking at our 4 key keypad example, the values for the fourth switch are: Ideal: 70kΩ, Real: 68KΩ . We could use 68KΩ plus 2.2KΩ in series and having a center at 70.2kΩ, much closer to the ideal resistor.



With this setup, we get Ideal, vs Real for the analogRead(A0):

Switch 1	128 > 133 vs 133
Switch 2	384 > 381 vs 392
Switch 3	639 > 641 vs 630
Switch 4	895 > 895 vs 892

With 4 keys, it is useless… with 16, well, it could help.

If going this route, why not measure the real value of the first resistor with a multimeter and compensate with the second resistor, eliminating the first resistor's ±5% inaccuracy.

I sincerely hope that this AnalogSwitch Library will help you in your projects.
Jacques Bellavance