COMBINATORIAL KEYPAD TUTORIAL

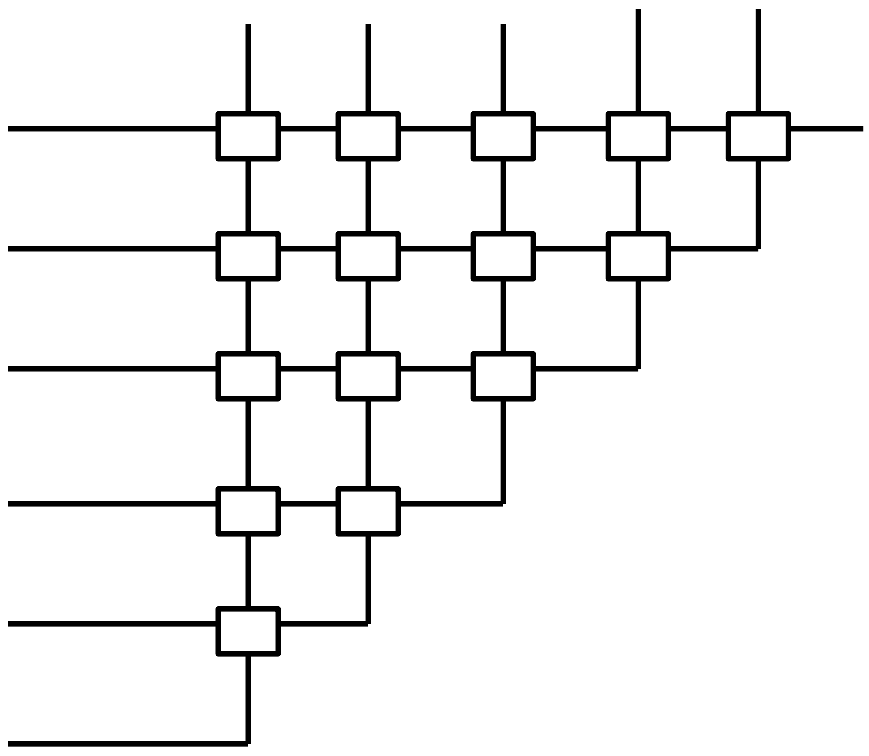
Chirag Sangani

INTRODUCTION

The combinatorial keypad design is an efficient form of keypad design which saves hardware resources while trading off for programming simplicity. With this design, if one allots n I/O pins of the microcontroller for the keypad, he/she can implement $^{n}C_{2}$ independent keys.

CONCEPT

Let us assume that n I/O pins are available. Select all possible pairs of pins, and connect them by a switch. Basic combinatorial algebra tells us that $^{n}C_{2}$ switches will be implemented. Whenever the user presses any switch, a unique pair of I/O pins will be shorted. The microcontroller can now run a program to detect which pair of pins has been shorted. Here is a diagram of the design logic:



Here, the boxes represent the switches and the lines represent the I/O wires.

STROKE DETECTION

To detect a key stroke, the microcontroller has to constantly run a program in the main loop to detect shorting between any two pins.

Let us assume that n pins of a PORT are allotted for the keypad. These pins have to be necessarily in line for the program to work – no pin in between can be used for any other work while the keypad is being polled. The first pin is set to output and its value is set to LOW. The other pins are set to input and are pulled-up, i.e. in the event that there is no external input, the default value of these input pins will be HIGH. The code to achieve this is as follows:

Please note that all the code in this tutorial has been written assuming that the programming environment is AVR Studio using the avr-gcc compiler which can be found in WinAVR. If your programming environment differs, please adapt the code suitably.

```
DDR = (1<<sourcePin);
PORT = ~DDR;
```

Where, sourcePin is the bit number of the pin to be set as output.

The remaining pins are now checked to see if any of the pins have a LOW value. If they do, it means that the pair consisting of the sourcePin, and the input pin with the LOW value, called as the sinkPin, is shorted and the corresponding key has been pressed. The function now has to return a value unique to this combination in order to distinguish this pair. The code to achieve this is as follows:

```
for(sinkPin = sourcePin + 1; sinkPin <= LastBit; sinkPin++)
{
      if(((PIN & (1<<sinkPin)) >> sinkPin == 0)
      {
            return (10*sourcePin + sinkPin);
      }
}
```

Finally, the entire code has to be wrapped inside a for loop which progressively increments the value of sourcePin. The entire code reads like:

```
int pollKeypad(void)
{
      int sourcePin, sinkPin = 0;
      for(sourcePin = FirstBit; sourcePin < LastBit; sourcePin++)
      {
            DDR = (1<<sourcePin);
            PORT = ~DDR;
            for(sinkPin = sourcePin + 1; sinkPin <= LastBit, sinkPin++)
            {
                  if(((PIN & (1<<sinkPin)) >> sinkPin == 0)
                  {
                        DDR = 0;
                        PORT = ~DDR;
                        return (10*sourcePin + sinkPin);
                  }
            }
      }
      DDR = 0;
      PORT = ~DDR;
      return -1;
}
```

Assuming both sourcePin and sinkPin remain less than 10 (the return code has to be suitably modified if this is not true), this function will return a two digit integer which will represent the unique pair of pins which has been shorted. If pins n_1 and n_2 have been shorted, assuming $n_1 < n_2$, then the integer returned will have the value $10*n_1 + n_2$. With this information, one can construct a function with a switch case which accepts this integer value and returns the appropriate ASCII character which one wants to be associated with that key. Such a function would look like:

```
char parsePollResult(int pollResult)
{
      switch(pollResult)
      {
            case 12: return ‘a’;
            case 13: return ‘b’;
            case 14: return ‘c’;
            .
            .
            .
            .
            default: return -1;
      }
}
```

Usage

An example code which makes use of the above functions to wait for a user input would be like:

```
int pollResult = pollkeypad();
while(pollResult == -1)
{
      pollResult = pollKeypad();
}
char parseResult = parsePollResult(pollResult);
_delay_ms(450);         // Optional
```

The result will be stored in the variable parseResult. Please note that to use the optional `_delay_ms();` command, you will need to add the following line of code at the beginning of your program:

```
#include <util/delay.h>
```

Shortcomings of this method

While the combinatorial method is highly useful method of implementing a large number of keys, it does have a few shortcomings. There are a few limitations to how many simultaneous keystrokes it can detect – if one presses two keys simultaneously which share a common bit, then a third key, while not actually pressed, will be triggered. Thus, this method is useful in making keypads where only one key is expected to be pressed at a time.