EIE3105: Keyboard and Debouncing Buttons

Dr. Lawrence Cheung

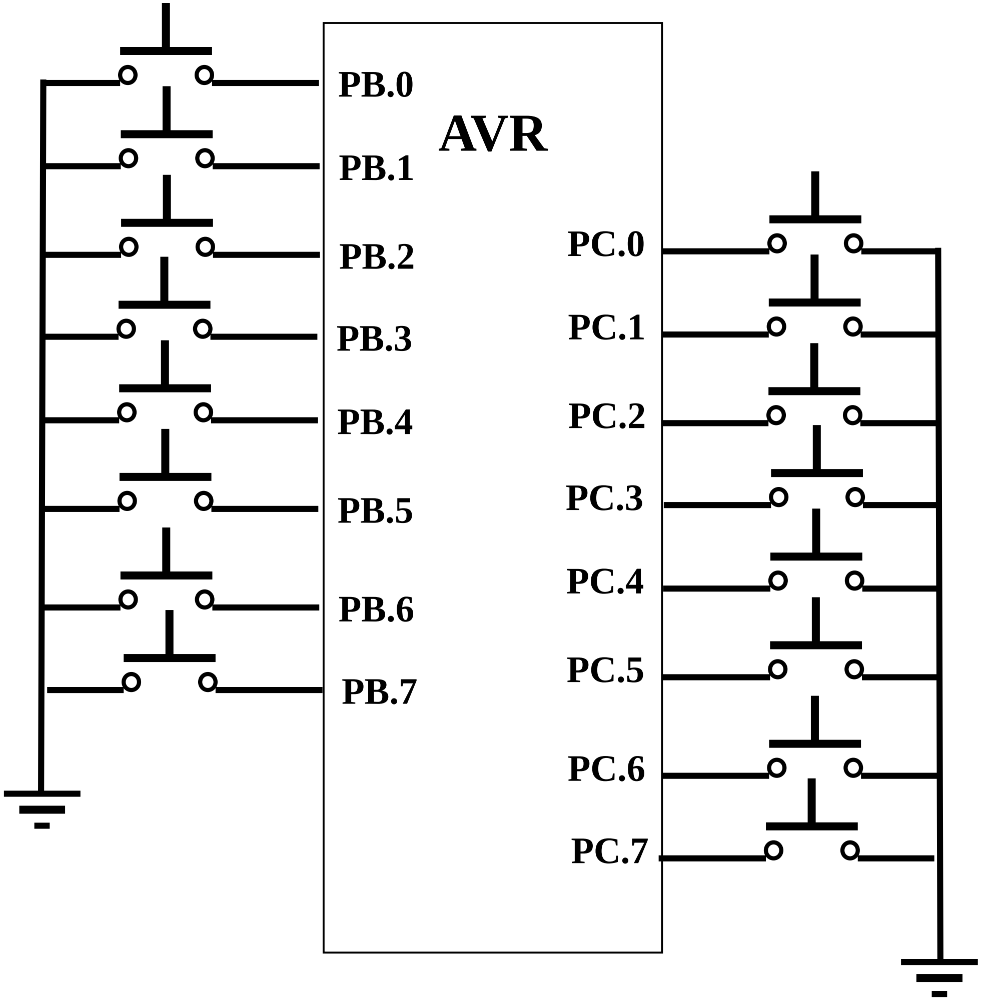
Semester 2, 2021/22

Topics

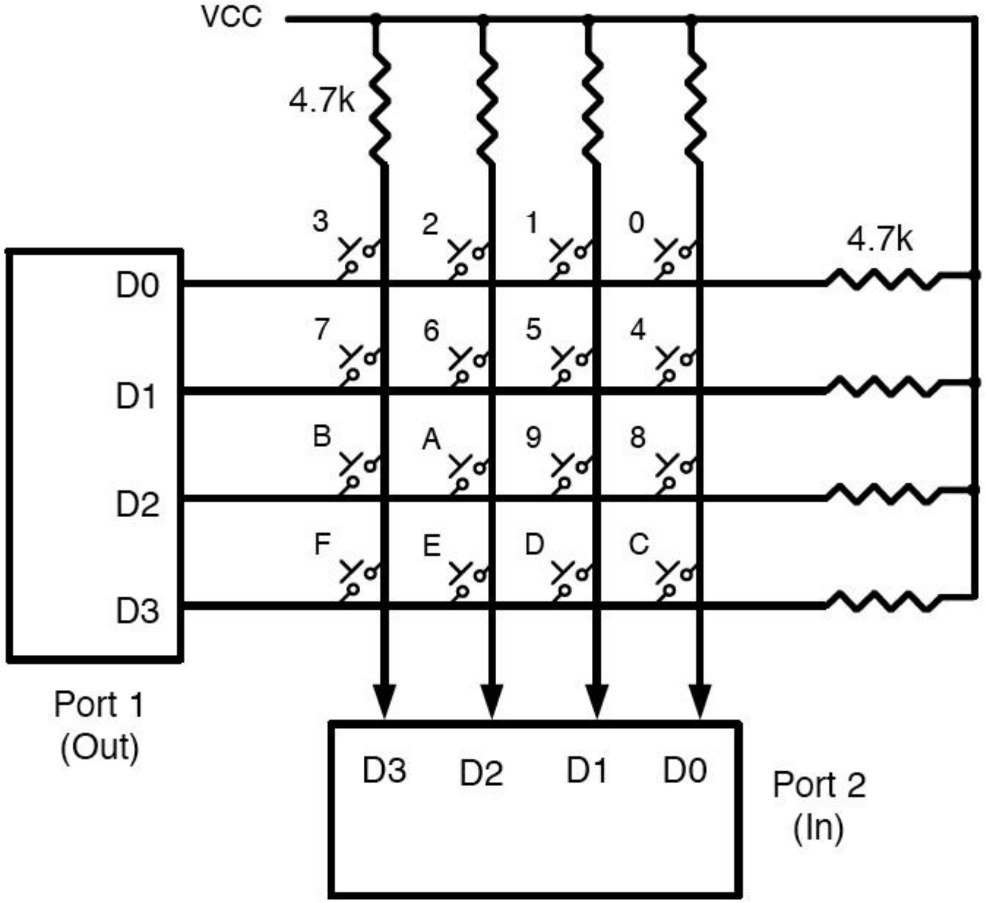
- Keyboard
  - Key identification
  - Programming
- Remote Control
- Debouncing Buttons

Keyboard

- If we connect each key to a pin of the AVR, we use so many pins. So we use scanning as shown in the next slide.



Keyboard


VCC
4.7k
4.7k
3
2
1
0
7
6
5
4
B
A
9
8
F
E
D
C
D0
D1
D2
D3
Port 1
(Out)
D3
D2
D1
D0
Port 2
(In)

Key identification

- Output “0000” to port 1 and read D3-D0 from port 2.
- If “1111” is read, that means no key is pressed.
- If one of the column bits has a zero, that means a key is pressed at that column.
  - Example: If key ‘8’ is pressed, the data read from port 2 should be “1110” (first detection).

Key identification

- Starting with the top row, the microcontroller grounds it by providing a low to each row (one row at one time only).
    - 0111, 1011, 1101, 1110
- If the data read is all 1s, no key in that row is pressed.
- If the data read has one 0, that means the key with the corresponding row and column is pressed.

Key identification

- Key ‘8’ is pressed:
  - P1 outputs 0111 → P2 obtains 1111
  - P1 outputs 1011 → P2 obtains 1110
  - P1 outputs 1101 → P2 obtains 1111
  - P1 outputs 1110 → P2 obtains 1111

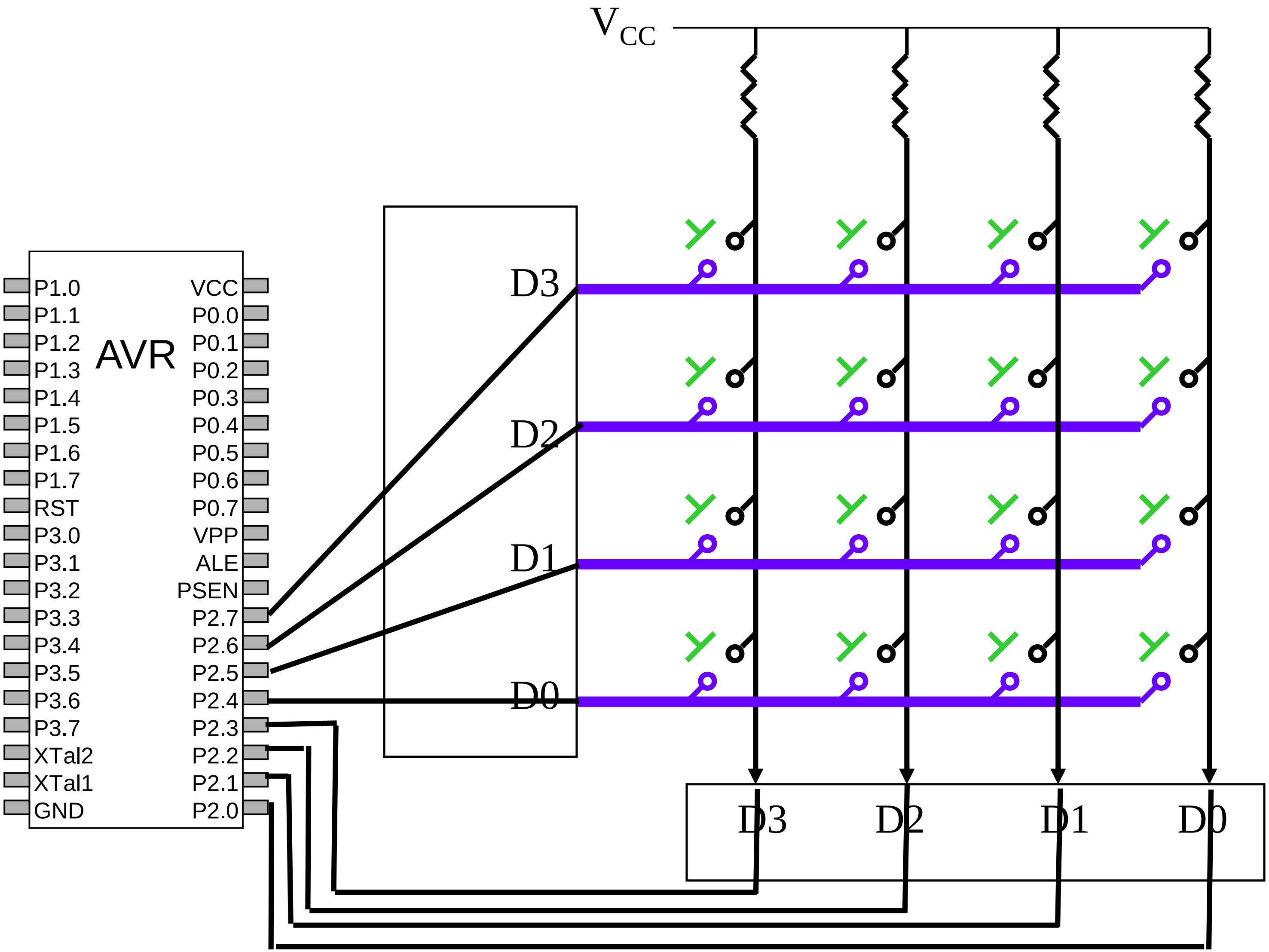
Another Example

- Identify the which key is pressed according to the following data:
  a. Column (first detection): 1101; Row (second detection): output 1110 and obtain 1101
  b. Column (first detection): 0111; Row (second detection): output 1011 and obtain 0111

Another Example

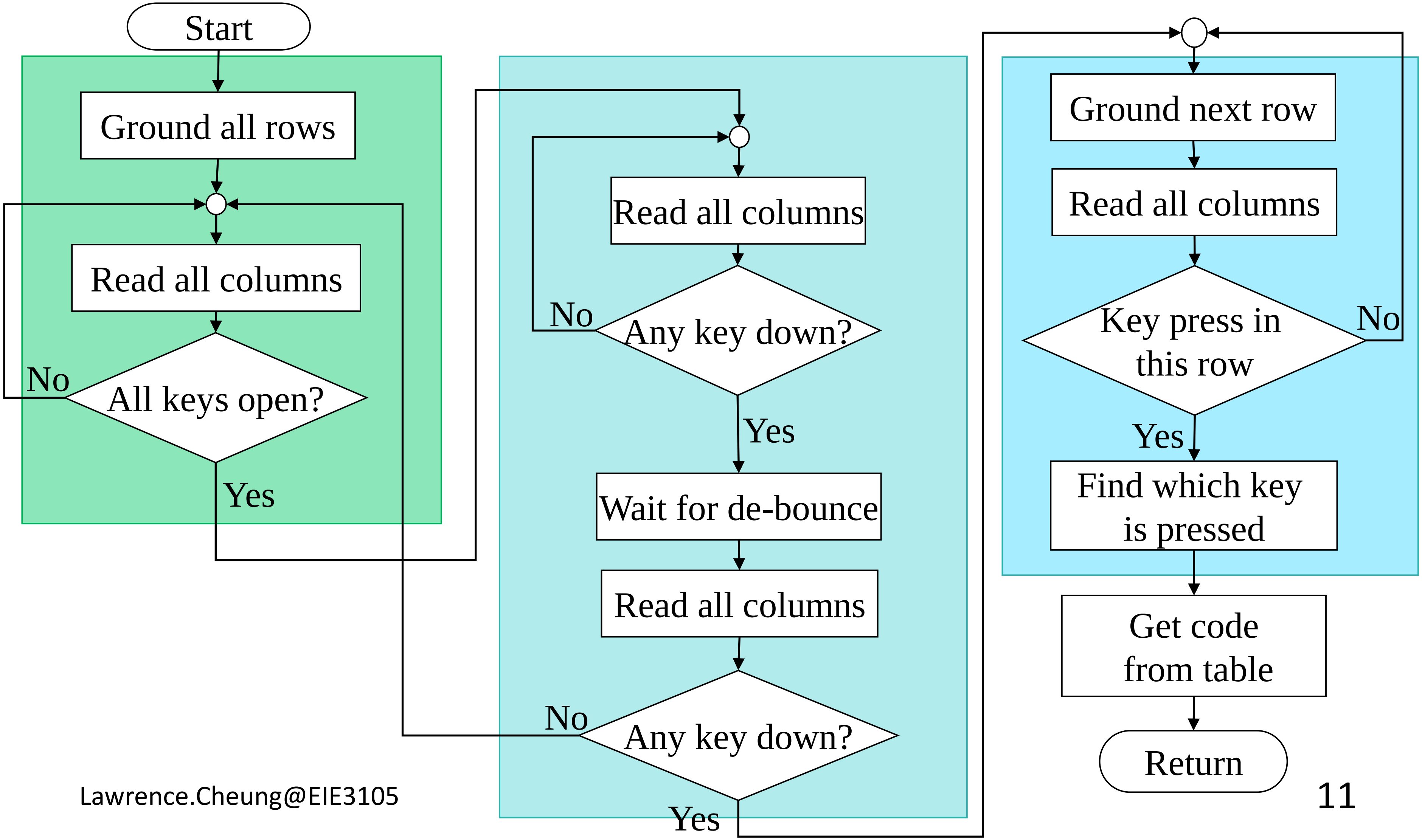
a. Key 1
b. Key B

Programming



Programming

- Algorithm



Programming

- The algorithm to detect and identify a key:
  1. To make sure that the preceding key has been released, 0s are output to all rows at once and the columns are read and checked repeatedly until all the columns are high.
  2. To see if a key is pressed, the columns are scanned over and over in an infinite loop until one of them has a 0 on it. After the key press detection, the microcontroller waits 20 ms for the bounce and hence scans the columns again.

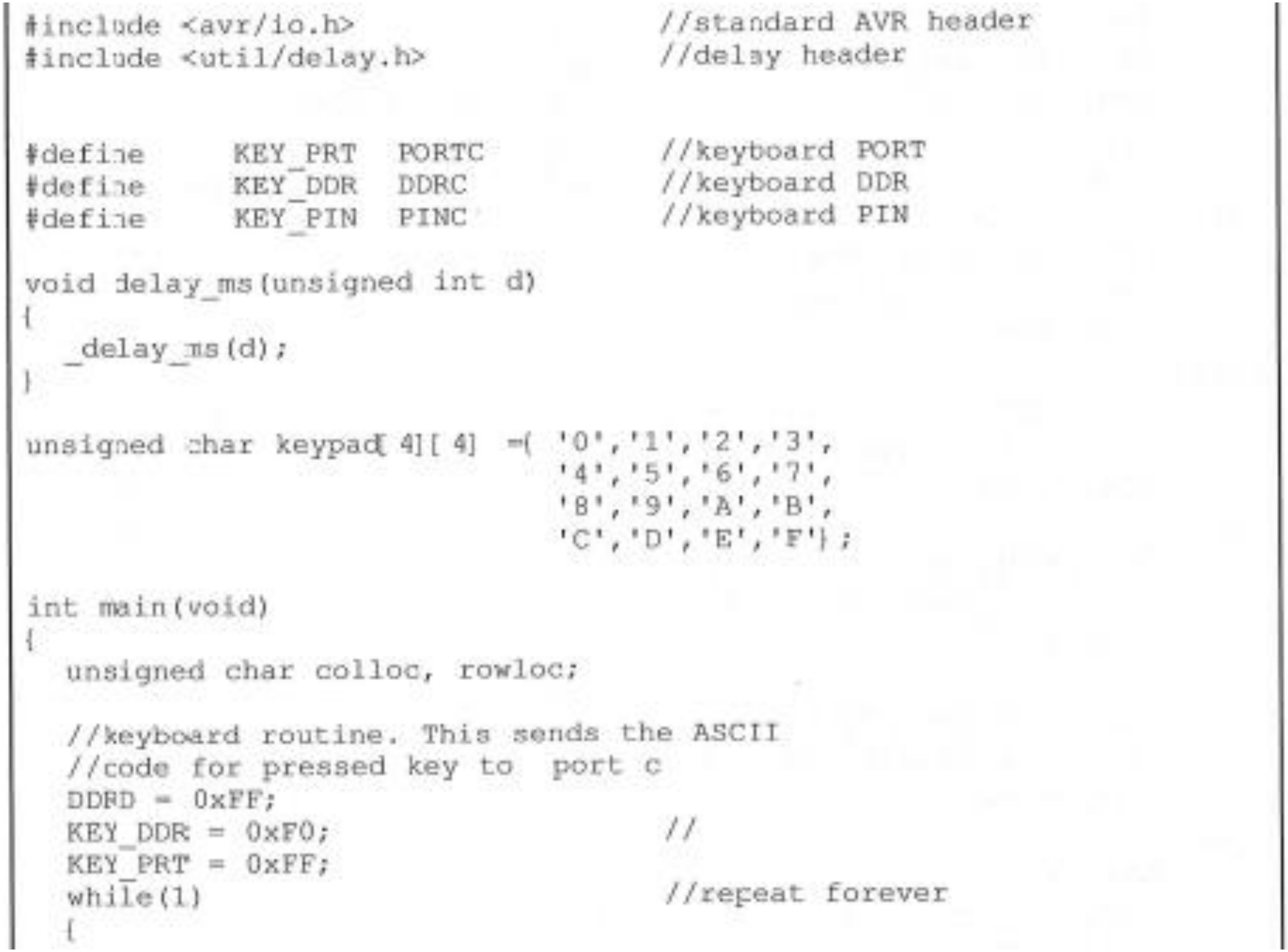
Programming

3. To detect which row the key belongs to, the microcontroller grounds one row at a time, reading the columns each time until it finds the row the key press belongs to.
4. Based on the row and the column, it pulls out the ASCII code for that key from the look-up table.

Programming

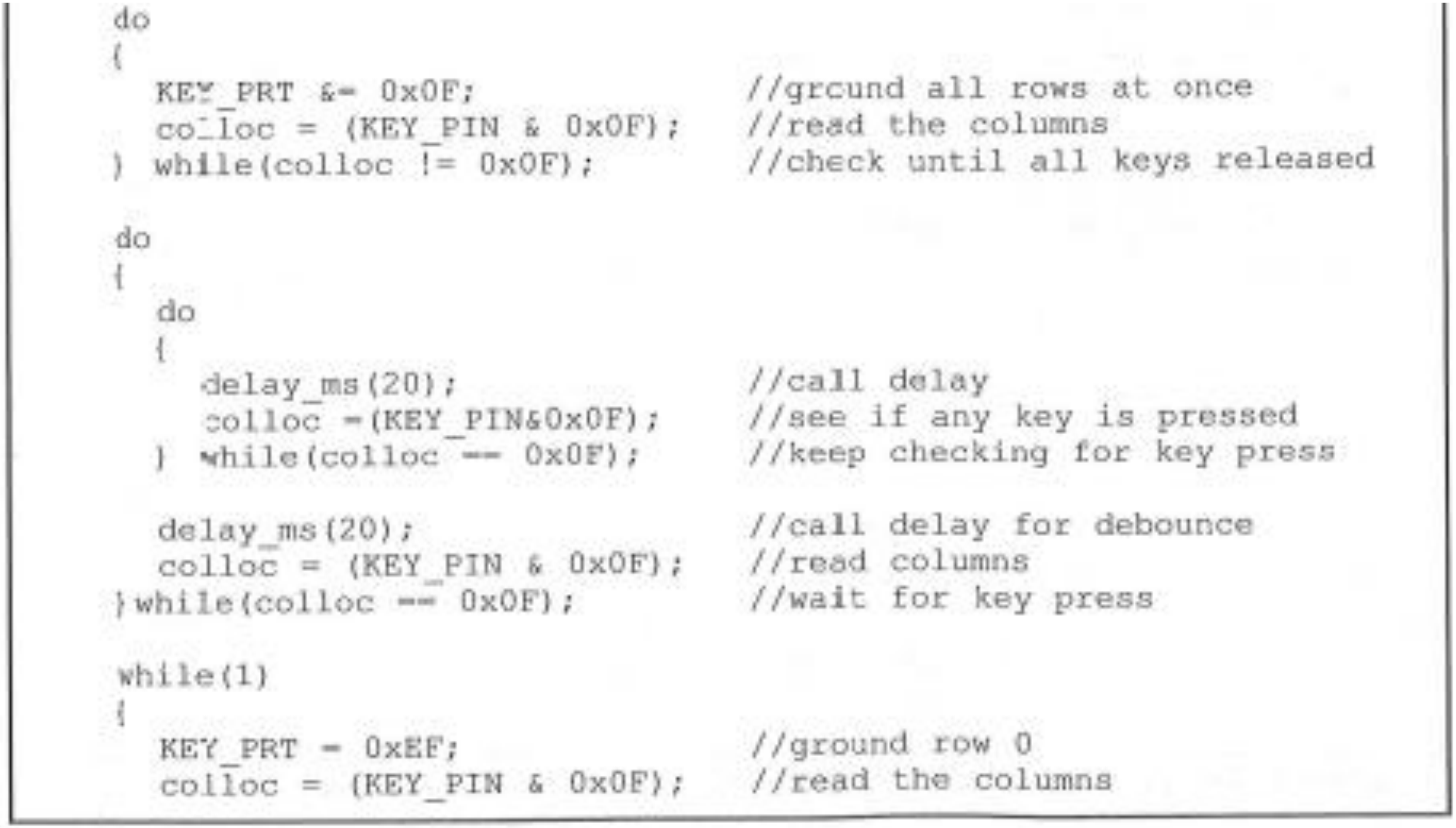
- Example: Write a C program to red the keypad and send the result to Port D.
  - PC0 to PC3 are connected to columns.
  - PC4 to PC7 are connected to rows.

Programming



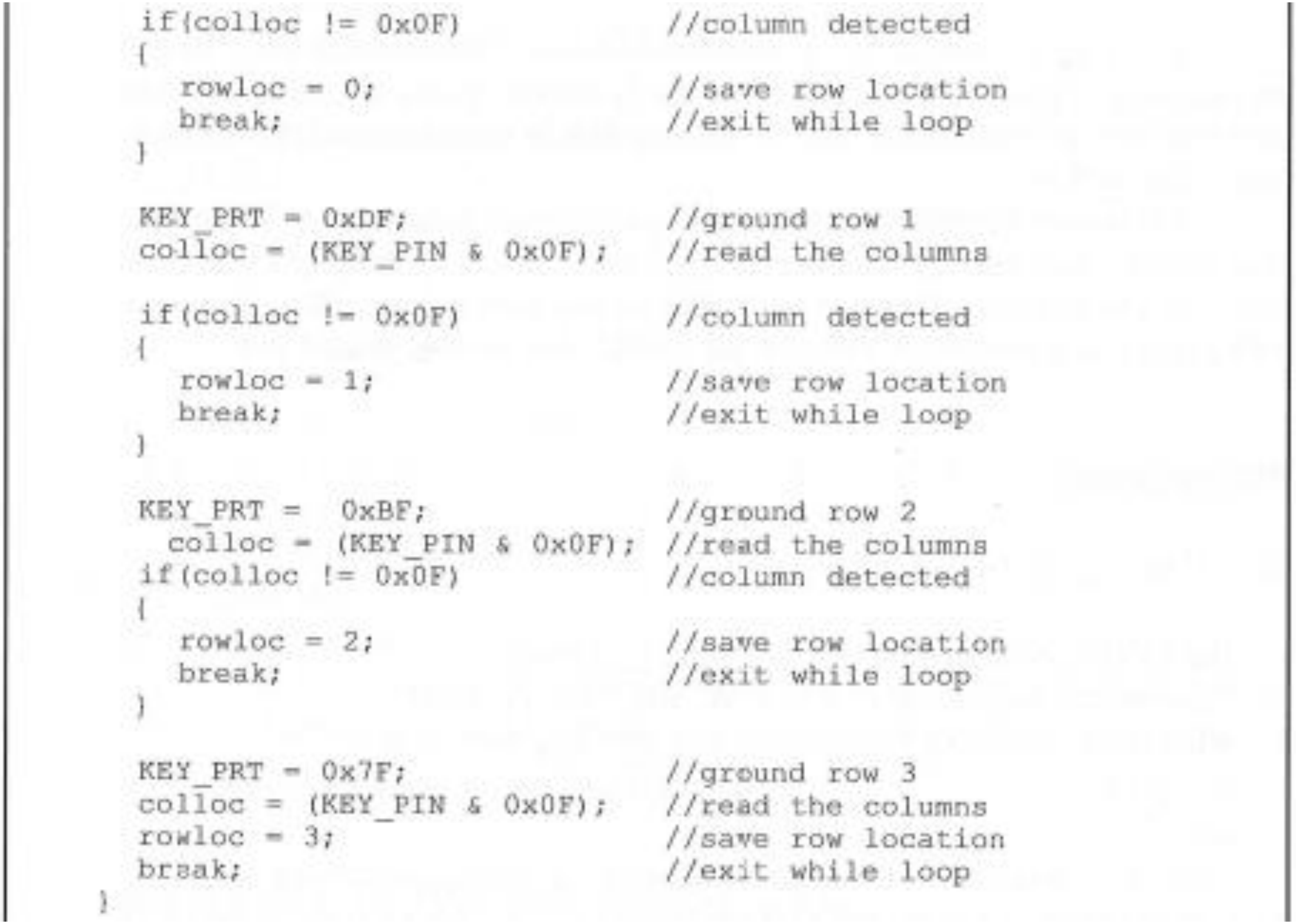
```
#include <avr/io.h>                    //standard AVR header
#include <util/delay.h>                //delay header


#define     KEY_PRT   PORTC            //keyboard PORT
#define     KEY_DDR   DDRC             //keyboard DDR
#define     KEY_PIN   PINC             //keyboard PIN

void delay_ms(unsigned int d)
{
  _delay_ms(d);
}

unsigned char keypad[ 4][ 4] ={ '0','1','2','3',
                                '4','5','6','7',
                                '8','9','A','B',
                                'C','D','E','F'};


int main(void)
{
  unsigned char colloc, rowloc;

  //keyboard routine. This sends the ASCII
  //code for pressed key to  port c
  DDRD = 0xFF;
  KEY_DDR = 0xF0;                      //
  KEY_PRT = 0xFF;
  while(1)                             //repeat forever
  {
```

Programming



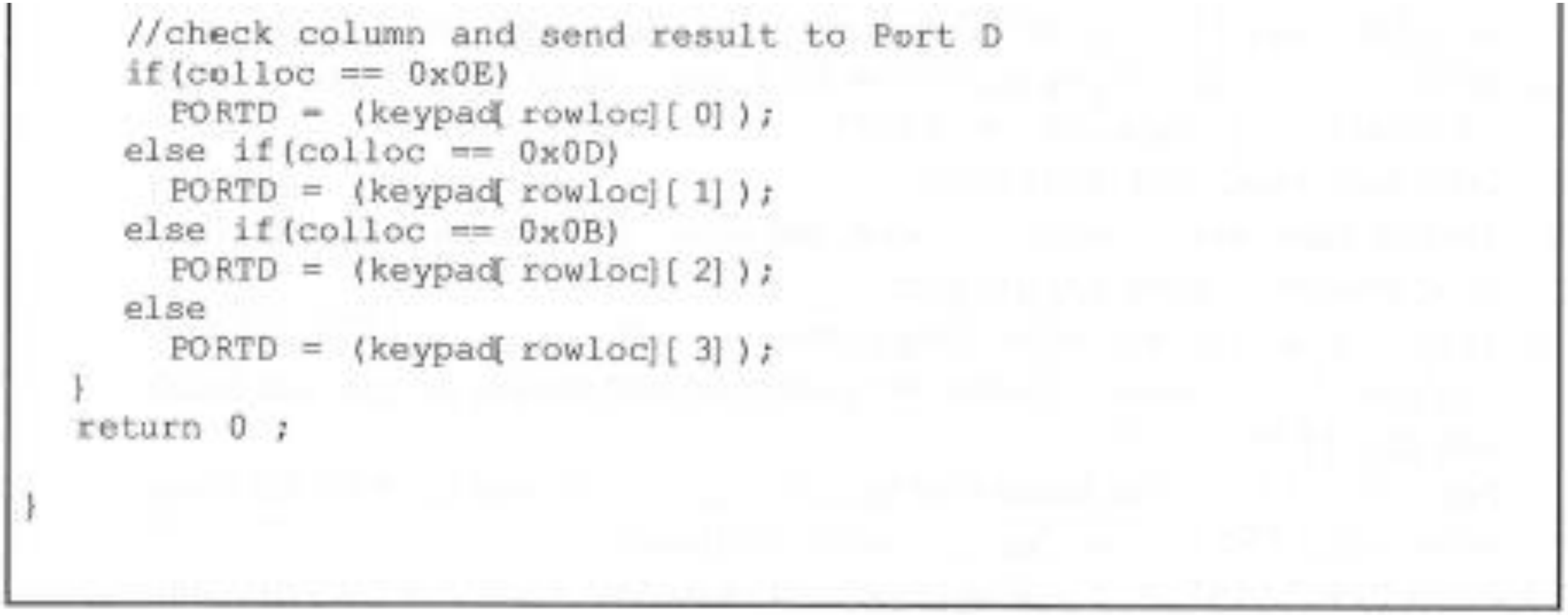
```
do
{
   KEY_PRT &= 0x0F;                  //grcund all rows at once
   colloc = (KEY_PIN & 0x0F);        //read the columns
} while(colloc != 0x0F);             //check until all keys released

do
{
   do
   {
      delay_ms(20);                  //call delay
      colloc =(KEY_PIN&0x0F);        //see if any key is pressed
   } while(colloc == 0x0F);          //keep checking for key press

   delay_ms(20);                     //call delay for debounce
   colloc = (KEY_PIN & 0x0F);        //read columns
}while(colloc == 0x0F);              //wait for key press

while(1)
{
   KEY_PRT = 0xEF;                   //ground row 0
   colloc = (KEY_PIN & 0x0F);        //read the columns
```

Programming



```
   if(colloc != 0x0F)          //column detected
   {
     rowloc = 0;               //save row location
     break;                    //exit while loop
   }

   KEY_PRT = 0xDF;             //ground row 1
   colloc = (KEY_PIN & 0x0F);  //read the columns

   if(colloc != 0x0F)          //column detected
   {
     rowloc = 1;               //save row location
     break;                    //exit while loop
   }

   KEY_PRT =  0xBF;            //ground row 2
     colloc = (KEY_PIN & 0x0F); //read the columns
   if(colloc != 0x0F)          //column detected
   {
     rowloc = 2;               //save row location
     break;                    //exit while loop
   }

   KEY_PRT = 0x7F;             //ground row 3
   colloc = (KEY_PIN & 0x0F);  //read the columns
   rowloc = 3;                 //save row location
   brsak;                      //exit while loop
 }
```

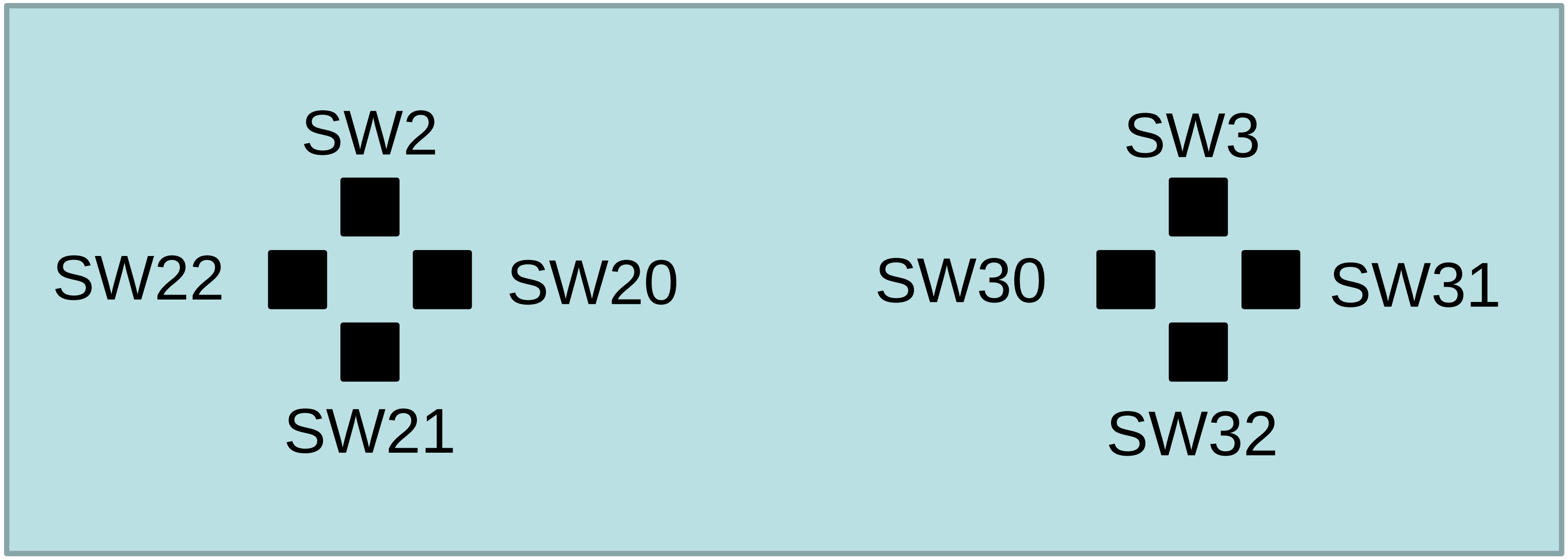
Programming



```
    //check column and send result to Port D
    if(colloc == 0x0E)
      PORTD = (keypad[ rowloc][ 0] );
    else if(colloc == 0x0D)
      PORTD = (keypad[ rowloc][ 1] );
    else if(colloc == 0x0B)
      PORTD = (keypad[ rowloc][ 2] );
    else
      PORTD = (keypad[ rowloc][ 3] );
  }
  return 0 ;

}
```

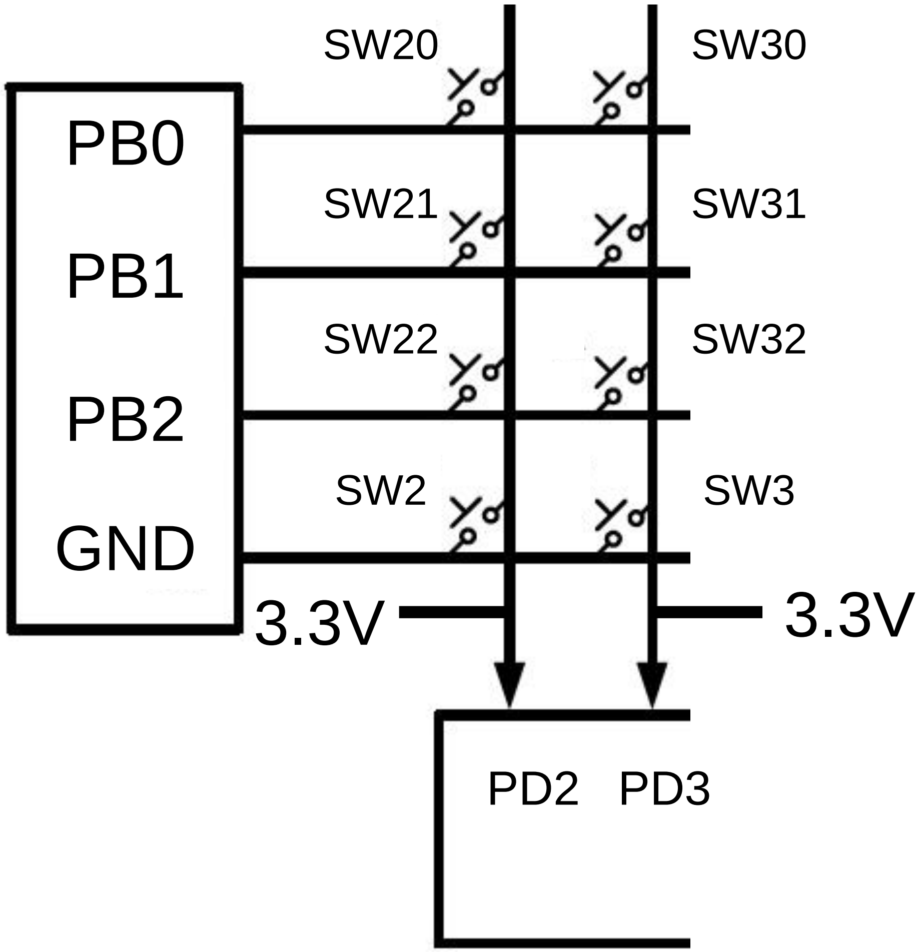
Remote Control

- Remote control block diagram



Remote Control

- Remote control schematic diagram



Remote Control

- Algorithm:
  - Initialization
    - Set PB0, PB1 and PB2 as output pins.
    - Set PD2 and PD3 as input pins (pull-up).
    - Output "000" to PB0, PB1 and PB2.
    - If no key is pressed, the reading should be "11" (it means PD2 = 1 and PD3 = 1).
  - If one of the column bits has a zero, that means a key is pressed at that column.

Remote Control

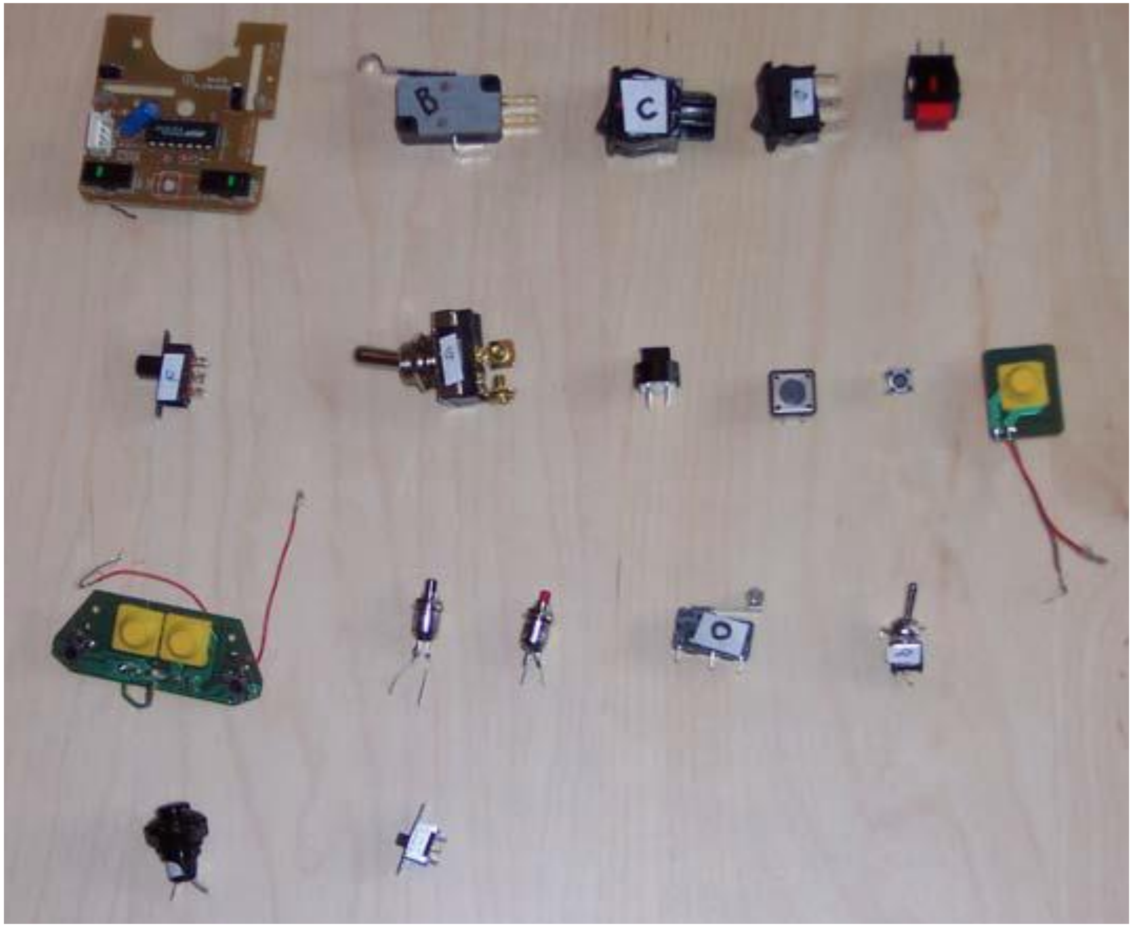
- Example: If key “SW21” is pressed, the reading should be “01” (first detection).
- Starting with the top row, the microcontroller grounds it by providing a low to each row (one row at one time only).
  - Output “011” to PB0, PB1 and PB2.
  - Then, output “101” to PB0, PB1 and PB2.
  - Then, output “110” to PB0, PB1 and PB2.
- The reading should be 11, 01 and 11 respectively.
- Thus, the key “SW21” is detected.

Remote Control

- How about “SW2”?
  - If key “SW2” is pressed, the reading should be “01” (first detection).
  - Starting with the top row, the microcontroller grounds it by providing a low to each row (one row at one time only).
  - Then the reading should always be 01.
  - Thus, the key “SW2” is detected.
- Reminder: do the de-bouncing by yourself.

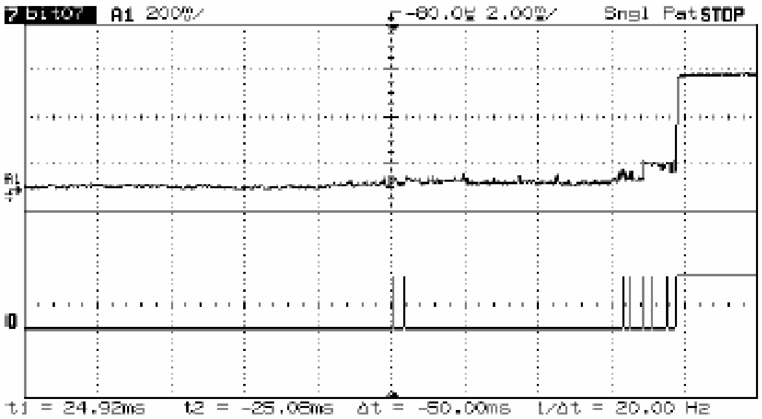
Buttons/Switches

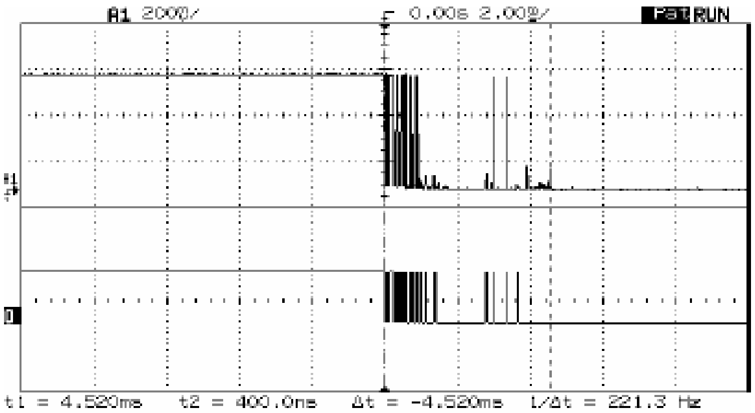
- All buttons (switches) are bouncing.

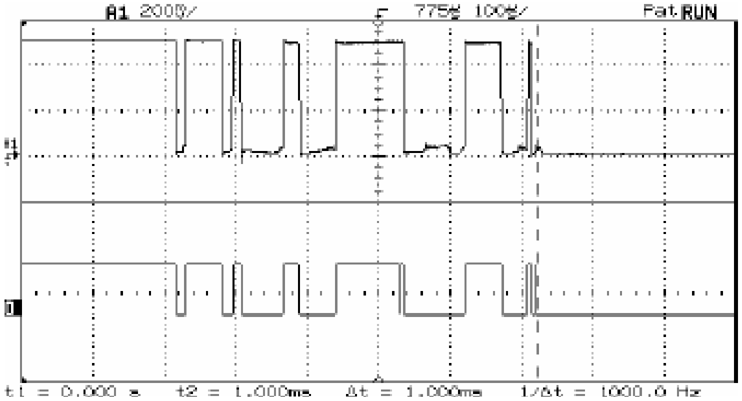


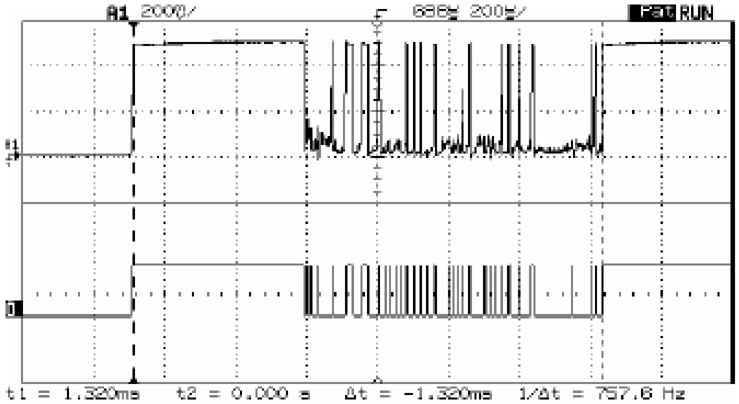
Button (Switch) Behaviours

- Different buttons (switches) have different bouncing behaviours.



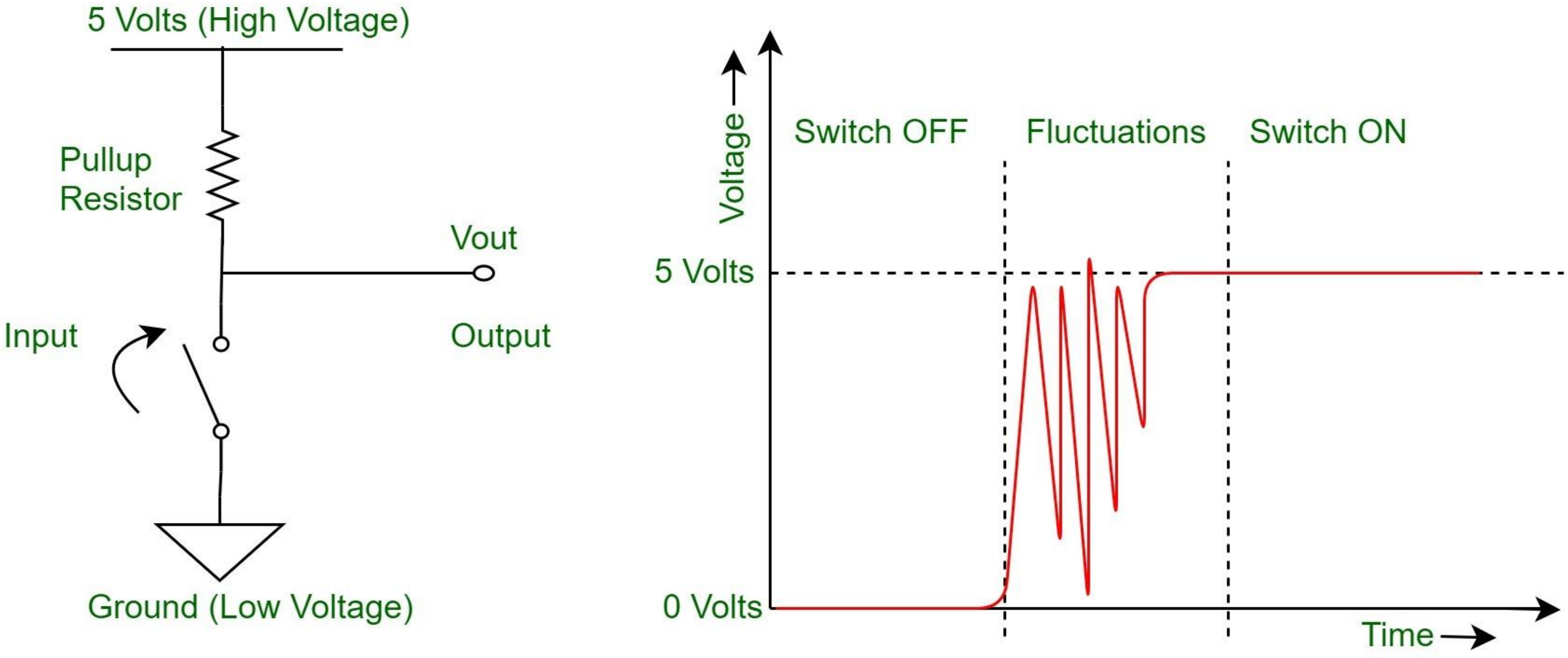






Button (Switch) Behaviours

- Pressing a button (switch)

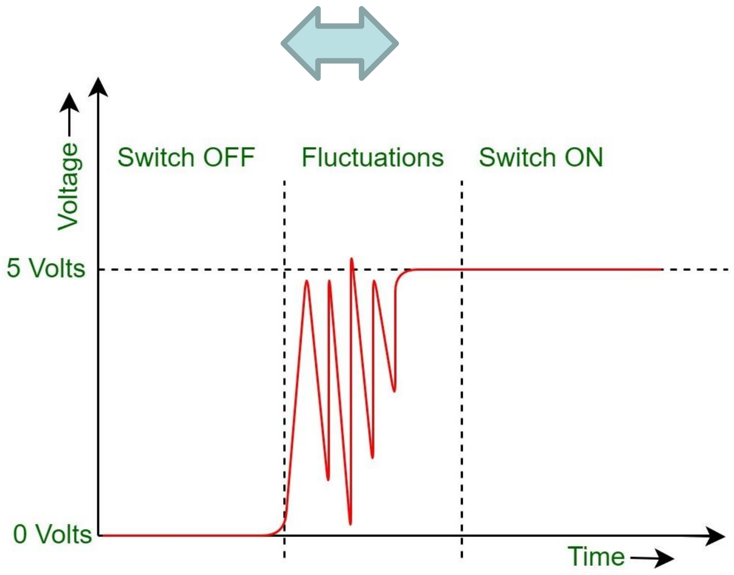


□The image comes from https://www.geeksforgeeks.org

Button (Switch) Behaviours

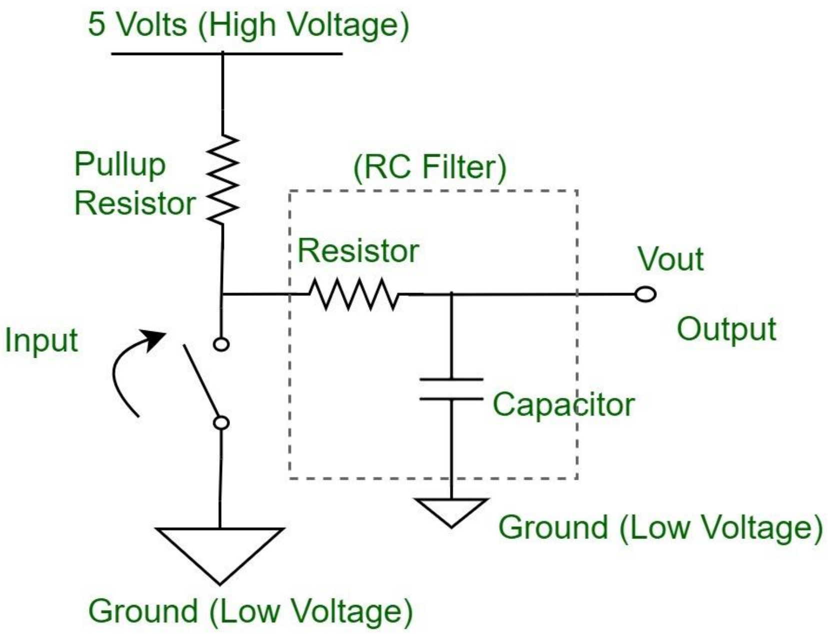
- The bouncing period
  - 100 ns are common.
  - Most of them less than 10 ms
  - Longest: 157 ms!
  - Conservative: 20 ms
  - Standard: < 200 ms

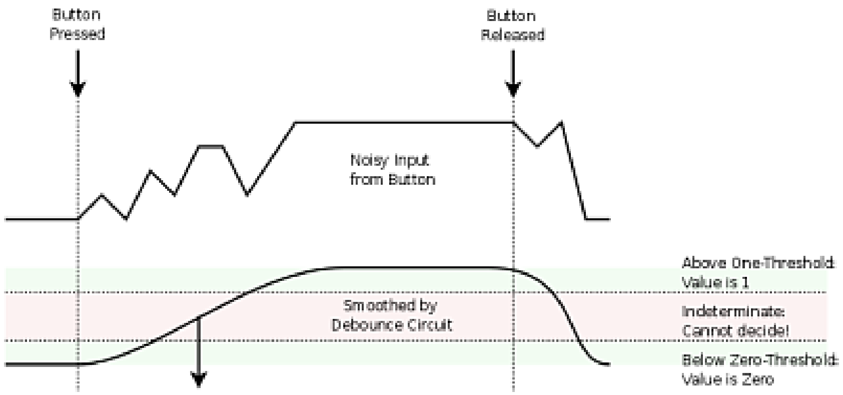
□Bouncing period



De-Bouncing Button (Hardware)

- Method 1: RC circuit
  - Act as a filter to smooth out the output.




Button Pressed
Button Released
Noisy Input from Button
Smoothed by Debounce Circuit
Above One-Threshold: Value is 1
Indeterminate: Cannot decide!
Below Zero-Threshold: Value is Zero

□The image comes from https://www.geeksforgeeks.org

□The image comes from hbfs.files.wordpress.com

De-Bouncing Button (Hardware)

- What are the values of R and C?

$$V_c = V_i e^{-\frac{t}{RC}}$$

where

V_c is the voltage across the capacitor at time t,

V_i is the initial voltage across the capacitor,

R and *C* are the values of the resistor and capacitor.

- Given *C*, find *R*

$$R = -\frac{t}{Cln\left(\frac{V_{th}}{V_i}\right)}$$

where V_{th} is the worst case for a signal going low.

De-Bouncing Button (Hardware)

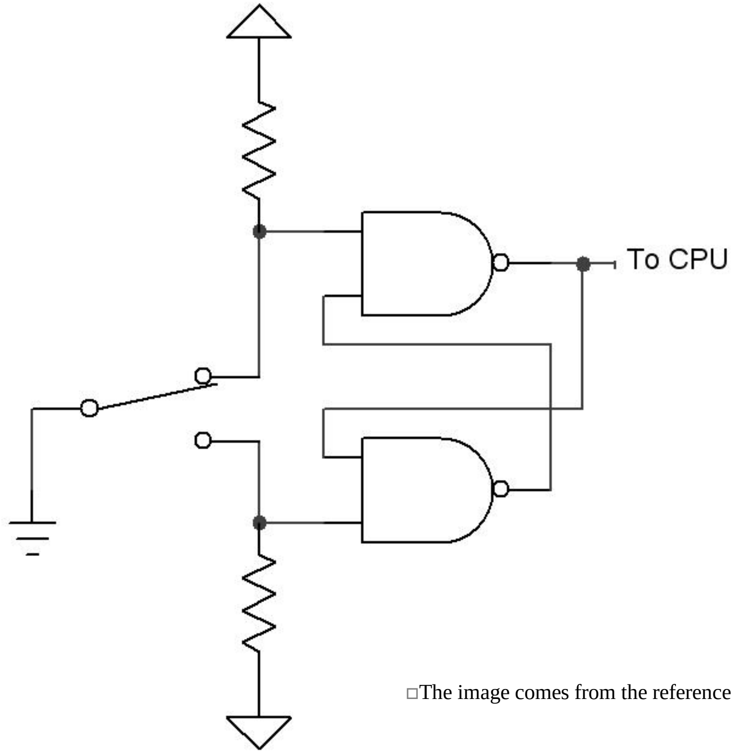
- Example

$$R = -\frac{t}{Cln\left(\frac{V_{th}}{V_i}\right)}$$

  - V_{th} = 1.7 V, $V_{initial}$ = 5V, C = 0.1 μF, t = 20 ms
  - Then R = 185 kΩ
  - Take R as a 180 kΩ or 200 kΩ resistor.

De-Bouncing Button (Hardware)

- Method 2: Using NAND gates



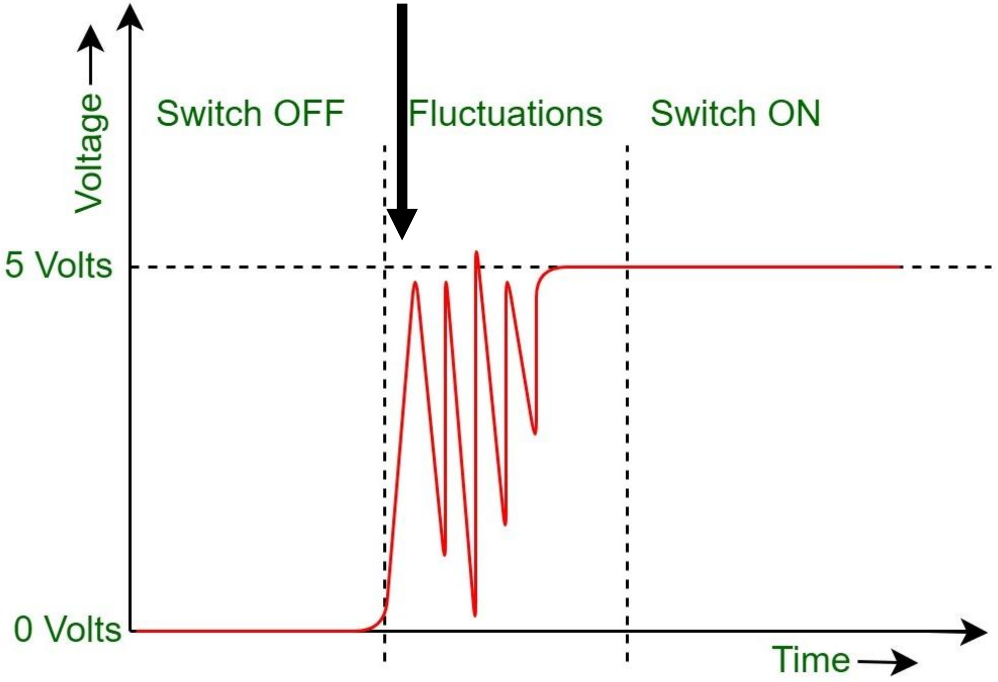
□The image comes from the reference

  - Two NOT gates for the output is also fine.

De-Bouncing Button (Software)

- Method 1 (Polling): Get the first signal and ignore the rest of them.

Read the signal and get next one after 1 second



De-Bouncing Button (Software)

```
#include <avr/io.h>

void delay_1s()
 {
    TCCR1A = 0b00000000; // CTC TOP = OCR1A
    TCCR1B = 0b00001011; // Prescaler = 64
    OCR1A = 12500; // 1 / (16 MHz / 64) = 0.004 ms,  50ms / 0.004 ms = 12500

    unsigned char count = 0;
    while (count < 20) // 20 x 50 ms = 1 s
    {
        while(!(TIFR1 & (1<<OCF1A))) ;
        TIFR1 = (1<<OCF1A);
        count++;
    }
 }
```

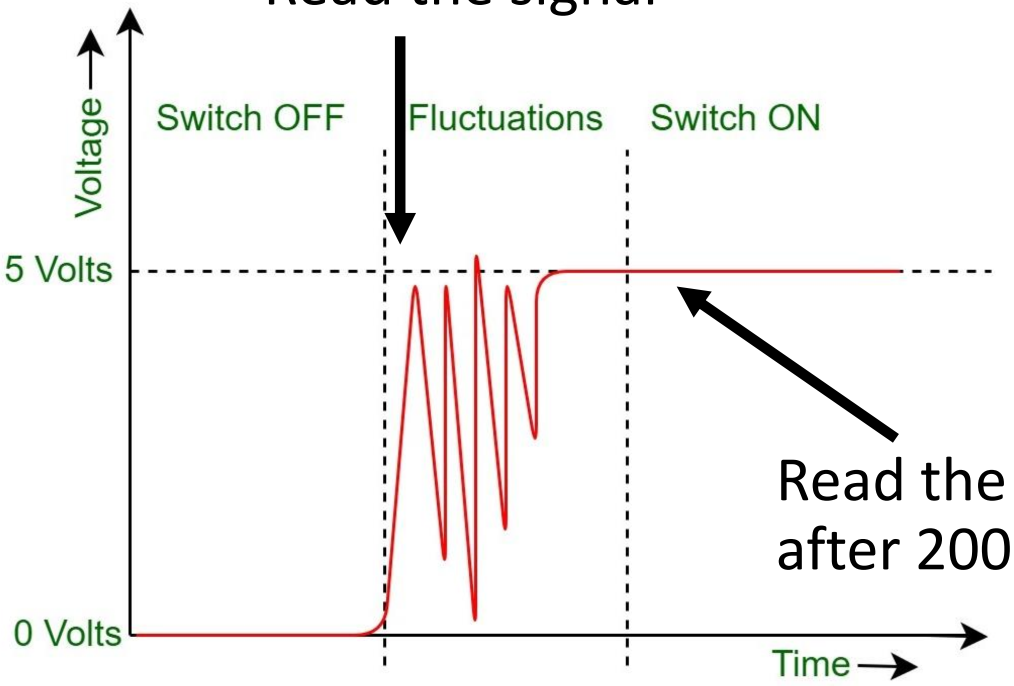
De-Bouncing Button (Software)

```
int main(void)
 {
    DDRB = 0x00;
    DDRD = 0xFF;

    unsigned char state = 0;
    while(1){
        if(PINB & (1 << 0)) // PB0 is pressed
        {
          if(state == 0){
                   state = 1;
                   PORTD = 0xFF; // any PDx is ON
          }
          else{
                   state = 0;
                   PORTD = 0x00; // any PDx is OFF
          }
          delay_1s(); // remove the debouncing effect
        }
    }
 }
```

De-Bouncing Button (Software)

- Method 2 (Polling): Get the first signal and check it again after 200 ms.

Read the signal



Read the signal again after 200 ms

De-Bouncing Button (Software)

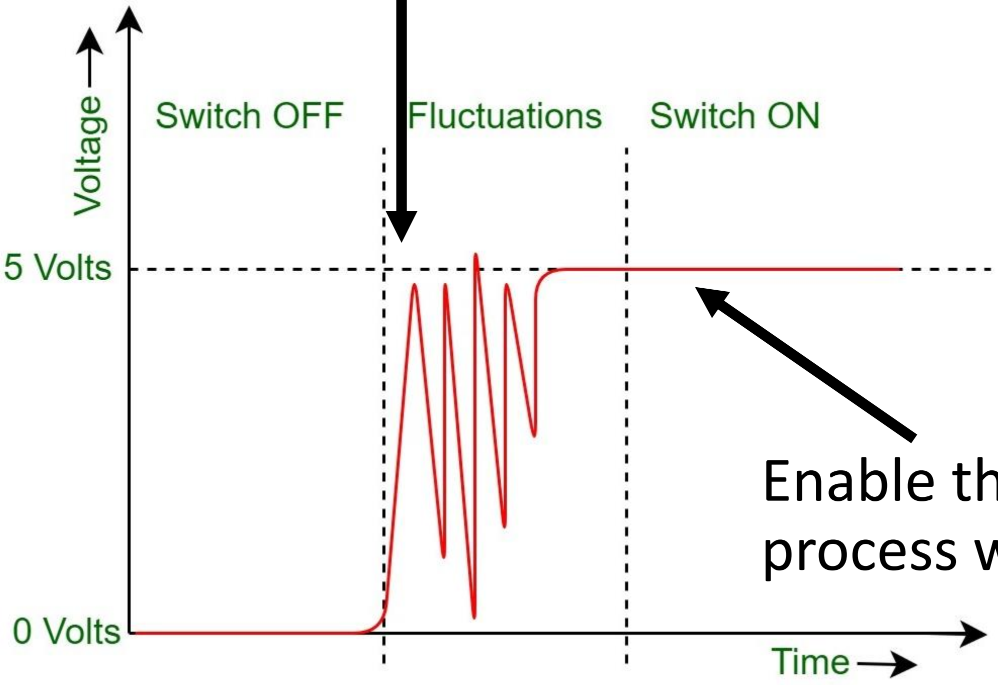
```
#include <avr/io.h>

void delay_200ms()
{
   TCCR1A = 0b00000000; // CTC TOP = OCR1A
   TCCR1B = 0b00001011; // Prescaler = 64
   OCR1A = 12500; // 1 / (16 MHz / 64) = 0.004 ms,  50ms / 0.004 ms = 12500

   unsigned char count = 0;
   while (count < 4) // 4 x 50 ms = 200 ms
   {
      while(!(TIFR1 & (1<<OCF1A))) ;
      TIFR1 = (1<<OCF1A);
      count++;
   }
}
```

De-Bouncing Button (Software)

```
int main(void){
    DDRB = 0x00;
    DDRD = 0xFF;
    unsigned char state = 0;
    while(1){
       if(PINB & (1 << 0)) // PB0 is pressed
       {
         delay_200ms(); // remove the debouncing effect
         if(PINB & (1 << 0)){
                  if(state == 0){
                           state = 1;
                           PORTD = 0xFF; // any PDx is ON
                  }
                  else{
                           state = 0;
                           PORTD = 0x00; // any PDx is OFF
                  }
         }
       }
    }
}
```

De-Bouncing Button (Software)

- Method 3 (Interrupt): Get the first signal from the interrupt. Then enable the timer and process when time-out.



Read the signal

Enable the timer and process when time-out

De-Bouncing Button (Software)

```
#include <avr/io.h>
#include <avr/interrupt.h>

unsigned char status = 0;
unsigned char state = 0;
unsigned char count = 0;

int main(void)
{
    DDRB = 0x00;
    DDRD = 0xF8;

    EIMSK = 0b00000001; //INT0 enable
    EICRA = 0b00000010; //INT0 falling edage interrupt ISC01:ISC00 1:0

    sei();

    while(1){}
}
```

De-Bouncing Button (Software)

```
ISR(TIMER1_COMPA_vect)
{
   count++;
   if (count == 4) // 4 x 50 ms = 200 ms
   {
      if(state == 0){
        state = 1;
        PORTD = 0xF8; // any PDx is ON
      }
      else{
        state = 0;
        PORTD = 0x00; // any PDx is OFF
      }
      count = 0;
      status = 0;

      // disable Timer 0
      TCCR1B = 0b00000000;
   }
}
```

De-Bouncing Button (Software)

```
ISR(INT0_vect)
{
    if (status == 0) // first time
    {
        status = 1;

        // enable Timer 0
        TCCR1A = 0b00000000; // CTC TOP = OCR1A
        TCCR1B = 0b00001011; // Prescaler = 64
        OCR1A = 12500; // 1 / (16 MHz / 64) = 0.004 ms,  50ms / 0.004 ms = 12500
        TIMSK1 = 0b00000010; //OCIE1A
    }
    if (status == 1); // not the first time, do nothing
}
```

Reference Readings

- Chapter 13 – *The AVR Microcontroller and Embedded Systems : Using Assembly and C*, M. A. Mazidi, S. Naimi, and S. Naimi, Pearson, 2014.
- Jack G. Ganssle, “A Guide to Debouncing”, technical report, 2008.

End