Microcontrollers

Embedded Programming

Team Emertxe

Important Terms

Microcontrollers

Important Terms

- Host:

  A system which is used to develop the target.

- Target:

  A system which is being developed for specific application.

- Cross Compiler:

  An application used to generated code for another architecture being in another architecture

EMERTXE

Lets Start Coding

Microcontrollers

Embedded Programming - Let's Start Coding

- Well, come on lets make our hand a bit dirty in embedded coding with the following code

Example

```c
#include <stdio.h>

int main()
{
    int x = 20;

    printf("%d\n", x);

    return 0;
}
```

- Nice, but few questions here
  - Why did you write this code?

    Hmm, Just to print x to embedded programming
  - Fine, where are you planning to run this code?

    Of course on a embedded target!
  - Does it have a OS already running?

    Ooink, Hmm noo, may be …

EMERTXE

Microcontrollers

Embedded Programming - Let's Start Coding

- So questions raised in the previous slide has to be answered before we can start our code.
- The answers to these questions are little tricky and depends on
  - Complexity of the work you do
  - The requirement of the project

  and many other factors
- Now the scope of this module is to learn low level microcontroller programming which is non OS (called as bare metal)
- So let's rewrite the same example as shown in the next slide

Microcontrollers

Embedded Programming - Let's Start Coding

Example

```
#include <stdio.h>

void main(void)
{
    int x = 20;

    printf("%d\n", x);
}
```

- The change you observe is **void main(void)**
- Why?
  - As mentioned generally the low end embedded system are non OS based
  - The code you write would be the first piece of code coming to existence
- Now, lets not take this too seriously. This could again depend the development environment
- There could be some startup codes, which would call the main

EMERTXE

Microcontrollers

Embedded Programming - Let's Start Coding

Example

```
#include <stdio.h>

void main(void)
{
    int x = 20;

    printf("%d\n", x);
}
```

- The next questions is, where are trying to print? On Screen?
  - Does your target support that?
  - Does your development environment support that?

- Now again, all these are depends on your target board and development environment
- Maaan, So many questions? Well, what should I write then?
  - Well that too depends on your target board!! 😜

EMERTXE

Microcontrollers

Embedded Programming - Let's Start Coding

- Well, my principle is simple. No matter on what type of board you work, the first code you write, should give you the confidence that you are on the right path.
- Try to identify the simplest possible interface which can be made work with lesser overhead, so that, we are sure about our setup like
  - Hardware is working
  - Toolchain setup is working
  - Connectivity between the host and target is established

  and so on.

Microcontrollers

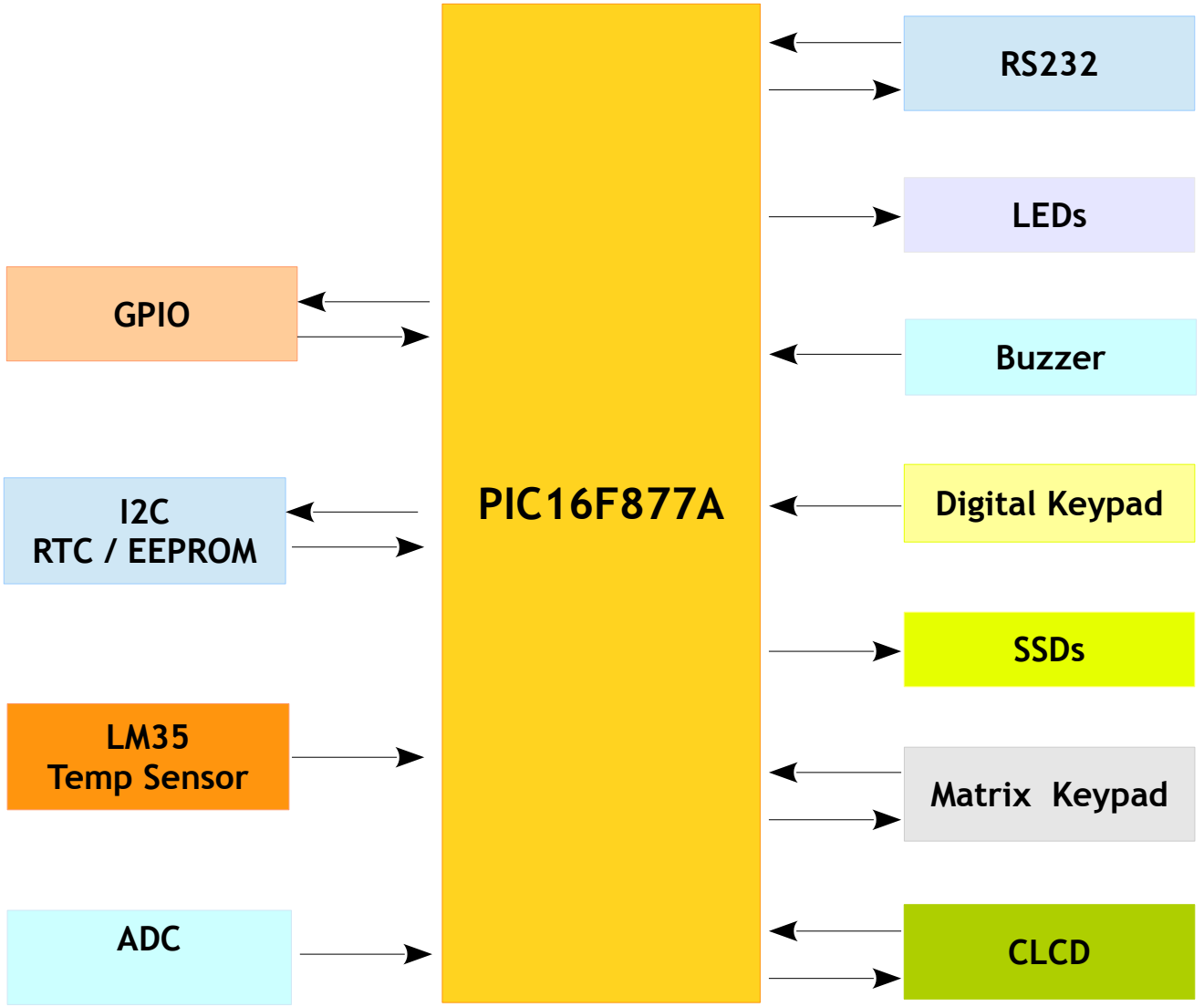
Embedded Programming - Let's Start Coding

- It is good to know what your target board is, what it contains by its architecture
- Board architecture generally gives you overview about your board and its peripheral interfaces
- In our case we will be using PICSimLab simulator board which is shown in the next slide

EMERTXE

Microcontrollers

EP - Let's Start Coding - PICSimLab Architecture



Microcontrollers

Embedded Programming - Let's Start Coding

- So from the architecture we come to know that the board has few LEDs, So why don't we start with it?

Example

```
#include <stdio.h>

void main(void)
{
    int led;

    led = 0;
}
```

- So simple right? Well I hope you know whats going happen with this code!!
- Any C programmer knows that the led is just a integer variable and we write just a value in it, hence no point in this code

Now what should we do?

- Hmm, refer next slide

EMERTXE

Microcontrollers

Embedded Programming - Let's Start Coding

- LED is an external device connected to microcontroller **port**
- A **port** is interface between the controller and external peripheral.
- Based on the controller architecture you will have **N** numbers of **ports**
- The target controller in PICGenios board is **PIC16F877A** from Microchip
- The next question arises is how do I know how many ports my target controller has?
  - From **Microcontroller Architecture** which will be detailed in the data sheet provided by the maker

Microcontrollers

Embedded Programming - Let's Start Coding

- By reading the data sheet you come to know that there are 5 Ports
- Again a question. Where are the LEDs connected. You need the **Schematic** of the target board to know this.
- A **Schematic** is document which provides information about the physical connections on the hardware.
- From the schematic we come to know the the LEDs are connected to PORTB and PORTD
- Port is a peripheral and we need need to know on how to access and address. This info will be available in the data sheet in **PORTB, PORTD** and **Data Memory** sections

EMERTXE

Microcontrollers

Embedded Programming - Let's Start Coding

- From the section of PORTB it clear that there is 1 more register associated with it named, **TRISB**
- The TRISB register is very important for IO configuration. The value put in this register would decide pin direction as shown below

TRIS Register		PORT Register		Pin Direction
1	TRISx7	?	Rx7	Input
0	TRISx6	?	Rx6	Output
1	TRISx5	?	Rx5	Input
1	TRISx4	?	Rx4	Input
0	TRISx3	?	Rx3	Output
1	TRISx2	?	Rx2	Input
1	TRISx1	?	Rx1	Input
0	TRISx0	?	Rx0	Output

Microcontrollers

Embedded Programming - Let's Start Coding

- So from previous slide its clear that we have to use the TRIS register to control the pin direction
- LEDs are driven by external source, so the port direction should be made as output
- In this case the LEDs are connected to the controller and will be driven by it
- Fine, what should write to the port to make it work?

  It depends on the hardware design.
- By considering all these point we can modify our code as shown in the next slide

EMERTXE

Microcontrollers

Embedded Programming - Let's Start Coding

Example

```
void main(void)
{
    /*
     * Defining a pointer to PORTB register at address 0x06,
     * pointing to 8 bit register. Refer data sheet
     */
    unsigned char *portb = (unsigned char *) 0x06;
    /*
     * Defining a pointer to PORTB tri-state register at address 0x86,
     * pointing to 8 bit register. Refer data sheet
     */
    unsigned char *trisb = (unsigned char *) 0x86;

    /* Setting the pin direction as output (0 - output and 1 - Input) */
    *trisb = 0x00;

    /*
     * Writing just a random value on the portb register where
     * LEDs are connected
     */
    *portb = 0x55;
}
```

EMERTXE

Microcontrollers

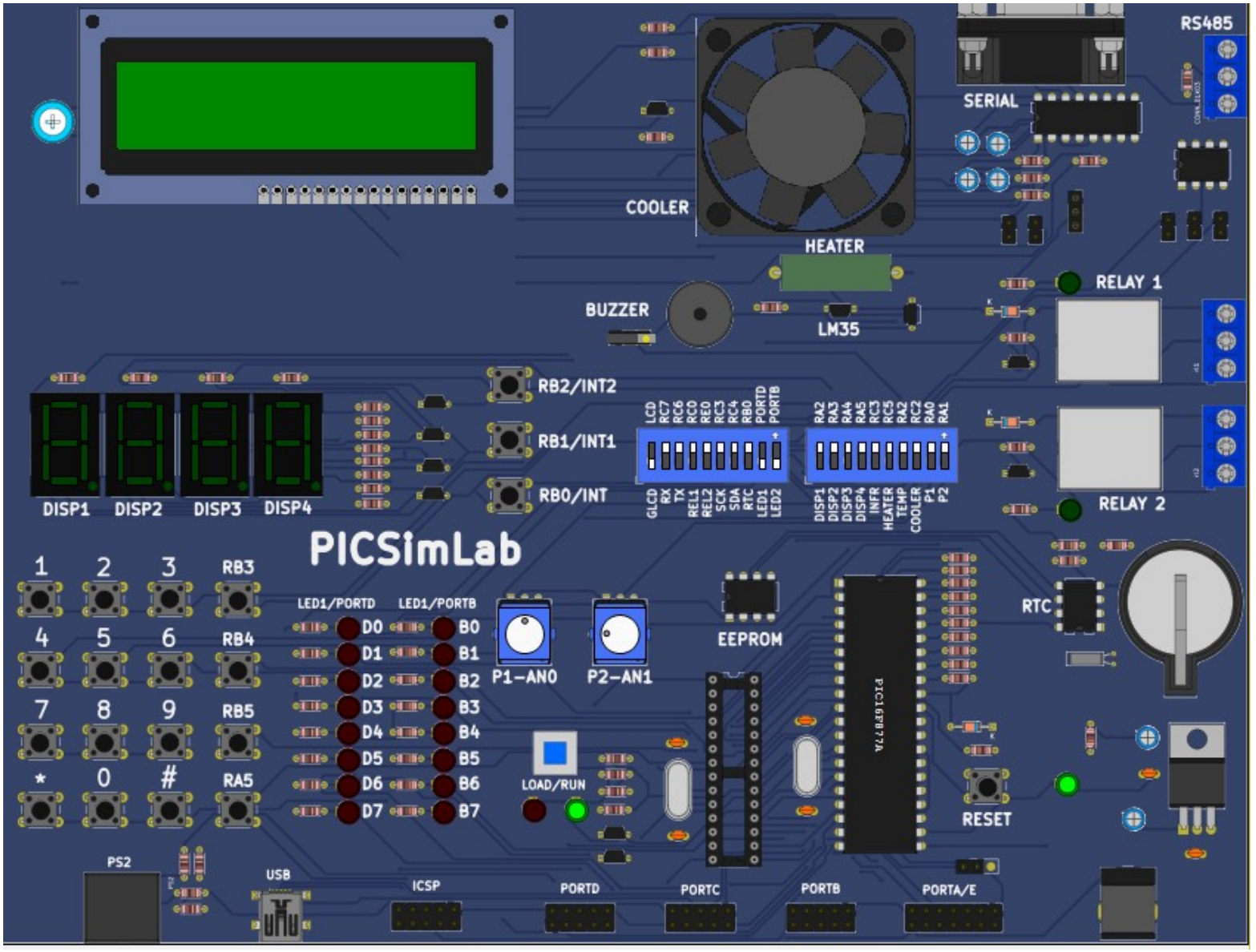
Embedded Programming - Let's Start Coding

- Hurray!!, we wrote our first Embedded C code for our target board
- Come on let's move forward, how do I compile this code?
- Obviously with a compiler!, Yes but a cross compiler since this code has to run on the target board.
- The target controller, as mentioned, is by Microchip. So we will be using **XC8 (Free Version)**
- You need to download it and install it in your system
  - You can use MPLABX

EMERTXE

Microcontrollers

EP - Let's Start Coding –PICSimLab Board



Microcontrollers

Embedded Programming - Let's Start Coding

- The thrill of having your first code working is different.
- But, this is just the beginning, you might like to design some good application based on your board
- Proceeding forward, the way how we wrote the code with indirect addressing would require good amount of time
- So it is common to use the definitions and libraries provided by the cross compiler to build our applications **else we end up** **“Reinventing the Wheel”**
- The same code can be re-written the the way provided in the next slide

EMERTXE

Microcontrollers

Embedded Programming - Let's Start Coding

Example

```
#include <xc8.h>

void main(void)
{
    /* Setting the pin direction as output (0 - output and 1 - Input) */
    TRISB = 0x00;

    /*
     * Writing just a random value on the data portb register where
     * LEDs are connected
     */
    PORTB = 0x55;
}
```

- So simple. Isn't it?

EMERTXE

Project Creation

Microcontrollers

Project Creation - Code Organization

- Please organize the code as shown below to increase productivity and modularity
- Every .c file should have .h file

```c
#inclu
void f
{

}
```

```c
#include "supporting_file.h"

void function_1(void)
{
        /* Function Code */
}
```

modules.c

```c
#include "main.h"

void init_config(void)
{
    /* Initilization Code */
}

void main(void)
{
    init_config();

    while (1)
    {
        /* Application Code */
    }
}cc
```

main.c

```c
#ifndef MAIN_H
#define MAIN_H

#include <htc.h>

#endif
```

main.h

```c
#ifndef SUPPORTING_FILE_H
#define SUPPORTING_FILE_H

#endif
```

modules.h

EMERTXE

Microcontrollers

Project Creation - Code Template

main.c

```c
#include "main.h"

void init_config(void)
{
    /* Initilization Code */
}

void main(void)
{
    init_config();

    while (1)
    {
        /* Application Code */
    }
}
```

main.h

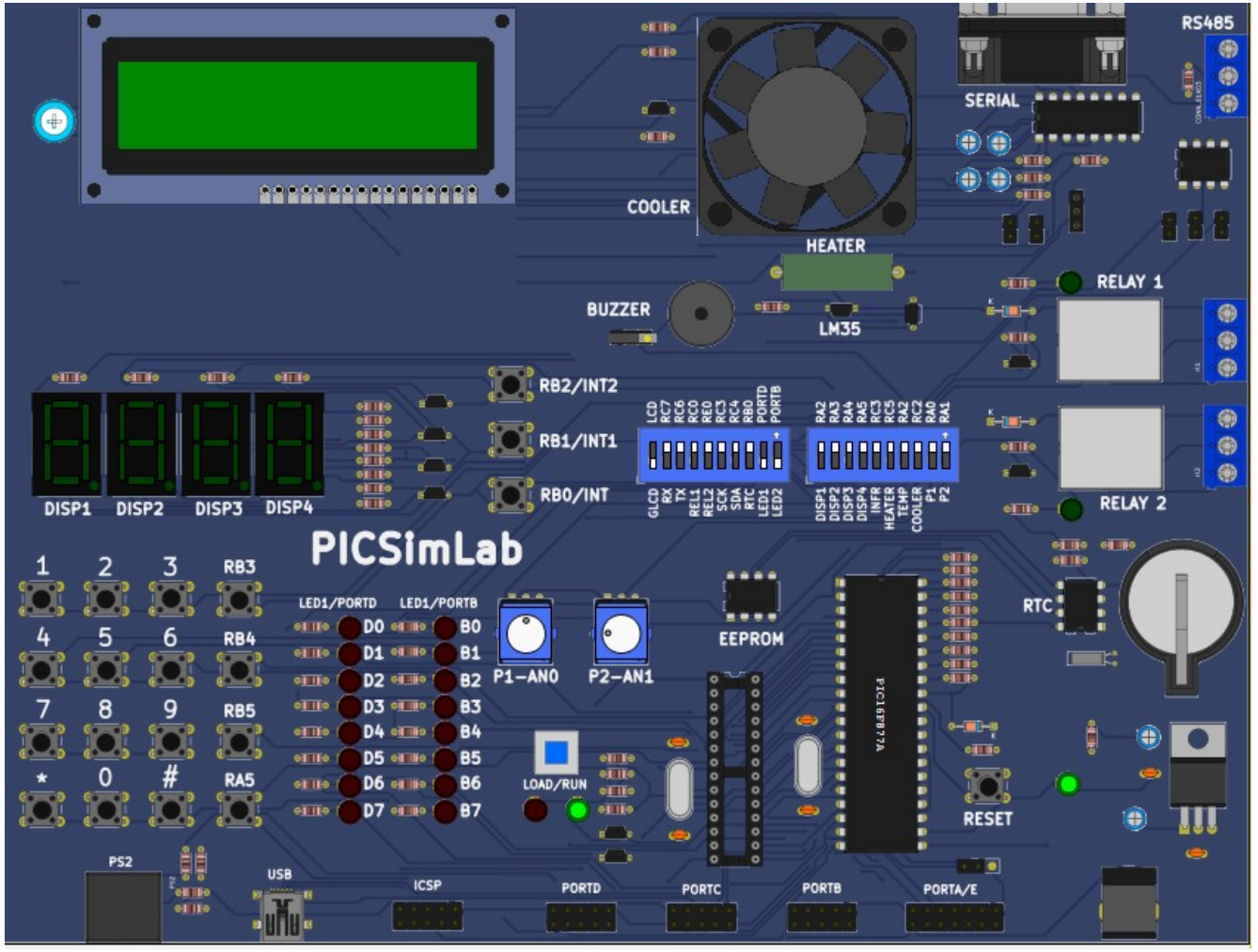
```c
#ifndef MAIN_H
#define MAIN_H

#include <htc.h>

#endif
```

EMERTXE

Lets Roll Out on Interfaces

Microcontrollers

Lets Role Out on Interfaces



- LEDs
- Digital Keypad
- Interrupts
- Timers
- Clock I/O
- SSDs
- CLCD
- Matrix Keypad
- Analog Inputs

EMERTXE

Light Emitting Diodes

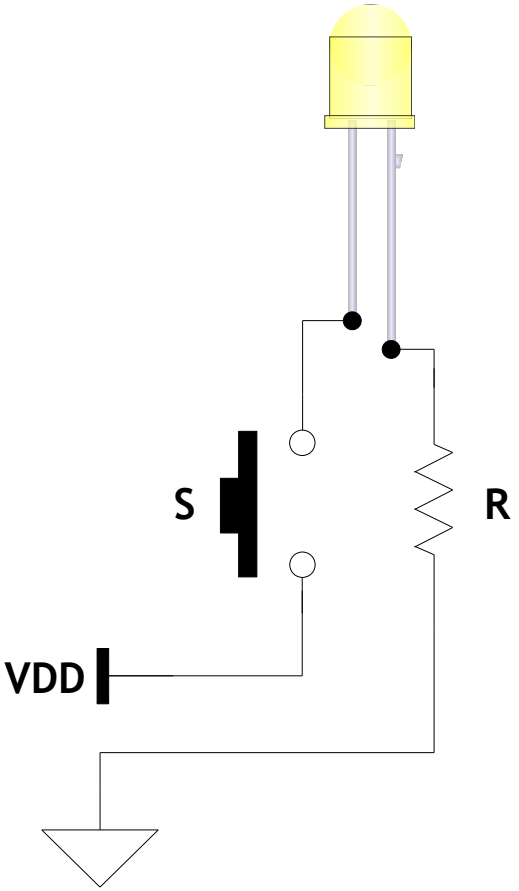
Microcontrollers

Interfaces – LEDs - Introduction

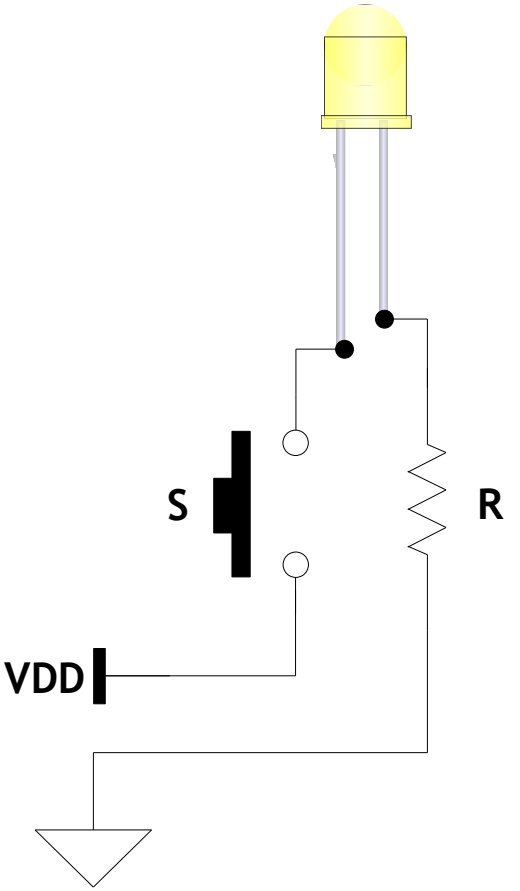
- Simplest device used in most on the embedded applications as feedback
- Works just like diodes
- Low energy consumptions, longer life, smaller size, faster switching make it usable in wide application fields like
  - Home lighting,
  - Remote Controls, Surveillance,
  - Displays and many more!!

Microcontrollers

Interfaces – LEDs – Working Principle



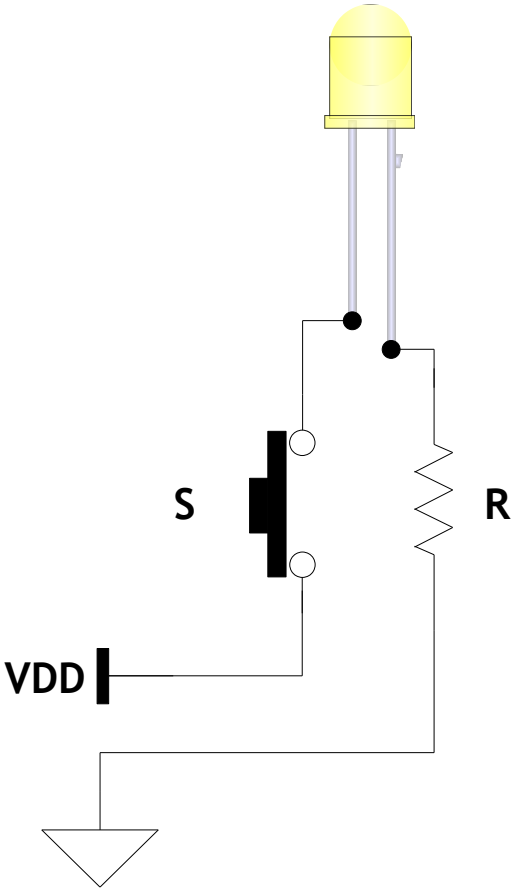
Which side will work?



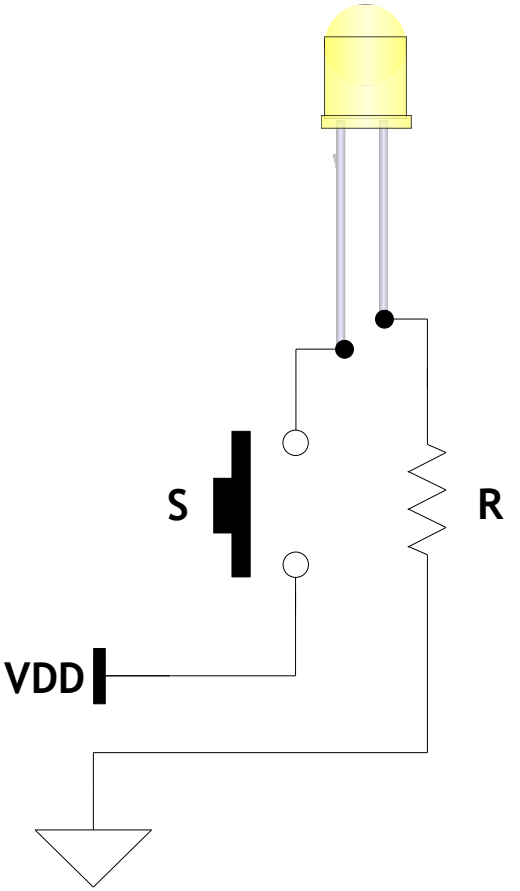
EMERTXE

Microcontrollers

Interfaces – LEDs – Working Principle

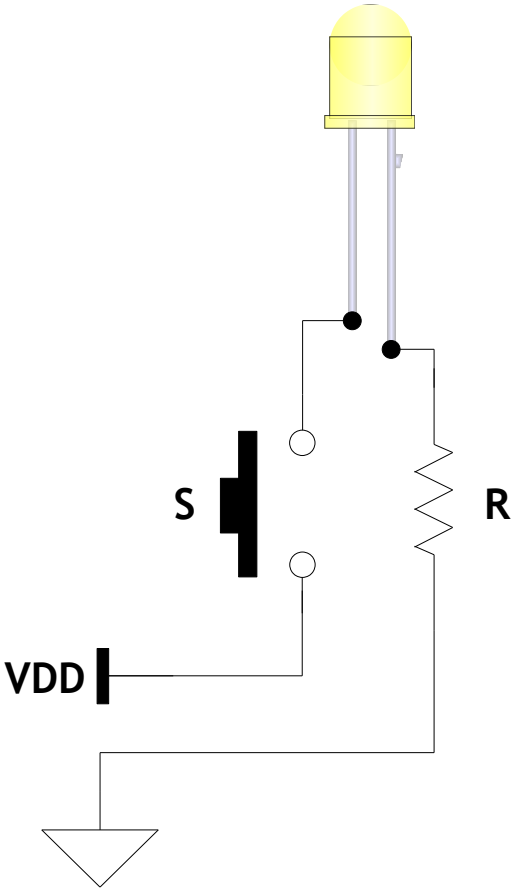


Oops, wrong choice. Can you explain why?

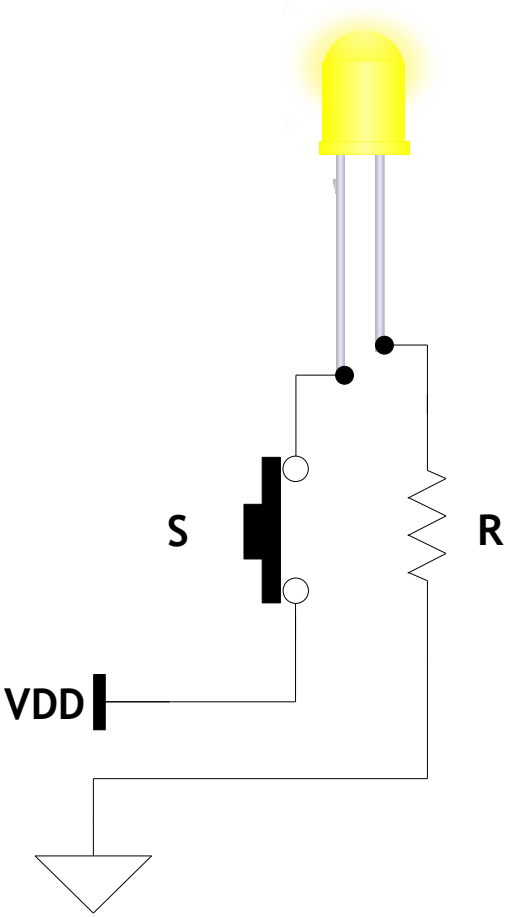


Microcontrollers

Interfaces - LEDs - Working Principle



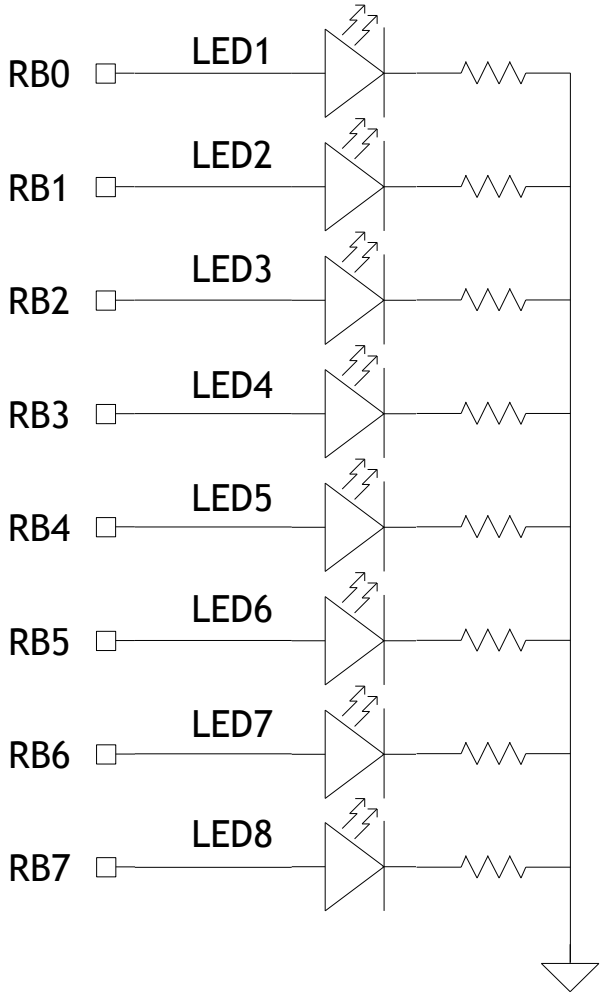
Ooh, looks like you know the funda.

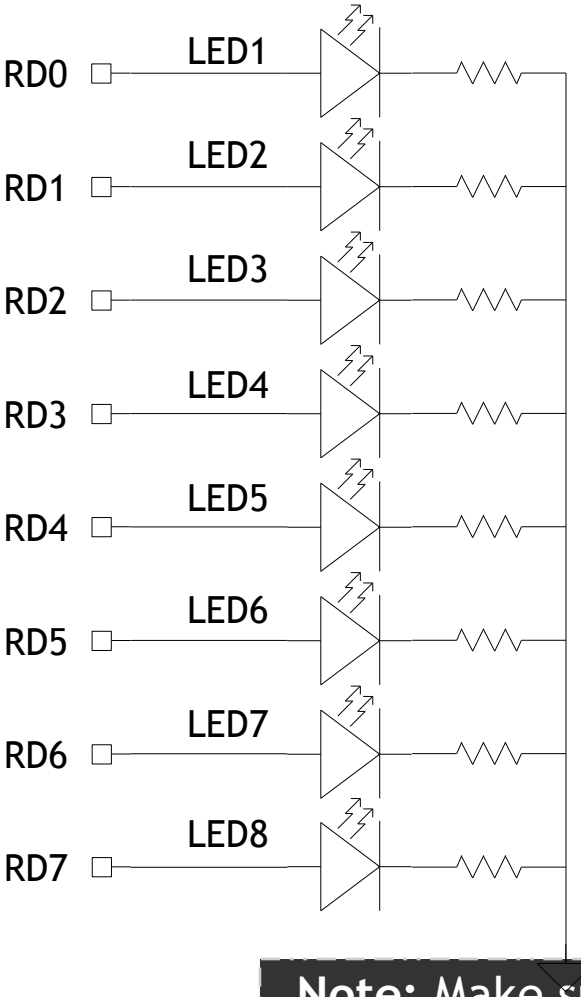


EMERTXE

Microcontrollers

Interfaces – LEDs – Circuit on Board





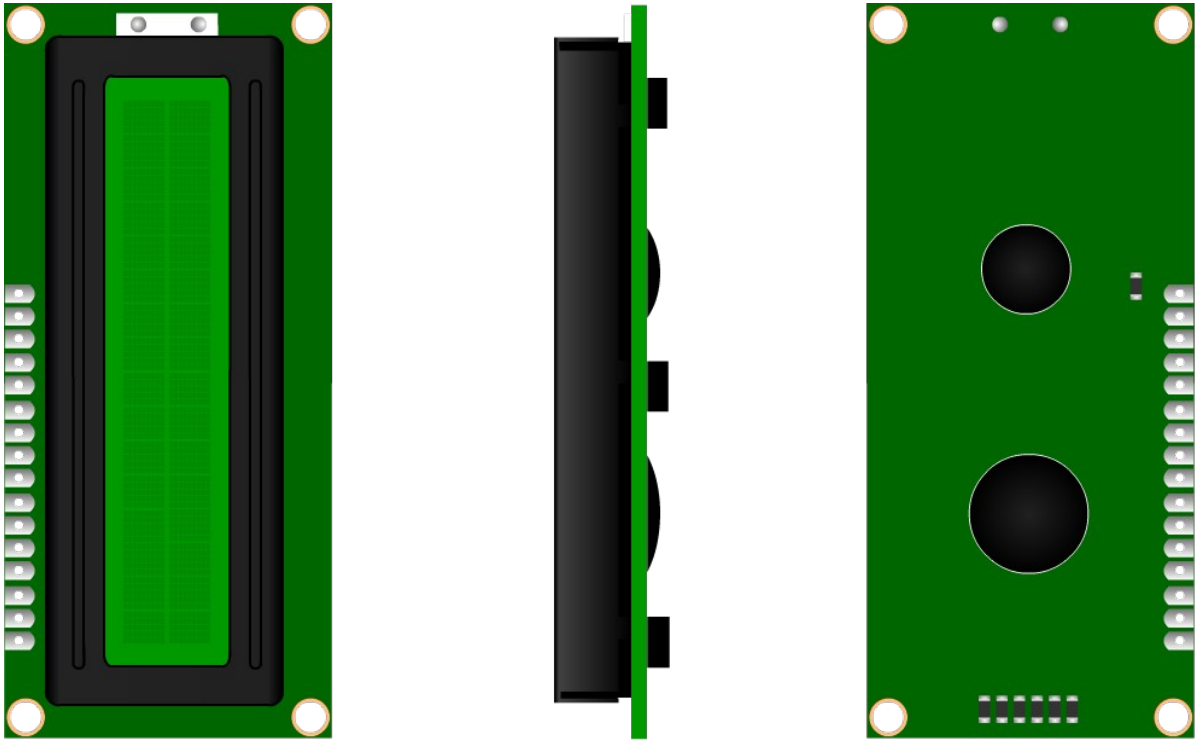
Note: Make sure the DP switch its towards LEDs

Character Liquid Crystal Display

Microcontrollers

Interfaces – CLCD - Introduction

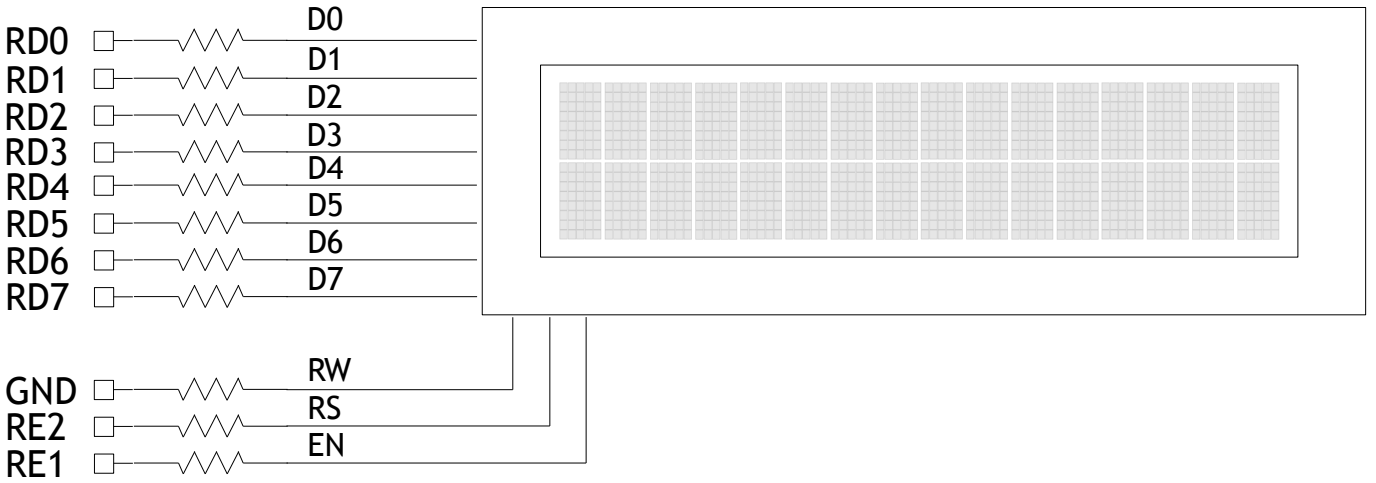
- Most commonly used display ASCII characters
- Some customization in symbols possible
- Communication Modes
  - 8 Bit Mode
  - 4 Bit Mode



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Vdd	Vdd	Vo	RS	R/W	E	D0	D1	D2	D3	D4	D5	D6	D7	A	K

Microcontrollers

Interfaces – CLCD - Circuit on Board

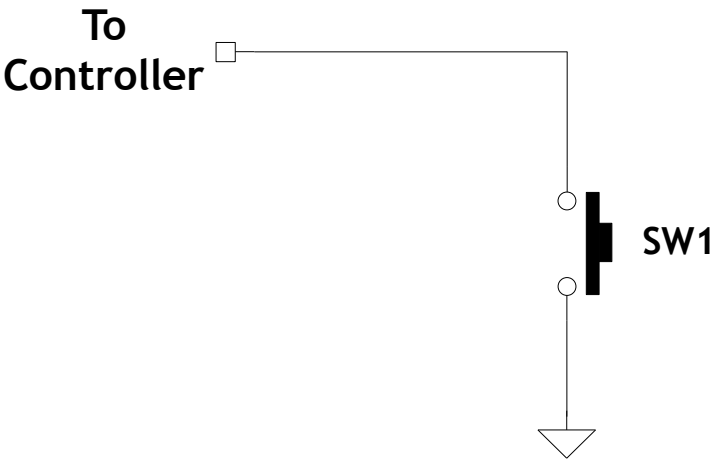


Tactile Switch

Microcontrollers

Interfaces – Tactile Switch

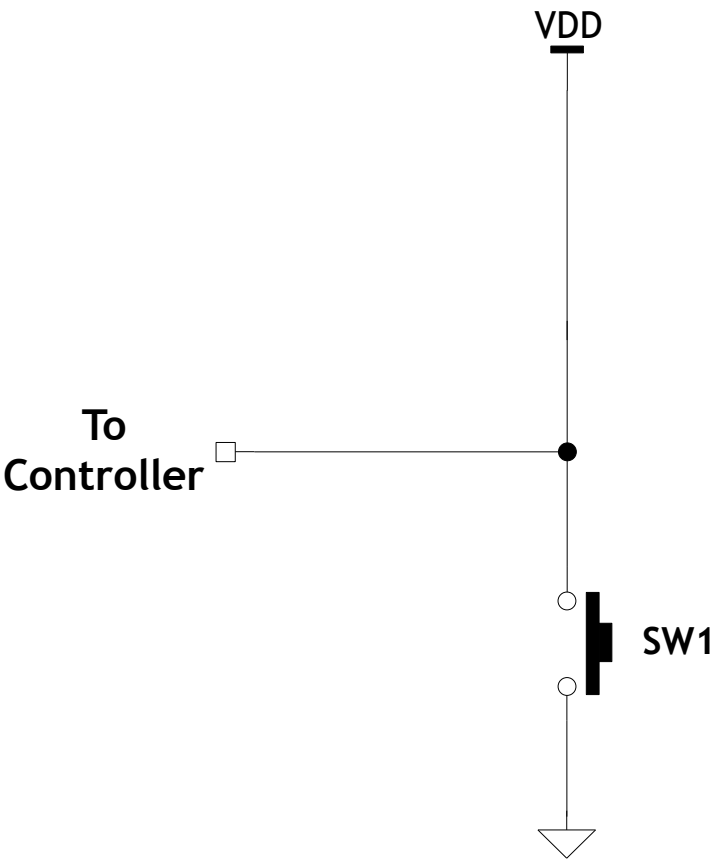
- Considering the below design what will be input to the controller if the switch is pressed?



Microcontrollers

Interfaces – Tactile Switch

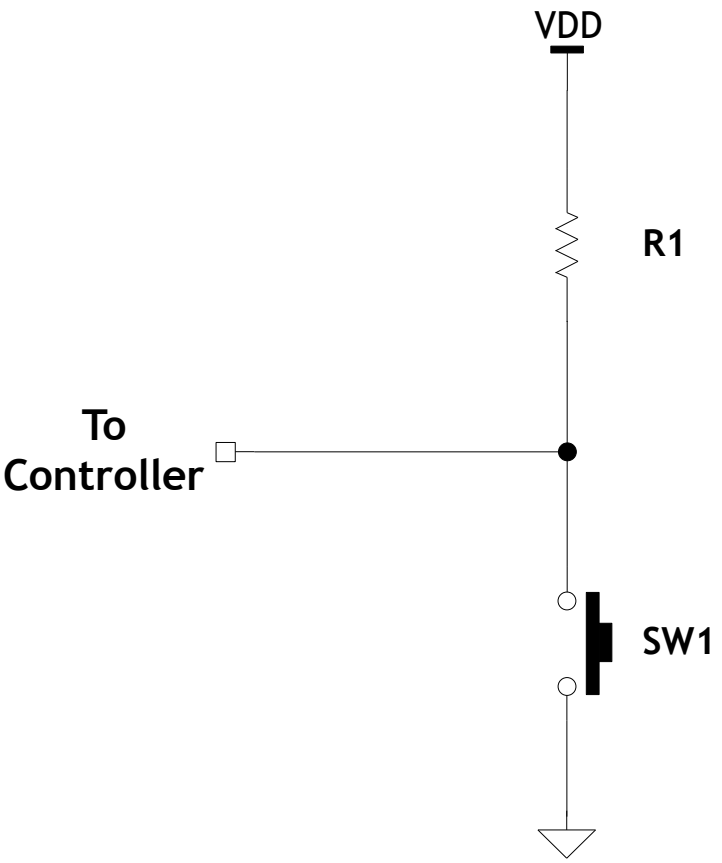
- Will this solve the problem which may arise in the design mentioned in previous slide?



Microcontrollers

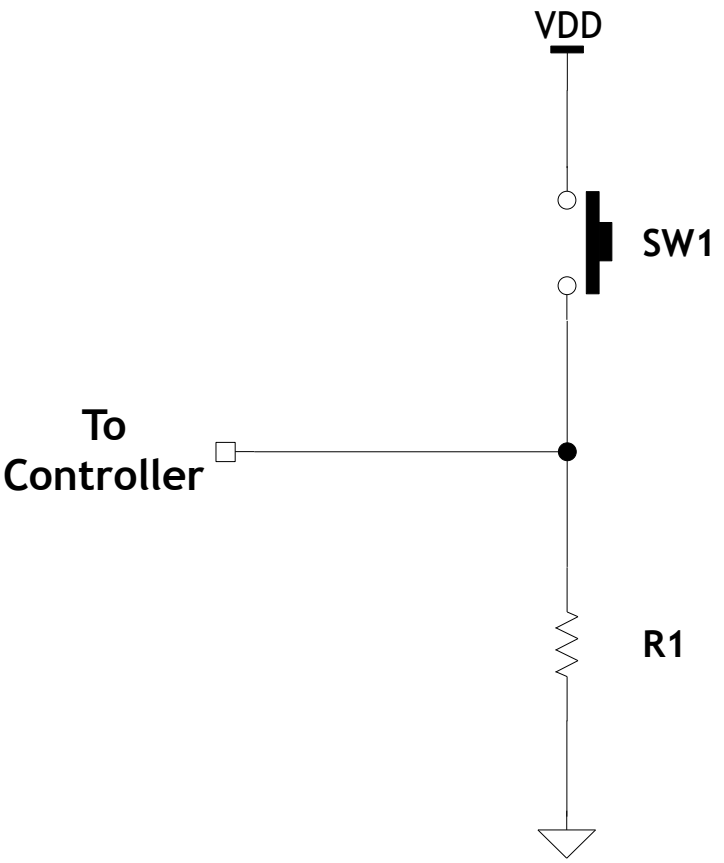
Interfaces – Tactile Switch

- Now will this solve the problem which may arise in the design mentioned in previous slides?



Microcontrollers
Interfaces – Tactile Switch

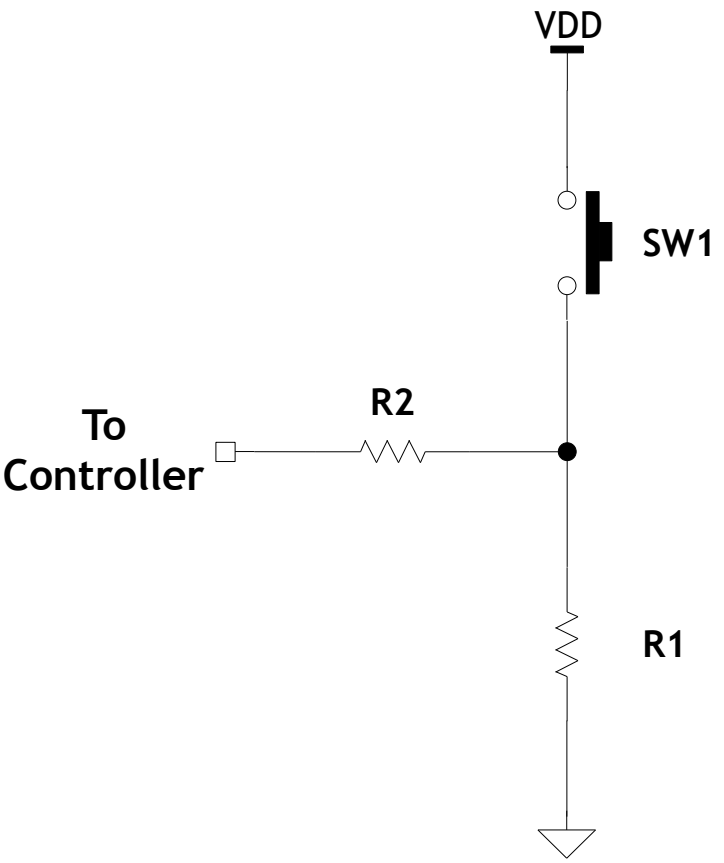
- What would you call the this design?
- Is there any potential problem?



Microcontrollers

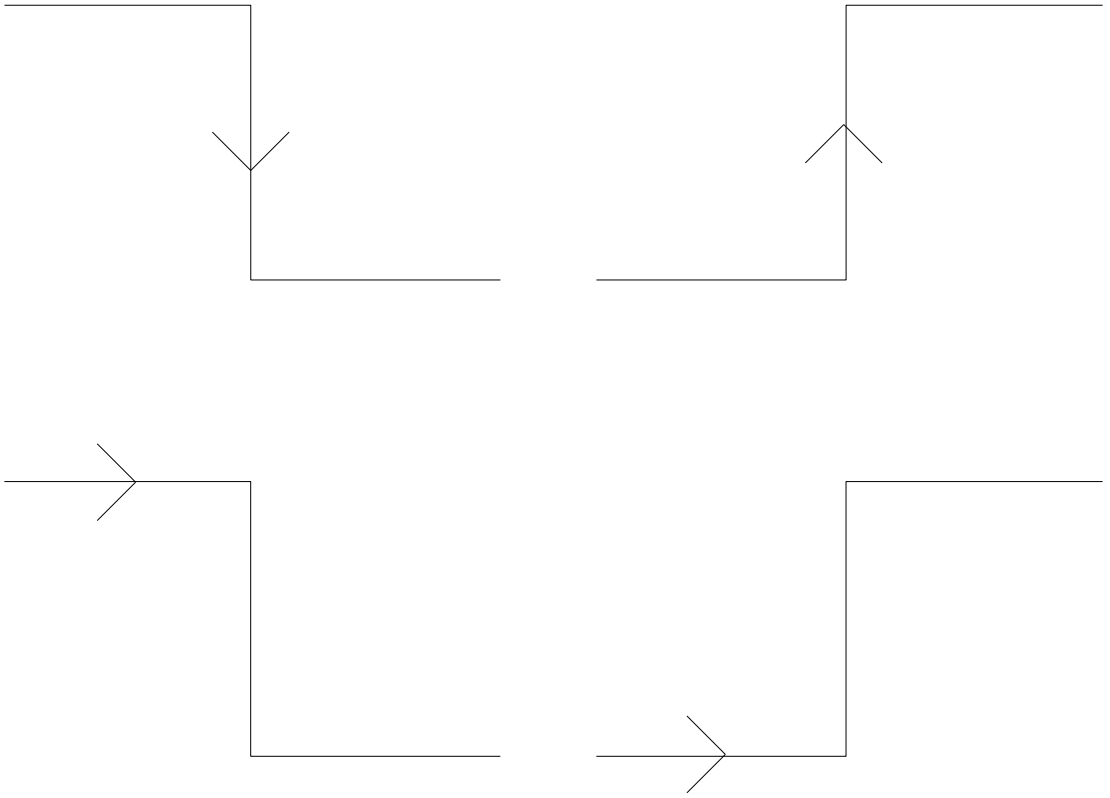
Interfaces – Tactile Switch

- What would you call the this design?
- Is there any potential problem?



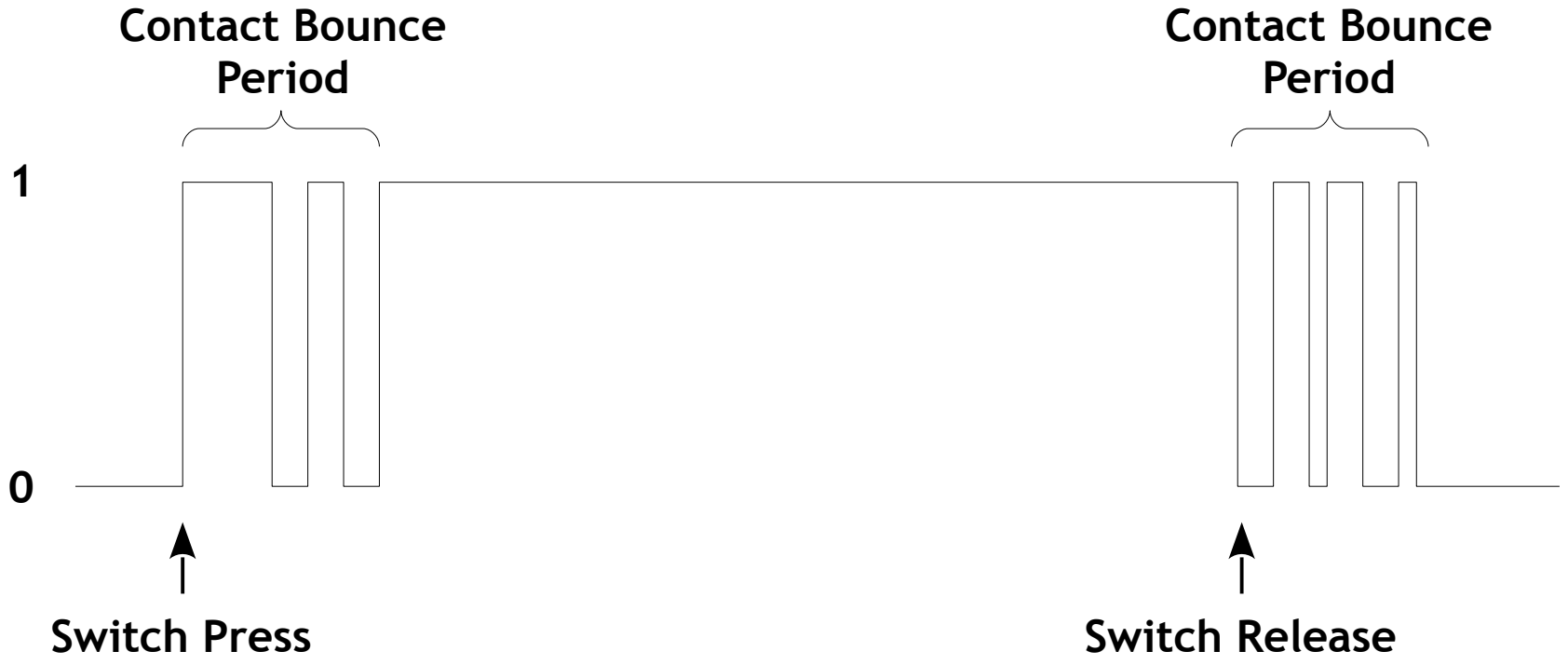
Microcontrollers

Interfaces – Tactile Switch - Triggering Methods



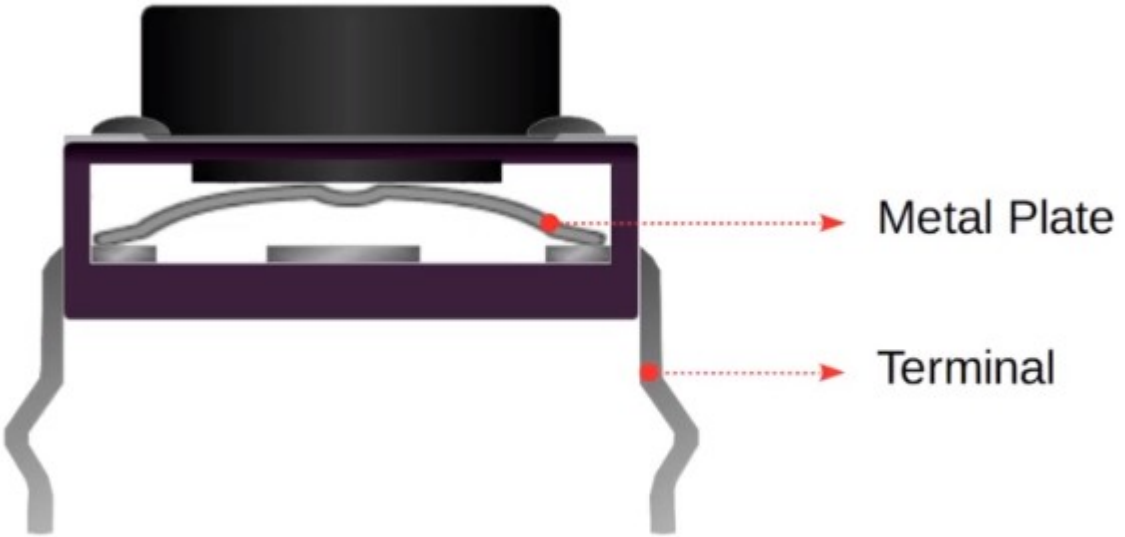
Microcontrollers

Interfaces – Tactile Switch - Bouncing Effects



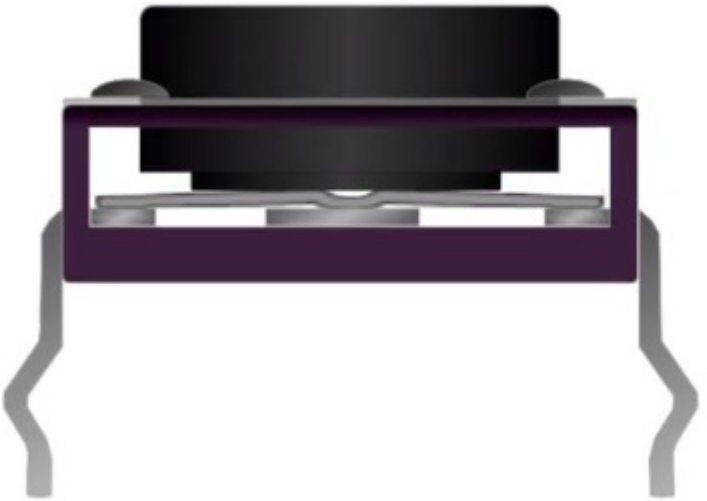
Microcontrollers

Interfaces – Tactile Switch - Bouncing Effects



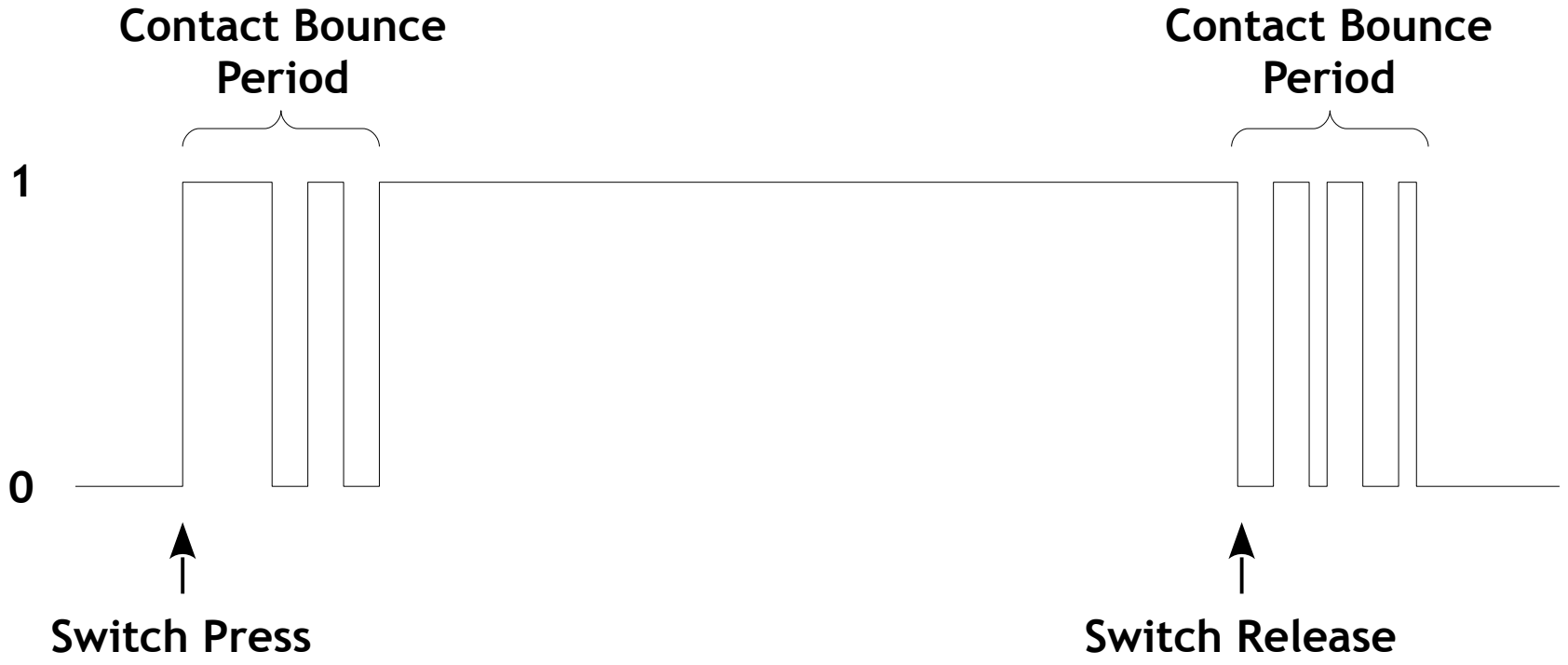
Microcontrollers

Interfaces – Tactile Switch- Bouncing Effects



Microcontrollers

Interfaces – Tactile Switch - Bouncing Effects



Matrix Keypad

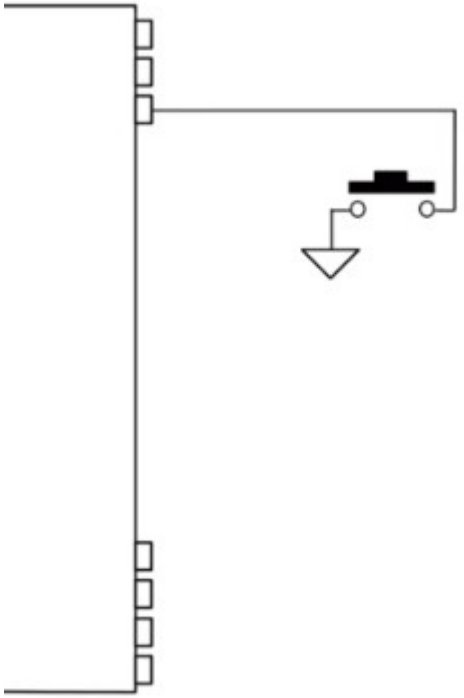
Microcontrollers

Interfaces – Matrix Keypad - Introduction

- Used when the more number of user inputs is required and still want to save the some controller I/O lines
- Uses rows and columns concept
- Most commonly used in Telephonic, Calculators, Digital lockers and many more applications

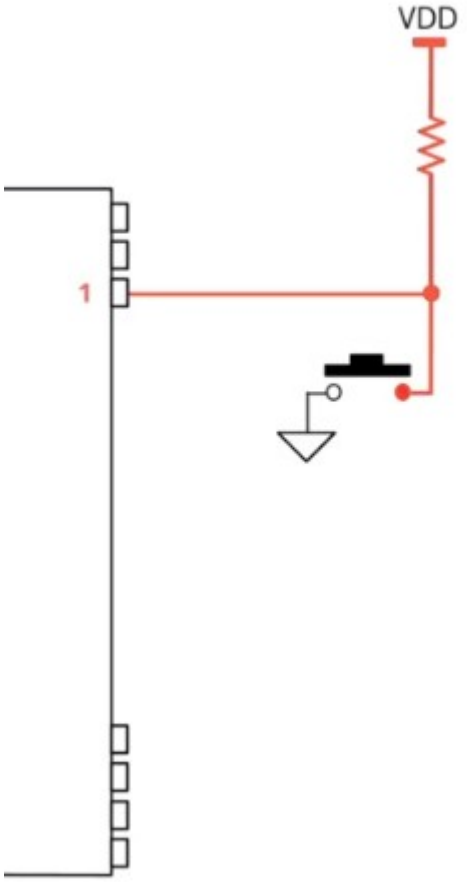
Microcontrollers

Interfaces – Matrix Keypad - Introduction



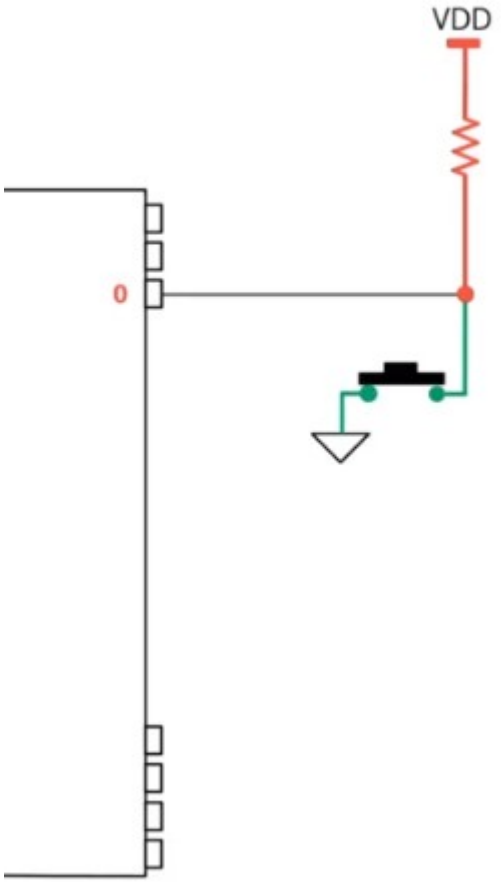
Microcontrollers

Interfaces – Matrix Keypad - Introduction



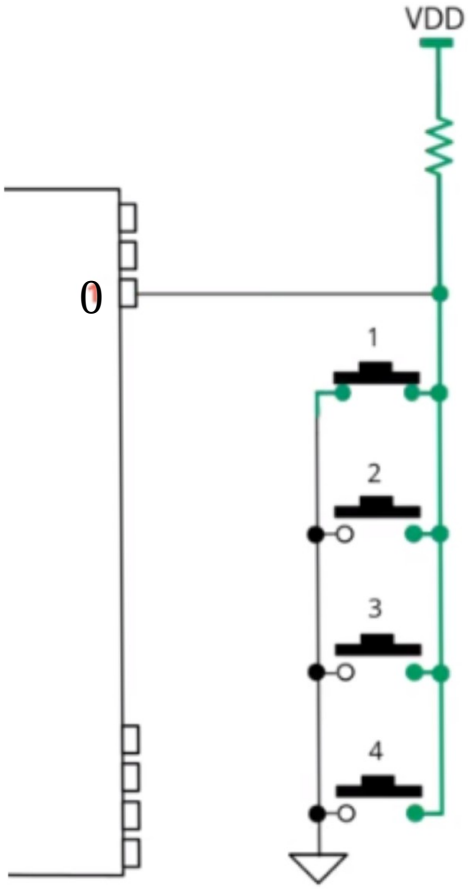
Microcontrollers

Interfaces – Matrix Keypad - Introduction



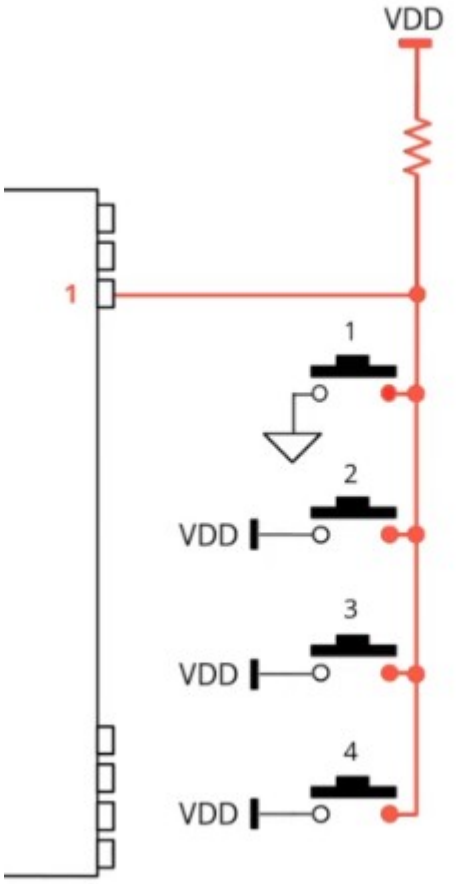
Microcontrollers

Interfaces – Matrix Keypad - Introduction



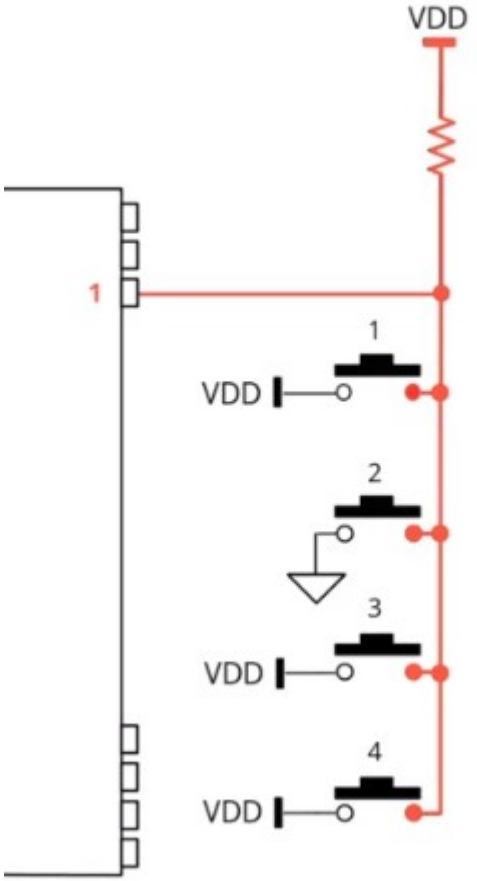
Microcontrollers

Interfaces – Matrix Keypad - Introduction



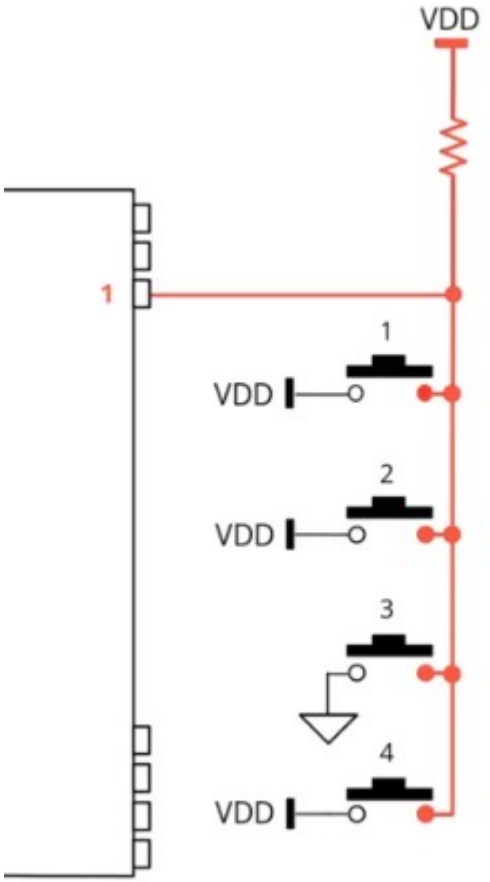
Microcontrollers

Interfaces – Matrix Keypad - Introduction



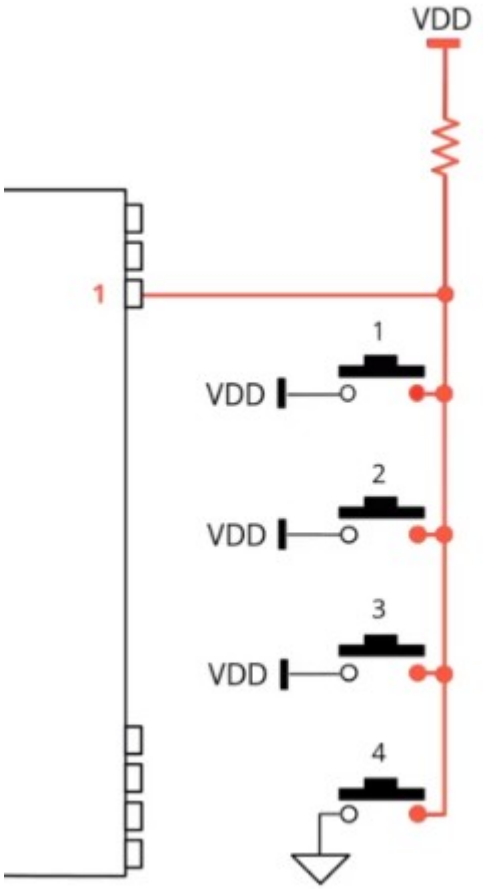
Microcontrollers

Interfaces – Matrix Keypad - Introduction



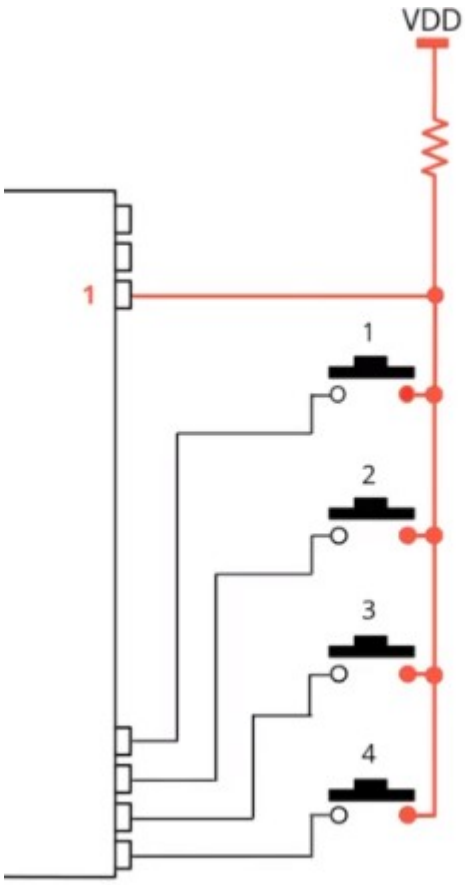
Microcontrollers

Interfaces – Matrix Keypad - Introduction



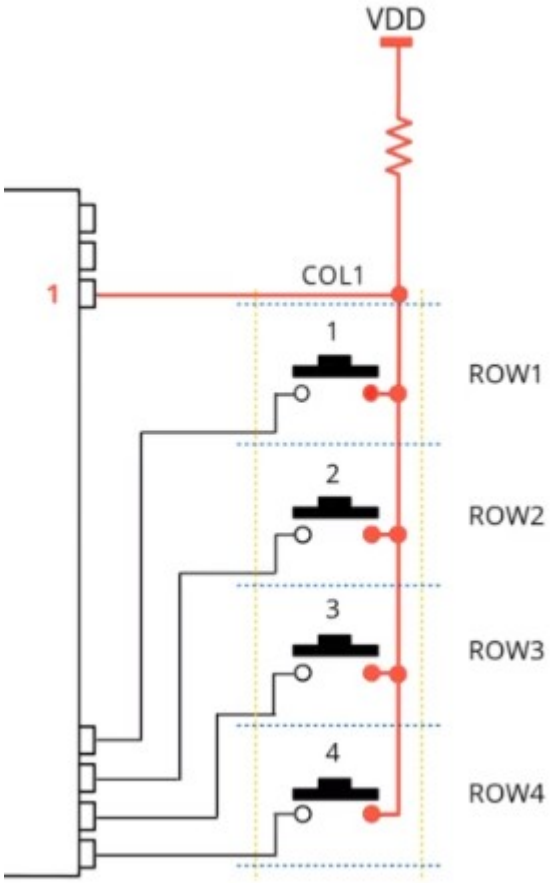
Microcontrollers

Interfaces – Matrix Keypad - Introduction



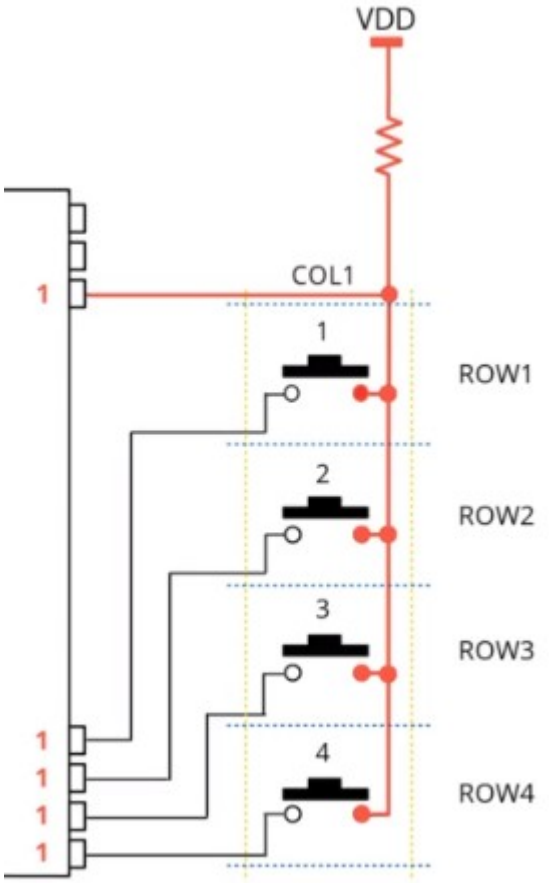
Microcontrollers

Interfaces – Matrix Keypad - Introduction



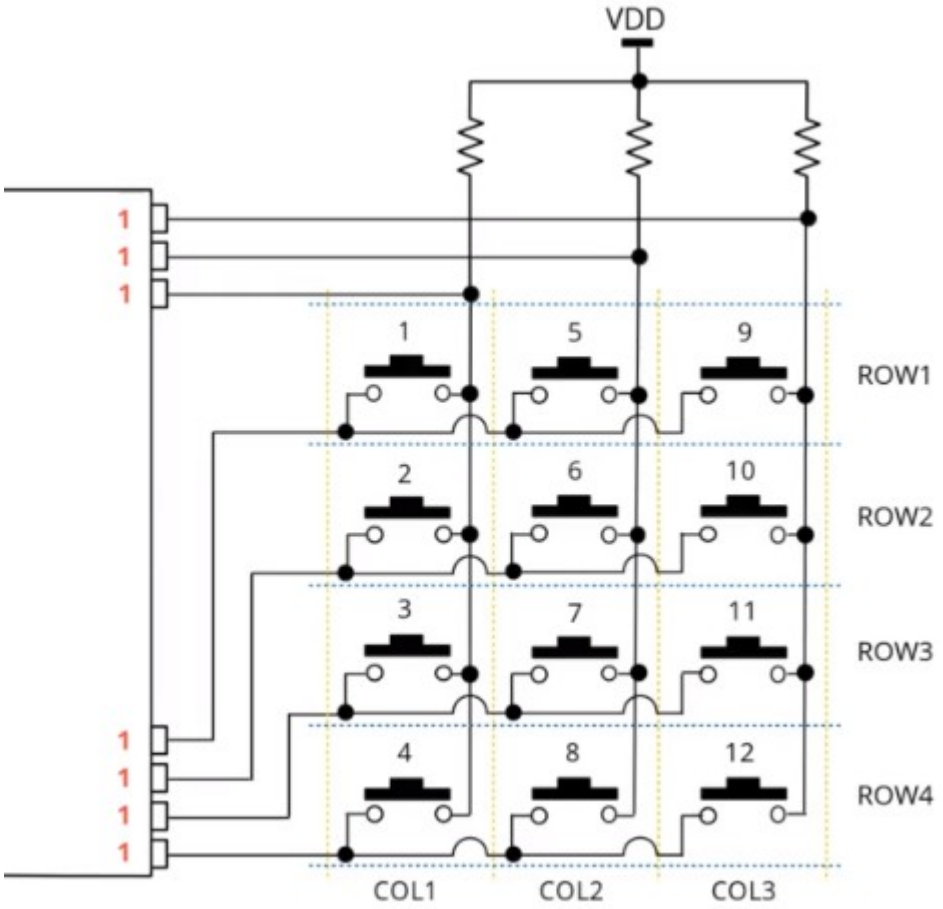
Microcontrollers

Interfaces – Matrix Keypad - Introduction



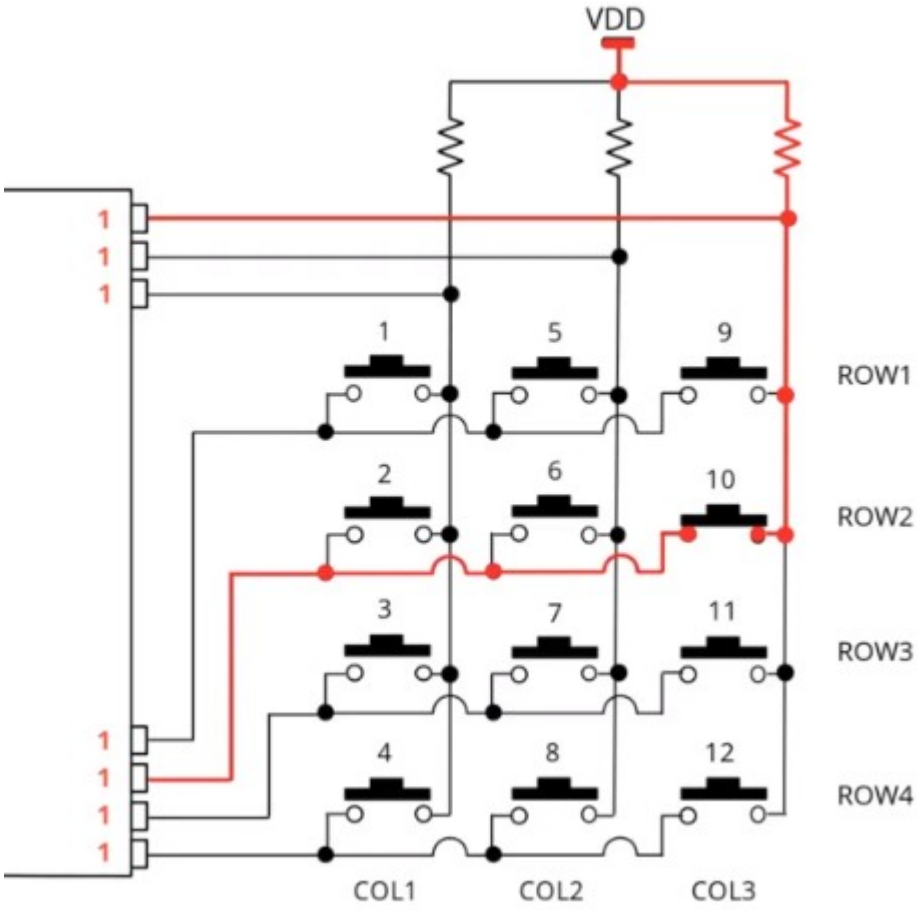
Microcontrollers

Interfaces – Matrix Keypad - Introduction



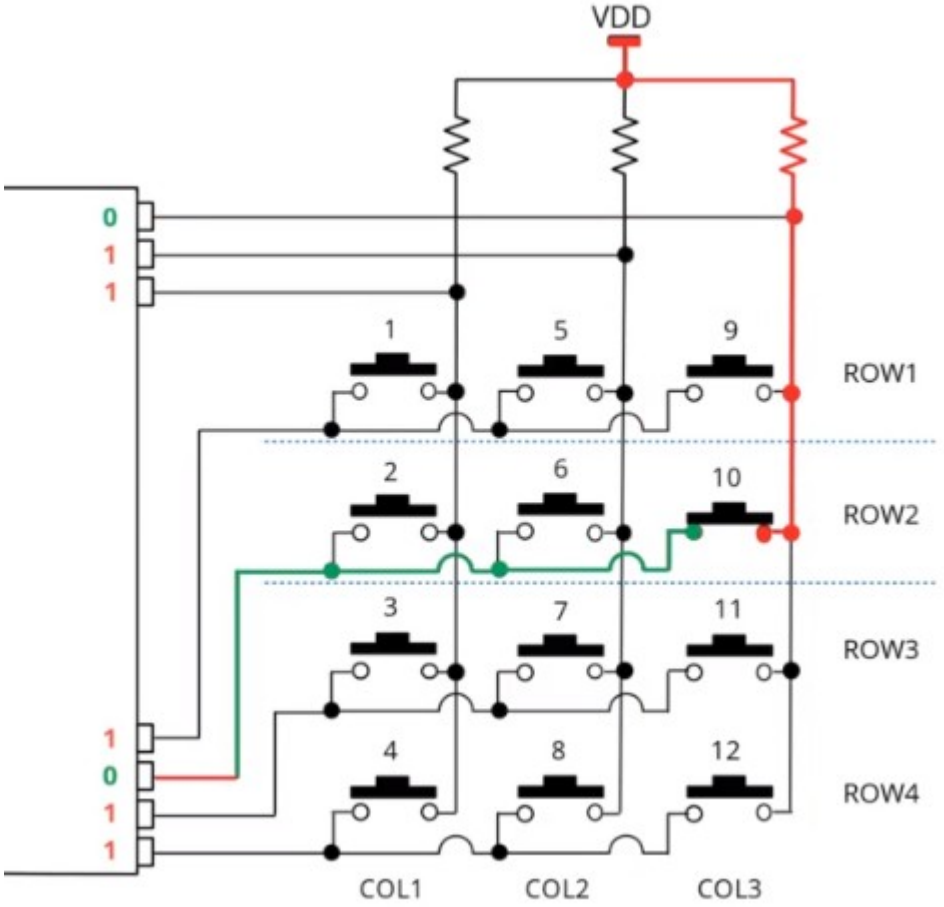
Microcontrollers

Interfaces – Matrix Keypad - Introduction



Microcontrollers

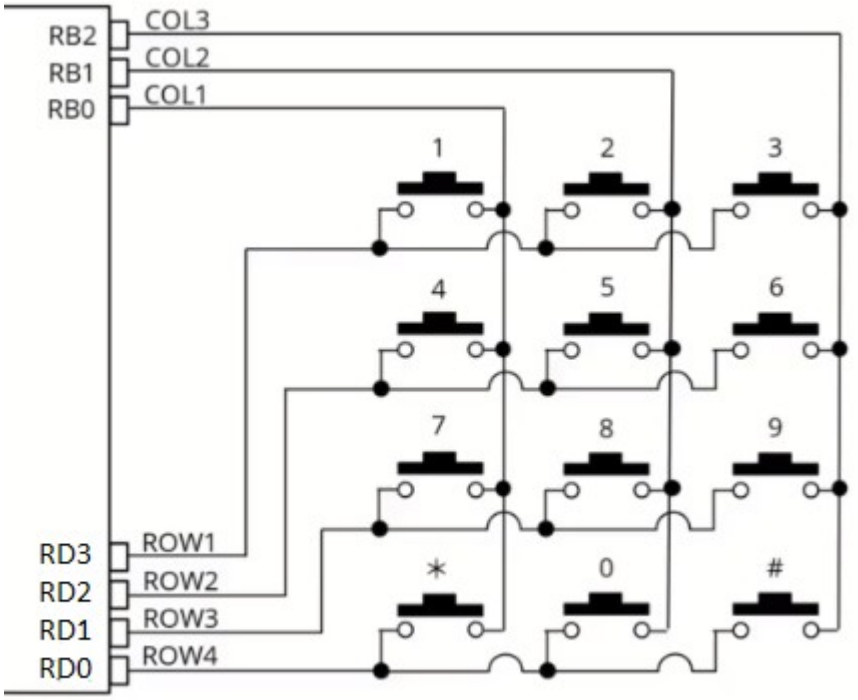
Interfaces – Matrix Keypad - Introduction



EMERTXE

Microcontrollers

Interfaces – Matrix Keypad - Introduction



Interrupts

Microcontrollers
Interrupts

- Basic Concepts
- Interrupt Source
- Interrupt Classification
- Interrupt Handling

EMERTXE

Microcontrollers

Interrupts - Basic Concepts

- An interrupt is a communication process set up in a microprocessor or microcontroller in which:
  - An internal or external device requests the MPU to stop the processing
  - The MPU acknowledges the request
  - Attends to the request
  - Goes back to processing where it was interrupted
- Polling

EMERTXE

Microcontrollers

Interrupts - Interrupt vs Polling

- Loss of Events
- Response
- Power Management

EMERTXE

Microcontrollers

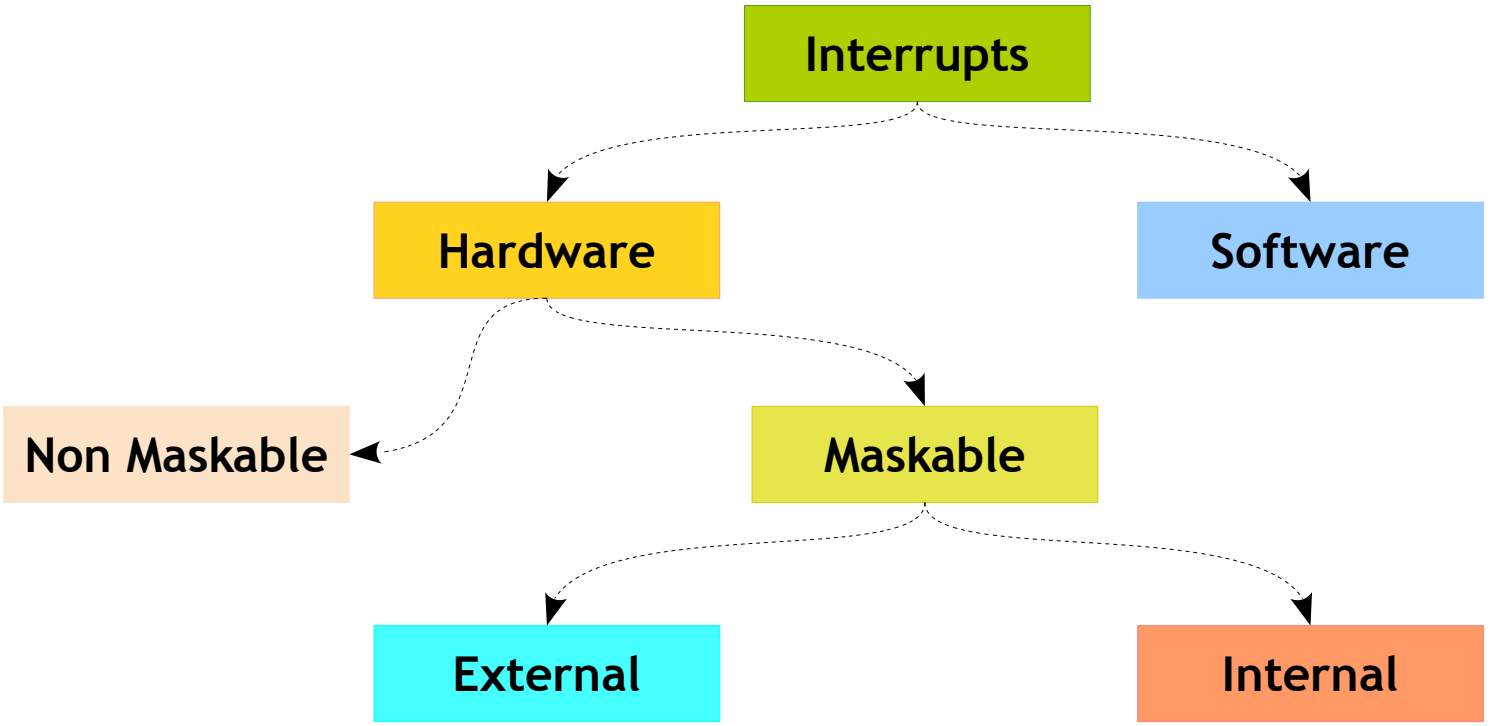
Interrupts - Sources

- External
- Internal

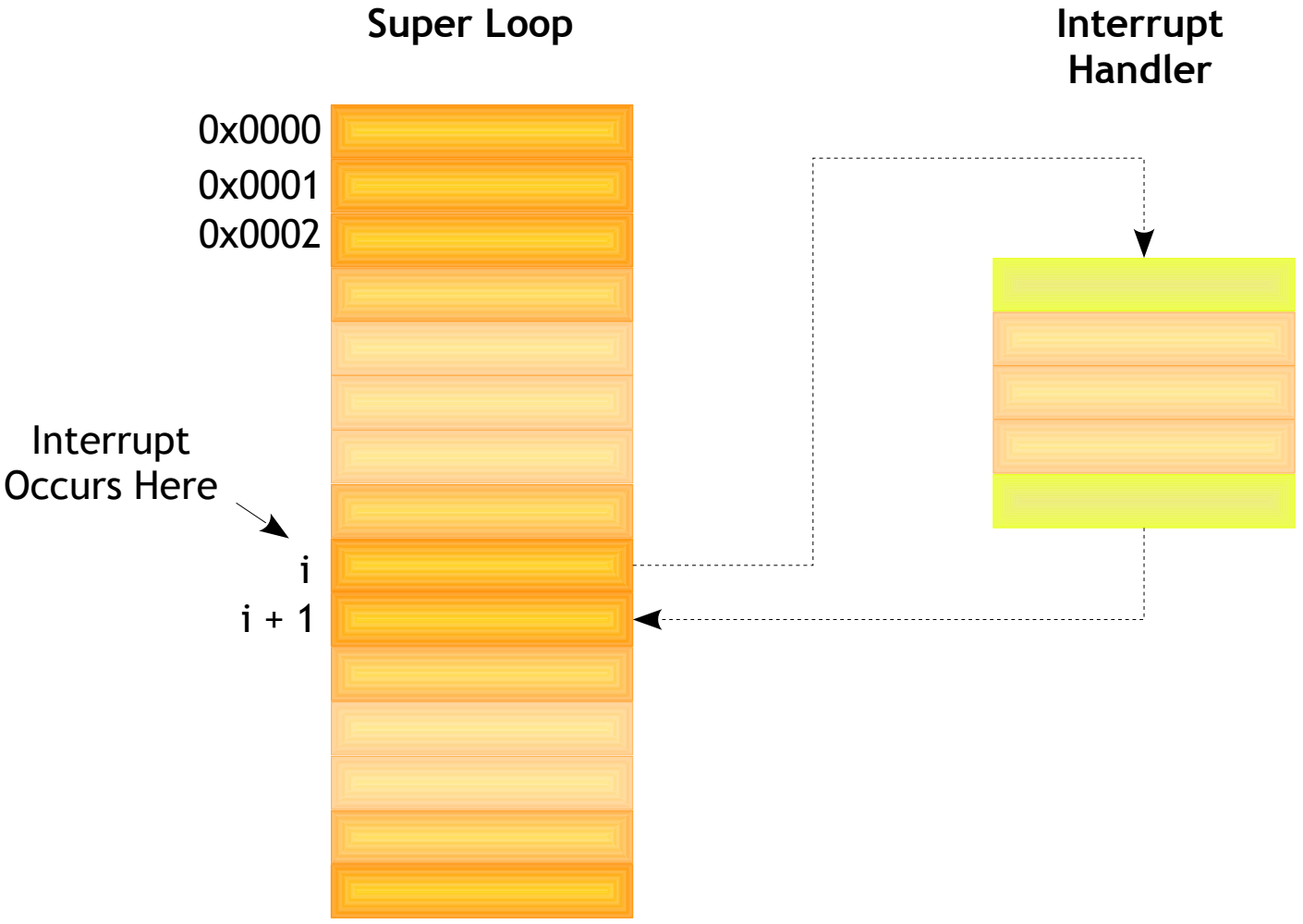
EMERTXE

Microcontrollers

Interrupts - Classification



Microcontrollers
Interrupts - Handling



Microcontrollers

Interrupts - Service Routine (ISR)

- Similar to a subroutine
- Attends to the request of an interrupting source
  - Clears the interrupt flag
  - Should save register contents that may be affected by the code in the ISR
  - Must be terminated with the instruction RETFIE
- When an interrupt occurs, the MPU:
  - Completes the instruction being executed
  - Disables global interrupt enable
  - Places the address from the program counter on the stack
- Return from interrupt

Microcontrollers

Interrupts - Service Routine (ISR)

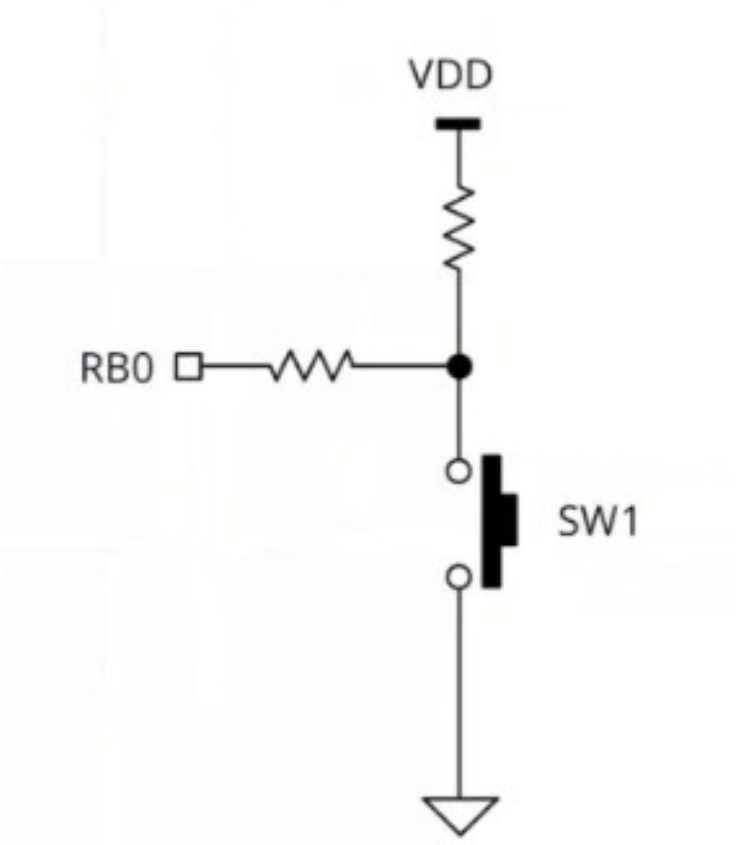
- What / What Not

Microcontrollers
Interrupts - Latency

- Latency is determined by:
  - Instruction time (how long is the longest)
  - How much of the context must be saved
  - How much of the context must be restored
  - The effort to implement priority scheme
  - Time spend executing protected code

Microcontrollers

Interrupts - External Interrupt – Circuit on Board



Microcontrollers

Interrupts - External Interrupt - Example

Example

```
#include <xc8>

static void init_config(void)
{
     init_external_interrupt();
}

void main(void)
{
     unsigned char i;
     init_config();

     while (1)
     {
          while (!glow_led)
          {
               if (SWITCH == 1)
                    glow_led = 1;

               for (i = 10000; i--; );
          }
          if (glow_led)
               LED = 0;
     }
}
```

```
bit glow_led;

void interrupt external_pin(void)
{
     if (INTFLAG)
     {
          glow_led = 1;
          INTFLAG = 0;
     }
}
```

EMERTXE

Timers

Microcontrollers

Timers - Introduction

- Resolution → Register Width
- Tick → Up Count or Down Count
- Quantum → System Clock settings
- Scaling → Pre or Post
- Modes
    - Counter
    - PWM or Pulse Generator
    - PW or PP Measurement etc.,
- Examples

Microcontrollers

Timers - Example

- Requirement – 5 pulses of 8 µsecs
- Resolution – 8 Bit
- Quantum – 1 µsecs
- General

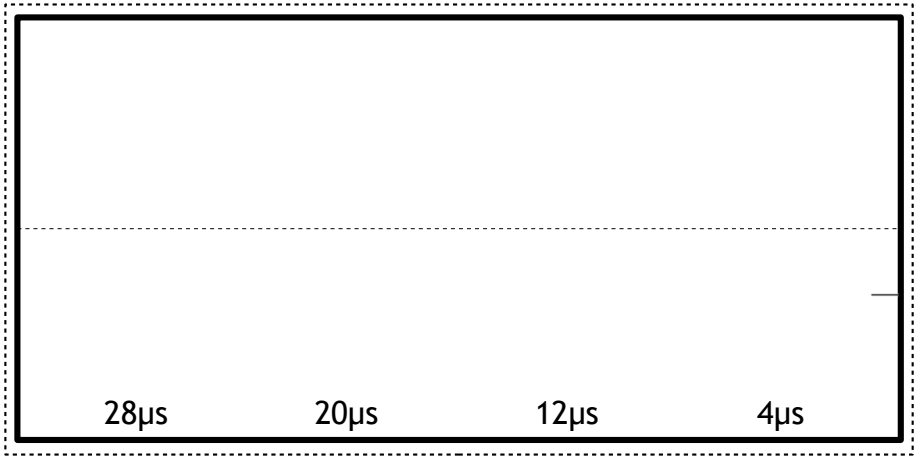
EMERTXE

Microcontrollers

Timers - Example

Timer Register 252

Overflows 0



Thank You