
# Embedded Systems Interfacing

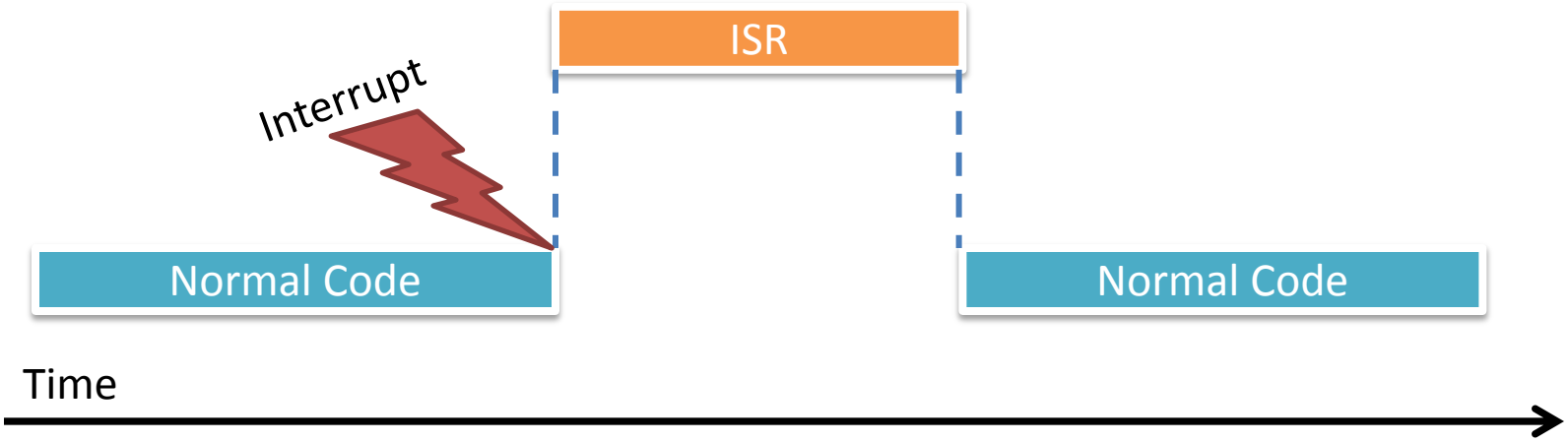
## Lecture Six

## Interrupts

# Introduction to Interrupts

***Definition:***

Interrupt is *an event* that disturbs the normal execution sequence of the code and makes the processor *jump* to another piece of code to execute called *Interrupt Service Routine (ISR)*. After the processor finish executing of the ISR, it returns back to the interrupted code to continue from the interrupted instruction.



# Examples of interrupts

***Analogy:***

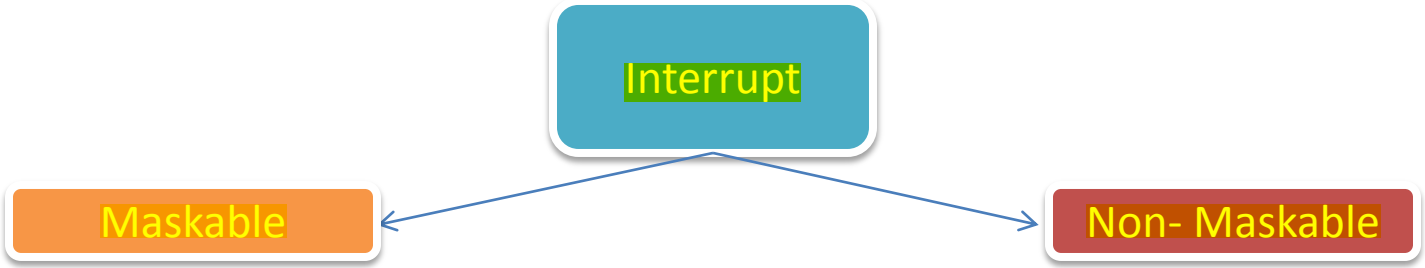
Imagine that you are at your home watching your favorite movie and at certain time your doorbell rang ! ***This is the interrupt.***

You will open your door to see who is in then you will return back to continue watching your movie ! ***This is the interrupt service routine***.


**In the microcontroller**, there are ***many sources*** of interrupts. For example:

1- When the Analog to Digital Converter ***(ADC)*** finishes its conversion.
2- When the Serial Peripheral Interface ***(SPI)*** sends or receives a message.
3- When the ***Timer*** finishes counting a preconfigured period.

# Interrupt Masking




Could be enabled or disabled

When it happens, it must be served. It can't be disabled.

**Each interrupt** in the microcontroller has a ***specific indicator*** bit called ***peripheral interrupt flag*** ***(PIF)***. This bit is set to 1 when the interrupt happens **whatever** the interrupt was enabled or disabled.

**In Maskable interrupts**, another bit exist called ***peripheral interrupt Enable*** ***(PIE)***. When Maskable happens, its PIF is set to 1. If its PIE was set to 1, then the interrupt would be served. If not, the processor would continue its normal code execution.

**In Non-Maskable interrupts**, there is no PIE. Whenever the interrupt happens, it is being served.

# GIE Vs PIE

**As mentioned before**, each interrupt source has a specific enable bit called ***PIE***. But what if we need to ***disable all interrupts*** for a while and re-enable them. Shall we pass by each interrupt PIE and set it 0 to disable then pass by each interrupt PIE and set it to 1 to re-enable ?!. It would take a lot of execution time.

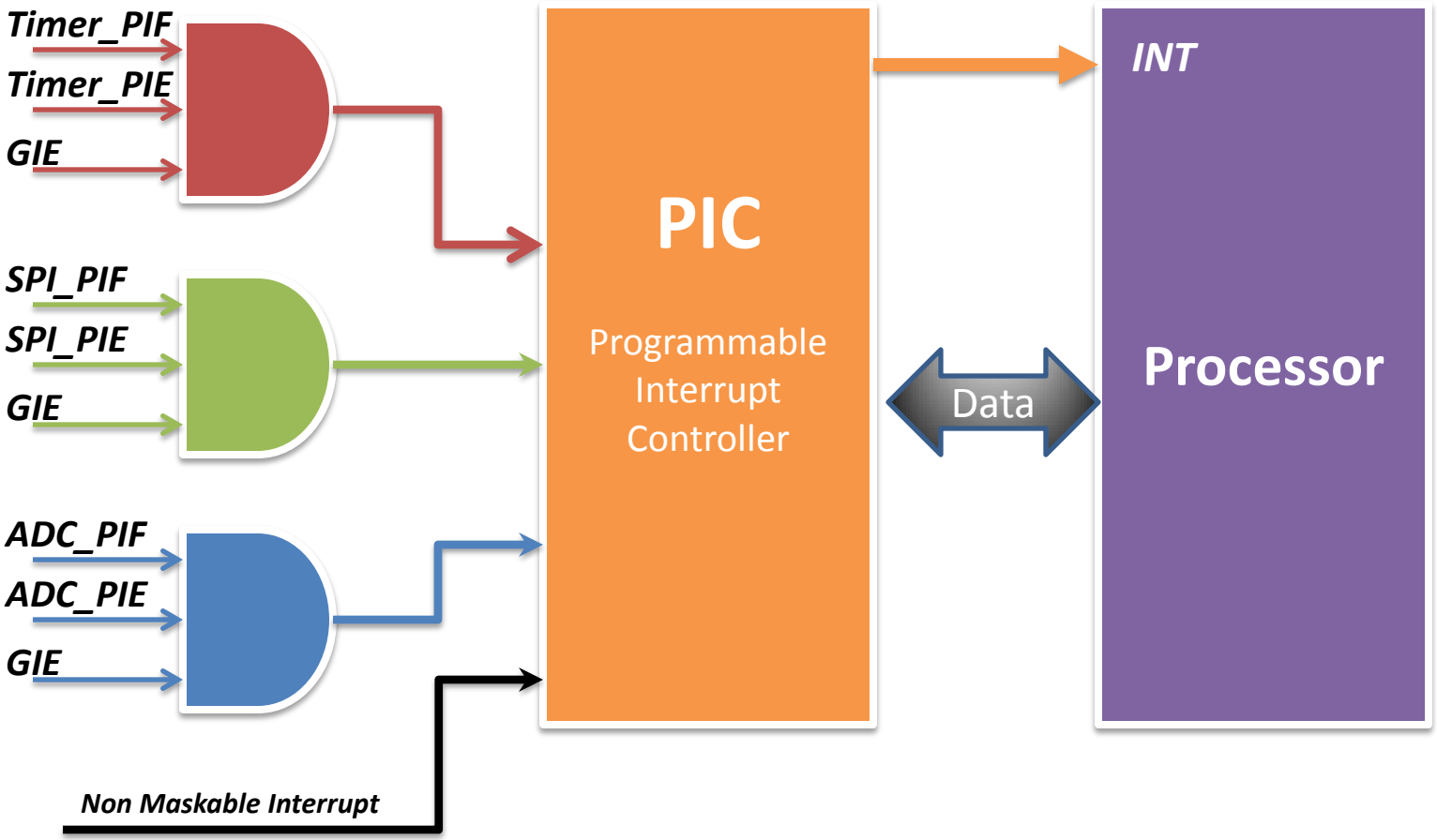
***Instead,*** another bit called ***Global Interrupt Enable*** ***(GIE).*** When this bit is 0, All maskable interrupt are disabled **whatever** the PIE was 1 or 0.

But when this bit is set to 1, it ***allows*** the maskable interrupt to happen if its PIE is set to 1 too.

**Remember**

The Non- Maskable interrupts have no PIE and the GIE doesn't affect them. These interrupts when happen, they must be served !

# Programmable Interrupt Controller



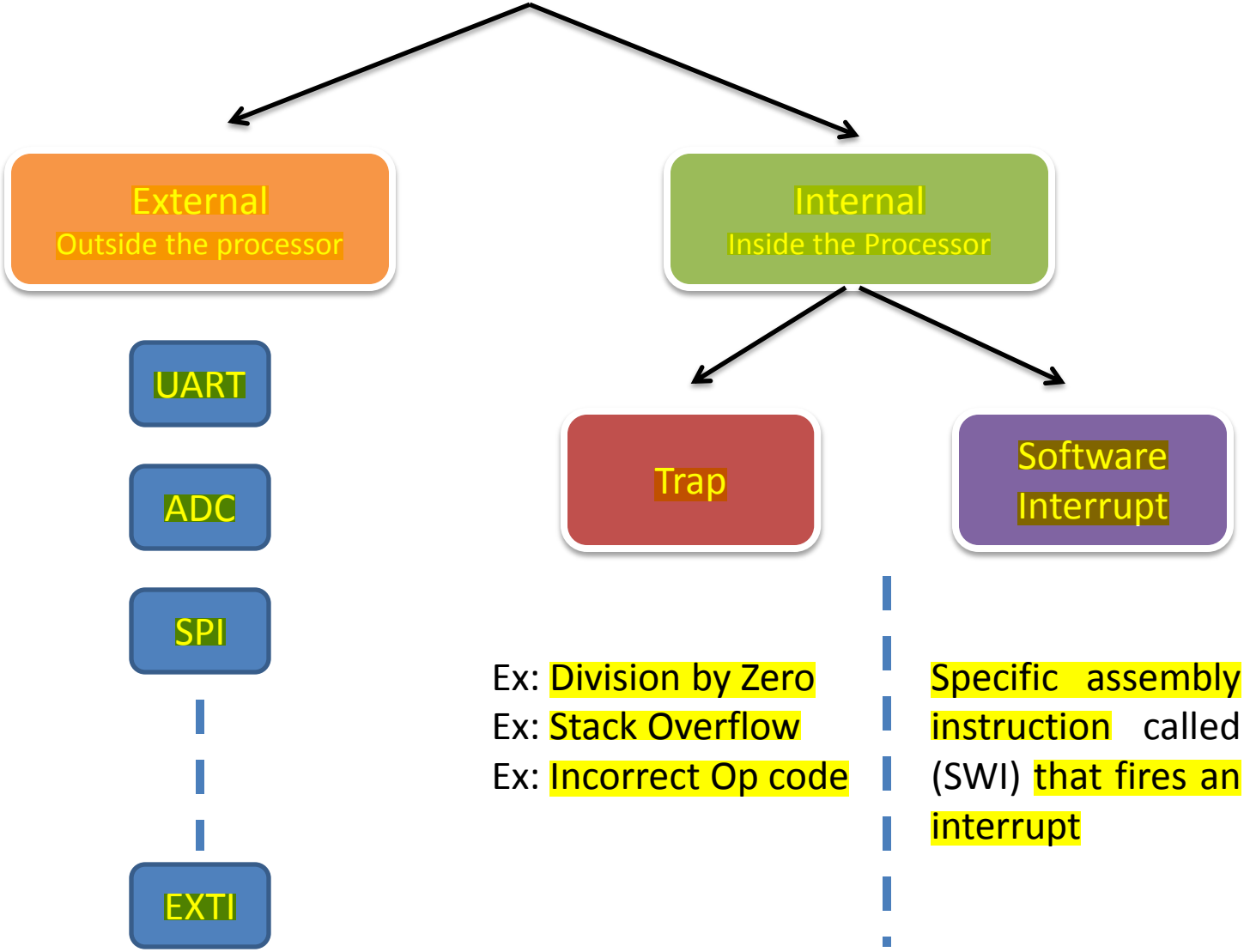
# Programmable Interrupt Controller

***The Programmable Interrupt Controller*** ***(PIC)*** is a hardware element that handles the interrupt serving and priorities. All interrupts are connected to the PIC, when any interrupt happens, the PIC receives the interrupt request and ***generate a signal*** to the processor on its ***INT*** pin. Then the PIC tells the processor the ***ID of the interrupt*** happened through a special data bus.

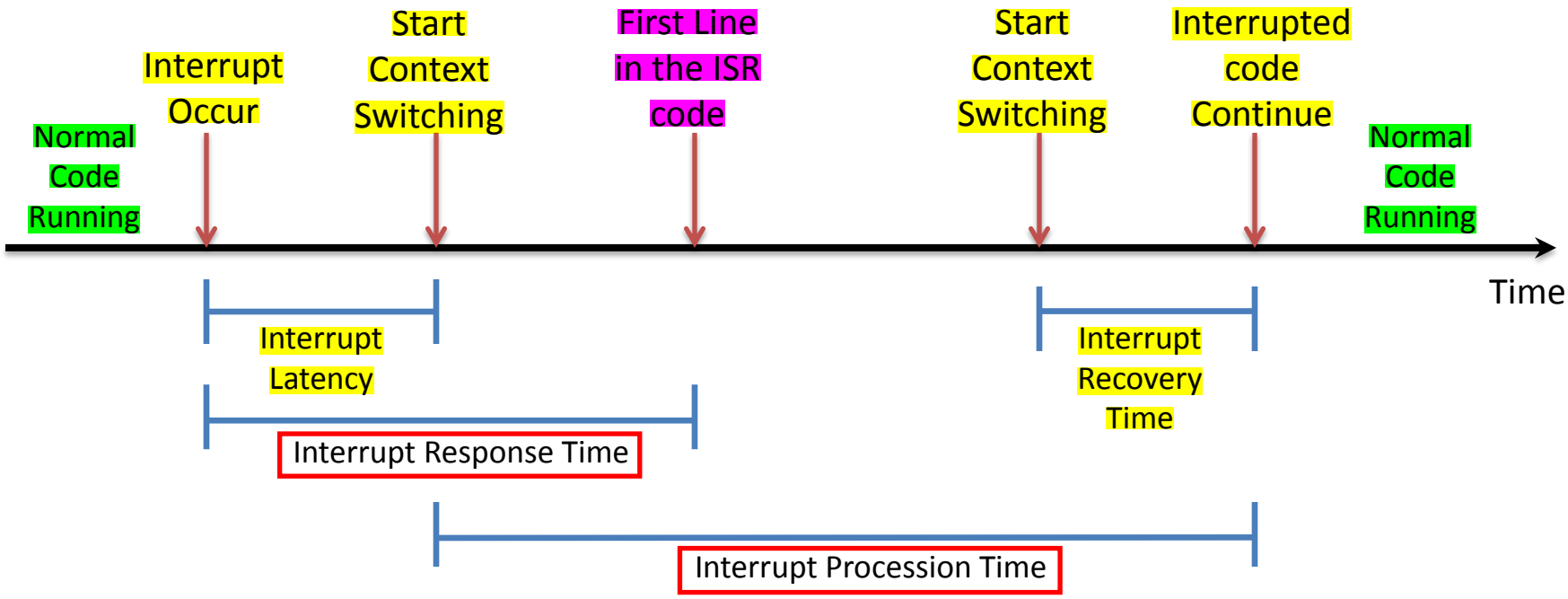
Inside the processor there is a piece of memory called ***vector table***. Every location of the this memory ***(called vector)*** corresponding to a certain interrupt. When the process receives an interrupt request from the PIC and gets its ID, it jumps to the ***corresponding location*** in the vector table to find the address of its ISR.

The PIC also handles the ***interrupt priorities***, if two interrupts happened at the same time, the upper interrupt would have higher priority. The PIC tells the processor about the upper interrupt first, then tells the processor about the lower one.

# Interrupt Types

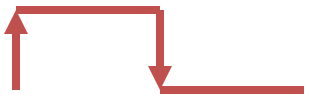


# Interrupt Performance Parameters



# External Interrupt Peripheral
# EXTI

The External interrupt pins are external pins on the microcontroller that are sensitive to digital events. Any digital signal has four main events, Falling Edge, Rising Edge, Low Level and High Level



In our microcontroller ***Atmega32*** there are three external interrupts. These interrupts have four triggering options:

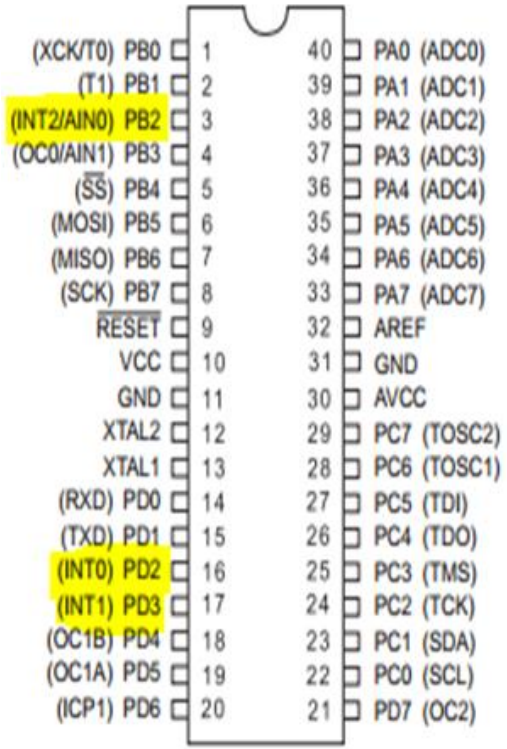
1- Falling Edge

2- Rising Edge

3- Any Digital Change (Rising Edge or Falling Edge)

*This type of interrupt called Interrupt On Change (IOC)*

4- Low Level



# Register Description

## *Microcontroller Control Register* MCUCR

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| SE | SM2 | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00 | MCUCR |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| $ISC_x1$ | $ISC_x0$ | Description |
|---|---|---|
| 0 | 0 | The low level of $INT_x$ generates an interrupt request. |
| 0 | 1 | Any logical change on $INT_x$ generates an interrupt request. |
| 1 | 0 | The falling edge of $INT_x$ generates an interrupt request. |
| 1 | 1 | The rising edge of $INT_x$ generates an interrupt request. |

# Register Description

## *General Interrupt Control Register* GICR

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| INT1 | INT0 | INT2 | – | – | – | IVSEL | IVCE | GICR |
| R/W | R/W | R/W | R | R | R | R/W | R/W | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

When the INTx bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled.

# Register Description

## *General Interrupt Flag Register* GIFR

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| INTF1 | INTF0 | INTF2 | – | – | – | – | – | GIFR |
| R/W | R/W | R/W | R | R | R | R | R | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The bit INTFx is set to one when the corresponding interrupt event happens. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it.

# Register Description

## *Status Register* SREG

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| I | T | H | S | V | N | Z | C | SREG |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

The I bit is the global interrupt enable bit. When it is set to 1, the global interrupt is enabled. When it is set to 0 the global interrupt is disabled.

# Polling on a flag

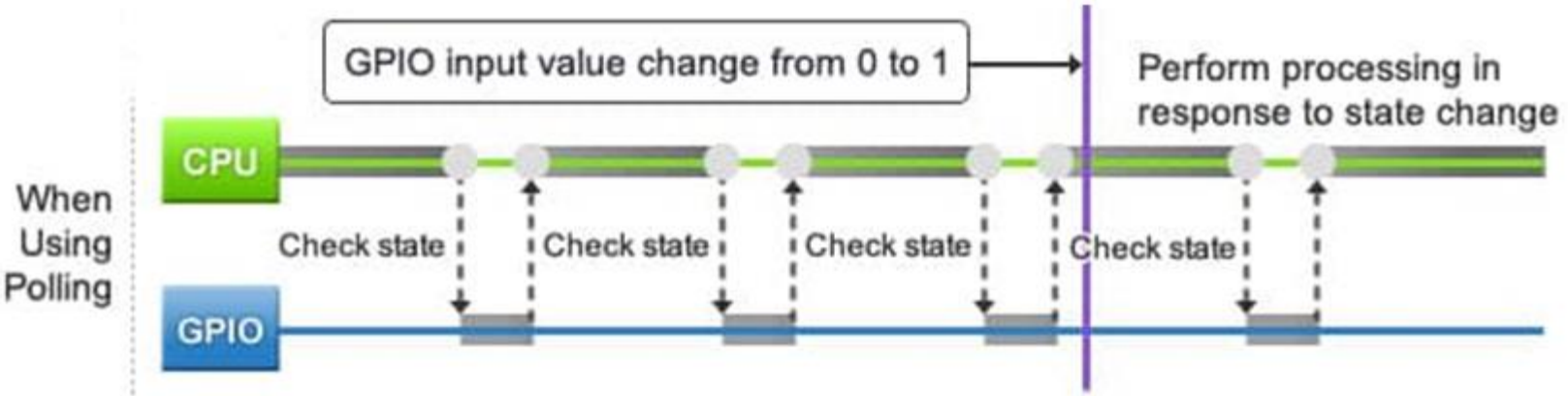
Polling is a protocol that keeps checking the flag bits (PIF) to notify whether a device has something to execute.

**Advantages:**

- Deterministic Technique. All events would be expected.

**Disadvantages:**

- May cause high latency time



# Defining ISR in AVR GCC

To be able to define an ISR in our tool chain AVR GDD, you to need include the library AVR/interrupt.h

```
#include "avr/interrupt.h"
```

Then to write the ISR body write the word ***ISR*** and between round brackets write the vector name:

```
ISR (INT0_vect)
{
    /* Interrupt Body */
}
```


1- ISR shall have no prototype

2- ISR is not callable, it means that it can not be called directly like a function. It is called by hardware when the corresponding interrupt happens.

3- ISR has no argument or return

Write a C code that uses 2 buttons of a DIP switch on 2 External Interrupt pins. The interrupt triggering source shall be configured as IOC. Each button has its own ISR that toggles different LED.

# Time To Code

# LAB2

Write a C code that apply Certain LED animation on 8 LEDs continuously. The animation shall change when a tactile switch is pressed. The tactile switch is connected to EXTI pin.

## Time To Code

# The End …

IMT SCHOOL

**www.imtschool.com**

**ww.facebook.com/imaketechnologyschool/**