Embedded System Interfacing

Lecture 6
Interrupts

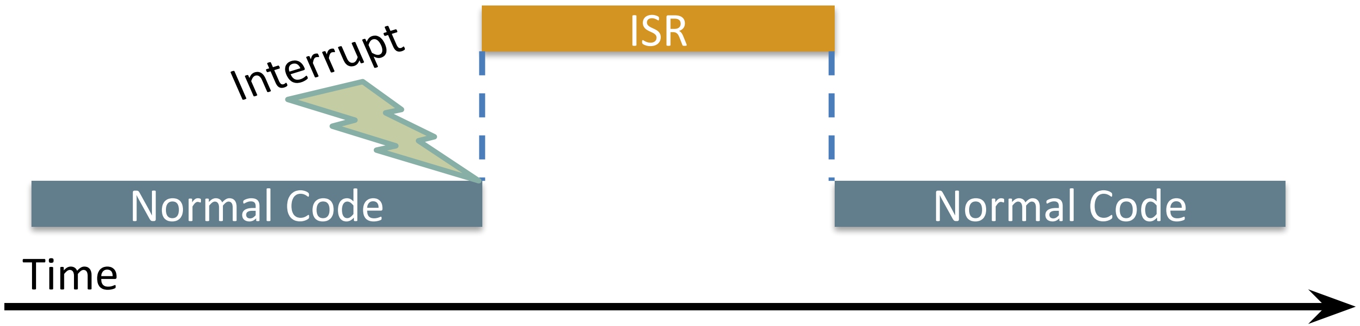
01

Interrupts

Interrupts

<u>***Definition:***</u>

Interrupt is ***an event*** the disturbs the normal execution sequence of the code and makes the processor to ***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.



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.

IMT SCHOOL

Interrupts Masking

1. Maskable Interrupt :

An Interrupt that can be disabled or ignored by the instructions of CPU are called as Maskable Interrupt.The interrupts are either edge-triggered or level-triggered or level-triggered.

2. Non-Maskable Interrupt :

An interrupt that cannot be disabled or ignored by the instructions of CPU are called as Non-Maskable Interrupt.A Non-maskable interrupt is often used when response time is critical or when an interrupt should never be disable during normal system operation. Such uses include reporting non-recoverable hardware errors, system debugging and profiling and handling of species cases like system resets.

Interrupts in Micro-Controller

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.

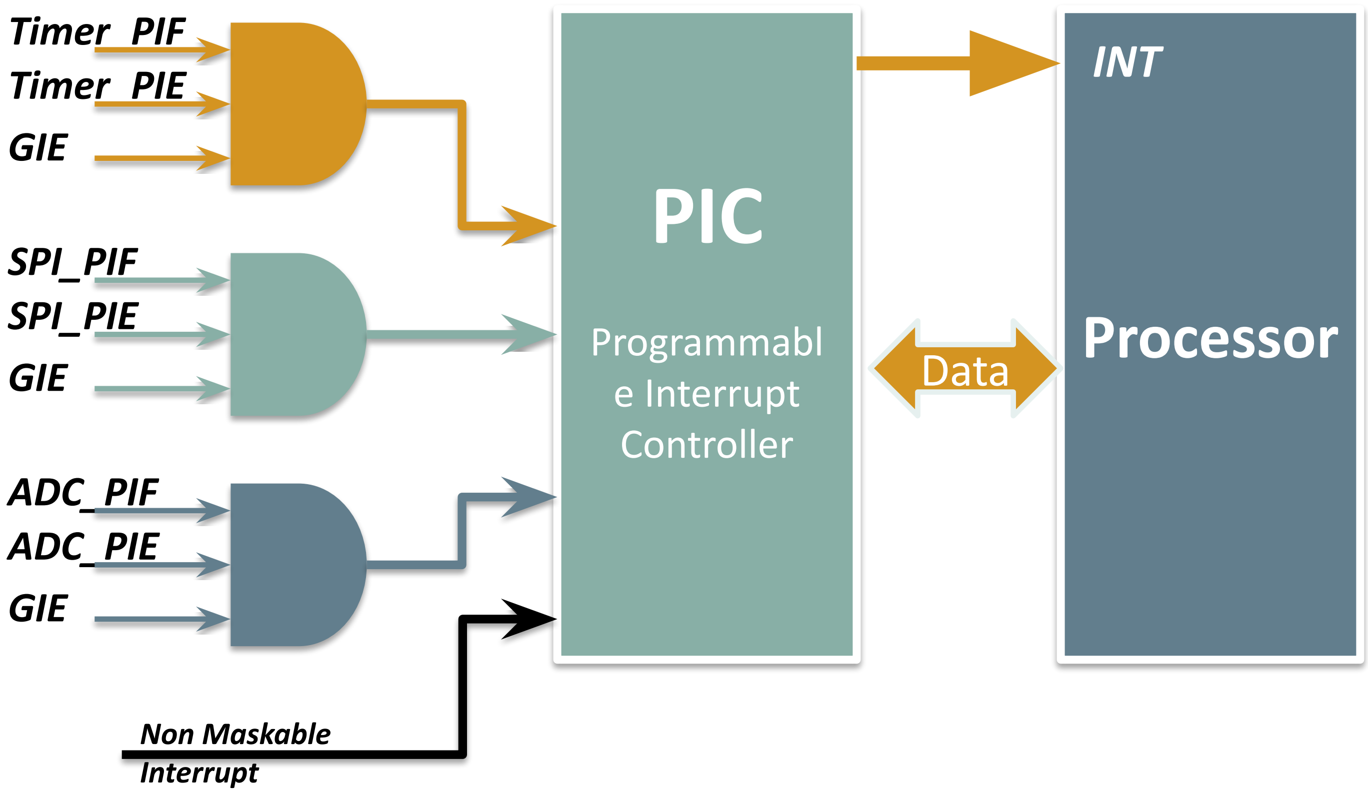
Instead, another bit called ***Global Interrupt Enable (GIE).*** When this bit is 0, All maskable interrupt are disabled **whatever** the PIE was 1 or 0.

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

External
Outside the processor

- UART
- ADC
- SPI
- EXTI

Internal
Inside the Processor

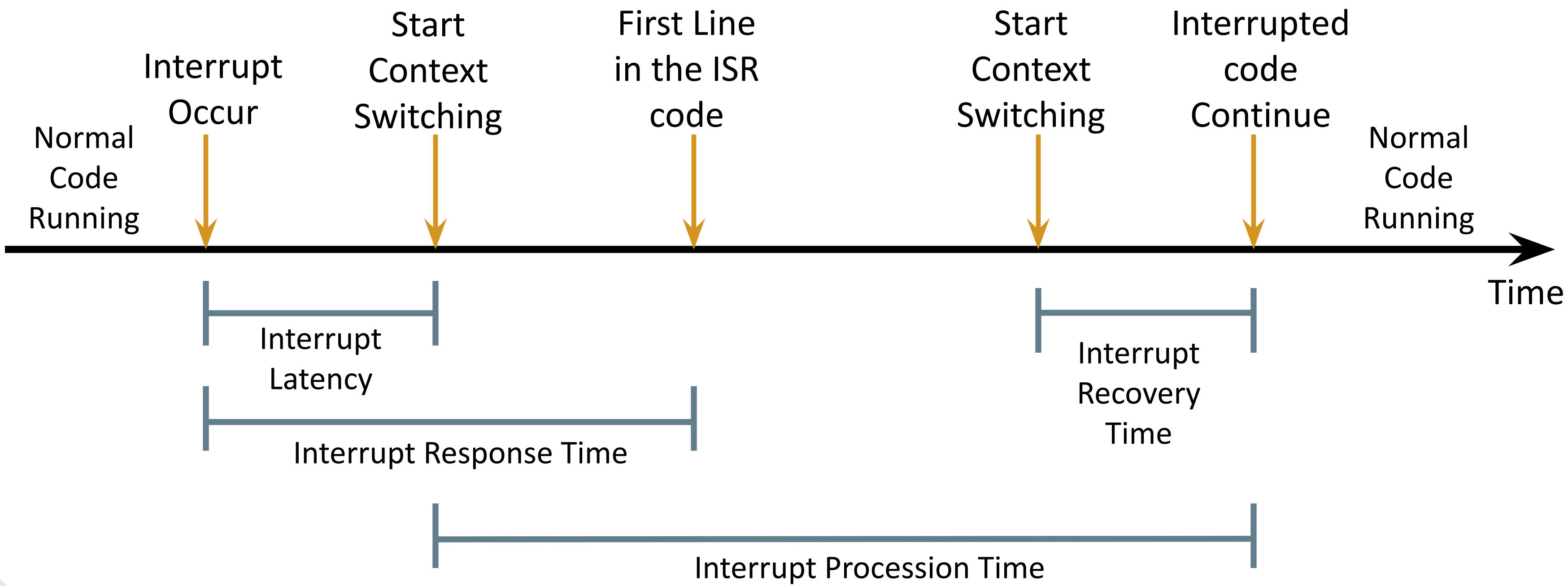
Trap

Ex: Division by Zero
Ex: Stack Overflow
Ex: Incorrect Opcode

Software Interrupt

Specific assembly instruction called (SWI) that fires an interrupt

Interrupt Performance Parameters



External Interrupt Peripheral

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:

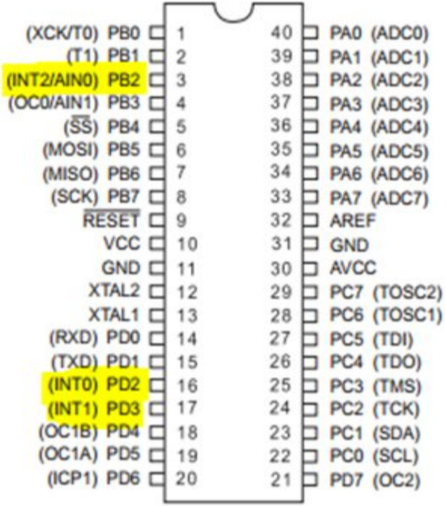
1- Falling Edge

2- Rising Edge

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

This type of interrupt called Interrupt On Change (IOC)

4- Low Level



SCHOOL

External Interrupt Peripheral

Microcontroller Control Register MCUCR

Bit	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.

External Interrupt Peripheral

General Interrupt Control Register GICR

Bit	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.

External Interrupt Peripheral

General Interrupt Flag Register GIFR

	7	6	5	4	3	2	1	0
GIFR	INTF1	INTF0	INTF2	–	–	–	–	–
	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.

External Interrupt Peripheral

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 Flag

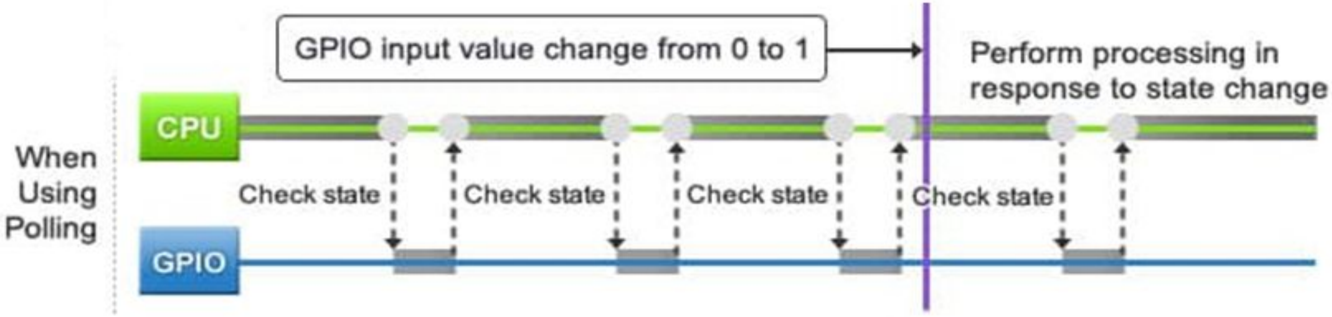
Polling is a protocol that keeps checking the flag bits (PIF) to notify whether a device has something to execute.

Advantages:

- Deterministic Technique. All events would be expected.

Disadvantages:

- May cause high latency time



Polling on Flag

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 */
}
```

Important Notice

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

LAb 1

LAP

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

Welcome | LAP

Users > abdelrahmanhossam > Desktop > LAP

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

LAb 2

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

www.facebook.com/imaketechnologyschool/

This material is developed by IMTSchool for educational use only