

Benha University

Shoubra Faculty of Engineering

**Communication and Computer
Engineering Department**

2025/2026 Academic Year

كلية الهندسة بشبرا
FACULTY OF ENGINEERING- SHOUBRA



Report on What key skills did you develop during your Embedded Systems studies?

Presented by

NOURA EMAD HAMEDY Ahmed

Summer

Contents

1. Embedded Systems Fundamentals	3
1. Introduction.....	3
How an Embedded System Works	3
Core Components:.....	3
Concept of Operation Using C Language:.....	3
2. Microchip PIC Microcontrollers	5
2.1 Overview of PIC Microcontrollers	5
2.2 Core Features of PIC Microcontrollers.....	5
3.2 Common Oscillator Types	6
3. Basic Connections & I/O Ports	6
4. First Program with MicroC & Proteus.....	7
5.7-Segment Displays	7
5.1 Overview	7
5.2 Types	7
Example: Displaying '0' on Common Cathode	8
6. Interrupts.....	8
6.1 What Are Interrupts?.....	8
6.2 How They Work	9
6.3 Types of Interrupts in the PIC16F877A Microcontroller	9
6.4 The RB0/INT External Interrupt	10
INTCON Register: Contains the main control bits:	10
• OPTION_REG Register	10
6.5 Conditions for Interrupt Execution.....	10
6.6 Programming Example: Configuring an RB0 Interrupt	11
Conclusion	12

1.Embedded Systems Fundamentals

1. Introduction

An **Embedded System** is a dedicated computer system designed to perform one or a few specific tasks, often within a larger mechanical or electrical system. These systems are characterized by real-time performance constraints, low power consumption, compact size, and cost-effectiveness.



How an Embedded System Works

An embedded system is a small computer dedicated to performing one or more specific functions within a larger system. Unlike a personal computer, an embedded system is designed to perform a specific task with precision and efficiency.

Core Components:

- **Central Processing Unit (CPU):** The brain of the system, it executes commands and programs.
- **Memory:** Used to store the program (non-volatile memory - ROM) and temporary data during execution (volatile memory - RAM).
- **Input/Output Ports (I/O Ports):** Allow the system to interact with the outside world, such as receiving data from sensors or controlling motors.
- **Peripherals:** Such as Timers, Counters, and Analog-to-Digital Converters (ADCs).

Concept of Operation Using C Language:

1. **Code Writing:**
 - Code is written in the C programming language using a program like MikroC.
 - The code contains the control logic that defines how the system interacts with its inputs and outputs. For example, you can write code that reads the temperature from a sensor and then turns on a fan if the temperature is above a certain limit.
2. **Compilation:**
 - The MikroC program compiles the C code into machine language (Assembly Code), which the CPU can understand and execute directly.

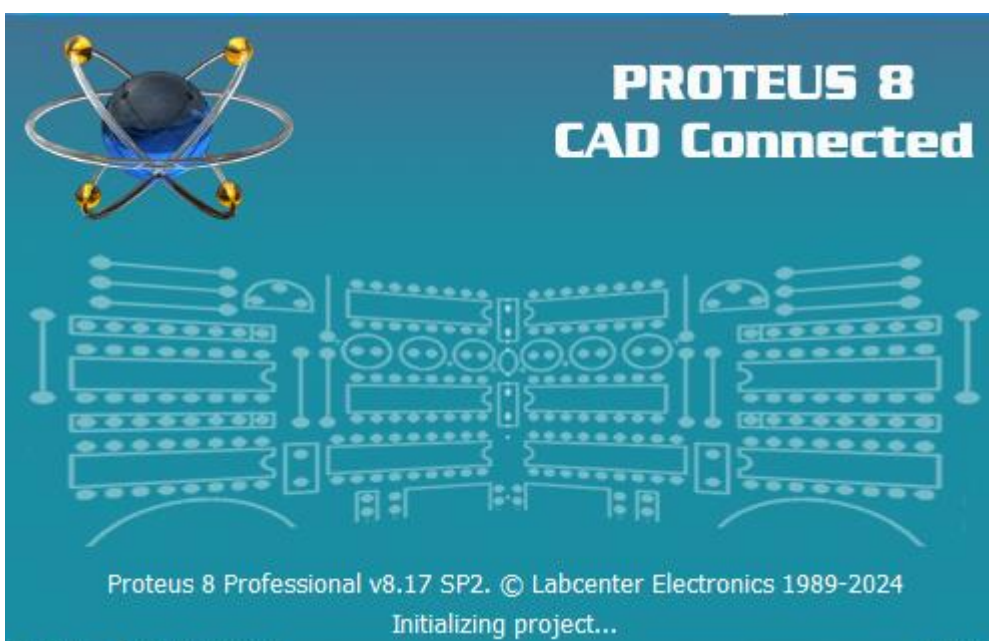
- The machine language is saved in a .hex file.

3. Simulation:

- The Proteus software is used to simulate the embedded system's electronic circuit.
- The .hex file created in the previous step is loaded into the microcontroller within the Proteus environment.
- Proteus then simulates the entire system's operation, allowing you to test the code and the circuit without the need to build a real one.

4. Programming:

- After verifying that the code works correctly in the simulation, the actual microcontroller is programmed by copying the .hex file to it.
 - The microcontroller then becomes capable of performing its function independently.
-



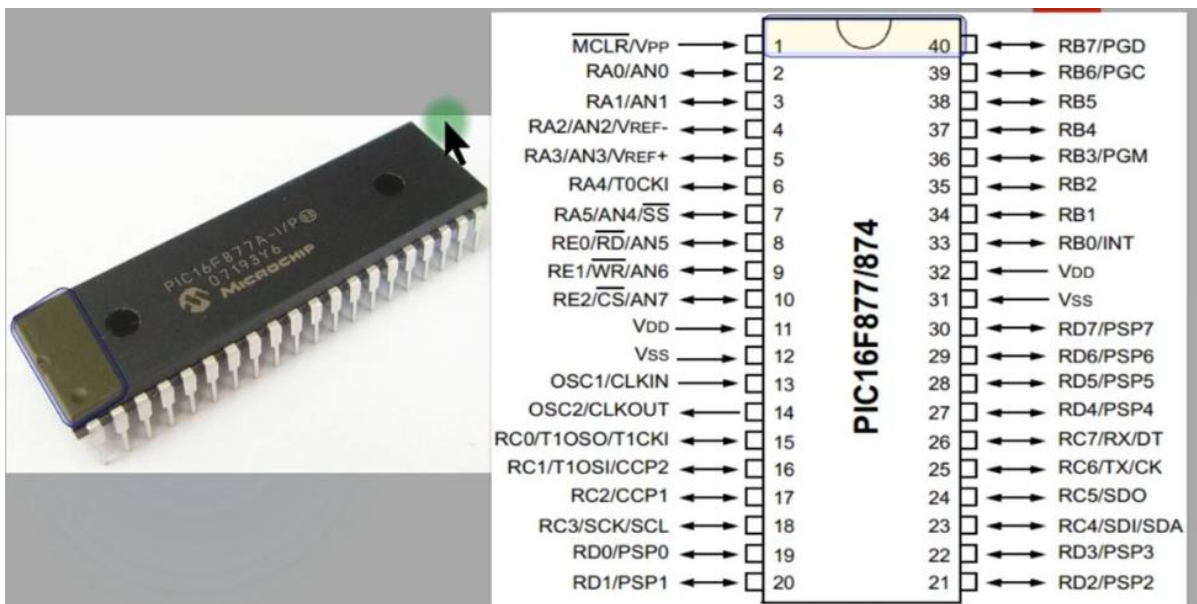
2. Microchip PIC Microcontrollers

2.1 Overview of PIC Microcontrollers

Peripheral Interface Controllers (PIC) from Microchip are widely used microcontrollers in embedded applications. They are known for:

- Simplicity
- Efficiency
- Wide range of models: PIC10, PIC12, PIC16, PIC18

The **PIC16F877A** is a popular educational and practical microcontroller due to its balance between features and simplicity.



2.2 Core Features of PIC Microcontrollers

Unit	Description
CPU	Executes instructions and controls data flow (the "brain").
I/O Ports	Configurable pins for input (e.g. sensors) or output (e.g. LEDs).
Memory	3 types: <ul style="list-style-type: none">– Flash: Stores program code (non-volatile)– RAM: Temporary data (volatile)– EEPROM: Permanent data (non-volatile)

3. Oscillator (Clock Generator)

3.1 Function

The oscillator provides timing signals that synchronize all internal operations of the microcontroller. Its frequency determines instruction execution speed.

3.2 Common Oscillator Types

Type	Frequency Range	Capacitor Range	Notes
LP	32 kHz	~33 pF	Low-power applications
XT	1–4 MHz	~15 pF	Stable crystal
HS	8–20 MHz	15–33 pF	High-speed operations
RC	Varies	None	Low-cost, less precise
Internal	Built-in	—	Requires no external parts

Note: Crystals often need capacitors to reduce noise and ensure stability.

3. Basic Connections & I/O Ports

To operate a PIC microcontroller:

1. **Power:** Connect Vcc (5V) and GND.
2. **Oscillator Circuit:** Crystal + capacitors connected to OSC1, OSC2.
3. **Reset (MCLR):** Connect to Vcc with a pull-up resistor.
4. **I/O Ports:** Pins configured as input/output using TRIS registers.

Example: Blinking LED on RB0

```
TRISB = 0x00;    // Set PORTB as output

while(1) {
    PORTB.b0 = 1;  // LED ON
    Delay_ms(1000);
    PORTB.b0 = 0;  // LED OFF
    Delay_ms(1000);
}
```

4. First Program with MicroC & Proteus

4.1 Tools

- **MicroC PRO for PIC:** Writing and compiling the code.
- **Proteus ISIS:** Simulating circuits using .hex file from MicroC.

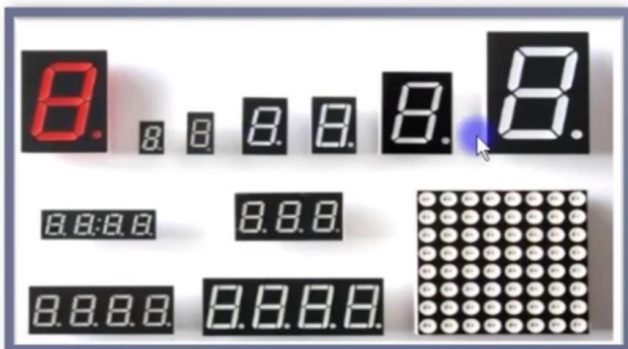
4.2 Steps

1. Write LED blinking code.
 2. Compile to generate .hex file.
 3. In Proteus, add PIC, crystal, LED, resistor.
 4. Load .hex into PIC.
 5. Run simulation – the LED blinks.
-

5. 7-Segment Displays

5.1 Overview

A **7-Segment Display** shows decimal numbers using seven LEDs labeled (a–g), and sometimes a decimal point (DP).

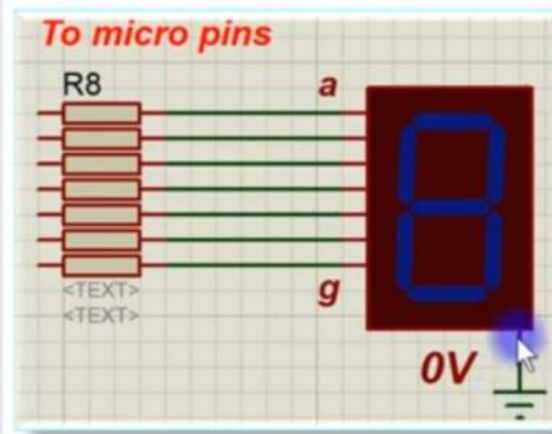


5.2 Types

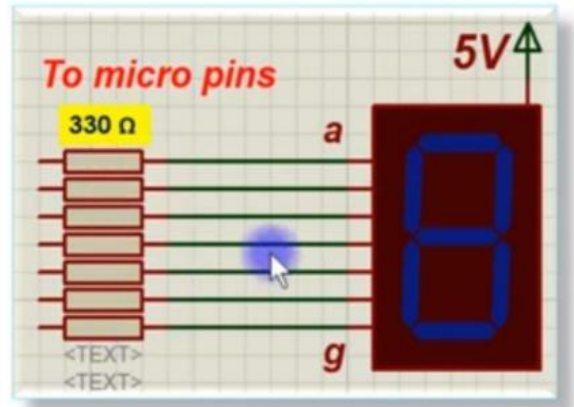
Type	Common Terminal	Lighting Logic
Common Anode (CA)	All anodes tied to V_{CC}	Segments lit by driving cathodes LOW
Common Cathode (CC)	All cathodes tied to GND	Segments lit by driving anodes $HIGH$

Each segment is connected to a microcontroller I/O pin through a resistor.

Common Cathode (CC)



Common Anode (CA)



Example: Displaying '0' on Common Cathode

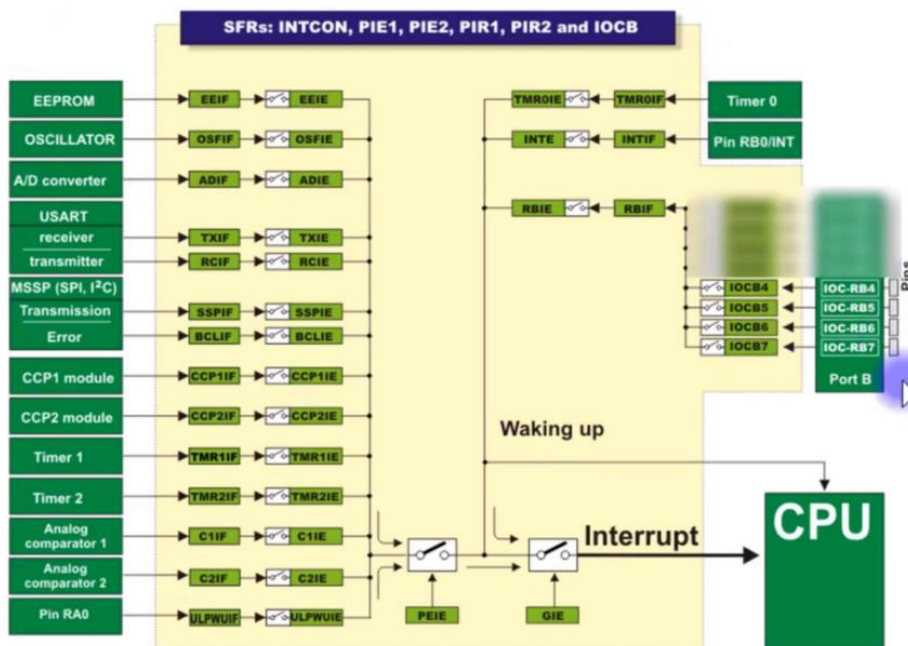
To light up a, b, c, d, e, f:

```
PORTD = 0b00111111; // Binary pattern for digit 0
```

6. Interrupts

6.1 What Are Interrupts?

Interrupts allow the microcontroller to react to internal/external events **asynchronously**, pausing current execution to handle urgent tasks.



6.2 How They Work

1. Event triggers an interrupt.
2. Controller saves current state.
3. Jumps to **Interrupt Service Routine (ISR)**.
4. Executes task (e.g., button press).
5. Returns to main code.

6.3 Types of Interrupts in the PIC16F877A Microcontroller

The PIC16F877A has 15 types of interrupts, broadly categorized into two types:

- **External Interrupts:** Triggered by signals from outside the microcontroller. Examples include:
 - MCLR Pin
 - RB0/INT Pin
 - Change on PORTB pins (RB4-RB7)
- **Internal Interrupts:** Triggered by events inside the microcontroller. Examples include:
 - Timers (Timer0, Timer1, Timer2)
 - Serial Communication (UART)
 - EEPROM

6.4 The RBO/INT External Interrupt

The RBO interrupt is a crucial external interrupt. To control it, specific registers are used:

INTCON Register: Contains the main control bits:

- **GIE (Global Interrupt Enable):** Enables/disables all interrupts. Must be set to 1 to enable any interrupt.
- **INTE (RBO/INT External Interrupt Enable):** Enables/disables the RBO interrupt specifically. Must be 1 to enable it.
- **INTF (RBO/INT External Interrupt Flag):** This flag is automatically set to 1 when the interrupt occurs. It must be cleared manually (set to 0) by the user within the ISR.

• **OPTION_REG Register:** Controls the behavior of the RBO interrupt:

- **INTEDG (Interrupt Edge Select):** Determines the signal edge that triggers the interrupt.
 - 1 = Interrupt on the rising edge.
 - 0 = Interrupt on the falling edge.
- **RBPU (PORTB Pull-up Enable):** Controls the internal pull-up resistors on PORTB pins.
 - 1 = Disables the pull-ups.
 - 0 = Enables the pull-ups.



6.5 Conditions for Interrupt Execution

For an interrupt to be executed successfully, three main conditions must be met:

1. **Cause Exists:** The event that causes the interrupt must occur, which automatically sets the flag bit (IF bit) to 1.

2. **Specific Interrupt Enabled:** The specific interrupt bit (IE bit), such as INTE_bit for the RBO interrupt, must be set to 1.
3. **Global Interrupts Enabled:** The global interrupt enable bit (GIE_bit) must be set to 1.

6.6 Programming Example: Configuring an RBO Interrupt

To configure the RBO interrupt using MicroC, you can use the following commands:

In the main program (void main()):

```
TrisB.B0 = 1; // Configure RBO as an input
GIE_bit = 1; // Enable global interrupts
INTE_bit = 1; // Enable the RBO interrupt
INTEDG_bit = 0; // Select falling edge trigger
// To enable internal pull-up resistors on PORTB
// NOT_RBPU_bit = 0;
```

In the Interrupt Service Routine (void interrupt()):

```
void interrupt() {
    if (INTF_bit == 1) { // Check if the interrupt is from RBO
        INTF_bit = 0; // Clear the interrupt flag
        //... Place your interrupt code here
    }
}
```

Note: It is crucial to always clear the interrupt flag bit (INTF_bit) within the ISR to prevent the interrupt from re-triggering immediately.

Conclusion

This report effectively summarizes the core concepts and practical skills developed in the field of embedded systems. It begins with a clear introduction to embedded systems and their key components. It then provides a detailed explanation of the workflow from

C programming to **MicroC compilation** and **Proteus simulation**. The document further explores the features of

PIC microcontrollers and the importance of components like **oscillators** and **7-segment displays**. A key highlight is the detailed explanation of

interrupts, covering how they work and how to program them, which is a critical skill for creating responsive embedded applications. Ultimately, this report demonstrates a strong grasp of both the theoretical fundamentals and the hands-on practical skills required for embedded systems development.