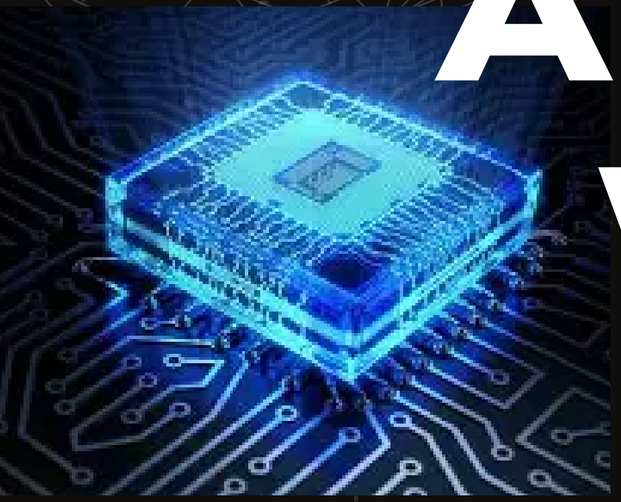


AUTOMATIC WASHING MACHINE USING PICSIMLAB

EMERTXE INTERNSHIP

AUTOMATIC WASHING MACHINE USING PICSIMLAB

EMERTXE INTERNSHIP

CREATIVE

2025

Y.MAHIMA,BTECH FINAL YEAR ECE,RGUKT NUZVID.

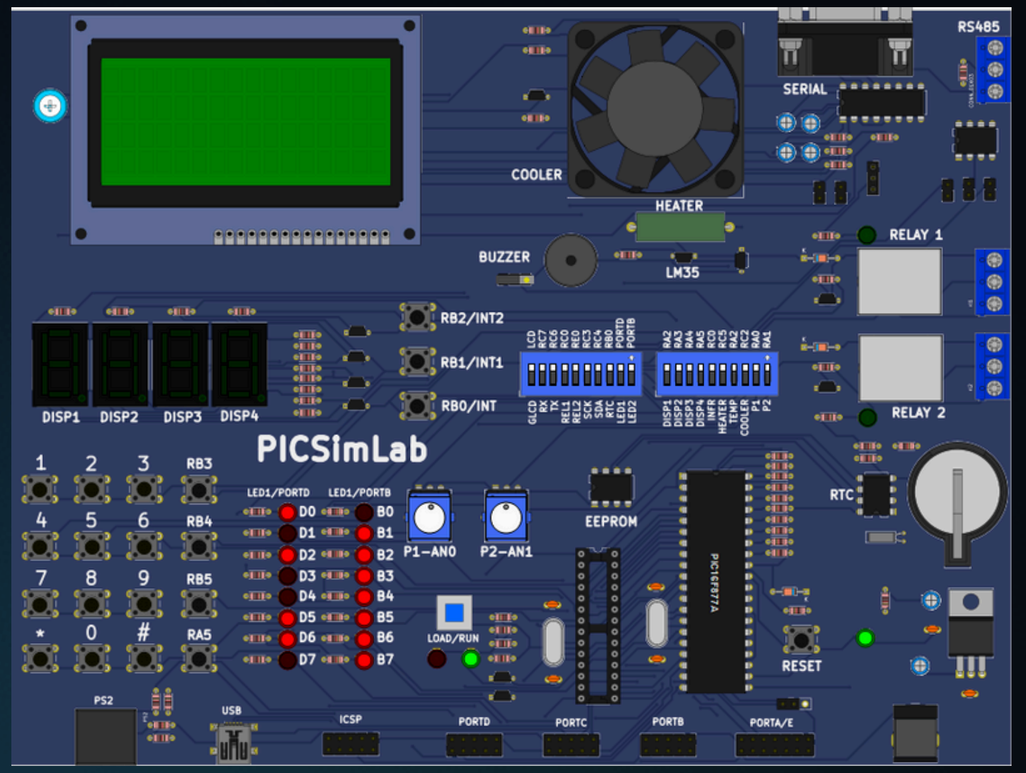
TABLE OF CONTENTS

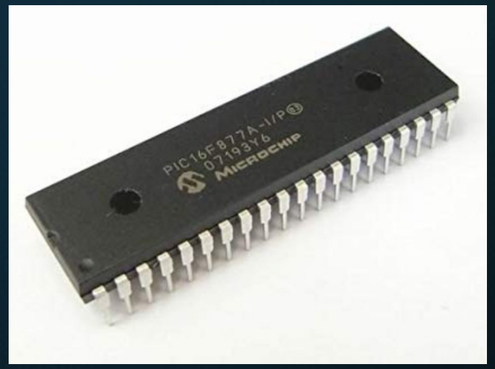
01 | Course Objective

02 | Project Implementation

what I learned:

- Gained hands-on experience in Embedded C for microcontroller programming.
- Understood the fundamentals of microcontrollers and how they work internally.
- Worked with the PIC16F877A microcontroller and simulated circuits using PICSimLab.
- Learned about I2C and UART communication protocols for data transfer.
- Implemented timers, interrupts, and basic interfaces like LEDs, keypads.
- Simulated a real-time project: Automatic Washing Machine.

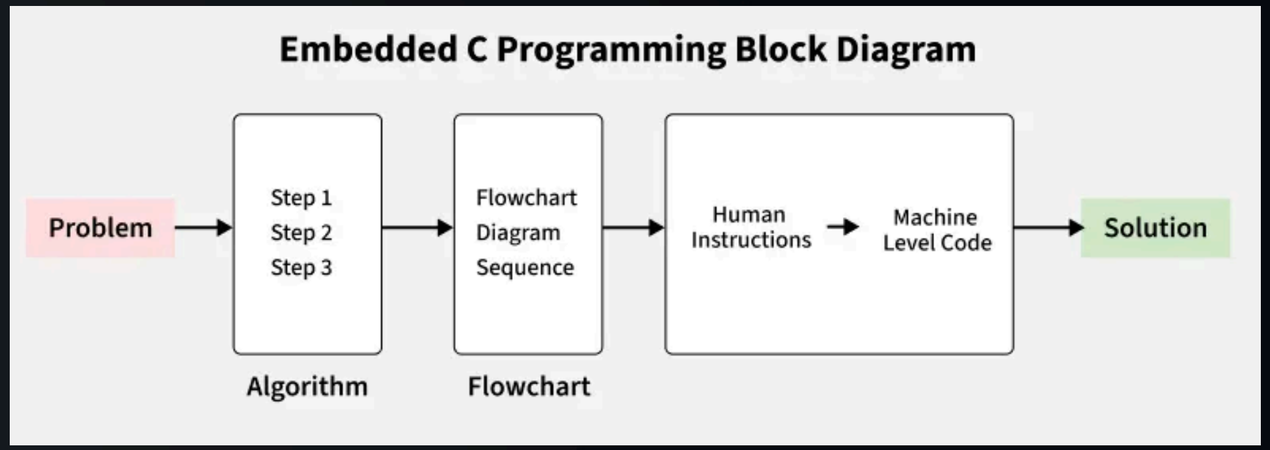




MPLAB® X IDE

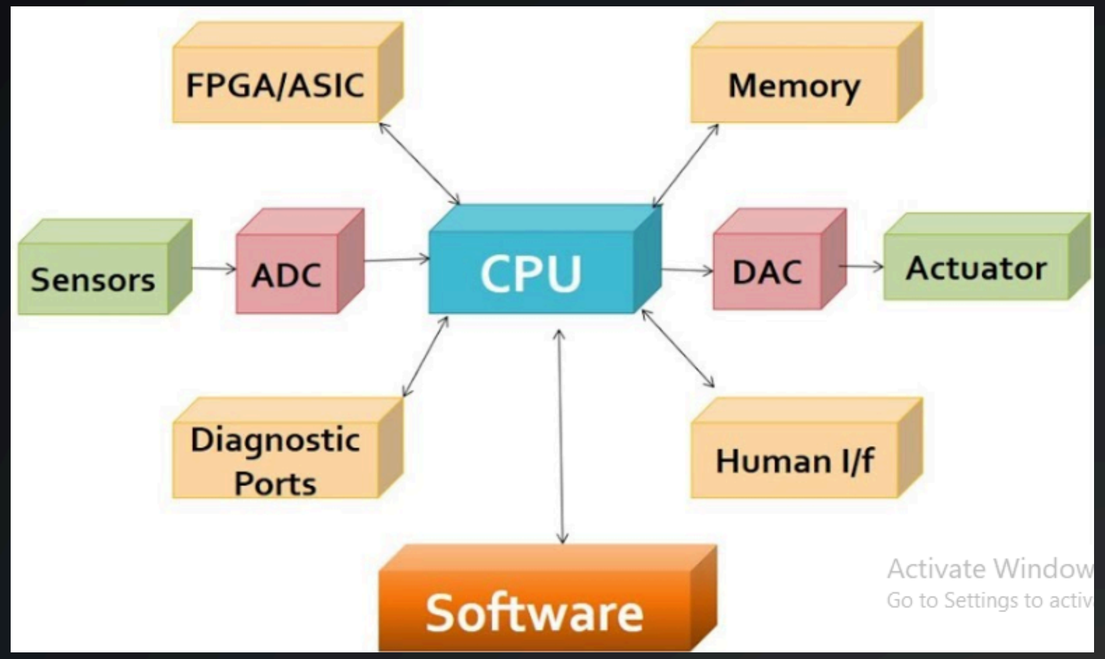
Introduction to Embedded C

- Embedded C is a version of C optimized for microcontroller programming.
- It allows direct control of hardware registers and I/O pins.
- Mainly used in bare-metal systems (without an operating system).
- Syntax is mostly like standard C, but adds access to hardware like PORTB, TRISB, etc.
- Common compilers: XC8, KEIL, GCC for ARM, etc.



C	Embedded C
Any desktop-based apps can be created using the general-purpose programming language C.	The programming language Embedded C is merely an extension of C, and it is used to create software for microcontrollers.
C is a straightforward language that is simple to read and edit.	The Embedded C language is difficult to read and change, making it difficult to use.
The C programming language's compilers are OS-dependent.	The Embedded C compilers are OS agnostic.
Fixing bugs in C language programmes is simple.	A software written in embedded C is challenging to bug fix.
When used, the C programming language also supports various other programming languages.	Only the processor that the application needs is supported by embedded C; other programming languages are not supported.

BLOCK DIAGRAM OF EMBEDDED SYSTEMS

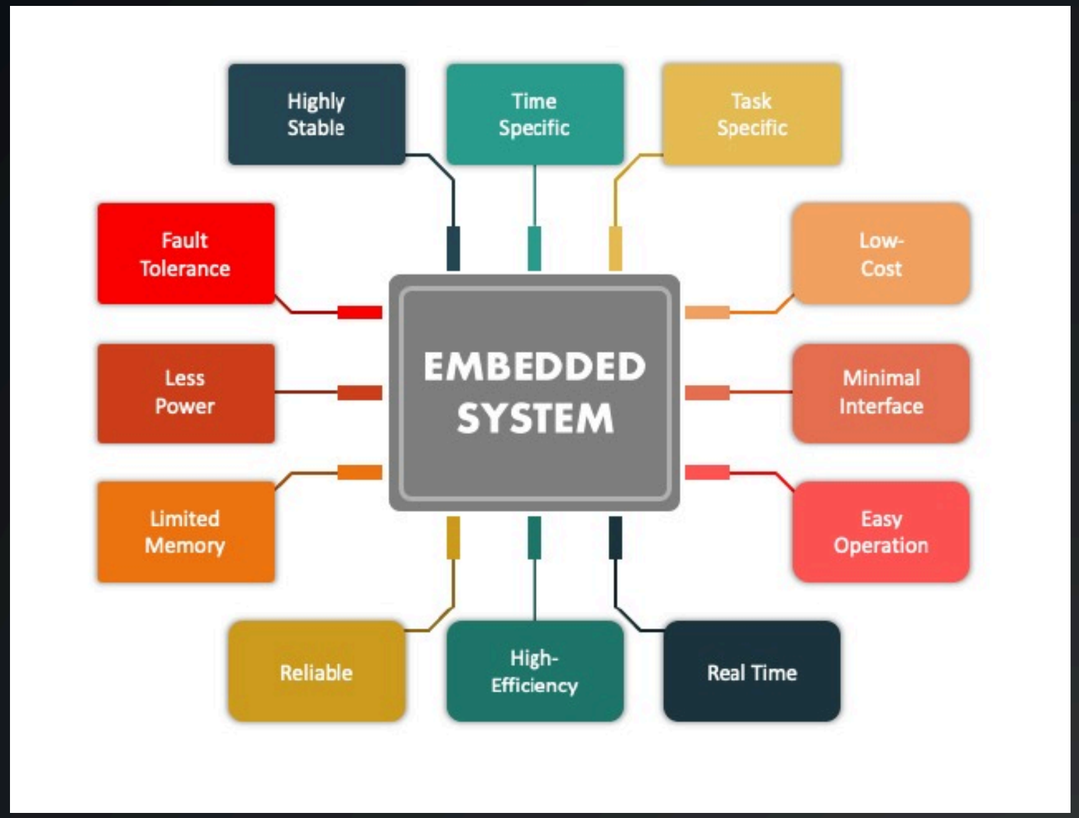


KEY EMBEDDED C CONCEPTS

- Data Types: int, char, float, unsigned, volatile, etc.
- Operators: Arithmetic, Logical, Bitwise (used for port manipulation).
- Control Structures: if, while, for, switch used to handle system logic.
- Registers: PORTB, TRISB are memory-mapped I/O registers.
- Pointers: Used to access specific memory addresses (e.g., *portb = 0xFF).

.....

FEATURES OF EMBEDDED SYSTEMS



BASIC EMBEDDED C CODE TO BLINK LEDS (FOR PIC16F877A)

```
#include <xc.h>          // Include header for PIC register definitions

void main(void)
{
   TRISB = 0x00;        // Set all PORTB pins as output (TRISB = 0 means output)
   PORTB = 0x00;        // Turn OFF all LEDs connected to PORTB initially
                .....
   while(1)             // Infinite loop to keep the program      running
   {
      PORTB = 0xFF;     // Turn ON all LEDs (all PORTB bits = 1)
      __delay_ms(1000); // Delay for 1 second (1000 milliseconds)

      PORTB = 0x00;     // Turn OFF all LEDs (all PORTB bits = 0)
      __delay_ms(1000); // Delay for 1 second again
   }
}
```

EXPLANATION

- Configures PORTB for output using TRISB.
- Toggles all LEDs ON and OFF using PORTB.
- Uses delay to blink LEDs every 1 second.
- Demonstrates real-time control over GPIO.

MICROCONTROLLERS OVERVIEW

- A microcontroller (µC) is an integrated circuit (IC) designed to perform specific control tasks. It combines:
- CPU (Central Processing Unit)
- Memory (RAM, ROM/Flash)
- Input/Output (I/O) ports
- Peripherals (Timers, ADCs, Serial Ports)
- It acts as a mini computer on a single chip, optimized for embedded applications like automation, robotics, and consumer electronics.

Popular Microcontroller Families

- 8051 – Classic 8-bit MCU, still widely taught.
- AVR – Used in Arduino boards (ATmega328).
- PIC – Used in education and industry (e.g., PIC16F877A).
- ARM Cortex-M – 32-bit powerful, low power, widely used in IoT.
- MSP430 – Ultra low power from Texas Instruments.

Microprocessor vs Microcontroller

Feature	Microcontroller	Microprocessor
Purpose	Specific task/control	General-purpose computing
Components	CPU + RAM + I/O + Peripherals	Only CPU
Cost	Low	High
Design	Compact (single chip)	Requires external memory/I/O
Example	PIC16F877A, 8051, AVR	Intel i5, AMD Ryzen

PIC16F877A – ARCHITECTURE & PERIPHERALS

OVERVIEW OF PIC16F877A

- A popular 8-bit microcontroller from Microchip
- Based on Harvard Architecture (separate instruction and data buses)
- Belongs to the PIC16 series, used widely in embedded systems and education
- Supports RISC instruction set (35 simple instructions)
- Operates up to 20 MHz, with powerful built-in peripherals

I/O PORTS

- PORTA (5 bits): Analog inputs (A0–A4) and digital I/O
- PORTB (8 bits): Digital I/O; used for LEDs, switches, etc.
- PORTC (8 bits): Used for serial communication (USART/SPI/I2C)
- PORTD (8 bits): General-purpose digital I/O
- PORTE (3 bits): Used mainly for analog input and control

Each port has:

- TRISx register → Configures pin direction (input/output)
- PORTx register → Controls output or reads input

KEY FEATURES

Feature	Specification
CPU	8-bit RISC
Program Memory	8K x 14-bit Flash
Data Memory (RAM)	368 Bytes
EEPROM	256 Bytes
I/O Ports	5 Ports (A, B, C, D, E)
Timers	3 Timers (TMR0, TMR1, TMR2)
ADC	10-bit resolution, 8 channels
Serial Interfaces	USART (UART), SPI, I2C
Interrupts	Multiple external and internal sources
Packages	40-pin PDIP, 44-pin QFP, 44-pin PLCC

SERIAL COMMUNICATION PERIPHERALS

USART (UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER)

- Supports asynchronous (UART) and synchronous serial communication
- Connects microcontroller to PC, GSM modules, or other microcontrollers

I2C (INTER-INTEGRATED CIRCUIT)

- Two-wire communication: SDA (data), SCL (clock)
- Used for communicating with EEPROM, RTC, sensors

SPI (SERIAL PERIPHERAL INTERFACE)

- 4-wire communication: SDO, SDI, SCK, SS
- Faster than I2C, used for short-distance high-speed communication

PWM – PULSE WIDTH MODULATION (VIA CCP)

- CCP Module (Capture/Compare/PWM) used to generate PWM signals
- Useful for:
- Motor speed control
- LED brightness control
- Signal generation

SPECIAL FUNCTION REGISTERS (SFRS)

- Control various peripherals and functions of the microcontroller
- Examples:
- TRISB – Controls direction of PORTB
- PORTB – Holds data for output/input on PORTB
- TMR0, TMR1, TMR2 – Timer registers
- TXREG, RCREG – USART transmission/reception
- ADRESH:ADRESL – ADC result register pair

COMMUNICATION PROTOCOLS IN EMBEDDED SYSTEMS

Microcontrollers often need to communicate with other devices like sensors, displays, memory chips, or even other microcontrollers. Communication protocols define how this data exchange happens — ensuring that the information is sent and received correctly.
The two most commonly used protocols in microcontrollers like PIC16F877A are I2C and UART.

1. I2C – INTER-INTEGRATED CIRCUIT

OVERVIEW

- Pronounced as “I-squared-C” or “I-two-C”
- Developed by Philips
- Synchronous, half-duplex, serial communication protocol
- Uses only two wires for communication:
  - SDA (Serial Data Line)
  - SCL (Serial Clock Line)

KEY FEATURES

- Multi-master, multi-slave support
- Master controls the clock; slaves respond accordingly
- Each device has a unique 7-bit or 10-bit address

BUS SPEED MODES

Mode	Speed
Standard Mode	100 kbps
Fast Mode	400 kbps
Fast Mode Plus	1 Mbps
High Speed Mode	3.4 Mbps

HOW I2C COMMUNICATION WORKS

- Start Condition – Initiated by master
- Address Frame – Master sends slave address + read/write bit
- ACK/NACK Bit – Slave acknowledges the request
- Data Transfer – 8-bit data is sent/received with acknowledgment
- Stop Condition – Communication ends

APPLICATIONS OF I2C

- Interfacing EEPROM, RTC (Real-Time Clock), and sensors
- Connecting multiple devices with only 2 lines
- Used where short-distance, slow-to-moderate speed communication is needed

COMMUNICATION PROTOCOLS IN EMBEDDED SYSTEMS (CONTI....)

2. UART – UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER

OVERVIEW

- Asynchronous serial communication protocol
- Requires no clock line — uses timing to sync data
- Uses 2 wires:
- TX (Transmit)
- RX (Receive)

KEY FEATURES

- Point-to-point communication (one transmitter, one receiver)
- Simple to use and implement
- Widely used for debugging, PC communication, GSM modules, etc.
- Requires setting of baud rate (data transfer speed)

COMMON UART SETTINGS

Parameter	Description
Baud Rate	Speed (e.g., 9600, 115200 bps)
Data Bits	Typically 8
Stop Bits	1 or 2
Parity Bit	Optional for error checking

HOW UART COMMUNICATION WORKS

- Start Bit – Indicates the beginning of data
- Data Bits – Usually 8 bits per frame
- Optional Parity Bit – Error detection
- Stop Bit(s) – End of the data frame
- Unlike I2C, UART doesn’t use a clock, so both devices must agree on baud rate.

APPLICATIONS OF UART

- Communication between microcontroller and PC (via USB to serial)
- Debug logging
- Interfacing GSM, GPS, Bluetooth modules
- Serial terminals (like PuTTY or QCOM)

TIMERS AND INTERRUPTS IN PIC MICROCONTROLLER (PIC16F877A)

WHAT ARE TIMERS?

Timers are internal counters in microcontrollers used to:

- Generate accurate delays
- Count external events
- Trigger periodic interrupts
- Create PWM signals

In PIC16F877A, timers increment based on internal clock frequency or an external clock source.

TYPES OF TIMERS IN PIC16F877A

Timer	Size	Features
TMR0	8-bit	Simple counter/timer, can use prescaler
TMR1	16-bit	High-resolution timer, supports external clock
TMR2	8-bit	Has prescaler and postscaler, used in PWM

TIMER REGISTERS

- Each timer has specific control and value registers:
- TMRx: Holds current timer count
- TMRxIF: Timer Interrupt Flag (set when overflow occurs)
- TMRxIE: Timer Interrupt Enable
- TMRxON: Enables the timer (via associated control bit)

PRESCALER AND POSTSCALER

- Prescaler: Divides the input clock frequency to slow down timer counting.
- Postscaler (Timer2): Controls how often the interrupt is triggered after timer overflow.

TIMERS AND INTERRUPTS IN PIC MICROCONTROLLER (PIC16F877A)

WHAT ARE INTERRUPTS?

Interrupts allow the microcontroller to respond immediately to specific events asynchronously, by pausing the main program and executing an Interrupt Service Routine (ISR).

TYPES OF INTERRUPTS IN PIC16F877A

Type	Examples
External Interrupt	Change on INT (RB0 pin)
Timer Interrupt	Timer0, Timer1, Timer2 overflow
Peripheral Interrupt	UART receive complete, ADC conversion done

HOW INTERRUPTS WORK

- An event (like timer overflow or pin change) triggers an interrupt flag.
- If that interrupt is enabled, and global interrupt is ON:
- The CPU jumps to the ISR (Interrupt Service Routine).
- After ISR execution, the main program resumes where it left off.

KEY REGISTERS IN INTERRUPTS

- INTCON – Controls global and peripheral interrupts.
- PIE1 / PIR1 – Peripheral interrupt enable and flag registers.
- GIE – Global Interrupt Enable bit
- PEIE – Peripheral Interrupt Enable

TOOLS USED IN INTERNSHIP

1. MPLAB X IDE

OVERVIEW:

- Official Integrated Development Environment (IDE) from Microchip.
- Used for writing, compiling, and debugging Embedded C code for PIC microcontrollers.

KEY FEATURES:

- Supports all PIC, dsPIC, AVR, and SAM devices.
- Integrates with XC8 compiler for 8-bit MCUs like PIC16F877A.
- GUI-based environment with editor, build tools, and simulator/debugger.
- Allows project management, error checking, and register-level debugging.

HOW I USED IT:

- Wrote Embedded C code for the automatic washing machine project.
- Compiled the code using XC8 compiler.
- Generated .hex file to simulate the code in PICSIMLab.

.....

ADVANTAGES:

- Free, feature-rich, and officially supported by Microchip.
- Simplifies code building, debugging, and memory viewing.

TOOLS USED IN INTERNSHIP

2. PICSIMLAB

OVERVIEW:

- An open-source simulator for PIC microcontroller boards.
- Simulates real hardware like buttons, LEDs, LCDs, sensors, keypads, EEPROM, etc.

KEY FEATURES:

- Supports PIC16F877A, the MCU used in your project.
- Simulates digital and analog interfaces.
- Includes virtual boards like PICGenios, PICUno, etc.
- Real-time simulation of LED blinking, keypad input, timers, interrupts.

HOW I USED IT:

- Loaded the .hex file generated from MPLAB.
- Simulated hardware behavior of my washing machine project.
- Verified LED control, timing, and switch input without needing real hardware.

ADVANTAGES:

- No physical hardware needed for testing.
- Interactive GUI with real-time feedback.
- Great for students and beginners to practice embedded applications.

.....

WORKFLOW SUMMARY

Write Code (MPLAB) → Compile to .hex → Load into PICSimLab → Simulate Hardware

WHY THESE TOOLS?

- Easy to set up and use for embedded development.
- Helps understand real-time system behavior without actual circuits.
- Supported all features needed for your internship project, like timers, GPIO, keypad, and LEDs.

PROJECT OVERVIEW: AUTOMATIC WASHING MACHINE SIMULATION

OBJECTIVE

To develop and simulate a fully functional automatic washing machine control system using the PIC16F877A microcontroller, written in Embedded C, and tested virtually on PICSimLab. The system mimics real washing machine operations with cycle selection, water level control, and automated execution of wash-rinse-spin cycles.

HOW I USED IT:

- User can select wash program and set water level using digital keypad input.
- Timers and interrupts control the duration of each cycle.
- The system provides visual feedback via a simulated CLCD display and LEDs.
- Ensures safety via door status check and pause/resume functionality.
- Designed using modular C code and organized across multiple functional files.

WORKING STAGES / LOGIC FLOW:

- Power On Screen
→ Displayed using animated CLCD loading effect

- Wash Program Selection
→ Scroll through 12 modes (Daily, Heavy, Woolens, etc.) using SW4

- Water Level Selection
→ Choose between Auto, Low, Medium, High, Max using SW4

- Start/Stop Screen
→ Use SW5 to Start, SW6 to Stop/Pause

- Run Program
→ Timers and ISR manage wash, rinse, spin cycle durations
→ CLCD displays mode, time remaining, and current stage

- Completion or Error Handling
→ “Program Completed” + Buzzer beep
→ Door Open: Shows warning, pauses cycle, resumes safely

DEMO.....

PROJECT OVERVIEW: AUTOMATIC WASHING MACHINE SIMULATION

CORE MODULES & FUNCTIONALITIES (FROM CODE)

Module/File	Purpose
main.c	Initializes system, manages mode transitions
Washing_machine_function_def.c	Contains all cycle logic: wash, rinse, spin, time setting, CLCD updates
digital_keypad.c	Reads short/long presses from virtual keypad
clcd.c	CLCD control and string display functions
timers.c + isr.c	Timer2 setup and ISR that decrements time every second
Washing_machine_header.h	Header file for modular coding

RESULT:

- Successfully simulated 12 washing programs with realistic time settings.
- Integrated user input, safety check, and status feedback on CLCD.
- Demonstrated solid understanding of microcontroller programming, peripheral handling, and real-time control using timers and interrupts.

KEY CODE SNIPPETS:

1. INTERRUPT SERVICE ROUTINE (ISR)

```
void __interrupt() isr(void)
{
   static unsigned long int count = 0;
   if (TMR2IF == 1)
   {
      if (++count == 20000) {
         if (sec != 0)
            sec--;
         else if (min != 0) {
            min--;
            sec = 60;
         }
         count = 0;
      }
      TMR2IF = 0;
   }
}
```

2. TIMER2 INITIALIZATION

```
void init_timer2(void)
{
   PR2 = 250;
   TMR2IE = 1;
   TMR2ON = 0;
}
```

3. WASHING PROGRAM SELECTOR

```
void washing_program_display(unsigned char key)
{
   if (key == SW4)
   {
      clear_screen();
      program_no++;
      if (program_no == 12) program_no = 0;
   }
   clcd_print("Washing program", LINE1(0));
   clcd_putch('*', LINE2(0));
   clcd_print(washing_program[program_no],
LINE2(1));
}
```

KEY CODE SNIPPETS:

4. WATER LEVEL SELECTION

```
void water_level_screen(unsigned char key)
{
   if (key == SW4)
   {
      clear_screen();
      level++;
      if (level == 5) level = 0;
   }
   clcd_print("WATER_LEVEL:", LINE1(0));
   clcd_print("*", LINE2(0));
   clcd_print(water_level[level], LINE2(2));
}
```

5.DOOR STATUS CHECK (SAFETY FEATURE)

```
void door_status_check()
{
   if (RB0 == 0)
   {
      FAN = OFF;
      TMR2ON = OFF;
      BUZZER = ON;
      clcd_print("Door is open", LINE1(0));
      while (RB0 == 0);
      BUZZER = OFF;
      FAN = ON;
      TMR2ON = ON;
   }
}
```

6. TIMER DISPLAY ON CLCD

```
clcd_putch(min / 10 + '0', LINE2(6));
clcd_putch(min % 10 + '0', LINE2(7));
clcd_putch(':', LINE2(8));
clcd_putch(sec / 10 + '0', LINE2(9));
clcd_putch(sec % 10 + '0', LINE2(10));
```

CHALLENGES FACED

1. Managing Multi-File Modular Code
   - Project was divided into multiple .c and .h files.
   - Ensuring proper linking of functions and avoiding scope errors was tricky at first.

2. Handling User Inputs via Keypad
   - Reading short vs long key presses and debouncing virtual keys required logical handling.
   - Implementing intuitive mode and level switching logic took iterative testing.

3. Simulating Real-Time Behavior in PICSIMLab
   - PICSIMLab didn’t behave exactly like real hardware.
   - Visualizing hardware behavior like LED blinking and fan response demanded accurate timing.

4. Debugging Logical Bugs
   - Logic bugs (like timer not stopping or wrong mode display) required careful step-by-step debugging.
   - Using CLCD outputs as debug messages helped track the flow

KEY LEARNINGS

1. Strong Foundation in Embedded C
   - Gained confidence in writing efficient and modular Embedded C code.
   - Learned to use bit manipulation, pointers, and register-level programming.

2. Real-Time System Design
   - Understood how to implement real-time tasks using timers and interrupts.
   - Learned the value of ISR-driven timing vs delay-based loops.

3. Microcontroller Interfacing
   - Hands-on experience with I/O ports, timers, CLCDs, keypads, and safety logic.
   - Learned to control hardware simulation based on logical flow.

4. Importance of Code Modularity
   - Realized the value of organizing code across multiple modules:
     - Improves readability, reusability, and debugging.

5. Simulation and Testing Skills
   - Became comfortable with using PICSIMLab as a real hardware substitute.
   - Understood how to simulate, test, and validate MCU-based systems without physical boards.

CONCLUSION

- This internship provided me with a strong practical foundation in Embedded Systems, particularly in working with microcontrollers and real-time logic design.

- I gained hands-on experience in Embedded C programming, using MPLAB X IDE, and simulating hardware using PICSimLab — all crucial tools in embedded development.

- By designing and simulating the Automatic Washing Machine, I understood how to:
  - Interface peripherals like keypads, CLCDs, LEDs
  - Use timers and interrupts for real-time control

- Overall, this project helped me connect theoretical knowledge with practical implementation and prepared me for future roles in the embedded systems industry.