Indian Institute of Engineering Science and Technology (IIEST), Shibpur

Summer Research Internship Report
on

Implementation of Open-Loop V/f Control for Low-Power Induction Motors Using PIC18F4550-Based Embedded Systems

Under the Guidance of
Prof. (Dr.) Mainak Sengupta

by

ATRI DATTA
Roll - 2022EEB091, 4th year

Department of Electrical Engineering
IIEST Shibpur

Acknowledgements

I would like to thank Prof. (Dr.) Mainak Sengupta for providing such a valuable internship experience at the Advanced Power Electronics Laboratory at IIEST Shibpur. The internship experience was truly educational where we were able to map theory studied in lectures to the practical laboratory environment.

Also, I would like to thank the Department of Electrical Engineering for providing us the time to be involved in such a valuable learning experience.

My gratitude to Dr. Debraj Roy and Mr. Subhajit Dey for the valuable technical help. It helped me bridge the gap between theory and real-time systems.

My special thanks to Mr. Sambhuti Pathak and Mr. Arnab Bose for making this experience memorable and special. Without their help and support this internship would have been incomplete.

Last but not the least, thanks to my mother for the inspiration and support.

_____
ATRI DATTA
ROLL - 2022EEB091
Department of Electrical Engineering, IIEST Shibpur

Contents:

I. Abstract

This report demonstrates the internship work based on "**Implementation of Open-Loop V/f Control for Low-Power Induction Motors Using PIC18F4550-Based Embedded Systems**". This is a case study of V/f control specifically using PIC18F4550, which is not specific for motor control.

The report contains details of the basic Embedded C codes tested and the concept of the novel idea of code algorithm for 3-phase SPWM generation using the aforementioned embedded controller. Then for the verification of this code, the mathematical analysis of SPWM based inverter stack operation and the derivation of why V/f control is necessary is discussed. Finally the results are analysed in detail with graphs and trends.

II. Introduction

In modern power electronics and motor control applications, embedded controllers have become essential due to their flexibility, programmability, and precision. Unlike traditional analog controllers, which are limited by fixed circuitry and susceptibility to drift and noise, embedded microcontrollers like the PIC18F4550 offer digital control, real-time adaptability, and integration of complex algorithms. This report explores the application of such a controller in implementing SPWM for inverter operation and machine drive control. With increasing demand for compact, efficient, and intelligent drive systems, embedded solutions provide a scalable platform for innovation in both academic research and industrial automation systems.

1. PIC18F Architecture

For this internship project, a PIC18F4550 microcontroller was used. Here, some relevant information regarding the microcontroller is listed below:

- This is a microcontroller with an 8-bit processor. Its logic levels are tied at 0 and 5 volts.

- The program memory (Flash) and data memory (RAM/EEPROM) are separate, allowing parallel access and faster operation (i.e., Harvard Architecture).

- I/O Handling (TRIS, LAT, PORT) :
  - TRISx: Sets pin direction (input/output).
  - LATx: Writes output data to pins (avoids read-modify-write issues).
  - PORTx: Reads the actual pin states.

- It is used for precise delays, frequency generation, PWM, and event counting. Important timers used are Timer0, Timer1, Timer2 & Timer3 (Timer 3 for PIC18F4550).

- It allows PWM generation, frequency measurement, and time interval generation. ECCP (Enhanced CCP) offers advanced PWM options like multiple outputs and dead-time control.

- It supports high-priority and low-priority interrupts.

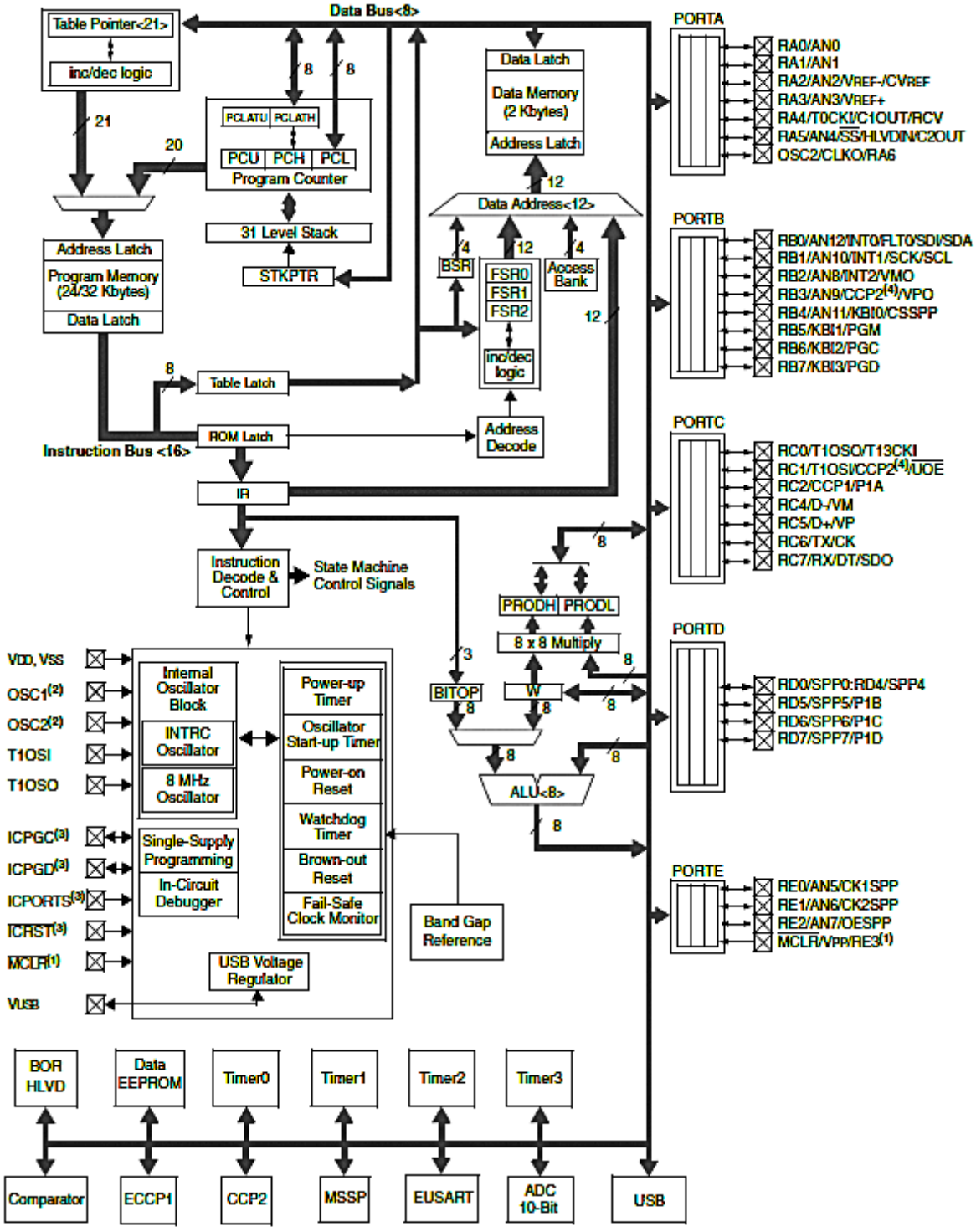
- It can work without external crystals using an internal oscillator (up to 8 MHz). PLL allows multiplying clock frequency (up to 48 MHz). Maximum frequency of oscillating crystal that can be interfaced with PIC18F is upto 20MHz for safe operation.

- Memory Types:
  - Flash Memory: Stores the main program (user code).
  - RAM: Temporary data storage during execution.
  - EEPROM: Stores data that must be retained after power-off (settings, calibration data).

- It has a built-in 8-bit ADC module. To access it PORT A is to be configured for input of analog signals to produce 8-bit digital data.

- It has a Watchdog Timer that prevents the system from hanging by resetting the microcontroller if the code gets stuck.

- In-Circuit Serial Programmer (ICSP) enables the microcontroller to be reprogrammed without removing it from the circuit using the PGD and PGC pins.



Note 1: RE3 is multiplexed with $\overline{MCLR}$ and is only available when the $\overline{MCLR}$ Resets are disabled.
2: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O. Refer to Section 2.0 "Oscillator Configurations" for additional information.
3: These pins are only available on 44-pin TQFP packages under certain conditions. Refer to Section 25.9 "Special ICPORT Features (Designated Packages Only)" for additional information.
4: RB3 is the alternate pin for CCP2 multiplexing.

Figure 1.a: Block Diagram Architecture

The figure below shows the pin diagram of the PIC18F4550, and the table lists the functions of its pins:

Pin	Name(s)	Function
1	MCLR / VPP	Reset & programming voltage input
2–5	RA0–RA3 / AN0–AN3	Analog inputs / Digital I/O
6	RA4 / T0CKI	Digital I/O / Timer0 external clock
7	RA5 / AN4 / SS	Analog input / SPI Slave Select
8–10	RE0–RE2 / AN5–AN7	Analog input / Digital I/O
11–12	AVDD / AVSS	Analog & USB power pins (must be connected)
13–14	OSC1 / OSC2	Crystal/resonator connections
16–17	RC1 / CCP2, RC2 / CCP1	PWM outputs (CCP1, CCP2)
18	RC3 / SCK / SCL	SPI/I²C Clock
23–24	RC4 / D+, RC5 / D–	USB data lines
25–26	RC6 / TX, RC7 / RX	UART communication
33	VUSB	USB regulator capacitor (connect 470 nF)
34–36	RB0–RB2 / INT0–INT2	External interrupts
37–40	RB4–RB7	Digital I/O with interrupt-on-change

Table 1.a: Pin description of PIC18F4550

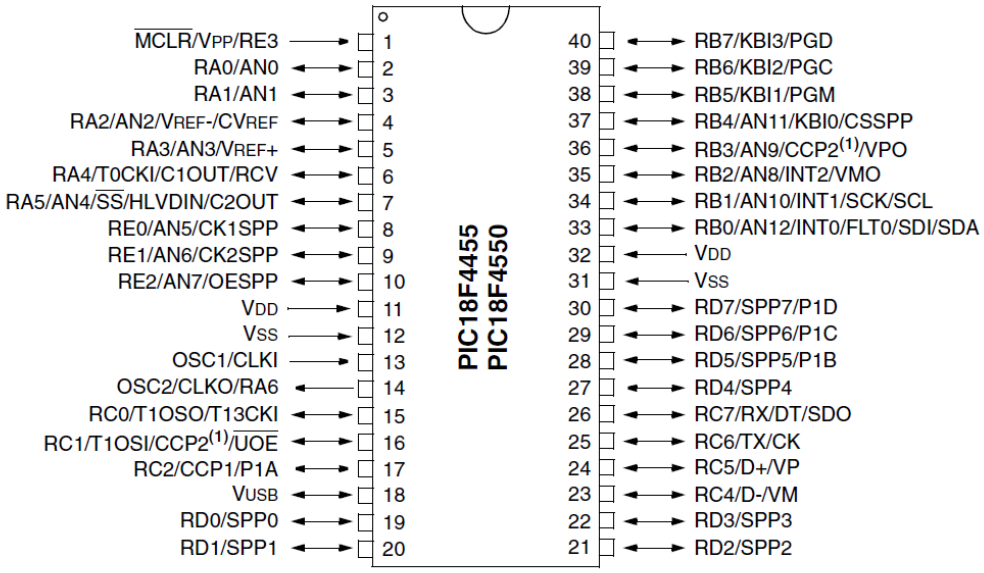


Figure 1.b: Pin Diagram of PIC18F4550 (40 pin PDIP)

The figure below shows the pin diagram of the PIC18F4431. This MCU is specific for SPWM, motor control, and precision analog sensing. The table lists the functions of its pins:

Pin(s)	Name(s)	Function
1	MCLR / VPP	Reset input / Programming voltage
2–5	RA0–RA3 / AN0–AN3	Analog inputs (12-bit ADC) or digital I/O
6	RA4 / T0CKI / C1OUT	Timer0 clock / Comparator1 out / Digital I/O
7	RA5 / AN4 / C2OUT	Analog input / Comparator2 out
8–10	RE0–RE2 / AN5–AN7	More analog inputs / Digital I/O
11–12	AVDD / AVSS	Power for analog & PWM modules (must connect)
13–14	OSC1 / OSC2	Crystal/resonator connections
15–18	RC0–RC3 / PWM0–PWM3	4 hardware PWM outputs for SPWM / motor control
19–22	RC4–RC7	General digital I/O
23–26	RD0–RD3	General-purpose digital I/O
27–30	RD4–RD7 / PDC0–PDC3	Dead-time control PWM outputs
33–35	RB0–RB2 / INT0–INT2	External interrupt pins
37–40	RB4–RB7	I/O with interrupt-on-change capability

Table 1.b: Pin description of PIC18F4431

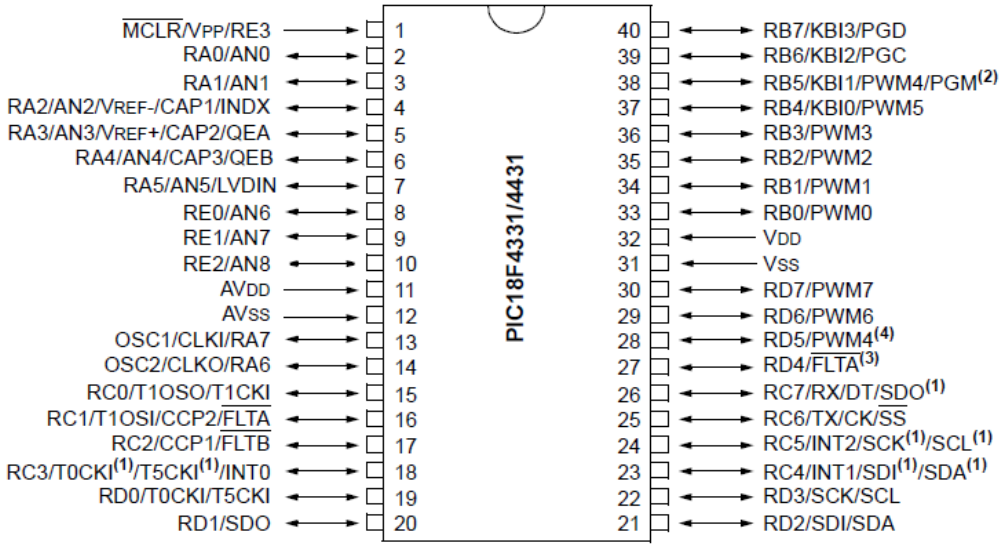


Figure 1.c: Pin Diagram of PIC18F4431 (40 pin PDIP)

Here is the development board used for the project:

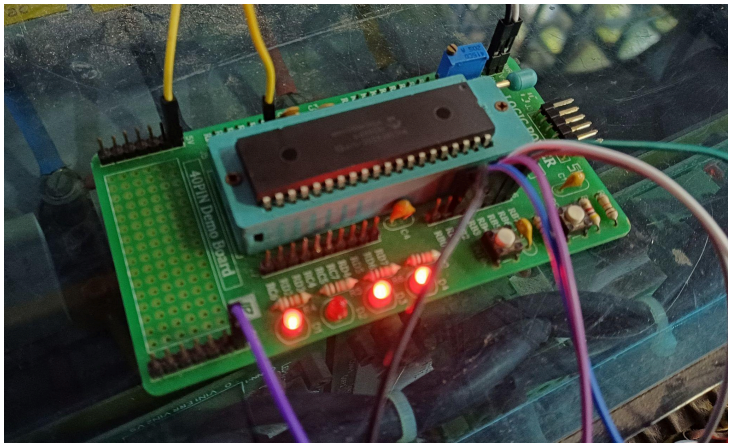


Figure 1.d: PIC18F4550 development board used

This is the schematic diagram of the PIC18F family board. All development boards of this family has the same PCB layout:

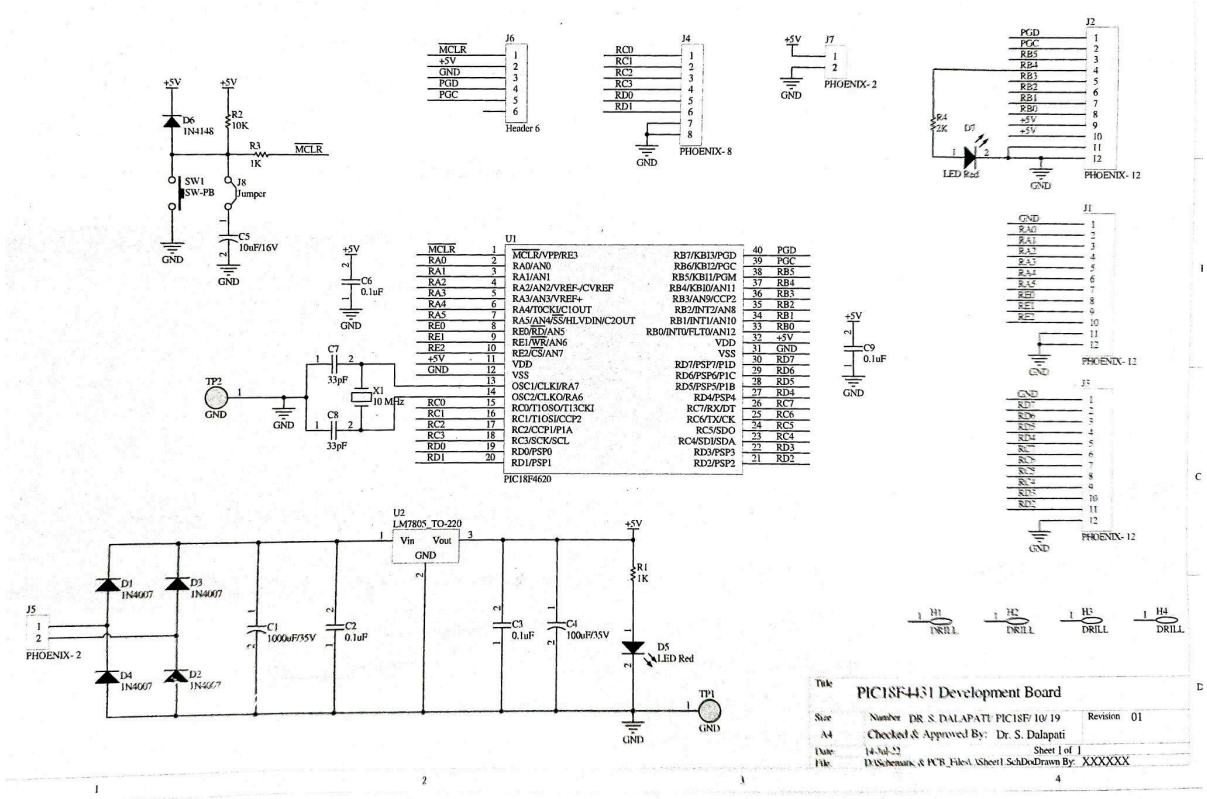


Figure 1.e: PIC18F4431 development board schematic
(approved by Dr. Suvarun Dalapati)

2. PIC18F Registers (8 bit) Used

There are 371 control registers in the architecture of PIC18F4550. Here is the broad classification of the major registers:

Category	Registers
General Purpose Control	STATUS, INTCON, RCON, OSCCON, WDTCON, T0CON, T1CON, T2CON, T3CON
I/O Port Control	TRISA, TRISB, TRISC, TRISD, TRISE, PORTA, PORTB, PORTC, PORTD, PORTE, LATA, LATB, LATC, LATD, LATE
Interrupt System	PIE1, PIE2, PIR1, PIR2, IPR1, IPR2, INTCON, INTCON2, INTCON3
Timer Modules	T0CON, T1CON, T2CON, T3CON, TMR0L, TMR0H, TMR1L, TMR1H, TMR2, PR2, TMR3L, TMR3H, CCPR1L, CCPR1H, CCPxCON
Analog Modules	ADCON0, ADCON1, ADCON2, ADRESH, ADRESL, CMCON, CVRCON
Communication Modules	TXSTA, RCSTA, SPBRG, SSPCON1, SSPCON2, SSPSTAT, SSPBUF, UCON, UCFG, UADDR, USTAT, UIR, UIE
Memory Control	EECON1, EECON2, TABLAT, TBLPTRU, TBLPTRH, TBLPTRL, FSR0L, FSR0H, FSR1L, FSR1H, FSR2L, FSR2H
Special Features	BORCON, HLVDCON, SPPCON (for 40/44-pin), LCDCON (other variants if applicable)

Table 2.a: Broad classification of registers in PIC18F4550

Out of all them, my work included extensive use of the following 5 registers.

a. TIMER0 register
b. TIMER1 register
c. TIMER2 register
d. Interrupt control register
e. Capture/Compare/PWM register 1
f. Peripheral Interrupt Request 1 (PIR1) register
g. Peripheral Interrupt Enable 1 (PIE1) register

I. TIMER0 register

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

Figure 2.a: TIMER0 register layout

II. TIMER1 register

R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **RD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer1 in one 16-bit operation
0 = Enables register read/write of Timer1 in two 8-bit operations

bit 6 **T1RUN:** Timer1 System Clock Status bit
1 = Device clock is derived from Timer1 oscillator
0 = Device clock is derived from another source

bit 5-4 **T1CKPS1:T1CKPS0:** Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN:** Timer1 Oscillator Enable bit
1 = Timer1 oscillator is enabled
0 = Timer1 oscillator is shut off
The oscillator inverter and feedback resistor are turned off to eliminate power drain.

bit 2 **$\overline{\text{T1SYNC}}$:** Timer1 External Clock Input Synchronization Select bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS:** Timer1 Clock Source Select bit
1 = External clock from RC0/T1OSO/T13CKI pin (on the rising edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR1ON:** Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

Figure 2.b: TIMER1 register layout

III. TIMER2 register

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **T2OUTPS3:T2OUTPS0**: Timer2 Output Postscale Select bits

0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON**: Timer2 On bit

1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Figure 2.c: TIMER2 register layout

IV. Interrupt Control Register (INTCON)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all high priority interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit(1)
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Note 1: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

Figure 2.d: Interrupt Control Register (INTCON) register layout

V. Capture-Compare-PWM x Control Register (CCPxCON)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—(1)	—(1)	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-6 **Unimplemented:** Read as '0'(1)

bit 5-4 **DCxB1:DCxB0**: PWM Duty Cycle Bit 1 and Bit 0 for CCPx Module

Capture mode:
Unused.

Compare mode:
Unused.

PWM mode:
These bits are the two LSbs (bit 1 and bit 0) of the 10-bit PWM duty cycle. The eight MSbs of the duty cycle are found in CCPR1L.

bit 3-0 **CCPxM3:CCPxM0**: CCPx Module Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
0001 = Reserved
0010 = Compare mode: toggle output on match (CCPxIF bit is set)
0011 = Reserved
0100 = Capture mode: every falling edge
0101 = Capture mode: every rising edge
0110 = Capture mode: every 4th rising edge
0111 = Capture mode: every 16th rising edge
1000 = Compare mode: initialize CCPx pin low; on compare match, force CCPx pin high (CCPxIF bit is set)
1001 = Compare mode: initialize CCPx pin high; on compare match, force CCPx pin low (CCPxIF bit is set)
1010 = Compare mode: generate software interrupt on compare match (CCPxIF bit is set, CCPx pin reflects I/O state)
1011 = Compare mode: trigger special event, reset timer, start A/D conversion on CCP2 match (CCPxIF bit is set)
11xx = PWM mode

Note 1: These bits are not implemented on 28-pin devices and are read as '0'.

Figure 2.e: Capture-Compare-PWM x Control Register (CCPxCON) register layout

VI. Peripheral Interrupt Request 1 (PIR1) Register

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
SPPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **SPPIF:** Streaming Parallel Port Read/Write Interrupt Flag bit(1)
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred

bit 6 **ADIF:** A/D Converter Interrupt Flag bit
1 = An A/D conversion completed (must be cleared in software)
0 = The A/D conversion is not complete

bit 5 **RCIF:** EUSART Receive Interrupt Flag bit
1 = The EUSART receive buffer, RCREG, is full (cleared when RCREG is read)
0 = The EUSART receive buffer is empty

bit 4 **TXIF:** EUSART Transmit Interrupt Flag bit
1 = The EUSART transmit buffer, TXREG, is empty (cleared when TXREG is written)
0 = The EUSART transmit buffer is full

bit 3 **SSPIF:** Master Synchronous Serial Port Interrupt Flag bit
1 = The transmission/reception is complete (must be cleared in software)
0 = Waiting to transmit/receive

bit 2 **CCP1IF:** CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode.

bit 1 **TMR2IF:** TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF:** TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

Note 1: This bit is reserved on 28-pin devices; always maintain this bit clear.

Figure 2.f: Peripheral Interrupt Request 1 (PIR1) register layout

VII. Peripheral Interrupt Enable 1 (PIE1) Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SPPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **SPPIE:** Streaming Parallel Port Read/Write Interrupt Enable bit(1)
1 = Enables the SPP read/write interrupt
0 = Disables the SPP read/write interrupt

bit 6 **ADIE:** A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit 5 **RCIE:** EUSART Receive Interrupt Enable bit
1 = Enables the EUSART receive interrupt
0 = Disables the EUSART receive interrupt

bit 4 **TXIE:** EUSART Transmit Interrupt Enable bit
1 = Enables the EUSART transmit interrupt
0 = Disables the EUSART transmit interrupt

bit 3 **SSPIE:** Master Synchronous Serial Port Interrupt Enable bit
1 = Enables the MSSP interrupt
0 = Disables the MSSP interrupt

bit 2 **CCP1IE:** CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE:** TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE:** TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Note 1: This bit is reserved on 28-pin devices; always maintain this bit clear.

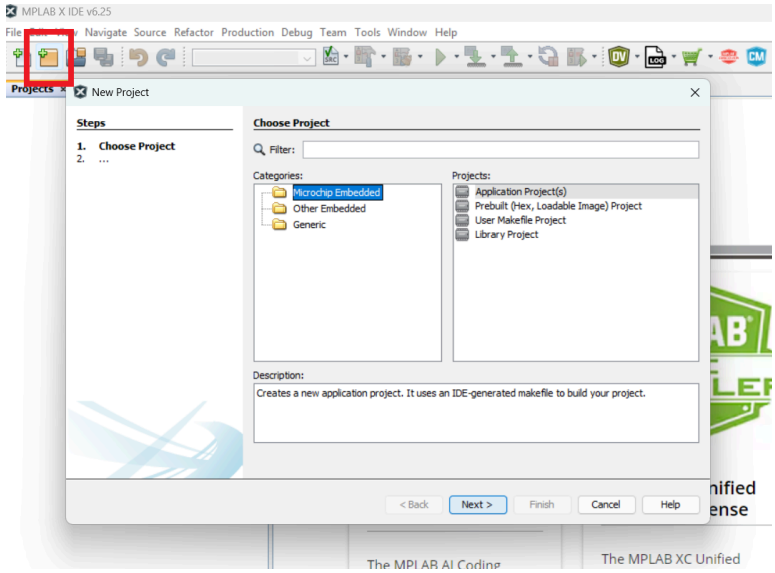
Figure 2.g: Peripheral Interrupt Enable 1 (PIE1) register layout

3. Getting Started with MPLAB X IDE (by Microchip)

In this section, the process of creating and building a project in MPLAB X IDE and testing the code in Proteus and hardware is discussed in detail. The following coding methods were used and are regarded as safe and practical coding habits:

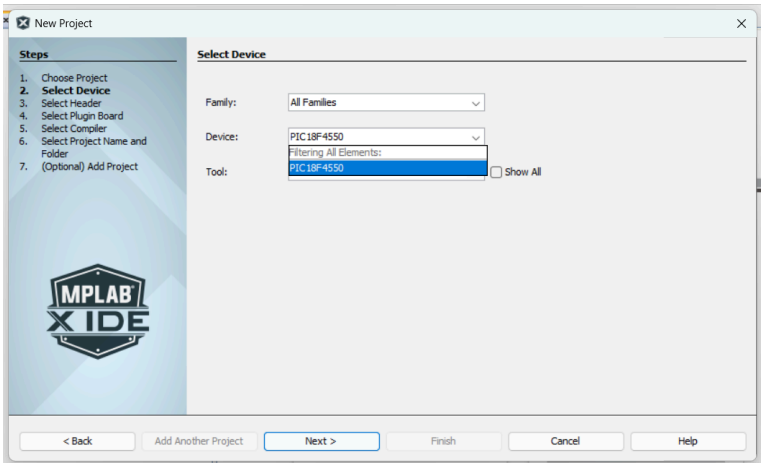
I. Creating a project and building it

Step 1: New Project creation:



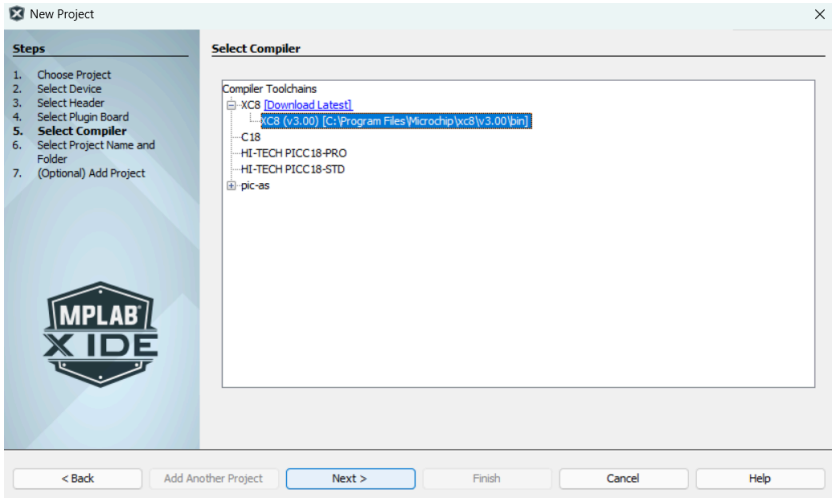
`Microchip Embedded > Application Project(s) > Next`

Step 2: Select device families:



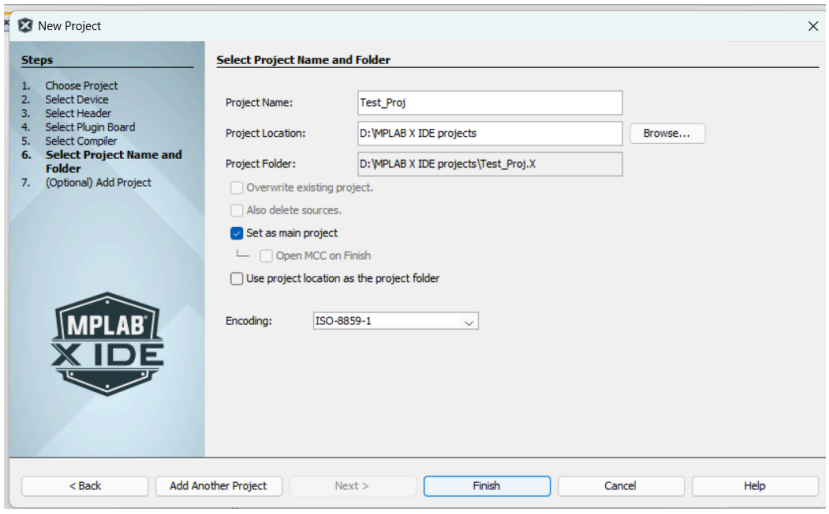
`The device is chosen as PIC18F4550 > Next (in Tool keep simulator)`

Step 3: Select Compiler:



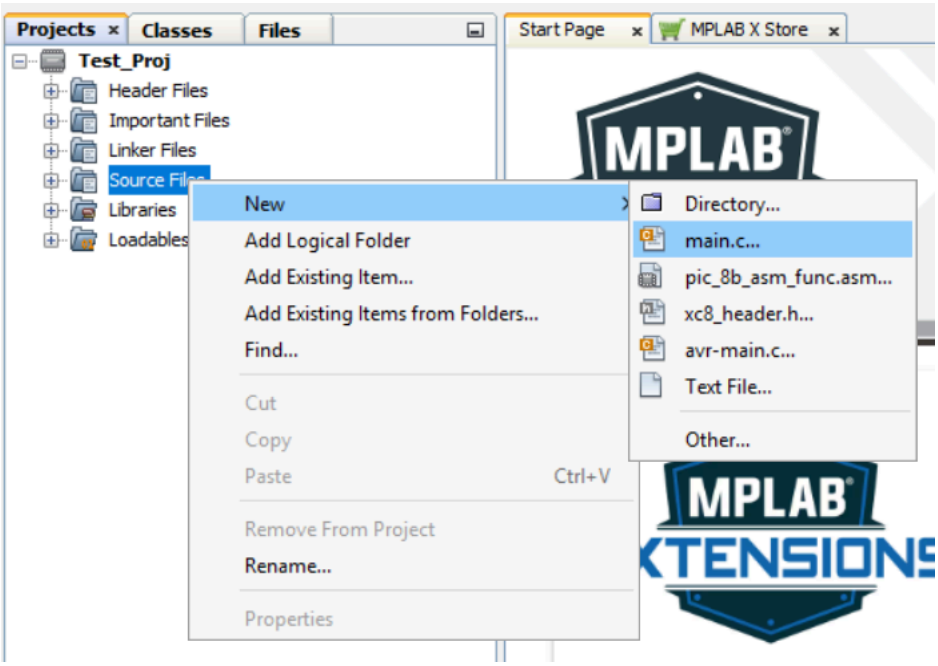
XC8 chosen as compiler (auto selects based on data bus width of the device) > Next

Step 4: Select project name:



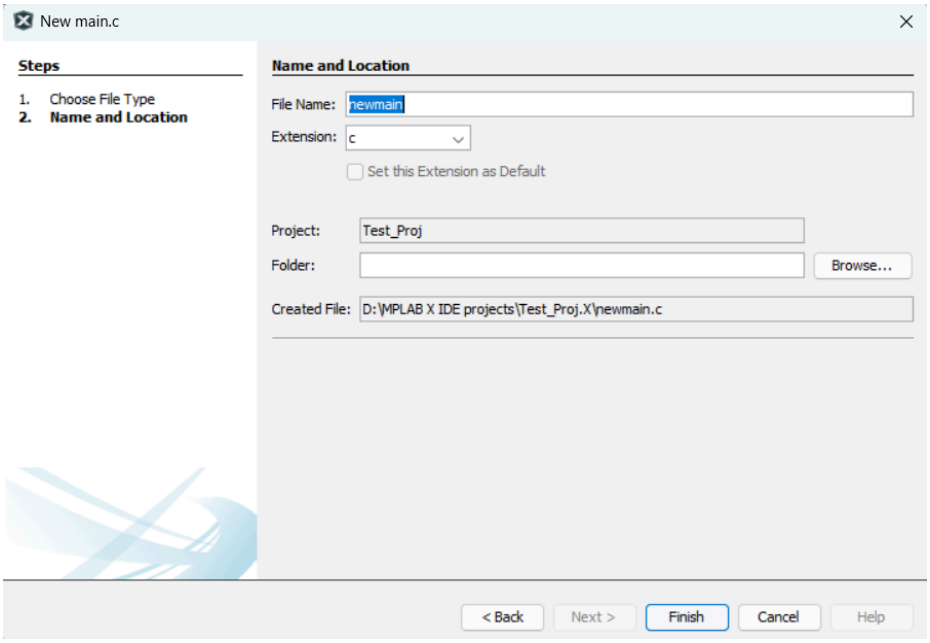
Project name and file directory location > Finish

Step 5: Create the .c file:



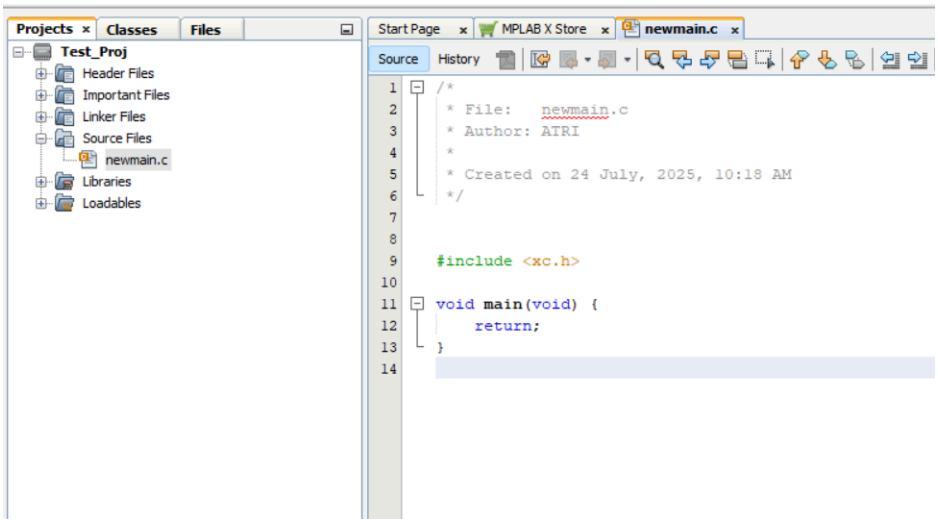
`right click Source Files > New > main.c`

Step 6: Name the .c file:



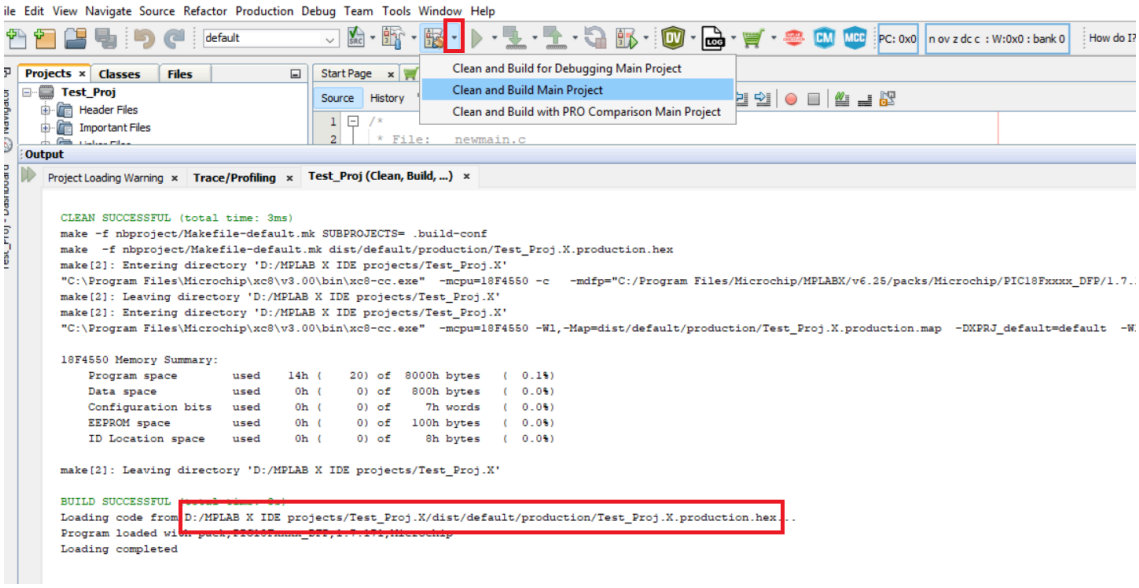
`Name of the file > select directory > Finish`

Step 7: Write the code:



The c code is written in the space provided

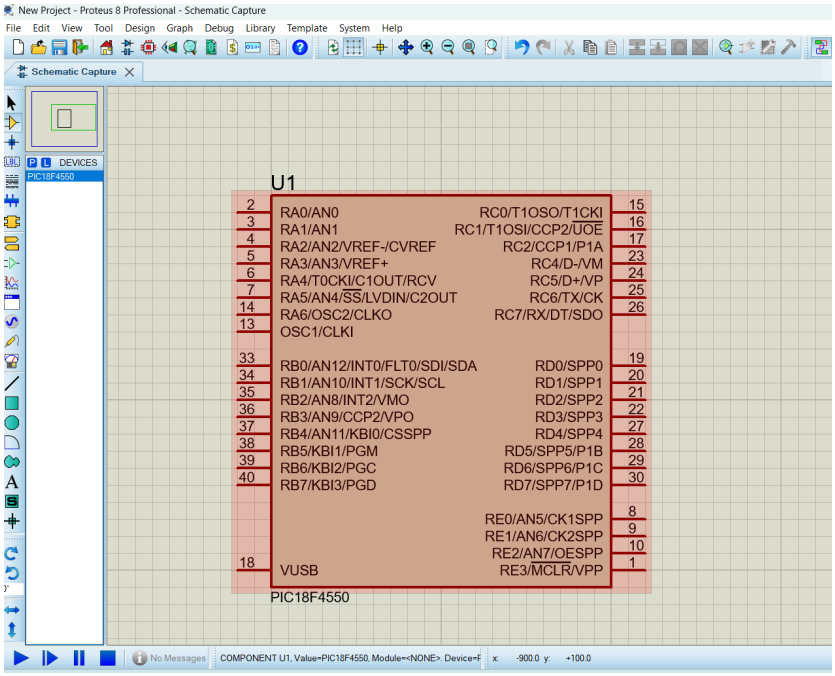
Step 8: Clean & Build project:



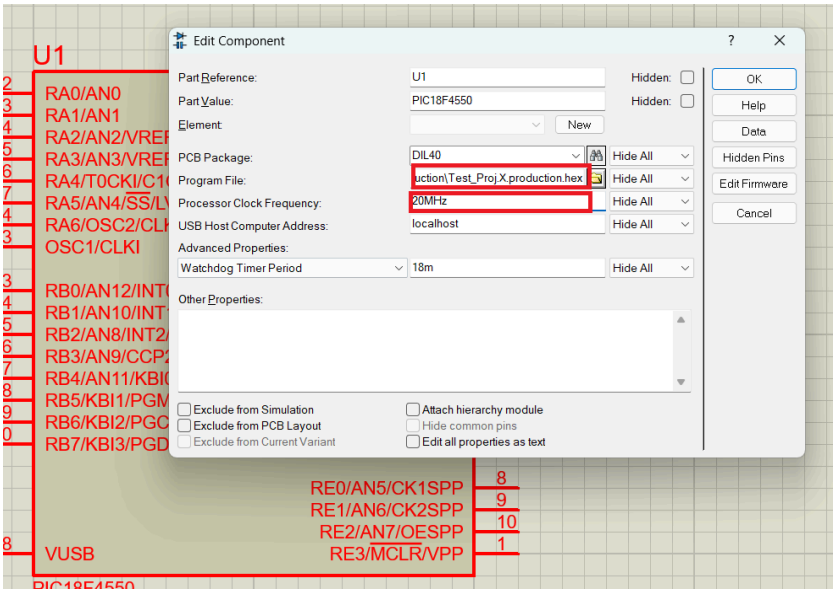
Click ‘Clean and Build Main Project’ > get the .hex file directory

II. Testing the code in Proteus

Step 1: Open Schematic in Proteus and select the PIC device:



Step 2: Double click on device to open edit component:



`Set Program File to .hex file directory > Set Processor Clock Frequency to 20MHz (as per board) > Make the circuit and simulate!!`

III. Hardware Simulation

All PIC development boards have a separate programmer requirement. At the Advanced Power Electronics Lab, IIEST Shibpur, I have worked with the available PICkit2 programmer for programming and debugging PIC microcontrollers.

For using PICkit2, one has to install the PICkit2 debugger application and upload the .hex file.

P.S. Due to the unavailability of the PICkit2 programmer during report preparation, I regretfully could not include its operational documentation.

For complete verification, one needs to sit with the development board, the programmer, and an oscilloscope, and keep coding!!

4. Code Snippets and Explanations

Here is a list of 11 basic to advanced Embedded C programs which were broadly simulated and tested on the PIC18F4550 development board in order to finally achieve SPWM gate pulses with V/f slow start control logic.

I. Led blink using __delay_ms() or __delay_us()

__delay_ms() & __delay_us() measures accurate time intervals in milliseconds and microseconds respectively. As a cautionary measure ensure the code declares the same _XTAL_FREQ (crystal oscillator frequency) as is there on the development board.

Code:

```
#pragma config FOSC = HS        // high-speed oscillator
#pragma config LVP = OFF       // Disable Low-Voltage Programming
#pragma config WDT = OFF     // Disable Watchdog Timer
#include <xc.h>
#define _XTAL_FREQ 20000000
void main(void)
{
    TRISBbits.TRISB0 =0;     //setting B0 as output
    while(1)
    {
        LATBbits.LATB0 ^= 1;
        __delay_ms(500);          //setting delay of 500ms
    }
    return;
}
```

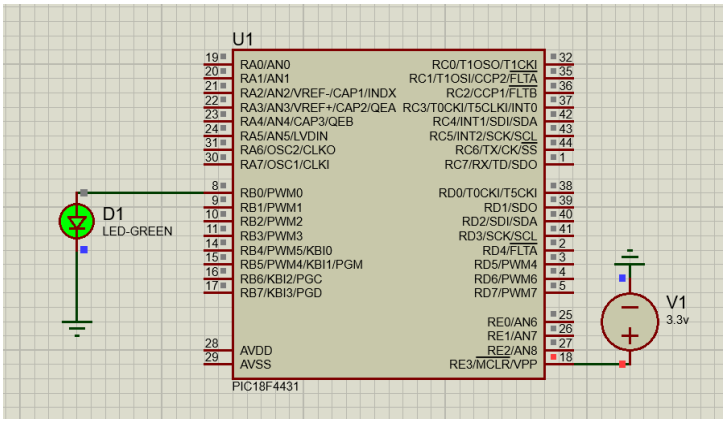


Figure 4.a: Proteus circuit for all LED blinking codes

II. Implementation of Timer0 (8-bit) ISR : Generating 10ms delay

Timer0 can be used both as an 8-bit timer (from 0 to 256 bits) or as a 16-bit timer (from 0 to 65536 bits). Count starting point is referred to as TMR0 (i.e., from TMR0 to 256 bits for 8-bit mode and to 65536 bits for 16 bit mode). Refer Figure 4.a for circuit diagram

Code:

```
#pragma config FOSC = HS        // high-speed oscillator
#pragma config LVP = OFF       // Disable Low-Voltage Programming
#pragma config WDT = OFF     // Disable Watchdog Timer

#include <xc.h>
#define _XTAL_FREQ 20000000     //Fosc

volatile unsigned int t1_overflow =0; //global variable

void timer0_initialise()
{
  T0CON = 0b11000111;     //refer Timer0 bit layout; preloader
(1:256), timer = 8 bits
  TMR0 = 60;              //timer count initialiser
  INTCONbits.TMR0IF =0;   //timer0 flag set to 0
  INTCONbits.TMR0IE =1;
  INTCONbits.GIE =1;
  INTCONbits.PEIE=1;    //peripheral interrupts enabled
}

void main(void)
{
  TRISBbits.TRISB0 =0;     //set RB0 to output
  LATBbits.LATB0=0;        //latch 0 to RB0

  timer0_initialise();


  while(1)
  {
      if (t1_overflow >= 1)     //delay count
```

```
        {
            LATBbits.LATB0 ^= 1;
            TMR0 = 60;            //reset
            t1_overflow =0;       //reset
        }
    }
    // delay = [(256 - TMR0) x preloader]/[Fosc/4]  secs
}
void __interrupt() timer0_isr(void) //ISR
{
    if(INTCONbits.TMR0IF == 1)
    {
         TMR0 = 60;            //Reset timer count initialiser
         INTCONbits.TMR0IF =0;//Reset timer0 flag set to 0
         t1_overflow++;
    }
}
```

Explanation:

Bits	Status	Function
TMR0ON	1	Timer0 ON
T08BIT	1	1 = 8-bit mode
T0CS	0	0 = Clock source is internal clock
T0SE	X	–
PSA	0	0 = Prescaler mode ON
T0PS2	1	111 = Prescaler is 1:256
T0PS1	1	
T0PS0	1	

Calculation:

$$\text{Internal Clock Frequency} = F_{osc} = 20MHz$$

$$\text{Instruction Clock Frequency} = \frac{F_{osc}}{4} = 5MHz$$

$$\Rightarrow \text{Instruction Clock Period} = \frac{1}{5}\mu s = 0.2\mu s = \text{Time required to execute 1 instruction}$$

$$\text{after prescaling (1:256), Timer Clock Period} = 0.2\mu s \times 256 = 51.2\mu s$$

$$\Rightarrow \text{one 'tick' of timer clock} = 51.2\mu s$$

$$\text{Required delay: 10 ms}$$

$$\Rightarrow \text{Required number of 'ticks'} = \left\lceil \frac{(10ms \times 10^3)\mu s}{51.2\mu s} \right\rceil = \lceil 195.31 \rceil = 196$$

$$\text{for 8-bit timer, maximum count before timer flag becomes high} = 2^8 = 256$$

$$\Rightarrow \text{timer count initialiser (TMR0)} = 256 - 196 = 60 \text{ (in decimal system)}$$

$$\text{So, generalised delay formula:}$$

$$\text{delay} = [2^{(\text{timer bits})} - (\text{TMR0})_{10}] \times \frac{\text{preloader}}{\frac{F_{osc}}{4}} \text{ s}$$

III. Implementation of Timer0 (16-bit) ISR : Generating 1s delay

Timer0 in 16 bit mode counts from TMR0 to 65536. This is used for bigger intervals of delay.
Refer Figure 4.a for circuit diagram

Code:

```
#pragma config FOSC = HS        // high-speed oscillator
#pragma config LVP = OFF       // Disable Low-Voltage Programming
#pragma config WDT = OFF     // Disable Watchdog Timer

#include <xc.h>
#define _XTAL_FREQ 20000000     //Fosc

volatile unsigned int t1_overflow=0; //global variable


void timer0_initialise()
{
  T0CON = 0b10000111;     //refer Timer0 bit layout; preloader
(1:256), timer = 16 bits
  TMR0H = 0xB3;             //timer count initialiser; TMR0 = 46004
(decimal)
  TMR0L = 0xB4;
  INTCONbits.TMR0IF =0;  //timer0 flag set to 0
  INTCONbits.TMR0IE =1;
  INTCONbits.GIE =1;    //interrupt enable
  INTCONbits.PEIE=1;    //peripheral interrupts enabled
}

void main(void)
{
  TRISBbits.TRISB0 =0;     //set RB0 to output
  LATBbits.LATB0=0;        //latch 0 to RB0
  timer0_initialise();
```

```
  while(1)          // delay = [(65536 - TMR0) x preloader]/[Fosc/4]
secs
  {
      if(t1_overflow >= 1) //delay count measure
      {
          LATBbits.LATB0 ^= 1;
          t1_overflow =0;
      }
  }
}

void __interrupt() timer0_isr(void)      //ISR
{
  if(INTCONbits.TMR0IF == 1)
  {
       TMR0H = 0xB3;              //timer count initialiser using hex
numbers
       TMR0L = 0xB4;
       INTCONbits.TMR0IF =0;//Reset timer0 flag set to 0
       t1_overflow++;
  }
}
```

Explanation:

Bits	Status	Function
TMR0ON	1	Timer0 ON
T08BIT	0	0 = 16-bit mode
T0CS	0	0 = Clock source is internal clock
T0SE	X	–
PSA	0	0 = Prescaler mode ON
T0PS2	1	111 = Prescaler is 1:256
T0PS1	1	
T0PS0	1	

Calculations:

$$\text{Internal Clock Frequency} = F_{osc} = 20MHz$$

$$\text{Instruction Clock Frequency} = \frac{F_{osc}}{4} = 5MHz$$

$$\Rightarrow \text{Instruction Clock Period} = \frac{1}{5}\mu s = 0.2\mu s = \text{Time required to execute 1 instruction}$$

$$\text{after prescaling (1:256), Timer Clock Period} = 0.2\mu s \times 256 = 51.2\mu s$$

$$\Rightarrow \text{one 'tick' of timer clock} = 51.2\mu s$$

$$\text{Required delay: 1 s}$$

$$\Rightarrow \text{Required number of 'ticks'} = \left\lceil \frac{(1s \times 10^6)\mu s}{51.2\mu s} \right\rceil = \lceil 19531.25 \rceil = 19532$$

$$\text{for 8-bit timer, maximum count before timer flag becomes high} = 2^{16} = 65536$$

$$\Rightarrow \text{timer count initialiser (TMR0)} = 65536 - 19532$$

$$= 46004 \text{ (in decimal system)} = B3B4 \text{(in hex)}$$

$$\text{So, generalised delay formula:}$$

$$\text{delay} = [2^{(\text{timer bits})} - (\text{TMR0})_{10}] \times \frac{\text{preloader}}{\frac{F_{osc}}{4}} \text{ s}$$

IV. Implementation of Timer1 (16-bit) ISR : Generating 100 ms delay

Timer1 is strictly a 16 bit timer. Timer1 can access 16-bit data either as a single 16-bit block or as two separate 8-bit registers depending on the value of RD16.
Refer Figure 4.a for circuit diagram

Code:

```
#pragma config FOSC = HS        // high-speed oscillator
#pragma config LVP = OFF       // Disable Low-Voltage Programming
#pragma config WDT = OFF     // Disable Watchdog Timer

#include <xc.h>
#define _XTAL_FREQ 20000000     //Fosc

volatile unsigned int t1_overflow=0; //global variable


void timer1_initialise()
{
  T1CON = 0b00110001;     //refer Timer1 bit layout; preloader (1:8),
timer = 16 bits
  TMR1H = 0x0B;              //timer count initialiser; TMR1 = 3036
(decimal)
  TMR1L = 0xDC;
  PIR1bits.TMR1IF =0;   //timer1 flag set to 0
  PIE1bits.TMR1IE =1;   // timer1 interrupts enabled
  INTCONbits.GIE =1;    //global interrupts enabled
  INTCONbits.PEIE=1;    //peripheral interrupts enabled
}



void main(void)
{
  TRISBbits.TRISB0 =0;     //set RB0 to output
  LATBbits.LATB0=0;        //latch 0 to RB0
  timer1_initialise();
```

```
  while(1)          // delay = [(65536 - TMR1) x preloader]/[Fosc/4]
secs
  {
      if(t1_overflow >= 1) //delay count measure
      {
          LATBbits.LATB0 ^= 1;
          t1_overflow =0;
      }
  }
}

void __interrupt() timer1_isr(void)      //ISR
{
  if(PIR1bits.TMR1IF == 1)
  {
       TMR1H = 0x0B;              //timer count initialiser using hex
numbers
       TMR1L = 0xDC;
       PIR1bits.TMR1IF =0;//Reset timer0 flag set to 0
       t1_overflow++;
  }
}
```

Explanation:

Bits	Status	Function
RD16	0	16-bit accessible as 2 8-bit operations
-	0	x
T1CKPS1	1	Prescaler value: 1:8
T1CKPS0	1	
T1OSCEN	0	External oscillator OFF
T1SYNC	0	Unnecessary as T1OSCEN = 0
TMR1CS	0	Internal clock ON (freq. = Fosc/4)
TMR1ON	1	Enable Timer1

Calculations:

$$\text{Internal Clock Frequency} = F_{osc} = 20MHz$$

$$\text{Instruction Clock Frequency} = \frac{F_{osc}}{4} = 5MHz$$

$$\Rightarrow \text{Instruction Clock Period} = \frac{1}{5}\mu s = 0.2\mu s = \text{Time required to execute 1 instruction}$$

$$\text{after prescaling (1:8), Timer Clock Period} = 0.2\mu s \times 8 = 1.6\mu s$$
$$\Rightarrow \text{one 'tick' of timer clock} = 1.6\mu s$$

$$\text{Required delay: 100 ms}$$

$$\Rightarrow \text{Required number of 'ticks'} = \left\lceil \frac{(100ms \times 10^3)\mu s}{1.6\mu s} \right\rceil = \lceil 62500 \rceil = 62500$$

$$\text{for 8-bit timer, maximum count before timer flag becomes high} = 2^{16} = 65536$$
$$\Rightarrow \text{timer count initialiser (TMR0)} = 65536 - 62500$$
$$= 3036 \text{ (in decimal system)} = 0BDC(\text{in hex})$$

$$\text{So, generalised delay formula:}$$

$$\text{delay} = [2^{(\text{timer bits})} - (\text{TMR0})_{10}] \times \frac{\text{preloader}}{\frac{F_{osc}}{4}} \text{ s}$$

V. Implementation of Timer2 (8-bit) ISR : Generating 1 ms delay

Timer2 is strictly an 8 bit timer. This is the only timer which allows postscaling if the required number of Timer2 cycles goes beyond the 8-bit limit of 255 bits. This timer is used with CCP1 for PWM generation which is discussed later.
Refer Figure 4.a for circuit diagram

Code:

```
#pragma config FOSC = HS        // high-speed oscillator
#pragma config LVP = OFF       // Disable Low-Voltage Programming
#pragma config WDT = OFF     // Disable Watchdog Timer

#include <xc.h>
#define _XTAL_FREQ 20000000     //Fosc

volatile unsigned int t1_overflow=0; //global variable


void timer2_initialise()
{
  T2CON = 0b00001110;     //refer Timer2 bit layout; prescaler
(1:16), postscaler (1:2)
  PR2 = 157;              //Period Register 2; value at which Timer2
resets
  PIR1bits.TMR2IF =0;   //timer1 flag set to 0
  PIE1bits.TMR2IE =1;   // timer1 interrupts enabled
  INTCONbits.GIE =1;    //global interrupts enabled
  INTCONbits.PEIE=1;    //peripheral interrupts enabled
}


void main(void)
{
  TRISBbits.TRISB0 =0;     //set RB0 to output
  LATBbits.LATB0=0;        //latch 0 to RB0
  timer2_initialise();
```

```
  while(1)
{
      if(t1_overflow >= 1) //delay count measure
      {
          LATBbits.LATB0 ^= 1;
          t1_overflow =0;
      }
  }
}


void __interrupt() timer2_isr(void)     //ISR
{
  if(PIR1bits.TMR2IF == 1)
  {
       PIR1bits.TMR2IF =0;//Reset timer0 flag set to 0
       t1_overflow++;
  }
}
```

Explanations:

Bits	Status	Function
X	0	X
TOUTPS3	0	Post-scaling (1:2)
TOUTPS2	0	
TOUTPS1	0	
TOUTPS0	1	
TMR2ON	1	Timer2 ON
T2CKPS1	1	Pre-scaling (1:16)
T2CKPS0	0	

Calculations:

$$\text{Internal Clock Frequency} = F_{osc} = 20MHz$$

$$\text{Instruction Clock Frequency} = \frac{F_{osc}}{4} = 5MHz$$

$$\Rightarrow \text{Instruction Clock Period} = \frac{1}{5}\mu s = 0.2\mu s = \text{Time required to execute 1 instruction}$$

$$\text{after prescaling (1:16), Timer Clock Period} = 0.2\mu s \times 16 = 3.2\mu s$$

$$\Rightarrow \text{one 'tick' of timer clock} = 3.2\mu s$$

$$\text{Required delay: 1 ms}$$

$$\Rightarrow \text{Required number of 'ticks'} = \left\lceil \frac{(1ms \times 10^3)\mu s}{3.2\mu s} \right\rceil = \lceil 312.5 \rceil = 313$$

$$\text{However, for 8-bit timer2, range of 'ticks' is 0 - 255}$$

$$\Rightarrow \text{after postscaling (1:2), number of 'ticks'} = \left\lceil \frac{313}{2} \right\rceil = \lceil 156.5 \rceil = 157$$

$$\Rightarrow \text{PR2} = 157$$

VI. LED switching code (Trivial practice code)

Here the PIC18F4550 is used as a switch. This code is trivial but it is necessary to build a toggle switch. This exercise is important for developing ON/OFF logic for the controller.

Code:

```
#pragma config FOSC = HS        // high-speed oscillator
#pragma config LVP = OFF       // Disable Low-Voltage Programming
#pragma config WDT = OFF     // Disable Watchdog Timer

#include <xc.h>
#define _XTAL_FREQ 8000000
void main(void)
{
    TRISBbits.TRISB0 = 0; //output pin
    TRISBbits.TRISB1 = 1; //input pin

    while(1)
    {
        if(PORTBbits.RB1 == 1)   //if switch is ON
            LATBbits.LATB0 = 1;
        else
            LATBbits.LATB0 = 0;
    }
    return;
}
```

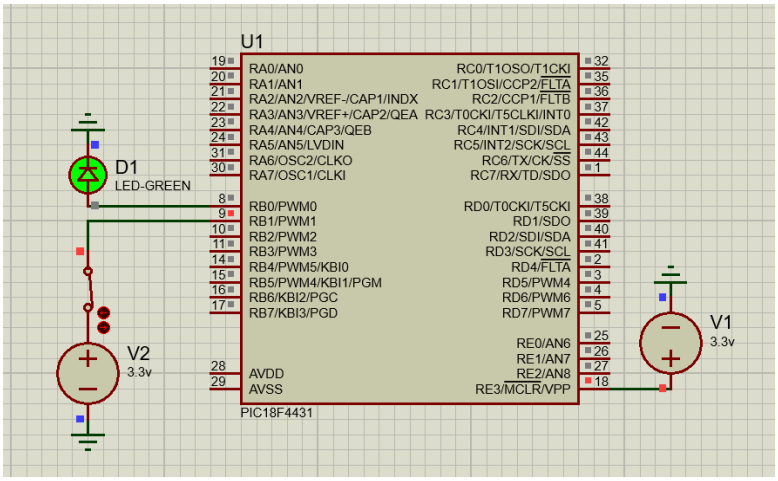


Figure 4.b: Proteus circuit diagram for LED switching

VII. Fixed Duty (50%) 50kHz PWM Generation: using CCP1 and Timer2

Here is a brief background regarding PWM generation using Timer2 and CCP1.

How does CCP1 and Timer2 based PWM generation work?

- Timer2 Generates the PWM Time Base. Timer2 is configured to run in 8-bit mode with a prescaler and PR2 register defining the PWM period.

- The PWM frequency is set by:
  PWM Period = $(PR2+1) \times 4 \times T_{OSC} \times$ Prescaler

- CCP1 in PWM Mode Controls Duty Cycle. The CCP1 module is set to PWM mode by configuring the CCP1CON register (CCP1M3:CCP1M0 = 1100).

- The duty cycle is set using 10 bits:
  8 MSBs → CCPR1L
  2 LSBs → CCP1CON<5:4>

- PWM Output Logic: At every Timer2 overflow (TMR2 = PR2), the CCP1 output is set (high). When TMR2 matches the 10-bit duty cycle value, the output is cleared (low). This generates a pulse where high time = duty × period.

- The frequency remains constant (set by PR2), while duty cycle is varied by modifying CCPR1L and CCP1CON<5:4> in real time.

Code:

```
#pragma config OSC = HS        // high-speed oscillator
#pragma config LVP = OFF       // Disable Low-Voltage Programming
#pragma config WDTEN = OFF     // Disable Watchdog Timer


#include <xc.h>
#define _XTAL_FREQ 8000000     //Fosc
unsigned int P = 39;     //PR2 value
unsigned int pD = 19;    //percentage duty (D% of P)


void timer2_initialise(unsigned int n)
{
  TRISCbits.TRISC2 =0;     //set RC2 to output
  T2CON = 0b00000100;     //refer Timer2 bit layout; prescaler (1:1)
  CCP1CON = 0b00001100;   //PWM mode
  PR2 = P;                //Period Register 2; value at which Timer2
resets
  CCPR1L = n;
  PIR1bits.TMR2IF =0;  //timer1 flag set to 0
  PIE1bits.TMR2IE =1;  // timer1 interrupts enabled
  INTCONbits.GIE =1;   //global interrupts enabled
  INTCONbits.PEIE=1;   //peripheral interrupts enabled
}


void main(void)
{
  LATCbits.LATC2=0;        //latch 0 to RC2
  while(1)
  {
     timer2_initialise(pD);
  }
}
void __interrupt() timer2_isr(void)      //ISR
{
  if(PIR1bits.TMR2IF == 1)
       PIR1bits.TMR2IF =0; //Reset timer0 flag set to 0
}
```

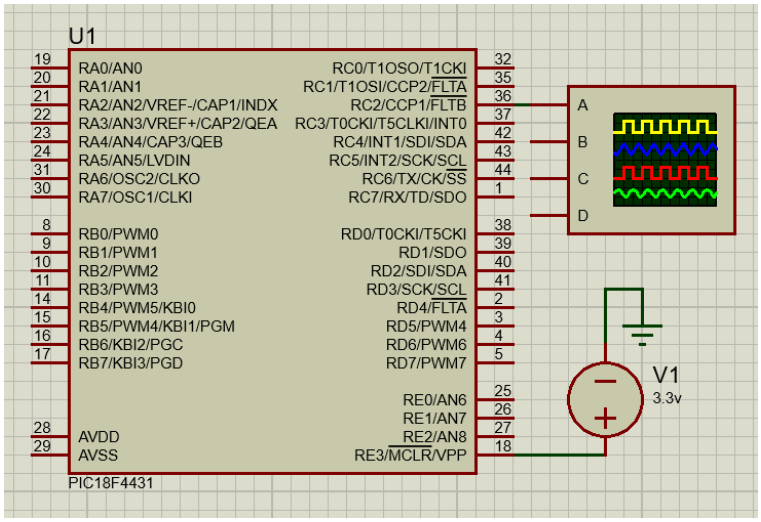


Figure 4.c: Proteus circuit diagram for PWM generation

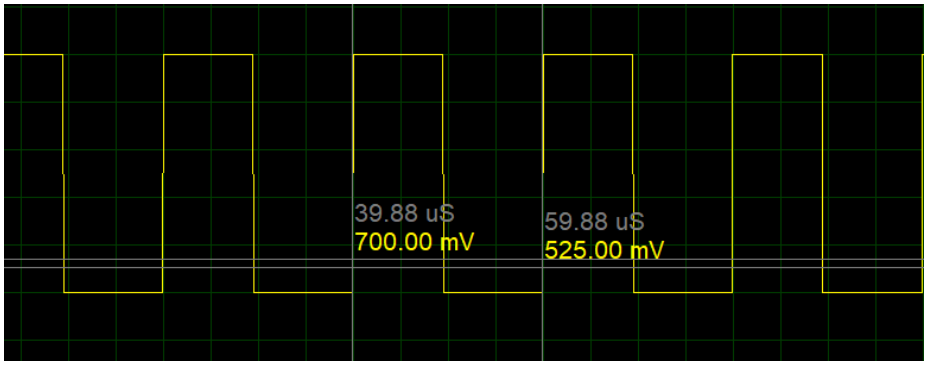


Figure 4.d: Proteus simulation DSO output for PWM generation

VIII. Variable duty 50kHz PWM Generation: using CCP1 and Timer2

Unlike the last code, this code generates variable duty ratios which vary continuously with time. This was the first attempt towards SPWM programming which turned out to be a failure.

Code:

```
#pragma config OSC = HS        // high-speed oscillator
#pragma config LVP = OFF       // Disable Low-Voltage Programming
#pragma config WDTEN = OFF     // Disable Watchdog Timer

#include <xc.h>
#define _XTAL_FREQ 8000000     //Fosc
unsigned int P = 39;           //PR2 value initializer

void timer2_initialise(unsigned int n)
{
  TRISCbits.TRISC2 =0;     //set RB0 to output
  T2CON = 0b00000100;     //refer Timer2 bit layout; prescaler (1:1)
  CCP1CON = 0b00001100;
  PR2 = P;                //Period Register 2; value at which Timer2
resets
  CCPR1L = n;
  PIR1bits.TMR2IF =0;  //timer1 flag set to 0
  PIE1bits.TMR2IE =1;  // timer1 interrupts enabled
  INTCONbits.GIE =1;   //global interrupts enabled
  INTCONbits.PEIE=1;   //peripheral interrupts enabled
}

void main(void)
{
  unsigned int n;
  LATCbits.LATC2=0;        //latch 0 to RC2: CCP1 pin
  while(1)
  {
       for(n=0; n< P; n++)
       {
           timer2_initialise(n);

       }
```

```
        for(n=P; n>0; n--)
        {
            timer2_initialise(n);
        }
    }
}

void __interrupt() timer2_isr(void)      //ISR
{
  if(PIR1bits.TMR2IF == 1)
        PIR1bits.TMR2IF =0; //Reset timer0 flag set to 0
}
```

NOTE: This is not an SPWM as all duties vary simultaneously. Taking a waveform snapshot from this experiment is quite challenging.

IX. 50Hz Triangular wave generation (using DAC MCP4921 & PIC18F4431)

Here is a brief background regarding SPI communication. This experiment was solely done on software because somehow the MCP4921 DACs couldn't be configured. Nevertheless, this exercise gives a brilliant insight into SPI communication.

SPI-Based Communication

- MCP4921 communicates via SPI protocol (Serial Peripheral Interface).
- It uses 3 main lines:
  - SCK (Serial Clock)
  - SDI or MOSI (Data Input)
  - CS or ~LDAC (Chip Select / Latch)

16-Bit Data Word Format

- A 16-bit control + data word must be sent:
  - Bits 15–12: Control bits (e.g., DAC channel, gain, shutdown)
  - Bits 11–0: 12-bit digital data to convert
- Example control bits: 0x3 for active mode, 1× gain, buffered input.

Timing and Write Sequence

- The CS pin must be pulled low to start communication.
- The 16 bits are sent to MSB first on the rising SCK edge.
- After transmission, toggle ~LDAC or configure it low to update the DAC output.

Voltage Output Depends on Reference

- The analog output VOUT = (D/4096) × VREF, where D is the 12-bit digital value.
- VREF is typically connected to a stable reference (e.g., 2.048 V or 5 V).
- The DAC output appears on the VOUT pin and can be filtered (RC low-pass) for waveform generation.

Registers used:

SSPSTAT (Status Register): Holds status flags for SPI data transmission, including buffer full (`BF`) and data sampling phase (`SMP`).
SSPCON1 (Control Register 1): Configures SPI mode, clock polarity (`CKP`), master/slave selection, and enables the SSP module.
SSPBUF (Buffer Register): A read/write buffer for SPI data; writing to it starts transmission, reading from it gives received data.

Code:

```
#pragma config OSC = HS
#pragma config WDTEN = OFF
#pragma config LVP = OFF

#include <xc.h>
#include <stdint.h>

#define _XTAL_FREQ 20000000UL

static const uint8_t triangle[100] =
{0,  5, 10, 15, 20, 25, 30, 36, 41, 46, 51, 56, 61,
66, 71, 76, 81, 86, 91, 97,102,107,112,117,122,127,
132,137,142,147,152,157,163,168,173,178,183,188,193,
198,203,208,213,218,224,229,234,239,244,249,254,249,
244,239,234,229,224,218,213,208,203,198,193,188,183,
178,173,168,163,157,152,147,142,137,132,127,122,117,
112,107,102, 97, 91, 86, 81, 76, 71, 66, 61, 56, 51,
46, 41, 36, 30, 25, 20, 15, 10,  5
};

volatile uint8_t index =0;

#define CS_DIR TRISBbits.TRISB0    //set RB0 to output
#define CS   LATBbits.LATB0     //latch 0 to RB0

void spi_init();
void dac_send(unsigned char val);
void spi_send(unsigned char data);
void timer2_init();


void spi_init()
{
   TRISCbits.TRISC5 = 0;  //SDO
   TRISCbits.TRISC7 = 0;  //SCK

   SSPSTAT = 0x40;
   SSPCON1 = 0x20;
}
```

```
void dac_send(unsigned char val)
{
   uint16_t word = 0x3000 | (val << 4);
   CS =0;
   spi_send(word >>8);
   spi_send(word & 0x00FF);
   CS =1;
}

void spi_send(unsigned char data)
{
   PIR1bits.SSPIF = 0;
   SSPBUF = data;
   while(!SSPSTATbits.BF);
}



void timer2_init()
{
   T2CON = 0b00000101;
   PR2 = 250;
   TMR2 = 0;
   PIE1bits.TMR2IE = 1;
   INTCONbits.PEIE = 1;
   INTCONbits.GIE  = 1;
   T2CONbits.TMR2ON = 1;
}

void main(void)
{
   CS_DIR= 0;
   CS=1;
   spi_init();
   timer2_init();
   while(1);
}
```

```
void __interrupt() ISR()
{
    if(PIR1bits.TMR2IF == 1)
    {
        PIR1bits.TMR2IF =0;
        dac_send (triangle[index]);
        index = (index + 1) % 100;
    }
}
```

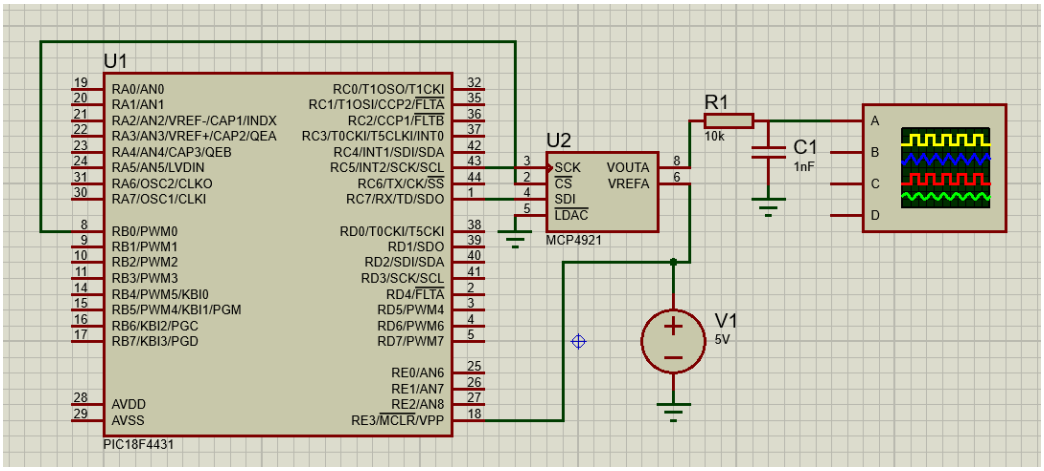


Figure 4.e: Triangular wave generation Proteus circuit

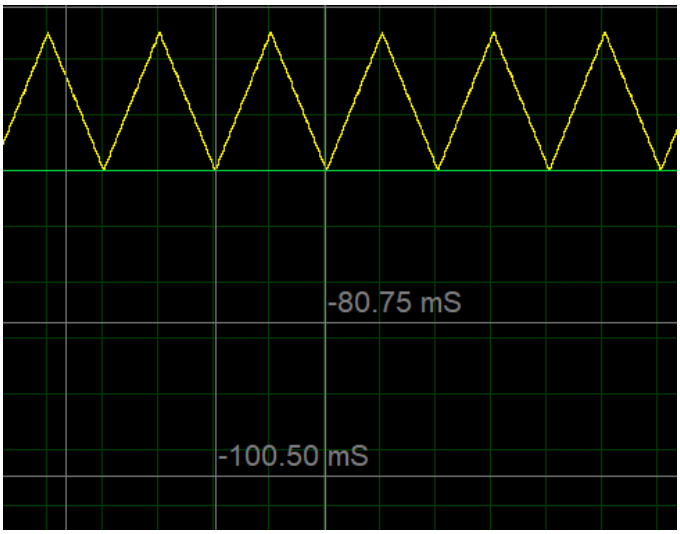


Figure 4.f: 50Hz triangular wave in Proteus

X. 50Hz triangular wave & sine wave completely in phase (using DAC MCP4921 & PIC18F4431)

This is the next SPI communication experiment where a 50 Hz triangular wave and a 50 Hz sine wave are to be obtained which are completely in phase. The objective of this experiment was to increase the frequency of triangular wave to 5kHz and compare the individual values of sine and triangle to obtain SPWM. However due to some unknown error 5kHz triangular wave was never obtained. Nevertheless a good knowledge of LUTs was obtained.

Code

```
#pragma config OSC = HS
#pragma config WDTEN = OFF
#pragma config LVP = OFF

#include <xc.h>
#include <stdint.h>

#define _XTAL_FREQ 20000000UL

static const uint8_t triangle[100] =
{0,  5, 10, 15, 20, 25, 30, 36, 41, 46, 51, 56, 61,
66, 71, 76, 81, 86, 91, 97,102,107,112,117,122,127,
132,137,142,147,152,157,163,168,173,178,183,188,193,
198,203,208,213,218,224,229,234,239,244,249,254,249,
244,239,234,229,224,218,213,208,203,198,193,188,183,
178,173,168,163,157,152,147,142,137,132,127,122,117,
112,107,102, 97, 91, 86, 81, 76, 71, 66, 61, 56, 51,
46, 41, 36, 30, 25, 20, 15, 10,  5
};
static const uint8_t sine[100] =
{127,135,143,151,159,166,174,181,188,195,202,208,214,
220,225,230,234,238,242,245,248,250,252,253,254,254,
254,253,252,250,248,245,242,238,234,230,225,220,214,
208,202,195,188,181,174,166,159,151,143,135,127,119,
111,103, 95, 88, 80, 73, 66, 59, 52, 46, 40, 34, 29,
24, 20, 16, 12,  9,  6,  4,  2,  1,  0,  0,  0,  1,
 2,  4,  6,  9, 12, 16, 20, 24, 29, 34, 40, 46, 52,
59, 66, 73, 80, 88, 95,103,111,119};
```

```
volatile uint8_t index1 =0;
volatile uint8_t index2 =75;

#define CS_DIR_1 TRISEbits.TRISE0    //set RB0 to output
#define CS_1   LATEbits.LATE0     //latch 0 to RB0
#define CS_DIR_2 TRISEbits.TRISE1    //set RB1 to output
#define CS_2   LATEbits.LATE1     //latch 0 to RB1

void spi_init();
void dac_send1(unsigned char val);
void dac_send2(unsigned char val);
void spi_send(unsigned char data);
void timer2_init();

void spi_init()
{
   TRISCbits.TRISC5 = 0;  //SDO
   TRISCbits.TRISC7 = 0;  //SCK

   SSPSTAT = 0x40;
   SSPCON1 = 0x20;
}

void dac_send1(unsigned char val)
{
   uint16_t word = 0x3000 | (val << 4);
   CS_1 =0;
   spi_send(word >>8);
   spi_send(word & 0x00FF);
   CS_1 =1;
}

void dac_send2(unsigned char val)
{
   uint16_t word = 0x3000 | (val << 4);
   CS_2 =0;
   spi_send(word >>8);
   spi_send(word & 0x00FF);
   CS_2 =1;
}
```

```
void spi_send(unsigned char data)
{
   PIR1bits.SSPIF = 0;
   SSPBUF = data;
   while(!SSPSTATbits.BF);
}

void timer2_init()
{
   T2CON = 0b00000101;
   PR2 = 250;
   TMR2 = 0;
   PIE1bits.TMR2IE = 1;
   INTCONbits.PEIE = 1;
   INTCONbits.GIE  = 1;
   T2CONbits.TMR2ON = 1;
}

void main(void)
{
   CS_DIR_1 =0;
   CS_1     =1;
   CS_DIR_2 =0;
   CS_2     =1;
   spi_init();
   timer2_init();
   while(1);
}

void __interrupt() ISR()
{
   if(PIR1bits.TMR2IF == 1)
   {
       PIR1bits.TMR2IF =0;
       dac_send1 (triangle[index1]);
       dac_send2 (sine[index2]);
       index1 = (index1 + 1) % 100;      //intelligent!!
       index2 = (index2 + 1) % 100;
   }
}
```

C code for 100 point sine wave LUT generation (this is *not* Embedded C):

```
#include <stdio.h>
#include <math.h>

#define PI 3.14159

int main() {
    double t;

    for (float x = 0; x <= 360; x+=3.6) {
        double rad = x * (PI / 180.0);        // degrees to radians
        double y = 1 - sin(rad);              // still in range [0,2]
        t = (y / 2.0) * 100.0;                // scale to [0,100]
        printf("%.2f, ", t);                  // print with 2 decimal
places
    }

    return 0;
}
```

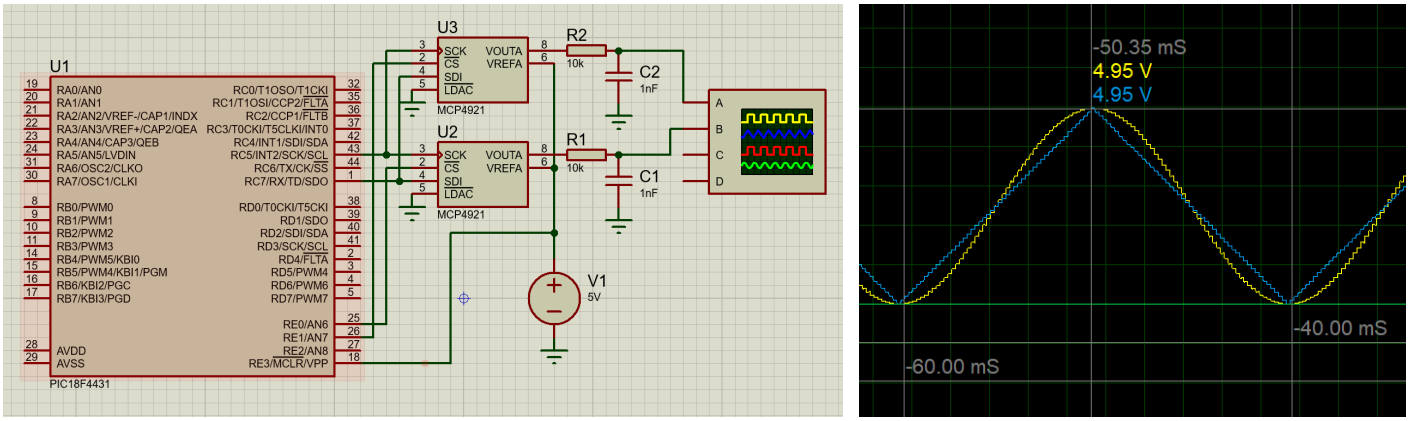


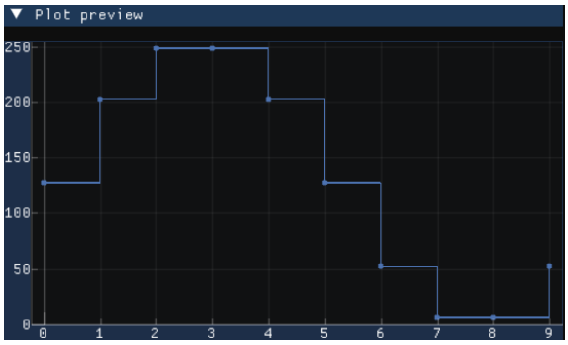
Figure 4.g: Proteus Circuit and output wave for in phase sine and triangle

XI. SPWM Generation for V/f control operation

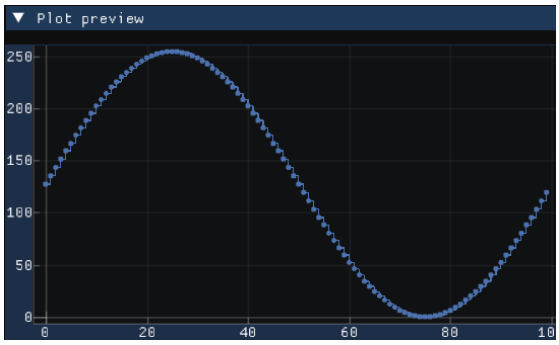
Concepts:

- We use a **1000 point 8-bit sine LUT** to describe the complete sine wave

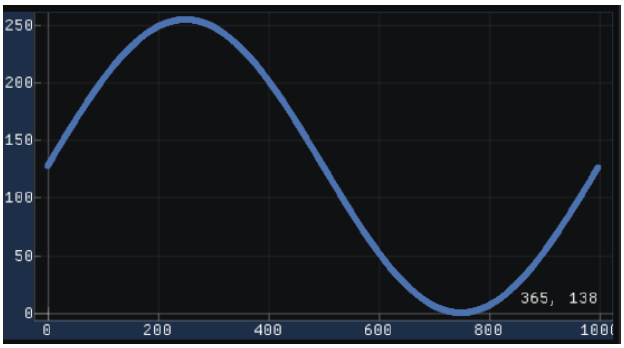
```
int sine[1000]= {127,...,255,...,127,...,0,...,127};
//This is a sine LUT. It's made using unsigned 8 bit numbers (from 0
to 255) centered around 127.
```



i. 10 point 8-bit sine wave.



ii. 100 point 8-bit sine wave.



iii. 1000 point 8-bit sine wave.

Figure 4.h: LUT description for variable point sine wave (from Dr. LUT website)

Hence, the sine wave to be compared is actually a quasi-sinusoidal waveform with 1000 tiny steps. Now we need to compare each such step with an 8 bit triangular wave (from 0 to 255 and back to 0).

- A software 8-bit compare logic which depicts a triangular wave was developed. Then we compare each value of the sine[] array (LUT) with the following software triangular wave:

```
for(j=0; j<1000; j+=F) //F = Sine Frequency count**
{
      for(i=0; i<256; i+=k) //here k is for frequency optimisation*
      {
            //SPWM logic
            NOP (); //minimum delay PIC18F4550 can offer***
      }
      for(i=255; i>=0; i-=k)//here k is for frequency optimisation*
      {
            //SPWM logic
            NOP (); //minimum delay PIC18F4550 can offer***
      }
}    //here, j= sine value & i= triangular value
```

SPWM Logic: set a desired pin high if and only if the sine value is greater than the corresponding triangular value. So, for one value of sine, the inner loops create a **dual edge modulated SPWM pulse**. This continues for all the 1000 points.

This is exactly how the sine wave and triangular wave should look like in theory:

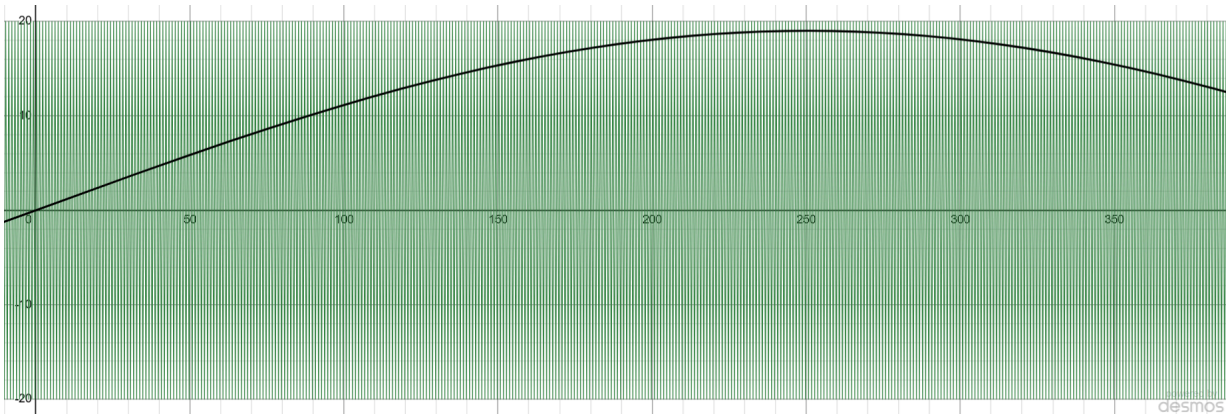
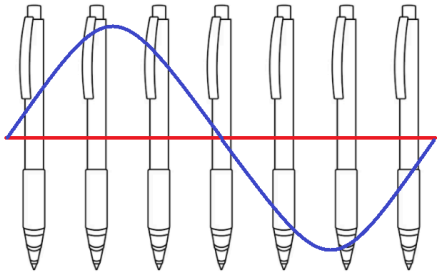


Figure 4.i: Actual conceptual sine and triangular wave compare realised in desmos

- Thus, the imaginary sine wave is actually sampled into:
  1000×(255+255) = 5,10,000 samples [255 steps up count, 255 steps down count]
  So, net delay for each sample to produce 50 Hz sine wave (time period = 20ms) is:
  $(\frac{20\text{ ms } \times 1000}{5,10,000})\mu s = 0.039\mu s$, which is much less than the instruction cycle of PIC18F4550 with a 20MHz crystal oscillator (= 0.2 μs).

  Hence, the solution is "**Bit skipping SPWM**" technique for frequency optimization and variation by keeping the delay fixed at `NOP()`*** (= 0.2 μs).
  [Refer code snippet in page 4.29]

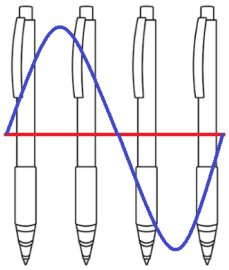
- Say there are 7 pens, each representing an SPWM pulse:



[0] [1] [2] [3] [4] [5] [6]

Figure 4,j.i: Sine wave represented by 7 pens

Now imagine pens (SPWM pulses) at index numbers 1, 3 & 5 are removed. Since all the pens are identical, we assume that the amplitude of the sine wave does not change. But the frequency changes!!
This is the concept of **Bit skipping SPWM.**



[0] [2] [4] [6]

Figure 4,j.ii: Sine wave represented by 4 pens

Now this analogy of pens is put in real time SPWMs. If we skip a few indexes in the sine LUT, then we can achieve frequency modulation**. This is also applicable for the triangular wave*. [Refer code snippet in page 4.29]

- **Amplitude Modulation**:

  Maximum value of sine in LUT for 8 bits is 255, centred around 127.
  So, Sine Amplitude = 255 - 127 = 128 bits.
  Now, if we reduce the Sine Amplitude, our conceptual sine wave will reduce its amplitude.

```
int scaled_sine = 127 + ((sine[i1] - 127)* mod_index)/100;

//sine[i1] is an index in 1000 point LUT. mod_index is the
modulation index in percentage form
i.e., if modulation index is 0.6, then mod_index = 60
```

  Thus, from 1 standard LUT, we can achieve variable amplitude in the sine wave. This saves MCU data memory.

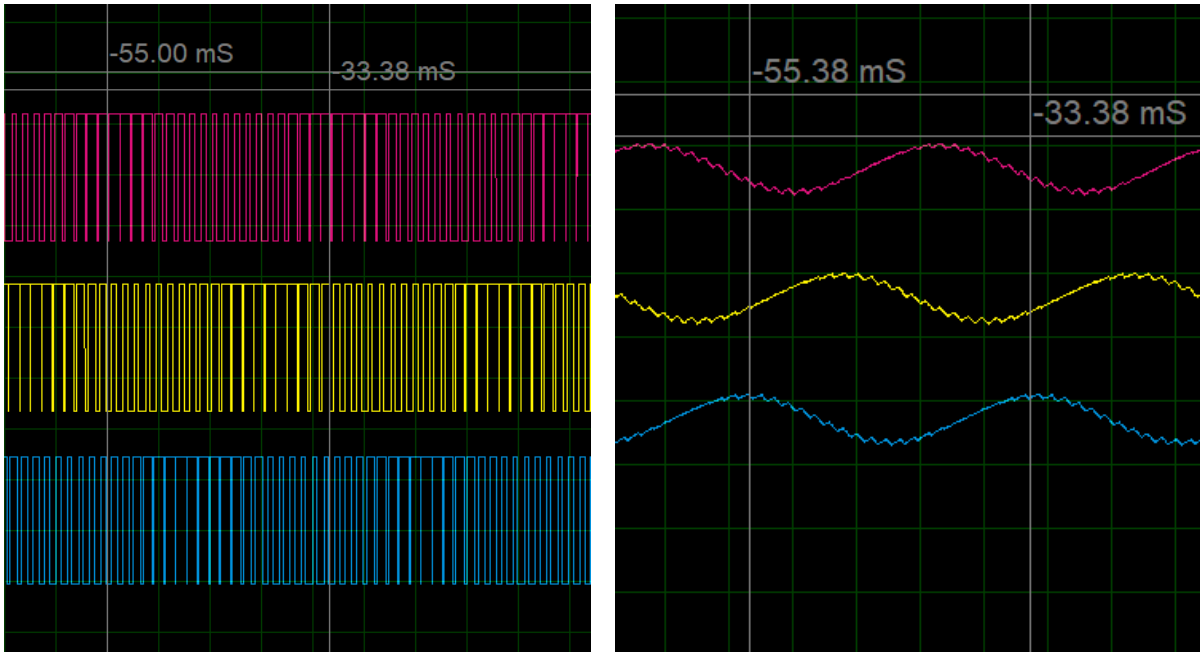
- **Slow start mechanism:**

  It was chosen that a slow start would take around 25 seconds. So, accordingly a V/f relation was chosen by establishing a relationship between `mod_index` (in percentage) and frequency (F).

  In the experiment, the chosen relationship was :
  `mod_index` (in percentage) `= 2 × F.`

  Since there are 50 possible frequencies for slow start, each frequency starting from 1 was provided a delay for 500 ms to achieve 25 s slow start. For doing that a Timer0 ISR was used.

- **Properties of SPWM achieved so far:**
  1. 1000 point sine LUT
  2. Dual edge modulated SPWM pulse
  3. Bit skipping SPWM
  4. Amplitude modulated SPWM
  5. Slow start 25 s ramp mechanism with 500 ms Timer0 ISR



I II

Figure 4.k: Proteus output of the V/f SPWM code after 25s. The SPWM is filtered via an R-C passive filter which causes a phase lag (~60°) in the sine wave at -33.38 ms

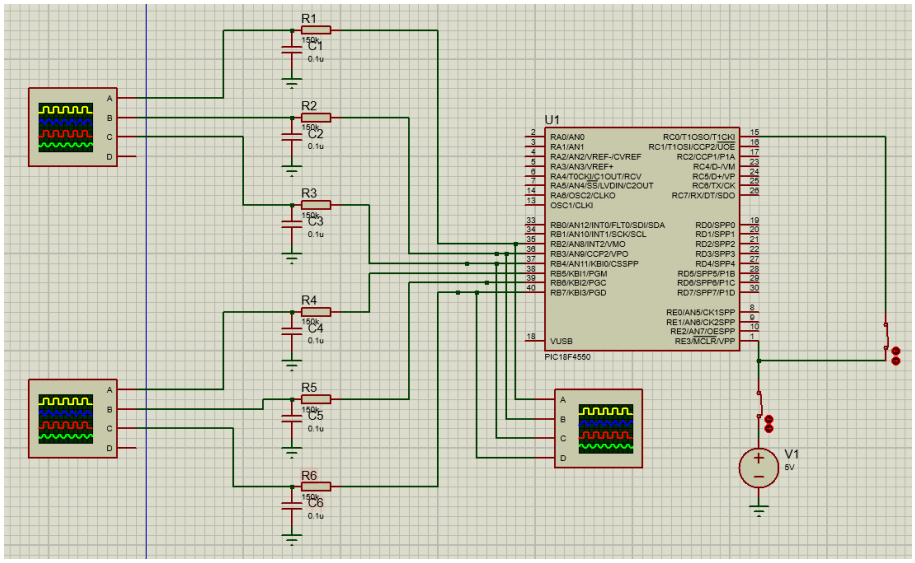


Figure 4.l: Circuit diagram of V/f control

Putting all these concepts together, one can achieve V/f SPWM pulses to drive an inverter stack. It is expected that rms and line to line voltage versus frequency characteristics would be linear.

5. V/f control of Induction Machine theory

The internship project focused on V/f control of an induction machine using an embedded controller like PIC18F4550. This section focuses on the basic V/f control logic and related basic concepts required for maintaining constant flux in the machine.

This is the equivalent circuit of an Induction Machine:

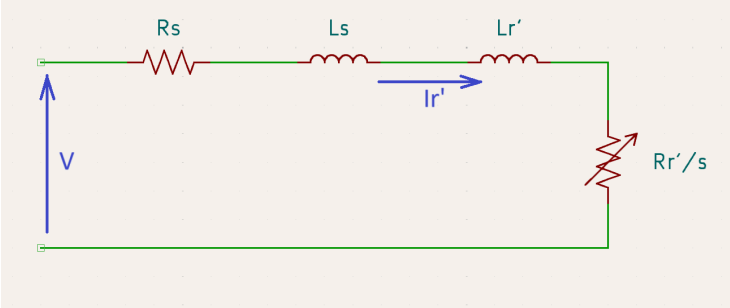


Figure 5.a: Equivalent Circuit of Induction Machine referred to stator

From the expression of stator voltage:

$$V = 4.44\, f\, \Phi\, k_{\omega 1}\, N_1$$

$$\Rightarrow \Phi = \frac{V}{4.44\, k_{\omega 1}\, N_1\, f}$$

$$\Rightarrow \Phi = k\,\frac{V}{f}$$

$$\Rightarrow \Phi \propto \frac{V}{f}$$

where,
- f = normalised stator frequency
- Φ = developed flux in the stator
- k_{w1} = fundamental winding factor
- N_1 = number of Stator turns
- V = stator armature voltage

Thus it is proved that by maintaining V/f ratio it is possible to achieve constant flux.

Now the effect on Electromagnetic Torque (T_{em}) and Maximum torque (T_{max}) is considered:

$$T_{em} = \frac{3}{\omega_{sm}} \times I_r^{'2} \times \frac{R_r^{'}}{s}$$

Assuming stator resistance and leakage reactance are negligible,

$$\Rightarrow T_{em} = \frac{3}{\omega_{sm}} \times \frac{V^2}{(\frac{R_r^{'}}{s})^2 + X_r^{'2}} \times \frac{R_r^{'}}{s}$$

$$\Rightarrow T_{em} = \frac{3P}{\omega_{se}} \times \frac{V^2 s^2}{(\frac{R_r^{'}}{X_r^{'}})^2 + s^2} \times \frac{\frac{R_r^{'}}{X_r^{'}}}{s} \times \frac{1}{X_r^{'}}$$

replacing $\frac{R_r^{'}}{X_r^{'}}$ by α,

$$\Rightarrow \frac{3P}{\omega_{se}} \times \frac{V^2}{X_r^{'}} \times \frac{s\alpha}{s^2 + \alpha^2}$$

$$\Rightarrow k_T \, . \, \frac{s\alpha}{s^2 + \alpha^2} \quad \text{— (1)}$$

$$\text{where, } k_T = \frac{3P}{\omega_{se}} \times \frac{V^2}{X_r^{'}}$$

where, ω_{sm} = mechanical stator flux frequency
ω_{se} = electrical stator flux frequency
s = rotor slip of induction machine
I_r' = rotor current
R_r' = rotor equivalent resistance
X_r' = rotor equivalent leakage reactance

Now differentiating the Electromagnetic Torque Equation to find condition for maximum torque:

$$\frac{dT_{em}}{ds} = 0 \Rightarrow \frac{(\alpha^2 + s^2) - \alpha s(2s)}{(\alpha^2 + s^2)^2} = 0$$

$$\Rightarrow \alpha^3 + \alpha s^2 - 2\alpha s^2 = 0$$

$$\Rightarrow \alpha^2 = s^2$$

$$\Rightarrow \alpha = \pm s$$

So, $\alpha = \pm s$ is the condition for maximum torque.
For Motoring operation, $\alpha = s$

$$\therefore \text{ putting } \alpha = s \text{ in } (1),$$

$$T_{max} = \frac{k_T}{2} = \frac{3P}{2\,\omega_{se}} \times \frac{V^2}{X_r'}$$

$$= \frac{3P}{4\pi f} \times \frac{V^2}{2\pi f L_r'}$$

$$= \frac{3P}{8\pi^2 L_r'} \times \frac{V^2}{f^2}$$

$$\therefore T_{max} \propto (\frac{V}{f})^2$$

So, if the V/f ratio is kept constant, Maximum Torque of the Induction Machine is constant. This is visible in the Torque-Speed characteristics of V/f IM drive.

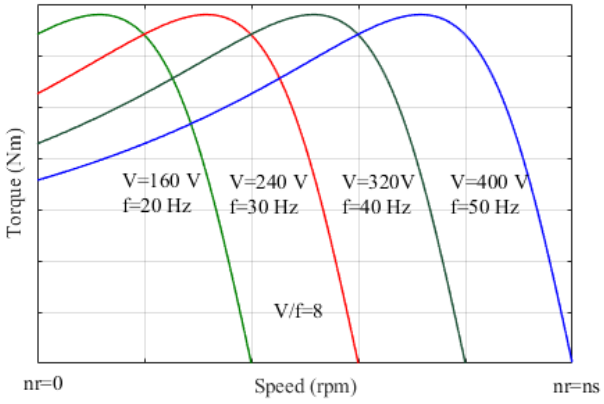


Figure 5.b: Torque-Speed characteristics of V/f IM drive (not experiment data)

6. SPWM based 3-phase inverter operation

The basic 3-phase inverter operation with SPWM gate driving signal is explained in this section.

Assume the basic structure of a 3-phase star connected inverter:

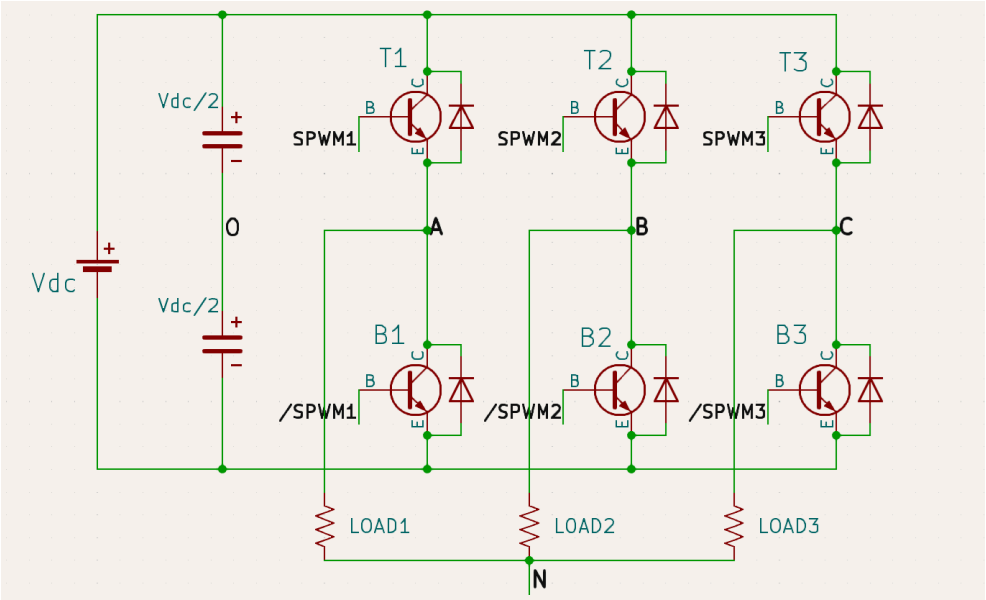


Figure 6.a: 3-phase inverter with leg 1 (T1, B1) receiving SPWM1, leg 2 (T2, B2) receiving SPWM2, leg 3 (T3, B3) receiving SPWM3

Principle of working:

$$\text{Considering operation on Leg 1 :}$$

$$\text{When T1 is ON , B1 is OFF} : V_{AO} = \frac{V_{dc}}{2}$$

$$\text{When T1 is OFF , B1 is ON} : V_{AO} = -\frac{V_{dc}}{2}$$

Where, V_{AO} = pole voltage of inverter
V_{dc} = DC-link voltage of inverter

Considering the voltage values of the reference sine wave:

$$(V_{pole,\,p-p})_{sine} = (V_{AO,\,p-p})_{sine} = m_a V_{dc} \quad \text{---(1)}$$

$$\Rightarrow (V_{AO,\,peak})_{sine} = \frac{m_a V_{dc}}{2} \quad \text{---(2)}$$

$$\Rightarrow (V_{AO,\,rms})_{sine} = \frac{m_a V_{dc}}{2\sqrt{2}} \quad \text{---(3)}$$

Now comparing pole voltage and phase voltage,

$$V_{AN} = V_{AO} - V_{NO} \quad \text{---(4)}$$

Considering point O as the reference point of all calculations,

$$V_{NO} = V_N \quad \text{---(5)}$$

$$\Rightarrow V_{AN} = V_{AO} - V_N \quad \text{---(6)}$$

$$\text{But, } V_N = \frac{V_{AO} + V_{BO} + V_{CO}}{3} \quad \text{---(7)}$$

Since all 3-phase SPWMs are 120° phase shifted,
all the pole voltages are also 120° phase shifted.

$$v_{AO} = \frac{m_a V_{dc}}{2}\sin(\omega t) \text{ [from (2)]} \quad \text{---(8)}$$

$$v_{BO} = \frac{m_a V_{dc}}{2}\sin\left(\omega t - \frac{2\pi}{3}\right) \quad \text{---(9)}$$

$$v_{CO} = \frac{m_a V_{dc}}{2}\sin\left(\omega t + \frac{2\pi}{3}\right) \quad \text{---(10)}$$

putting (8), (9), (10) in (7),

$$V_N = \frac{1}{3} \times \frac{m_a V_{dc}}{2} \times \left[\sin(\omega t) + \sin\left(\omega t - \frac{2\pi}{3}\right) + \sin\left(\omega t + \frac{2\pi}{3}\right)\right] = 0$$

[as, $\sin(x) + \sin(x+120) + \sin(x-120) = 0$] ---(11)

Hence, equation (11) proves that,

$$V_N = 0$$

$$\therefore \text{from } (4),\ V_{AN} = V_{AO} \quad \text{---}(12)$$

$$\begin{aligned} \therefore\ (V_{LL,\,rms})_{sine} &= (V_{AB,\,rms})_{sine} \\ &= \sqrt{3}\,(V_{ph,\,rms})_{sine} \\ &= \sqrt{3}\,(V_{AN,\,rms})_{sine} \\ &= \sqrt{3}\,(V_{AO,\,rms})_{sine} \\ &= \frac{\sqrt{3}}{2\sqrt{2}}\,m_a\,V_{dc} = 0.612\,m_a\,V_{dc} \quad \text{---}(13) \end{aligned}$$

The result obtained in equation (13) is the expression for the RMS value of the fundamental component of line-to-line voltage of the 3-phase inverter output.

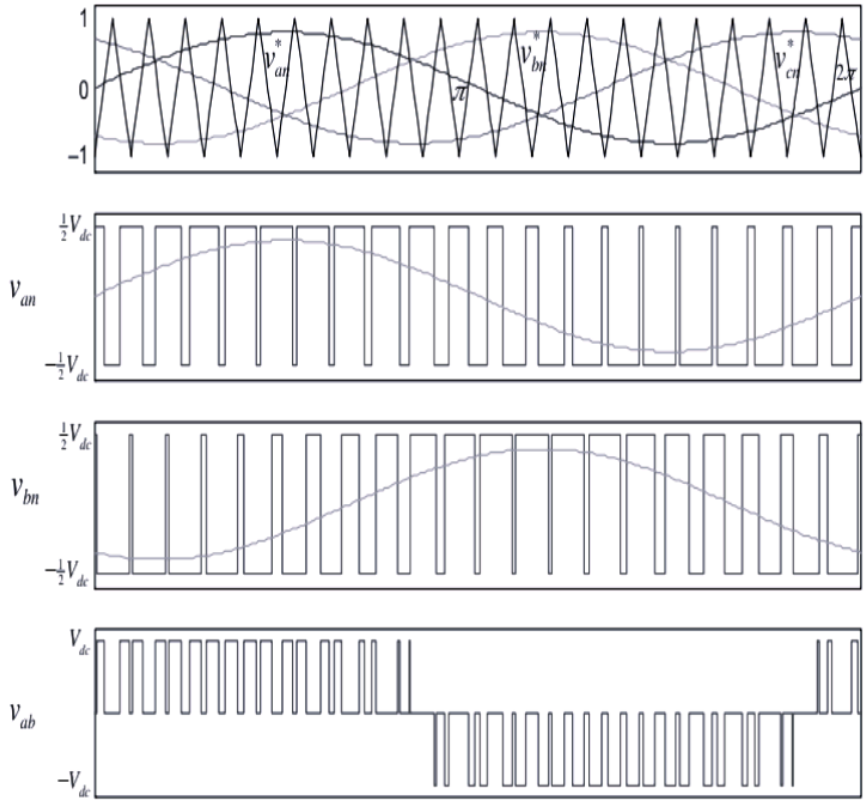


Figure 6.b: Sine wave compare, Phase voltage (v_{an}, v_{bn}) & Line voltage (v_{ab}) waveforms

7. Methods Used & Results Obtained

This section focuses on the various methods and results obtained on the real-time power electronic and machine drive system available in the Advanced Power Electronics Laboratory to achieve V/f control.

The following figure represents the circuit diagram of the Inverter Stack used. It is manufactured by Semikron. The IGBTs used are SKM75GB123D.

IGBT specifications:

- **Collector–Emitter Voltage (V_{CES})**: 1200 V
- **Continuous Collector Current (I_C)**: 75 A @ 25 °C
- **Peak Collector Current**: 150–225 A (short pulse)
- **Gate–Emitter Voltage (V_{GE})**: ±20 V max
- **Collector–Emitter Saturation Voltage ($V_{CE(sat)}$)**: ~2.5 V
- **Junction Temperature Range**: −40 °C to +150 °C
- **Power Dissipation**: ~460 W max
- **Rise Time (t_r)**: ~56 ns
- **Module Type**: SEMITRANS 2 dual IGBT with fast recovery diode
- **Package Isolation Voltage**: 2500 V RMS (baseplate to terminals)

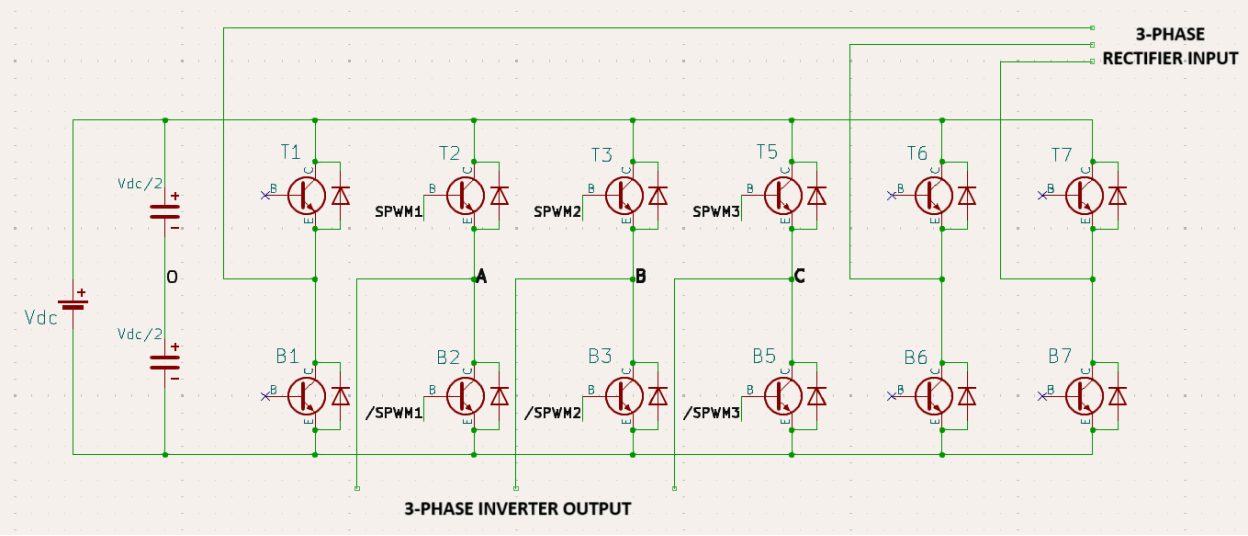


Figure 7.a: Inverter stack implemented circuit diagram

I. Gate Driver Functionality Assessment Test

Before starting the experiment, the gate driver circuits were tested using a Techtronix function generator with 5v square wave pulses of 1 Hz with 180° phase difference taking one leg at a time.

Step 1: All the IGBT gates in the remaining 5 legs were shorted to ground.

Step 2: In the gate driver circuit under test, the top and bottom IGBT gates are driven by 5V square wave pulses that are 180 degrees out of phase.

Step 3: Blinking LEDs in the gate driver circuit indicate that the circuit is functional.

Results:

IGBT legs 1 and 7 were found to be non-functional, while the remaining legs operated correctly. However, the body diodes of all IGBTs were functional, as confirmed by the accurate DC-link voltage observed when the devices were configured as a rectifier.

II. V/f characteristics analysis of unsymmetrical practical L load

The following circuit was used in the experiment:

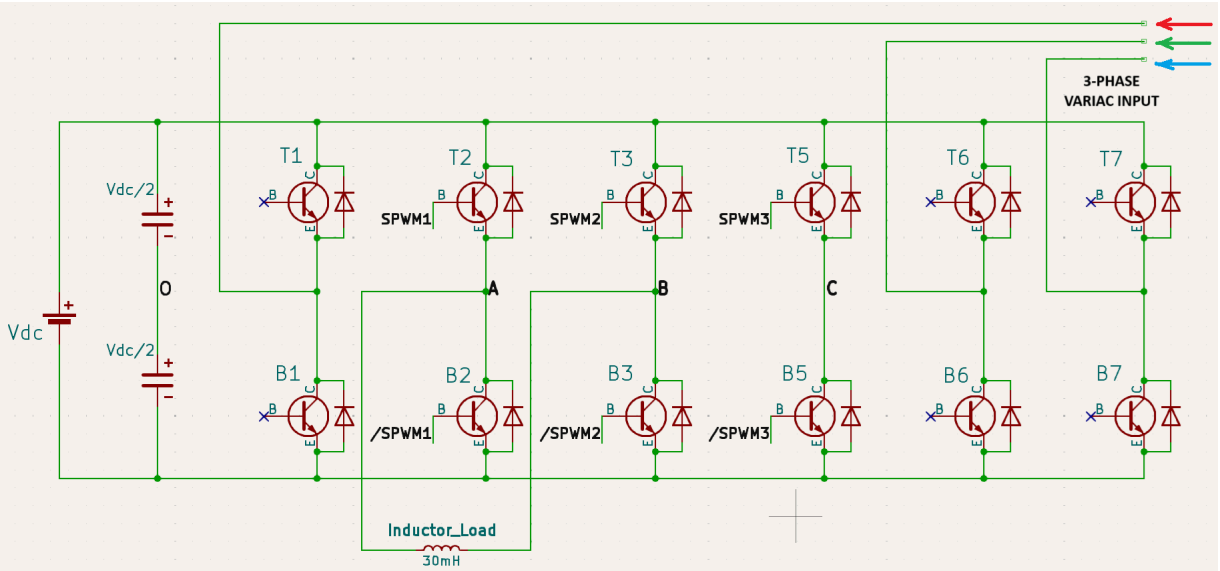


Figure 7.b: Unsymmetrical practical L load circuit

There are various methods of verifying V/f control. The method used in this experiment is discussed below:

➢ **Predictive Estimation of the Peak-to-Peak Amplitude of the Fundamental Line-to-Line Voltage**

Step 1: An inductor unit of suitable value is connected between the phases, in this case between A and B. This yields a sinusoidal current waveform from which the peak-to-peak current can be accurately measured, providing a valuable alternative in situations where a sinusoidal voltage waveform is not directly available (refer Figure 6.b).

Step 2: Readings of frequencies from 0 to 50 Hz and their corresponding peak-to-peak current values are taken from the oscilloscope.

Step 3: Impedance (Z) is calculated for each frequency. To obtain the value of DC resistance, phase lag is calculated from the voltage and current waveform at 50Hz and the following formula is used:

$$\tan^{-1}(\frac{\omega L}{R}) = \theta \;\Rightarrow\; R = \frac{\omega L}{\tan\theta}$$

where, $R \rightarrow$ DC resistance of inductor

$L \rightarrow$ Inductance used

$\omega \rightarrow$ operating frequency

$\theta \rightarrow$ phase lag angle

Step 4: By multiplying impedance and peak-to-peak current, peak-to-peak voltage is obtained in each case.

For the experiment, calculated values are:

- L = 30 mH
- ω = ($2\pi \times 50$) rad/s = 314.16 rad/s, at 50 Hz
- θ = 75° (approx)

So, DC resistance calculated is R = 2.5 Ω

Results:

Time (s)	Peak-to-peak current - I_{p-p} (A)	Frequency of input voltage - f (Hz)	Impedance - Z (Ω)	Peak-to-peak voltage - V_{p-p} (V) $I_{p-p} \times Z$	V/f ratio V_{p-p}/f
3	1.56	8	2.9	4.5	0.56
6	2.52	15.8	3.8	9.5	0.6
9	3.16	22.2	4.8	15.16	0.6
12	3.6	27	5.6	20.16	0.7
15	3.7	36	7.2	26.64	0.74
18	3.7	40.3	7.9	29.23	0.7
21	3.9	45	8.8	34.32	0.76
24	4.16	51	9.9	41.18	0.8

Table 7.a: Results of Unsymmetrical Practical L load experiment

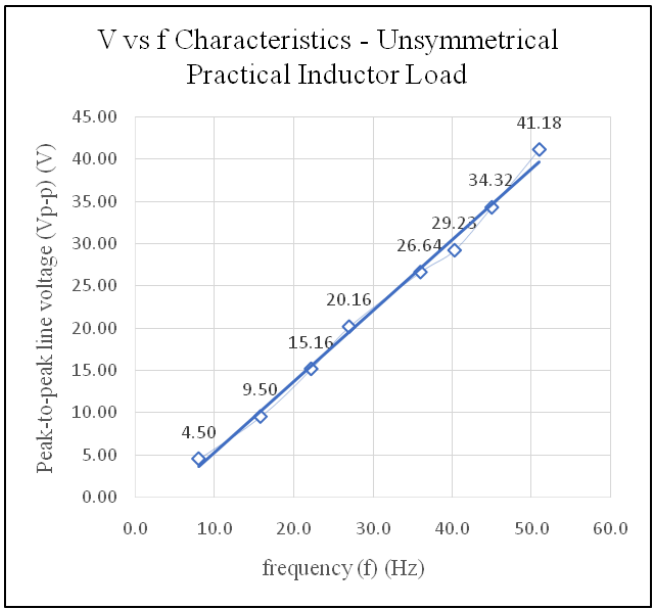


Figure 7.c: V/f characteristics for Unsymmetrical Practical L load

Since this was a test experiment, the waveform image was not taken. For future experiments, output waveform images are well documented.

III. V/f characteristics analysis of symmetrical star-connected practical L load

The following circuit was used in the experiment:

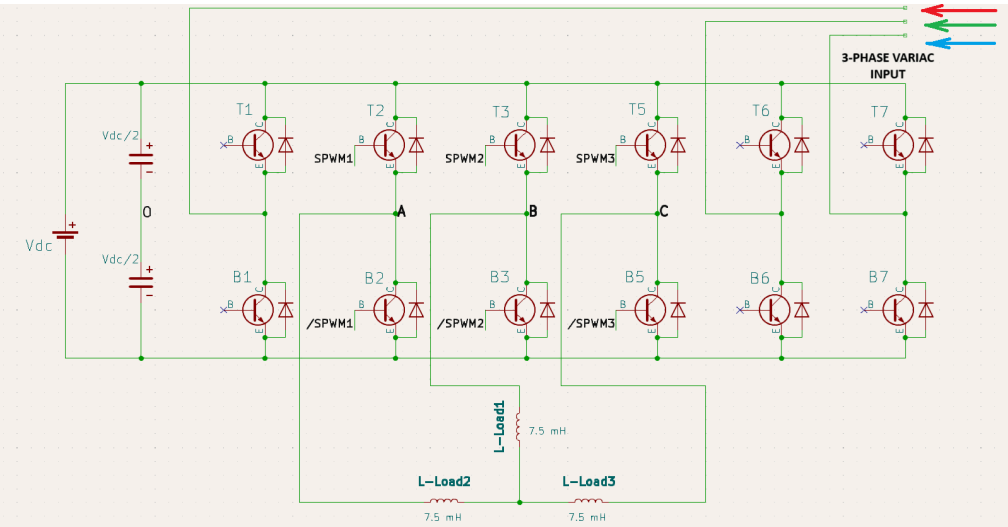


Figure 7.d: Symmetrical practical L load circuit

The method followed for this experiment remains the same - "**Predictive Estimation of the Peak-to-Peak Amplitude of the Fundamental Line-to-Line Voltage**"

For the experiment, calculated values are:

- L = 7.5 mH
- $\omega = (2\pi \times 50)$ rad/s = 314.16 rad/s, at 50 Hz
- $\boldsymbol{\theta} = 75^{\circ}$ (approx)

So, DC resistance calculated is R = 0.6 Ω

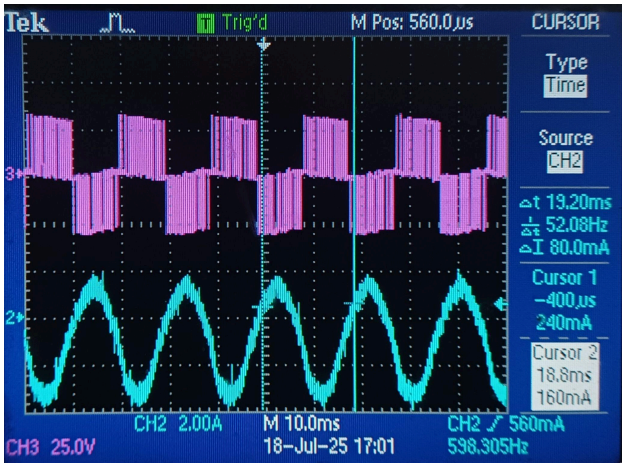


Figure 7.e: Symmetrical practical L load Voltage (purple) and current (blue) waveform at 50 Hz

Results:

Time (s)	Peak-to-peak current - I_{p-p} (A)	Frequency of input voltage - f (Hz)	Impedance - Z (Ω)	Peak-to-peak voltage - V_{p-p} (V) $I_{p-p} \times Z$	V/f ratio V_{p-p}/f
3	2.56	7	0.68	1.74	0.24
6	3.76	14.2	0.89	3.34	0.24
9	4.4	20.8	1.15	5.06	0.24
12	4.64	28	1.47	6.82	0.24
15	4.88	33.3	1.68	8.19	0.25
18	5.04	38	1.88	9.47	0.25
21	5.04	41.67	2.05	10.33	0.25
24	5.52	50	2.4	13.24	0.26

Table 7.b: Results of Symmetrical Star-connected Practical L load experiment

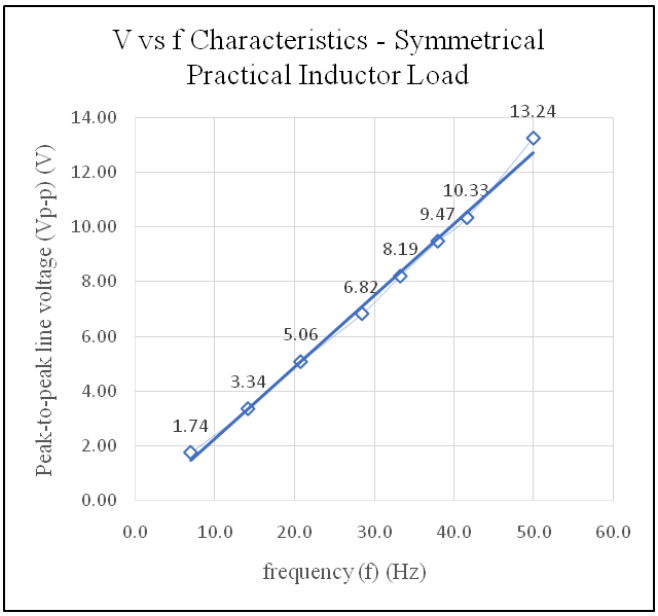


Figure 7.f: V/f characteristics for Symmetrical Practical L load

IV. V/f characteristics analysis of Induction Machine load (no-load operation of machine)

The following circuit was used in the experiment:

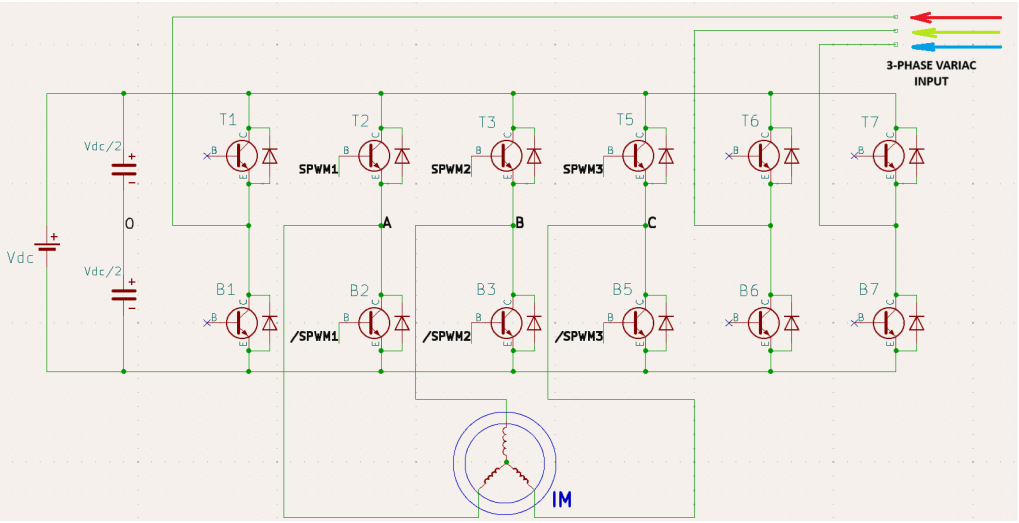


Figure 7.g: Induction Motor load circuit

The method used in this experiment is discussed below:

➢ **Absolute Determination of V/f ratio from the RMS of the Fundamental Line-to-Line Voltage**

In this method the RMS voltage reading is directly taken from the inverter output nodes using a Digital Multimeter.

Induction Machine Specifications:

a. Power rating = 1.1 kW (1.5 hp)
b. $V_{L\text{-}L,\ rated}$ (RMS) = 415V
c. f_{rated} = 50 Hz
d. I_{rated} = 2.7 A
e. Star-connected, 4 pole
f. Rated speed = 1410 rpm

A few important points before discussion of results obtained:

$$V_{L-L,\text{rated (RMS)}} = 415 \text{ V}$$
$$f_{rated} = 50 \text{ Hz}$$
$$\therefore (\frac{V}{f})_{rated} = \frac{415}{50} = 8.3$$

$$\text{at } f = 50 \text{ Hz}, m_a = 1 \text{ (i.e., } 100\%)$$
$$\therefore \text{we know, } 0.612 \cdot m_a \cdot V_{dc} = V_{L-L,\, rms}$$
$$\Rightarrow V_{dc} = \frac{V_{L-L}}{0.612 \cdot m_a} = \frac{415}{0.612 \cdot 1} \simeq 678 \text{ V}$$

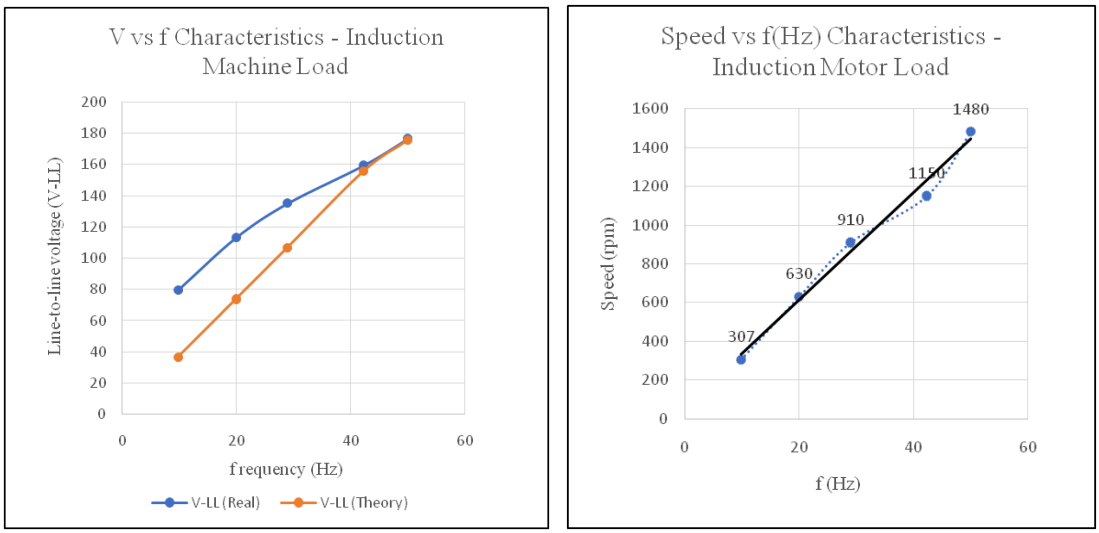
$$\text{But, permitted } V_{dc} = 300 \text{ V}$$
$$\therefore V_{L-L,\,\text{rms}} = 0.612 \cdot m_a \cdot V_{dc} = 0.612 \cdot 1 \cdot 300 = 183.6 \text{ V}$$
$$\therefore (\frac{V}{f})_{new} = \frac{183.6}{50} = 3.672$$

$$\text{Hence, flux content in IM stator} = \frac{3.672}{8.3} = 44.2\%$$

Results:

Frequency of input voltage - f (Hz)	**Line-to-Line voltage (Real) - V_{L-L} (V)**	**Line-to-Line voltage (Theory) - V_{L-L} (V)**	**DC link voltage - V_{dc} (V)**	**V/f (Real)**	**V/f (Theory)**	**Speed - N_r (rpm)**
9.9	79.5	37	304	8.03	3.73	307
20	113	74	303.5	5.6	3.7	630
29	135	107	304	4.6	3.68	910
42.3	159.4	155.84	301	3.7	3.68	1150
50	176.4	175.6	302	3.5	3.5	1480

Table 7.c: Results of Induction Motor load experiment



i ii

Figure 7.h: V/f characteristics of IM for practical and ideal case (i) and Speed vs frequency characteristics of IM (ii)

Here are the waveforms observed for the slow start of IM:

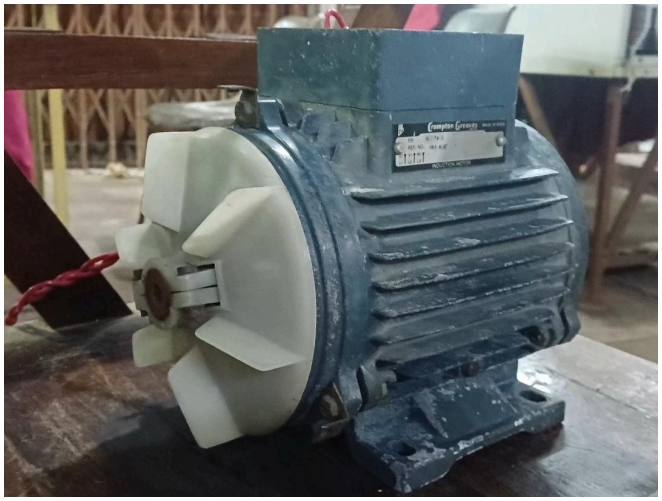
i. f= 10 Hz ii. f= 25 Hz

iii. f= 50 Hz

Figure 7.i: Output voltage and phase current waveform during slow start V/f control.

The images below illustrate the complete experimental setup used for validation, highlighting the arrangement of power stages, control circuitry, and measurement instruments:

i. Experimental setup



ii. Induction Motor used for the experiment

Figure 7.j: Experimental setup and Induction Motor used for the experiments.

8. Conclusion

The internship work concludes with the study of **open-loop V/f control of Induction Motor using embedded controllers** like PIC18F4550. The study had valuable insights into practical versus theoretical concepts of 3 major domains: **Embedded Systems**, **Power Electronics** and **Machine Drives**.

Highlights of the project:

1. Development and Implementation of a Novel SPWM Logic on a General-Purpose Microcontroller like PIC18F4550 which does not have dedicated PWM pins implementable for motor control.

2. Operational Analysis of a Three-Phase Inverter Stack Driven by SPWM-Based Gate Signals.

3. Comparative Analysis of V/f Characteristics Under Symmetrical, Unsymmetrical Inductive Loads, and Induction Motor Loads

Future prospects:

1. Closed-Loop V/f Control of an Induction Motor: Implementation and Performance Analysis

2. Optimization of Embedded C Code for SPWM Generation Using Motor Control Modules of PIC18F4431

9. References and Additional Links

References:

1. https://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf - PIC18F4550 datasheet - Chapters 1 (Device Overview), 9 (Interrupts), 10 (I/O ports), 11 (Timer0 module), 12 (Timer1 module), 13 (Timer2 module), 15 (Capture/Compare/PWM module)

2. Rolin D McKinley, Danny Causey, Muhammad Ali Mazidi - PIC Microcontroller and Embedded Systems Using ASM & C for PIC18 - Pearson Prentice Hall

3. https://ww1.microchip.com/downloads/en/DeviceDoc/21897B.pdf - MCP4921 datasheet

4. https://youtu.be/nTq-OKy5kHs?si=GKZuGD91cSjJMe30 - MITOpenCourseware - MIT 6.622 Power Electronics, Spring 2023 - Lecture 23 - Three Phase Inverters

5. https://youtu.be/NXAAhArHang?si=6aNPNLe0wgd6ZIW9 - SPWM (Sinusoidal PWM) Three Phase Inverters - Ozan Keysan

6. https://pe-iitr.vlabs.ac.in/exp/three-phase-spwm/theory.html - IITR Virtual Labs

7. https://youtu.be/k3-Nh2RmX6k?si=On-lStWdHOnSHQ5D - NPTEL IISc - “Design and Simulation of Power Conversion using Open Source Tools” by Prof. L. Umanand - Lecture 43 - single phase PWM for single phase inverter

8. https://youtu.be/VnAg5kfjFdo?si=n6tbr6gNr7iCSbGN - NPTEL IITKGP - NOC Jan 2019 - Electrical Machines - II by Prof Tapas Kumar Bhattacharya - Lecture 61 - Idea of VVVF Speed Control of Induction Motor

9. https://www.researchgate.net/figure/Speed-torque-curves-with-constant-V-f-ratio_fig5_342156543 - V/f drive Torque-Speed characteristics

Additional Links:

1. https://www.desmos.com/calculator - Desmos Graphing Calculator
2. https://ppelikan.github.io/drlut/ - Dr. LUT - Optimized LUT calculator
3. https://latexeditor.lagrida.com/ - Online LaTeX editor
4. https://github.com/ATRI-DATTA/PIC18F4550-Embedded-codes-for-power-electronic-application