Birla Institute of Technology and Science Pilani, Dubai Campus

Dubai International Academic City

CS/ECE/INSTR/EEE F241

MICROPROCESSORS AND INTERFACING

LABORATORY MANUAL

II Semester 2022-23

Name:……………………..

ID No:…………………….

MICROPROCESSORS AND INTERFACING LABORATORY MANUAL

List of Experiments

Expt No	Name of the Experiment
7	Hardware Design and Simulation using PROETUS Software - 7 Segment LED
8	Hardware Design and Simulation using PROETUS Software - Stepper Motor
9	Hardware Interfacing –Stepper Motor
10	Hardware Interfacing – Elevator Control & Traffic Light Controller

EXPERIMENT-7

HARDWARE DESIGN AND SIMULATION USING PROTEUS
7-SEGMENT LED INTERFACING

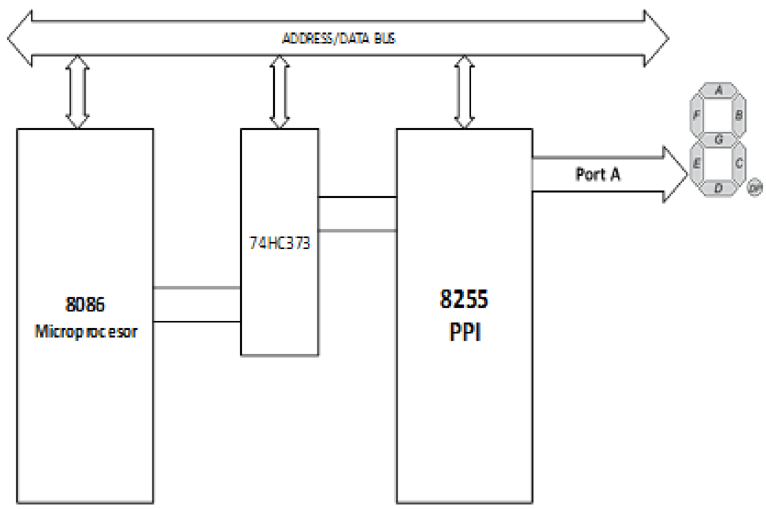
Aim: To Interface a 7 Segment LED to 8086 Microprocessor through 8255 Programmable Peripheral Controller using PROTEUS

What is Proteus Design Suite?

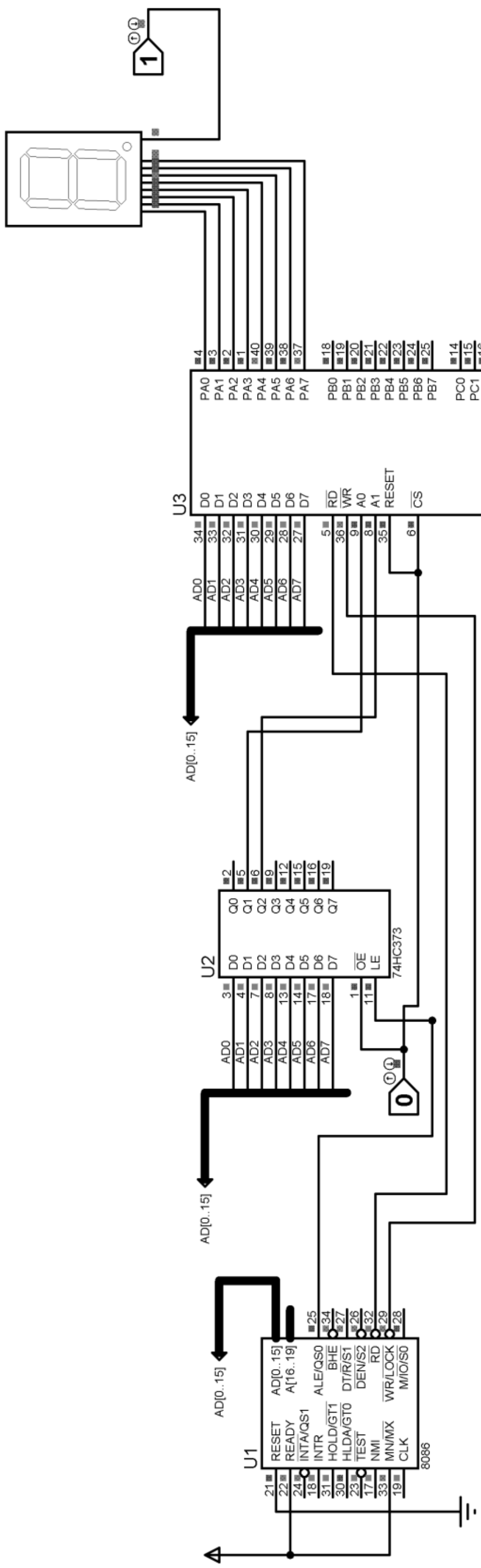
Proteus combines ease of use with powerful features to help you design, test and layout professional PCBs like never before. With nearly 800 microcontroller variants ready for simulation straight from the schematic, one of the most intuitive professional PCB layout packages on the market and a world class shape-based auto router included as standard, Proteus Design Suite 8 delivers the complete software package for today and tomorrow's engineers.

The Proteus simulation products all use the schematic capture module as the electronic circuit and our customized mixed-mode SPICE engine to run the simulation. Proteus VSM then allows the microcontroller to also be simulated on the schematic while Proteus IoT Builder enables the design and test of the remote user interface for the circuit. For embedded engineers, Proteus VSM bridges the gap in the design life cycle between schematic capture and PCB layout. It enables you to write and apply your firmware to a microcontroller component on the schematic (PIC, AVR, ARM, 8051, etc.) and then co-simulate the program within a mixed-mode SPICE circuit simulation.

The Block diagram of interfacing is shown below in Figure 1



Here you will be simulating the schematic of above interfacing, which is shown in the next page Figure 2. This schematic entry is done in the Proteus design suite. Corresponding assembly program also shown below.

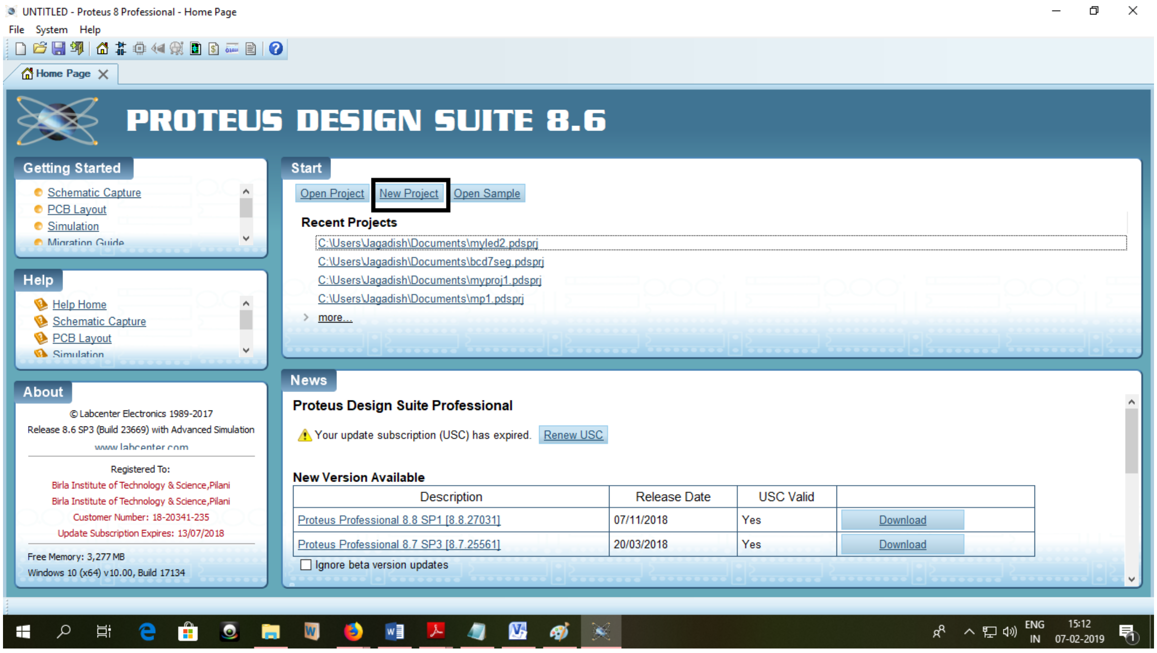


```
DATA SEGMENT
PORTA EQU 00H
PORTB EQU 02H
PORTC EQU 04H
PORT_CON EQU 06H
DATA ENDS
CODE SEGMENT
MOV AX, DATA
MOV DS, AX
ORG 0000H
START:
MOV DX, PORT_CON
MOV AL, 10000000B
OUT DX, AL
MOV SI, 0
MOV DI, 0
L0: MOV CX, 1FFFH
L1: MOV AL, S1[SI]
MOV DX, PORTA
OUT DX, AL
LOOP L1
INC SI
CMP SI, 16
JL L0
MOV DX, PORT_CON
MOV AL, 10000000B
OUT DX, AL
JMP START
ORG 1000H
S1 DB 11000000B
DB 11111001B
DB 10100100B
DB 10110000B
DB 10011001B
DB 10010010B
DB 10000010B
DB 11011000B
DB 10000000B
DB 10010000B
DB 10001000B
DB 10000011B
DB 11000110B
DB 10100001B
DB 10000110B
DB 10001110B
CODE ENDS
END
```

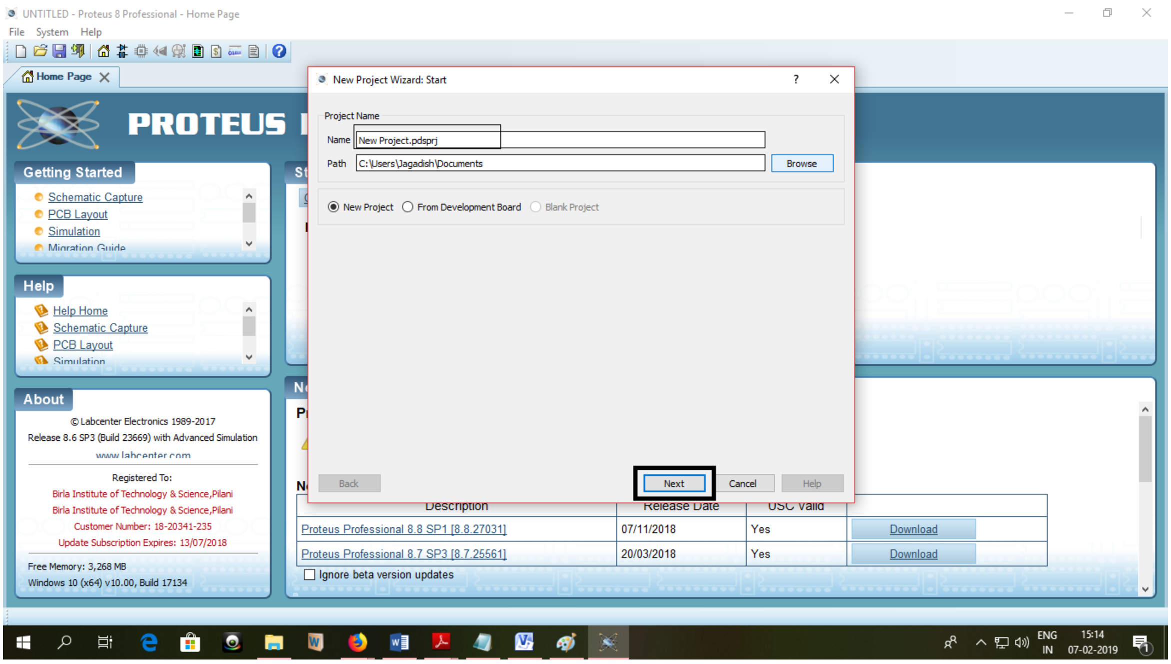
Procedure:

Step 1: Double click on the Proteus 8 Professional

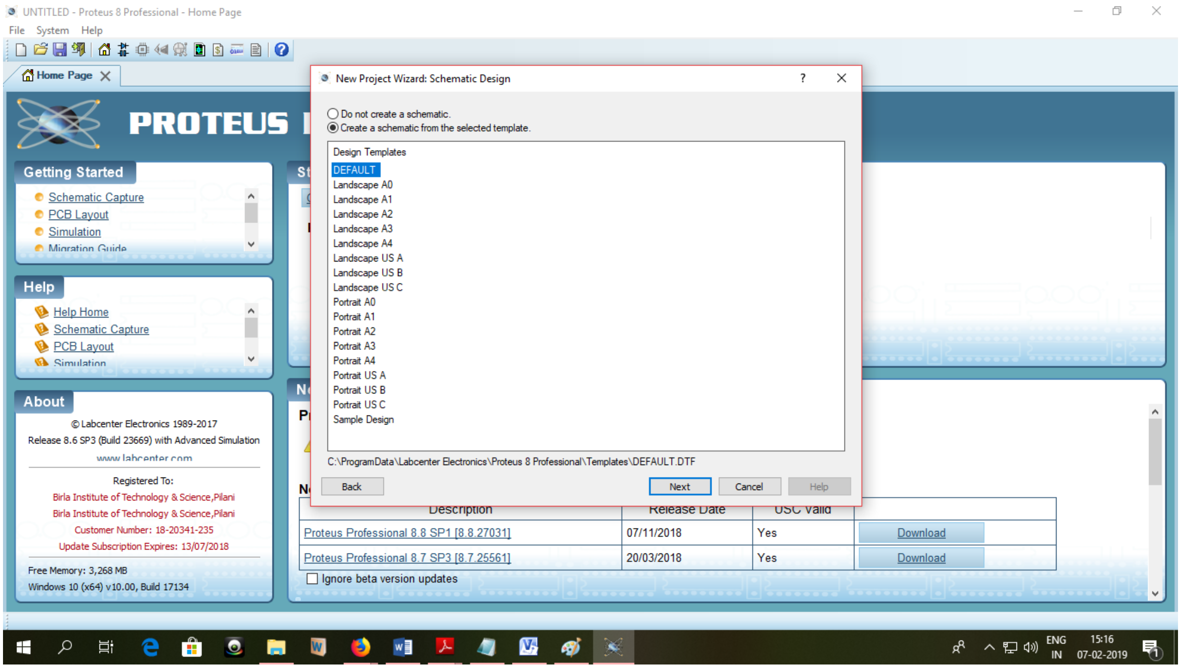
Step 2: Click on the New Project on following screen



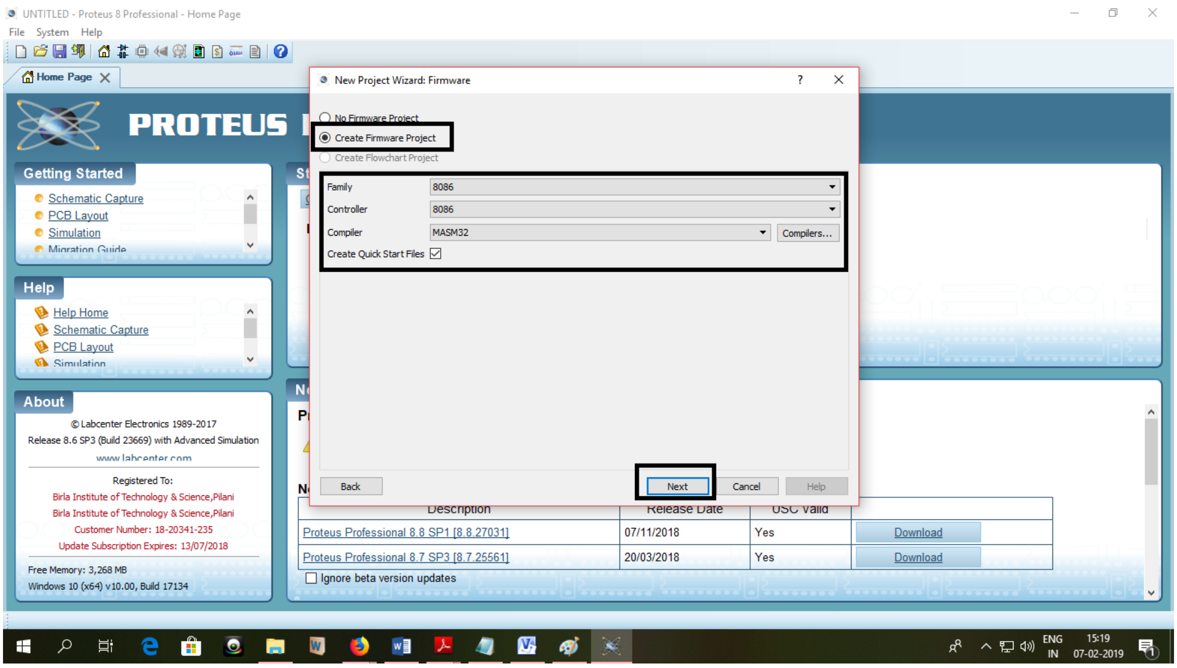
Step 3: Enter the Project name and click on Next



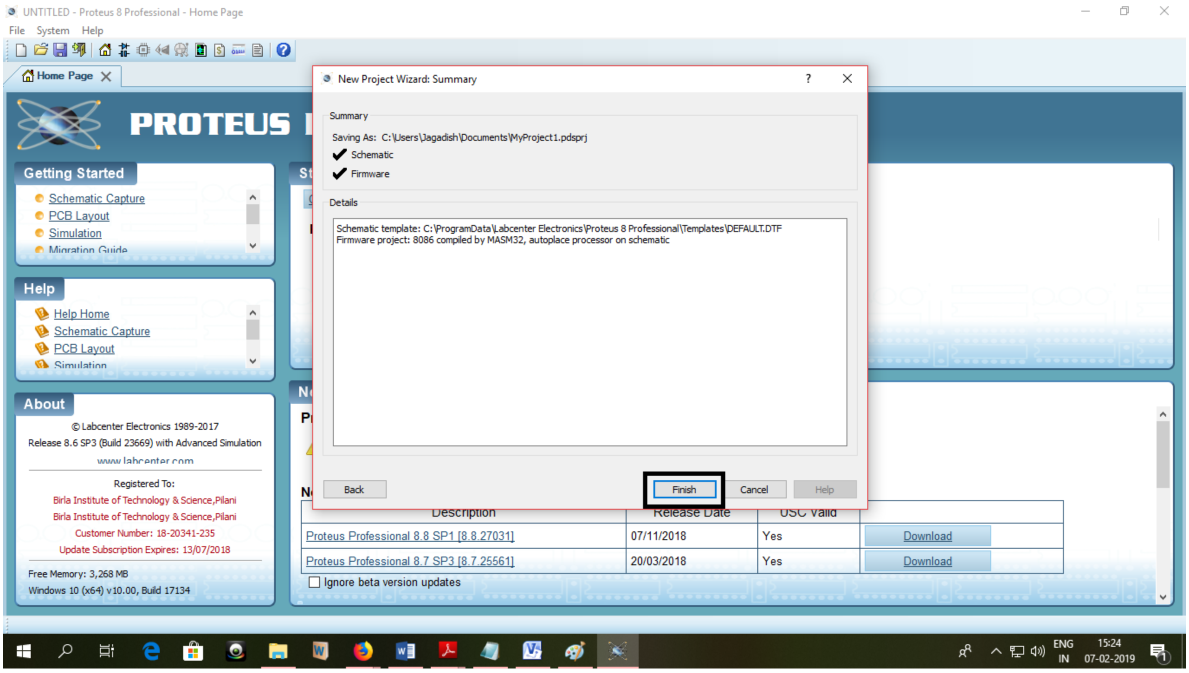
Step 4: Select the design template DEFAULT and click Next



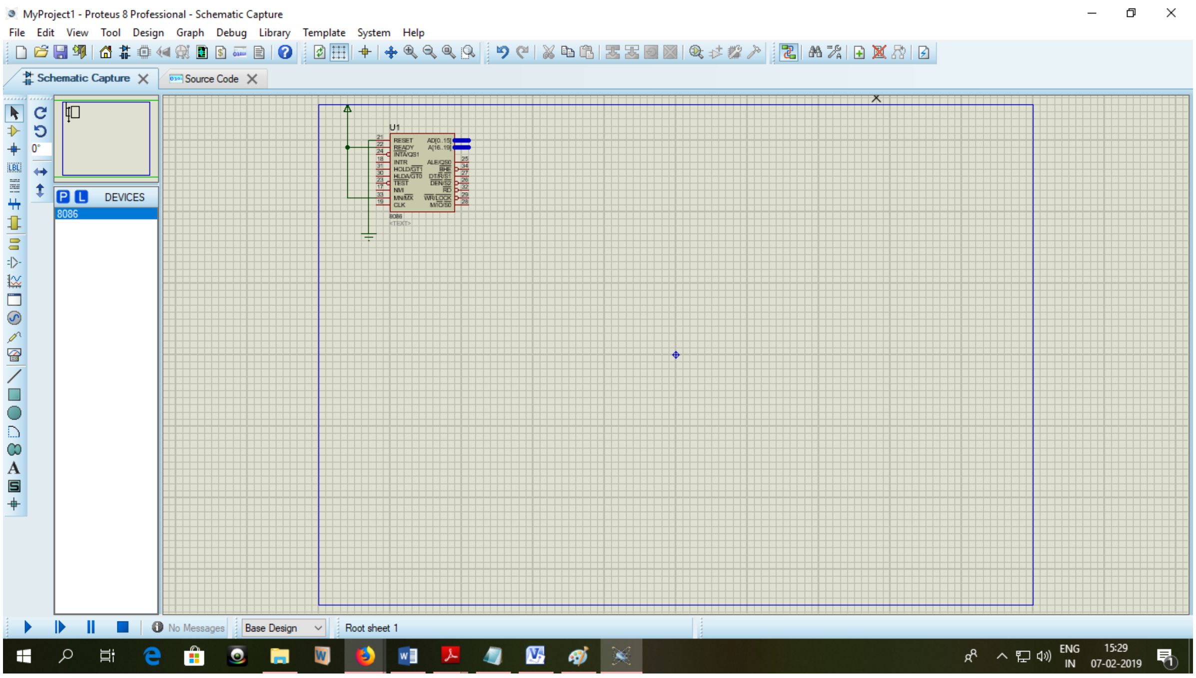
Step 5: Click on Create Firmware Project radio button, Select 8086 in Family, 8086 in Controller and MASM32 in Compiler. Then click Next.



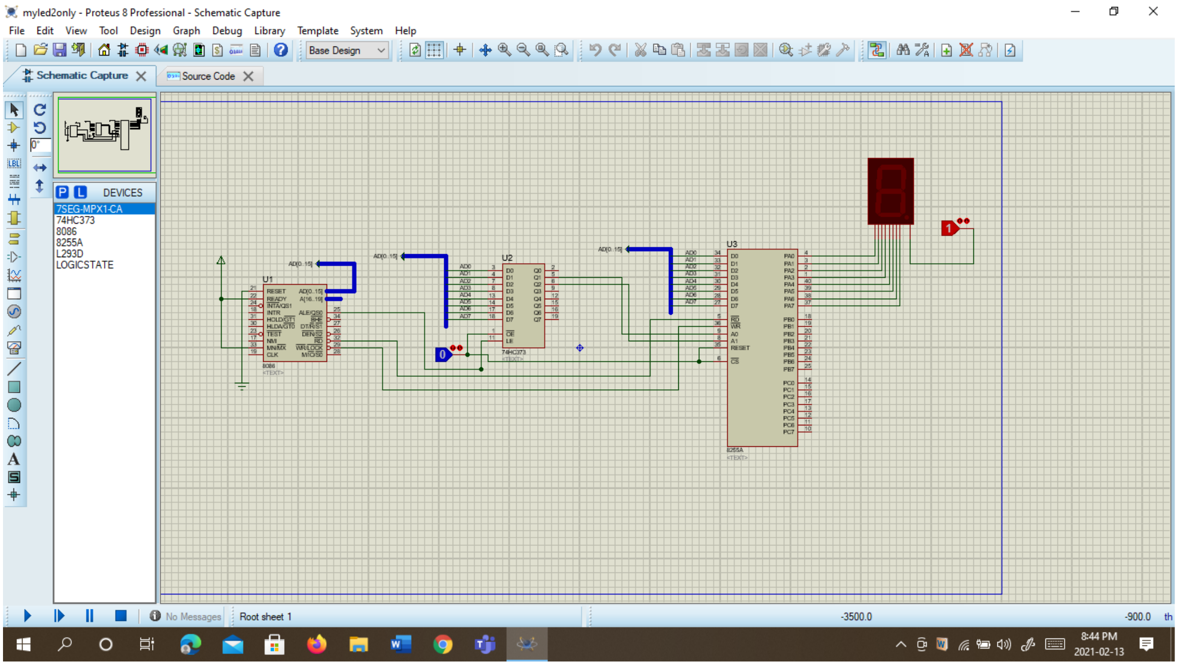
Step 6: Then Click Finish



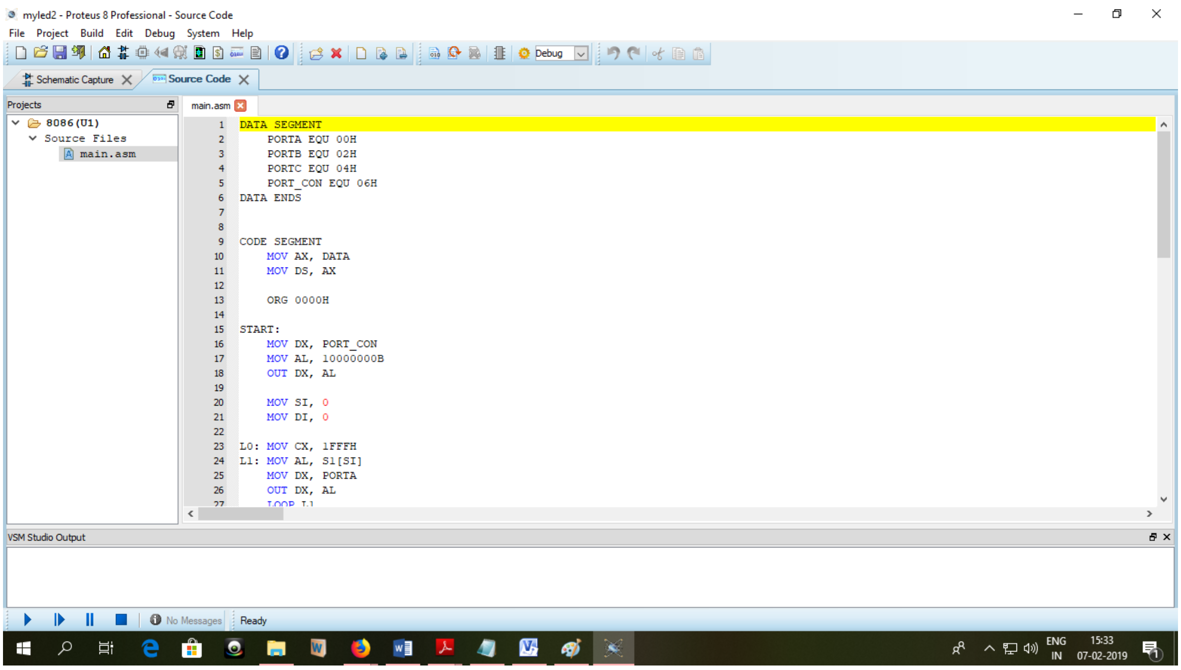
Step 7: You will get the following screen and Click on Schematic Capture.



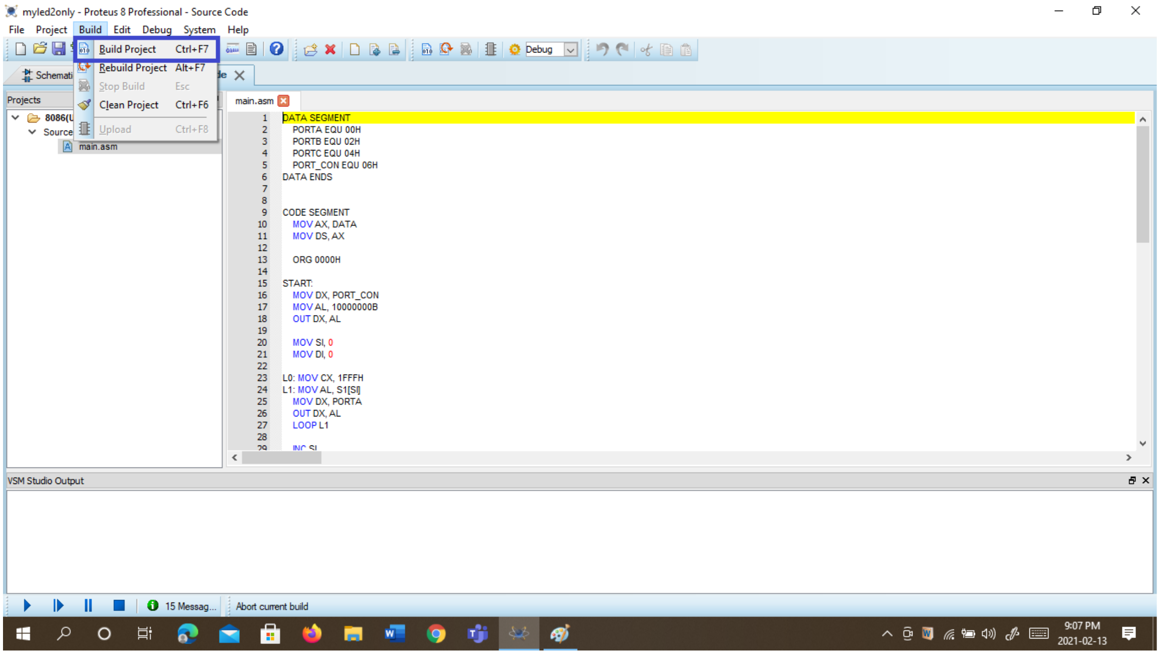
Step 8: Now draw the schematic as shown in Figure 2 as follows.



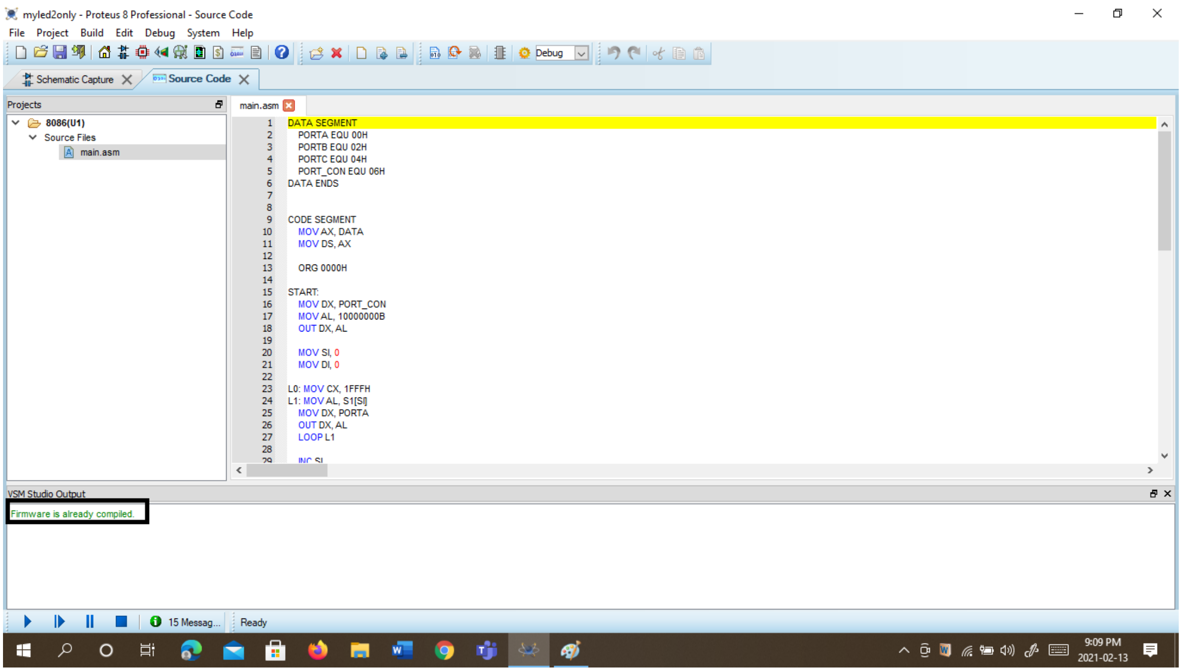
Step 9: Enter the assembly program in Source Code Menu as shown below.



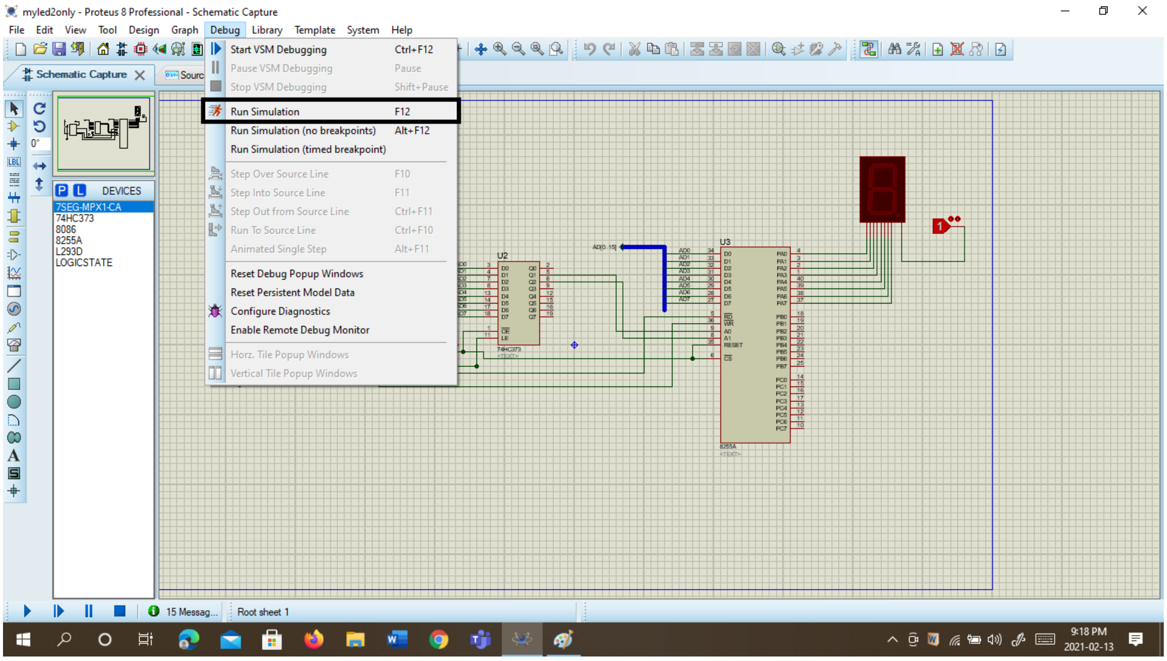
Step 10: Click on the Build and Build Project



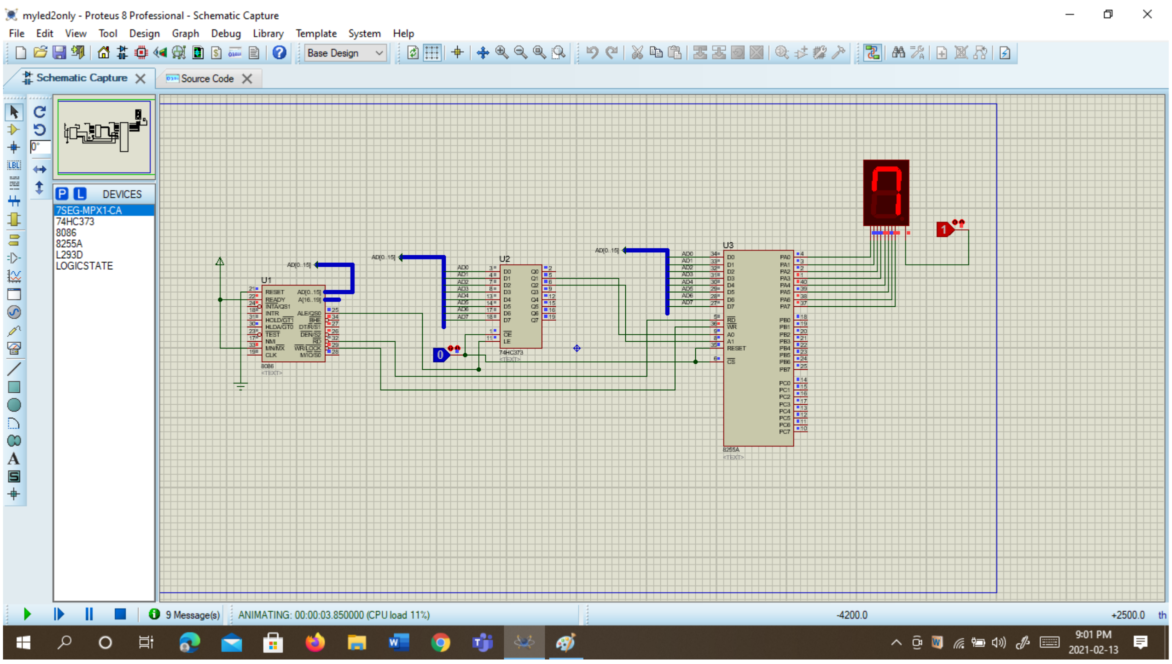
You can see the Build message in the bottom as shown



Step 11: Now in schematic Capture click on Debug and Run Simulation



You can observe LED will be displaying 0 to F continuously.



EXPERIMENT - 8

HARDWARE DESIGN AND SIMULATION USING PROTEUS

STEPPER MOTOR INTERFACING

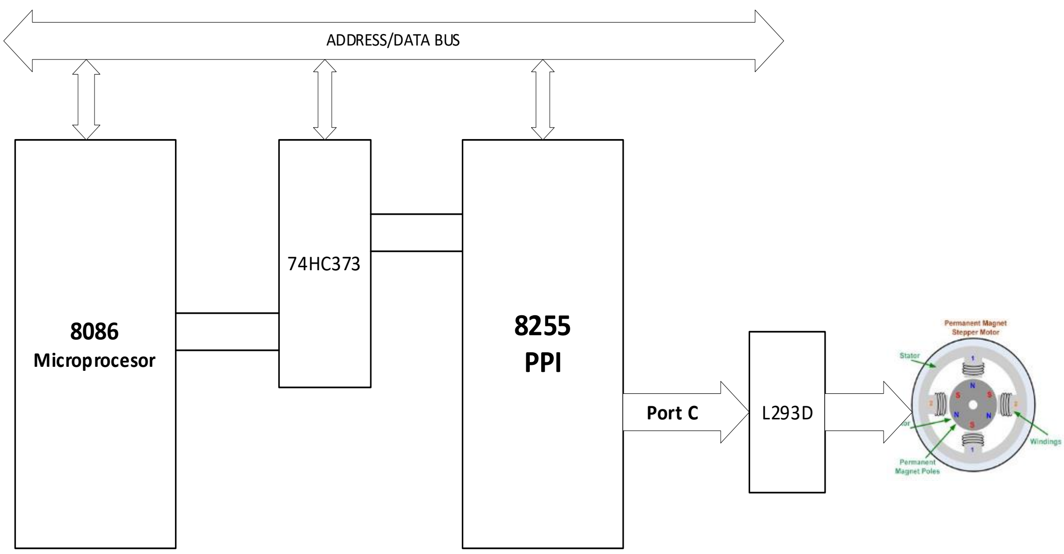
Aim: To Interface a Stepper Motor to 8086 Microprocessor through 8255 Programmable Peripheral Controller using Proteus

What is Proteus Design Suite?

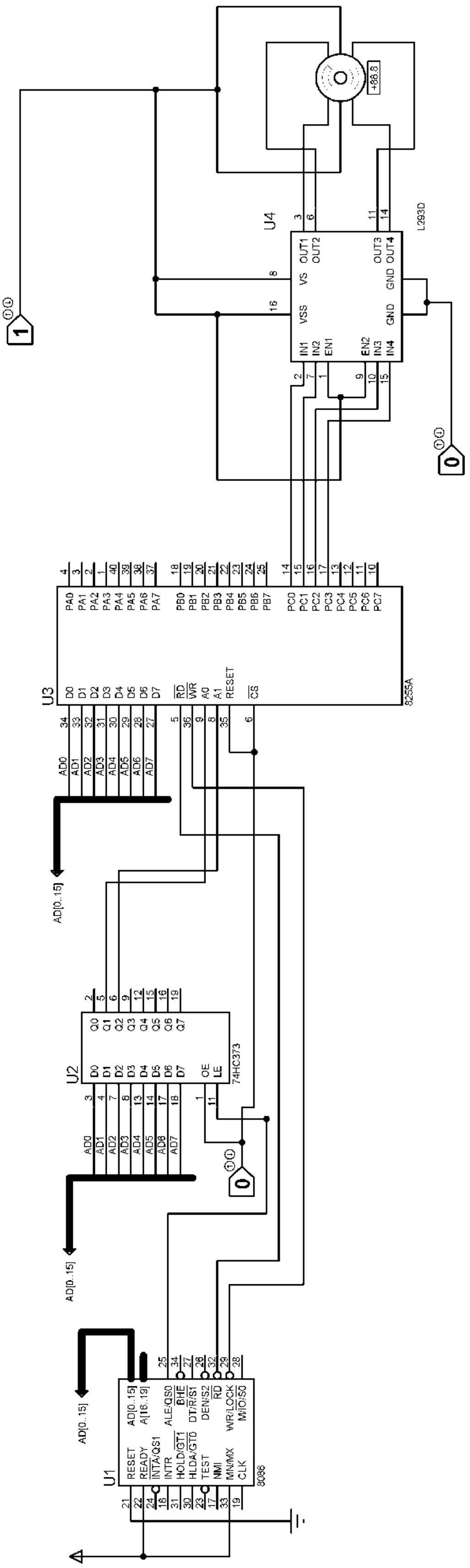
Proteus combines ease of use with powerful features to help you design, test and layout professional PCBs like never before. With nearly 800 microcontroller variants ready for simulation straight from the schematic, one of the most intuitive professional PCB layout packages on the market and a world class shape based autorouter included as standard, Proteus Design Suite 8 delivers the complete software package for today and tomorrow's engineers.

The Proteus simulation products all use the schematic capture module as the electronic circuit and our customized mixed-mode SPICE engine to run the simulation. Proteus VSM then allows the microcontroller to also be simulated on the schematic while Proteus IoT Builder enables the design and test of the remote user interface for the circuit. For embedded engineers, Proteus VSM bridges the gap in the design life cycle between schematic capture and PCB layout. It enables you to write and apply your firmware to a microcontroller component on the schematic (PIC, AVR, ARM, 8051, etc.) and then co-simulate the program within a mixed-mode SPICE circuit simulation.

The Block diagram of interfacing is shown below



Here you will be simulating the schematic of above interfacing, which is shown in the next page Figure 2. This schematic entry is done in the Proteus design suite. Corresponding assembly program also shown below.



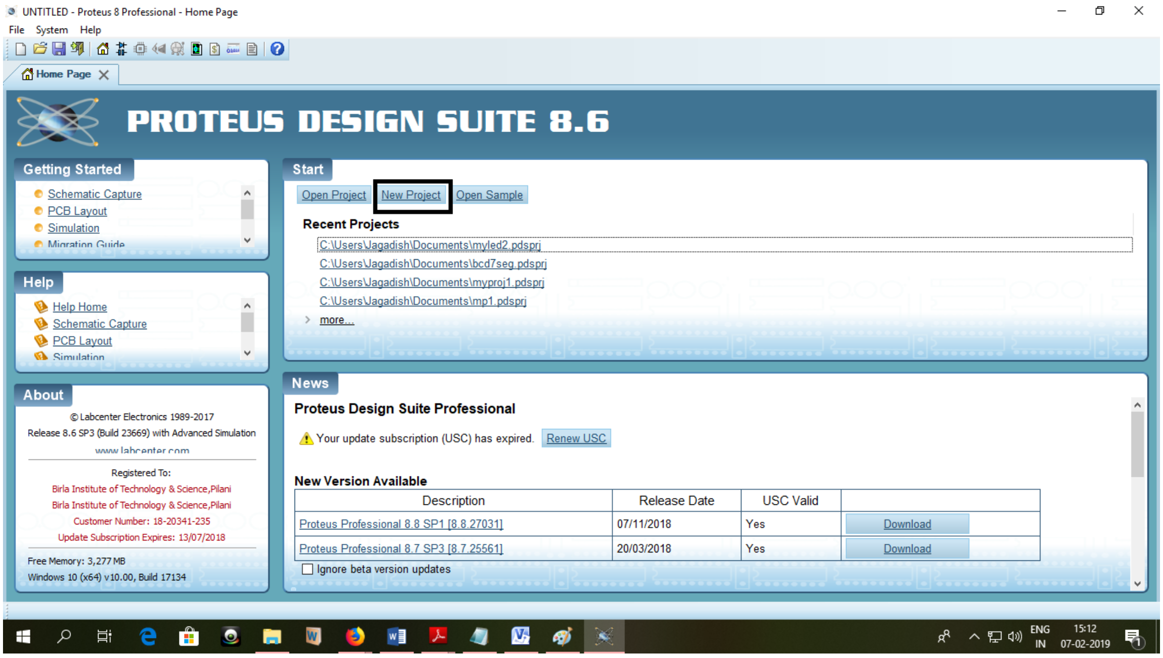
Assembly Program:

```
DATA SEGMENT
PORTA EQU 00H
PORTB EQU 02H
PORTC EQU 04H
PORT_CON EQU 06H
DATA ENDS
CODE SEGMENT
MOV AX, DATA
MOV DS, AX
ORG 0000H
START:
MOV DX, PORT_CON
MOV AL, 10000000B
OUT DX, AL
MOV SI, 0
MOV DI, 0
LL0:MOV CX, 2FFFH
LL1:MOV AL, S2[DI]
MOV DX, PORTC
OUT DX, AL
LOOP LL1
INC DI
CMP DI, 4
JL LL0
JMP START
ORG 1000H
S2 DB 1101B
DB 1011B
DB 0111B
DB 1110B
CODE ENDS
END
```

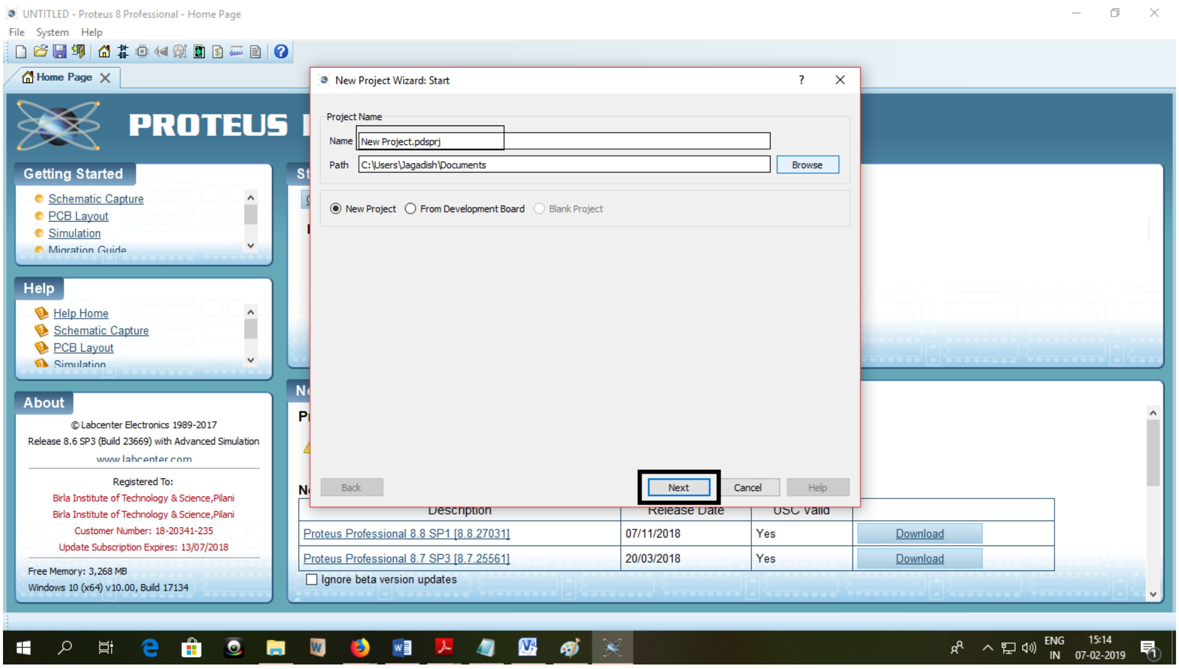
Procedure:

Step 1: Double click on the Proteus 8 Professional

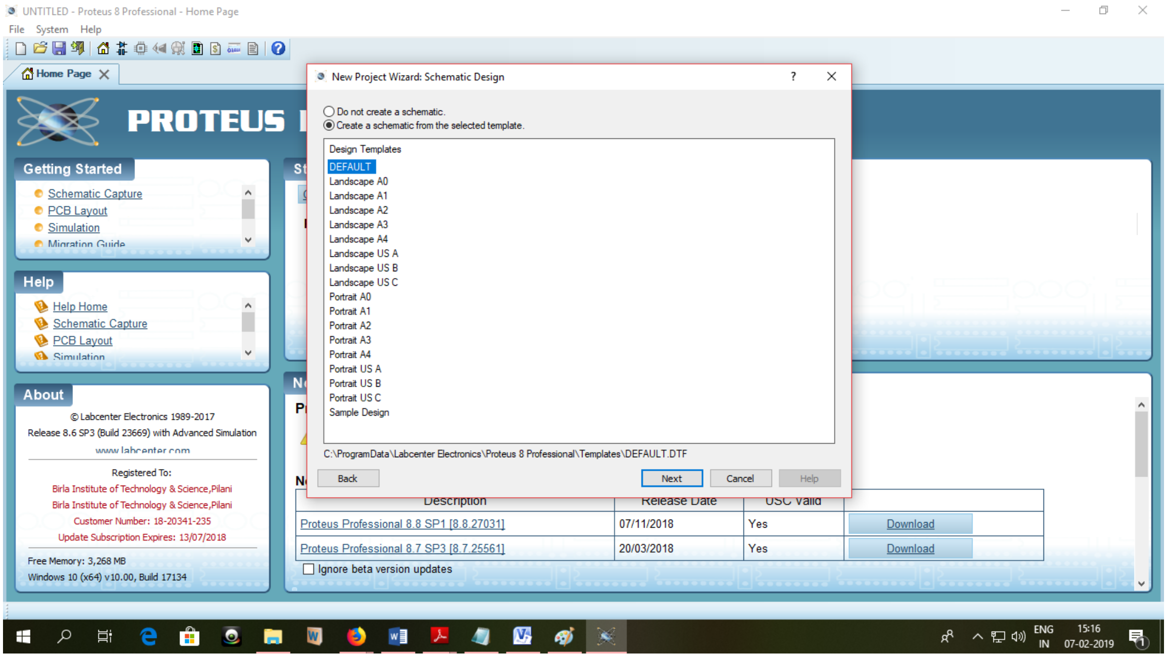
Step 2: Click on the New Project on following screen



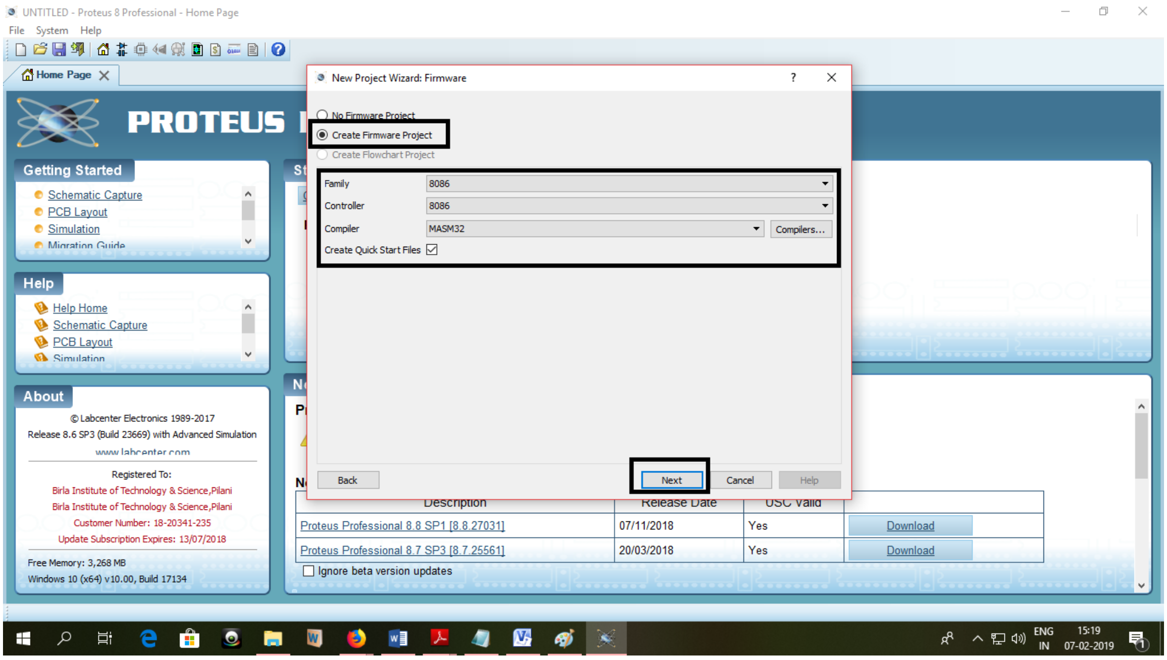
Step 3: Enter the Project name and click on Next



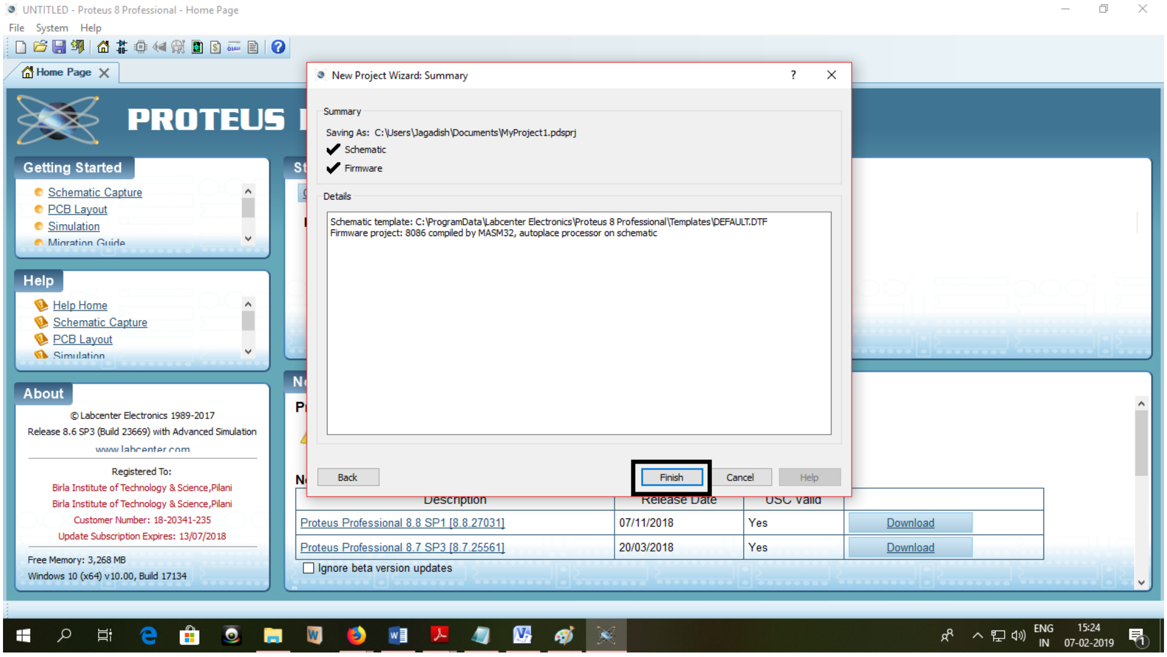
Step 4: Select the design template **DEFAULT** and click **Next**



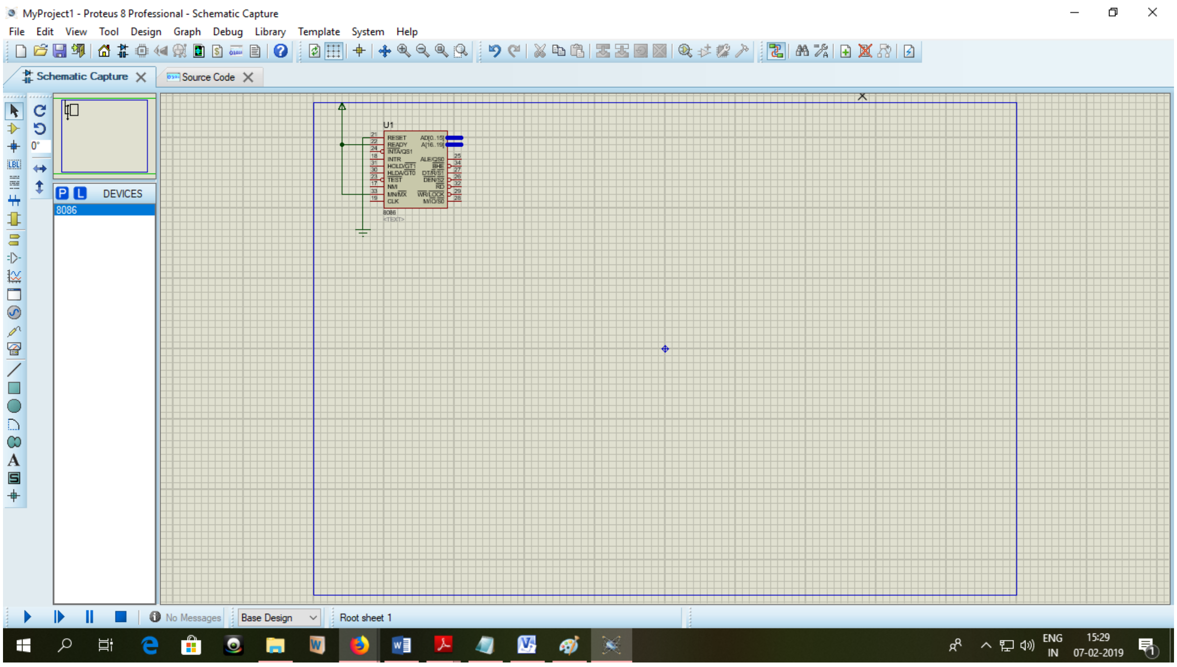
Step 5: Click on Create Firmware Project radio button, Select 8086 in Family , 8086 in Controller and MASM32 in Compiler. Then click Next.



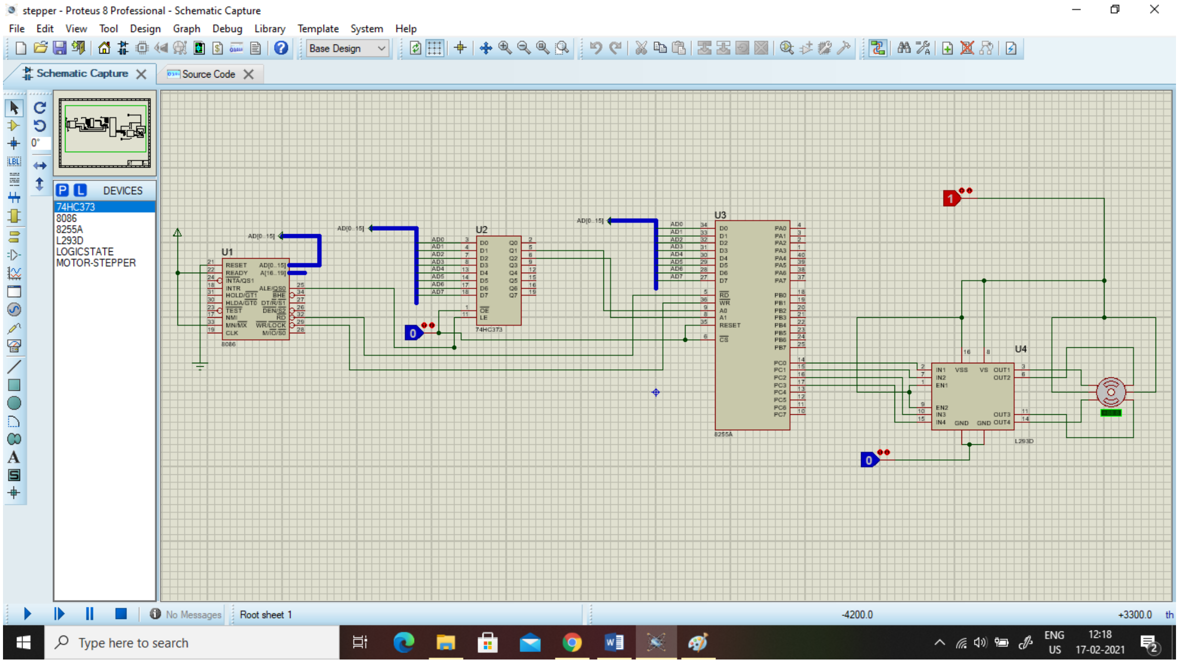
Step 6: Then Click Finish



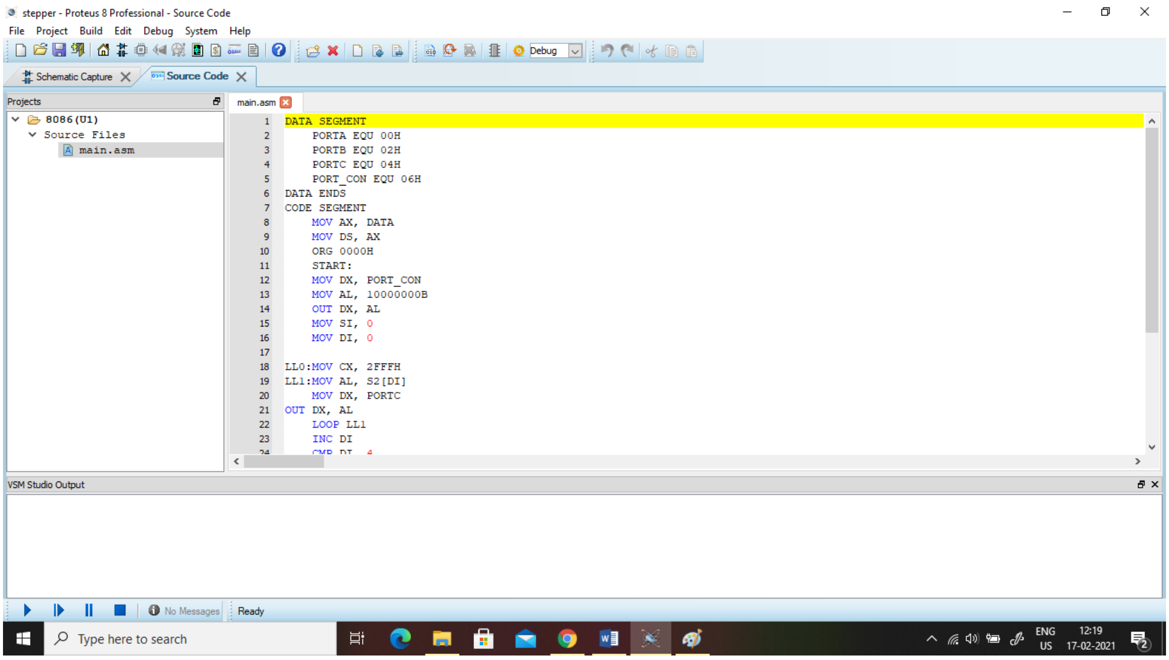
Step 7: You will get the following screen and Click on Schematic Capture.



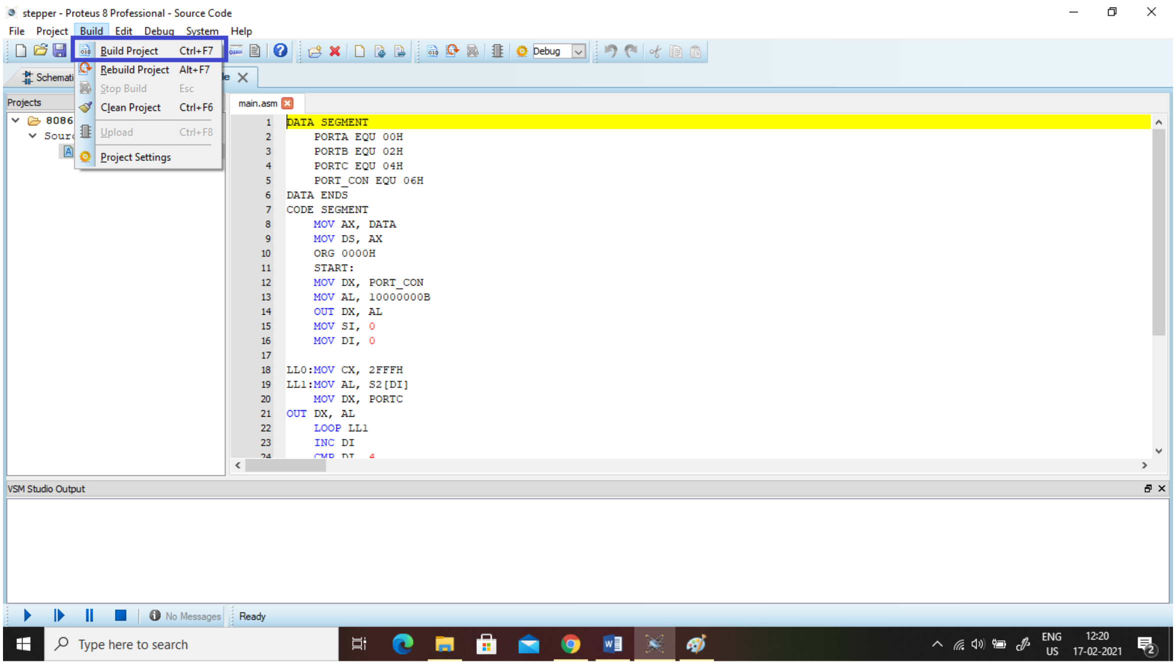
Step 8: Now draw the schematic as shown in Figure 2 as follows.



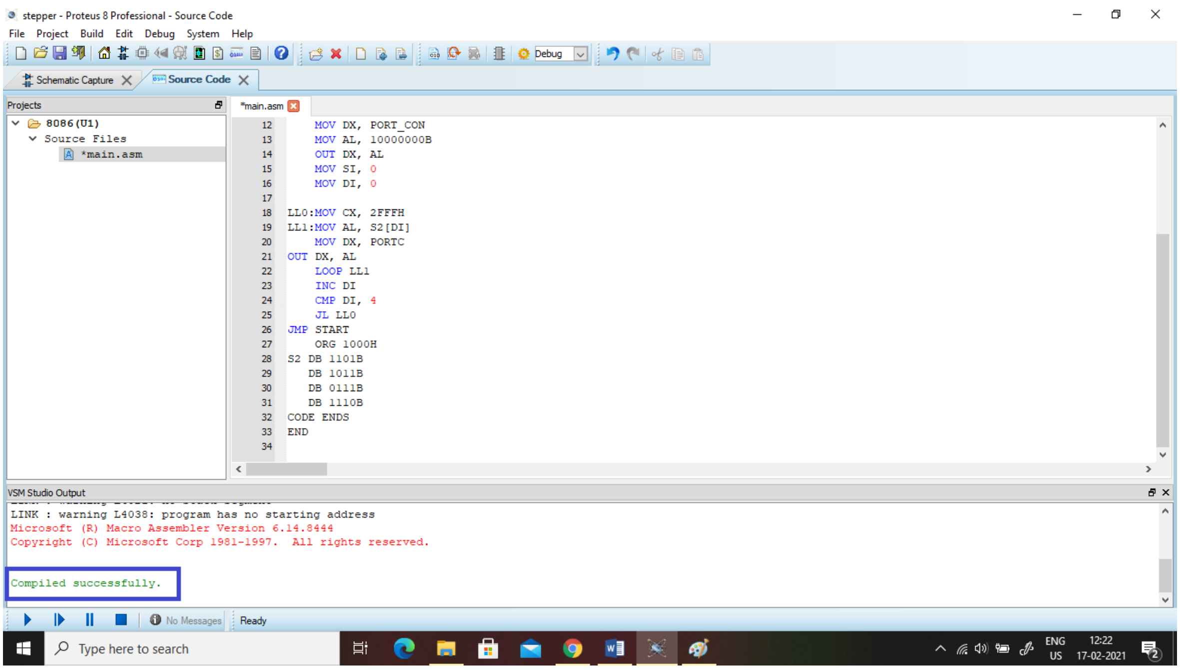
Step 9: Enter the assembly program in Source Code Menu as shown below.



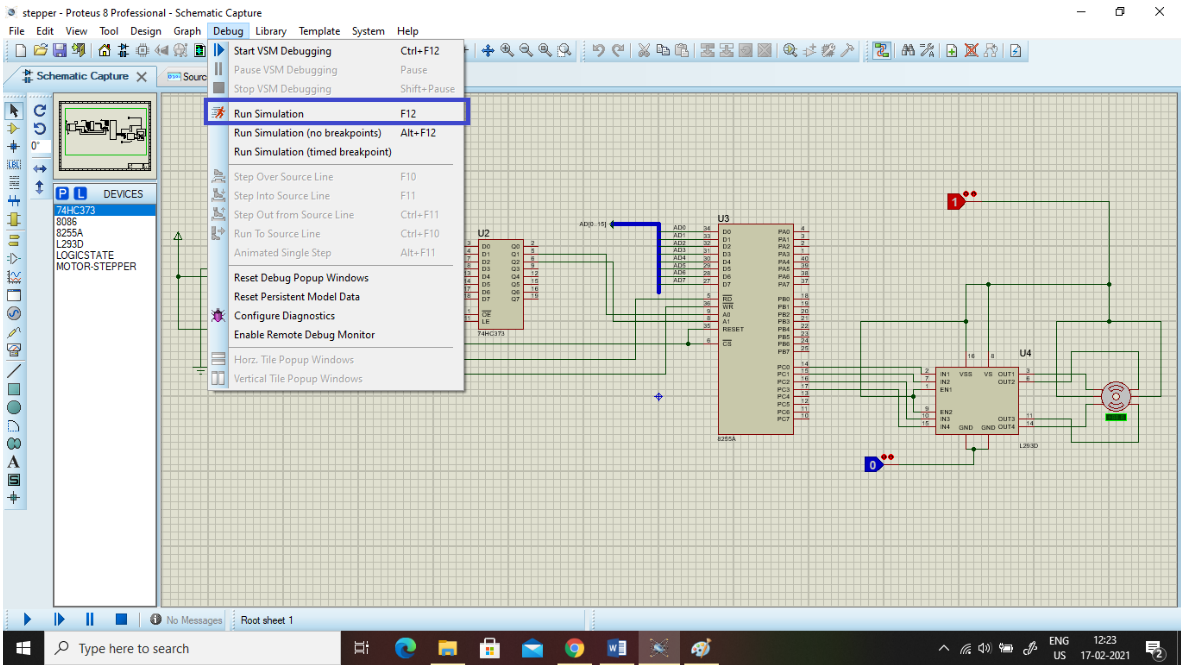
Step 10: Click on the Build and Build Project



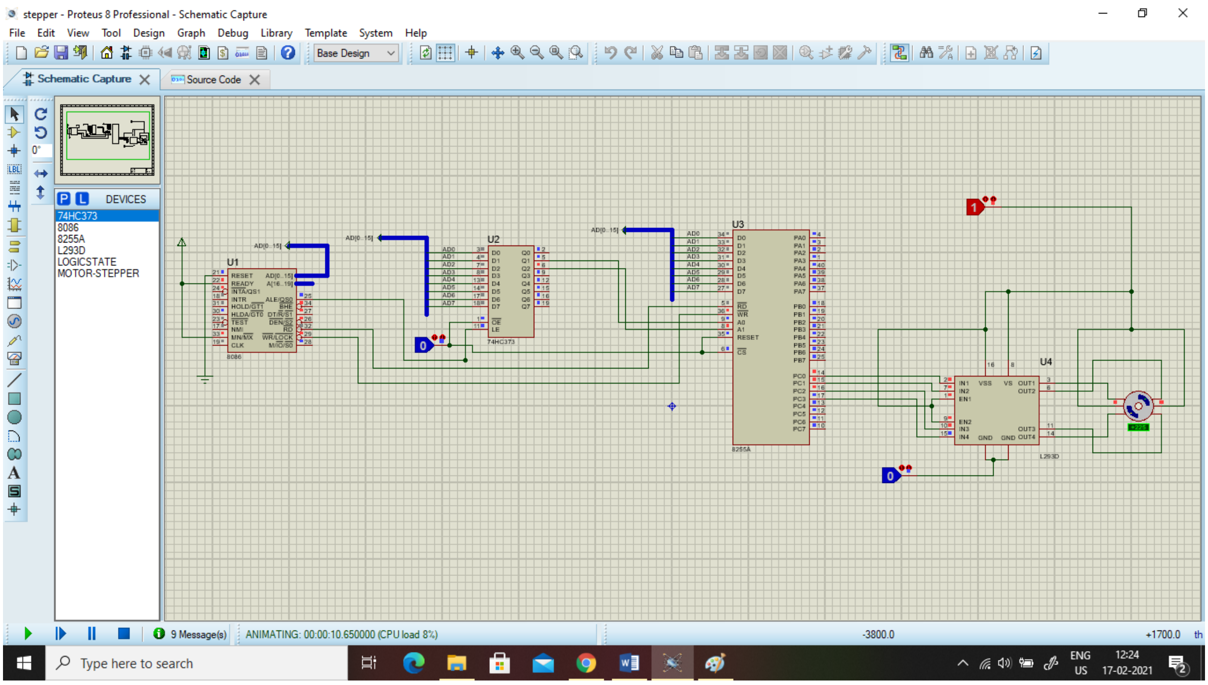
You can see the Build message in the bottom as shown



Step 11: Now in schematic Capture click on Debug and Run Simulation



You can observe the stepper motor rotating.



EXPERIMENT-9

HARDWARE INTERFACING – STEPPER MOTOR

Aim: To Interface a stepper motor to 8086 Microprocessor through 8255 Programmable Peripheral Interface (PPI).

Equipment's: PC with Linux operating system Loaded
8255 Interfacing card (PCI Card)
Stepper Motor Driver board
Stepper Motor

Theory:

This experiment is done through the PC interface. 8255 device is mounted on PCI card and is fixed into the PCI slot of the PC. We are considering the entire PC as 8086 Microprocessor with 8255 PPI connected to it. As usual program is written using dos editor and can be executed using TASM or MASM.

Procedure:

- The PCIDIOT card is already connected to PCI slot of the PC

- Connect the FLAT cable coming out of the PCI card to the stepper motor driver board
- Connect the stepper motor connection to the driver board. Proper care should be taken while doing connection as wrong connection may spoil the entire board. The connection slots have to be properly identified.
- Connect the power supply to the driver board

- Call any one of the Lab Instructors and ensure the connections are correct before switching on the power supply

- Switch on the PC and Load Ubuntu Linux
- Login as **Micro** and password is ***micro***
- Open the **Terminal** window in Ubuntu Linux
- Run below command

  sudo virt-manager
- Enter the password : ***micro***
- Load **Windows XP** through Virtual manager, select user **micro1**
- Open **C Drive -> micro2** folder
- Right click on below files and select **Run as Administrator**, Password: **micro**
  1. **chkdiot**
  2. **iopm**

- Close the above window and open the **command prompt**
- Go to the micro2 directory by typing

  > **cd\micro2**
- Open editor window to type your program

  edit *filename*.asm (Give filename as last 4 digit of your id)
- Save and exit from editor window
- Compile the program using **MASM/LINK** command and generate the executable file **(.exe) file.**

  masm *filename*.asm (press Enter key 3 times)

  link *filename*.obj (press Enter key 3 times**)**
- Run the following command for executing the .exe file and see the stepper motor rotating

  iopm *filename*.exe 0xc260 0xc263
- The Program shown below will rotate the stepper motor by 360 degrees i.e one full round.

Assembly language program to rotate the Stepper Motor

```
. MODEL SMALL     ; Specify the model for the executable. Must for every program
. STACK 100h
.DATA
CR EQU 0c263H     ; User must change the port addresses as assigned by the PC.
PA EQU 0c260H
PB EQU 0c261H
PC EQU 0c262H
Message1    DB 'DEMONSTRATION PROGRAM FOR STEPPER MOTOR',13,10,'$'
Message2    DB    13,10,'The program is running...',13,10,'$'
. CODE
START:
MOV AX, @DATA
MOV DS, AX
MOV AH,9h  ; Display the message line1.
MOV DX, OFFSET Message1
INT 21h
MOV AH,9h  ; Display the message line3.
```

```
MOV DX, OFFSET Message2
INT 21h
MOV DX, CR
MOV AL,80h
OUT DX, AL
MOV BL,50
begin:  MOV  AL,11h ; To rotate in opposite direction, change the data as 88H
                    ; instead of 11H
CALL  OUT_A
CALL  DELAY
MOV AL,22h ; To rotate in opposite direction, change the data as 44H instead of 22H
CALL  OUT_A
CALL  DELAY
MOV AL,44h ; To rotate in opposite direction, change the data as 22H instead of 44H
CALL  OUT_A
CALL  DELAY
MOV AL,88h; To rotate in opposite direction, change the data as 11H instead of 88H
CALL  OUT_A
CALL DELAY
DEC Bl
JNZ begin
MOV AH, 4CH
INT 21H
OUT_A: MOV DX, PA
OUT DX, AL
RET
DELAY: MOV CX, 0FFFH
D2: MOV AX, 05FFH
D1: DEC AX
JNZ D1
DEC CX
JNZ D2
RET
END START
```

- Now change the program to rotate the stepper motor in reverse direction.
- Change the program to rotate the motor for 3 rounds in any direction.

EXPERIMENT-10A
HARDWARE INTERFACING – ELEVATOR

Aim:

To Interface an elevator to 8086 Microprocessor through 8255 Programmable Peripheral Interface (PPI).

Equipment's: PC, 8255 Interfacing card (PCI Card) Elevator Interface

Objective:

To show the operation of the elevator as follows:

Initially, the elevator is at ground floor, When the elevator reaches any floor, it stays at that floor until a request from another floor is made. When such a request is detected, it moves to that floor. The floor request are scanned in fixed order i.e. floors 0, 1, 2 and 3.

Theory:

This experiment is done through the PC interface. 8255 device is mounted on PCI card and is fixed into the PCI slot of the PC. We are considering the entire PC as 8086 Microprocessor with 8255 PPI connected to it. As usual program is written using dos editor and can be executed using TASM or MASM.

This interface simulates the control and operation of an elevator. Four floors assumed and for each floor a key and corresponding LED indicator are provided to serve as request buttons and request status indicator. The elevator itself is represented by a column of ten LEDs. The motion of elevator can be simulated by turning on successive LEDs one at a time. The delay between turning off one LED and turning on the next LED can simulate the "speed" of the elevator. User can read the request status information through one port, reset the request indicators through another port and control the elevator (LED column) through another port.

This interface has four keys, marked 0, 1, 2, and 3 representing the request buttons at the four floors. Pressing of a key causes a corresponding Flip-Flop to be set. The outputs of the four Flip-Flop can be read through port B (PBO, PBI, PB2 and PB3). Also, the status of these signals is reflected by a set of 4 LEDs. The Flip-Flop can be rest (LEDs are cleared) through port A (PA54, PA5, PA6, and PA7). A column of 10 LEDs, representing the elevator can be controlled through Port A (PA0, PA1, PA2 and PA3). These port lines are fed to the inputs of the decoder 7442 whose outputs are used to control the on/off states of the LEDs which simulate the motion of the elevator.

Procedure:

- The PCIDIOT card is already connected to PCI slot of the PC

- Connect the FLAT cable coming out of the PCI card to the elevator interface

- Switch on the PC and Load Ubuntu Linux
- Login as **Micro** and password is ***micro***
- Open the **Terminal** window in Ubuntu Linux
- Run below command

  sudo virt-manager
- Enter the password : ***micro***
- Load **Windows XP** through Virtual manager, select user **micro1**
- Open **C Drive -> micro2** folder
- Right click on below files and select **Run as Administrator**, Password: **micro**
  1. **chkdiot**
  2. **iopm**
- Close the above window and open the **command prompt**
- Go to the micro2 directory by typing

  > **cd\micro2**
- Open editor window to type your program

  edit *filename*.asm (Give filename as last 4 digit of your id)
- Save and exit from editor window
- Compile the program using **MASM/LINK** command and generate the executable file **(.exe) file.**

  masm *filename*.asm (press Enter key 3 times)

  link *filename*.obj (press Enter key 3 times**)**
- Run the following command for executing the .exe file

  iopm *filename*.exe 0xc260 0xc263
- Now enter the buttons at any floor

Assembly language program for Elevator interface to ESA PCI-DIOT

```
. MODEL SMALL; Specify the model for the executable. Must for every program.
. STACK 5000h
.DATA              ; Any data declarations here.
Message1 DB 'DEMONSTRATION PROGRAM FOR ELEVATOR
INTERFACE',13,10,'$'
Message2 DB 'Press the switches on the interface and see what happens.',13,10,'$'
Message3 DB 'This program is running...',13,10,'Press any key to EXIT.',13,10,'$'
Delay Rate DW 04FFH
CR EQU 0c263h
PA EQU 0c260h
PB EQU 0c261h
PC EQU 0c262h
FCODE DB 00h,03h,06h,09h
FCLR DB 0E0h, 0D3h, 0B6h, 79h

. CODE                  ; Start your coding here.
MOV AX, @DATA           ; Initialize all segment registers as needed here.
MOV DS, AX
MOV AH,9h               ; Display the message line1.
MOV DX,                 OFFSET Message1
INT 21h
MOV AH,9h               ; Display the message line2.
MOV DX,                 OFFSET Message2
INT 21h
MOV AH,9h               ; Display the message line3.
MOV DX,                 OFFSET Message3
INT 21h
MOV DX, CR
MOV AL,082h             ; Port A input Port B output
OUT DX, AL
XOR AX, AX
LOOP1:
MOV AL, AH
```

```
OR AL,0F0H
MOV DX, PA
OUT DX, AL
MOV DX, PB
LOOP2:
MOV CH, AH
MOV AH,01H
INT 16H
JNZ EXITP
MOV AH, CH
IN AL, DX
AND AL,0FH
CMP AL,0FH
JZ LOOP2
MOV SI,00H
FINDF:
ROR AL,01H
JNC FOUND
INC SI
JMP SHORT FINDF
FOUND:
MOV AL, FCODE[SI]
CMP AL, AH
JA GOUP
JB GODN
CLEAR:MOV AL, FCLR[SI]
MOV DX, PA
OUT DX, AL
JMP SHORT LOOP1
GOUP:
CALL DELAY
INC AH
XCHG AL, AH
OR AL,0F0H
MOV DX, PA
OUT DX, AL
```

```
AND AL,0FH
XCHG AH, AL
CMP AL, AH
JNZ GOUP
JMP SHORT CLEAR
GODN:
CALL DELAY
DEC AH
XCHG AH, AL
OR AL,0F0H
MOV DX, PA
OUT DX, AL
AND AL,0FH
XCHG AL, AH
CMP AL, AH
JNZ GODN
JMP SHORT CLEAR
MOV AH,4CH
NT 21H
DELAY:
PUSH CX
PUSH AX
MOV CX,0FFFFH
LOOP3: MOV AX,0AFFH
LOOP4: DEC AX
JNZ LOOP4
LOOP LOOP3
POP AX
POP CX
RET
EXITP:
MOV AH,4CH
INT 21H
END
```

The program will be in the loop and to come out, press ENTER key on the trainer.

EXPERIMENT-10B

HARDWARE INTERFACING – TRAFFIC LIGHT

Aim: To Interface a traffic light to 8086 Microprocessor through 8255 Programmable Peripheral Interface (PPI).

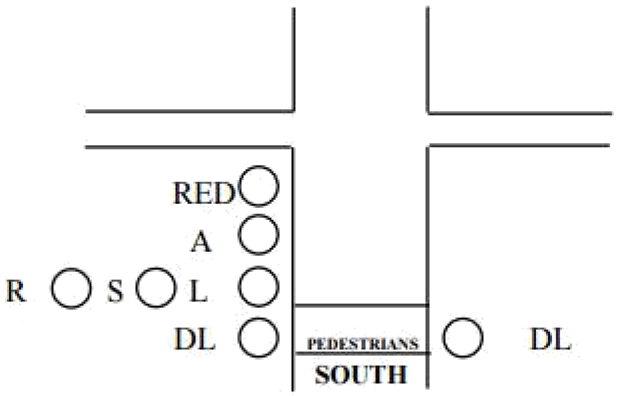
Equipment's: PC, 8255 Interfacing card (PCI Card) Traffic Light Interface

Theory:

This experiment is done through the PC interface. 8255 device is mounted on PCI card and is fixed into the PCI slot of the PC. We are considering the entire PC as 8086 Microprocessor with 8255 PPI connected to it. As usual program is written using dos editor and can be executed using TASM or MASM

The traffic light interface simulates the control and operation of traffic lights at a junction of four roads. The interface provides a set of 6 LED indicators at each of the four corners.

The LEDs at SOUTH-WEST corner are organized as follows:



RED: RED

A: AMBER

L: LEFT

S: STRAIGTH

R: RIGHT

DL: PEDESTRIAN (DL is a set of 2 dual-colour LEDs)

Of these, the first five LEDs will be ON or OFF depending on the state of the corresponding port line (LED is ON if the port line is Logic HIGH and LED is OFF if the port line is Logic LOW). The last one marked as DL is a set of two dual-colour LEDs and they both will be either RED or GREEN depending on the state of the corresponding port line (RED if the port line is Logic HIGH and GREEN if the port line is Logic LOW).

There are four sets of such LEDs and are controlled by 24 port lines. The 24 LEDs and their corresponding port lines are as follows

The 24 LEDs and their corresponding port lines are summarized below

	LED	Port Line
SOUTH	RED	PA3
	AMBER	PA2
	LEFT	PA0
	STRAIGHT	PC3
	RIGHT	PA1
	PEDESTRIAN	PC6
EAST	RED	PA7
	AMBER	PA6
	LEFT	PA4
	STRAIGHT	PC2
	RIGHT	PA5
	PEDESTRIAN	PC7
NORTH	RED	PB3
	AMBER	PB2
	LEFT	PB0
	STRAIGHT	PC1
	RIGHT	PB1
	PEDESTRIAN	PC4
WEST	RED	PB7
	AMBER	PB6
	LEFT	PB4
	STRAIGHT	PC0
	RIGHT	PB5
	PEDESTRIAN	PC5

Vehicles coming from one direction are controlled by the LEDs at the opposite corner.

Procedure:

- The PCIDIOT card is already connected to PCI slot of the PC

- Connect the FLAT cable coming out of the PCI card to the traffic light interface

- Switch on the PC and Load Ubuntu Linux
- Login as **Micro** and password is ***micro***
- Open the **Terminal** window in Ubuntu Linux
- Run below command

  sudo virt-manager
- Enter the password : ***micro***
- Load **Windows XP** through Virtual manager, select user **micro1**
- Open **C Drive -> micro2** folder
- Right click on below files and select **Run as Administrator**, Password: **micro**
  1. **chkdiot**
  2. **iopm**
- Close the above window and open the **command prompt**

- Go to the micro2 directory by typing

  > **cd\micro2**
- Open editor window to type your program

  edit *filename*.asm (Give filename as last 4 digit of your id)
- Save and exit from editor window
- Compile the program using **MASM/LINK** command and generate the executable file **(.exe) file.**

  masm *filename*.asm (press Enter key 3 times)

  link *filename*.obj (press Enter key 3 times**)**
- Run the following command for executing the .exe file

  iopm *filename*.exe 0xc260 0xc263
- The Program shown below will simulate traffic movement sequence. Now press "**,**" to change the traffic conditions.
- The sequence of traffic movement is listed in APPENDIX.

TRAFFIC LIGHT Interface program uses DOS Interrupts

```
. MODEL SMALL; Specify the model for the executable. Must for every program.
. STACK 5000H
.DATA                       ; any data declarations here.
Message1 DB 'DEMONSTRATION PROGRAM FOR TRAFFIC LIGHT INTERFACE' ,13, 10,'$'
Message2 DB 'This program will change LED signals when "," key is pressed on the
            keyboard.' 13, 10,'$'
Message 3 DB 'This program is running..',13,10,'Press "Enter" key to EXIT.',13,10,'$'
CMD_PORT EQU 0c263H
PORT_AEQU 0c260H
PORT_BEQU 0c261H
PORT_CEQU 0c262H
. CODE                      ; Start your coding here.
MOV AX, @DATA               ; Initialize all segment registers as needed here.
MOV DS, AX
MOV AH,9H                   ; Display the message line1.
MOV DX, OFFSET Message1
INT 21H
```

```
MOV AH,9H                   ; Display the message line2.
MOV DX, OFFSET Message2
INT 21H
MOV AH,9H                   ; Display the message line3.
MOV DX, OFFSET Message3
INT 21H

START:
MOV AL, 80H                 ; Initializing of ports port A, B and C as o/p
MOV DX, CMD_PORT
OUT DX, AL

AGAIN:
MOV CX, 05H
MOV SI, OFFSET PORT         ; store ports address in SI reg

NEXTST:
MOV AL, CS: [SI]
MOV DX, PORT_A
OUT DX, AL                  ; out port the data through port A
INC SI                      ; increment to next port address
INC DX
MOV AL, CS: [SI]
OUT DX, AL                  ; out port the data through port B
INC SI                      ; increment to next port address
INC DX
MOV AL, CS: [SI]
OUT DX, AL                  ; out port the data through port C
INC SI
PUSH SI
PUSH CX
WSER:
NOP                         ; Keyboard mode
PUSH AX
```

```
MOV AH, 0H                ; read key "," for increment to next data
INT 16H
CMP AL,','
JNE WSER
POP AX                    ; Sequence for turning ON; AMBER LED
POP CX
POP     SI
MOV AL, CS: [SI]
MOV DX, PORT_A
OUT     DX, AL
INC     SI
INC     DX
MOV     AL, CS: [SI]
OUT     DX, AL
INC     SI
INC     DX
MOV     AL, CS: [SI]
OUT     DX, AL
INC     SI
CALL    DELAY             ; call for delay routine
PUSH AX
MOV     AH, 0H
INT     16H
CMP     AL, 0DH
JNE     L1
MOV     AX, 4C00H
INT     21H
L1:
POP AX
LOOP NEXTST
JMP     AGAIN
DELAY: MOV BL, 0FH        ; Delay routine
PUSH CX
DLY5:
MOV CX, 1FFFH
```

```
DLY10:
NOP
LOOP    DLY10
DEC BL
JNZ DLY5
POP CX
RET
PORTS:
DB88H, 83H, 0F2H                ; STATE      1
DB 88H, 87H, 0F2H               ; ALL AMBERS ON
DB 38H, 88H, 0F4H               ; STATE      2
DB 78H, 88H, 0F4H               ; ALL AMBERS ON
DB 83H, 88H, 0F8H               ; STATE      3
DB 87H, 88H, 0F8H               ; ALL AMBERS ON
DB 88H, 38H, 0F1H               ; STATE      4
DB 88H, 78H, 0F1H               ; ALL AMBERS ON
DB 88H, 88H, 00H                ; STATE      5
DB 88H, 88H, 00H                ; ALL AMBERS ON
END
```

APPENDIX

The sequence shown below is simulated:

Vehicles from SOUTH can go NORTH, EAST and WEST.

Vehicles from WEST can go NORTH, SOUTH and EAST.

Vehicles from NORTH can go SOUTH, WEST and EAST.

Vehicles from EAST can go WEST, NORTH and SOUTH.

Pedestrians can cross on all roads.

The system stays in one state until user types " ," as explained in the programs. Then it moves into the next state. After the last state, the system again moves to the first state. AMBER LED is set ON and then OFF (after suitable delay), at the appropriate direction when the corresponding red LED changes from OFF to ON state.