# INTERFACING OF RTC AND ADC WITH PIC16F877A MICROCONTROLLER

**A PROJECT REPORT**

*Submitted by*

| | |
|---|---|
| **SEEVAGAN S D** | **2020505026** |
| **SOWMY NARAYANAN** | **2020505029** |
| **SRIVATSAN S** | **2020505030** |
| **SUJEETH S** | **2020505031** |

*in partial fulfillment for the award of the degree of*

**BACHELOR OF ENGINEERING**

***in***

**ELECTRONICS AND INSTRUMENTATION ENGINEERING**

**DEPARTMENT OF INSTRUMENTATION ENGINEERING**
**MADRAS INSTITUTE OF TECHNOLOGY CAMPUS**
**ANNA UNIVERSITY: CHENNAI-600 044**
**DECEMBER 2022**

**MADRAS INSTITUTE OF TECHNOLOGY CAMPUS**
**ANNA UNIVERSITY: CHENNAI 600 044**

## BONAFIDE CERTIFICATE

This is to certify that the project report entitled" **Interfacing of RTC and ADC using PIC16f877A microcontroller"** is a bonafide work of Seevagan S D (2020505026),Sowmy Narayanan S (2020505029), SrivatsanS (2020505030), Sujeeth S (2020505031) carried out under my guidance.

| | |
|---|---|
| **SIGNATURE** | **SIGNATURE** |
| **Dr. S. SRINIVASAN** | **Dr. S. MEYYAPPAN** |
| Professor And Head, | Asst. Prof (Sr. G) and Guide |
| Dept. Of Instrumentation Engg. | Dept. Of Instrumentation Engg. |
| Madras Institute of Technology, | Madras Institute of Technology, |
| Anna University, | Anna University, |
| Chromepet, Chennai – 600044. | Chromepet, Chennai – 600044 |

## ACKNOWLEDGEMENT

We sincerely thank **Dr. J. Prakash, Dean, MIT Campus, Anna University** for providing facilities to complete the project.

We express our gratitude to **Dr. S. Srinivasan, Head of The Department, Department of Instrumentation Engineering, MIT Campus, Anna University** for supporting us to complete the project.

We sincerely express our thankfulness to **Dr. S. Meyyappan, Asst. Professor (Sr.Gr.), Department of Instrumentation Engineering, MIT Campus, Anna University,** and our guide for providing complete guidance throughout the project.

## ABSTRACT

The main aim of this project is to interface RTC and ADC with microcontroller and display both the time from the RTC and the value from ADC with 16x2 LCD module. The main components used in making this prototype are PIC16F877A, DS1307 and 16x2 LCD Module. The software application used is MPLABX and for simulation PROTEUS is used.

# TABLE OF CONTENTS

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 OVERVIEW:

Time is the most essential non-physical quantity which brings meaning and quality of life which we live. Since ancient times, measurement of time was a important task. The overview of this project is to make a digital clock using modern devices. We are also aiming to show some analogue value along with the time measured digitally.

## 1.2 OBJECTIVE

- To interface RTC and ADC using PIC16F877A microcontroller.
- To display the time and digital value using 16 x 2 LCD display

## 1.3 ORGANISATION OF THE REPORT

- Chapter 1 focus on the objective of the project and a brief introduction.
- Chapter 2 shows the specifications of the hardware components used in the project. It also talks about the software that has been implemented in this project.
- Chapter 3 presents the protocols and registers use in this project.
- Chapter 4 presents the experimentation of the project.
- Chapter 5 presents the references.

# CHAPTER 2
# SPECIFICATION

The motive of the project is the interface the DS1307(RTC) and In-Built ADC of PIC16F877A microcontroller and display time and digital value using 16 x 2 LCD display. The output is implemented in such a way that the time is displayed in the first line and the digital value from the ADC is displayed in the second line of the LCD display.

## 2.1 HARDWARE REQUIREMENTS

The materials required for the implementation of the project are:

### 2.1.1 PIC16F877A microcontroller

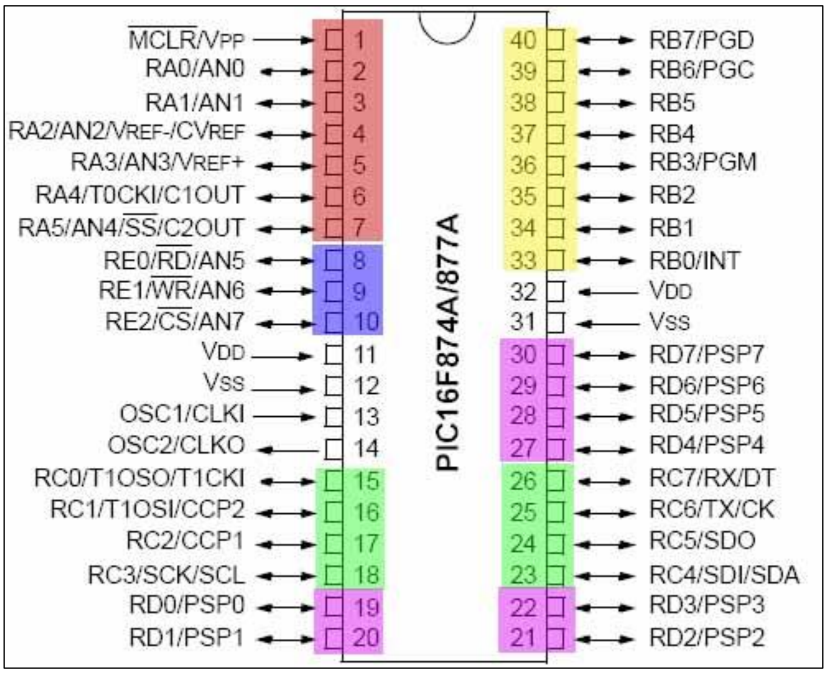


**Figure 2.1.1** ***PIC16F877A Pin diagram.***

PIC is a family of microcontrollers made by Microchip Technology, derived from the PIC1650 originally developed by General Instrument's Microelectronics Division. The name PIC initially referred to Peripheral Interface Controller, and is currently expanded as Programmable Intelligent Computer. The first parts of the family were available in 1976, by 2013 the company had shipped more than twelve billion individual parts, used in a wide variety of embedded systems. The manufacturer supplies computer software for development known as MPLAB X, assemblers and C/C++ compilers, and programmer/debugger hardware under the MPLAB and PIC Kit series. Third party and some opensource tools are also available. PIC devices are popular with both industrial developers and hobbyists due to their low cost, wide availability, large user base, an extensive collection of application notes, and availability of low cost or free development tools, serial programming, and re-programmable flash-memory capability. The PIC16F877A features 256 bytes of EEPROM data memory, self-programming, an ICD, 2 Comparators, 8 channels of 10-bit Analog-to-Digital (A/D) converter, 2 capture/compare/PWM functions, the synchronous serial port can be configured as either 3-wire Serial Peripheral Interface (SPI) or the 2-wire Inter-Integrated Circuit ($I^2C$) bus and a Universal Asynchronous Receiver Transmitter (USART). In this project the synchronous serial port has been configured as $I^2C$ bus.

**Ports in PIC16F877a**:

• Port A has 8 Pins in total and it is an analogue Port.

• Port B also has 8 Pins but these all are digital Pins.

• Port C is also a digital Port having 8 Pins and are also used for Serial Communication.

• Port D has 8 Pins and all are digital Pins.

• Port E has 3 Pins, they are read (RD), write (WR) and control signal (CS) pins.

**Peripheral Features**:

• Timer0: 8-bit timer/counter with 8-bit prescaler

• Timer1: 16-bit timer/counter with prescaler, can be incremented during Sleep via external crystal/clock

• Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler

• Two Capture, Compare, PWM modules – Capture is 16-bit, max. Resolution is 12.5 ns – Compare is 16-bit, max. Resolution is 200 ns – PWM max. Resolution is 10-bit

• Synchronous Serial Port (SSP) with SPI (Master mode) and I²C (Master/Slave)

• Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection

• Parallel Slave Port (PSP) – 8 bits wide with external RD, WR and CS controls (pin 40 only)

• Brown-out detection circuitry for Brown-out Reset (BOR)

**Analog Features:**

• 10-bit, up to 8-channel Analog-to-Digital Converter (A/D)

• Brown-out Reset (BOR)

• Analog Comparator module with:

- Two analogue comparators
- Programmable on-chip voltage reference (VREF) module
- Programmable input multiplexing from device inputs and internal voltage reference
- Comparator outputs are externally accessible

**Special Microcontroller Features:**

• 100,000 erase/write cycle Enhanced Flash program memory typical

• 1,000,000 erase/write cycle Data EEPROM memory typical

• Data EEPROM Retention > 40 years

• Self-reprogrammable under software control

• In-Circuit Serial Programming™ (ICSP™) via two pins

• Single-supply 5V In-Circuit Serial Programming

• Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation

• Programmable code protection

• Power saving Sleep mode

• Selectable oscillator options

• In-Circuit Debug (ICD) via two 4 pins

**CMOS Technology:**

• Low-power, high-speed Flash/EEPROM technology

• Fully static design

• Wide operating voltage range (2.0V to 5.5V)

• Commercial and Industrial temperature ranges

• Low-power consumption

## 2.1.2 REAL TIME CLOCK (RTC):

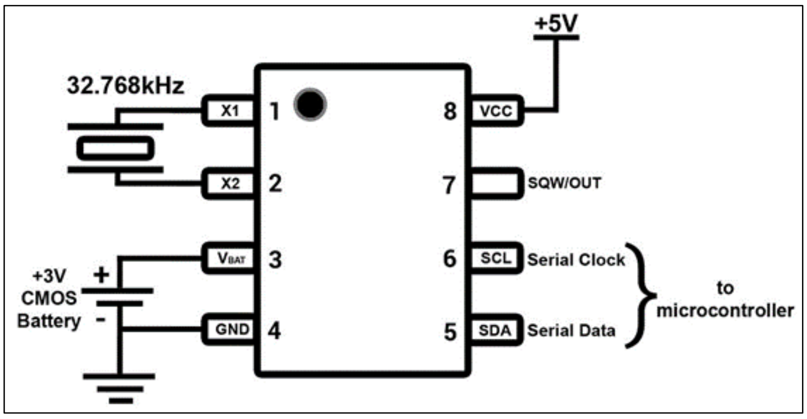


**Figure 2.1.2(a)** ***DS1307 pin diagram.***

A real-time clock (RTC) is a computer clock, usually in the form of an integrated circuit that is solely built for keeping time. Naturally, it counts hours, minutes, seconds, months, days and even years. RTCs can be found running in personal computers, embedded systems and servers, and are present in any electronic device that may require accurate time keeping. Being able to still function even when the computer is powered down through a battery or independently from the system's main power is fundamental. The RTC used in this project is DS1307.

**DS1307 GENERAL DESCRIPTION**

• The DS1307 serial real-time clock (RTC) is a low power, full binary coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM.

• Address and data are transferred serially through an I²C, bidirectional bus.

• The clock/calendar provides seconds, minutes, hours, day, date, month, and year information.

• The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year.

• The clock operates in either the 24-hour or 12- hour format with AM/PM indicator.

• The DS1307 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply.

• Timekeeping operation continues while the part operates from the backup supply.

**Real Time Clock DS1307 RTC I²C Module**

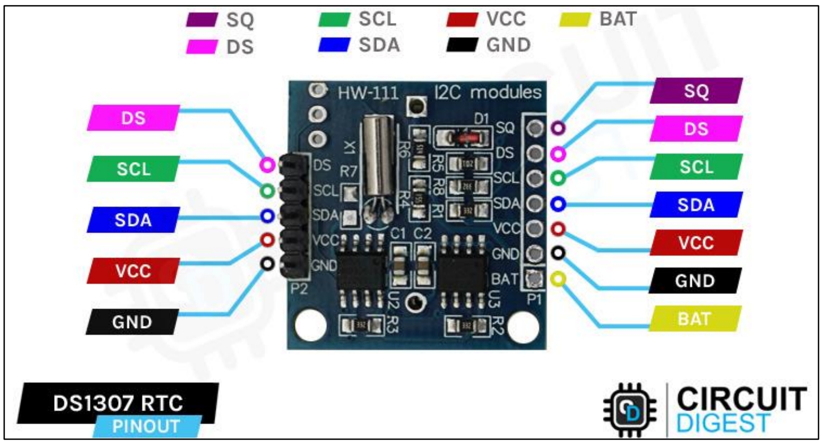


**Figure 2.1.2(b)** ***DS1307 I2C module.***

**Pin Configuration**

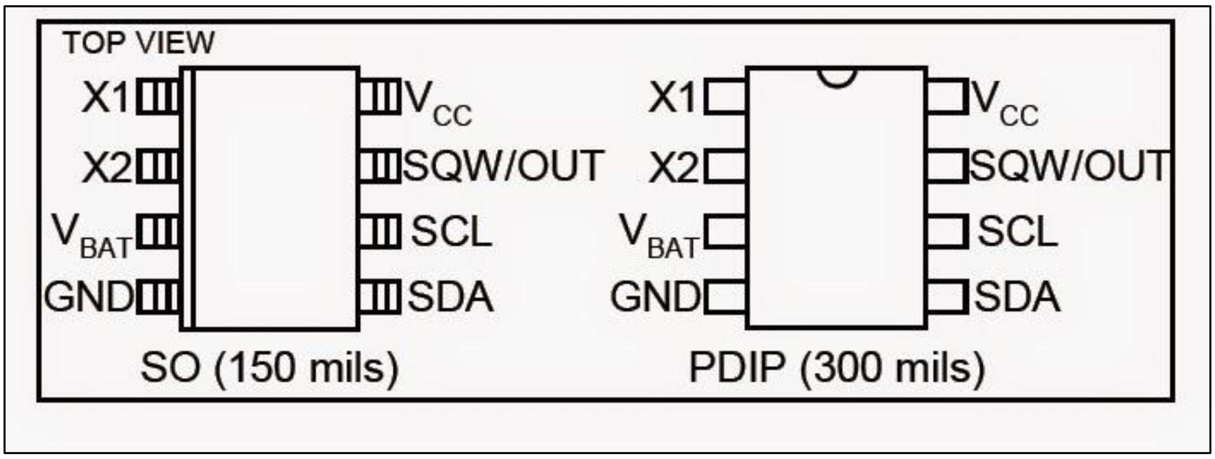


**Figure 2.1.2(c)** ***pin configuration.***

**BENEFITS AND FEATURES:**

• Completely Manages All Timekeeping Functions

♣ Real-Time Clock Counts Seconds, Minutes, Hours, Date of the Month, Month, Day of the Week, and Year with Leap Year Compensation Valid Up to 2100

♣ 56-Byte, Battery-Backed, General-Purpose RAM with Unlimited Writes

♣ Programmable Square-Wave Output Signal

• Simple Serial Port Interfaces to Most Microcontrollers: I²C Serial Interface

• Low Power Operation Extends Battery Backup Run Time

♣ Consumes Less than 500Na in Battery Backup Mode with Oscillator Running

♣ Automatic Power-Fail Detect and Switch Circuitry

• 8-Pin DIP and 8-Pin SO Minimizes Required Space

• Optional Industrial Temperature Range: -40°C to +85°C Supports Operation in a Wide Range of Applications.

### 2.1.3 LCD DISPLAY

**Features of LCD16x2**

The features of this LCD mainly include the following.

- The operating voltage of this LCD is 4.7V-5.3V
- It includes two rows where each row can produce 16-characters.
- The utilization of current is 1Ma with no backlight
- Every character can be built with a 5×8 pixel box
- The alphanumeric LCDs alphabets & numbers
- Is display can work on two modes like 4-bit & 8-bit

- These are obtainable in Blue & Green Backlight
- It displays a few custom generated characters

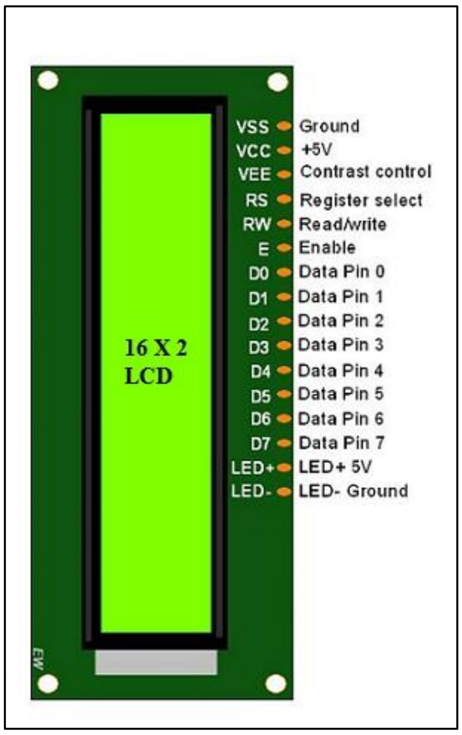


**Figure 2.1.3** ***16X2 LCD display***

**Registers of LCD**

A 16×2 LCD has two registers like data register and command register. The RS (register select) is mainly used to change from one register to another. When the register set is ‘0’, then it is known as command register. Similarly, when the register set is ‘1’, then it is known as data register.

**Command Register**

The main function of the command register is to store the instructions of command which are given to the display. So that predefined tasks can be performed such as clearing the display, initializing, set the cursor place, and display control. Here commands processing can occur within the register.

**Data Register**

The main function of the data register is to store the information which is to be exhibited on the LCD screen. Here, the ASCII value of the character is the information which is to be exhibited on the screen of LCD. Whenever we send the information to LCD, it transmits to the data register, and then the process will be starting there. When register set =1, then the data register will be selected.

## 2.2 SOFTWARE REQUIREMENT

### 2.2.1 MPLABX IDE

MPLAB is a proprietary freeware integrated development environment for the development of embedded applications on PIC and dsPIC microcontrollers, and is developed by Microchip Technology.

MPLAB X is the latest edition of MPLAB, and is developed on the NetBeans platform. MPLAB and MPLAB X support project management, code editing, debugging and programming of Microchip 8-bit PIC and AVR (including ATMEGA) microcontrollers, 16-bit PIC24 and dsPIC microcontrollers, as well as 32-bit SAM (ARM) and PIC32 (MIPS) microcontrollers.

MPLAB is designed to work with MPLAB-certified devices such as the MPLAB ICD 3 and MPLAB REAL ICE, for programming and debugging PIC microcontrollers using a personal computer. PICKit programmers are also supported by MPLAB.

MPLAB X supports automatic code generation with the MPLAB Code Configurator and the MPLAB Harmony Configurator plugins.

It is also used for dumping of c code to the pic microcontroller.

### 2.2.2 PROTEUS 8 PROFESSIONAL

Proteus 8 Professional is a software which can be used to draw schematics, PCB layout, code and even simulate the schematic. It is developed by LabCenter Electronic Ltd.

# CHAPTER 3

# PROTOCOLS AND REGISTERS

## 3.1 I2C PROTOCOL AND ITS OVERVIEW

I$^2$C stands for Inter-Integrated Circuit. It is a bus interface connection protocol incorporated into devices for serial communication. It was originally designed by Philips Semiconductor in 1982. Recently, it is a widely used protocol for short-distance communication. It is also known as Two Wired Interface (TWI).

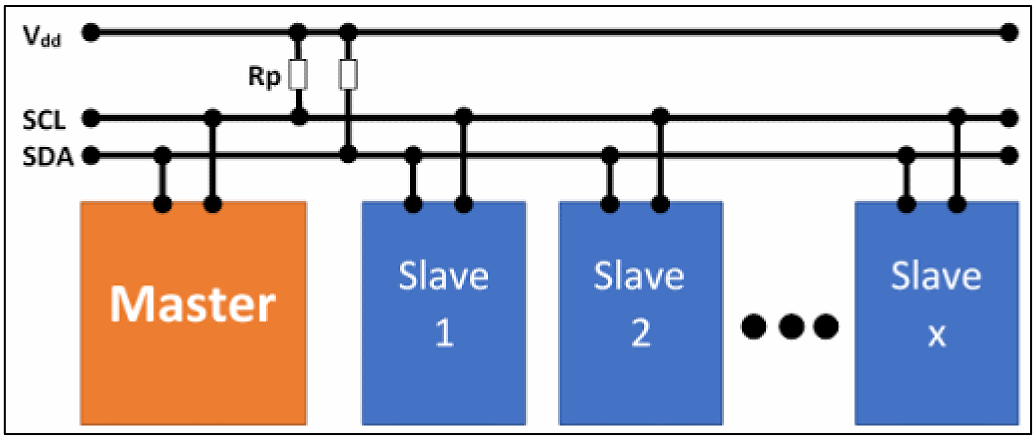


**Figure 3.1 (a)** ***I2C protocol***

It is a bidirectional two-wire serial bus that uses serial clock (SCL) and serial data (SDA) wires to send and manage data between devices connected to the bus. Serial Data (SDA) – Transfer of data takes place through this pin.

Serial Clock (SCL) – It carries the clock signal.

I$^2$C operates in 2 modes –

❖ Master mode

- Slave mode

Each data bit transferred on SDA line is synchronized by a high to the low pulse of each clock on the SCL line.

According to I²C protocols, the data line cannot change when the clock line is high, it can change only when the clock line is low. The 2 lines are open drain; hence a pull-up resistor is required so that the lines are high since the devices on the I²C bus are active low. The data is transmitted in the form of packets which comprises 9 bits.

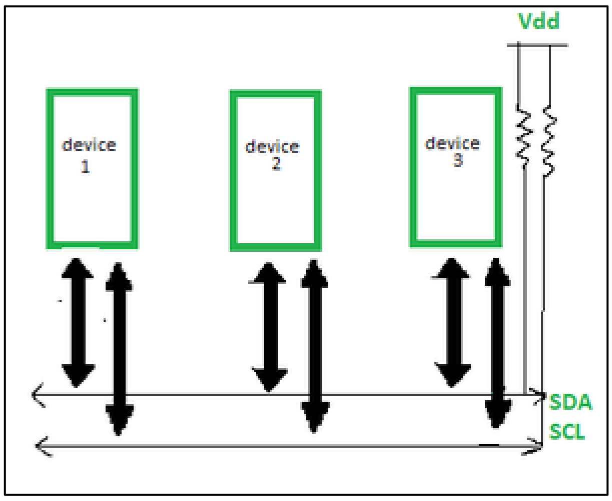


**Figure 3.1 (b)** ***I2C protocol with open drain system***

The sequence of these bits is –

- Start Condition – 1 bit
- Slave Address – 8 bits
- Acknowledge – 1 bit

**Start and Stop Conditions:**

START and STOP can be generated by keeping the SCL line high and changing the level of SDA. To generate START condition the SDA is changed from high to low while keeping the SCL high. To generate STOP condition SDA goes from low to high while keeping the SCL high, as shown in the figure below.

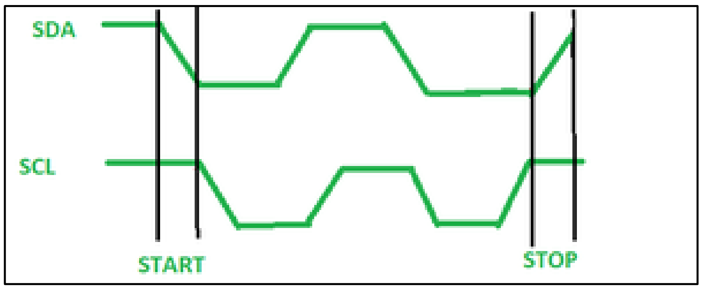


**Figure 3.1(c)** ***Start and Stop representation***

**REPEATED START:**

Between each start and stop condition pair, the bus is considered as busy and no master can take control of the bus. If the master tries to initiate a new transfer and does not want to release the bus before starting the new transfer, it issues a new START condition. It is called a REPEATED START condition.

**Read/Write Bit:**

A high Read/Write bit indicates that the master is sending the data to the slave, whereas a low Read/Write bit indicates that the master is receiving data from the slave.

**ACK/NACK Bit:**

After every data frame, follows an ACK/NACK bit. If the data frame is received successfully then ACK bit is sent to the sender by the receiver.

**Addressing:**

The address frame is the first frame after the start bit. The address of the slave with which the master wants to communicate is sent by the master to every slave connected with it. The slave then compares its own address with this address and sends ACK.

## I²C Packet Format:

In the I²C communication protocol, the data is transmitted in the form of packets. These packets are 9 bits long, out of which the first 8 bits are put in SDA line and the 9th bit is reserved for ACK/NACK i.e. Acknowledge or Not Acknowledge by the receiver.

**START** condition plus address packet plus one more data packet plus STOP condition collectively form a complete Data transfer.

## Features of I²C Communication Protocol:

- **Half-duplex Communication Protocol –**

Bi-directional communication is possible but not simultaneously.

- **Synchronous Communication –**

The data is transferred in the form of frames or blocks and  Can be configured in a multi-master configuration.

- **Clock Stretching –**

The clock is stretched when the slave device is not ready to accept more data by holding the SCL line low, hence disabling the master to raise the clock line. Master will not be able to raise the clock line because the wires are AND wired and wait until the slave releases the SCL line to show it is ready to transfer next bit.

- **Arbitration –**

I²C protocol supports multi-master bus system but more than one bus cannot be used simultaneously. The SDA and SCL are monitored by the masters. If the SDA is found high when it was supposed to be low it will be inferred that another master is active and hence it stops the transfer of data.

- **Serial transmission –**

I$^2$C uses serial transmission for transmission of data and Used for low-speed communication.

**Advantages:**

- Can be configured in multi-master mode.
- Complexity is reduced because it uses only 2 bi-directional lines (unlike SPI Communication).
- Cost-efficient.
- It uses ACK/NACK feature due to which it has improved error handling capabilities.

**Limitations:**

- Slower speed.
- Half-duplex communication is used in the I$^2$C communication protocol.

### 3.1.1 I$^2$C MODE IN PIC16F877A

The MSSP (**MASTER SYNCHRONOUS SERIAL PORT MODULE**) module in I$^2$C mode fully implements all master and slave functions (including general call support) and provides interrupts on Start and Stop bits in hardware to determine a free bus (multi-master function). The MSSP module implements the standard mode specifications, as well as 7-bit and 10-bit addressing.

Two pins are used for data transfer:

- Serial clock (SCL) – RC3/SCK/SCL
- Serial data (SDA) – RC4/SDI/SDA

**MSSP BLOCK DIAGRAM (I²C MODE):**

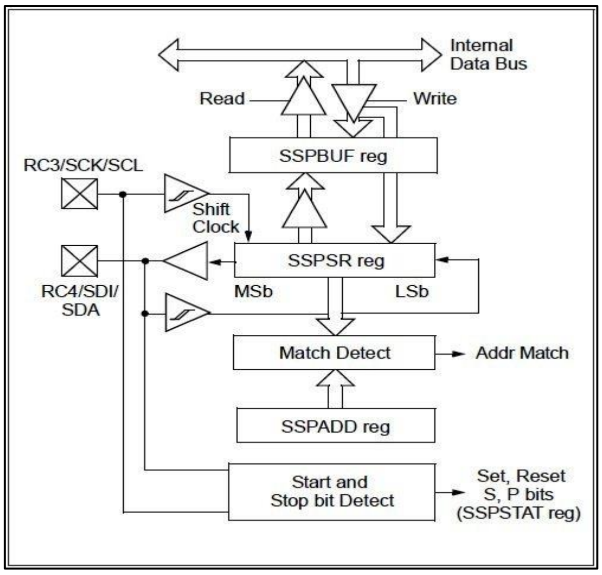


**Figure 3.1.1** ***MSSP BLOCK DIAGRAM***

## 3.2 ANALOG-TO-DIGITAL CONVERTER (A/D):

The Analog-to-Digital (A/D) Converter module has five inputs for the 28-pin devices and eight for the 40/44-pin devices.

The conversion of an analogue input signal results in a corresponding 10-bit digital number. The A/D module has high and low-voltage reference input that is software selectable to some combination of VDD, VSS, RA2 or RA3.

The A/D converter has a unique feature of being able to operate while the device is in Sleep mode. To operate in Sleep, the A/D clock must be derived from the A/D's internal RC oscillator.

The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register 0 (ADCON0)
- A/D Control Register 1 (ADCON1)

The ADCON0 register, controls the operation of the A/D module. The ADCON1 register, shown in Register 11-2, configures the functions of the port pins. The port pins can be configured as analogue inputs (RA3 can also be the voltage reference) or as digital I/O.

Additional information on using the A/D module can be found in the PICmicro® Mid-Range MCU Family Reference Manual (DS33023).

## 3.3 REGISTERS:

These are

- PIR1 Register
- ADCON0 Register
- ADCON1 Register
- MSSP Control Register (SSPCON)
- MSSP Control Register 2 (SSPCON2)
- MSSP Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer Register (SSPBUF)
- MSSP Shift Register (SSPSR) – Not directly accessible
- MSSP Address Register (SSPADD)

SSPCON, SSPCON2 and SSPSTAT are the control and status registers in $I^2C$ mode operation. The SSPCON and SSPCON2 registers are readable and writable. The lower six bits of the SSPSTAT are read-only. The upper two bits of the SSPSTAT are read/write. ¬

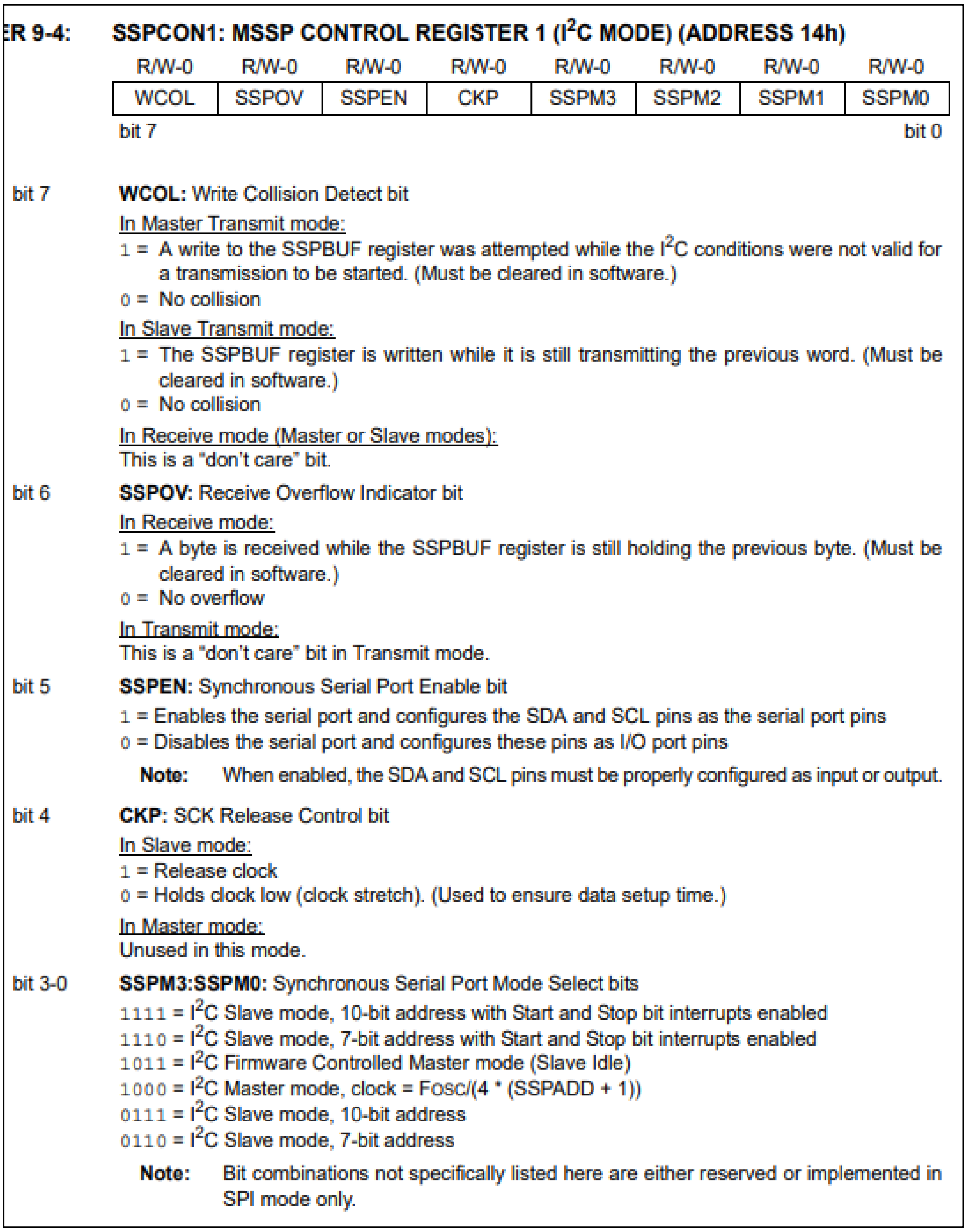
SSPSR is the shift register used for shifting data in or out. SSPBUF is the buffer register to which data bytes are written to or read from. ¬

SSPADD register holds the slave device address when the SSP is configured in I²C Slave mode. When the SSP is configured in Master mode, the lower seven bits of SSPADD act as the baud rate generator reload value. ¬

In receive operations, SSPSR and SSPBUF together create a doublebuffered receiver. When SSPSR receives a complete byte, it is transferred to SSPBUF and the SSPIF interrupt is set. ¬

During transmission, the SSPBUF is not double buffered. A write to SSPBUF will write to both SSPBUF

## 3.3.1 MSSP Control Register (SSPCON1)



**ER 9-4: SSPCON1: MSSP CONTROL REGISTER 1 ($I^2C$ MODE) (ADDRESS 14h)**

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| WCOL | SSPOV | SSPEN | CKP | SSPM3 | SSPM2 | SSPM1 | SSPM0 |
| bit 7 | | | | | | | bit 0 |

bit 7 **WCOL:** Write Collision Detect bit

In Master Transmit mode:

1 = A write to the SSPBUF register was attempted while the $I^2C$ conditions were not valid for a transmission to be started. (Must be cleared in software.)

0 = No collision

In Slave Transmit mode:

1 = The SSPBUF register is written while it is still transmitting the previous word. (Must be cleared in software.)

0 = No collision

In Receive mode (Master or Slave modes):

This is a "don't care" bit.

bit 6 **SSPOV:** Receive Overflow Indicator bit

In Receive mode:

1 = A byte is received while the SSPBUF register is still holding the previous byte. (Must be cleared in software.)

0 = No overflow

In Transmit mode:

This is a "don't care" bit in Transmit mode.

bit 5 **SSPEN:** Synchronous Serial Port Enable bit

1 = Enables the serial port and configures the SDA and SCL pins as the serial port pins

0 = Disables the serial port and configures these pins as I/O port pins

**Note:** When enabled, the SDA and SCL pins must be properly configured as input or output.

bit 4 **CKP:** SCK Release Control bit

In Slave mode:

1 = Release clock

0 = Holds clock low (clock stretch). (Used to ensure data setup time.)

In Master mode:

Unused in this mode.

bit 3-0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits

1111 = $I^2C$ Slave mode, 10-bit address with Start and Stop bit interrupts enabled

1110 = $I^2C$ Slave mode, 7-bit address with Start and Stop bit interrupts enabled

1011 = $I^2C$ Firmware Controlled Master mode (Slave Idle)

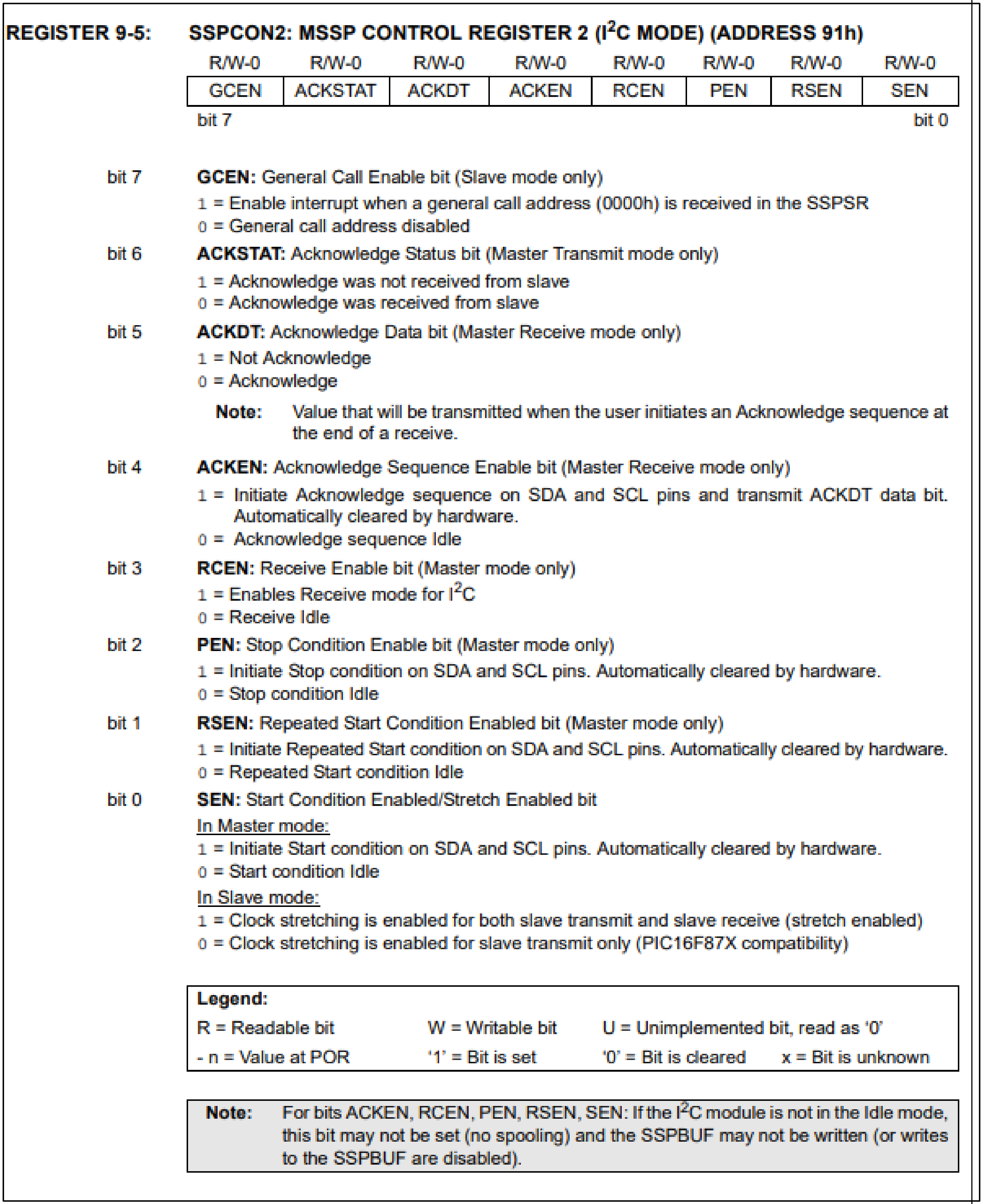
1000 = $I^2C$ Master mode, clock = FOSC/(4 * (SSPADD + 1))

0111 = $I^2C$ Slave mode, 10-bit address

0110 = $I^2C$ Slave mode, 7-bit address

**Note:** Bit combinations not specifically listed here are either reserved or implemented in SPI mode only.

### 3.3.2 MSSP Control Register (SSPCON2)



**REGISTER 9-5: SSPCON2: MSSP CONTROL REGISTER 2 (I$^2$C MODE) (ADDRESS 91h)**

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| GCEN | ACKSTAT | ACKDT | ACKEN | RCEN | PEN | RSEN | SEN |
| bit 7 | | | | | | | bit 0 |

bit 7 **GCEN:** General Call Enable bit (Slave mode only)
1 = Enable interrupt when a general call address (0000h) is received in the SSPSR
0 = General call address disabled

bit 6 **ACKSTAT:** Acknowledge Status bit (Master Transmit mode only)
1 = Acknowledge was not received from slave
0 = Acknowledge was received from slave

bit 5 **ACKDT:** Acknowledge Data bit (Master Receive mode only)
1 = Not Acknowledge
0 = Acknowledge

**Note:** Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.

bit 4 **ACKEN:** Acknowledge Sequence Enable bit (Master Receive mode only)
1 = Initiate Acknowledge sequence on SDA and SCL pins and transmit ACKDT data bit. Automatically cleared by hardware.
0 = Acknowledge sequence Idle

bit 3 **RCEN:** Receive Enable bit (Master mode only)
1 = Enables Receive mode for I$^2$C
0 = Receive Idle

bit 2 **PEN:** Stop Condition Enable bit (Master mode only)
1 = Initiate Stop condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Stop condition Idle

bit 1 **RSEN:** Repeated Start Condition Enabled bit (Master mode only)
1 = Initiate Repeated Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Repeated Start condition Idle

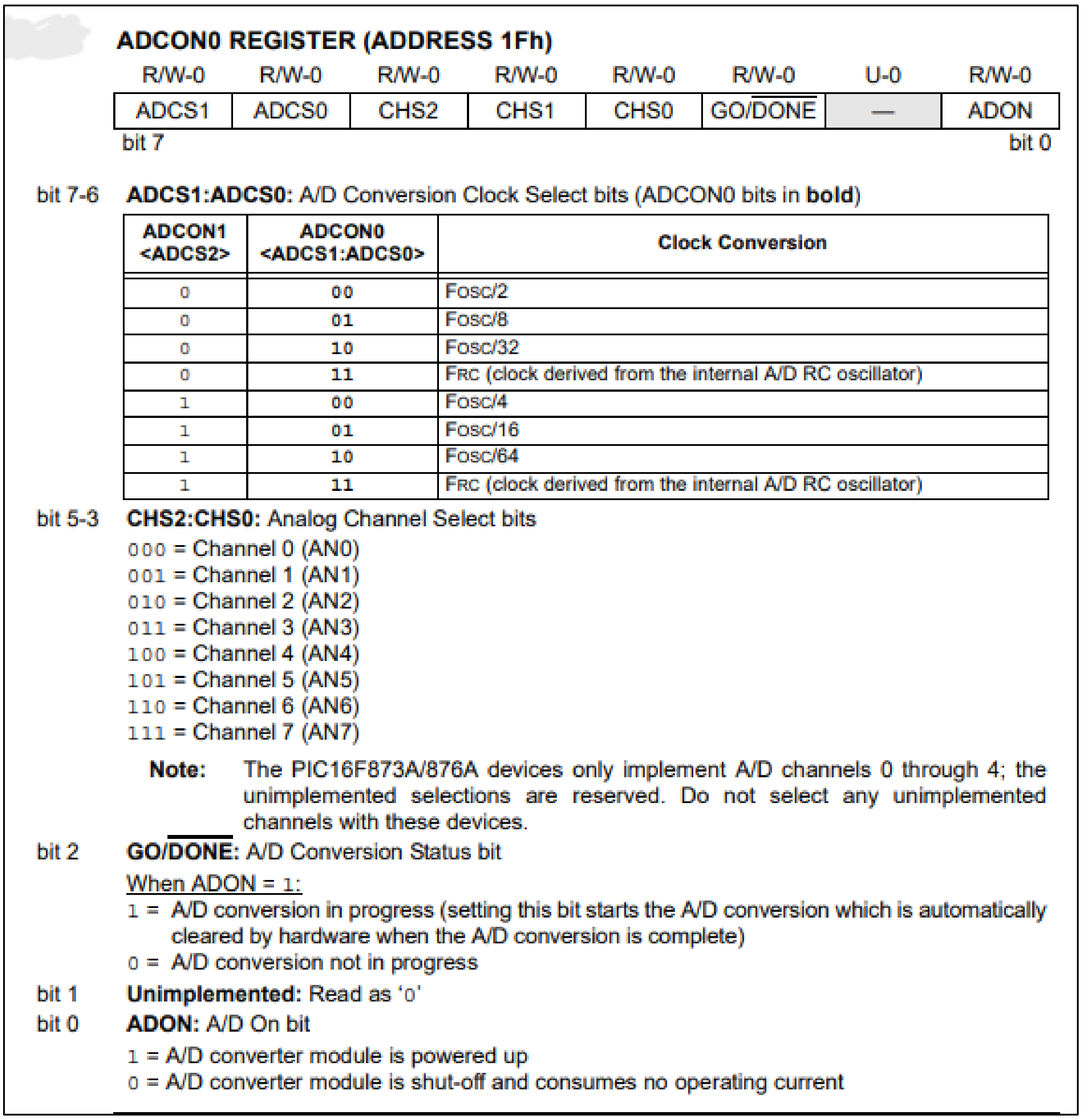
bit 0 **SEN:** Start Condition Enabled/Stretch Enabled bit
In Master mode:
1 = Initiate Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Start condition Idle
In Slave mode:
1 = Clock stretching is enabled for both slave transmit and slave receive (stretch enabled)
0 = Clock stretching is enabled for slave transmit only (PIC16F87X compatibility)

**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 |

**Note:** For bits ACKEN, RCEN, PEN, RSEN, SEN: If the I$^2$C module is not in the Idle mode, this bit may not be set (no spooling) and the SSPBUF may not be written (or writes to the SSPBUF are disabled).

## 3.3.3 ADCON0 Register



**ADCON0 REGISTER (ADDRESS 1Fh)**

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | U-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| ADCS1 | ADCS0 | CHS2 | CHS1 | CHS0 | GO/$\overline{\text{DONE}}$ | — | ADON |
| bit 7 | | | | | | | bit 0 |

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

| **ADCON1 <ADCS2>** | **ADCON0 <ADCS1:ADCS0>** | **Clock Conversion** |
|---|---|---|
| 0 | **00** | Fosc/2 |
| 0 | **01** | Fosc/8 |
| 0 | **10** | Fosc/32 |
| 0 | **11** | FRC (clock derived from the internal A/D RC oscillator) |
| 1 | **00** | Fosc/4 |
| 1 | **01** | Fosc/16 |
| 1 | **10** | Fosc/64 |
| 1 | **11** | FRC (clock derived from the internal A/D RC oscillator) |

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits

000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)
110 = Channel 6 (AN6)
111 = Channel 7 (AN7)

**Note:** The PIC16F873A/876A devices only implement A/D channels 0 through 4; the unimplemented selections are reserved. Do not select any unimplemented channels with these devices.

bit 2 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit

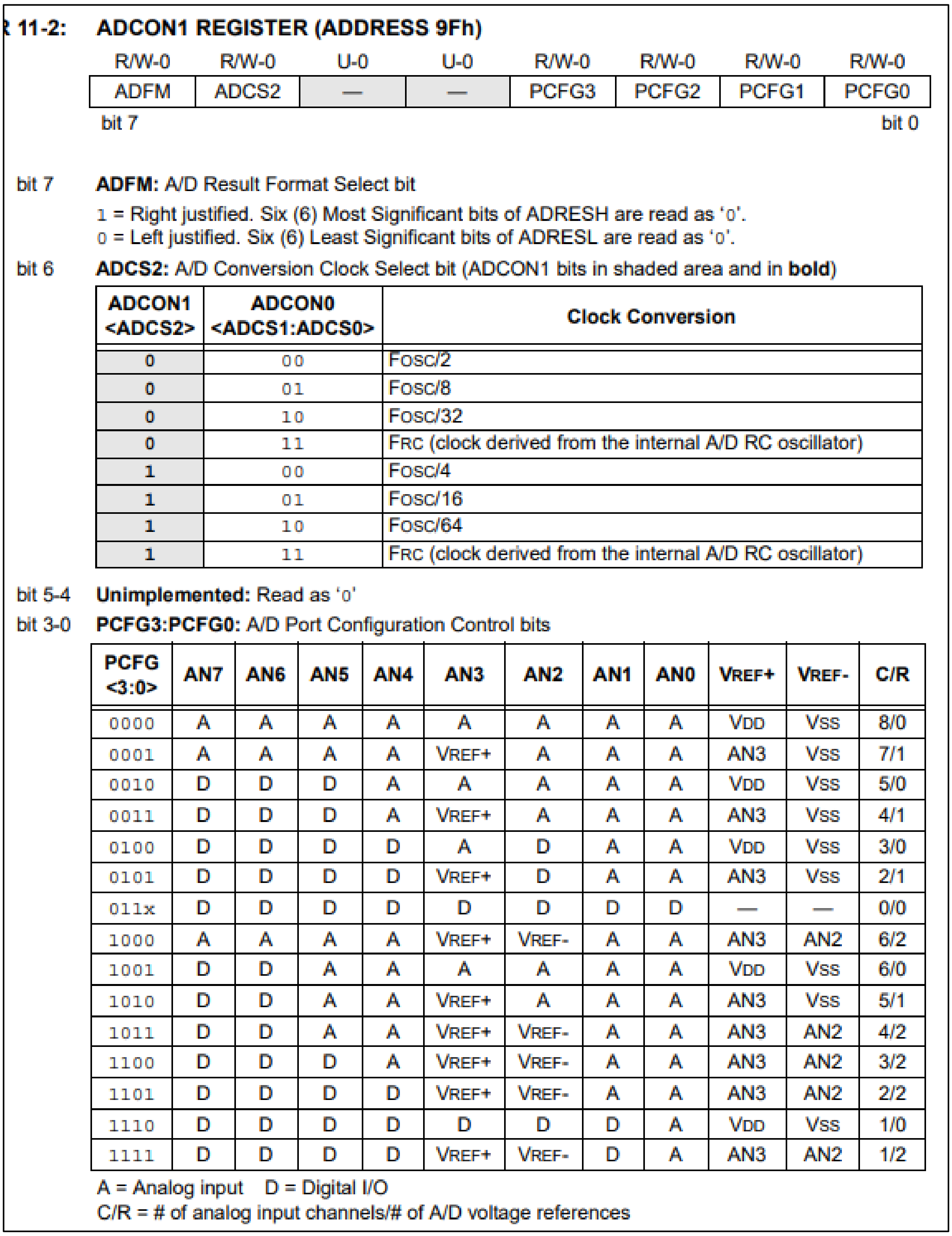
When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit

1 = A/D converter module is powered up
0 = A/D converter module is shut-off and consumes no operating current

## 3.3.4 ADCON1 Register



11-2: **ADCON1 REGISTER (ADDRESS 9Fh)**

| R/W-0 | R/W-0 | U-0 | U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| ADFM | ADCS2 | — | — | PCFG3 | PCFG2 | PCFG1 | PCFG0 |
| bit 7 | | | | | | | bit 0 |

bit 7 **ADFM:** A/D Result Format Select bit

1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

| ADCON1 <ADCS2> | ADCON0 <ADCS1:ADCS0> | Clock Conversion |
|---|---|---|
| **0** | 00 | FOSC/2 |
| **0** | 01 | FOSC/8 |
| **0** | 10 | FOSC/32 |
| **0** | 11 | FRC (clock derived from the internal A/D RC oscillator) |
| **1** | 00 | FOSC/4 |
| **1** | 01 | FOSC/16 |
| **1** | 10 | FOSC/64 |
| **1** | 11 | FRC (clock derived from the internal A/D RC oscillator) |

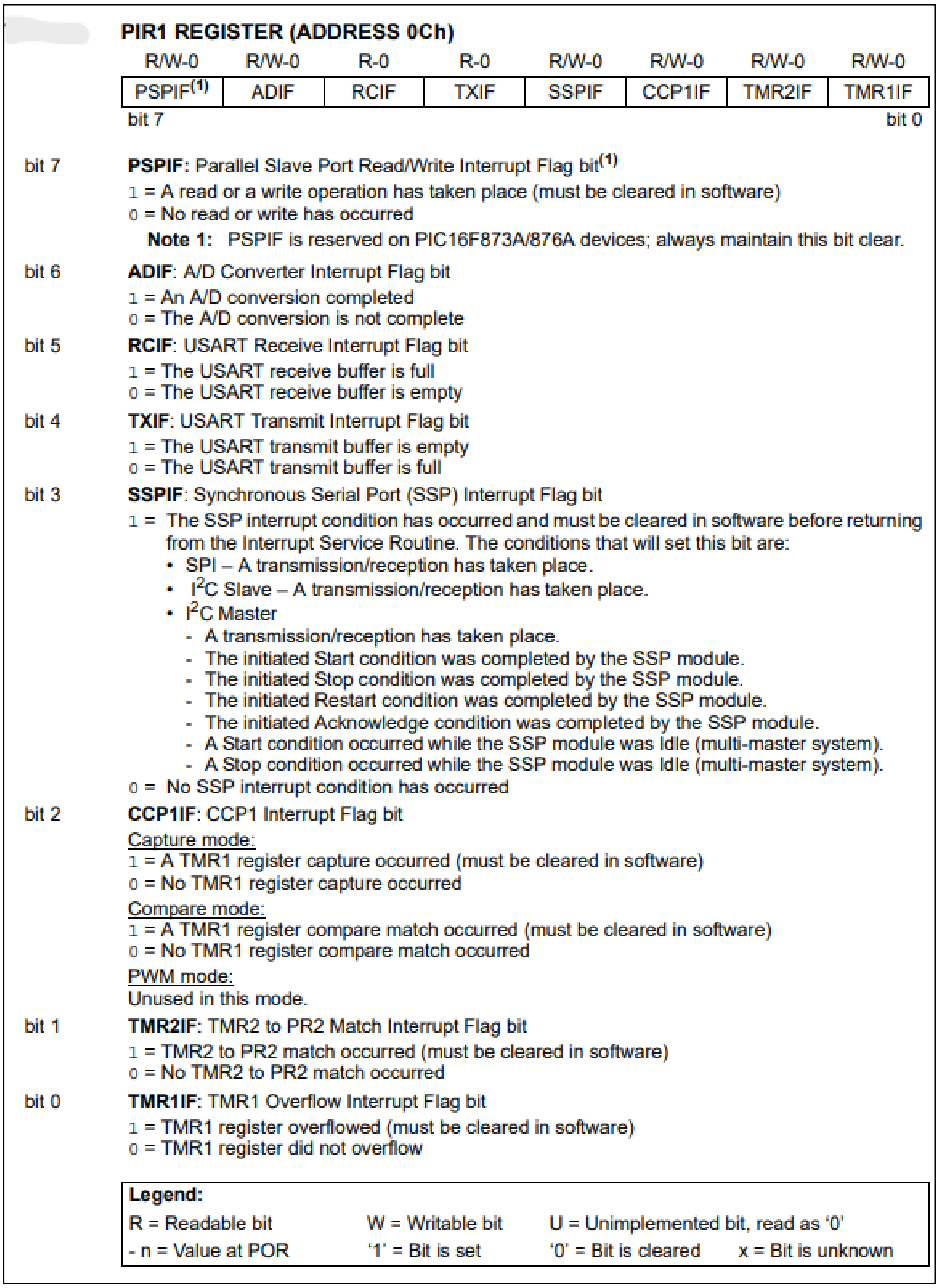
bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

| PCFG <3:0> | AN7 | AN6 | AN5 | AN4 | AN3 | AN2 | AN1 | AN0 | VREF+ | VREF- | C/R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | A | A | A | A | A | A | A | A | VDD | VSS | 8/0 |
| 0001 | A | A | A | A | VREF+ | A | A | A | AN3 | VSS | 7/1 |
| 0010 | D | D | D | A | A | A | A | A | VDD | VSS | 5/0 |
| 0011 | D | D | D | A | VREF+ | A | A | A | AN3 | VSS | 4/1 |
| 0100 | D | D | D | D | A | D | A | A | VDD | VSS | 3/0 |
| 0101 | D | D | D | D | VREF+ | D | A | A | AN3 | VSS | 2/1 |
| 011x | D | D | D | D | D | D | D | D | — | — | 0/0 |
| 1000 | A | A | A | A | VREF+ | VREF- | A | A | AN3 | AN2 | 6/2 |
| 1001 | D | D | A | A | A | A | A | A | VDD | VSS | 6/0 |
| 1010 | D | D | A | A | VREF+ | A | A | A | AN3 | VSS | 5/1 |
| 1011 | D | D | A | A | VREF+ | VREF- | A | A | AN3 | AN2 | 4/2 |
| 1100 | D | D | D | A | VREF+ | VREF- | A | A | AN3 | AN2 | 3/2 |
| 1101 | D | D | D | D | VREF+ | VREF- | A | A | AN3 | AN2 | 2/2 |
| 1110 | D | D | D | D | D | D | D | A | VDD | VSS | 1/0 |
| 1111 | D | D | D | D | VREF+ | VREF- | D | A | AN3 | AN2 | 1/2 |

A = Analog input D = Digital I/O
C/R = # of analog input channels/# of A/D voltage references

## 3.3.5 PIR1 Register



**PIR1 REGISTER (ADDRESS 0Ch)**

| R/W-0 | R/W-0 | R-0 | R-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| PSPIF[(1)] | ADIF | RCIF | TXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF |
| bit 7 | | | | | | | bit 0 |

bit 7 **PSPIF:** Parallel Slave 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

**Note 1:** PSPIF is reserved on PIC16F873A/876A devices; always maintain this bit clear.

bit 6 **ADIF**: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed
0 = The A/D conversion is not complete

bit 5 **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer is full
0 = The USART receive buffer is empty

bit 4 **TXIF**: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer is empty
0 = The USART transmit buffer is full

bit 3 **SSPIF**: Synchronous Serial Port (SSP) Interrupt Flag bit
1 = The SSP interrupt condition has occurred and must be cleared in software before returning from the Interrupt Service Routine. The conditions that will set this bit are:
- SPI – A transmission/reception has taken place.
- I$^2$C Slave – A transmission/reception has taken place.
- I$^2$C Master
  - A transmission/reception has taken place.
  - The initiated Start condition was completed by the SSP module.
  - The initiated Stop condition was completed by the SSP module.
  - The initiated Restart condition was completed by the SSP module.
  - The initiated Acknowledge condition was completed by the SSP module.
  - A Start condition occurred while the SSP module was Idle (multi-master system).
  - A Stop condition occurred while the SSP module was Idle (multi-master system).

0 = No SSP interrupt condition has occurred

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.

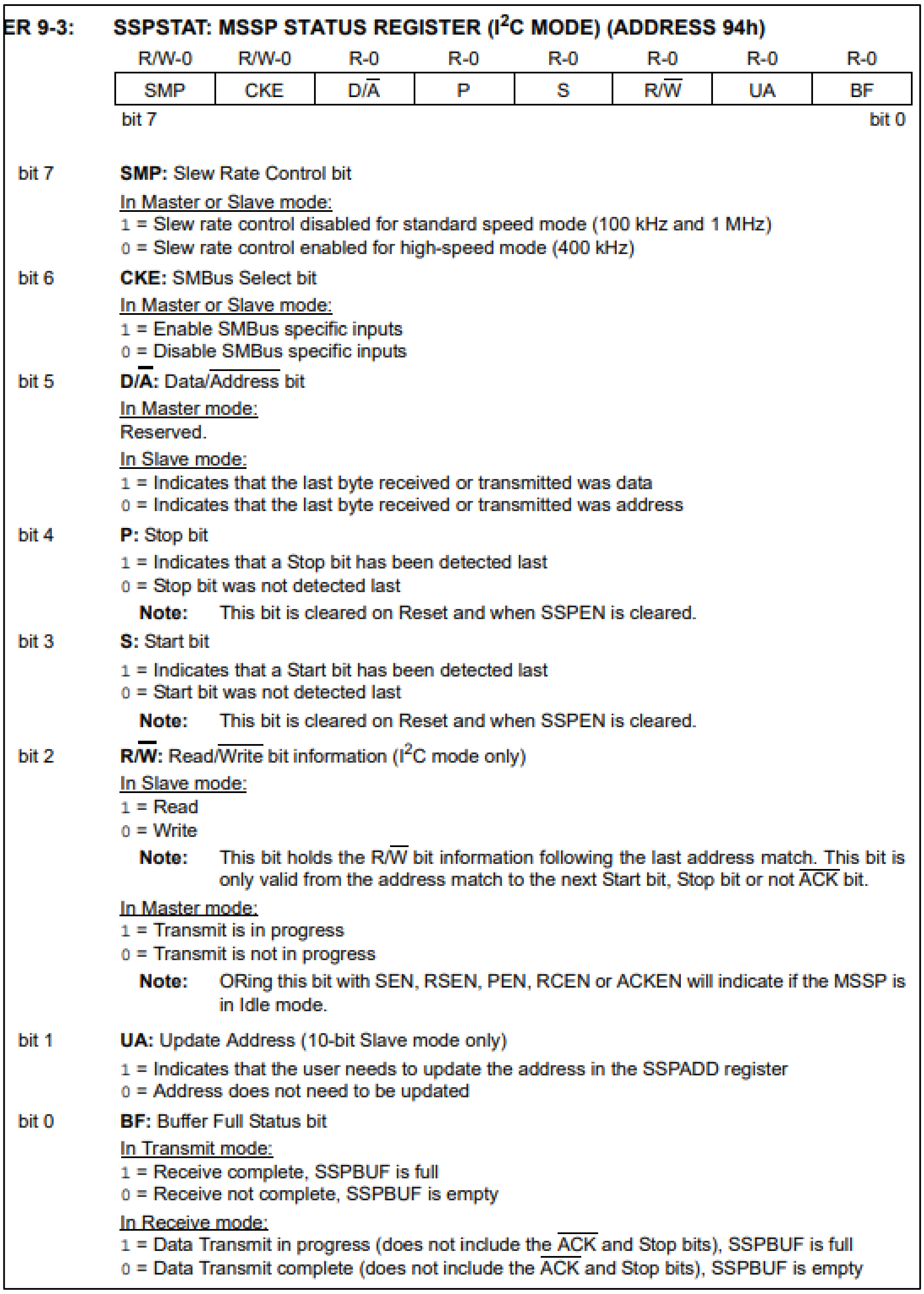
bit 1 **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

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

**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 |

## 3.3.6MSSP Status Register (SSPSTAT)



**ER 9-3: SSPSTAT: MSSP STATUS REGISTER ($I^2C$ MODE) (ADDRESS 94h)**

| R/W-0 | R/W-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
|---|---|---|---|---|---|---|---|
| SMP | CKE | D/$\overline{\text{A}}$ | P | S | R/$\overline{\text{W}}$ | UA | BF |
| bit 7 | | | | | | | bit 0 |

bit 7 **SMP:** Slew Rate Control bit
In Master or Slave mode:
1 = Slew rate control disabled for standard speed mode (100 kHz and 1 MHz)
0 = Slew rate control enabled for high-speed mode (400 kHz)

bit 6 **CKE:** SMBus Select bit
In Master or Slave mode:
1 = Enable SMBus specific inputs
0 = Disable SMBus specific inputs

bit 5 **D/$\overline{\text{A}}$:** Data/$\overline{\text{Address}}$ bit
In Master mode:
Reserved.
In Slave mode:
1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P:** Stop bit
1 = Indicates that a Stop bit has been detected last
0 = Stop bit was not detected last
**Note:** This bit is cleared on Reset and when SSPEN is cleared.

bit 3 **S:** Start bit
1 = Indicates that a Start bit has been detected last
0 = Start bit was not detected last
**Note:** This bit is cleared on Reset and when SSPEN is cleared.

bit 2 **R/$\overline{\text{W}}$:** Read/$\overline{\text{Write}}$ bit information ($I^2C$ mode only)
In Slave mode:
1 = Read
0 = Write
**Note:** This bit holds the R/$\overline{\text{W}}$ bit information following the last address match. This bit is only valid from the address match to the next Start bit, Stop bit or not $\overline{\text{ACK}}$ bit.
In Master mode:
1 = Transmit is in progress
0 = Transmit is not in progress
**Note:** ORing this bit with SEN, RSEN, PEN, RCEN or ACKEN will indicate if the MSSP is in Idle mode.

bit 1 **UA:** Update Address (10-bit Slave mode only)
1 = Indicates that the user needs to update the address in the SSPADD register
0 = Address does not need to be updated

bit 0 **BF:** Buffer Full Status bit
In Transmit mode:
1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty
In Receive mode:
1 = Data Transmit in progress (does not include the $\overline{\text{ACK}}$ and Stop bits), SSPBUF is full
0 = Data Transmit complete (does not include the $\overline{\text{ACK}}$ and Stop bits), SSPBUF is empty

# CHAPTER 4
# EXPERIMENT

## 4.1 CODE:

```
// PIC16F877A Configuration Bit Settings

// CONFIG

#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)

#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)

#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)

#pragma config BOREN = OFF // Brown-out Reset Enable bit (BOR disabled)

#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial
Programming

 //Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)

#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit
(Data EEPROM

//code protection off)

#pragma config WRT = OFF // Flash Program Memory Write Enable bits
(Write

//protection off; all program memory may be written to by EECON control)

#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code

//protection off)

#include <xc.h>

#define _XTAL_FREQ 20000000

#define RS PORTDbits.RD0

#define RW PORTDbits.RD1

#define EN PORTDbits.RD2

#define DS1307_address 0xD0

#define second 0x00

#define minute 0x01
```

```
#define hour 0x02
#define day 0x03
#define date 0x04
#define month 0x05
#define year 0x06
#define control_reg 0x07
unsigned char __sec,__min,__hr,__day,__date,__month,__yr,__con;
unsigned int ADC_value,z,y,x,q,h,g;

void bcd_to_ascii(unsigned char value);
char decimal_to_bcd(unsigned char value);
void lcd_cmd(unsigned char cmd);
void lcd_data(unsigned char data);
void lcd_initialise();
void lcd_string(unsigned char a);
void lcd_word(const unsigned char *word, unsigned char num);
void DS1307_write(char sec, char min, char hr, char _day, char _date, char
_month, char
_year);
void DS1307_read(char slave_address,char register_address);
void lcd_data(unsigned char data)
{
 PORTB = data;
 RS = 1;
 RW = 0;
 EN = 1;
 __delay_ms(5);
 EN = 0;
```

```
}
void lcd_cmd(unsigned char cmd)
{
 PORTB = cmd;
 RS = 0;
 RW = 0;
 EN = 1;
 __delay_ms(5);
 EN = 0;
}
void lcd_word(const unsigned char *word, unsigned char num)
{
 unsigned char i;
 for(i=0;i<num;i++)
 {
 lcd_data(word[i]);
 }
}
void lcd_initialise()
{
 lcd_cmd(0x38);
 lcd_cmd(0x06);
 lcd_cmd(0x0C);
 lcd_cmd(0x01);
}
char decimal_to_bcd(unsigned char value)
{
```

```
unsigned char msb,lsb,hex;
 msb=value/10;
 lsb=value%10;
 hex = ((msb<<4)+lsb);
 return hex;
}
void bcd_to_ascii(unsigned char value)
{
 unsigned char bcd;
 bcd= value;
 bcd=bcd&0xf0;
 bcd=bcd>>4;
 bcd=bcd|0x30;
 lcd_data(bcd);
 bcd=value;
 bcd=bcd&0x0f;
 bcd=bcd|0x30;
 lcd_data(bcd);
}
void DS1307_write(char _second, char _minute, char _hour, char _day, char
_date, char
_month, char _year)
{
 /* start bit*/
 SSPCON2bits.SEN = 1; // initiate start condition
 while(SEN); // wait for start condition to complete
 PIR1bits.SSPIF = 0; // clear SSPIF flag
```

```
/* slave address bits*/
SSPBUF = DS1307_address; // send the slave address high and r/w = 0 for
write
while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
PIR1bits.SSPIF = 0; // clear SSPIF flag
if(SSPCON2bits.ACKSTAT)
{
SSPCON2bits.PEN = 1; // initiate stop condition
while(PEN); // wait for stop condition to complete
return; // exit write (no acknowledgment)
}

SSPBUF = second; // send the slave address high and r/w = 0 for write
while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
PIR1bits.SSPIF = 0; // clear SSPIF flag
if(SSPCON2bits.ACKSTAT)
{
SSPCON2bits.PEN = 1; // initiate stop condition
while(PEN); // wait for stop condition to complete
return; // exit write (no acknowledgment)
}

SSPBUF = decimal_to_bcd(_second); // send the data
while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
PIR1bits.SSPIF = 0; // clear SSPIF flag

SSPBUF = decimal_to_bcd( _minute); // send the data
while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
```

```
PIR1bits.SSPIF = 0; // clear SSPIF flag

SSPBUF = decimal_to_bcd(_hour); // send the data
while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
PIR1bits.SSPIF = 0; // clear SSPIF flag

SSPBUF = decimal_to_bcd(_day); // send the data
while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
PIR1bits.SSPIF = 0; // clear SSPIF flag
SSPBUF = decimal_to_bcd(_date); // send the data
while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
PIR1bits.SSPIF = 0; // clear SSPIF flag

SSPBUF = decimal_to_bcd(_month); // send the data
while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
PIR1bits.SSPIF = 0; // clear SSPIF flag

SSPBUF = decimal_to_bcd(_year); // send the data
while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
PIR1bits.SSPIF = 0; // clear SSPIF flag

SSPBUF = 0x00; // send the data
while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
PIR1bits.SSPIF = 0; // clear SSPIF flag

/* stop bits */
SSPCON2bits.PEN = 1; // initiate stop condition
```

```
 while(PEN); // wait for stop condition to complete
}
void DS1307_read(char slave_address,char register_address)
{
 /* start bit*/
 SSPCON2bits.SEN = 1; // initiate start condition
 while(SEN); // wait for start condition to complete
 PIR1bits.SSPIF = 0; // clear SSPIF flag

 /* slave address bits*/
 SSPBUF = slave_address; // send the slave address high and r/w = 0 for write
 while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
 PIR1bits.SSPIF = 0; // clear SSPIF flag
 if(SSPCON2bits.ACKSTAT)
 {
 SSPCON2bits.PEN = 1; // initiate stop condition
 while(PEN); // wait for stop condition to complete
 return; // exit write (no acknowledgment)
 }
 /* slave address bits*/
 SSPBUF = register_address; // send the slave address high and r/w = 0 for write
 while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
 PIR1bits.SSPIF = 0; // clear SSPIF flag
 if(SSPCON2bits.ACKSTAT)
 {
 SSPCON2bits.PEN = 1; // initiate stop condition
 while(PEN); // wait for stop condition to complete
return; // exit write (no acknowledgment)
}
/* Repeated start bit*/
SSPCON2bits.RSEN = 1; // initiate start condition
while(RSEN); // wait for start condition to complete
PIR1bits.SSPIF = 0; // clear SSPIF flag

/* slave address bits*/
SSPBUF = (slave_address + 1); // send the slave address high and r/w = 1 bit
while(!SSPIF); // wait for acknowledge SSPIF is set for every 9th clock cycle
PIR1bits.SSPIF = 0; // clear SSPIF flag
if(SSPCON2bits.ACKSTAT)
{
SSPCON2bits.PEN = 1; // initiate stop condition
while(PEN); // wait for stop condition to complete
return ; // exit write (no acknowledgment)
}

/* Receive enable bit*/
SSPCON2bits.RCEN = 1; // initiate receive enable
while(!SSPSTATbits.BF); // wait for BUFFER TO BE FULL
__sec = SSPBUF; // clear SSPIF flag

SSPCON2bits.ACKDT = 0; // Prepare to send NACK
SSPCON2bits.ACKEN = 1; // Initiate to send NACK
while(ACKEN);
```

```
/* Receive enable bit*/
SSPCON2bits.RCEN = 1; // initiate receive enable
while(!SSPSTATbits.BF); // wait for BUFFER TO BE FULL
__min = SSPBUF; // clear SSPIF flag


SSPCON2bits.ACKDT = 0; // Prepare to send NACK
SSPCON2bits.ACKEN = 1; // Initiate to send NACK
while(ACKEN);


/* Receive enable bit*/
SSPCON2bits.RCEN = 1; // initiate receive enable
while(!SSPSTATbits.BF); // wait for BUFFER TO BE FULL
__hr = SSPBUF; // clear SSPIF flag


SSPCON2bits.ACKDT = 0; // Prepare to send NACK
SSPCON2bits.ACKEN = 1; // Initiate to send NACK
while(ACKEN);

/* Receive enable bit*/
SSPCON2bits.RCEN = 1; // initiate receive enable
while(!SSPSTATbits.BF); // wait for BUFFER TO BE FULL
__day = SSPBUF; // clear SSPIF flag

SSPCON2bits.ACKDT = 0; // Prepare to send NACK
```

```
SSPCON2bits.ACKEN = 1; // Initiate to send NACK
while(ACKEN);

/* Receive enable bit*/
SSPCON2bits.RCEN = 1; // initiate receive enable
while(!SSPSTATbits.BF); // wait for BUFFER TO BE FULL
__date = SSPBUF; // clear SSPIF flag



SSPCON2bits.ACKDT = 0; // Prepare to send NACK
SSPCON2bits.ACKEN = 1; // Initiate to send NACK
while(ACKEN);



/* Receive enable bit*/
SSPCON2bits.RCEN = 1; // initiate receive enable
while(!SSPSTATbits.BF); // wait for BUFFER TO BE FULL
__month = SSPBUF; // clear SSPIF flag



SSPCON2bits.ACKDT = 0; // Prepare to send NACK
SSPCON2bits.ACKEN = 1; // Initiate to send NACK
while(ACKEN);

/* Receive enable bit*/
SSPCON2bits.RCEN = 1; // initiate receive enable
while(!SSPSTATbits.BF); // wait for BUFFER TO BE FULL
```

```
__yr = SSPBUF; // clear SSPIF flag
SSPCON2bits.ACKDT = 0; // Prepare to send NACK
SSPCON2bits.ACKEN = 1; // Initiate to send NACK
while(ACKEN);

/* Receive enable bit*/
SSPCON2bits.RCEN = 1; // initiate receive enable
while(!SSPSTATbits.BF); // wait for BUFFER TO BE FULL
__con = SSPBUF; // clear SSPIF flag

SSPCON2bits.ACKDT = 1; // Prepare to send NACK
SSPCON2bits.ACKEN = 1; // Initiate to send NACK
while(ACKEN); // WAIT FOR NACK TO COMPLETE

/* stop bits */
SSPCON2bits.PEN = 1; // initiate stop condition
while(PEN); // wait for stop condition to complete

lcd_cmd(0x85);
bcd_to_ascii(__hr);
lcd_data(':');
bcd_to_ascii(__min);
lcd_data(':');
bcd_to_ascii(__sec);
```

```
}
void adcinit()
{
     ADCON0bits.ADCS0 = 1;ADCON0bits.ADCS1= 0 ;
    ADCON1bits.ADCS2 = 0;
    ADCON0bits.ADON = 1;
    ADCON1bits.PCFG0 = 0; ADCON1bits.PCFG1 = 1; ADCON1bits.PCFG2
= 1; ADCON1bits.PCFG3 = 1;
    ADCON1bits.ADFM = 1;
}
void adcread()
{lcd_cmd(0xc0);
    lcd_word("pot value",10);

     ADCON0bits.CHS0 = 0;ADCON0bits.CHS1 = 0;ADCON0bits.CHS2 = 0;
    ADCON0bits.GO_DONE = 1;
    while(PIR1bits.ADIF == 0);
    ADC_value = ADRESH<<8;
    ADC_value = ADC_value + ADRESL;
      z = ADC_value / 10;
  y = ADC_value % 10;
  x = z / 10;
  q = z % 10;
  h = x / 10;
  g = x % 10;
  lcd_cmd(0xcB);
  lcd_data(h+0x30);
  lcd_data(g+0x30);
  lcd_data(q+0x30);
  lcd_data(y+0x30);
}
void main(void)
{
 TRISC = 0xFF;
 SSPADD = 49;
 SSPCON = 0x28;
 TRISB = 0x00;
 TRISD = 0x00;
 lcd_initialise();
 adcinit();
 lcd_cmd(0x80);
 lcd_word("TIME:", 5);



 DS1307_write(0,30,12,1,7,2,22);
 while(1)
 {
 DS1307_read(DS1307_address,0);
 adcread();

 }
 return;
}
```

## 4.2 SCHEMATIC DIAGRAM:

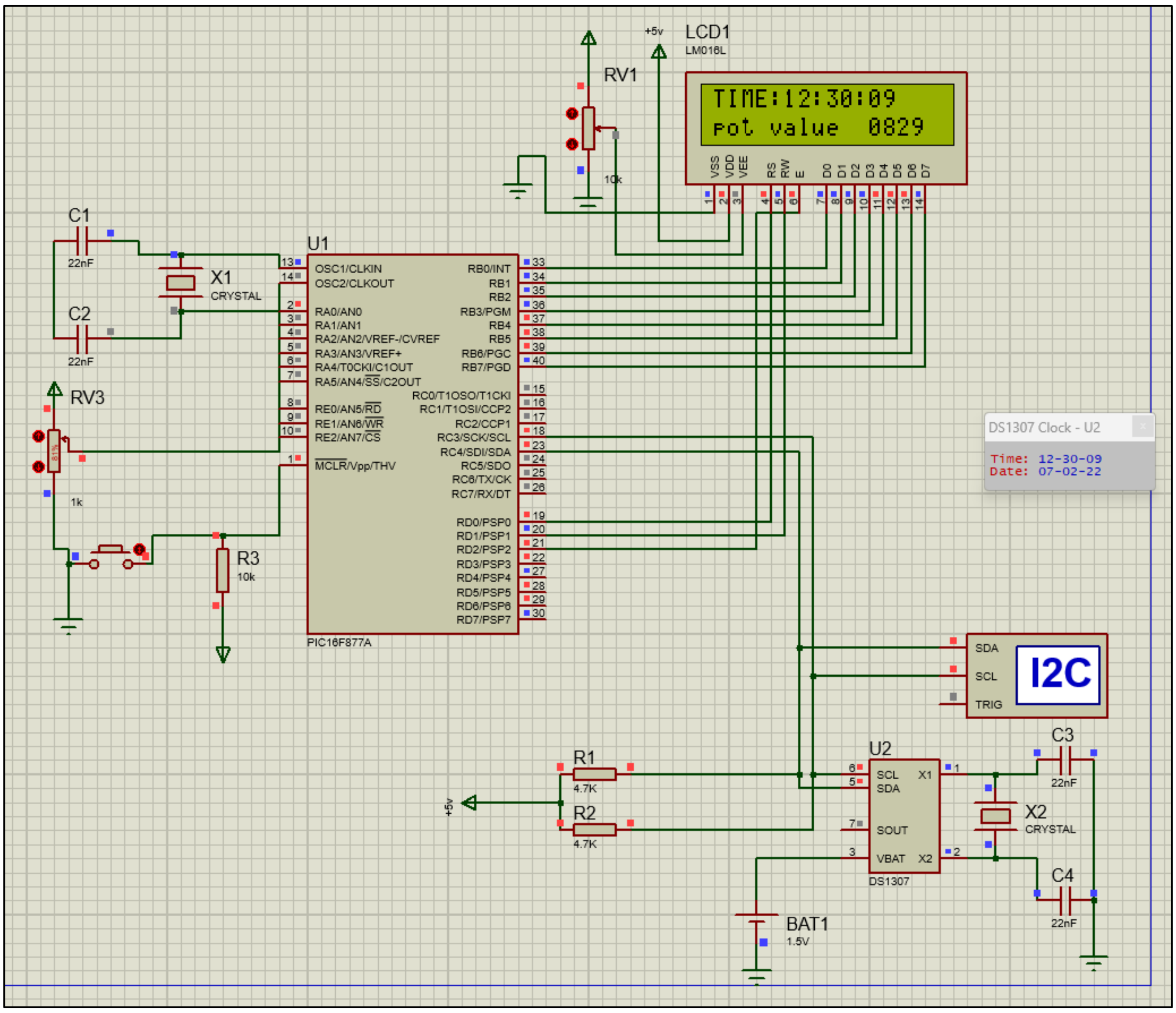


Figure 4.2(a) *RTC and ADC interface with LCD Display*

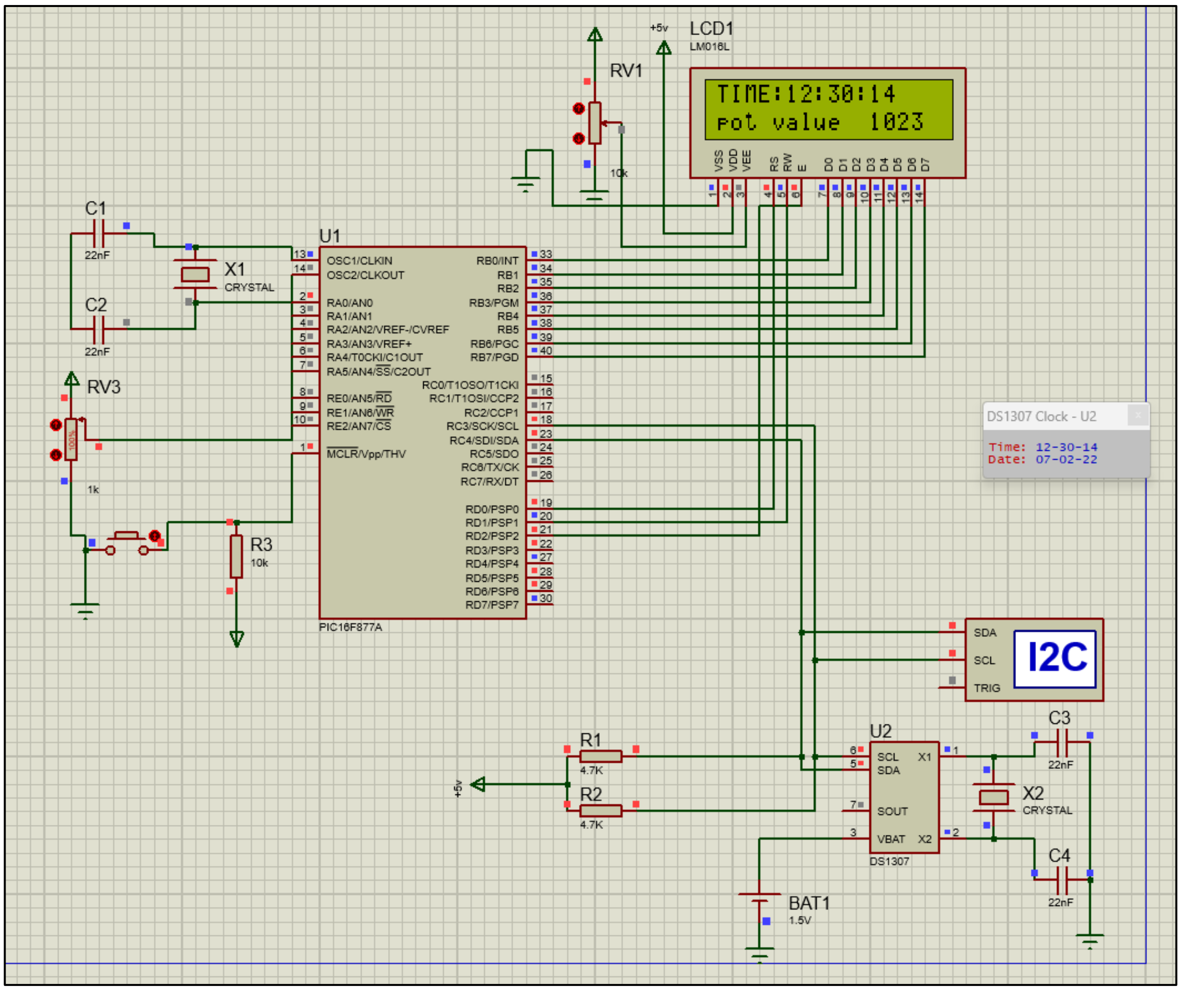
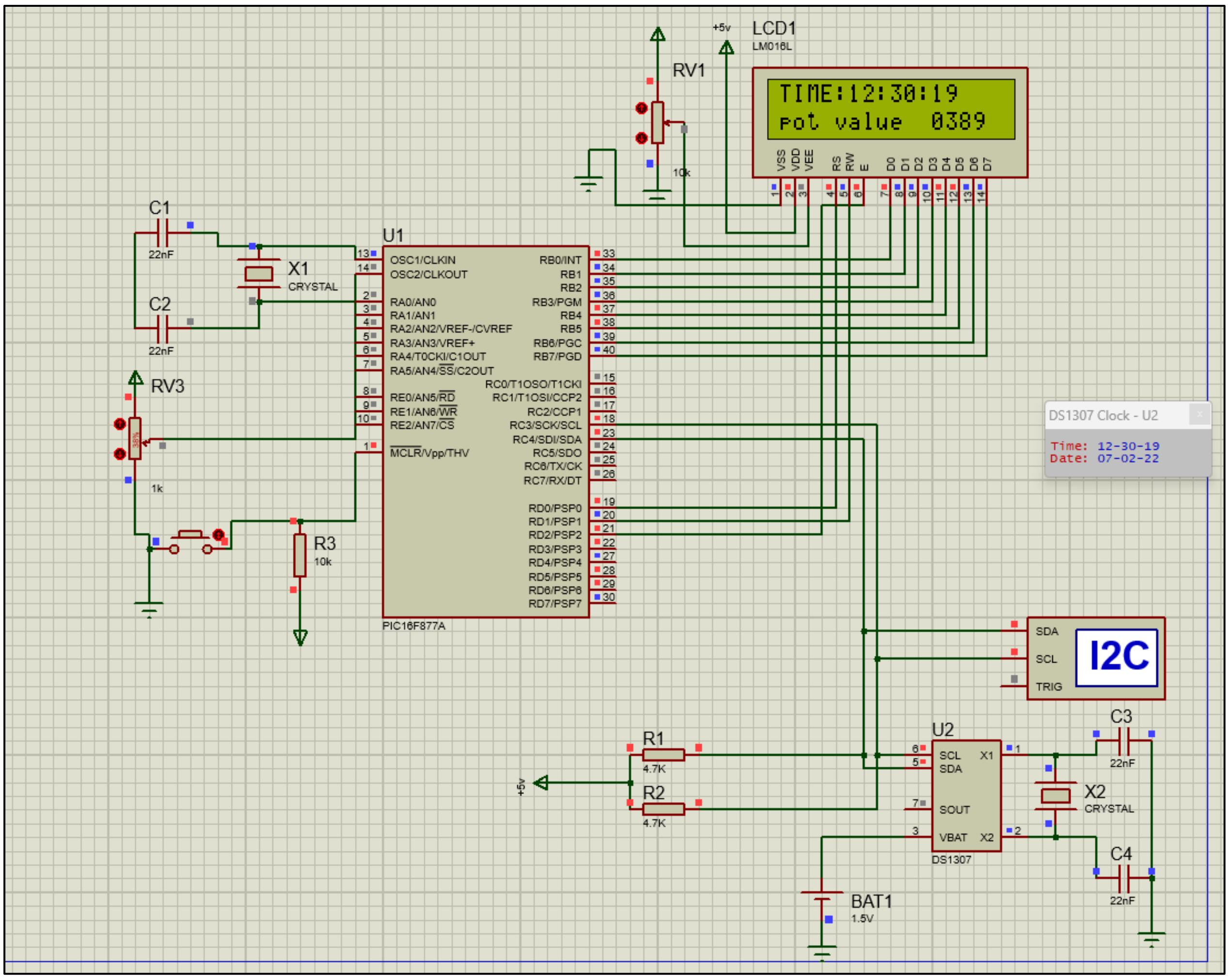


**Figure 4.2(b)** ***change in analog value***



```
TIME:12:30:19
pot value  0389
```

```
Time: 12-30-19
Date: 07-02-22
```

Figure 4.2(c) *change in analog value.*

# CHAPTER 5

## REFERENCES

- https://www.alldatasheet.com/view.jsp?Searchword=Pic16f877a%20datasheet&gclid=Cj0KCQiAr5iQBhCsARIsAPcwROPQtd4Unkm4BIq87WPo8VtdtRnLovkqQsImyi66XBHQDaThFSQtdTcaAqt5EALw_wcB
- https://www.sparkfun.com/datasheets/Components/DS1307.pdf
- https://microcontrollerslab.com/i2c-communication-pic-microcontroller/
- https://www.geeksforgeeks.org/i2c-communication-protocol/
- https://smtrainingacademy.com/
- https://www.youtube.com/watch?v=210ForgwE3Y&list=PL_zvrXFdKgZpTrM99mypGVW5JBZ6tQZiR