Re-create yourself

QUEST
INNOVATIVE SOLUTIONS

PIC MICROCONTROLLER

- **Classification:**

- PIC devices are grouped by the size of their instruction word length Classification
  - Base line: 12 bit instruction word length.
  - Mid-range:14 bit instruction word length.
  - High-end: 16 bit instruction word length.

PIC
Peripheral Interface Controller
Microcontrollers from
Microchip Technology Inc.

- Microchip Technology Inc. is a leading provider of microcontroller and analog semiconductors in the world. Headquartered in Chandler, Arizona.

Device Structure

- Each part of PIC can be placed into one of the three groups:
  - Core
  - Peripherals
  - Special Features

- **The Core:**
  - The core includes the basic features that are required to make the device operate. These include:
    - CPU (Central Processing Unit) operation
    - ALU (Arithmetic Logical Unit) operation
    - The Core
    - Device memory map organization
    - Device Oscillator
    - Reset logic
    - Interrupt operation
    - Instruction set

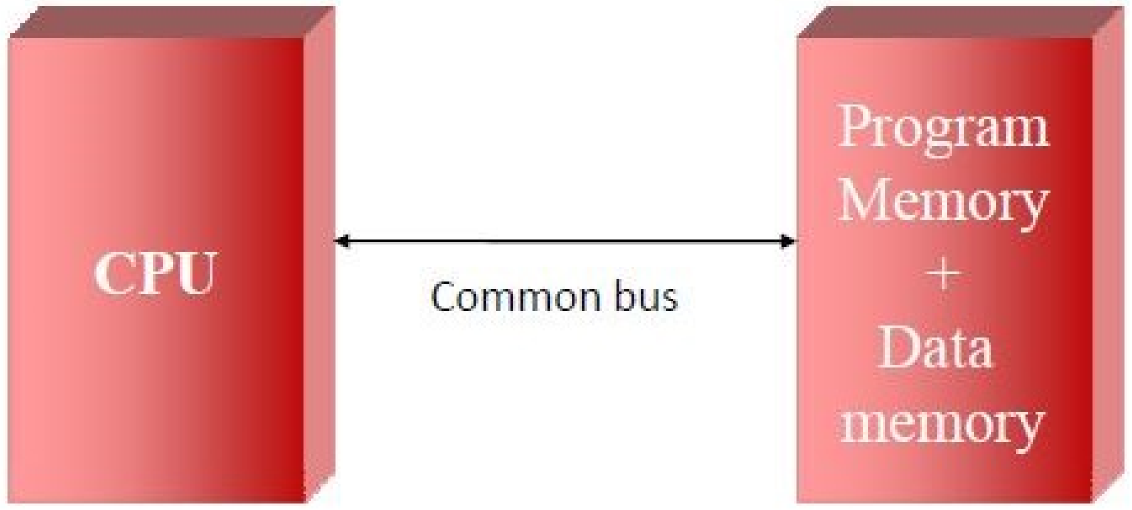
Peripherals

- Peripherals are the features that add a differentiation from a microprocessor. These include
  - General purpose I/O
  - Timer0
  - Timer1
  - Timer2
  - Capture, Compare, and PWM (CCP)
  - Synchronous Serial Port (SSP)
  - USART
  - Analog to Digital (A/D) Converter

- **Special Features**
  - Special features are the unique features that help to
    - Decrease system cost
    - Increase system reliability
    - Increase design flexibility
    - The Mid-Range PIC offers several features such as
    - On-chip Power-on Reset (POR)
    - Brown-out Reset (BOR) logic
    - Watchdog Timer
    - Low power mode (Sleep)

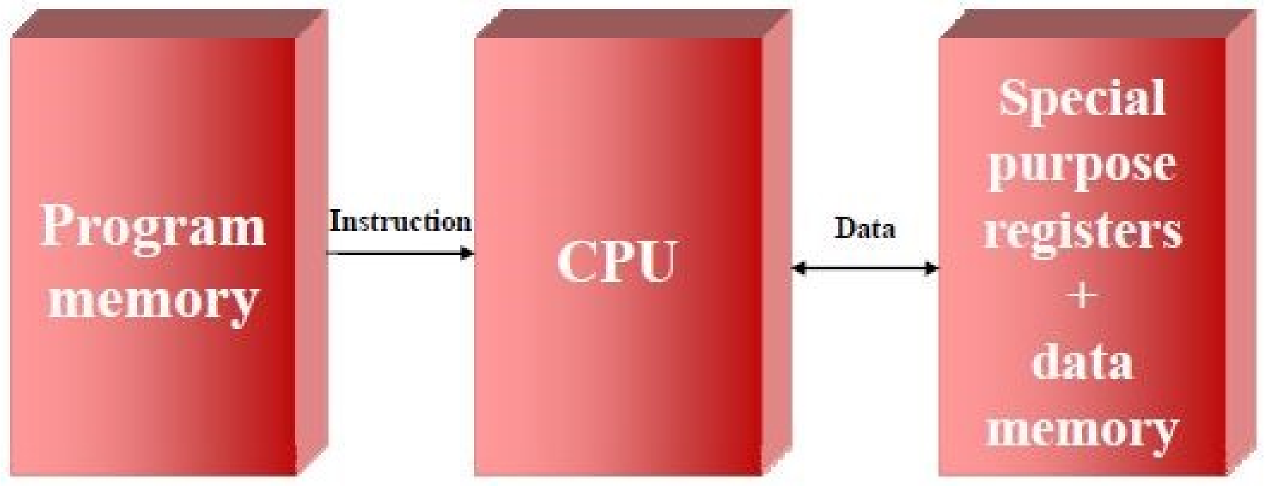
- **CPU Architectures**
  - Von-Neumann’s architecture
  - Harvard architecture

Von-Neumann's architecture



- Common data and address bus.
- More overhead.

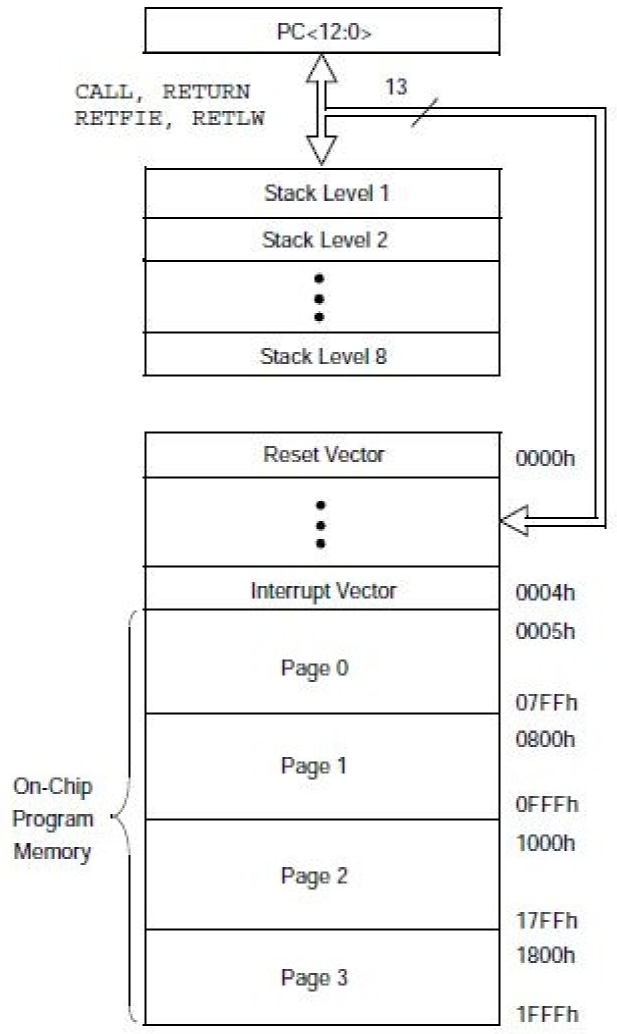
Harvard architecture



- Data bus and address bus are separate.
- So greater speed of work.

- **Memory Organization**
  - Three memory blocks
    - Program memory – 14 bit
    - Data memory – 8 bit
    - Internal EEPROM data memory -8 bit

Program Memory Map and STACK



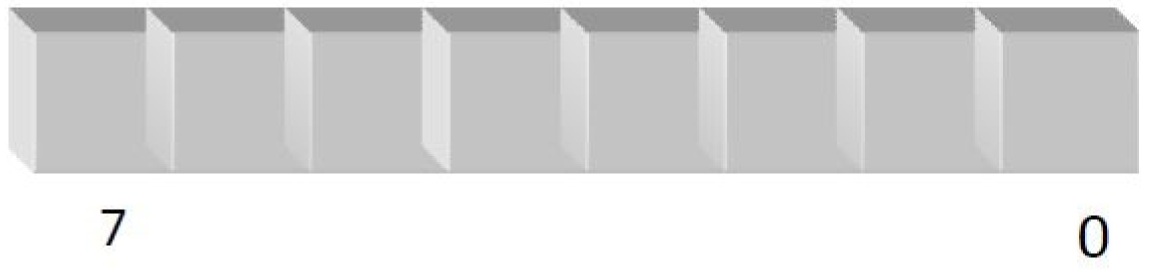
QUEST INNOVATIVE SOLUTIONS

General Purpose & Special Function Registers

- **General Purpose and SFRs**
  - Three memory blocks
    - Program memory – 14 bit
    - Data memory – 8 bit
    - Internal EEPROM data memory -8 bit

CPU Registers

- Working Register



- Use as source operand
- Destination for the result

PIC16F876A

Naming of PIC

- Lets take PIC16F876A

  - PIC – Peripheral Interface Controller
  - 16 – Midrange series
  - F – Flash memory
  - 876 A– 28 pin,10bit ADC, with internal EEPROM

Naming of PIC

- 12 – Base line
- 16 – Midrange
- 17/18 – High end

- C – EPROM
- CR – ROM
- F – Flash

- 7X – 28 pin,8bit ADC, without internal EEPROM
- 87XX - 28 pin,10bit ADC, with internal EEPROM

- **Features**
  - 8-bit RISC ALU
  - Harvard architecture
  - 28 pins with 22 I/O pins
  - Operating Frequency - DC – 20 MHz
  - Flash Program Memory (14-bit words) – 8K
  - Data Memory (bytes) – 368
  - EEPROM Data Memory (bytes) – 256
  - Interrupt sources – 14
  - I/O Ports - Ports A, B, C
  - Timers – 3
  - 5 channel - 10-bit Analog-to-Digital Module
  - Serial Communications - MSSP, USART

- Crystal oscillator
- Instruction cycle

CLOCK INPUT

- Assume that your PIC is running at 4 MHz.
- There's an internal divide-by-4 between the oscillator frequency and the instruction clock
- This means that instruction clocks occur at a 1 MHz rate.

- **Question for you?**

- PIC is an 8-bit microcontroller? Why?
- The CPU can work on only 8-bits of data at a time.

An Introduction to MPLAB integrated Development Environment

- What is MPLAB IDE?

- MPLAB IDE is a software program that runs on your PC to provide a development environment for your embedded system design.

FIRST PROJECT

- Select Device
- Create Project
- Select Language Tools
- Put Files in Project
- Create Code
- Build Project
- Test Code with Simulator

Introduction to XC8 compiler

XC8 Compiler

- XC8 compiler is a full featured, highly – optimized ANSI C compiler for the PIC 10/12/16/18 microcontroller families.
- This compiler integrates into Microchips MPLAB IDE, and runs on Windows, Linux and Mac OS.

Introduction to PROTEUS simulator

Proteus Design Suite

- Proteus Design Suite is a proprietary software tool suite used primarily for electronic design automation.
- Developed in England, by Labcenter Electronics Ltd.
- This software is mainly used to create schematics and electronic prints for manufacturing PCBs.

PIC16F876A

I/O PORTS

I/O PORTS

- A port is a group of pins on a microcontroller on which the desired combinations of zeros and ones can be set simultaneously or the present status can be read.

- PIC16F76A has 3 ports
- PORTA - 6 pins
- PORTB - 8 pins
- PORTC - 8 pins

TRISx Register

- If a bit of TRIS Register =1, corresponding PORT pin will be configured as input
- If a bit of TRIS Register =0, corresponding PORT pin will be configured as output

TASK

1. LED Blinking
2. PORT Rotation
3. LED Blinks only when a switch is pressed
4. LED Patterns for each switch press
5. Create a 3bit Encoder

Polling and Interrupt method:

- **Polling:**
  - It is the process where the computer or controlling device waits for an external device to check for its state.
- **Interrupt:**
  - It is the process where the computer or controlling device to temporarily work on a different task, and then return to its previous task.

PERIPHERALS

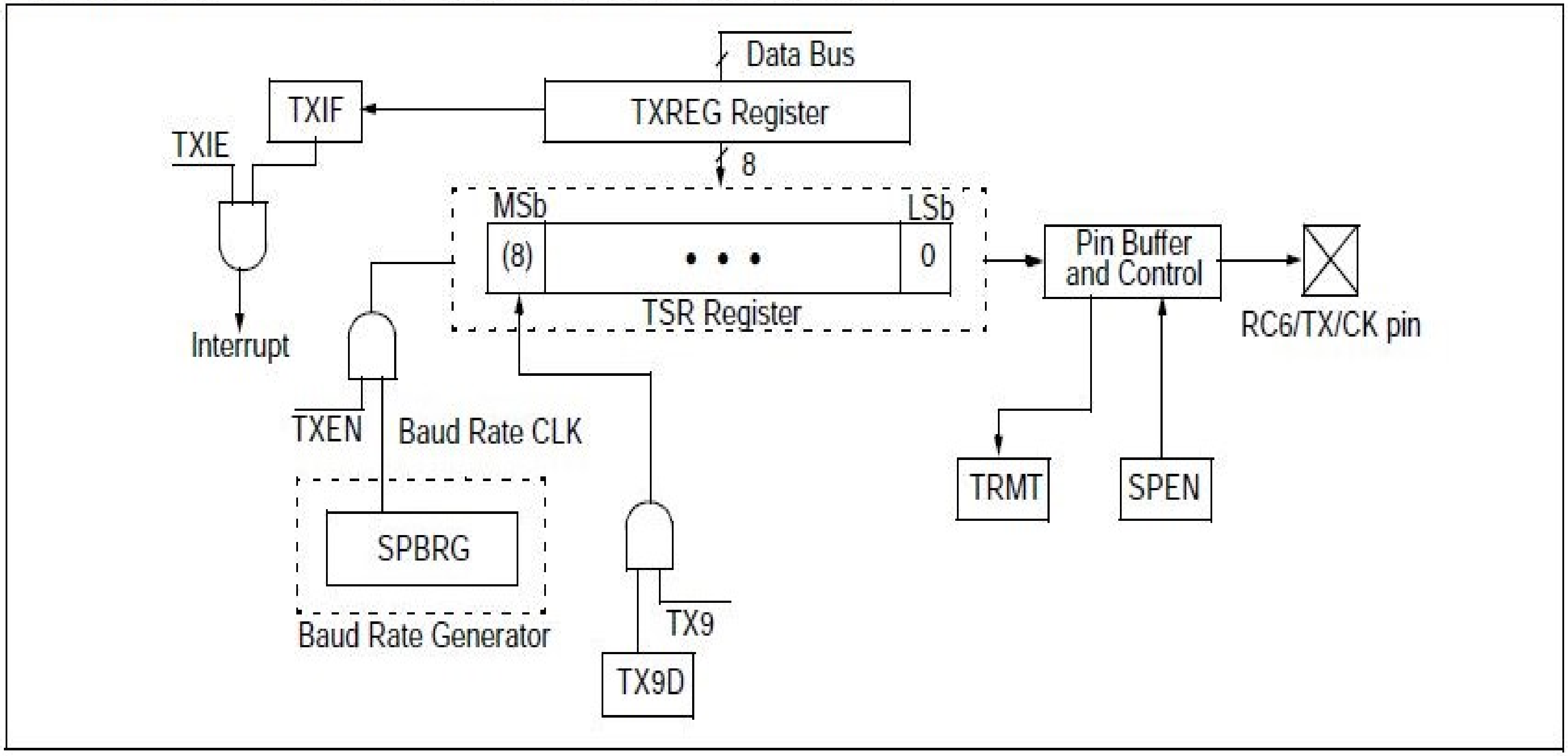
- UART
- ADC
- INTERRUPT
- TIMER
- PWM

Universal Asynchronous Receiver Transmitter

Control Registers

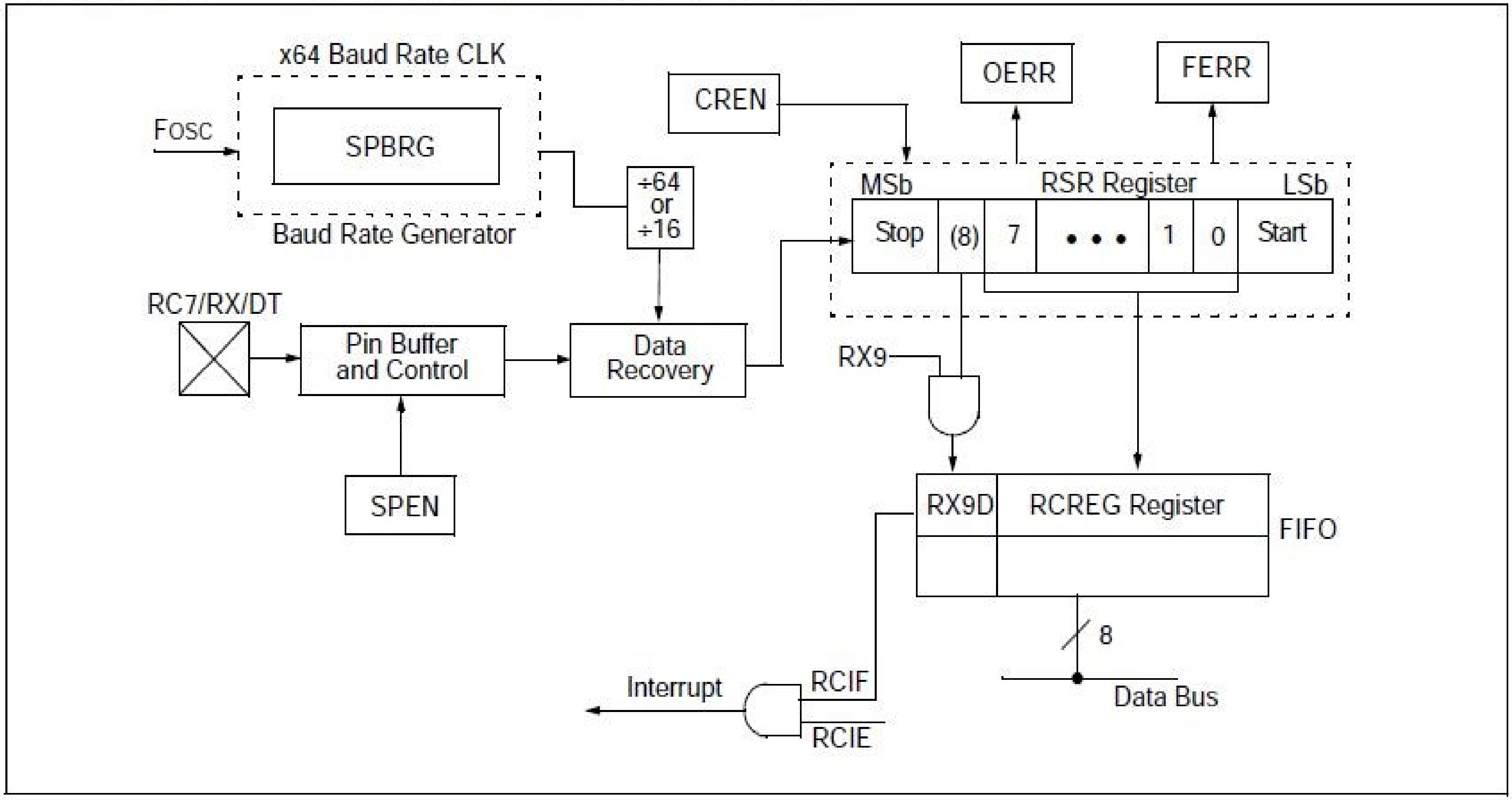
❖ **TXSTA**	- Transmit Status and Control Register
❖ **RCSTA**	- Receive Status and Control Register
❖ **SPBRG**	- Baud Rate Generator Register
❖ **TXIE**	- USART Transmit Interrupt Enable Bit
❖ **RCIE**	- USART Receive Interrupt Enable Bit
❖ **RCIF**	- USART Receive Interrupt Flag Bit
❖ **RCREG**	- USART Receive Register
❖ **TXREG**	- USART Transmit Register
❖ **TRMT**	- Transmit Shift Register Status Bit

FIGURE 10-1: USART TRANSMIT BLOCK DIAGRAM

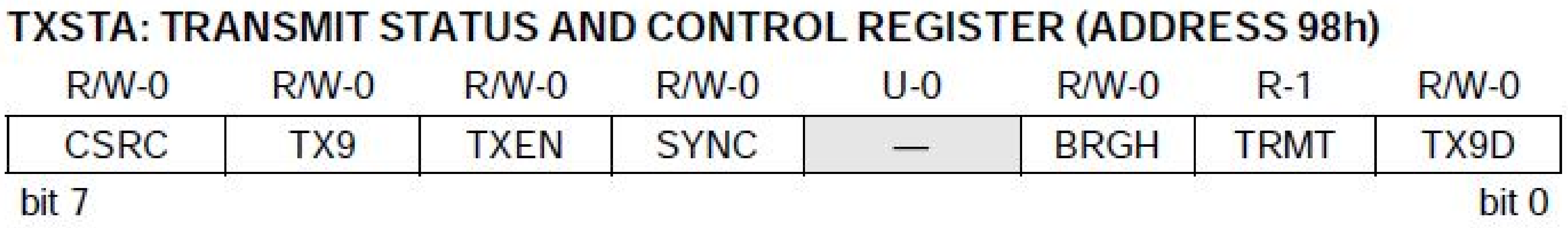


QUEST INNOVATIVE SOLUTIONS

FIGURE 10-4: USART RECEIVE BLOCK DIAGRAM



TXSTA: Transmit Status and Control Register

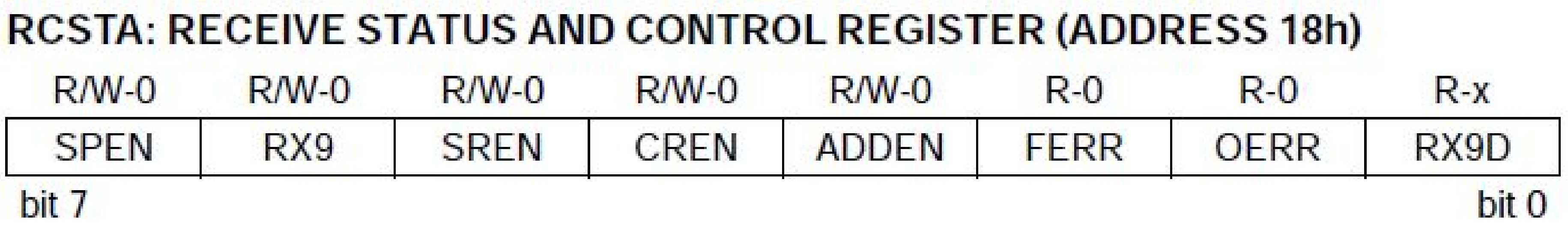


TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (ADDRESS 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

- **TXEN: Transmit Enable bit**
  - 1 = Transmit enabled
  - 0 = Transmit disabled
- **BRGH: High Baud Rate Select bit**
  - Asynchronous mode:
    - 1 = High speed
    - 0 = Low speed

RCSTA: Receive Status and Control Register



RCSTA: RECEIVE STATUS AND CONTROL REGISTER (ADDRESS 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

- **SPEN : Serial Port Enable bit**
  - 1 = Serial port enabled (configures RC7/RX/DT and RC6/TX/CK pins as serial port pins)
  - 0 = Serial port disabled
- **CREN : Continuous Receive Enable bit**
  - Asynchronous mode:
    - 1 = Enables continuous receive
    - 0 = Disables continuous receive

SPBRG: Serial Port Baud Rate Generator

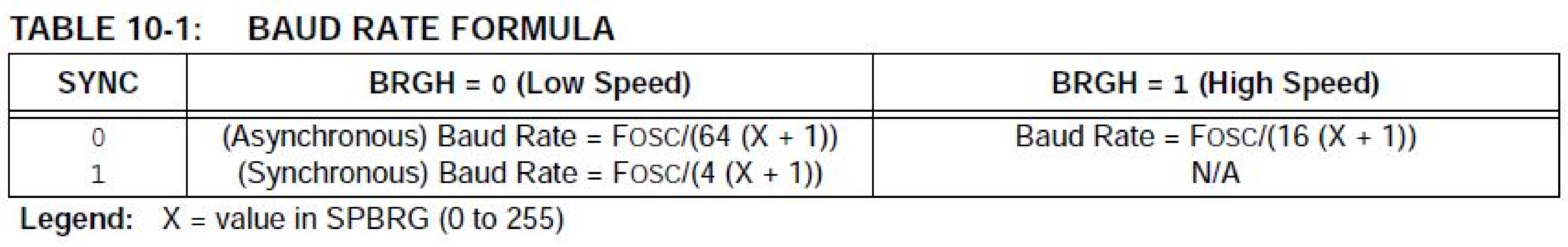


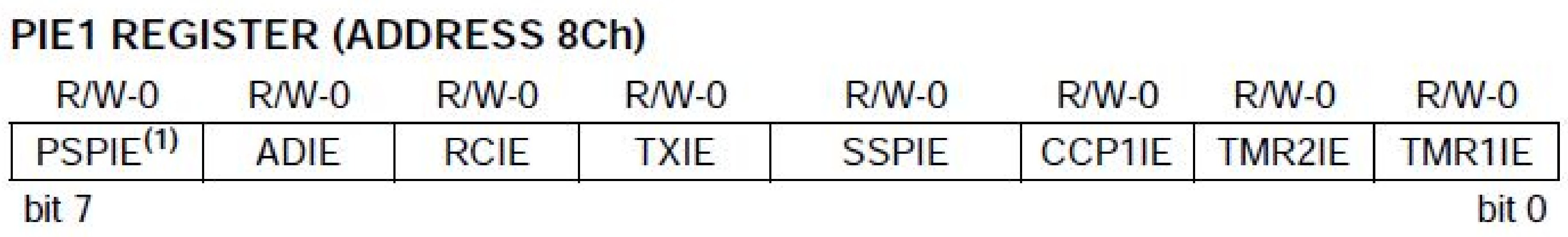
TABLE 10-1: BAUD RATE FORMULA

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = FOSC/(64 (X + 1))	Baud Rate = FOSC/(16 (X + 1))
1	(Synchronous) Baud Rate = FOSC/(4 (X + 1))	N/A

Legend: X = value in SPBRG (0 to 255)

- **SPBRG = 25** (for 9600 bits/second)

PIE1: Peripheral Interrupt Enable Register

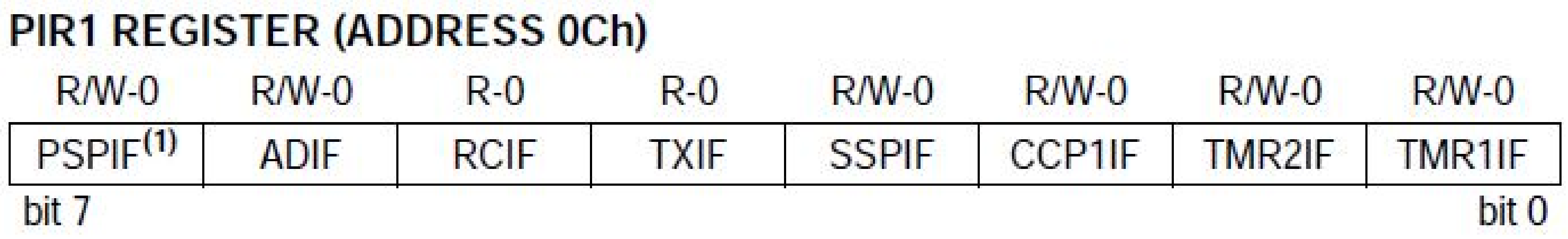


PIE1 REGISTER (ADDRESS 8Ch)

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

- **RCIE : USART Receive Interrupt Enable bit**
  - 1 = Enables the USART receive interrupt
  - 0 = Disables the USART receive interrupt
- **TXIE : USART Transmit Interrupt Enable bit**
  - 1 = Enables the USART transmit interrupt
  - 0 = Disables the USART transmit interrupt

PIR1: Peripheral Interrupt Flag Register



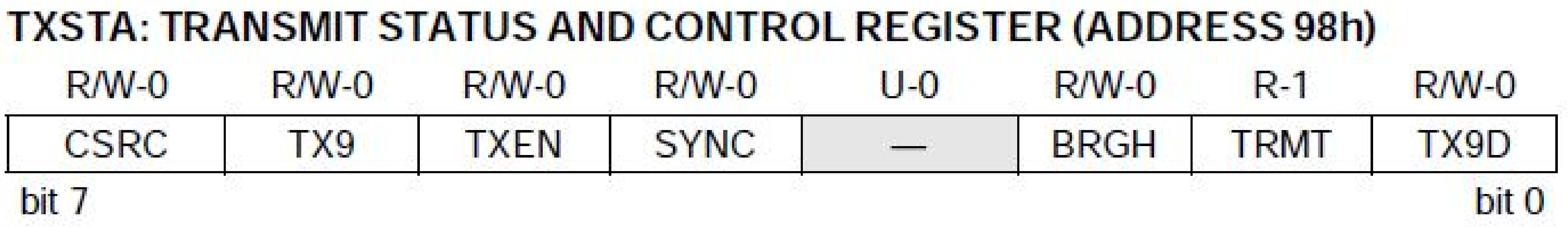
PIR1 REGISTER (ADDRESS 0Ch)

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

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

QUEST INNOVATIVE SOLUTIONS

TXSTA: Transmit Status and Control Register



TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (ADDRESS 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

- **TRMT: TRMT: Transmit Shift Register Status bit**
  - 1 = TSR empty
  - 0 = TSR full

- RCREG - USART Receive Register
- TXREG - USART Transmit Register

UART Rx and Tx Code

```
void main()
{
    TXSTA = 0x24;
    RCSTA = 0x90;
    SPBRG = 25;
    TXIE  = 1;   // Transmit is enabled
    RCIE  = 1;
    char a;
    while(1)
    {
        if(RCIF==1)
        {
                RCIF = 0
                 a = RCREG;
                TXREG = a;
                while(!TRMT); // if transmit shift register status bit is full
        }
    }
}
```

QUEST INNOVATIVE SOLUTIONS

TASK

- Transmit and Receive string using UART
- Create Functions for Tx and Rx
- Write a program to detect password

Analog to Digital Converter

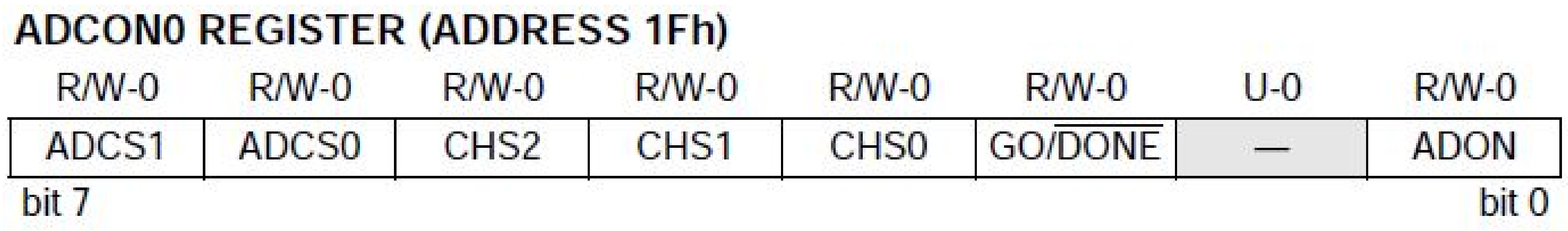
ADC : Analog to Digital Converter

- The Analog-to-Digital (A/D) Converter module has five inputs.
- The conversion of an analog input signal results in a corresponding 10-bit digital number.

Control Registers

- The A/D module has **four** registers. These registers are:
  - **ADRESH**: A/D Result High Register
  - **ADRESL**: A/D Result Low Register
  - **ADCON0**: A/D Control Register 0
  - **ADCON1**: A/D Control Register 1

ADCON0



ADCON0 REGISTER (ADDRESS 1Fh)

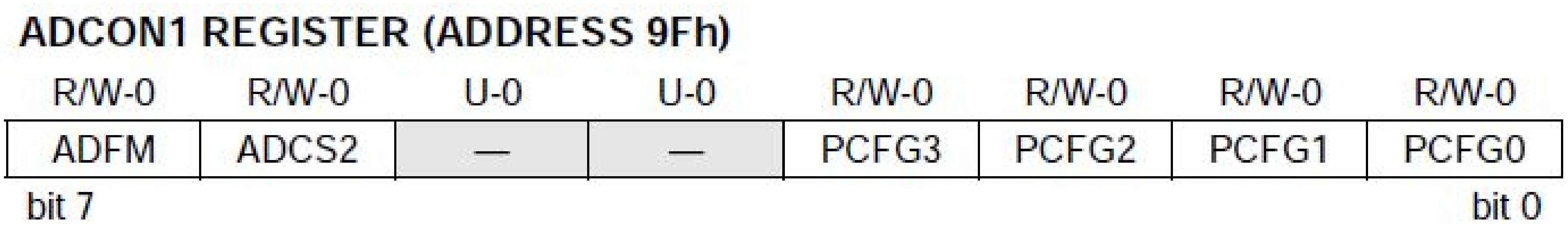
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

- ADCON0 Register controls the operation of the A/D module.
  - **ADCS1:ADCS0** – '0 1' (FOSC/16)
  - **CHS2:CHS0**: Analog Channel Select bits
    - 000 = Channel 0 (AN0)
  - **ADON**: A/D On bit
    - 1 = A/D converter module is powered up
    - 0 = A/D converter module is shut-off

ADCON1

- ADCON1 register **configures** the functions of the **port pins**.
- The port pins can be configured as analog inputs (RA3 can also be the voltage reference) or as digital I/O.

ADCON1



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

- **ADFM** - A/D Result Format Select bit
  - **1** = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
- **ADCS2** - A/D Conversion Clock Select bit
  - **ADS2: ADS1: ADS0** – **1 0 1**
- **PCFG3:PCFG0** - A/D Port Configuration Control bits
  - **1110** - AD0 will be analog pin and all other will be set as digital pins

QUEST INNOVATIVE SOLUTIONS

G0_DONE

- **GO/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

ADC with LED

```
void main()
{
    TRISA0= 1;
    TRISB = 0x00;
    PORTB = 0x00;
    TRISC = 0x00;
    PORTC = 0x00;
    ADCON0 = 0x41;  // FOSC/16, AN0 , For TAD=4us
    ADCON1 = 0xCE;
    while(1)
    {
        GO_DONE = 1;  // start the conversion
        while(GO_DONE); // wait for the conversion to end
        PORTB = ADRESL;  // 0-7
        PORTC = ADRESH;  // 0-1
    }
}
```

ADC with UART

```
GO_DONE = 1;
while(GO_DONE);
a = ADRESL|(ADRESH<<8);
for(i=0;i<=3;i++)
{
    a = a/10;
    r = a%10;
    b[3-i] = r+'0';
}
b[i] = '\0';
uart_tx(b);
```

2 Channel ADC with UART

- *TRISA = 0x03;*
- ***Adc0:***
  - *ADCON0= 0x41;*
  - *ADCON1= 0xCE*
- ***Adc1:***
  - *ADCON0= 0x49;*
  - *ADCON1= 0xC4; // AN0, AN1*

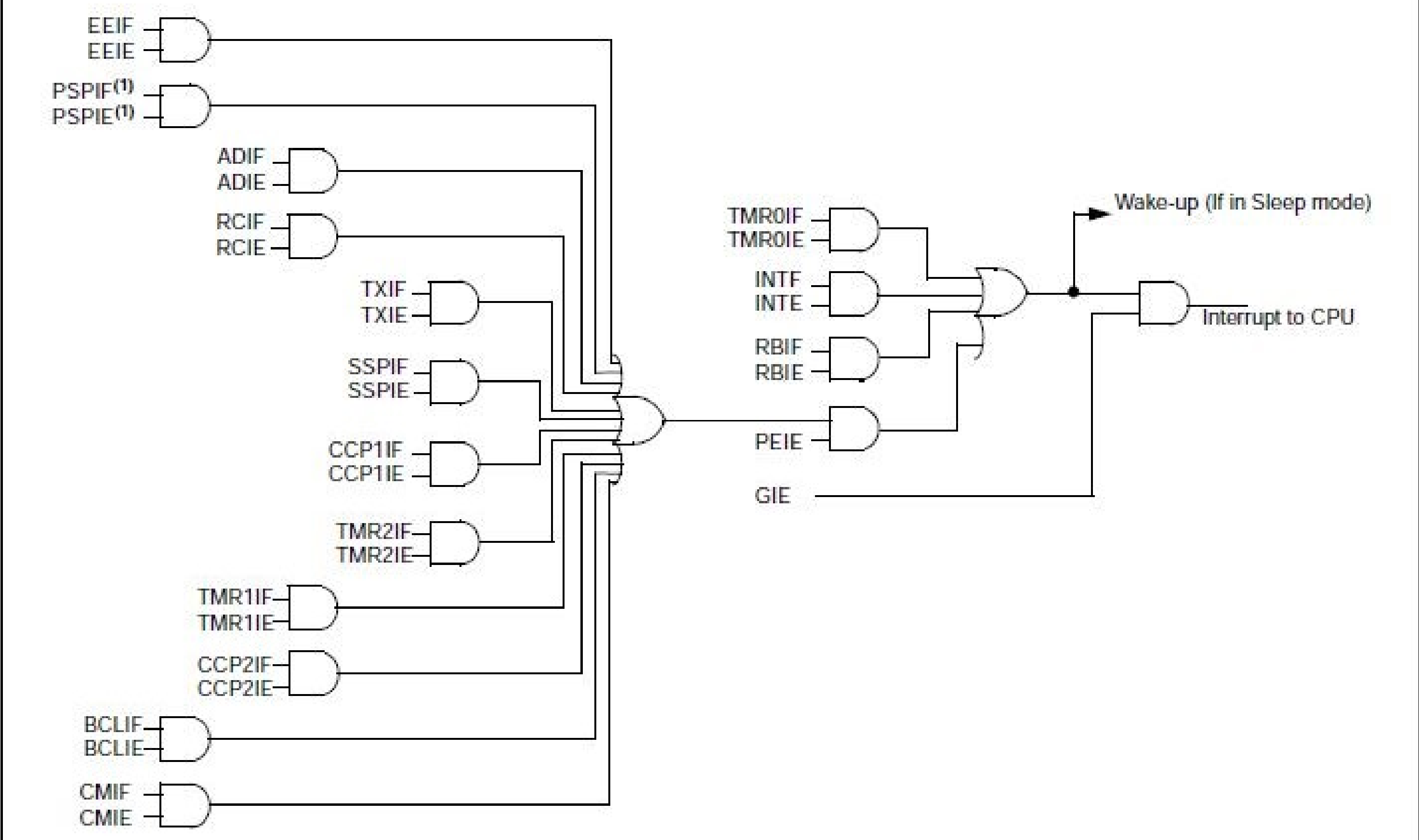
INTERRUPTS

Polling and Interrupt method:

- **Polling:**
  - It is the process where the computer or controlling device waits for an external device to check for its state.
- **Interrupt:**
  - It is the process where the computer or controlling device to temporarily work on a different task, and then return to its previous task.
- **PIC16F876A has 14 interrupt sources**
- **PIC16F877A has 15 interrupt sources**

15 Interrupt Sources

FIGURE 14-10: INTERRUPT LOGIC



Note 1: PSP interrupt is implemented only on PIC16F874A/877A devices.

15 Interrupt Sources

1. **EEIE**:	EEPROM Write Operation Interrupt Enable bit
2. **PSPIE**:	Parallel Slave Port Read/Write Interrupt Enable bit
3. **ADIE**:	A/D Converter Interrupt Enable bit
4. **RCIE**:	USART Receive Interrupt Enable bit
5. **TXIE**:	USART Transmit Interrupt Enable bit
6. **SSPIE**:	Synchronous Serial Port Interrupt Enable bit
7. **CCP1IE**:	CCP1 Interrupt Enable bit
8. **TMR2IE**:	TMR2 to PR2 Match Interrupt Enable bit

15 Interrupt Sources

9.	**TMR1IE**:	TMR1 Overflow Interrupt Enable bit
10.	**CCP2IE**:	CCP2 Interrupt Enable bit
11.	**BCLIE**:	Bus Collision Interrupt Enable bit
12.	**CMIE**:	Comparator Interrupt Enable bit
13.	**TMR0IE**:	TMR0 Overflow Interrupt Enable bit
14.	**INTE**:	RB0/INT External Interrupt Enable bit
15.	**RBIE**:	RB Port Change Interrupt Enable bit

- ***CMIE: Comparator Interrupt is not available in PIC16F876A***

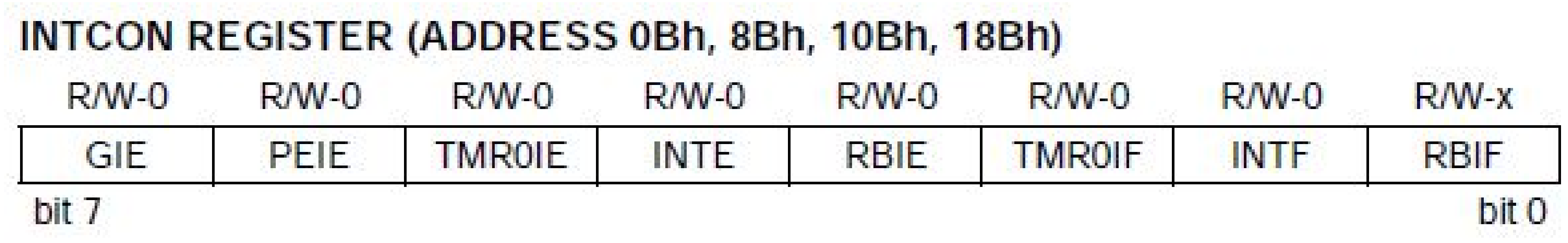
Interrupts

- **External Interrupt:** External pin - RB0/INT
- **UART Interrupt:** UART pins – RC6 & RC7
- **Timer Interrupts:** Timer0, Timer1, Timer2

EXTERNAL INTERRUPT

- Control Register:
  - INTCON

INTCON Register:


INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

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	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

- **GIE = Global Interrupt Enable bit**
  - 1= Enables all unmasked interrupts
  - 0=Disables all interrupts
- **INTE= RB0/INT External Interrupt Enable bit**

  1 = Enables the RB0/INT external interrupt

  0 = Disables the RB0/INT external interrupt

INTCON Register:

- **INTF:** **RB0/INT External Interrupt Flag bit**
  - 1 = The RB0/INT external interrupt occurred (must be cleared in software)
  - 0 = The RB0/INT external interrupt did not occur

External Interrupt Code:

```
static void interrupt isr()
{
  if(INTF)
   {
     INTF = 0;
    RC2 = ~RC2;
   }
}
```

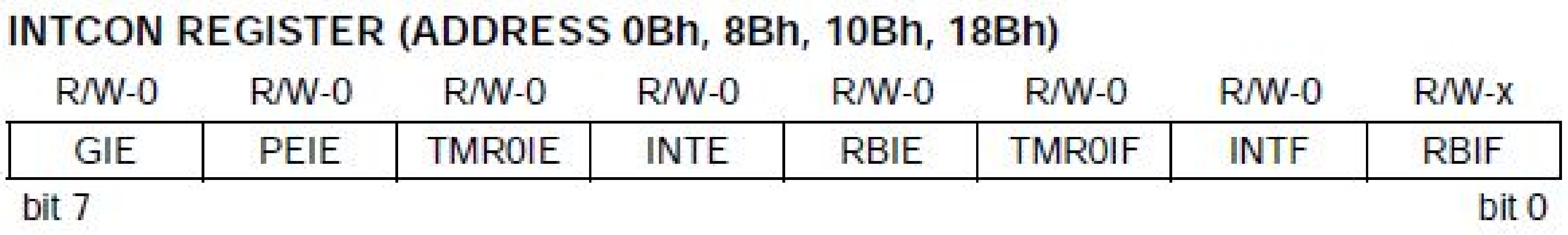
```
void main()
{
        INTCON = 0x90;
        TRIC2 = 0;
        RC2 = 0;
        while(1)
        {

        }
}
```

UART INTERRUPT

- Control Register:
  - INTCON
  - PIE1
  - PIR1

INTCON Register

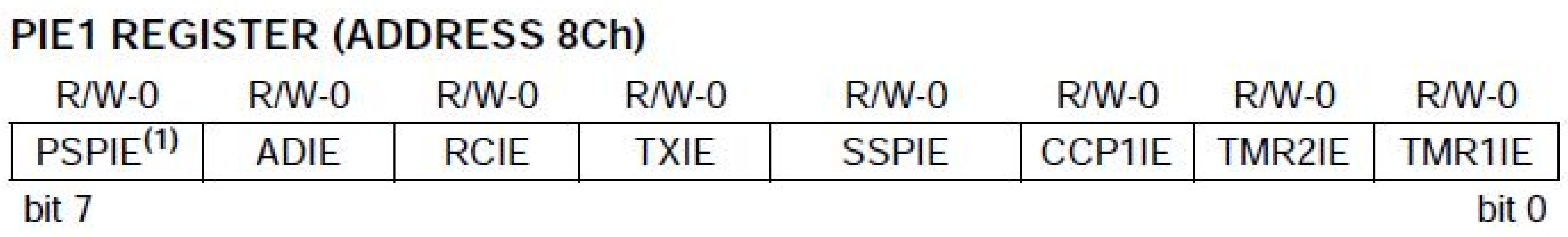


INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

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	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

- **GIE = Global Interrupt Enable bit**
  - 1= Enables all unmasked interrupts
  - 0=Disables all interrupts
- **PEIE: Peripheral Interrupt Enable bit**
  - 1 = Enables all unmasked peripheral interrupts
  - 0 = Disables all peripheral interrupts

PIE1: Peripheral Interrupt Enable Register

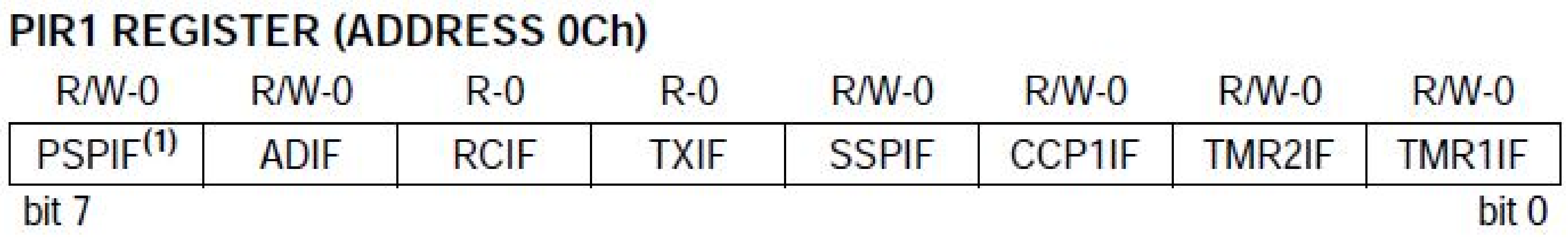


PIE1 REGISTER (ADDRESS 8Ch)

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

- **RCIE : USART Receive Interrupt Enable bit**
  - 1 = Enables the USART receive interrupt
  - 0 = Disables the USART receive interrupt
- **TXIE : USART Transmit Interrupt Enable bit**
  - 1 = Enables the USART transmit interrupt
  - 0 = Disables the USART transmit interrupt

PIR1: Peripheral Interrupt FLAG 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

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

QUEST INNOVATIVE SOLUTIONS

UART Code

```
static void interrupt isr()
{
if(RCIF)
{
    RCIF = 0;
    a = uart_rx();
}
}
```

```
void main()
{
        GIE=1;
        PEIE=1;
        TXSTA = 0x24;
        RCSTA = 0x90;
        SPBRG = 25;
        //TXIE  = 1;
        RCIE  = 1;
}
```

TIMERS

TIMERS

- **PIC16F876A has 3 Timers**
  - TIMER0 - 8 bit
  - TIMER1 - 16 bit
  - TIMER2 - 8 bit

TIMER0

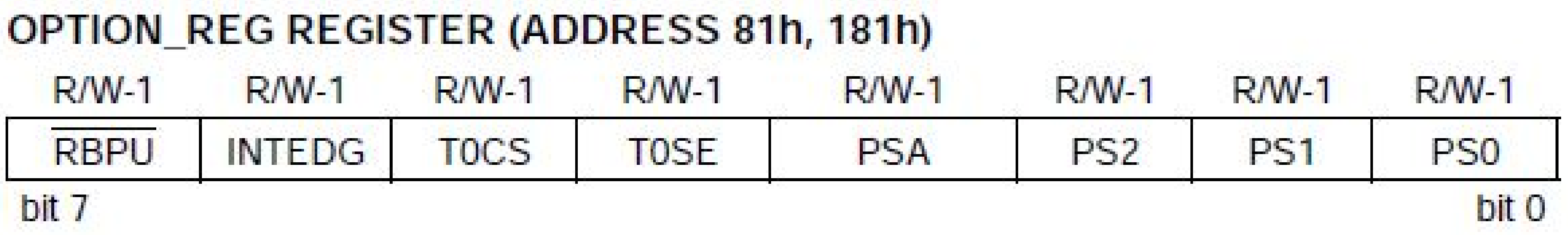
- **Features of Timer0 module:**
  - 8-bit timer/counter
  - Readable and writable
  - 8-bit software programmable prescaler
  - Interrupt on overflow from FFh to 00h

TIMER0 Registers

- **OPTION_REG:** Configuration Register
- **TMR0:** Load the count value
- **INTCON:** Wait for overflow flag to set

  Clear the overflow flag

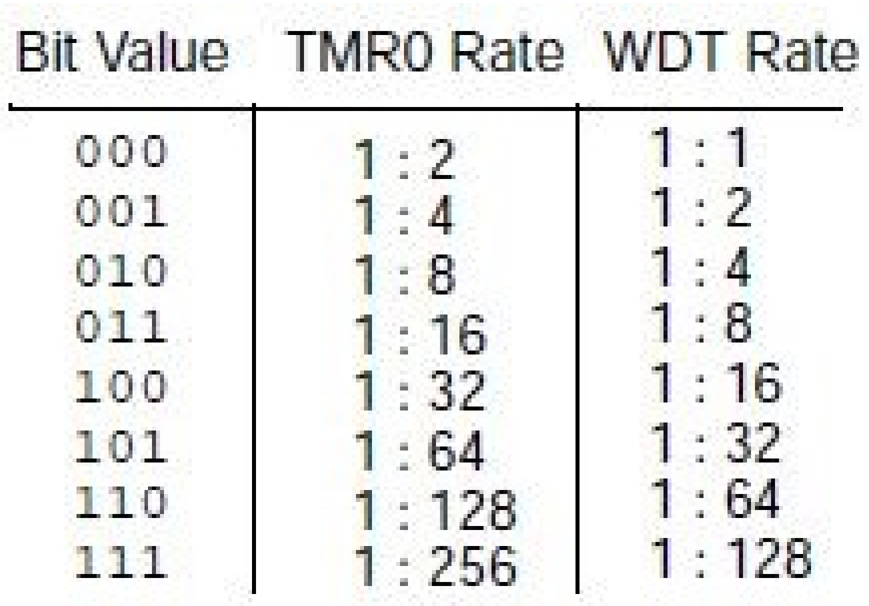
OPTION REGISTER



OPTION_REG REGISTER (ADDRESS 81h, 181h)

	bit 7						bit 0
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

- **PS2:PS0 - Prescaler Rate Select Bits**



Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

OPTION REGISTER

- **PSA - Prescaler Assignment bit**
  - 1 = Prescaler is assigned to the WDT
  - 0 = Prescaler is assigned to the Timer0 module
- **T0SE - TMR0 Source Edge Select bit**
  - 1 = Increment on high-to-low transition on T0CKI pin
  - 0 = Increment on low-to-high transition on T0CKI pin
- **T0CS - TMR0 Clock Source Select bit**
  - 1 = Transition on T0CKI pin **(RA4) - (Counter Mode)**
  - 0 = Internal instruction cycle clock (CLKOUT) - **(Timer Mode)**

TMR0 Value

- **TMR0 Reg value**

  = 256 – ((Delay * Fosc)/(Prescalar * 4))

  or

- **Delay = $\dfrac{\textbf{(Prescaler * 4) * (256 – TMR0 Reg Value)}}{\textbf{Fosc}}$**

- **Minimum Delay using TMR0:**

$$= \frac{(1*4)*(256-255)}{4*10^6}$$

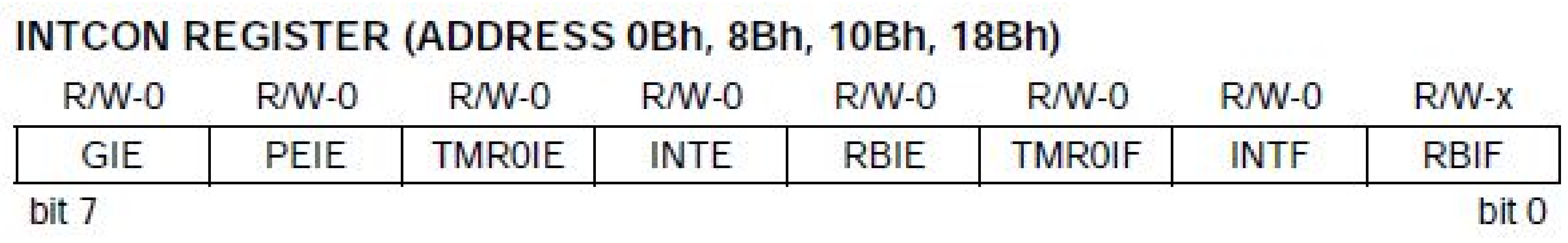
$$= 1 \text{ usec}$$

- **Maximum Delay using TMR0:**

$$= \frac{(256*4)*(256-0)}{4*10^6}$$

$$= 65.536 \text{ msec}$$

INTCON Register:


INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

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	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

- **GIE = Global Interrupt Enable bit**
  - 1= Enables all unmasked interrupts
  - 0=Disables all interrupts
- **PEIE - Peripheral Interrupt Enable bit**
  - 1 = Enables all unmasked peripheral interrupts
  - 0 = Disables all peripheral interrupts

INTCON Register:

- **TMR0IE - TMR0 Overflow Interrupt Enable bit**
  - 1 = Enables the TMR0 interrupt
  - 0 = Disables the TMR0 interrupt
- **TMR0IF - TMR0 Overflow Interrupt Flag bit**
  - 1 = TMR0 register has overflowed (must be cleared in software)
  - 0 = TMR0 register did not overflow

Timer0 Program for 1Sec Delay

```
void main()
{
    TRISC0=0;
    RC0=0;
    int count=0;
    TMR0=251;
    OPTION_REG=0x07;
    INTCON=0x20;
    while(1)
    {
        if(TMR0IF==1)
        {
            count++;
            TMR0IF=0;
            TMR0=251;
            if(count==1000)
            {
                RC0=~RC0;
                count=0;
            }
        }
    }
}
```

TASK

1. Timer0 as **Timer** with **min** Prescaler
2. Timer0 as **Timer** with **max** Prescaler
3. Timer0 as **counter** with **min** Prescaler
4. Generate **1Sec** Delay using Timer0 **(Polling Method)**
5. Generate **1Sec** Delay using Timer0 **(Interrupt Method)**

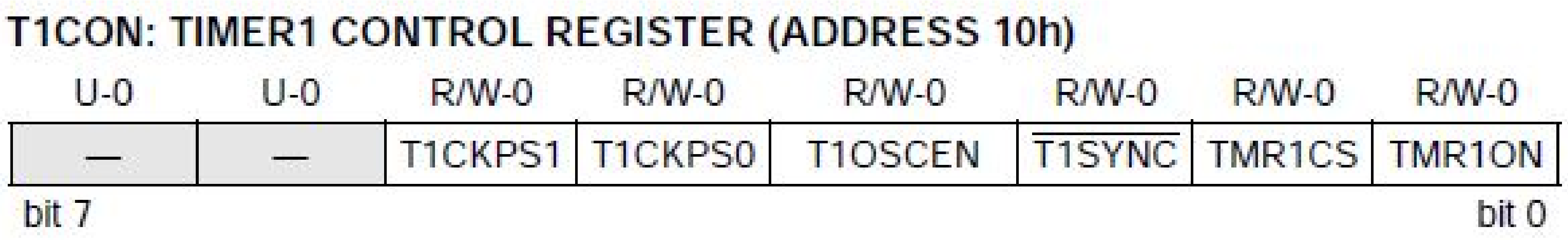
TIMER1

- The Timer1 module is a 16-bit timer consisting of two 8-bit registers (**TMR1H** and **TMR1L**)
- The TMR1 Register pair (TMR1H:TMR1L) increments from 0000h to FFFFh

TIMER1 Registers

- **T1CON:** Configuration Register
- **TMR1L & TMR1H:** Count Registers
- **PIR1: (0th bit)** Overflow flag
- **PIE1: (0th bit)** Interrupt Enable
- **INTCON:** Interrupt Control

T1CON Register



T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

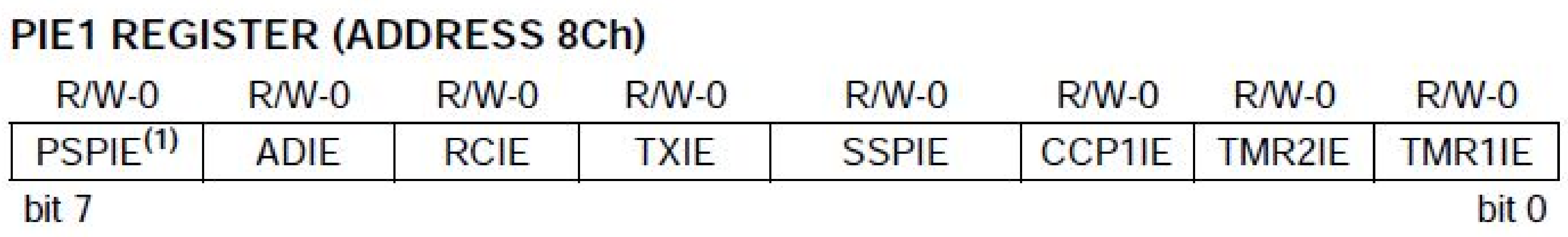
U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

- **TMR1ON - Timer1 On bit**
  - 1 = Enables Timer1
  - 0 = Stops Timer1
- **TMR1CS - Timer1 Clock Source Select bit**
  - 1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
  - 0 = Internal clock (FOSC/4)

T1CON Register

- **T1OSCEN - Timer1 Oscillator Enable Control bit**
  - 1 = Oscillator is enabled
  - 0 = Oscillator is shut off
- **T1CKPS1:T1CKPS0 - Timer1 Input Clock Prescale Select bits**
  - 1 1 = 1:8 prescale value
  - 1 0 = 1:4 prescale value
  - 0 1 = 1:2 prescale value
  - 0 0 = 1:1 prescale value

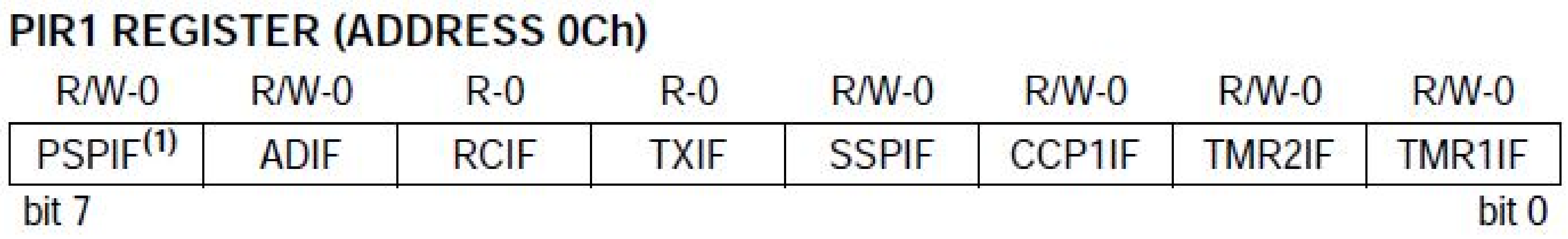
PIE1: Peripheral Interrupt Enable Register


PIE1 REGISTER (ADDRESS 8Ch)

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

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

PIR1: Peripheral Interrupt Flag Register

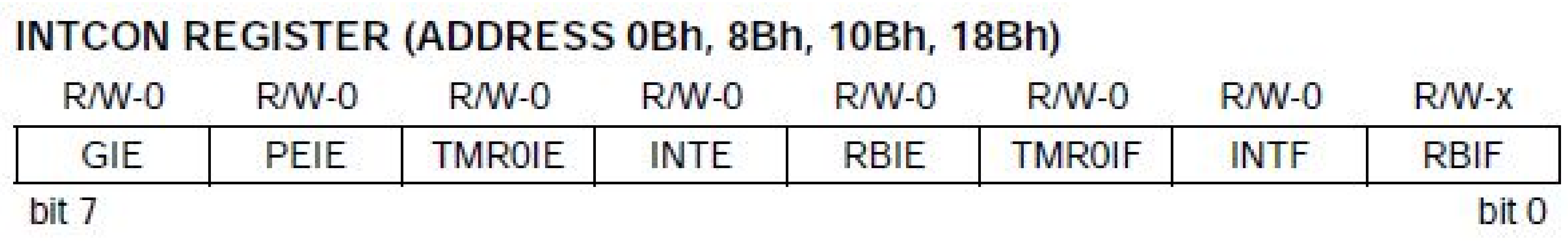


PIR1 REGISTER (ADDRESS 0Ch)

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

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

INTCON Register:


INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

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	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

- **GIE = Global Interrupt Enable bit**
  - 1= Enables all unmasked interrupts
  - 0=Disables all interrupts

TMR1 Value

- **TMR0 Reg value**

  = 65536 – ((Delay * Fosc)/(Prescalar * 4))

  or

- **Delay =**

$$\frac{(\text{Prescaler} * 4) * (65536 - \text{TMR1 Reg Value})}{\text{Fosc}}$$

- **Minimum Delay using TMR1:**

$$= \frac{(1*4)*(65536-65535)}{4*10^6}$$

$$= 1\ \text{usec}$$

- **Maximum Delay using TMR1:**

$$= \frac{(8*4)*(65536-0)}{4*10^6}$$

$$= 524.288\ \text{msec}$$

TMR1 Program for 1second delay

```
void main()
{
    TRISB=0x00;
    PORTB=0xFF;
    int cnt=0;
    TMR1H=0xD8; // for 1ms
    TMR1L=0xEF;
    T1CON=0x01;
    while(1)
    {
        while(!TMR1IF);
        TMR1IF=0;
        cnt++;
        if(cnt==1000)
        {
            PORTB=~PORTB;
            cnt=0;
        }
    }
}
```

TIMER2

- Features:
  - Two 8-bit Registers (TMR2 and PR2)
  - Readable and Writable
  - A Prescaler and a Postscaler
  - Connected only to an Internal Clock - 4 MHz Crystal
  - Interrupt on overflow

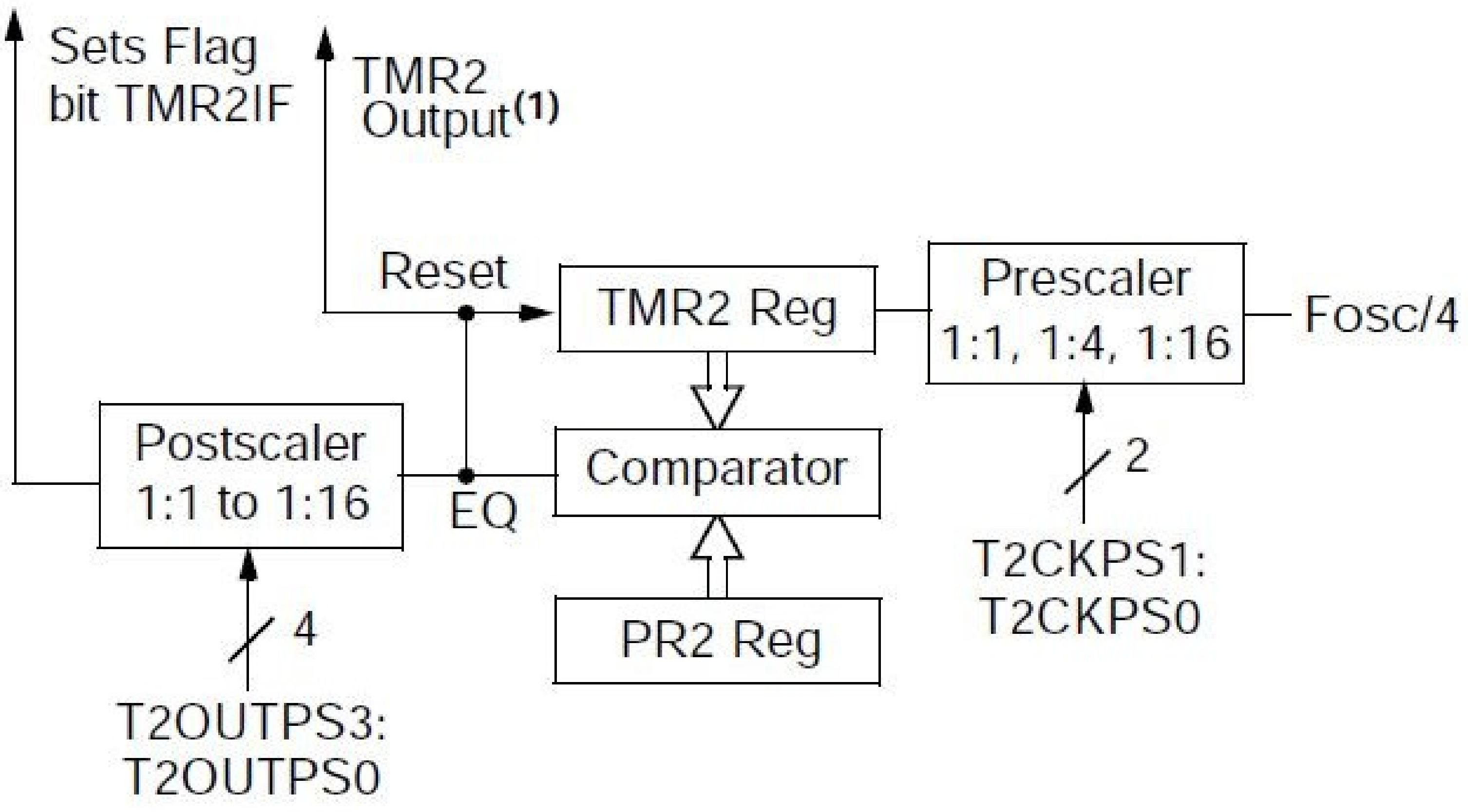
TIMER2

- Timer2 is an **8-bit timer** with a **prescaler** and a **postscaler**.
- Timer2 can be used as the **PWM time** base for the **PWM mode** of the **CCP module(s).**
- The Timer2 module has an **8-bit period register PR2**.
- Timer2 **increments** from **00h** until it **matches PR2** and then **resets** to **00h** on the next increment cycle.

TIMER2 Registers

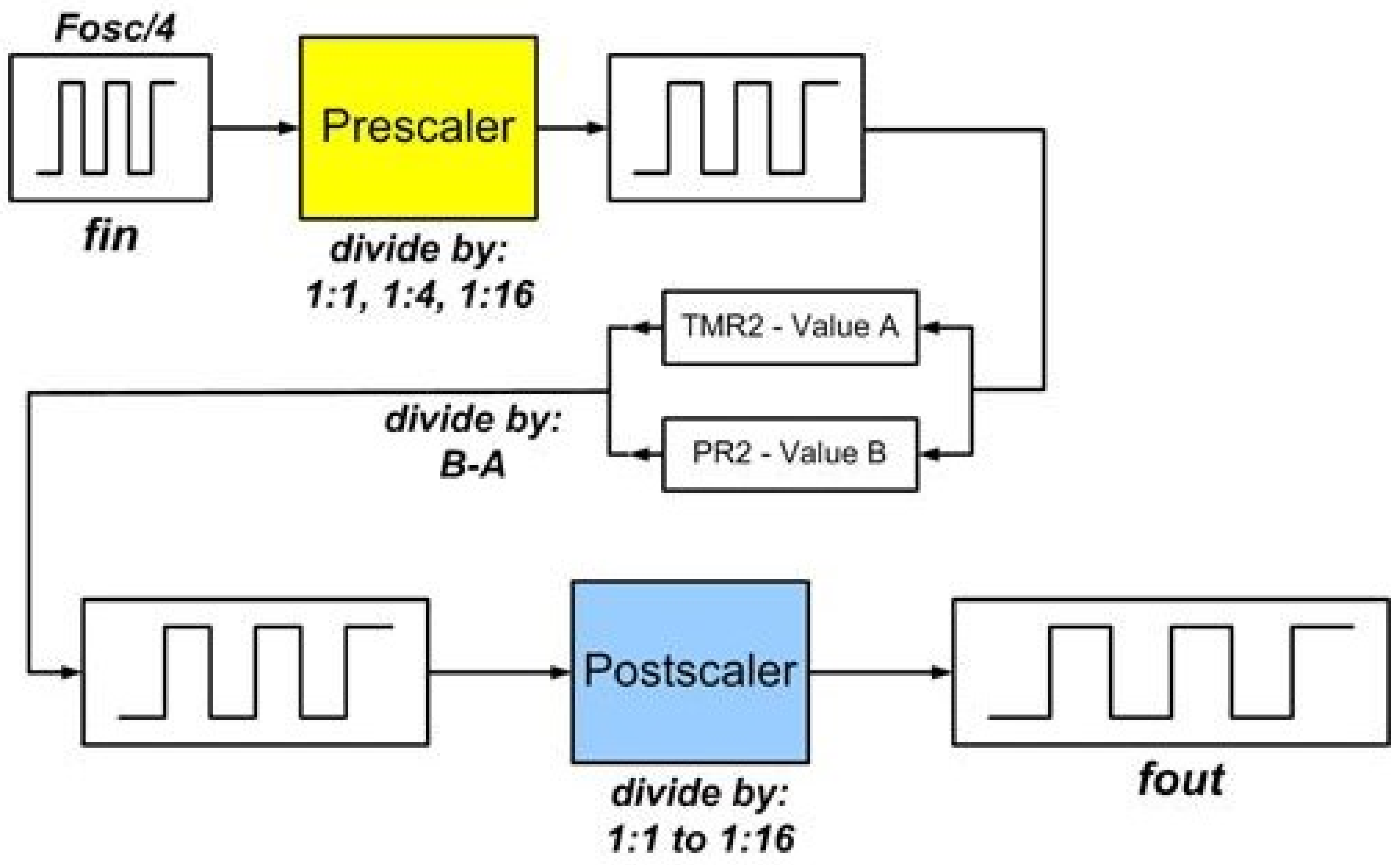
- **T2CON - Configuration Register**
- **TMR2 - Count Register**
- **PR2 - Period Register**
- **PIR1 - (1st BIT) Overflow Flag**
- **PIE1 - (1st BIT) Interrupt Enable**
- **INTCON - Interrupt Control**

Timer2 Block Diagram

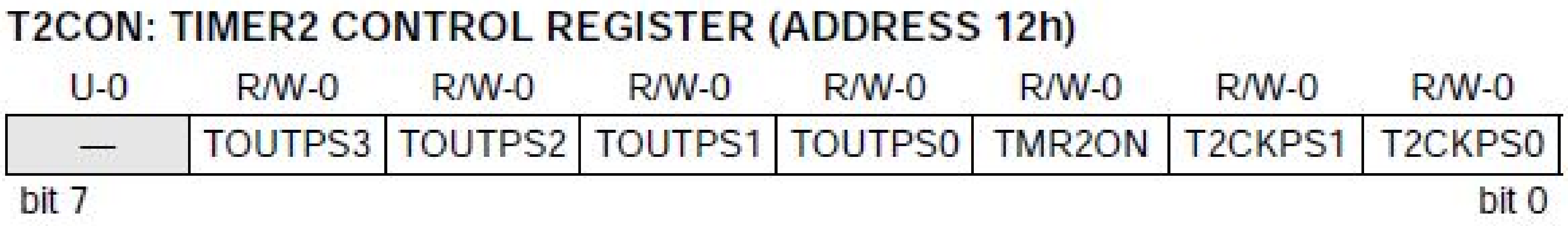


Note 1: TMR2 register output can be software selected by the SSP module as a baud clock.

Timer2 Flow Diagram



T2CON Registers



T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

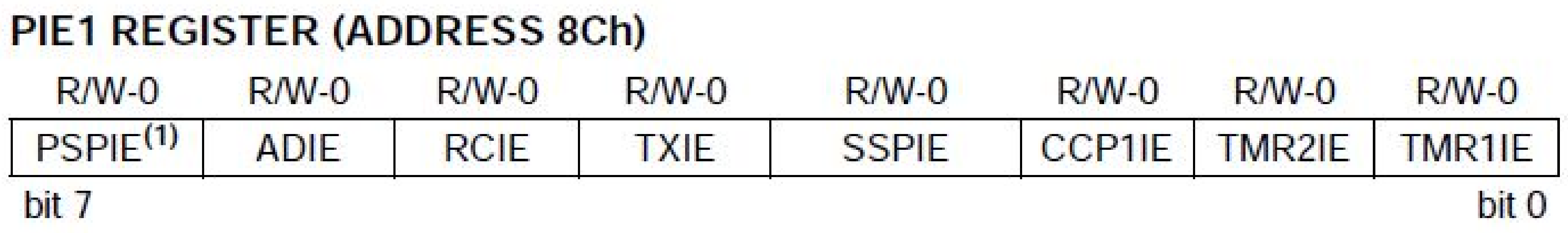
- **T2CKPS1:T2CKPS0 - Timer2 Clock Prescale Select bits**
  - 00 = Prescaler is 1
  - 01 = Prescaler is 4
  - 1x = Prescaler is 16
- **TMR2ON - Timer2 On bit**
  - 1 = Timer2 is on
  - 0 = Timer2 is off

T2CON Registers

- **TOUTPS3:TOUTPS0: Timer2 Output Postscale Select bits**
  - 0000 = 1:1 Postscale
  - 0001 = 1:2 Postscale
  - 0010 = 1:3 Postscale

    ……
  - 1111 = 1:16 Postscale

PIE1: Peripheral Interrupt Enable Register

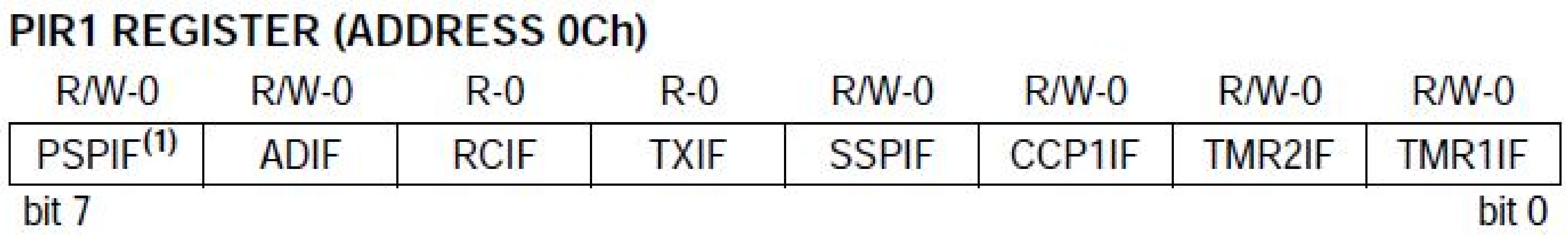


PIE1 REGISTER (ADDRESS 8Ch)

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

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

PIR1: Peripheral Interrupt Flag 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

- **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

TMR2 Value with Min Postscaler

- **TMR2 Reg value**

$$= 256 - ((\text{Delay} * \text{Fosc})/(\text{Prescalar} * 4))$$

or

- **Delay** $= \dfrac{(\text{Prescaler} * 4) * (256 - \text{TMR0 Reg Value})}{\text{Fosc}}$

- **Minimum Delay using TMR2:**

$$= \frac{(1 * 4) * (256 - 255)}{4*10^6}$$

$$= \mathbf{1\ usec}$$

- **Maximum Delay using TMR2:**

$$= \frac{(256 * 4) * (256 - 0)}{4*10^6}$$

$$= \mathbf{65.536\ msec}$$

Find “Count” Value for desired Delay using TMR2:

Eg., Fout = 1 Hz (i.e, 1second Delay [T=1/F])

$$\textbf{Fout} = \frac{\text{Fosc}}{4 * \text{Prescaler} * (\text{PR2-TMR2}) * \text{Postscaler} * \text{Count}}$$

$$\textbf{Count} = \frac{\text{Fosc}}{4 * \text{Prescaler} * (\text{PR2-TMR2}) * \text{Postscaler} * \text{Fout}}$$

Count value using Max Prescaler & Max Postscaler

Fosc = 4 MHz
Fout = 1 Hz
Prescaler = 16
PR2 = 256
TMR2 = 0
Postscaler = 16

$$\text{Count} = \frac{4 * 10^6}{4 * 16 * (256 - 0) * 16 * 1}$$

$$= \mathbf{15}$$

Timer2 Program for 1 second delay

```
#include <xc.h>
void main()
{
    T2CON = 0x7F;
    TMR2 = 0;
    PR2     = 0xFF;
    int count = 0;
    while(1)
    {
        if(TMR2IF)
        {
                count++;
                TMR2IF=0;
                TMR2=0;
        }
            if(count==15)
            {
                RC2=~RC2;
                count=0;
            }
    }
```

CCP MODULE

CAPTURE / COMPARE / PWM

- **Two** Capture/Compare/PWM modules: **CCP1 & CCP2**
- It contains:
  - **16-bit register** which can operate as a:
    - **16-bit Capture register**
    - **16-bit Compare register**
    - PWM Master/Slave **Duty Cycle register**

- **CCP1** and **CCP2** modules are **identical** in **operation**, with the **exception** being the **operation** of the **special event trigger**.
- In the following sections, the operation of a CCP module is described with respect to CCP1.
- CCP2 operates the same as CCP1 except where noted.

CCP MODE – TIMER RESOURCES REQUIRED

CCP Mode		Timer Resource
Capture	-	Timer1
Compare	-	Timer1
PWM	-	Timer2

CCP1 Module

- Capture/Compare/PWM Register 1 (CCPR1) is comprised of two 8-bit registers:
  - **CCPR1L** (low byte) and **CCPR1H** (high byte).
- The **CCP1CON** register controls the operation of CCP1.
- The special event trigger is generated by a compare match and will reset Timer1.

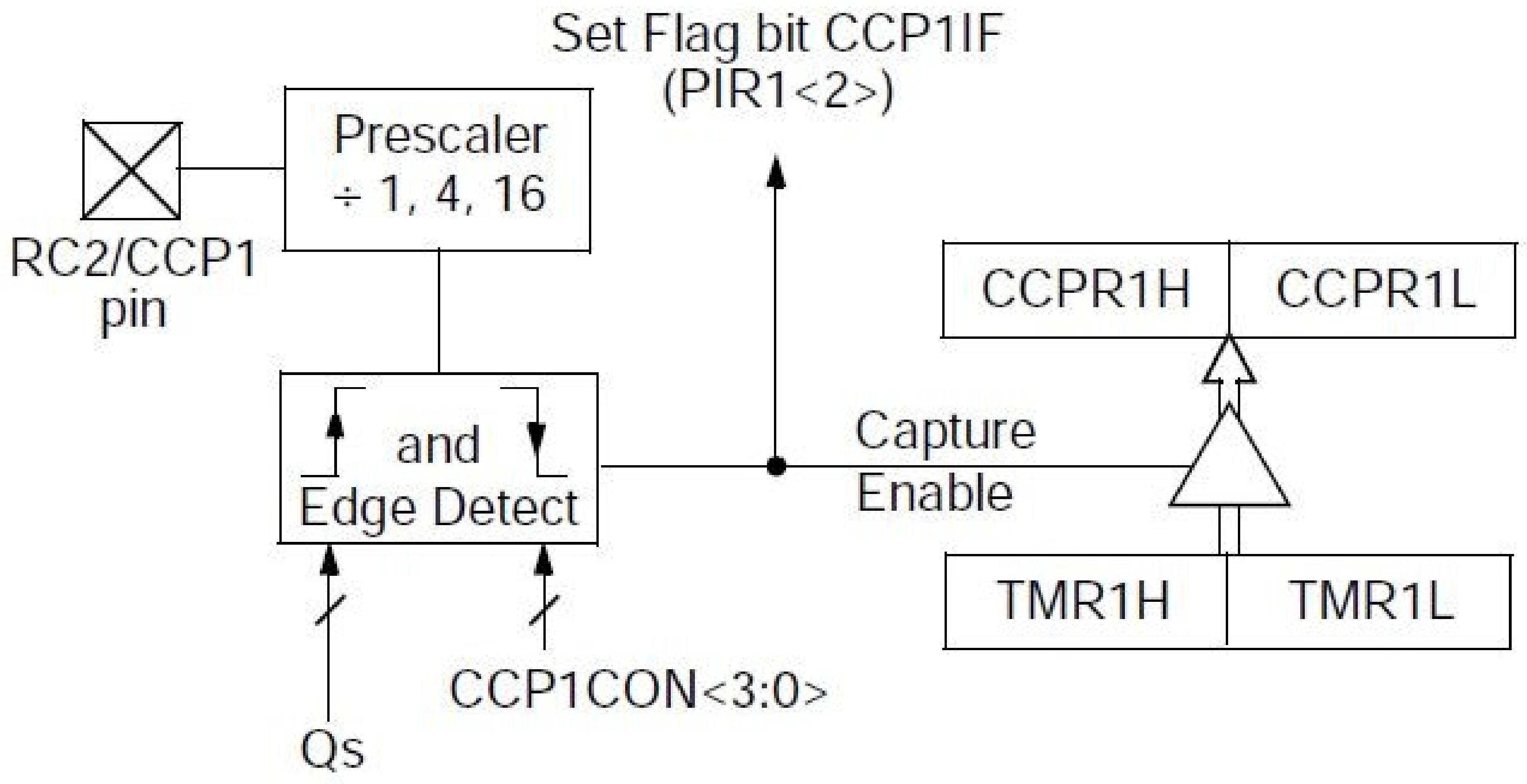
CCP2 Module

- Capture/Compare/PWM Register 2 (CCPR2) is comprised of two 8-bit registers:
  - **CCPR2L** (low byte) and **CCPR2H** (high byte).
- The **CCP2CON** register controls the operation of CCP2.
- The special event trigger is generated by a compare match and will reset Timer1 and start an A/D conversion (if the A/D module is enabled).

Special Event Trigger

- An internal hardware trigger is generated which may be used to initiate an action.
- In CCP1 (Compare Mode):
  - The special event trigger output of CCP1 resets the TMR1 register pair.
- In CCP2 (Compare Mode):
  - The special event trigger output of CCP2 resets the TMR1 register pair and starts an A/D conversion (if the A/D module is enabled).
- The special event trigger from the CCP1 and CCP2 modules will not set interrupt flag bit TMR1IF (PIR1<0>).

CAPTURE Mode Block Diagram

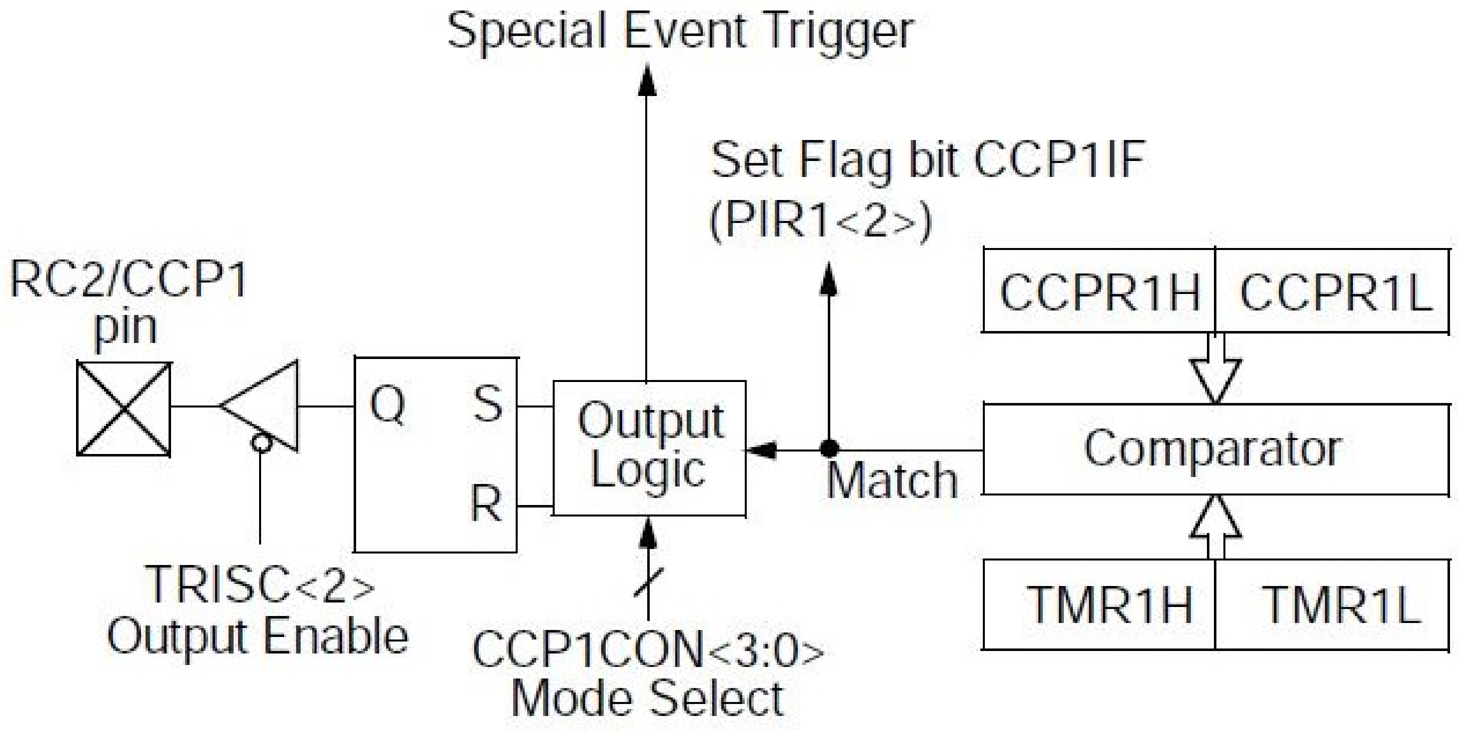


CAPTURE Mode Working

- **CCPR1H:CCPR1L captures** the **16-bit value** of the **TMR1** register when an **event** occurs on pin **RC2/CCP1**.
- An **event** is **defined** as one of the following:
  - Every falling edge
  - Every rising edge
  - Every 4th rising edge
  - Every 16th rising edge

COMPARE Mode Block Diagram

Special event trigger will:
reset Timer1, but not set interrupt flag bit TMR1IF (PIR1<0>)
and set bit GO/DONE (ADCON0<2>).



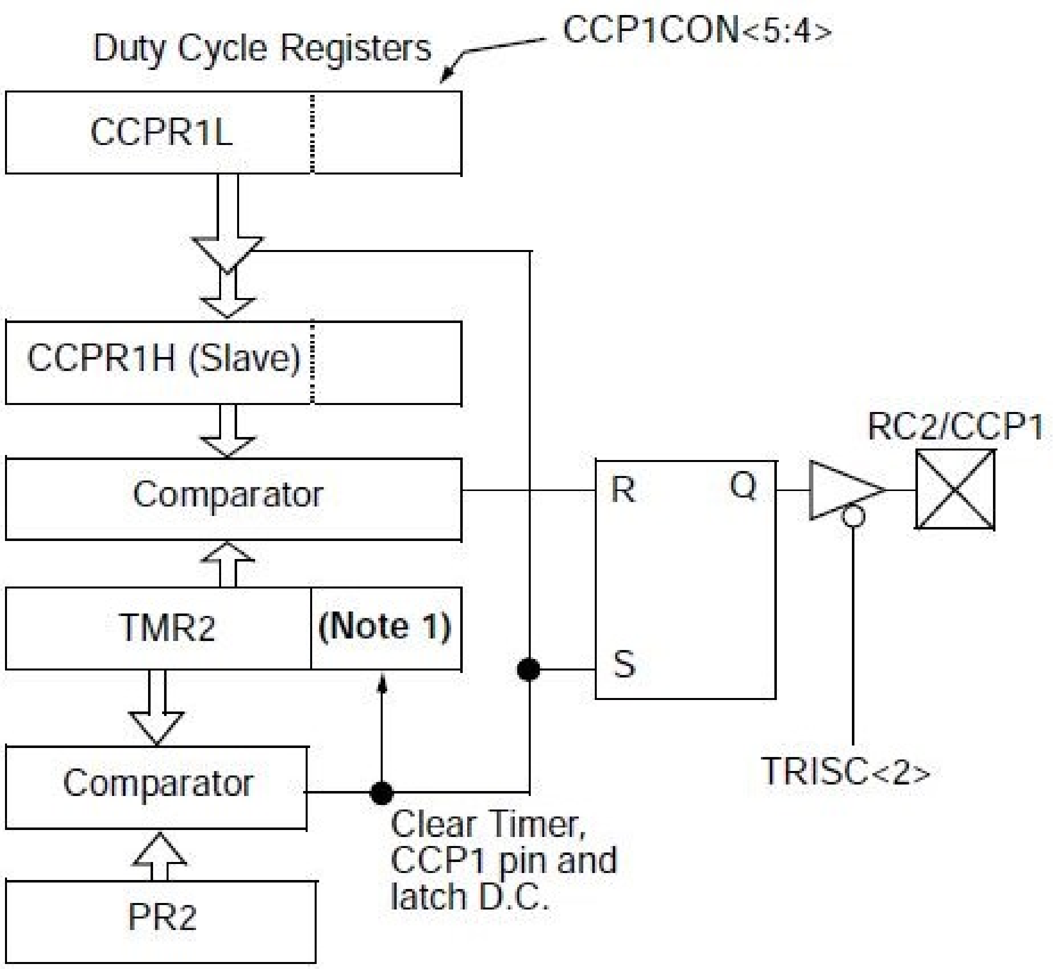
COMPARE Mode Working

- **16-bit CCPR1** register value is constantly compared against the **TMR1** register pair value.
- When a match occurs, the **RC2/CCP1** pin is:
  - Driven high
  - Driven low
  - Remains unchanged
- The action on the pin is based on the value of control bits, **CCP1M3:CCP1M0 (CCP1CON<3:0>).**
- At the same time, interrupt flag bit **CCP1IF** is set.

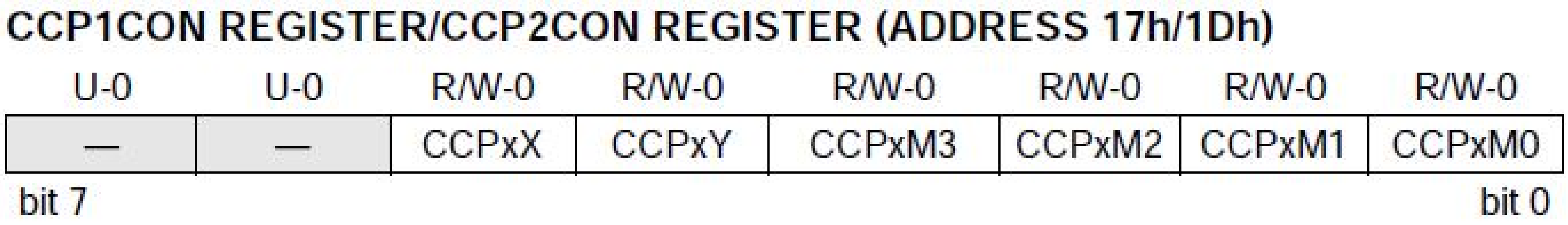
PWM Mode

- In PWM mode, the **RC2/CCP1** pin produces up to a **10-bit resolution PWM output**.
- **TRISC<2>** bit must be cleared to make the **RC2/CCP1** pin an output.
- Configure **CCP1CON** Register.
- Configure **T2CON** Register.

PWM Mode Block Diagram



PWM Registers

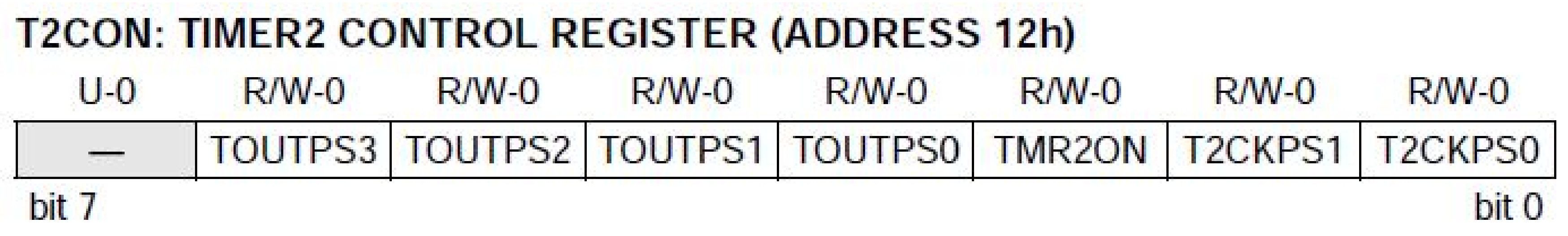


CCP1CON REGISTER/CCP2CON REGISTER (ADDRESS 17h/1Dh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

- **CCP1M3:CCP1M0: CCP1 Mode Select bits**
  - 11xx = PWM mode
- **CCP1X:CCP1Y: CCP1 Mode Select bits**
  - **PWM mode**:
    - **LSbs** of the PWM duty cycle are found in **CCPxX:CCPxY.**
    - The **eight MSbs** are found in **CCPR1L**.

T2CON Register


T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

- **TMR2ON: Timer2 On bit**
  - 1 = Timer2 is on
  - 0 = Timer2 is off
- **T2CKPS1:T2CKPS0: Timer2 Clock Prescale Select bits**
  - 00 = Prescaler is 1
  - 01 = Prescaler is 4
  - 1x = Prescaler is 16

PWM Calculations

- PWM Period =

$$[(PR2) + 1] \cdot 4 \cdot TOSC \cdot (TMR2\ Prescale\ Value)$$

- PWM Duty Cycle =

$$(CCPR1L:CCP1CON<5:4>) \cdot TOSC \cdot (TMR2\ Prescale\ Value)$$

- (CCPR1L:CCP1CON<5:4>)

$$= \frac{PWM\ Duty\ Cycle}{TOSC \cdot (TMR2\ Prescale\ Value)}$$

QUEST INNOVATIVE SOLUTIONS

Generate a PWM wave of 1000Hz Frequency, 50% Duty Cycle & 1:16 Prescaler

- **For 1 KHz frequency**
- **PR2** $= \dfrac{\text{PWM Period}}{4 \cdot \text{TOSC} \cdot (\text{TMR2 Prescale Value})}$

$$= \frac{1/(1\text{x}10\text{\^{}}3)}{4 \text{ x } 1/(4\text{x}10\text{\^{}}6) \text{ x } 16} = \mathbf{62.5} \;\; \mathbf{\sim=} \;\; \mathbf{0x3F}$$

- **(CCPR1L:CCP1CON<5:4>)**

$$= \frac{\text{PWM Duty Cycle}}{\text{TOSC} \cdot (\text{TMR2 Prescale Value})}$$

QUEST INNOVATIVE SOLUTIONS

Generate a PWM wave of 1000Hz Frequency, 50% Duty Cycle & 1:16 Prescaler

- **(CCPR1L:CCP1CON<5:4>)**

$$= \frac{\text{PWM Duty Cycle \%}}{\text{TOSC x (TMR2 Prescale Value)}}$$

$$= \frac{(1/1x10\text{^}3) \text{ x } (50/100)}{(1/4x10\text{^}6) \text{ x } 16}$$

= **125** = **0x7D**

CCPR1L = 0x1F
CCP1CON<5:4> = ‘0 1’

PWM Program

```
void main( )
{
    TRISC2=0;
    T2CON=0x06; //    Timer2 ON, 1:16 Prescaler
    TMR2=0x00;
    CCP1CON=0x1C; // Duty Cycle LSBs = ‘0 1’ & PWM Mode
    CCPR1L=0x1F;    // Duty Cycle MSBs
    PR2=0x3F;            // Time Period Value

    while(1)
    {
    }
}
```

MSSP MODULE

(Master Synchronous Serial Ports)

Master Synchronous Serial Port

- It is a serial interface, useful for communicating with other peripheral or microcontroller devices such as:
  - **Serial EEPROMs,**
  - **Shift registers,**
  - **Display drivers,**
  - **A/D converters,** etc.

- The MSSP module can operate in one of two modes

- **1. Serial Peripheral Interface (SPI)**
- **2. Inter-Integrated Circuit (I2C)**

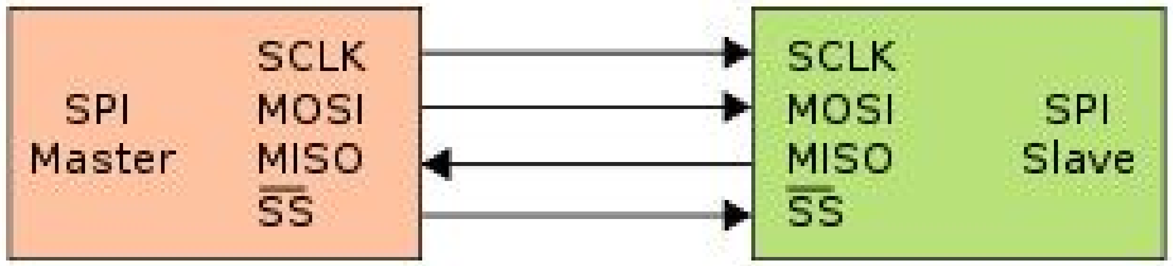
Control Registers

- Two special purpose registers control the MSSP operations
  - SSPSTAT : Status Register
  - SSPCON and SSPCON2 : Control Registers

Serial Peripheral Interface

SPI : Serial Peripheral Interface

- The **Serial Peripheral Interface bus (SPI)** is a synchronous serial communication interface specification used for short distance communication.
- The interface was developed by **Motorola** in the mid **1980s**
- SPI devices communicate in **full duplex** mode.
- The SPI bus can operate with **a single master device** and with **one or more slave devices**.



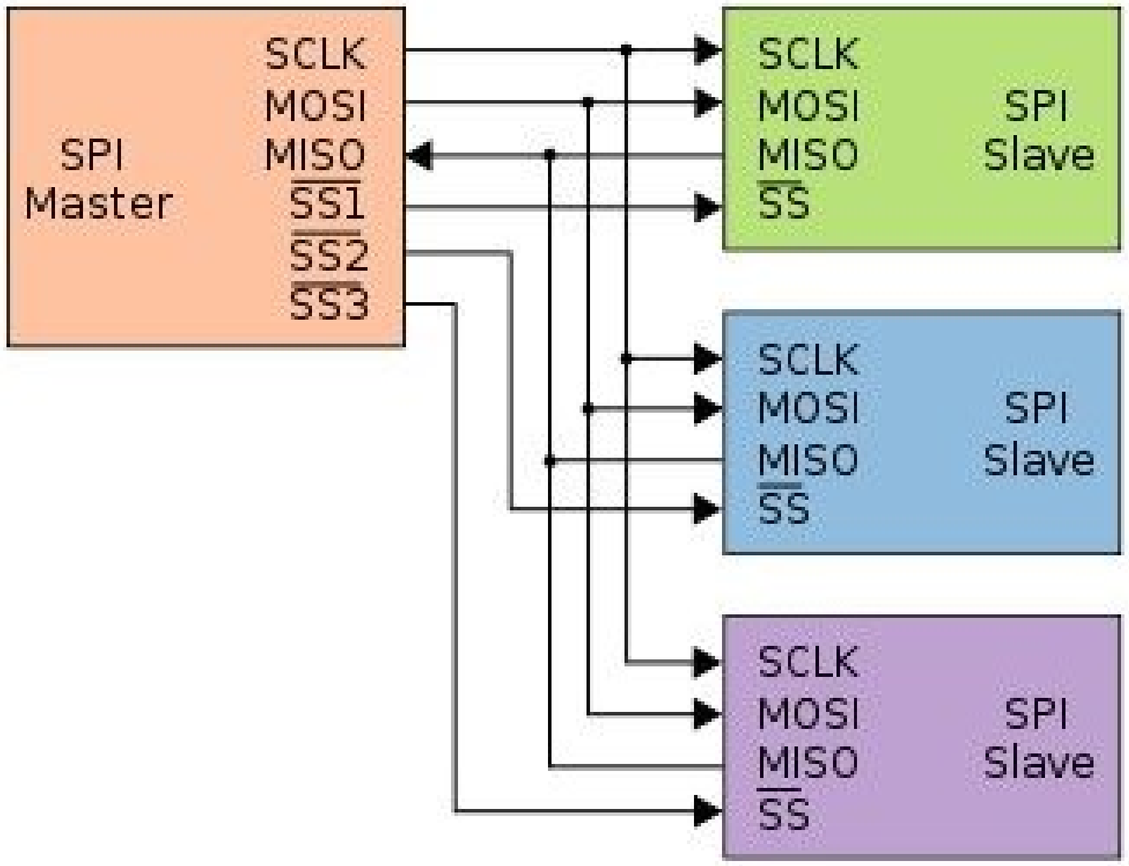
SPI : Serial Peripheral Interface

- **PINs Used:**

RC3/SCK/SCL:	Serial Clock
RC4/SDI :	Serial Data IN
RC5/SDO :	Serial Data Out
RA5/SS/AN4 :	Slave Select

Typical SPI bus:

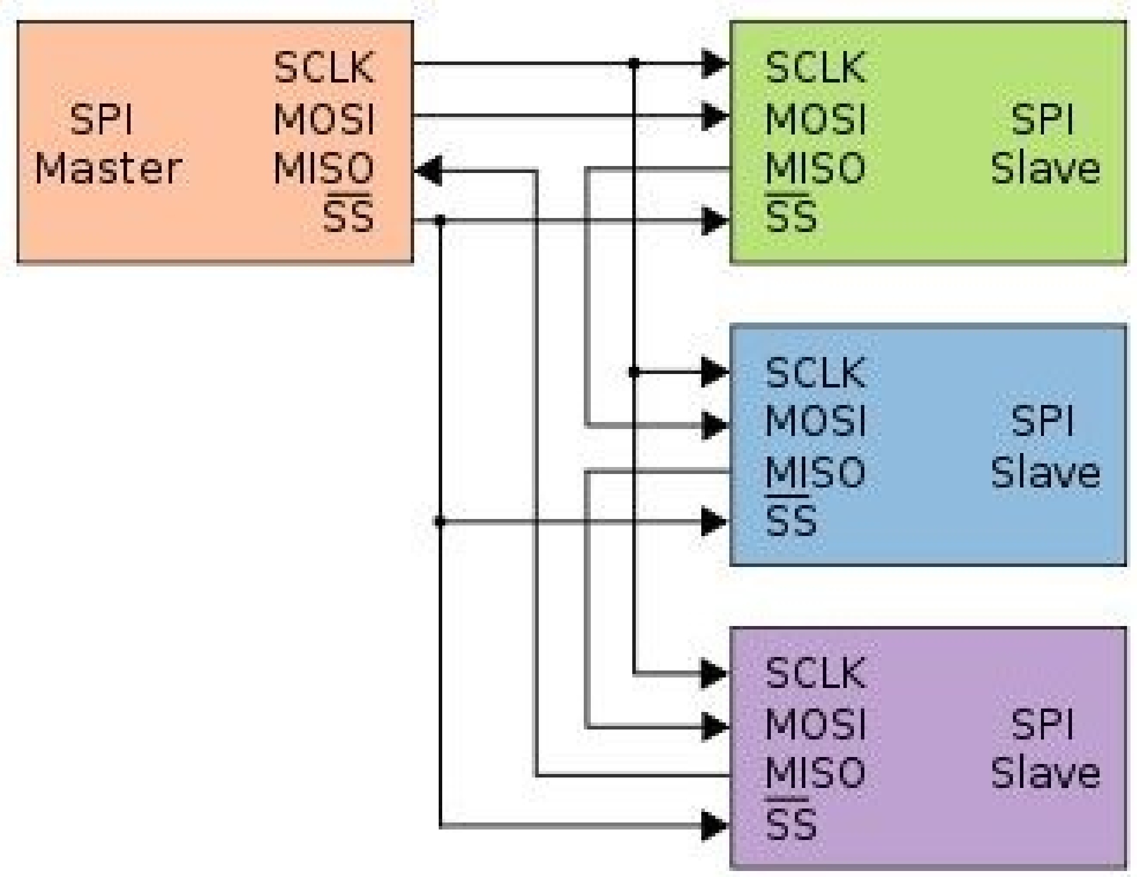
- **Single Master and Three Independent Slaves**



QUEST INNOVATIVE SOLUTIONS

Daisy-chained SPI bus:

- Single Master and Cooperative Slaves



Pin Configurations

- **Master Mode:**
  - Must Clear SCK pin (TRISC3=0)
  - Must Set SDI pin (TRISC4=1)
  - Must Clear SDO pin (TRISC5=0)
- **Slave Mode:**
  - Must Set SCK pin (TRISC3=1)
  - Must Set SDI pin (TRISC4=1)
  - Must Clear SDO pin (TRISC5=0)
  - Must Set SS pin (TRISA5=1)
- **Slave Receives data only when it gets a high to low pulse.**

Register Configurations

- **Master Mode:**
  - **SSPCON = 0x30**
  - **SSPSTAT = 0x40**
  - **SSPIF** is the flag
    - **1** = When transmission Completes
    - **0** = When transmission in progress

- **Slave Mode:**
  - **SSPCON = 0x34**
    (0x35 = Slave pin disabled, 0x34 = Slave pin Enabled)
  - **SSPSTAT = 0x40**
  - **BF** (Receive Mode Only)
    - **1** = Received Completely
    - **0** = Reception in progress

Inter - Integrated Circuit

I2C: Inter Integrated Circuit

- I2C was developed by **Philips in 1982**.
- I^{2}C protocol allow **multiple "slave"** digital integrated circuits ("chips") to communicate with **one or more "master"** chips.
- Like SPI, it is only for **short distance communications**.
- It requires **two signal wires** for information exchange.
- I2C support up to **1008 slave devices**.
- Unlike SPI, I2C can **support** a **multi-master system**.

I2C: Inter Integrated Circuit

- **Bidirectional** Data transfer.
- **Half Duplex** (have **only one Data line**).
- **Synchronous Bus** so **Data** is **clocked** with **Clock Signal**.
- **Clock** is **controlled** when **Data line** is **changed**.

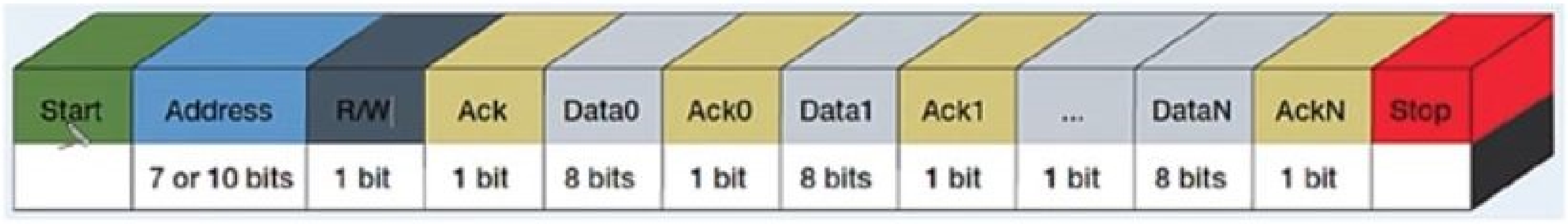
I2C: Inter Integrated Circuit

- **I2C bus consists of:**
  - **Serial clock (SCL): RC3/SCK/SCL**
  - **Serial data (SDA): RC4/SDI/SDA**
- Must configure these pins as inputs or outputs through the **TRISC<4:3>** bits.
- **I2C support up to:**

7 bit Addressing	-	**112** Slave devices (128 Idle)
10 bit Addressing	-	**1008** slave devices (1024 Idle)

QUEST INNOVATIVE SOLUTIONS

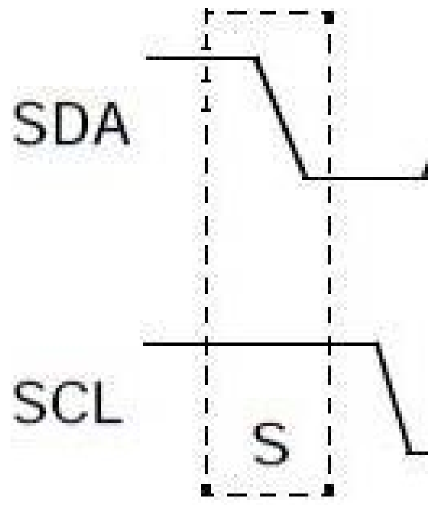
I2C Packet Format



Start	Address	R/W	Ack	Data0	Ack0	Data1	Ack1	...	DataN	AckN	Stop
	7 or 10 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit	8 bits	1 bit	

I2C Start Condition

- SCL is High when SDA goes Low.
- Enable SEN bit.
- Wait for SSPIF to Set.
- Then Clear SSPIF.



7 - bit Addressing

- **In Slave Mode:**

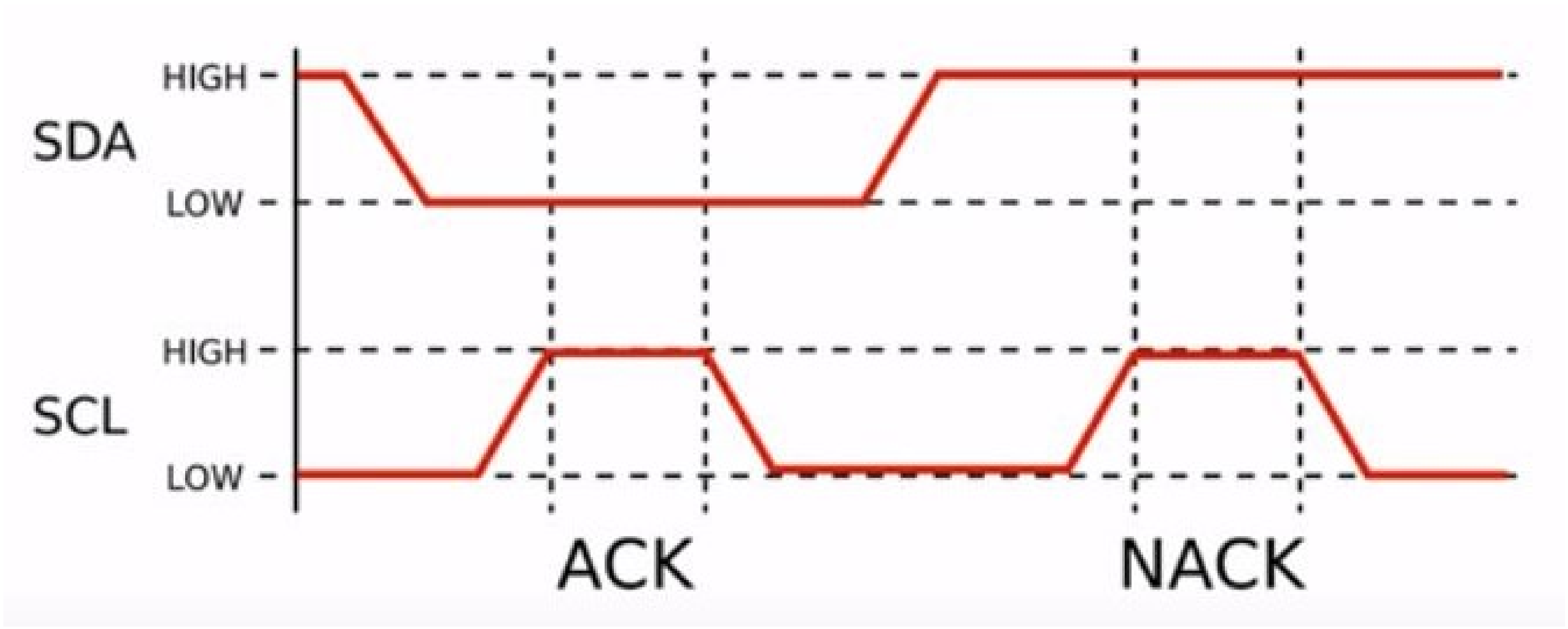
- **SSPADD** register stores the **Address of Slave Device.**
- **Start condition** occurs.
- Then **8 bits** are shifted into the **SSPSR** register.
- **SSPSR<7:1>** is compared to the value of the SSPADD register.

7 - bit Addressing Cont..

- If the addresses match, and the BF and SSPOV bits are clear, the following events occur:

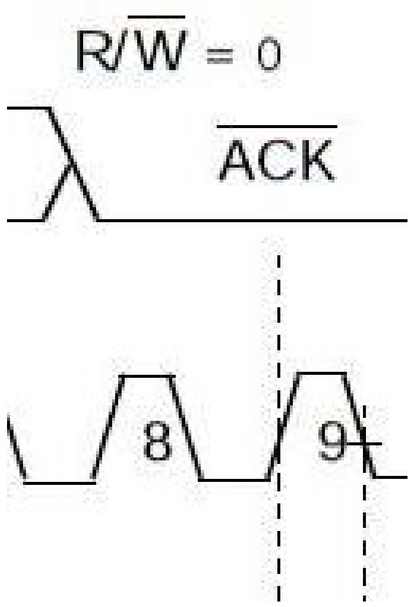
- The **SSPSR** register value is **loaded** into the **SSPBUF** register.
- **BF is set**, when SSPBUF is Full.
- An **ACK** pulse is **generated**.
- **SSPIF** (PIR1<3>), is **set** (interrupt is generated if enabled).

I2C ACK & NACK



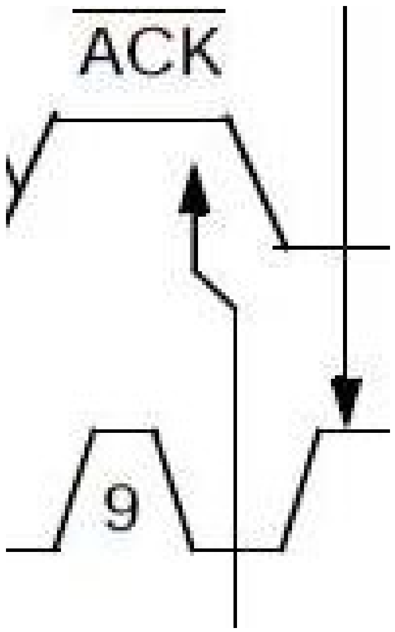
I2C ACK

- SCL toggles when ACK occurs on SDA Line.
- Clear ACKDT bit.
- Set ACKEN.
- Wait for SSPIF to Set.
- Then Clear SSPIF.



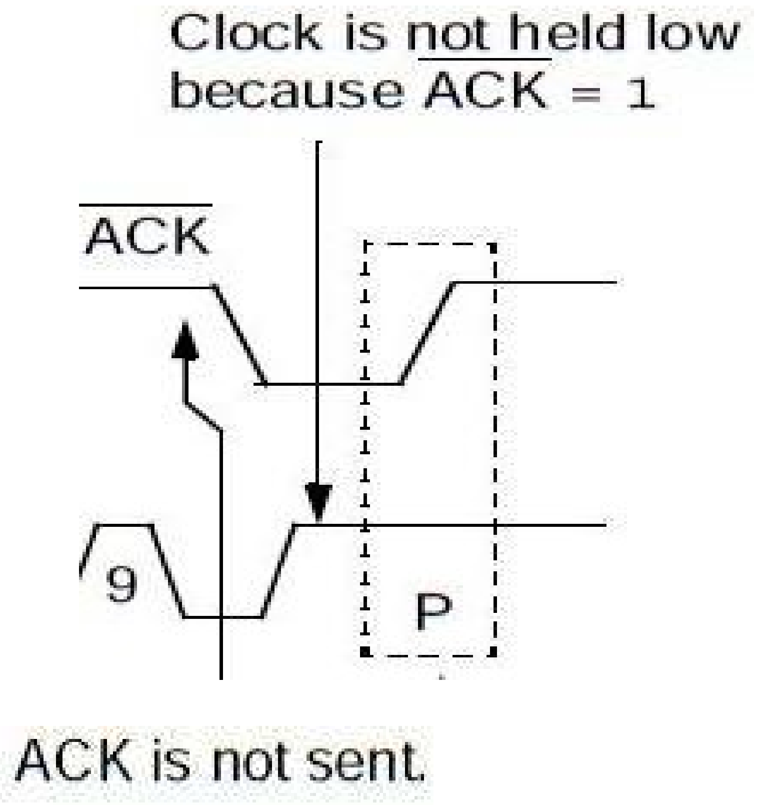
I2C NO_ACK

- SCL goes High when No ACK occurs on SDA Line.
- Set ACKDT bit.
- Set ACKEN.
- Wait for SSPIF to Set.
- Then Clear SSPIF.

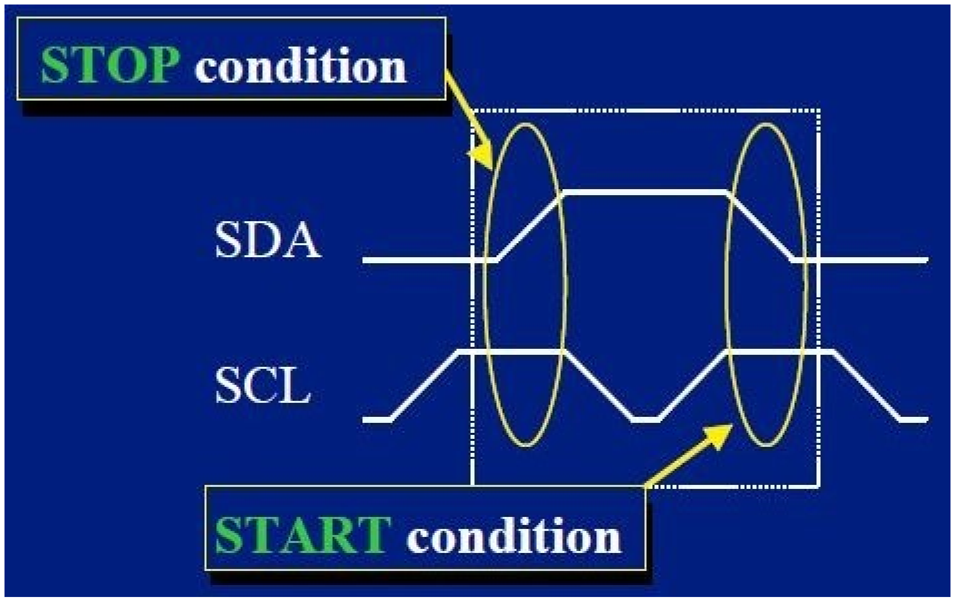


I2C Stop Condition

- SDA goes High SCL is High.
- Enable PEN bit.
- Wait for SSPIF to Set.
- Then Clear SSPIF.



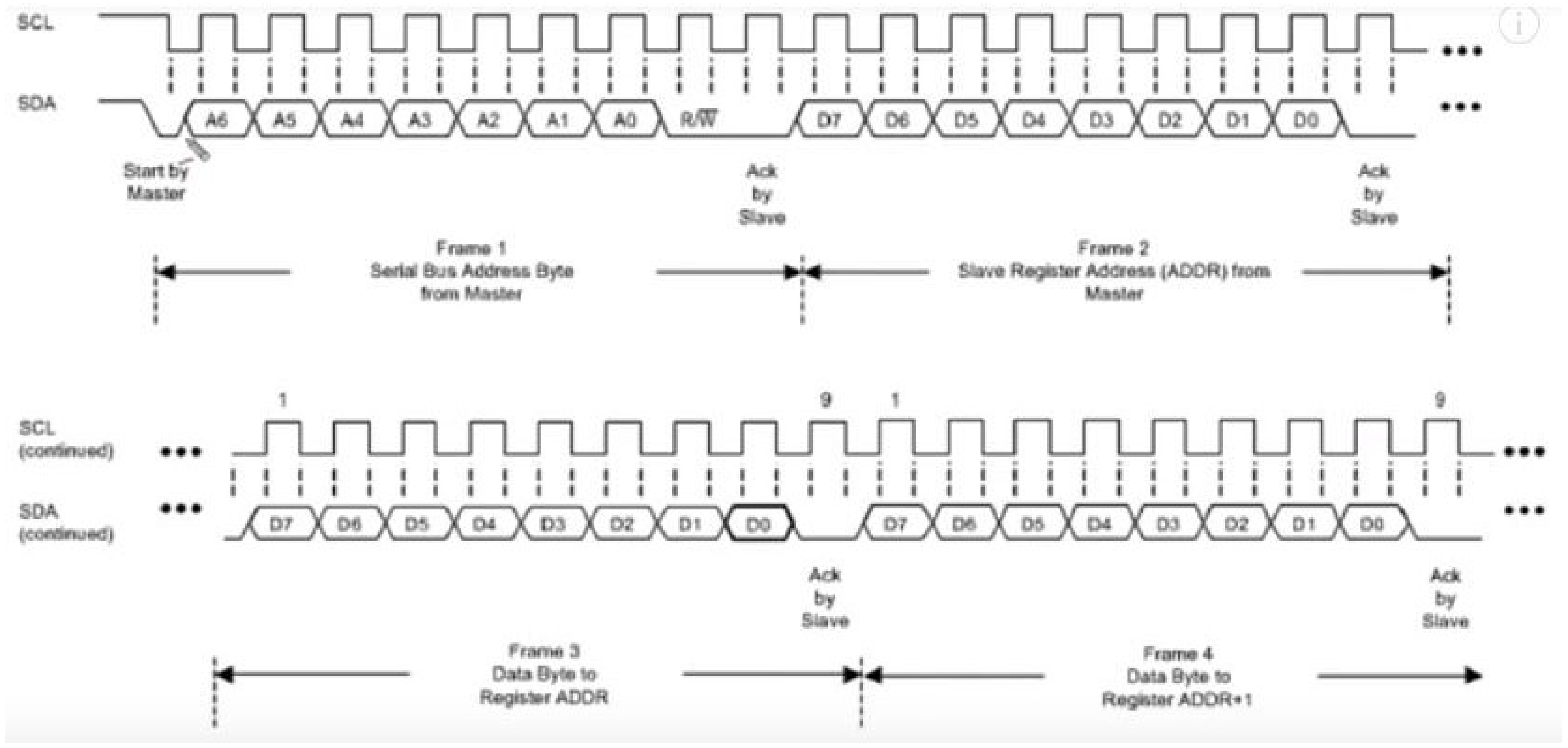
I2C Repeated Start



I2C Repeated Start

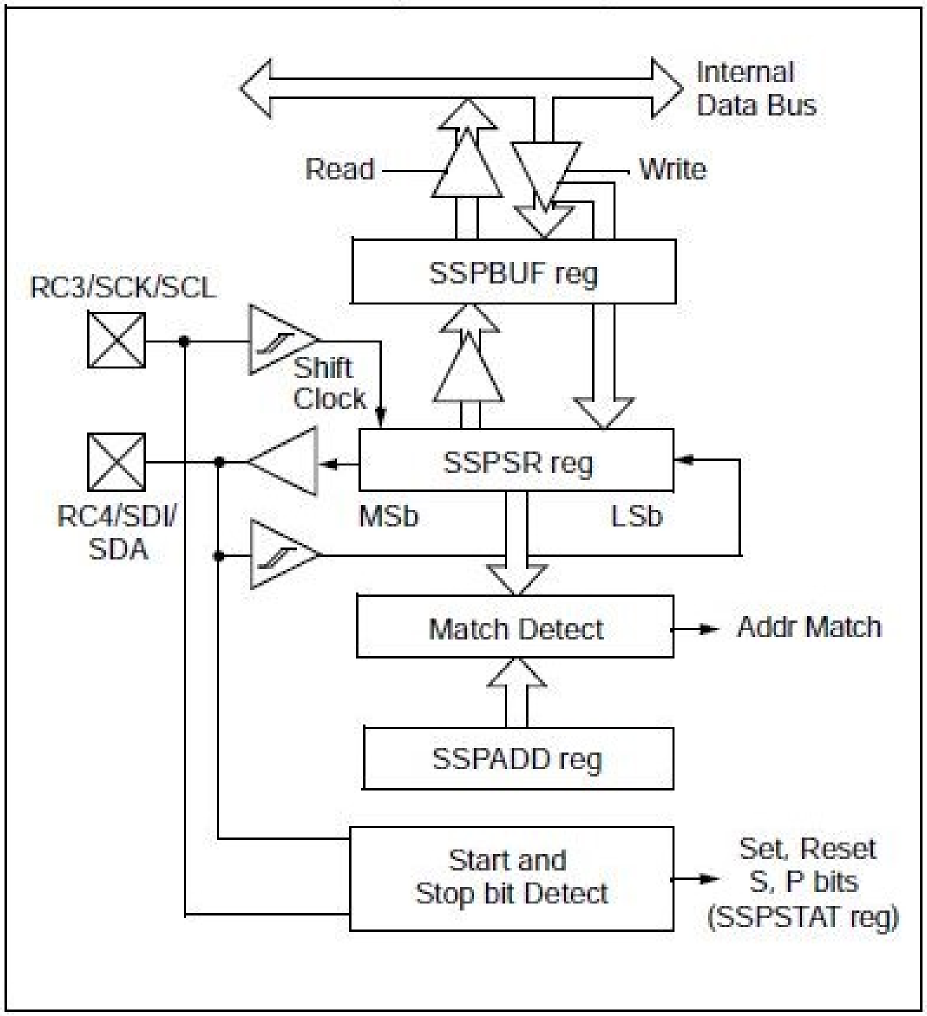
- **Repeated Start condition** indicates that a **device would like to transmit more data**, but **does not wish** to **release the line**.
- If Master is communicating with a Slave, and not want to be interrupted when transmitting addresses and gathering data. A restart condition will handle this.
- **Repeated Start Condition** is a **Stop Condition** quickly **followed** by a **Start Condition**.
- **Any number of Repeated starts is allowed**, and the master will maintain control of the bus until it issues a stop condition.

I2C Packet Transfer


SCL
SDA
A6
A5
A4
A3
A2
A1
A0
R/W̄
D7
D6
D5
D4
D3
D2
D1
D0
Start by Master
Ack by Slave
Ack by Slave
Frame 1 Serial Bus Address Byte from Master
Frame 2 Slave Register Address (ADDR) from Master
SCL (continued)
SDA (continued)
1
9
1
9
Ack by Slave
Ack by Slave
Frame 3 Data Byte to Register ADDR
Frame 4 Data Byte to Register ADDR+1

QUEST INNOVATIVE SOLUTIONS

I2C Block Diagram



QUEST INNOVATIVE SOLUTIONS

I2C Registers

SSPCON:	Control Register
SSPCON2:	Control Register 2
SSPSTAT:	Status Register
SSPBUF:	Serial Receive/Transmit Buffer Register
SSPSR:	Shift Register – Not directly accessible
SSPADD:	**Address Register** (Slave Mode) **Lower seven bits** act as the **baud rate generator** (Master Mode)

Double-buffered Receiver

- **During Receive Operation:**
  - **SSPSR** and **SSPBUF** together create a **double-buffered receiver**.
  - When **SSPSR** receives a **complete byte**, it is **transferred** to **SSPBUF** and the **SSPIF interrupt** is **set**.
- **During Transmission:**
  - **SSPBUF** is **not double buffered**.
  - A write to **SSPBUF** will write to both **SSPBUF** and **SSPSR**.

SSPCON 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
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

- **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.
  - **TRISC3 = 1;**
  - **TRISC4 = 1;**

QUEST INNOVATIVE SOLUTIONS

SSPCON Register

- **SSPM3:SSPM0:**
  Synchronous Serial Port Mode Select bits
  - **'1 0 0 0' =** I2C Master mode,
    clock = FOSC/(4 * (SSPADD + 1))

SSPSTAT Register

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

- **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).

SSPSTAT Register

- **D/A: Data/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**

- **R/W: Read/Write bit information (I2C mode only)**

In Slave mode:	**In Master mode:**
1 = Read	**1** = Transmit is in progress
0 = Write	**0** = Transmit is not in progress

SSPSTAT Register

- **BF: Buffer Full Status bit**

- **In Transmit mode:**
  - **1** = Data Transmit in progress (does not include the ACK and Stop bits), SSPBUF is full
  - **0** = Data Transmit complete (does not include the ACK and Stop bits), SSPBUF is empty

- **In Receive mode:**
  - **1** = Receive complete, SSPBUF is full
  - **0** = Receive not complete, SSPBUF is empty

SSPCON2 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
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

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

- **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.

SSPCON2 Register

- **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

SSPCON2 Register

- **RCEN: Receive Enable bit (Master mode only)**
  - **1** = Enables Receive mode for I2C
  - **0** = Receive Idle
- **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
- **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

SSPCON2 Register

- **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)

I2C Master Configuration

- Select I2C Master Mode (SSPCON, SSPSTAT, SSPCON2)
- Start I2C by Master.
- Transmit the First byte (Address of Slave).
    - Master waits for ACK from Slave for each write to Slave.
- Next byte is based on the last byte(R/W: 0=Write, 1= Read).
- If R/W =0
    - Next byte can be Internal Register Address and followed by data bytes.
    - Finally, Stop I2C by Master.

I2C Master Configuration Cont..

- If R/W =1
  - Now Master starts to receive from Slave.
  - For each receive from Slave, Master gives ACK to Slave.
  - When Last data byte is received, Master sends NACK to Slave.
  - Then Stop I2C by Master

I2C Slave Configuration

- Select I2C Slave Mode (SSPCON, SSPSTAT, SSPCON2)
- If SSPIF is Set, receives first byte to SSPBUF, then SSPBUF compares with SSPADD.
- If matches, gives ACK to Master.
  - Slave sends ACK to Master for each read from Master.
- Based on the last byte(R/W: 0=Write, 1= Read).
- If R/W =0
  - Next byte will be stored in Internal Registers of Slave.
- If R/W =1
  - Slave sends data bytes stored in the Internal Registers.
  - When Slave receives a NACK then it Stop sending data.

I2C Advantages

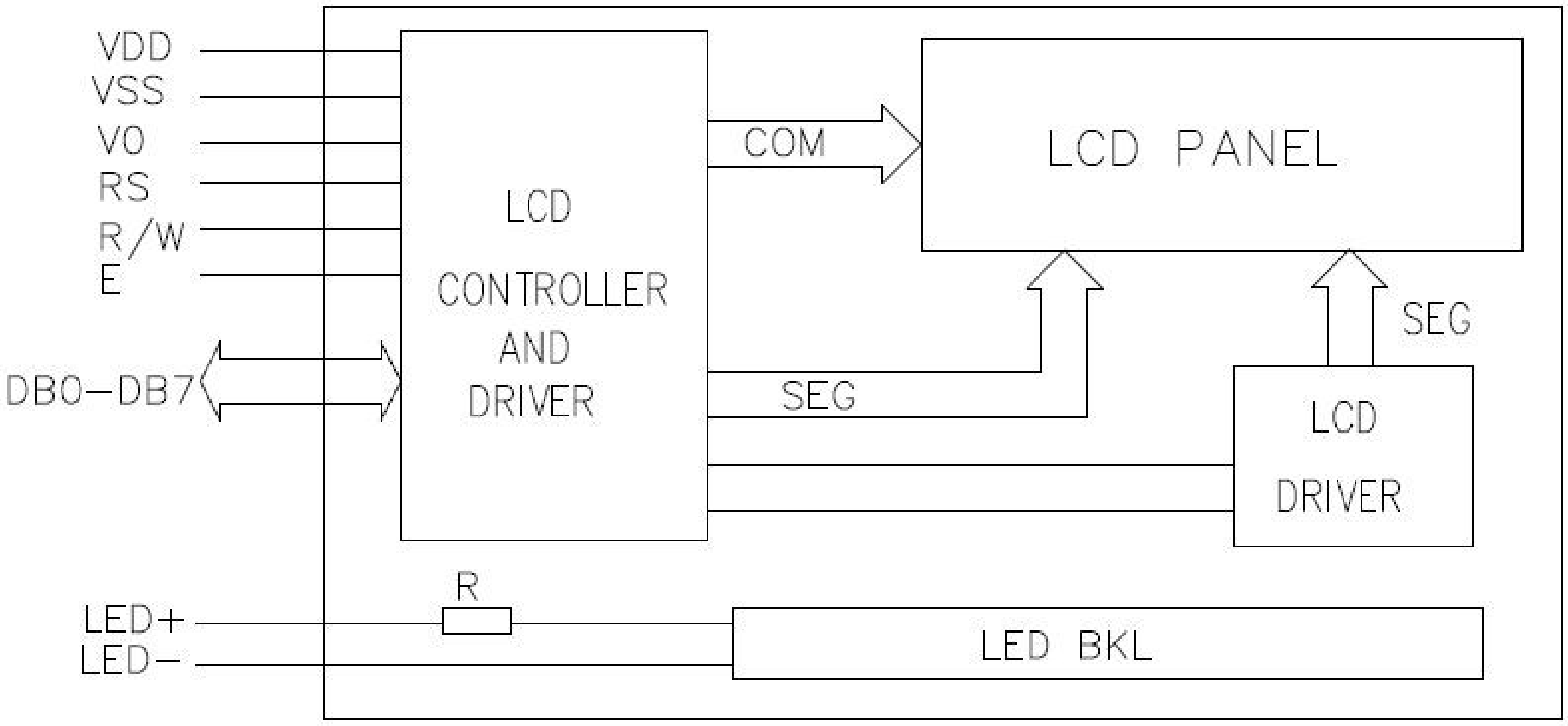
- Good for communication in On-System Devices.
- Easy link multiple devices because of addressing scheme.
- Cost and Complexity do not scale-up with the number of Devices.

LCD INTERFACING

LCD Features

- 5x8 dots with Cursor
- 16 Characters *2 Line Display
- 4-bit or 8-bit MPU Interfaces
- Built-in Controller (ST7066)
- Display Mode & Backlight Variations

LCD Block Diagram



LCD Pin Description

Pin no.	Symbol	External connection	Function
1	Vss	Power supply	Signal ground for LCM
2	V_{DD}		Power supply for logic for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU
15	LED+	LED BKL power supply	Power supply for BKL
16	LED-		Power supply for BKL

QUEST INNOVATIVE SOLUTIONS

LCD Control Instructions

- **Function set :** (0x33)
- **Function set :** (0x32)
- **Function set :** 8bit mode (0x38)
- **Function set :** 4bit mode (0x28)
- **Display ON/OFF Control:** (0x0C)
- **Clear Display:** (0x01)
- **Return Home:** (0x02)
- **Cursor or Display shift:** Cursor-right shift & entire display (0x03)
- **Cursor position 1st Row 1st Column :** (0x80)
- **Cursor position 2nd Row 1st Column :** (0xC0)
- **Set DDRAM Address:**

LCD Code

```
void LCD_command(char a)
{
   TRISC=0X00;
   char b;
   RS=0;
   RW=0;

   b=a & 0xF0;
   PORTC &= 0x0F;
   PORTC |=b;
   EN=1;
   __delay_ms(100);
   EN=0;

   b=a & 0x0F;
   PORTC &= 0x0F;
   PORTC |=(b<<4);
   EN=1;
   __delay_ms(100);
   EN=0;
  __delay_ms(1);
}
```

```
void LCD_char(char a)
{
   TRISC=0X00;
   char b;
   RS=1;
   RW=0;

   b=a & 0xF0;
   PORTC &= 0x0F;
   PORTC |=b;
   EN=1;
   __delay_us(10);
   EN=0;

   b=a & 0x0F;
   PORTC &= 0x0F;
   PORTC |=(b<<4);
   EN=1;
   __delay_us(10);
   EN=0;
}
```

```
void LCD_string(char *p)
{
   int count=0;
   while(*p)
   {
      count++;
      LCD_char(*p);
      p++;
   }
}
```

KEYPAD INTERFACING

CAN PROTOCOL

CAN

- **Controller Area Network (CAN)**
- CAN is a two wired half duplex high speed asynchronous serial communication protocol.
- It is basically used for communication in automobiles
- A CAN protocol is a CSMA-CD/ASM protocol or carrier sense multiple access collision detection arbitration on message priority protocol.
- CSMA ensures each node must wait for a given period before sending any message. Collision detection ensures that the collision is avoided by selecting the messages based on their prescribed priority.

History

- Development of the CAN bus started in **1983** at **Robert Bosch**.
- The protocol was officially released in **1986** at the **Society of Automotive Engineers (SAE) conference** in Detroit, Michigan, **USA**.
- The **first CAN controller chips**, produced by **Intel** and **Philips**, came on the market in **1987**.
- Released in **1991** the **Mercedes-Benz W140** was the first production vehicle to feature a CAN-based multiplex wiring system.

- Bosch published several versions of the CAN specification
- The latest is **CAN 2.0** published in **1991**.
- This specification has two parts:
  - **Part A** is for the standard format with an **11-bit identifier.**
  - **Part B** is for the extended format with a **29-bit identifier.**
- A CAN device that uses **11-bit** identifiers is commonly called **CAN 2.0A** and a CAN device that uses **29-bit** identifiers is commonly called **CAN 2.0B**.
- **CAN FD -** Flexible Data Rate

- In 1993, the International Organization for Standardization (ISO) released the CAN standard **ISO 11898**.
- CAN operates at:
  - 125 kbps - Low speed (500m)
  - 1 Mbps - High speed (40m).
  - 15 Mbps - Flexible data rate (10m)

Applications

- Automotive Industry
- Industrial Machinary
- Military
- Medical
- Marine
- Elevators

Properties of CAN

- **Prioritization of Messages:**
  - CAN is not an address/node oriented protocol.
  - CAN is a message based protocol.
  - High priority message will be transmitted first.
  - Priority will be decided based on identifier.
  - Lower the value of Identifier, priority will be higher and vice versa.
  - Eg., ECU1, Identifier = 0x100 // Highest priority
    ECU2, Identifier = 0x1A0 // 2nd Highest priority
    ECU3, Identifier = 0x1FF // 3rd Highest priority

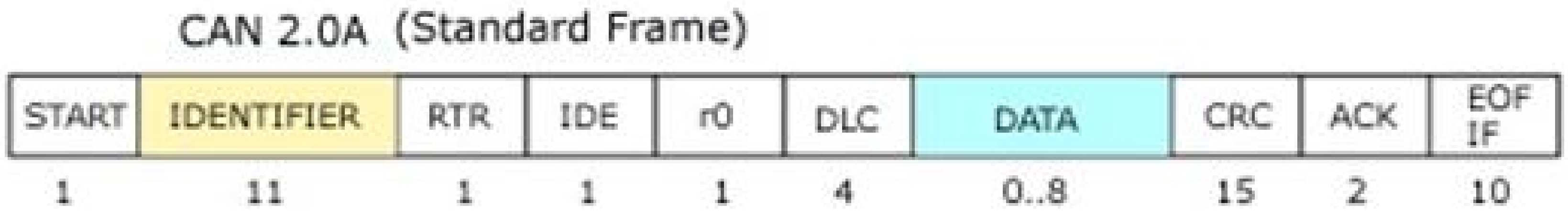
Properties of CAN

- **Guarantee of Latency time** (time taken to respond by a system after it receives an instruction).
- **Configuration Flexibility:** Connecting and Disconnecting the node from network will not effect the functionality of the network.
- **Multicast Reception with Time Synchronization:** Message can be broadcast to all nodes, based on system design.
- **Multimaster:** The node with high priority wins arbitration and will be the master and gets the bus access for its message transmission.

Properties of CAN

- Automatic transmission of corrupted messages as soon as the bus is idle again.
- Error detection and signaling.

CAN 2.0A Packet Format



CAN 2.0A (Standard Frame)

START	IDENTIFIER	RTR	IDE	r0	DLC	DATA	CRC	ACK	EOF IF
1	11	1	1	1	4	0..8	15	2	10

Thankyou

Quest Innovative Solutions Pvt. Ltd

MKS Towers, Above HDFC Bank
Kadavanthara, Cochin 682020
Email: mail@qis.co.in, training@qis.co.in
Website: www.qis.co.in

QUEST
INNOVATIVE SOLUTIONS

Architectural Features

- The high performance of the PIC micro™ devices can be attributed to a number of architectural features .These include:

- Harvard architecture
- Long Word Instructions
- Single Word Instructions
- Single Cycle Instructions
- Instruction Pipelining
- Reduced Instruction Set
- Register File Architecture
- Orthogonal (Symmetric) Instructions