Interrupts And I/O Ports :-

Interrupt Logic :-

In the timer2 circuitry of the PIC microcontroller, when the post scaler rolls over, it sets a flag called TMR2IF in the PIR1 register. If the TMR2IE bit in the PIE1 register and the PEIE bit in the INTCON register have been both initialized to one, then an interrupt signal with reach the main scaller.

GIE is automatically cleared when an interrupt occurs, suspending further interrupts for the duration of interrupt service routine execution. GIE is automatically set when the instruction 'retfile' is executed at the end of an interrupt service routine, reanabling interrupts at the same time that program control is returned to the mainline program. ~~GIE is set~~

Within the interrupt service routine, the TMR2IF flag that caused the interrupt must be cleared with the instruction ' bcf PIR1, TMR2IF '
otherwise the CPU will reenable interrupts.

PIC chip Reset :- When PIC reset, GIE, PEIE and TMR2IE among others are ~~set~~ cleared. They must be set in the initial subroutine to turn on the interrupts that will be used to control the loop time.

Timer 2 Scaler initialization

The 'divide by' scale of Timer 2 is controlled by T2CON and PR2. T2CON sets up the prescaler and postscaller as shown in the figure below. The number represented by the 4 bits that determine the postscaller value must be one less than the desired divider value. PR2 must be one less than the desired divider value for the main scaller.

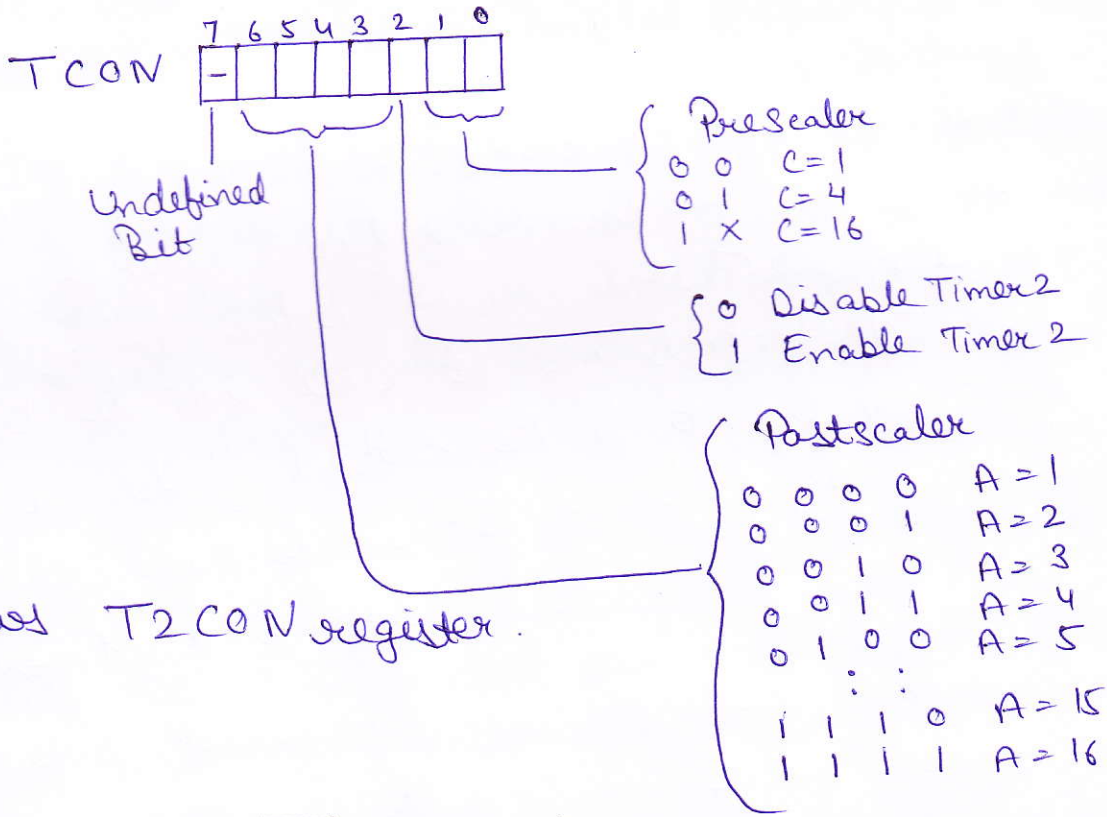


fig shows T2CON register.

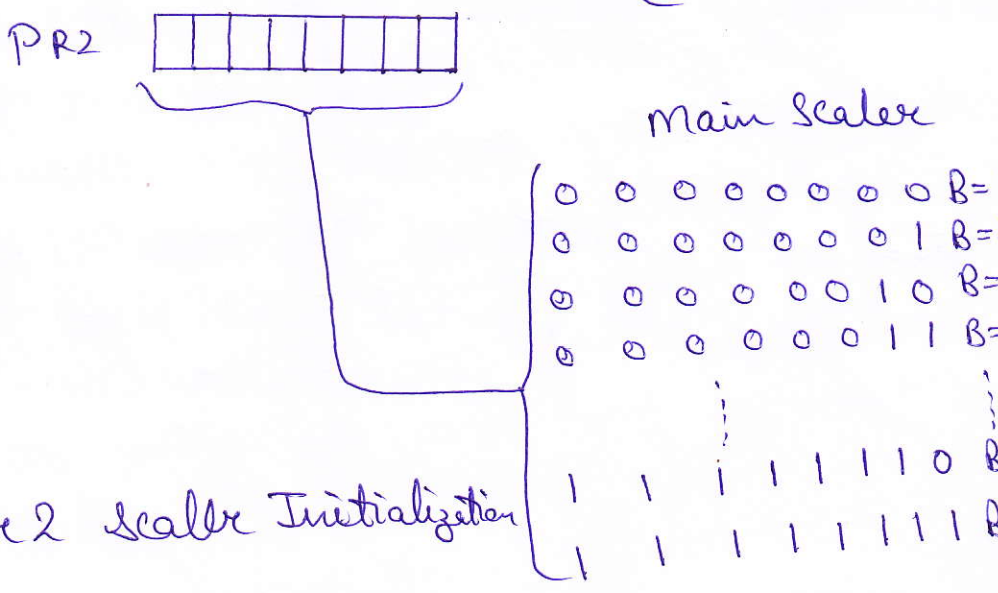


fig shows Timer 2 scaler Initialization

Timer 2 scaller initialization for oscillator = 4, 10, 20 MH_2 to obtain interrupts every two milliseconds are given is a, b & c statements respectively with 'd' being the Cmeneralization of T2 CON initialization :-

a) - T2 CON ← B'00001101' PR2 ← D'249'

b) - T2 CON ← B'00100101' PR2 ← D'249'

c) - T2 CON ← B'01001101' PR2 ← D'249'

d) - T2 CON ← (4 x freq) - 3

where freq represents the crystal frequency in MH_2.

The initialization required of T2 CON and PR2 for each of the three frequencies given in a, b, c statements above. In each case PR2 is initialized to 249. This initialization can be generalized to

(4x freq)-3

Assembler can handle arithmatic operations in the operand field of instruction — as long as the values being operated on are constants known at the time of assembly.

The last two lines of the Initial subroutine be

```
bsf   INTCON, GIE
return.
```

This ensures that all conditions required by an interrupt will have set-up before the first interrupt actually occurs.

IntService Interrupt Service Routine

Whenever an interrupt occurs, the CPU automatically pushes the return address in the program counter onto the stack and clears the GIE bit, disabling further interrupts. So, the IntService set asides the content of W and of STATUS. Then they can be restored at the end of interrupt service routine to exactly the same state they were in when the interrupt occured as required for the proper execution of mainline code. The three instructions for setting aside W and STATUS are

```
swapf STATUS, W
```

to move the content of STATUS to W

```
movf STATUS, W
```

The swapf instruction does not affect Z bit in the STATUS register when it makes this move. In Contrast, the movf instruction may corrupt the Z bit so it must not be used here.

The central code of IntService is a sequence of btfsc, call instruction pairs. Each pair tests the flag of an enabled interrupt source. If the flag is set, the source's interrupt service routine is called that provides the desired response and clears the flag. If a tested flag is not set, the call is skipped. This sequence is called a polling routine and it quickly gets the CPU to the service routine for the source that requested service.

LoopTime Subroutine:

With the help of Timer 2 subroutine in IntService the LoopTime subroutine, which is called from within the mainline loop, is able to make the time around the loop take exactly 10ms. For the LoopTime subroutine to work correctly the worst case execution time for all the interrupt service routines that could request service within a 10ms interval must be less that 10ms. This 'mainline overrun' condition is easily avoided for many, if not most, applications. If it is not avoided during one loop, the next looptime will be shortened to compensate. As a consequence, successive executions of some tasks may occur less than 10ms apart. On the other hand if even this mainline overrun condition does occur less than 10ms, the long range timing provided by the LoopTime subroutine will still be accurate as long as no counts of SCALER are ever lost.

Once the CPU enters the LoopTime subroutine it waits on the Timer2 subroutine in IntService, which in turn waits on successive interrupts from Timer2. When the Timer2 interrupt occurs that finally decrements SCALER down from H '00' to H 'FF' the goto LoopTime instruction will be skipped, five will be added to SCALE and the CPU will return from interrupt service routine.

Synchronous Serial Port Module :-

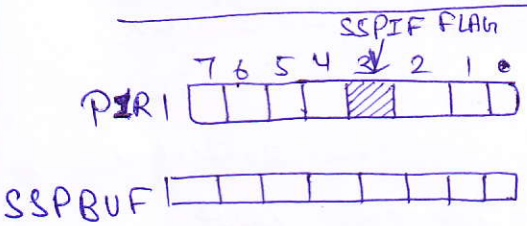
Serial Synchronous Port (SSP) module can be configured into either of the two modes :-

* Serial Peripheral Interface (SPI)
* Inter-Integrated Circuit (I^2C)

Either of these modes can be used to interconnect two or more PIC chips to each other using a minimal number of wires for the interconnections. Alternatively, either can be used to connect a PIC chip to a peripheral chip.

In the case of the I^2C mode, the peripheral chip must also include an I^2C interface. In contrast, the SPI mode provides the clock and serial data lines for direct connection to shift registers, adding an arbitary number of I/O pins to a PIC chip

Serial Peripheral Interface :-



RC5/SDO → output of SSPBUF register appers MSB first

RC4/SDI → Input goes into SSPBUF register appears MSB first

RC3/SCK → Eight clock pulses are emitted in response to a write to SSPBUF register

The figure shows three pins associated with the serial peripheral Interface when it is to be connected to shift register for I/O expansion.

PORTC I/O pins if neither of the two SSP mode is selected. SPI port requires the RC3/SCK pin to be an output that generates the clock signal used by the external shift register. This output clock line characterizes the SPI's master mode.

When a byte of data is written to the SSPBUF register it is shifted out of the SDO pin in synchronism with the emitted pulses on the SCK pin. The MSB of SSPBUF is the first bit to appear on the SDO pin. For output port expansion, this serial output must be clocked into a shift register.

The same write to the SSPBUF register that initiated the output data stream to appear on the SDO pin also initiates the 8-bit reception into SSPBUF of whatever appears on SDI pin at the time of the eight rising edges of the SCK pin. To create an input port, the PIC chip need only parallel load eight inputs into a shift register and then use SCK to shift them in to SDI one by one.

Recall that a write to SSPBUF iniates both the transmission of a byte out of the SDO p and the reception of a byte in from the SDI pin. If the SSPIF flag in the PIR1 register is clear before the SPI transmission is initiated, then it will be automatically set at the completio of the transfer. This is the signal that the transferred data is in place and ready to b used.

Output port Expansion:-

It uses two double-buffered octal shift registers to implement two external 8-bit output ports. In support of this additional circuitry, the initial subroutine needs added instructions to initialize TRISC, TRISD and SSPCON. Since the SPI's serial data input line is not being used in this application, the pin can be used as general purpose I/O pin simply by configuring it as an output. Since the external port cannot be read, a copy of its contents can be kept in a RAM variable. To change 1 bit of the external port, change the corresponding bit of the RAM variable and then copy of the variable to external port.

Input port Expansion:-

* Using SPI for expanding input ports is like output port expansion.

1)- Parallel-in serial out
2)- First loaded RD7 low to high
3)- First shift before first read.
4)- Feedback
5)- SDI not working
6)- RC5/SDO output not used.

UART

* Universal asynchronous receiver transmitter.

Waveforms and Baud-Rate Accuracy:-

When serial data is transmitted asynchronously, the data stream is generated with the transmitter's clock.

Baud-Rate Selection:-

If the crystal clock rate were selected to be a carefully chosen multiple of the desired baud rate, then the baud-rate generator would produce the desired baud rate exactly. The clock rates used are: 4MHz 10MHz 20MHz.

UART Initialization:-

* Data direction bits must be setup as inputs.
* This disables the general I/O port output circuitry associated with these two pins.
* Initialized by writes to SPBRG, TXSTA and RCSTA.
* The flag and interrupt enable bits of PIR1, PIE1 and INTCON register control the timing of the CPU interaction with the UART.

UART Use:-

* Provides two-wire serial interface to personal computer
* Couple two PIC's together using the maximum possible baud rate to obtain fast coupling between the two PICs.
* Use of UART interface to expand master PIC's resources.