Interrupt and Timer ISRs

Student's name & ID (1): ___________________________________________________

Partner's name & ID (2): ___________________________________________________

Your Section number & TA's name _________________________________________

Notes:

You must work on this assignment with your partner.

Hand in a printer copy of your software listings for the team.
Hand in a neat copy of your circuit schematics for the team.

These will be returned to you so that they may be used for reference.

----------------------------- do not write below this line -----------------------------

<table>
<thead>
<tr>
<th>POINTS (1)</th>
<th>TA init. (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grade for performance verification (50% max.)</td>
<td></td>
</tr>
<tr>
<td>Enhancement (5% max.)</td>
<td></td>
</tr>
<tr>
<td>Grade for answers to TA's questions (20% max.)</td>
<td></td>
</tr>
<tr>
<td>Grade for documentation and appearance (30% max.)</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
</tr>
</tbody>
</table>

Grader's signature: _________________________________________________________

Date: _____________________________________________________________________
Interrupt and Timer ISRs

GOAL

By doing this lab assignment, you will learn:
1. To use IRQ and on-chip Timer interrupts on the 8051.
2. To add interrupt service routines to a C program using the SDCC compiler.

PREPARATION

• References: C8051F12x-13x Reference Manual, Ch. 13, 14, 23, 24

Software Design
• Do a top-down design of your program. Provide a flowchart or equivalent.
• Write C programs for each module/subroutine. Recall that interrupt service routines are not subroutines. Provide planning documentation for both flowchart and tests. Specify test inputs and expected output or other indicators of correct operation.
• After all the submodules are operating correctly, they may be integrated to form a higher level module. Integrate your modules to and compile a program that is free from syntax errors.

PROGRAMMING TASKS FOR THE 8051

1. Write a simple C program that responds to a External Interrupt 0 (/INT0 wired through a pushbutton switch on the protoboard to ground). The interrupt handler should set a global variable that can be seen by the main routine. The main routine should handle any I/O, i.e. output to the terminal, related to the generated interrupt.
2. Write two simple C programs that respond to timer interrupts to display elapsed time in tenths of a second using two different operational modes of the programmable 8051 timers (different timers, different scaled clocks, up or down mode, etc.). One of the modes may approximate 0.1 sec by rounding but the other must be exact to the accuracy of the processor’s clock frequency. Responses through I/O to the console may be off by milliseconds due to serial line delays, but the internal interrupts should be generated every 0.1 seconds. Be prepared to explain the differences in operation between the two programs.
3. Combine the C programs from 1 and the exact 0.1 sec timer in 2 to test reaction time. Flash the screen after a random delay and measure the time it takes for the subject to react and ground the IRQ pin. (Use the pull-ups on inputs to avoid false interrupts.) Compute the average of the number of trials or a moving average of the last 5 trials and display it on the screen. Provide a means to reset or clear the program or display, the details of how it’s done are left up to you. Optional features you may choose to add could penalize the operator for reacting before the signal from the program.

*Good programmer's tip:* Design the program top-down. Then write the routines bottom-up. Write them one at a time and thoroughly test each one before integrating them. This way you will have isolated any errors to the routine that you are currently writing. Good programmers follow this method.

INTRODUCTION TO INTERRUPTS - H/W BIT SETTINGS

An interrupt in a microprocessor is a request for service by a process (a background process) that is running outside the direct control of the program. That is, the process is not a subroutine or other
subprogram that is called by the program at a particular point in its execution. For example, a keyboard input is not controlled by the user's program but by someone pressing on the keys. In real-time applications, interrupts are often used to perform periodic functions such as to sample data or to update the error input to a servo loop. Other functions are to measure the elapsed time from a reference starting time or to measure the time interval between external events. In this laboratory exercise, only time-based interrupts are used.

For a (background) timer process to request service, two preconditions must be present: the global interrupt mask bit EA in the Special Function Register, SFR Interrupt Enable, IE, must be enabled (EA = 1) and the local interrupt enable bit must be set.

Global Interrupt Mask

All the time-based interrupts are controlled by the global interrupt mask bit EA in the Interrupt Enable Register (IE). When EA = 0, it masks, i.e., prevents, interrupts. So, for a time-based interrupt to happen, EA must equal 1. This is done by simply executing \( EA = 1; \) in the code in order to enable interrupts. Alternatively, interrupted can be disabled by setting EA to 0. Refer to page 119 in the C8051F120 Documentation for complete details on all of the interrupts.

Local Interrupt Enable Bit

Each time-based interrupt has its own interrupt enable bit. For example, Timer 0 has the ET0, Enable Timer 0 Interrupt, bit (bit 1 in the IE register at 0xA9). The Enable Timer 0 Interrupt is enabled by writing a 1 to the ET0 bit.

Interrupt Request Flag

For an interrupt request to be made, the interrupting process must set its flag. When a timer rolls over from 0xFFFF to 0x0000, a Timer Overflow occurs. An interrupt can be enabled to handle the overflow. If both conditions are met, an interrupt request is made to the microprocessor.

INTRODUCTION TO TIMERS

The 8051 has five programmable timer systems which all offer different functionalities. These timers are labeled Timer 0-4.

Timer 0 and Timer 1

Timer 0 and Timer 1 can be used for several different purposes: a 13-bit counter/timer, 16-bit counter/timer, and an 8-bit counter/timer with auto-reload. Timer 0 can also be used as two 8-bit counter/timers. When either Timer 0 or Timer 1 are set up as timers, the corresponding registers detailed below are incremented at every clock tick. Clock ticks are set to either the system clock, or the system clock divided by 4, 8, 12, or 48. This configuration is specified in the Timer Clock Select bits, T0M & T1M in the CKCON register. SCA1 & SCA0 in the same SFR select the pre-scale divide-by value.

The interrupts for each timer can be enabled by the following statements. The following examples demonstrate how to use Timer 0, but the code for Timer 1 is identically interchangeable.

\[
\begin{align*}
EA &= 1; \quad \text{//enable global interrupts} \\
ET0 &= 1; \quad \text{//enable Timer 0 overflow interrupt}
\end{align*}
\]
A function declaration for the interrupt will also need to be made in the following manner:

```c
void Timer0_ISR (void) interrupt 1;
```

This ISR function will be executed whenever Timer 0 overflows. This timer can be used to generate interrupts at specific frequencies by adjusting TL0 and TH0 in the interrupt service routine (ISR) function. For example, if the desired interrupt rate was every 0.015 seconds, TL0 and TH0 would need to be set in such a way that Timer 0 would increment and overflow within 0.015 seconds. The C8051F120 generates 49,760,000 pulses per second, and it is assumed that the counter is running in divided by 12 mode, so approximately 4,146,667 increments, or ticks, are generated every second. This implies that 62,200 ticks are equivalent to 0.015 seconds. Since Timer 0 and Timer 1 are both 16-bit counters, $2^{16} = 65,536$, so the timer will overflow after it reaches this amount. Thus, in order to generate an interrupt every 0.015 seconds, the timer needs to be set to $(65,536 - 62,200 = 3,336)$ or $0x0D08$. Equivalently, TH0 should be set to $0x0D$, and TL0 should be set to $0x08$. For convenience, the calculations to achieve this are included below. For this to work continuously the timer mode must be set for 16 bits and running (SFR TMOD and TCON) and the ISR must manually reload the counter.

\[
\frac{49,760,000 \text{ticks}}{12 \text{ticks/second}} = 4,146,667 \text{ticks/second} = 62,200 \text{ticks}
\]
\[
0.015 \times 4,146,667 \text{ticks/second} = 62,200 \text{ticks}
\]
\[
2^{16} = 65,536 - 62,200 = 3336 = 0x0D08
\]

The code for this ISR is shown below:

```c
void Timer0_ISR (void) interrupt 1
{
    TH0 = 0x0D; // Reload the Timer 0 High Byte Value.
    TL0 = 0x08; // Reload the Timer 0 Low Byte Value.
}
```

**Timer 2**

This timer can be operated in three different modes. It can be configured as a 16-bit counter/timer with auto-reload or capture, or as a baud rate generator. The operating mode can be selected by configuring the Timer 2 Control register, T2CON. As with Timer 0 and Timer 1, it can be operated at the system clock, the system clock divided by 2 or the system clock divided by 12. It can also operate based on high-to-low transitions on the input pin T2 when configured in capture mode and can count up or down.

**Mode 0**

The timer modes offer distinctly different functionalities. The first mode that Timer 2 offers, Mode 0, is a 16-bit counter/timer with capture. This is an additional feature that Timer 0 and 1 do not offer. Each time a high-to-low transition occurs on input pin T2 or the selected clock, the counter is incremented. The counter overflows when counting from 0xFFFF to 0x0000, and will trigger an interrupt if it is enabled. Also, a high-to-low transition on the T1EX signal line from an external event can cause the 16-bit timer value registers, TH2 and TL2, to be loaded into the capture registers RCAP2H and RCAP2L. This would be used to record the time of an external incident.
**Mode 1**

In this mode, the timer will trigger the TF2 flag whenever an overflow occurs. Unlike the other modes, however, it will automatically reload the values from registers RCAP2H and RCAP2L into TH2 and TL2, and begin counting from those values instead of 0x0000. An interrupt will occur on an overflow if it is enabled. In up-count mode (DCEN2 = 1 and T2EX = 1) the flag is set on the transition from 0xFFFF to 0x0000 and then the reload occurs. In down-count mode (DCEN2 = 1 and T2EX = 0) the flag is set when the counter matches the RCAP2H and RCAP2L value (considered an underflow event) and the counter is reloaded with the value 0xFFFF.

**Mode 2**

In this mode, the timer can act as a baud rate generator for UART0. This is particularly important in correctly setting up the UART to output to the terminal, or for any other purposes. In this mode, the timer operates similarly to Mode 1, except no interrupt is generated on an overflow and the TF2 flag is not set.

**Timer 3**

Timer 3 is similar to Timer 2. This timer is a 16-bit timer that is made up of two 8-bit registers. It can be incremented by the system clock, or system clock divided by 2 or 12. Timer 3 also offers a real-time clock feature, configured by the T3XCLK bit in register TMR3CN. When this bit is set to 1, the timer will use the external oscillator input as its clock divided by 8. The system clock will have no impact on the timer increments. For auto-reloading, the TMR3LH and TMR3LL registers hold the values from which the timer is reloaded on an overflow. Timer 3 only lacks the toggle output feature of Timers 2 and 4.

**Timer 4**

Timer 4 is identical to Timer 2. The timer offers the same functionality, including the 16-bit counter/timer in capture and auto-reload modes, as well as the baud rate generator.

Only Timer 1 can be used for the baud rate generator for UART1. Any of Timers 2, 3, or 4 can act as a real-time clock or free-running counter (up or down) if one is desired. The reload value only needs to be set to 0.

**CLOCKS**

There are several ways to generate clock signals that are used by the timers. In addition to dividing down the SYSCLK by various values shown in Ch. 23, Figure 23.4 and 23.5, Ch. 14 shows a number of options for creating clock signals with different frequencies that may be fine tuned to be more appropriate for a specific application’s timing requirements.
<table>
<thead>
<tr>
<th>Interrupt Source</th>
<th>Interrupt Vector</th>
<th>Priority Order</th>
<th>Pending Flags</th>
<th>Bit addressable?</th>
<th>Cleared by HW?</th>
<th>Enable Flag</th>
<th>Priority Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>0x0000</td>
<td>Top</td>
<td>None</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
<td>Always Enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>Always Highest</td>
</tr>
<tr>
<td>External Interrupt 0 (INT0)</td>
<td>0x0003</td>
<td>0</td>
<td>IE0 (TCON.1)</td>
<td>Y</td>
<td>Y</td>
<td>0</td>
<td>EX0 (IE.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>PX0 (IP.0)</td>
</tr>
<tr>
<td>Timer 0 Overflow</td>
<td>0x000B</td>
<td>1</td>
<td>TF0 (TCON.5)</td>
<td>Y</td>
<td>Y</td>
<td>0</td>
<td>ET0 (IE.1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>PT0 (IP.1)</td>
</tr>
<tr>
<td>External Interrupt 1 (INT1)</td>
<td>0x0013</td>
<td>2</td>
<td>IE1 (TCON.3)</td>
<td>Y</td>
<td>Y</td>
<td>0</td>
<td>EX1 (IE.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>PX1 (IP.2)</td>
</tr>
<tr>
<td>Timer 1 Overflow</td>
<td>0x001B</td>
<td>3</td>
<td>TF1 (TCON.7)</td>
<td>Y</td>
<td>Y</td>
<td>0</td>
<td>ET1 (IE.3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>PT1 (IP.3)</td>
</tr>
<tr>
<td>UART0</td>
<td>0x0023</td>
<td>4</td>
<td>RI0 (SCON0.0)</td>
<td></td>
<td></td>
<td>Y</td>
<td>ES0 (IE.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>T1U (SCONU.1)</td>
<td></td>
<td></td>
<td>Y</td>
<td>PS0 (IP.4)</td>
</tr>
<tr>
<td>Timer 2</td>
<td>0x002B</td>
<td>5</td>
<td>TF2 (TMR2CN.7)</td>
<td></td>
<td></td>
<td>Y</td>
<td>ET2 (IE.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXF2 (TMR2CN.6)</td>
<td></td>
<td></td>
<td>Y</td>
<td>PT2 (IP.5)</td>
</tr>
<tr>
<td>Serial Peripheral Interface</td>
<td>0x0033</td>
<td>6</td>
<td>SPIF (SPIOCN.7)</td>
<td></td>
<td></td>
<td>Y</td>
<td>ESPI0 (EIE1.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WCOL (SPIOCN.6)</td>
<td></td>
<td></td>
<td>Y</td>
<td>PSPI0 (EIP1.0)</td>
</tr>
<tr>
<td>SMBus Interface</td>
<td>0x003B</td>
<td>7</td>
<td>SI (SMB0CN.3)</td>
<td></td>
<td></td>
<td>Y</td>
<td>ESMB0 (EIE1.1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>PSMB0 (EIP1.1)</td>
</tr>
<tr>
<td>ADC0 Window Comparator</td>
<td>0x0043</td>
<td>8</td>
<td>AD0WIN1 (ADCCN.1)</td>
<td></td>
<td></td>
<td>Y</td>
<td>EWADC0 (EIE1.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>PWADC0 (EIP1.2)</td>
</tr>
<tr>
<td>Programmable Counter Array</td>
<td>0x004B</td>
<td>9</td>
<td>CF (PCA0CN.7)</td>
<td></td>
<td></td>
<td>Y</td>
<td>EPCE0 (EIE1.3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CCFn (PCA0CN.n)</td>
<td></td>
<td></td>
<td>Y</td>
<td>PPC0 (EIP1.3)</td>
</tr>
<tr>
<td>Comparator 0 Falling Edge</td>
<td>0x0053</td>
<td>10</td>
<td>CP0F1F (CPT0CN.4)</td>
<td></td>
<td></td>
<td>Y</td>
<td>ECP0F (EIE1.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>PCCF0 (EIP1.4)</td>
</tr>
<tr>
<td>Comparator 0 Rising Edge</td>
<td>0x005B</td>
<td>11</td>
<td>CP0RIF (CPT0CN.5)</td>
<td></td>
<td></td>
<td>Y</td>
<td>ECP0R (EIE1.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>PCCF0 (EIP1.5)</td>
</tr>
<tr>
<td>Comparator 1 Falling Edge</td>
<td>0x0063</td>
<td>12</td>
<td>CP1F1F (CPT1CN.4)</td>
<td></td>
<td></td>
<td>Y</td>
<td>ECP1F (EIE1.6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>PCCF1 (EIP1.6)</td>
</tr>
<tr>
<td>Comparator 1 Rising Edge</td>
<td>0x006B</td>
<td>13</td>
<td>CP1RIF (CPT1CN.5)</td>
<td></td>
<td></td>
<td>Y</td>
<td>ECP1R (EIE1.7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>PCCF1 (EIP1.7)</td>
</tr>
<tr>
<td>Timer 3</td>
<td>0x0073</td>
<td>14</td>
<td>TF3 (TMR3CN.7)</td>
<td></td>
<td></td>
<td>Y</td>
<td>ET3 (EIE2.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXF3 (TMR3CN.6)</td>
<td></td>
<td></td>
<td>Y</td>
<td>PT3 (EIP2.0)</td>
</tr>
<tr>
<td>ADC0 End of Conversion</td>
<td>0x007B</td>
<td>15</td>
<td>AD0INT (ADCCN.5)</td>
<td></td>
<td></td>
<td>Y</td>
<td>EADC0 (EIE2.1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>PADC0 (EIP2.1)</td>
</tr>
<tr>
<td>Timer 4</td>
<td>0x0083</td>
<td>16</td>
<td>TF4 (TMR4CN.7)</td>
<td></td>
<td></td>
<td>Y</td>
<td>ET4 (EIE2.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXF4 (TMR4CN.7)</td>
<td></td>
<td></td>
<td>Y</td>
<td>PT4 (EIP2.2)</td>
</tr>
<tr>
<td>ADC2 Window Comparator</td>
<td>0x008B</td>
<td>17</td>
<td>AD2WIN1 (ADC2CN.0)</td>
<td></td>
<td></td>
<td>Y</td>
<td>EWADC2 (EIE2.3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>PWADC2 (EIP2.3)</td>
</tr>
<tr>
<td>ADC2 End of Conversion</td>
<td>0x0093</td>
<td>18</td>
<td>AD2INT (ADC2CN.5)</td>
<td></td>
<td></td>
<td>Y</td>
<td>EADC2 (EIE2.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>PADC2 (EIP2.4)</td>
</tr>
<tr>
<td>RESERVED</td>
<td>0x009B</td>
<td>19</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>UART1</td>
<td>0x00A3</td>
<td>20</td>
<td>RI1 (SCON1.0)</td>
<td></td>
<td></td>
<td>Y</td>
<td>ES1 (EIE2.6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TI1 (SCON1.1)</td>
<td></td>
<td></td>
<td>Y</td>
<td>PS1 (EIP2.6)</td>
</tr>
</tbody>
</table>
This program uses an interrupt to call the ISR handler function, SWR_ISR(), when the /INT0 line is grounded. Each time the signal makes a low transition, an interrupt will be generated. If the line is held down, the SWR_ISR() function will only be executed once, and not be called again unless the line is released, and grounded again.

/INT0 is configured to be on P0.2
UART0 is used to communicate to the user through ProCOMM

This code was written and tested using the SiLabs IDE and SDCC.

---

#include <c8051f120.h> // SFR declarations.
#include <stdio.h> // Necessary for printf.
#include <putget.h> // Necessary for printf

---

#define EXTCLK 22118400 // External oscillator frequency in Hz
#define SYSCLK 49760000 // Output of PLL derived from (EXTCLK * 9/4)
#define BAUDRATE 115200 // UART baud rate in bps
//define BAUDRATE 19200 // UART baud rate in bps

bit SW2press = 0;

---

void main(void);
void PORT_INIT(void);
void SYSCLK_INIT(void);
void UART0_INIT(void);

void SW2_ISR (void) interrupt 0;

---

void main (void)
{
    bit restart = 0;
    char character;
    unsigned int delay1, delay2 = 0;
    unsigned int randnum = 0;
    unsigned int ones, tenths = 0;

    SFRPAGE = CONFIG_PAGE;

    PORT_INIT(); // Configure the Crossbar and GPIO.
    SYSCLK_INIT(); // Initialize the oscillator.
    UART0_INIT(); // Initialize UART0.

    SFRPAGE = LEGACY_PAGE;
    IT0 = 1; // /INT0 is edge triggered, falling-edge.

    SFRPAGE = UART0_PAGE; // Direct output to UART0

    printf("\033[2J"); // Erase screen and move cursor to the home position.
    printf("MPS Interrupt Switch Test\n\n\r");
    printf("Ground /INT0 on P0.2 to generate an interrupt.\n\n\r");
SFRPAGE = CONFIG_PAGE;
EX0 = 1;       // Enable Ext Int 0 only after everything is settled.

while (1)
{
}

// Interrupt Service Routines
// NOTE: this is an example of what NOT to do in an interrupt handler. No I/O should be done
// in ISRs since I/O is very slow and the handler must execute very quickly.
// This routine stops Timer0 when the user presses SW2.

void SW2_ISR (void) interrupt 0 // Interrupt 0 corresponds to vector address 0003h.
{                          // the keyword "interrupt" defines this as an ISR and the number is determined by the
    printf("/INT0 has been grounded!\n\n\r");
}

// Configure the Crossbar and GPIO ports
void PORT_INIT(void)
{
    char SFRPAGE_SAVE = SFRPAGE;   // Save Current SFR page.
    SFRPAGE = CONFIG_PAGE;

    WDTCN = 0xDE;                 // Disable watchdog timer.
    WDTCN = 0xAD;
    EA = 1;                      // Enable interrupts as selected.

    XBR0 = 0x04;                  // Enable UART0.
    XBR1 = 0x04;                  // /INT0 routed to port pin.
    XBR2 = 0x40;                  // Enable Crossbar and weak pull-ups.

    P0MDOUT = 0x01;               // P0.0 (TX0) is configured as Push-Pull for output.
    P0 = 0x06;                   // P0.1 (RX0) is configure as Open-Drain input.
    EX0 = 1;                     // /INT0 routed to port pin.
    EX0 = 1;                     // Enable External Interrupt 0 when everything is settled.
    SFRPAGE = SFRPAGE_SAVE;      // Restore SFR page.
}

// Initialize the system clock
void SYSCLK_INIT(void)
{
    int i;

    char SFRPAGE_SAVE = SFRPAGE;  // Save Current SFR page.
    SFRPAGE = CONFIG_PAGE;

    OSCXCN = 0x67;                // Start external oscillator
    for(i=0; i < 256; i++);       // Wait for the oscillator to start up.
    while(!((OSCXCN & 0x80)));   // Check to see if the Crystal Oscillator Valid Flag is set.
CLKSEL = 0x01; // SYSCLK derived from the External Osc cct.
OSCICN = 0x00; // Disable the internal oscillator.

SFRPAGE = CONFIG_PAGE;
PLL0CN = 0x04;
SFRPAGE = LEGACY_PAGE;
FLSCL = 0x10;
SFRPAGE = CONFIG_PAGE;
PLL0CN |= 0x01;
PLL0DIV = 0x04;
PLL0FLT = 0x01;
PLL0MUL = 0x09;
for(i=0; i < 256; i++);
PLL0CN |= 0x02;
while(!(PLL0CN & 0x10));
CLKSEL = 0x02; // SYSCLK derived from the PLL.
SFRPAGE = SFRPAGE_SAVE; // Restore SFR page.

// UART0_Init
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.
void UART0_INIT(void)
{

char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page.

SFRPAGE = UART0_PAGE;
SCON0 = 0x50; // Set Mode 1: 8-Bit UART
SSTA0 = 0x10; // UART0 baud rate divide-by-two disabled (SMOD0 = 1).

SFRPAGE = TIMER01_PAGE;
TMOD &= ~0xF0;
TMOD |= 0x20; // Timer1, Mode 2: 8-bit counter/timer with auto-reload.
TH1 = -(SYSCLK/BAUDRATE/16); // Set Timer1 reload value for baudrate
CKCON |= 0x10; // Timer1 uses SYSCLK as time base.
TL1 = TH1;
TR1 = 1; // Start Timer1.

SFRPAGE = UART0_PAGE;
TI0 = 1; // Indicate TX0 ready.
SFRPAGE = SFRPAGE_SAVE; // Restore SFR page.