Experiment No. 8: 8051 Microcontroller - Serial Communication and Interrupts

1. Aim

To understand and implement serial communication (UART) and interrupt handling mechanisms (external and timer interrupts) using the 8051 microcontroller.

2. Objectives

Upon completion of this experiment, the student will be able to:

- **Explain the basics of serial communication, including UART, Baud Rate, and data framing.**
- **Configure the 8051's built-in UART for transmitting and receiving serial data.**
- **Differentiate between polling and interrupt-driven I/O.**
- **Understand the concept of interrupts, interrupt vectors, and Interrupt Service Routines (ISRs) in the 8051.**
- **Configure and handle external interrupts to respond to external events.**
- **Configure and utilize timer interrupts for periodic tasks.**
- **Write C programs for serial communication and interrupt-driven applications.**
- **Debug and verify the functionality of serial communication and interrupt handlers.**

3. Theory

The 8051 is a popular 8-bit microcontroller family known for its integrated peripherals, including a UART (Universal Asynchronous Receiver/Transmitter) for serial communication and multiple timers/counters that can generate interrupts. These features are fundamental for real-time control and data exchange in embedded systems.

3.1. Serial Communication (UART)

Serial communication involves transmitting data one bit at a time over a single wire. This is efficient for long-distance communication and connections between devices with fewer pins. The 8051 has a full-duplex (simultaneous send and receive) serial port.

- **UART (Universal Asynchronous Receiver/Transmitter): A hardware peripheral that converts parallel data from the microcontroller into a serial stream for transmission, and converts incoming serial data into parallel data for the microcontroller.**
- **Baud Rate: The rate at which data is transferred in bits per second (bps). Common baud rates include 9600, 19200, 115200.**

- **Data Framing: Serial data is typically transmitted in "frames," which include:**
  - **Start Bit: A '0' bit that signals the beginning of a data frame.**
  - **Data Bits: 5 to 9 bits (commonly 8 bits) representing the actual data.**
  - **Parity Bit (Optional): Used for error detection.**
  - **Stop Bit(s): 1 or 2 bits (commonly 1) that signal the end of the data frame.**
- **8051 Serial Port Registers:**
  - **SBUF (Serial Buffer): An 8-bit register used for both transmitting and receiving data. Writing to SBUF loads data for transmission; reading from SBUF retrieves received data.**
  - **SCON (Serial Control Register): An 8-bit Special Function Register (SFR) that controls the operating mode of the serial port.**
    - **SM0, SM1: Mode select bits (00: Mode 0, 01: Mode 1, 10: Mode 2, 11: Mode 3). Mode 1 is typically used for variable baud rate 8-bit UART.**
    - **REN (Receive Enable): Set to 1 to enable serial reception.**
    - **TI (Transmit Interrupt Flag): Set by hardware when a byte has been transmitted. Cleared by software.**
    - **RI (Receive Interrupt Flag): Set by hardware when a byte has been received. Cleared by software.**
  - **PCON (Power Control Register): The SMOD bit (PCON.7) doubles the baud rate in Modes 1, 2, and 3 if set to 1.**
- **Baud Rate Generation (Mode 1): In 8051's Mode 1, the baud rate is generated by Timer 1 in Mode 2 (8-bit auto-reload mode).**
  - **Timer 1 is configured to overflow at a specific rate to generate the desired baud rate.**
  - **The formula for Baud Rate in Mode 1 (assuming SMOD = 0) is:**
    Baud Rate = (Oscillator Frequency / 12) / (32 * (256 - TH1))
  - **If SMOD = 1 (PCON.7 = 1), Baud Rate = (Oscillator Frequency / 12) / (16 * (256 - TH1))**
  - **Where:**
    - **Oscillator Frequency: The crystal frequency of the 8051 (e.g., 11.0592 MHz).**
    - **TH1: The 8-bit reload value for Timer 1.**
  - **To find TH1 for a desired Baud Rate:**
    TH1 = 256 - [(Oscillator Frequency / 12) / (32 * Baud Rate)]
  - **Numerical Example: For an 11.0592 MHz crystal and a desired Baud Rate of 9600 bps (with SMOD = 0):**
    TH1 = 256 - [(11059200 / 12) / (32 * 9600)]
    TH1 = 256 - [921600 / 307200]
    TH1 = 256 - 3 = 253 (decimal) or FDH (hexadecimal)
    This specific crystal frequency (11.0592 MHz) is commonly chosen because it allows exact integer division for standard baud rates, minimizing baud rate errors.

3.2. Interrupts

Interrupts are hardware or software events that cause the microcontroller to temporarily suspend its normal program execution, jump to a special routine called an Interrupt Service Routine (ISR), execute the ISR, and then return to the point where it was interrupted. This allows the microcontroller to respond to asynchronous events without constantly polling for their occurrence.

- **Interrupt Sources in 8051: The 8051 has 5 (or 6 for 8052) interrupt sources:**
  - **External Interrupt 0 (overline{INT0}, P3.2)**
  - **Timer 0 Overflow Interrupt (TF0)**
  - **External Interrupt 1 (overline{INT1}, P3.3)**
  - **Timer 1 Overflow Interrupt (TF1)**
  - **Serial Port Interrupt (RI or TI)**
  - **(Timer 2 Overflow Interrupt for 8052 and above)**
- **Interrupt Enable Registers:**
  - **IE (Interrupt Enable Register): Used to enable/disable individual interrupt sources and global interrupts.**
    - **EA (IE.7): Global Interrupt Enable/Disable (set to 1 to enable all interrupts).**
    - **EX0 (IE.0): External Interrupt 0 enable.**
    - **ET0 (IE.1): Timer 0 Interrupt enable.**
    - **EX1 (IE.2): External Interrupt 1 enable.**
    - **ET1 (IE.3): Timer 1 Interrupt enable.**
    - **ES (IE.4): Serial Port Interrupt enable.**
  - **IP (Interrupt Priority Register): Used to set the priority level of each interrupt. (High or Low priority).**
- **Interrupt Service Routine (ISR): A special function written to handle a specific interrupt. When an interrupt occurs, the microcontroller jumps to the corresponding ISR. In C, ISRs are defined using specific keywords (e.g., `void ISR_function() interrupt vector_number`).**
- **Interrupt Vectors: Fixed memory addresses where the CPU jumps when a specific interrupt occurs.**
  - **External Interrupt 0: 0003H**
  - **Timer 0: 000BH**
  - **External Interrupt 1: 0013H**
  - **Timer 1: 001BH**
  - **Serial Port: 0023H**
- **External Interrupts (overline{INT0}, overline{INT1}): Triggered by events on dedicated pins (P3.2 and P3.3).**
  - **TCON (Timer Control Register): Controls external interrupt edge/level triggering.**
    - **IT0 (TCON.0): For overline{INT0}, 0=level triggered, 1=edge triggered (falling edge).**
    - **IE0 (TCON.1): External Interrupt 0 flag.**
    - **IT1 (TCON.2): For overline{INT1}, 0=level triggered, 1=edge triggered (falling edge).**
    - **IE1 (TCON.3): External Interrupt 1 flag.**
- **Timer Interrupts (TF0, TF1): Triggered when a timer/counter overflows.**

- **TMOD (Timer Mode Register): Configures the operating mode for Timer 0 and Timer 1.**
- **TLx, THx (Timer Low/High Byte): 8-bit registers holding the current count for Timer x.**
- **TCON (Timer Control Register):**
  - **TR0 (TCON.4): Run control bit for Timer 0.**
  - **TF0 (TCON.5): Timer 0 Overflow Flag.**
  - **TR1 (TCON.6): Run control bit for Timer 1.**
  - **TF1 (TCON.7): Timer 1 Overflow Flag.**
- **Timer Mode 1 (16-bit Timer): TMOD = 01H (for Timer 0 or 1). The timer counts from THx:TLx up to FFFFH. On overflow, TFx flag is set, and an interrupt is generated if enabled.**
- **Calculating Delay for Timer Interrupt:**
  Delay = (65536 - (Initial Value of THx:TLx)) * (12 / Oscillator Frequency)
  Initial Value = 65536 - (Desired Delay * (Oscillator Frequency / 12))
- **Numerical Example: For an 11.0592 MHz crystal and a desired delay of 50 ms (0.05 s) using Timer 0 in Mode 1:**
  Ticks per second = 11059200 / 12 = 921600
  Ticks for 50 ms = 921600 * 0.05 = 46080
  Initial Value = 65536 - 46080 = 19456 (decimal)
  In hexadecimal: 19456 decimal = 4C00H
  So, TH0 = 4CH, TL0 = 00H.

4. Materials Required

- **8051 Microcontroller Development Board (with serial port, LEDs, push button, LCD)**
- **USB-to-Serial Converter (if board uses RS232/UART for PC connection)**
- **PC with a Terminal Emulator Software (e.g., PuTTY, Tera Term, Realterm)**
- **Keil uVision IDE (or similar 8051 C compiler)**
- **USB Programmer/Debugger (e.g., USBASP, ST-Link, or built-in programmer on board)**
- **Connecting Wires/Jumpers**
- **External Push Button**
- **LED with current limiting resistor**
- **Oscilloscope (optional, for observing serial signals or timing)**

5. Procedure

Part A: Serial Communication (Transmit and Receive)

1. **Hardware Setup:**
   - **Connect the 8051 development board's serial port (TxD - P3.1, RxD - P3.0) to the PC via a USB-to-Serial converter. Ensure correct RxD-TxD cross-connection (TxD of 8051 to RxD of PC, RxD of 8051 to TxD of PC).**
   - **Power on the 8051 board.**
2. **Software Setup (PC):**
   - **Open a terminal emulator (e.g., PuTTY).**

- Configure the serial port settings:
  - Serial line: Select the COM port assigned to your USB-to-Serial converter (check Device Manager).
  - Speed (Baud): Set to 9600 (or desired baud rate).
  - Data bits: 8
  - Stop bits: 1
  - Parity: None
  - Flow control: None

3. C Program for Serial Communication (Transmit & Receive Loopback):
   - Aim: Transmit a string "Hello from 8051!" and then echo back any character received from the PC.
   - Microcontroller: 8051
   - Crystal Frequency: 11.0592 MHz (for accurate baud rate)
   - Baud Rate: 9600 bps
   - C Code (using Keil C51 syntax):
   - C

```c
#include <reg51.h>
#include <string.h>

void delay_ms(unsigned int ms) {
    unsigned int i, j;
    for(i = 0; i < ms; i++) {
        for(j = 0; j < 120; j++); // Approx. 1ms delay for 11.0592MHz
    }
}

void UART_Init() {
    TMOD = 0x20;    // Timer 1, Mode 2 (8-bit auto-reload) for baud rate
    TH1 = 0xFD;     // Reload value for 9600 baud rate @ 11.0592MHz (256 - 3 = 253 = FDH)
    SCON = 0x50;    // Serial Mode 1 (8-bit UART, variable baud rate), REN = 1 (Enable Rx)
    TR1 = 1;        // Start Timer 1
    PCON &= 0x7F;   // Clear SMOD bit (PCON.7 = 0) to keep baud rate normal
                    // Baud Rate = (Oscillator_Freq / 12) / (32 * (256 - TH1))
                    // 9600 = (11059200 / 12) / (32 * (256 - 253)) = 921600 / (32 * 3) = 921600 / 96 =
9600
}

void UART_TxChar(char ch) {
    SBUF = ch;      // Load data to SBUF for transmission
    while (TI == 0); // Wait until transmit interrupt flag is set (transmission complete)
    TI = 0;         // Clear TI flag for next transmission
}

char UART_RxChar() {
    while (RI == 0); // Wait until receive interrupt flag is set (reception complete)
    RI = 0;         // Clear RI flag for next reception
```

```
    return SBUF;    // Return received data
}

void main() {
    char message[] = "Hello from 8051!\r\n"; // \r\n for new line in terminal
    char received_char;
    int i;

    UART_Init();    // Initialize UART

    // Transmit initial message
    for (i = 0; i < strlen(message); i++) {
        UART_TxChar(message[i]);
        delay_ms(10); // Small delay between characters for clarity
    }

    while(1) {
        received_char = UART_RxChar(); // Wait for and receive a character from PC
        UART_TxChar(received_char);    // Echo the received character back to PC
    }
}
```

-
-

4. **Compilation and Flashing:**
    - **Compile the C code using Keil uVision. Resolve any errors.**
    - **Flash the generated .hex file to the 8051 microcontroller using your programmer.**
5. **Execution and Observation:**
    - **Open the terminal emulator.**
    - **Reset the 8051 board.**
    - **Observe the "Hello from 8051!" message appearing on the terminal.**
    - **Type characters on your PC's keyboard within the terminal. Observe if the 8051 echoes back the characters you type.**

Part B: External Interrupt Handling

1. **Hardware Setup:**
    - **Connect a push button to the overlineINT0 pin (P3.2) of the 8051. Connect the other end of the button to ground. Use a pull-up resistor (e.g., 10k Ohm) for P3.2 if the internal pull-up is not strong enough or disabled.**
    - **Connect an LED with a current-limiting resistor (e.g., 220 Ohm) to a general-purpose I/O pin (e.g., P1.0).**
2. **C Program for External Interrupt:**
    - **Aim: Toggle an LED connected to P1.0 every time a button connected to overlineINT0 (P3.2) is pressed.**
    - **Microcontroller: 8051**
    - **C Code (using Keil C51 syntax):**

- C

```c
#include <reg51.h>

sbit LED = P1^0; // Assign P1.0 to LED

void External_Int0_ISR() interrupt 0 // Interrupt Vector 0 for INT0
{
   LED = ~LED; // Toggle the LED
}

void main() {
   // Configure External Interrupt 0
   IT0 = 1;    // Configure INT0 for falling edge triggered
   EX0 = 1;    // Enable External Interrupt 0
   EA = 1;     // Enable Global Interrupt

   LED = 0;    // Initialize LED to OFF state

   while(1) {
      // Main program can do other tasks or simply wait
   }
}
```

-
-

3. **Compilation and Flashing:** Compile and flash the code to the 8051.
4. **Execution and Observation:**
   - Press the button connected to P3.2.
   - Observe the LED connected to P1.0. Each press (falling edge) should toggle the LED's state (ON to OFF, OFF to ON).

Part C: Timer Interrupt for Periodic Task

1. **Hardware Setup:**
   - Connect an LCD (e.g., 16x2 LCD) to the 8051's ports (typically P2 for data, and some P0/P1 pins for control RS, RW, E). (Assume LCD interfacing is already familiar or use a pre-wired setup on the trainer kit).
   - (No additional external components specific to timer interrupt for this task, as the LCD updates are internal).
2. **C Program for Timer Interrupt and LCD Update:**
   - Aim: Use Timer 0 interrupt to update a counter on an LCD display every 100 milliseconds.
   - Microcontroller: 8051
   - Crystal Frequency: 11.0592 MHz
   - Timer 0 Mode: Mode 1 (16-bit timer)
   - C Code (using Keil C51 syntax):

- C

```c
#include <reg51.h>
#include <stdio.h> // For sprintf, if using standard library

// LCD control pins - adjust according to your hardware
sbit RS = P2^0; // Register Select
sbit EN = P2^1; // Enable
#define LCD_DATA P2 // Assuming D4-D7 for data (4-bit mode)

unsigned int timer_count = 0;
char lcd_buffer[16]; // Buffer for LCD display

void LCD_Cmd(unsigned char cmd) {
    LCD_DATA = cmd & 0xF0; // Higher nibble
    RS = 0; EN = 1; delay_ms(1); EN = 0; delay_ms(1);
    LCD_DATA = (cmd << 4) & 0xF0; // Lower nibble
    RS = 0; EN = 1; delay_ms(1); EN = 0; delay_ms(1);
}

void LCD_Char(unsigned char dat) {
    LCD_DATA = dat & 0xF0; // Higher nibble
    RS = 1; EN = 1; delay_ms(1); EN = 0; delay_ms(1);
    LCD_DATA = (dat << 4) & 0xF0; // Lower nibble
    RS = 1; EN = 1; delay_ms(1); EN = 0; delay_ms(1);
}

void LCD_Init() {
    delay_ms(20);
    LCD_Cmd(0x02); // Return Home (for 4-bit init)
    LCD_Cmd(0x28); // 4-bit mode, 2 lines, 5x7 dots
    LCD_Cmd(0x0C); // Display ON, Cursor OFF
    LCD_Cmd(0x06); // Increment cursor, No shift
    LCD_Cmd(0x01); // Clear display
    delay_ms(2);
}

void LCD_String(char *str) {
    while (*str) {
        LCD_Char(*str++);
    }
}

// Delay function (already defined in Part A, copy/paste or include header)
void delay_ms(unsigned int ms) {
    unsigned int i, j;
    for(i = 0; i < ms; i++) {
        for(j = 0; j < 120; j++); // Approx. 1ms delay for 11.0592MHz
```

```
    }
}

void Timer0_ISR() interrupt 1 // Interrupt Vector 1 for Timer 0
{
    TH0 = 0x4C; // Reload TH0 for 100ms (19456 decimal)
    TL0 = 0x00; // Reload TL0 for 100ms (19456 decimal)
                // Calculated as 65536 - (0.1 * 11059200 / 12) = 65536 - 92160 = 19456 = 4C00H

    timer_count++; // Increment counter on each 100ms interrupt
}

void main() {
    LCD_Init(); // Initialize LCD

    // Configure Timer 0
    TMOD = 0x01;    // Timer 0, Mode 1 (16-bit timer)
    TH0 = 0x4C;     // Load initial value for 100ms delay
    TL0 = 0x00;     // (0x4C00 = 19456 decimal)
    ET0 = 1;        // Enable Timer 0 interrupt
    EA = 1;         // Enable Global Interrupt
    TR0 = 1;        // Start Timer 0

    LCD_String("Counter: "); // Display static text
    LCD_Cmd(0xC0);           // Go to second line

    while(1) {
        // Update LCD with timer_count value
        sprintf(lcd_buffer, "%u", timer_count); // Convert integer to string
        LCD_Cmd(0xC0);      // Go to second line (cursor position for counter)
        LCD_String(lcd_buffer);
        delay_ms(50); // Small delay to avoid flickering/too frequent updates
    }
}
```

-
-

3. **Compilation and Flashing: Compile and flash the code to the 8051.**
4. **Execution and Observation:**
    - **Observe the LCD display. A counter should be incrementing approximately every 100 milliseconds.**
    - **Verify the periodicity of updates. If an oscilloscope is available, you could toggle an unused pin in the ISR to visually confirm the interrupt frequency.**

6. Observations

Record your observations during the execution of each part.

- **Part A: Serial Communication**

    - **Initial Output: What message appeared on the terminal after resetting the 8051? (e.g., "Hello from 8051!")**
    - **Echo Functionality: Describe what happens when you type characters on the PC terminal. (e.g., "Each character typed on the PC was immediately echoed back and displayed on the terminal.")**
    - **Baud Rate Accuracy: Comment on the clarity and consistency of the communication, suggesting proper baud rate setup.**
    - **Analysis: Confirm that the 8051's UART successfully transmitted data to the PC and received data from the PC, demonstrating basic serial communication.**
- **Part B: External Interrupt**
    - **LED Behavior: Describe the behavior of the LED connected to P1.0. (e.g., "The LED toggled its state (ON to OFF, OFF to ON) precisely when the button connected to P3.2 was pressed.")**
    - **Responsiveness: Comment on the responsiveness of the LED to button presses.**
    - **Analysis: Confirm that the 8051 correctly detected the external falling edge interrupt on overlineINT0 and executed the ISR to toggle the LED, demonstrating successful external interrupt handling.**
- **Part C: Timer Interrupt**
    - **LCD Display: Describe what is displayed on the LCD. (e.g., "The LCD initially displayed 'Counter: ' and then a numerical value on the second line.")**
    - **Counter Increment: Observe how frequently the counter value on the LCD updates. (e.g., "The counter value on the LCD incremented approximately every 0.1 seconds (100 milliseconds).")**
    - **Stability: Comment on the stability and accuracy of the timing.**
    - **Analysis: Confirm that Timer 0 was successfully configured to generate periodic interrupts, and its ISR was executed to update the counter, demonstrating the use of timer interrupts for periodic tasks.**

7. Deliverables

1. **Explanation of Serial Communication Concepts: (UART, Baud Rate, Framing, 8051 Serial Registers, Baud Rate Calculation with numerical example).**
2. **Explanation of 8051 Interrupt Concepts: (Interrupt Sources, Interrupt Enable/Priority Registers, Interrupt Vectors, External Interrupt Configuration, Timer Interrupt Configuration, Timer Delay Calculation with numerical example).**
3. **C Code for Serial Communication: With detailed comments for UART initialization, transmission, and reception functions.**
4. **C Code for External Interrupt: With detailed comments for interrupt configuration and ISR.**
5. **C Code for Timer Interrupt: With detailed comments for timer configuration, ISR, and LCD update logic.**
6. **Serial Communication Output: A screenshot of the terminal emulator showing the transmitted string and echoed characters.**

7. **Demonstration of External Interrupt: A photograph or video clearly showing the LED toggling in response to button presses (description sufficient if visual media not possible).**
8. **Demonstration of Timer Interrupt: A photograph or video clearly showing the LCD displaying the incrementing counter (description sufficient if visual media not possible).**
9. **Observations and Analysis: (As recorded in Section 6).**

8. Conclusion

This experiment provided a comprehensive practical understanding of serial communication and interrupt handling in the 8051 microcontroller. We successfully implemented two-way serial communication between the 8051 and a PC, observing data transmission and reception. Furthermore, we configured and utilized both external interrupts to respond to immediate button presses and timer interrupts for precise periodic tasks like updating an LCD display. This hands-on experience is crucial for developing robust and responsive embedded systems that interact with external devices and perform real-time operations efficiently.
