ENGR 323L: Embedded System Design
Department of Engineering, Trinity College
Instructor: Professor. T. Ning

Laboratory #3: Multiprocessor communications via universal asynchronous receiver and transmitter (UART)

Problem Statement:

In this lab, SiliconLab 8051F120DK (or 8051F500DK) evaluation board, which has internal RAM (8K + 256 bytes), and your wire-wrapped 8051 board will be used to perform multiprocessor communications through UART. They will be configured, respectively, as the ***master*** and the ***salve*** to allow serial communication at a speed of 9600 baud, i.e., 9600 bits per second.

Transmitter (Master—8051F120DK/8051F500DK**)**
Invoke an interrupt service routine (ISR) that will fill a memory block (e.g., 0x00-0x3F) or a character array with hex values from 00 to 3F in the RAM block; specify the designated receiver(s) (within the local UART network) and start transmission. *This part should be implemented using C programming.*

Receiver (Slave—wire-wrapped 8051 board**)**
Determine if the slave is the targeted-receiver and, if so, start receiving data and saving them to RAM locations: 30-6F. Display "***DONE***" on a 7-seg display when all bytes are received. Invoke an interrupt service to display the received data sequentially with an interval delay of one second to allow visual verification. Display "***END***" when completed. *This part should be implemented using Assembly programming*.

8051 and Serial Communication

The 8051 CPUs are equipped with an on-chip serial interface to allow serial data transmission and reception. The serial port of 8051 microcontrollers is full duplex, i.e., it can both transmit and receive data. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the register. (However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost.) The serial port receive and transmit registers are both accessed at Special Function Register **SBUF** Writing to SBUF loads the transmit data register, and reading SBUF accesses a physically separate receive data register.

The serial data communication mode is controlled by configuring the Serial Port Control (SCON) SFR and the Power Mode Control (PCON) SFR.

Serial Control (SCON) SFR (098h):

Bit	Name	Bit Address	Explanation of Function
7	SM0	9Fh	Serial port mode bit 0
6	SM1	9Eh	Serial port mode bit 1.
5	SM2	9Dh	Multiprocessor Communications Enable
4	REN	9Ch	Receiver Enable. This bit must be set in order to receive characters.
3	TB8	9Bh	Transmit bit 8. The 9th bit to transmit in mode 2 and 3.
2	RB8	9Ah	Receive bit 8. The 9th bit received in mode 2 and 3.
1	TI	99h	Transmit Flag. Set when a byte has been completely transmitted.
0	RI	98h	Receive Flag. Set when a byte has been completely received.

SM0/SM1	Operation
0/0 Mode - 0	Serial data enters and exits through RxD. TxD outputs the shift clock. 8 bits are transmitted/received (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.
0/1 Mode -1	10 bits are transmitted (through TxD) or received (through RxD): a start bit, 8 data bits (LSB first), and a stop bit. On *receive*, the stop bit goes into RB8 in SCON. The baud rate is variable.
1/0 Mode -2	11 bits are transmitted (through TxD) or received (through RxD): start bit, 8 data bits (LSB first), 9th data bit, and a stop bit. On Transmit, the 9th data bit (TB8 in SOON) is assigned the value of 0 or 1 (or used for the parity bit in the PSW. On *receive*, the 9th data bit goes into RB8 in SCON. The baud rate is either 1/32 or 1/64 the oscillator frequency.
1/1 Mode -3	11 bits are transmitted (through TxD) or received (through RxD): a start bit, 8 data bits (LSB first), 9th data bit, and a stop bit. Mode 3 is the same as Mode 2 except the baud rate is variable.

* In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit it REN = 1.

Setting the Serial Port Baud Rate:

SM0	SM1	Serial Mode	Explanation	Baud Rate
0	0	0	8-bit Shift Register	Oscillator / 12
0	1	1	8-bit UART	Set by Timer 1 (*)
1	0	2	9-bit UART	Oscillator / 32 (*)
1	1	3	9-bit UART	Set by Timer 1 (*)

(*) Note: The baud rate indicated can be doubled if bit PCON.7 (i.e., SMOD) is set.

Once the Serial Port mode has been configured, the user must determine the serial baud rate for modes 1 and 3. The baud rate is automatically determined by the oscillator's

frequency in modes 0 and 2, e.g., if a crystal frequency is 11.059 MHz, the baud rate in mode 0 will always be 921,583 baud.

In modes 1 and 3, the baud rate is determined by Timer_1 overflow interrupt. A common approach is to put Timer_1 in 8-bit auto-reload mode and initialize TH1 with a proper reload value that causes Timer_1 to overflow at the frequency to generate the desired baud rate. The calculation is determined by the following equation:

$$Baud\_rate = \frac{2^{SMOD} * crystal\_frequency}{32*12*[256-(TH1)]}$$

i.e.,
TH1 = 256 - ((crystal frequency / 384) / Baud) *if PCON.7 is cleared or*

TH1 = 256 - ((crystal frequency / 192) / Baud) *if PCON.7 is set.*

For example, to generate 14400 baud using an 11.059 MHz crystal,
TH1 = 256 - ((11059000 / 384) / 14400)
TH1 = 256 - 2 = 254.

**Note that SiliconLab- 8051F500K kit uses different crystal frequencies*

Configuring a desired baud rate:
1. Configure Serial Port to operate in mode 1 or 3.
2. Configure Timer_1 to in 8-bit auto-reload based on the target crystal frequency.
3. Set TH_1 to an appropriate value (integer is always guaranteed) for a desired baud rate.
4. Set PCON.7 (SMOD) if want to double the baud rate.

Multiprocessor Communications via UART:
In order to allow multiprocessor communication, the serial port must operate in mode-2 or mode-3 for multiprocessor communications. In these modes, 9 data bits are received where the 9th bit stores in RB_8. The 9-bit transmission is to be programmed that the serial interrupt will be activated only if $RB_8 = 1$**.** This feature is enabled by setting bit SM_2 in SCON. SM_2 is a flag for multiprocessor communication.

Generally, whenever a byte has been received the 8051 will set the "RI" (Receive Interrupt) flag. This lets the program know that a byte has been received and that it needs to be processed. However, when SM2 is set the "RI" flag will only be triggered if the 9th bit is "1". In other words, if SM_2 is set and the 9th bit is cleared, the RI flag will not be set. A way to use this feature in multiprocessor systems is as follows:

When the master needs to transmit a block of data to a target receiver/slave, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is set to “1” while “0” in a data byte. With $SM_2 = 1$, no slave will

be interrupted by data bytes. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The identified slave will clear its SM_2 bit and begin to receive data bytes that follow. The other non-targeted slaves leave SM_2 set and ignore data bytes.

Suggested Implementation Steps:

- Configure the Serial Port Control Register (SCON) to operate in multiprocessor variable mode (Mode 3)
- *Serial Interrupt (ES0)*, *Timer_0 Overflow Interrupt (TF0)* and *External Interrupt_0* (IE0) should be activated
- Configure serial transmission at a minimum baud rate of 9600
- Configure the desired baud rate by setting Timer_1 into 8-bit auto-reload mode. Based on the target crystal frequency, load the most appropriate initial value into TH1 for the desired baud rate
- To double the baud rate the SMOD bit (bit 7 in PCON) can be set
- The special function registers IE, IP, TCON, TMOD, SCON should be properly configured upon initialization

In SiliconLabF120DK, the watchdog timers should be disabled because otherwise the software program is reset every time an interrupt occurs. The timer is disabled by writing 0xDE to the WDTCN register followed within four system clock pulses by 0xAD, e.g.,

```
WDTCN = 0xDE;
WDTCN = 0xAD;
```

To use the external oscillator at 22.1184 MHz, e.g.,

```
// Configure The External Oscillator to use a 22.1184 MHz frequency
        int i = 0;
        SFRPAGE   = CONFIG_PAGE;
//  enable the external oscillator.
        OSCXCN    = 0x67;
//  wait 1ms for initialization
        for (i = 0; i < 3000; i++);
// poll for XTLVLD => '1'
        while ((OSCXCN & 0x80) == 0);
// switch the system clock to the external oscillator.
        CLKSEL    = 0x01;
        OSCICN    = 0x00;
```