Assignment 5 ISR-driven Serial Communication Using UART – CSE 325, Fall 2010

Assignment Objectives

1. To learn the basic operations and programming techniques of serial communication, and to develop a set of interface functions.
2. To learn the ISR-driven approach for block data transfer.

Project Assignment

In Assignment 4, you developed a program to display sequences of patterns on an 8X8 LED matrix. The sequence is given statically. In this assignment, you will develop a UART communication driver to receive new sequences downloaded from a PC host. UART (Universal Asynchronous Receiver Transmitter), also known as serial port, is the most common peripheral in computing devices. While the operations of transmission and receiving on UART are done character by character (or byte by byte), a driver can be built to deal with a block of characters, such as sending and receiving a line of characters.

While the LED display of the current sequence is running, the driver to be developed in the assignment can receive downloads of new LED sequences. This means there should not be any busy waiting in UART operations. The UART receiving operations must be interrupt-driven. After a download operation is initialized, we will let ISRs take care of each receive operation until the transfer is completed. To test your driver, the COM port of M5211 demo board is connected to COM1 port of the host PC. You can use a hyperterminal program to send binary files containing LED sequences. The file format is shown in the example:

Field	*Type*	*Length (bytes)*	*comment*
start	*char[8]*	*8*	*"startLED"*
no_of_seq	*int*	*4*	*no of led sequences in the file*
seq_length	*array of int*	*variable*	*number of patters of the sequences*
duration	*array of int*	*variable*	*the display duration for each pattern*
seq_pattern	*array of PATTERN*	*variable*	*the patterns of the sequence*
stop	*char[4]*	*4*	*"stop"*

where PATTERN is defined as:

```
typedef struct
{
        uint8 red[8];     // 8 bytes (64 bits) for 64 red leds
        uint8 green[8]  // 8 bytes (64 bits) for 64 green leds
} PATTERN;
```

The driver implements a receive_seq function, a status checking function, and an open function to initialize a serial port, such as baud rate, parity bit, data size, stop bit, etc. The functions are defined as:

- int open_UART(int channel, int baud_rate): *channel_open* function initializes the UART channel device of Coldfire 5211 processor and set the given baud rate. We will consider two possible channels, 0 and 1 for UART0 and UART1, respectively. The format of transfer is assumed to be with 8 data bits, even parity, 1 stop bit, and the flow control of RTS and CTS is enabled. The function returns 0 and is ready for receiving operation.
- int receive_seq(int channel, uint8 *buf): *buf* points to a buffer where the received LED sequences from UART *channel* can be saved. The function returns 0 if the receiving operation is initialized and get started, or -1 if the channel is not available (e.g., the channel has not been initialized or busy on an ongoing receive operation). We will assume the buffer has a size of 1K bytes and the binary files to be downloaded are less than 1K bytes. Hence, there is no need to address any potential buffer overflow problem.

  The function should return back to the calling user program immediately after the receiving operation is set up. The UART receiving operation, after initiated, should be carried out by a receiver ready (RxRDY) of FIFO full (FFULL) ISR, which reads received data from UART receive buffer (UBR) and saves the data in the given buffer. The receiving operation will end once a *stop* field is received. Note that there is no need to save the start and stop fields into the buffer. A simple format error of incorrect start field should be detected.
- int receive_status (int channel): The function checks the status of any ongoing receive_seq function on the given channel. It returns the length of the number of bytes received if the previous receiving and transmission operations are done successfully. Or, it returns -1 when the receiving operation is in progress and -2 for format error..

You may need to declare two buffers of 1K bytes: one holds the sequences under display and the other is used to save the downloaded sequences. Initially, your program assumes no existing sequences in the buffers. After invoking *channel_open* and *receive_seq*, it is waiting for the completion of the 1st download by checking the return value of *receive_status*. Then, the program can enter an endless loop to display the newly arrived sequences of patterns and to receive additional downloads using the alternate buffer.

The LED display is similar to that of Assignment 4, and once the display starts to run, the LED matrix should show the 1st sequence. Upon a SW1 interrupt, the display switches to the next sequence and wraps around when finishing all sequences. To display a sequence, the list of patters are shown repeatedly and each pattern is displayed for the specific duration.

In addition to SW1 interrupt, you will need to handle the RxRDY (FFULL) interrupts of the communication channel(s). For each UART, RxRDY and TxRDY interrupts come in as a single interrupt source. Thus a single ISR should check the interrupt types and then carry out proper operations. There is no restriction about which the interrupt level and priority are assigned to these interrupts. Since a microprocessor may have a set of similar devices, a driver should be designed to work for all similar devices.

Extra Grade

You can earn extra points if you can add a print_seq function. When SW2 is pressed, the function is called and it initializes the printing operation of the sequences under display. The binary data of the sequences must be converted to ASCII characters and transmitted to the COM1 port of the host PC. Again, the UART transmission operations should be interrupt-driven (on TXRDY interrupts). While printing,

- The LED display is running constantly.
- An additional download can take place. When it is done, the display should switch to the new sequences.
- Additional SW2 interrupts can be ignored.