EXPERIMENT NUMBER 7
Bi-directional Serial Communication Using Interrupts

INTRODUCTION:

Communication with external devices over distances of more than a few feet is often done in a serial manner – one bit at a time over one or two lines (one line for receiving information and one for sending). The 8051 family of microcontrollers typically include a Universal Asynchronous Receiver Transmitter (UART) for this purpose. This is the same type of device used to implement the COM ports of a PC. Although it is possible to use simple software polling routines to service a UART, it wastes CPU time and prevents the CPU from performing other tasks, so interrupts are more commonly used. In this lab, you will design a program to send and receive serial data using interrupts. A UART device driver can be difficult to debug using standard software-simulation tools. In such cases, it is common to develop models that simulate external hardware and allow you to see how your hardware-software design will interact with this hardware. A model of an external serial-device will be provided which you can connect to the 8051 and can use to simulate and debug your design in hardware.

OBJECTIVES:

1. Develop a C-program that utilizes the 8051 UART to communicate with another serial devices.
2. Use interrupts to service serial-port requests to receive and transmit data.
3. Reinforce fundamentals of C-programming and hardware-software co-design, such as design re-use and model-development for co-simulation.

MATERIALS REQUIRED:

1. Keil µVision 2
2. Windows-based computer with an unused parallel port
3. Unix-based computer
4. Mentor Graphics software
5. Serial-port simulation module (http://www.ece.umr.edu/courses/cpe214/dist/serial_unit.vhd)
6. Xess40 simulation board
7. Ftp program
8. Multi-sync VGA display.
BACKGROUND:

A UART (Universal Asynchronous Receiver-Transmitter) provides all the features needed for simple serial communications. It is bi-directional in the sense that it can both send and receive data. Initializing the 8051 UART is easy: set a timer to program the baud-rate (the rate data is transmitted), enable the timer (turn it on), set the serial port mode, and then set the number of bits to be sent/received. Bytes may be sent or received through an internal register SBUF. When a byte is received, an internal flag, RI (Receive Interrupt), is set. To read the received byte, simply read SBUF. For example,

```c
If (RI==1){
    achar = SBUF;
}
```

reads the byte in SBUF into achar if a byte was received (RI==1). To transmit a byte, simply write it to SBUF. For example

```c
SBUF = achar;
```

writes the byte in variable achar out the serial port. The flag TI (Transmit interrupt) notifies the 8051 when a byte has finished being sent and gives the microcontroller an opportunity to send another byte. For example,

```c
if (TI==1){
    SBUF = msg[x];
    x++;
}
```

will send the xth byte of a character array msg if the TI flag is set — that is, if the UART says it is time to send another byte.

The simplest way to send and receive data is to constantly monitor the flags RI and TI from within your program, but then your program would never have time to do anything else. A better way is to use a serial-port interrupt. The 8051 is set up so that an interrupt can be generated anytime RI or TI is set — anytime that the UART needs attention. Because the interrupt will be generated on either RI or TI, the function that responds to the interrupt must poll RI and TI to see which of them caused the interrupt.

An example program using the 8051 serial-port UART is given below\(^1\). This program uses an interrupt to send and receive bytes through the serial port. In this case, the program acts like a short delay line, re-transmitting each byte that it receives. The interrupt and the main function communicate with one another using global variables newin, gone, incoming, and outgoing. When a byte is received, the interrupt reads the byte and sets newin. When the main function sees newin is set, it can read the received byte from global variable incoming. When the port finishes transmitting a byte, TI is set, generating an interrupt. The interrupt writes the byte in global variable outgoing to SBUF, thus sending it out the serial port, and sets the bit gone. When the main program sees that gone is set, it loads outgoing with the next byte to be transmitted. Notice that the interrupt service routine, serint, must poll RI and TI to see which has been set.

\(^1\) A similar program is also shown on pages 305-6 of *C and the 8051* by Shultz
/* Serial Port Usage Example */

#include <reg51.h>
define uchar unsigned char

uchar incoming,outgoing,lastinchar; /* received/transmited messages */
bit newin,gone; /* These are global flags */

/* Serial port interrupt-service routine. Interrupt 4 is generated
by RI or TI, the serial-port interrupt */
void serint(void) interrupt 4 using 1{
  if(RI){ /* Has a byte been received */
    incoming=SBUF; /* read the received byte */
    RI=0; /* reset the interrupt flag */
    newin=1; /* notify the main function */
  }else if (TI){ /* have we just finished sending a byte */
    SBUF=outgoing; /* send the next byte */
    TI=0; /* reset the interrupt flag */
    gone=1; /* notify the main function */
  }
}

/* The main function */
void main(void){

  /* INITIALIZE THE UART*/
  TMOD=0x20; /* use timer1, mode 2 */
  TH1=0xf1; /* 9600 baud with a 11.059mHz clock */
  TCON=0x40; /* start baud clock*/
  SCON=0x50; /* enable receive */
  IE=0x90; /* enable serial int */

  /* start by sending the character 'x' */
  lastinchar=outgoing='x';

  /* initialize the flags */
  gone=0;
  newin=0;

  /* Initiate the serial interrupt */
  TI=1; /* Since the interrupt occurs when TI=1,
       this command will cause an interrupt which
       will initiate execution of function
       serint */

  /* Main loop (do forever) */
  for(;;){
    /* if the last character is "gone", put the
       lastcharacter read into the output buffer, outgoing */
    if (gone){
      outgoing=lastinchar;
      gone=0; /* reset the flag */
    }
The serial stream for this setup consists of 10 bits of data. A start bit, ‘0’, will precede the actual 8 bits of data. A stop bit, ‘1’, will end the data stream. The data is written out in reverse order starting with the least significant bit and ending with the most significant bit. The following diagram from the Philips 8051 Family Hardware Guide shows example serial transmit and receive timings.

Text data is typically sent through the serial port in ASCII form. ASCII is simply a numeric representation of each character typically shown on an alpha-numeric display. An example of the first 128 ASCII characters is shown in Table 1. For example, if you were to send the message “Hey” through the serial port, you would see the numbers 72, then 101, then 121 (along with the start and stop bits) in the serial bit-stream.

For this lab, you will develop software to send and receive a message through the serial port interface. Received messages will be displayed on the VGA display using hardware and software developed in lab 6. Your software will be simulated in Keil µVision 2 and then in Mentor Graphics with a hardware module that simulates serial communication. This module has been developed in VHDL. After you have verified your design through simulation, you will download it to the Xess40 board. The serial port on your board will be connected to your neighbor’s board and you will send messages to one another and display that message on the VGA display to verify that your designs work.
Table 1. Printable ASCII characters. Table shows the character, followed by the numerical ASCII representation.

<table>
<thead>
<tr>
<th>Char</th>
<th>ASCII</th>
<th>Char</th>
<th>ASCII</th>
<th>Char</th>
<th>ASCII</th>
<th>Char</th>
<th>ASCII</th>
<th>Char</th>
<th>ASCII</th>
<th>Char</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>32</td>
<td>0</td>
<td>48</td>
<td>@</td>
<td>64</td>
<td>P</td>
<td>80</td>
<td>'</td>
<td>96</td>
<td>p</td>
<td>112</td>
</tr>
<tr>
<td>!</td>
<td>33</td>
<td>1</td>
<td>49</td>
<td>A</td>
<td>65</td>
<td>Q</td>
<td>81</td>
<td>a</td>
<td>97</td>
<td>q</td>
<td>113</td>
</tr>
<tr>
<td>&quot;</td>
<td>34</td>
<td>2</td>
<td>50</td>
<td>B</td>
<td>66</td>
<td>R</td>
<td>82</td>
<td>b</td>
<td>98</td>
<td>r</td>
<td>114</td>
</tr>
<tr>
<td>#</td>
<td>35</td>
<td>3</td>
<td>51</td>
<td>C</td>
<td>67</td>
<td>S</td>
<td>83</td>
<td>c</td>
<td>99</td>
<td>s</td>
<td>115</td>
</tr>
<tr>
<td>$</td>
<td>36</td>
<td>4</td>
<td>52</td>
<td>D</td>
<td>68</td>
<td>T</td>
<td>84</td>
<td>d</td>
<td>100</td>
<td>t</td>
<td>116</td>
</tr>
<tr>
<td>%</td>
<td>37</td>
<td>5</td>
<td>53</td>
<td>E</td>
<td>69</td>
<td>U</td>
<td>85</td>
<td>e</td>
<td>101</td>
<td>u</td>
<td>117</td>
</tr>
<tr>
<td>&amp;</td>
<td>38</td>
<td>6</td>
<td>54</td>
<td>F</td>
<td>70</td>
<td>V</td>
<td>86</td>
<td>f</td>
<td>102</td>
<td>v</td>
<td>118</td>
</tr>
<tr>
<td>(</td>
<td>39</td>
<td>7</td>
<td>55</td>
<td>G</td>
<td>71</td>
<td>W</td>
<td>87</td>
<td>g</td>
<td>103</td>
<td>w</td>
<td>119</td>
</tr>
<tr>
<td>)</td>
<td>40</td>
<td>8</td>
<td>56</td>
<td>H</td>
<td>72</td>
<td>X</td>
<td>88</td>
<td>h</td>
<td>104</td>
<td>x</td>
<td>120</td>
</tr>
<tr>
<td>*</td>
<td>41</td>
<td>9</td>
<td>57</td>
<td>I</td>
<td>73</td>
<td>Y</td>
<td>89</td>
<td>i</td>
<td>105</td>
<td>y</td>
<td>121</td>
</tr>
<tr>
<td>+</td>
<td>42</td>
<td>;</td>
<td>59</td>
<td>K</td>
<td>75</td>
<td>[</td>
<td>91</td>
<td>k</td>
<td>107</td>
<td>{</td>
<td>123</td>
</tr>
<tr>
<td>,</td>
<td>43</td>
<td>&lt;</td>
<td>60</td>
<td>L</td>
<td>76</td>
<td>\</td>
<td>92</td>
<td>l</td>
<td>108</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>44</td>
<td>=</td>
<td>61</td>
<td>M</td>
<td>77</td>
<td>]</td>
<td>93</td>
<td>m</td>
<td>109</td>
<td>}</td>
<td>125</td>
</tr>
<tr>
<td>.</td>
<td>45</td>
<td>&gt;</td>
<td>62</td>
<td>N</td>
<td>78</td>
<td>^</td>
<td>94</td>
<td>n</td>
<td>110</td>
<td>~</td>
<td>126</td>
</tr>
<tr>
<td>/</td>
<td>47</td>
<td>?</td>
<td>63</td>
<td>O</td>
<td>79</td>
<td>_</td>
<td>95</td>
<td>o</td>
<td>111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PRELIMINARY:**

Modify the above program to send and receive a 10-byte message and display the received message on a VGA display. This program is already very close to what you need. Give your program the following characteristics:

- Send a 10-byte message, declared as a constant in your code, out the serial port. Send this message over-and-over at some regular interval (e.g. Every 5 seconds, send the message “BLAH”). You may choose what message to send – you will be sending this message to another team later.
- Allow your program to receive a 10-byte message at any time (while simultaneously sending a message). Write this message to the VGA display. In addition, write each byte as it is received to port P1 bits 0 through 6 (this will allow you to debug your code in Mentor Graphics later). Be sure that you always write ‘1’ to P1.7. This is VERY IMPORTANT since you can cause contention between the FPGA and the 8031 if you write a ‘0’ to P1.7.
- Service the serial-port using interrupts
- Send data at 9600 baud, 8-data bits, no parity. This is the set-up used in the program above. Be sure to set the serial port up for a 12 MHz clock frequency because that is what the Xess40 board uses. If you cannot get exactly 9600 bits per second, try to make your baud rate as close as possible to 9600. Show your calculations.

Compile your code and simulate in Keil µVision 2. Serial port communication can be simulated, to an extent, using the serial window. It is important to simulate your software thoroughly to be sure you will have enough time to finish the rest of the lab in class. Turn in a hardcopy of your code at the beginning of class and have a hex file ready to use in this experiment.
PROCEDURE:

Summary: Download and compile the VHDL serial-port model. Use DA to add the serial-port model to the XS40 board schematic. Simulate your software with the serial-port model using QuickSim Pro to verify correct operation. Rewrite your code if necessary. Once correct, verify the final design in hardware by connecting the serial port from your XS40 board to the serial port of your neighbor. Your VGA display should show your neighbor’s message and vice-versa.

1. Development of Serial Module

A VHDL hardware model to test serial communications was developed by another engineer in your company, but they did not leave it in a form you can use in QuickSim Pro. The first step in this laboratory is to create a symbol for the VHDL model that can be used within Design Architect and can be simulated with QuickSim Pro.
A. Download the serial unit VHDL model from:
   http://www.ece.umr.edu/courses/cpe214/dist/serial_unit.vhd.
B. Build the serial unit model using the commands:
   swd
   sul
   vcom -93 -qspro_syminfo -source -explicit serial_unit.vhd
   vmap work work
C. Start DA. Generate a symbol by selecting the menu item File->Generate->Symbol. Click the Entity button. There are three things that need to be setup to instruct Mentor what to use for the symbol. Under the Entity button, choose “work” as the Library, “serial_driver” as the Entity, and the only choice under Arch (“["behave"]”, "HDL arch", ["hdl"])) should be used. Click Ok.
D. Check and save your symbol like you would any other symbol.

2. Hardware-Software Co-Simulation

At this point, you should be reasonably certain your code works because of simulation in µVision 2. However, µVision 2 is unable to simulate your hardware and software together. Hardware-software co-simulation will be done using QuickSim Pro as in the previous labs. To simulate the XS40 board being connected to another XS40 board serial port, the VHDL serial module will be connected to the serial port of the XS40 board. This module sends serial information from an ASCII text file to your 8051 and also can receive and display the data you transmit. In this way, you can see if you correctly receive and transmit information.
A. Create an ASCII text file containing a message of your choice. Name this file “testing.txt” and place it in your main design directory.
B. Open the Xess40_schematic sheet.
C. Place the serial unit symbol you just created on the sheet. Connect the 8051 to the serial module, connecting Rx on the 8051 to Tx on the module, and vice-versa. Remember the Rx and Tx are sent through P3.0 and P3.1, respectively, on the 8051. Check and save the sheet.
D. Run the QuickSim Pro using the command as in previous labs. Set QuickSim pro to trace waveforms on P3.0, P3.1, P1, the data to the VGA unit, and any other signals you feel are appropriate. Assuming that transmission will take place at 9600 baud, calculate the
approximate time needed to send your message via the serial port and write this time in your labbook. Run the simulation for this length of time, plus some additional time taken for initialization (i.e. clearing the VGA monitor and setting up the UART). The message in file testing.txt will be sent to the 8051 through the serial port. If your design is working properly, you will see the ASCII characters sent to the 8051 on the output port P1. Print out the waveform on P3.0, P3.1, and P1 and include them in your lab notebook. Verify that the message being sent to the 8031 is the message you created above. If it is not, there is likely an error in your program. Check the signal on RxD (P3.0). Is the baud rate correct? Does the number of bits you receive for each character correspond to the number you expected? Do the characters mirrored at P1 correspond to what is seen on RxD?

E. At the same time that messages are being received by the 8031, your program should be sending up to a 10-byte serial message to the serial-module. This message will be shown in a window under Mentor Graphics. Is your message coming through correctly?

F. [optional] Some variation in baud-rate is acceptable between communicating devices. Use the serial simulation module to test how much variation the 8031 will accept. Start by changing the baud-rate of the serial-module by plus-or-minus 10%, say to 10560 or 8640 bits-per-second. You can use Shift-F7 to change the value of the baud rate to any number. Does your program still work? Find the range of baud-rates (minimum and maximum baud) that the serial-module can be set to and still have a working serial communication link.

3. Hardware Verification

A. Get with another team and hook your XS40 boards together as shown below. Connect a VGA display to each board.

B. Download your hex and bit files to your board. Use the bit file created in lab 6. Once your neighbors have downloaded their files to their board, reset your boards by toggling D0 of the parallel interface using GXSPORT. If your design works properly, you should see the message your neighbor is sending you on your VGA display. You may wish to view P1 on the oscilloscope to check that data is being received correctly. If you have problems, you may want to recheck your simulation. When finished, show your lab instructor your working design.

C. View the serial data on TxD using the oscilloscope. Capture a portion of the transmitted
serial data and print it out for your lab notebook. Illustrate in your notebook that this data is the same as the serial data obtained from Mentor.

**QUESTIONS:**

1. Why is the oscillator value 11.059 Mhz often used in designs?
2. If characters arrive at a steady rate of 9600 bits per second and are separated by a single start and stop bit, what is the rate at which RI interrupts occur (in Hz)?
3. How fast must an 8051’s program respond to a TI interrupt if a maximum data rate is to be sustained at 9600 baud (in µs)?
4. On the timing diagram printed from your Mentor Graphics simulation, illustrate the start and stop bits for a single character, show the bits received for that character, and show that this character corresponds to the value written out to P1.
5. Using the timing diagram from the oscilloscope, verify that the transmission rate was 9600 baud. Show how you got your information and how you calculated your result.