GOALS:

The purpose of this lab is to introduce the concept of serial communications. In this laboratory exercise, sequential and combinatorial logic will be more fully explored. Specifically, combinatorial logic will be applied to a serial communication system demonstration.

BACKGROUND:

Serial communication is used widely throughout the electronics industry. Many peripheral devices, such as the oscilloscopes in the lab, now provide a means to communicate data to other devices, such as a computer. Sometimes, an error in serial communications may occur for a fraction of a moment. This could cause the received data to differ from the sent data.

In previous labs, we have seen an example of a combinational logic circuit made from basic logic gates (a full adder). We have also seen an example of a sequential logic circuit made from basic logic gates and flip flops (a counter). Now, we will use chips that have complete functions. That is, functions do not need to be made from the basics. With functions on chips, the focus will now be to explore some applications of digital logic, starting with serial communication.

In this lab, we will use a shift register (such as 74299), a comparator (such as 7485), and a register (such as 74185).

COMPARATOR

A comparator takes two binary numbers as input and provides an output based on the comparison.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A&lt;B</th>
<th>A=B</th>
<th>A&gt;B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

![Figure 1. Comparator – truth table and pinout of 7485](image)
**N-BIT REGISTER**

An N-bit register is an N-dimensional equivalent of a D flip-flop. At the edge of a clock signal, \( Q = D \) for each individual bit.

![N-bit register 74175](image)

Figure 2. N-bit register 74175

**SHIFT REGISTER**

A shift register is a special type of register in which the bits in its output array can be shifted left or right. A shift register can convert data from serial to parallel. Look up a datasheet for the 74299 8-bit shift register. Depending on two select inputs, it has four modes of operation; (i) store, (ii) shift right (From MSB to LSB), (iii) shift left (From LSB to MSB), and (iv) load. In this lab, the 74299 will be used in the shift right configuration. Determine what signals the select statements must be for shift right mode. The output from MSB to LBS is \( Q_A Q_B Q_C Q_D Q_E Q_F Q_G Q_H / Q_0 Q_1 Q_2 Q_3 Q_4 Q_5 Q_6 Q_7 \).

![Shift register 74299](image)

Figure 3. Shift register 74299

In the following simulation, the 74299 is in shift right mode. The serial input is applied to the SR (shift right input) pin. At every positive edge of the clock, the value of the SR input at that instant is shifted into the MSB of the output, \( Q_A/Q_0 \). The previous value of \( Q_A/Q_0 \) is shifted into \( Q_B/Q_1 \) and so on. After 8 clock cycles the serial input 10111001 is converted to a parallel output of \( Q_A Q_B Q_C Q_D Q_E Q_F Q_G Q_H = 10011101 \). Notice that the CLRN pin must be logic high for the outputs to be active. When CLRN is logic low, the outputs reset to zero, and the input is ignored.
**SERIAL COMMUNICATION**

In serial communication, two digital circuits will send and receive information one bit at a time.

![Figure 5. Serial communication](image)

To be able to implement this idea in reality, we will use a transmission line “TX” for the value of the bit being sent, and a clock line “CLK” to know when to copy the bit value. In this description, the “TX” and “CLK” will be controlled directly by the user. We will use a receiver composed of an 8-bit shift register, a 4-bit comparator, and a 4-bit register.

For the transmission circuit in this lab, a data line and a clock line was used. This is similar to the I2C mode where there are two lines: the SDL line (for data) and the SCL line (for the clock). With the clock line, the external circuit is able to synchronize and also because the clock can be applied only when a transmission is imminent. No clock – No transmission.

In this type of circuit, the transmission of data may be an occasional occurrence. Then the serial line may not be utilized often. In some designs it may be desired to talk to one of many external circuits or peripherals. Then there are two choices: build two or more circuits or multitask the single serial communications line. When there are two peripherals on a line, the problem exists of how to talk to a particular device. In I2C circuits, this is accomplished by giving each device a unique address. The first portion of a transmission is that address. The particular peripheral is activated, and the transmission occurs.
Figure 6. Receiver configuration

Figure 7. Serial communication organization – block diagram
The receiver will take the bits that are transmitted one at a time and convert them to parallel. The 4 highest bits are compared to a hardwire value that acts as the address (like a name) of the receiver. The 4 lowest bits act as data that is being transmitted. When the 4 highest bits become equal to the address of the receiver, then the output of the comparator goes from 0 to 1. This is a positive clock edge to the 4-bit register. When that occurs, the register copies the data bits. The receiver schematic is shown in more detail below.
On the DE2 board, make the following Pin Assignments:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_clock</td>
<td>KEY3</td>
</tr>
<tr>
<td>my_serial_input</td>
<td>SW[0]</td>
</tr>
<tr>
<td>my_clear</td>
<td>SW[17]</td>
</tr>
<tr>
<td>B7-B0</td>
<td>LEDG[7]-LEDG[0]</td>
</tr>
<tr>
<td>data4-data1</td>
<td>LEDR[3]-LEDR[0]</td>
</tr>
<tr>
<td>address_match1</td>
<td>LEDG[8]</td>
</tr>
</tbody>
</table>

When on the DE2 board, use SW[0] to determine if you want to transmit a 1 or 0. Press KEY3 when you want to send it. LEDG[7]-LEDG[0] will show the parallel output of the bits you have transmitted. When LEDG[7]-LEDG[4] match “1011”, then LEDG[8] will turn on, indicating an address match. Additionally, the receiver will copy the 4 data bits that were sent on LEDG[3]-LEDG[0] to LEDR[3]-LEDR[0].
**Parity Check**

How can we determine if a transmission error has occurred? There are several ways, but one method that has applications beyond serial communications is the use of a parity bit. Adding a parity bit with data, we will have some minor indication that an error has occurred. That is, if the parity check is incorrect, then we know that an error has happened, but if no parity error is indicated, then it is likely that no error has occurred, but there is no guarantee. In fact, if two transmission errors occur, the parity check would be correct but the data incorrect.

One problem with the parity bit addition is that the error detection forms a separate line. That is the parity output line. The reason to use a serial input is to reduce the number of lines in a circuit. Verify the circuit below to see that it will incorporate the parity bit into the bit-stream, making this a 5 bit serial word. In other words, attach this output into the circuit.

![Figure 12. Serial parity](image)

This is a viable solution, although it is unwieldy. The process for transmission of a word requires:

1. Activation of the clear line.
2. Manipulating the Load High line IF NECESSARY to indicate the D3 bit
3. Loading the D0 – D2 lines. After this, the load line must be activated so that the original 4-bit word now appears for the serial stream.
4. After the 4-bit word has been sent, then the load line must again be activated for the parity bit.
5. Once the parity bit has been sent, the process starts all over again for the next 4-bit word.

It would be a simpler process if the two lines (serial out, and parity) could be tied together, and the proper output chosen in its own time. One-way of accomplishing this is with the selector circuit.
Here, rather than taking the steps of manipulating the set and clear lines of the J-K flip-flop to get the D3 line loaded correctly, one can attach the two inputs to the A and B lines and use another line (select) to activate the output of the J-K parity generator on bit five. Still there are a lot of components for a simple process. One of the more innovative concepts in digital electronics is that of the tristate output. Recall that digital electronics will have either a 1 or 0. This is a bi-state. For various reasons, that will be obvious soon, there is a need for a third state. Another and more correct way of describing this is to recall that two outputs should not be tied together. It is possible to damage one of the devices, and the output cannot be guaranteed to represent either output. A third state was developed that will effectively shut off a component's output from the output of other components. In this manner, a number of outputs could be put onto the same output line. The final output line would represent the value of whatever device is selected.

Parity Check in Quartus

In this experiment, only use the first 7 bits as data bits, and the last bit as a parity bit (no address bits). A parity bit is a primitive way to check for errors. One common parity, even parity, is a “1” if there is an odd number of 1’s in the data bits, and “0” if there is an even number of 1’s in the data bits. To obtain a “1” if there is an odd number of 1’s in the 7 data bits, you can use the XOR function between all data bits. A simple Verilog file can do this for you as shown below. Open a new Verilog file in the same project, and type out the five lines.

```
module even_parity_checker(b7, b6, b5, b4, b3, b2, b1, even_parity);
    input b7, b6, b5, b4, b3, b2, b1;
    output even_parity;
    assign even_parity = ~b7 ^ b6 ^ b5 ^ b4 ^ b3 ^ b2 ^ b1;
endmodule
```

Figure 14. Verilog code for parity
Create symbol file to be able to use a block symbol (see lab #4 for the instruction on creating symbols).

![Symbol for parity check](image1)

**Figure 15. Symbol for parity check**

Use the symbol file to perform parity checking on the data bits, and compare your obtained parity bit to the transmitted parity bit.

![Schematic for parity check](image2)

**Figure 16. Schematic for parity check**

See the next page for lab deliveries.
LAB DELIVERIES:

PRELAB:

For the prelab, simulate the 74299 8-bit shift register in shift right mode. Demonstrate that you can obtain a parallel output of \(Q_AQ_BQ_CQ_DQ_EQ_FQ_H=11010101\). Include the schematic and simulation waveforms in the report document.

PRELAB DELIVERIES

Include in the report document:
1. Schematic from Quartus
2. Simulation waveforms

LAB EXPERIMENTS:

1. Experiment 1: 4-bit serial transmission (single receiver)
   1. Implement schematic from Figure 8 in Quartus
   2. Perform the timing simulation
   3. Perform the functional simulation
   4. Implement your single-receiver design on the DE2 board
   5. Demonstrate the operation to the TA.

6. Experiment 2: 4-bit serial transmission (multiple receivers)
   1. Modify schematic from Figure 8 to design the serial transmission with multiple receivers, having multiple addresses (two receivers)
   2. Implement your single-receiver design on the DE2 board
   3. Demonstrate the operation to the TA.

4. Experiment 3: 4-bit serial transmission with parity bit (single receiver) (EXTRA-CREDIT)
   1. Implement your single-receiver design on the DE2 board
      a. Make address size 3 bit
      b. Make data size 4 bit
      c. Make last single bit the parity bit
   2. Demonstrate the operation to the TA.
Postlab Report:

Include the following elements in the report document:

<table>
<thead>
<tr>
<th>Section</th>
<th>Element</th>
</tr>
</thead>
</table>
| 1       | Theory of operation  
          *Include a brief description of every element and phenomenon that appears during the experiments.* |
| 2       | Prelab report |
| 3       | Results of the experiments |
|         | **Experiment** | **Experiment Results** |
| 3       | 1 | a. Screenshot of schematic  
      |   | b. Functional simulation  
      |   | c. Picture of DE2 board operation |
| 3       | 2 | a. Screenshot of schematic  
      |   | b. Picture of DE2 board operation |
| 3       | 3 | a. Screenshot of schematic  
      |   | b. Picture of DE2 board operation |
| 4       | Answer the questions |
| Question no. | Question |
| 1       | 1 | What are main reasons to use the serial transmission, instead of parallel transmission? |
| 2       | 2 | List 4 examples of serial transmission standards |
| 3       | 3 | What is the minimal set of transmission lines, between two units? |
| 5       | Conclusions  
          *Write down your conclusions, things learned, problems encountered during the lab and how they were solved, etc.* |
| 6       | Attachments  
          *Zip your projects. Send through WebCampus as attachments, or provide link to the zip file on Google Drive / Dropbox, etc.* |
|           | List of attachments to deliver: |
|           | 1. Single-receiver transmission – Quartus project |
|           | 2. Multiple-receiver transmission – Quartus project |
|           | 3. Single-receiver transmission with parity bit – Quartus project (extra credit) |

References:

1. DE2 pin table: [http://faculty.unlv.edu/eelabs/docs/guides/DE2_Pin_Table.pdf](http://faculty.unlv.edu/eelabs/docs/guides/DE2_Pin_Table.pdf)
2. Altera DE2 Introduction  
   [http://faculty.unlv.edu/eelabs/docs/guides/DE2_Introduction_box.pdf](http://faculty.unlv.edu/eelabs/docs/guides/DE2_Introduction_box.pdf)
3. Getting started with Altera DE2 Board:  
   [http://faculty.unlv.edu/eelabs/docs/guides/DE2_tut_initialDE2.pdf](http://faculty.unlv.edu/eelabs/docs/guides/DE2_tut_initialDE2.pdf)
4. Altera: Quartus II Introduction Using Schematic Design  
   [http://faculty.unlv.edu/eelabs/docs/guides/DE2_tut_quartus_intro_schem.pdf](http://faculty.unlv.edu/eelabs/docs/guides/DE2_tut_quartus_intro_schem.pdf)
6. Datasheets of 7400 series chips:  