Term Project

Due:  
- ASM Chart (hardcopy) 23/12/2013, 09:40 (9:40 a.m.)
- Simulation Demo 02-03/01/2014 (will be scheduled)
- Project Report 08/01/2014, 23:55 (11:55 p.m.)
- Implementation Demo 09-10/01/2014 (will be scheduled)

A Simple Telephone Conversation

You will design a simple telephone conversation and implement it on the FPGA device. You will be asked to demonstrate your implementation on the BASYS FPGA board.

In your design, there will be two entities: the caller and the callee. Basically, the caller will initiate a telephone conversation with the callee and they will talk with each other by exchanging letters. When the call ends, your circuitry will calculate the cost of this call and display this value through the SSDs (Seven Segment Displayers). There will be two regions (e.g., Istanbul and Ankara), and the cost depends on whether the call is within the same region or across the regions.

Inputs

There will be 6 (six) inputs in your circuitry:

- **reset** will set your circuitry to its initial state. This input should be provided from BTN3.

- **callerEnter** will be used by the caller and it will represent the caller pressing an enter button. It will be used to initiate a call, to end a call, to enter the areaCode of the caller/callee, to send letters, etc. This input should be provided from BTN0.

- **calleeEnter** will be used by the callee and it will represent the callee pressing an enter button. Its functionality will be the same as that of callerEnter. This input should be provided from BTN1.

- **letter** will be used to define the character to be transferred between the caller and the callee. There will be 8 (eight) different such characters; hence, this input will be a 3 (three) bit number. These characters and their corresponding encodings are given in Table I. This input should be provided from SW[2:0], where SW0 will be the least significant bit of its value.

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>-</td>
<td>100</td>
<td>H</td>
</tr>
<tr>
<td>001</td>
<td>A</td>
<td>101</td>
<td>L</td>
</tr>
<tr>
<td>010</td>
<td>C</td>
<td>110</td>
<td>S</td>
</tr>
<tr>
<td>011</td>
<td>E</td>
<td>111</td>
<td>U</td>
</tr>
</tbody>
</table>

Table I: Code-Letter Pairs
• *areaCode* will be used to represent the caller/callee being from either Istanbul or Ankara. It will be a 1-bit number. The value of the *areaCode* will be 0 (zero) for Istanbul and it will be 1 (one) for Ankara. This input should be provided from SW3.

• *startEnd* will be used to decide on whether to start or end a telephone call. This input will also be a 1 (one) bit number. The value of the *startEnd* will be 0 (zero) to start a telephone call and it will be 1 (one) to end a telephone call. This input should be provided from SW4.

**Operation Steps**

There are 5 (five) different modes in which your design operates: *One-Sided Telephone Conversation*, *Two-Sided Telephone Conversation*, *Visualized Calling Process*, *Timeout* and *Recall*. Each of these modes is described below in detail and each has its own grade percentage. You should note that these modes are incremental. You should design your circuit in the order that they are presented. For instance, you **cannot** implement *Visualized Calling Process* unless you implement *Two-Sided Telephone Conversation*. Likewise, you **cannot** implement *Recall* unless you implement *Timeout*.

**One-Sided Telephone Conversation (75%)**

The circuitry will start in an IDLE state. The caller will initiate a call by setting the value of the *startEnd* input to 0 (zero) and pressing the *callerEnter* input. Then, the **caller** will define *his/her area code* by setting the value of the *areaCode* input to either 0 (zero) or 1 (one) and pressing the *callerEnter* input. Thereafter, the **caller** will define the *area code of the callee* by setting the value of the *areaCode* input to either 0 (zero) or 1 (one) and pressing the *callerEnter* input. Both of these area codes should appear on the SSD output during the selection process. The caller will **not** enter any telephone number information. At this point, the telephone of the callee will start ringing. The callee can either answer or reject the call. If the callee wants to answer the call, then (s)he will set the value of the *startEnd* input to 0 (zero) and press the *calleeEnter* input. At this point, the conversation starts and the caller (only the caller) can send characters to the callee. In order for the caller to send a character to the callee, (s)he will first set the value of the *letter* input and then (s)he will press the *callerEnter* input. During this process, the selected characters should appear on the SSD output. The caller can continue to transmit characters to the callee until either of the entities ends the conversation (i.e., hangs up the phone). Your circuitry should count the number of characters transferred so that it can use this information to calculate the cost when the conversation ends. If the callee wants to reject the call, then (s)he will set the value of the *startEnd* input to 1 (one) and press the *calleeEnter* input. In this case, your circuitry will display “BUSY” on the SSDs for approximately 4 (four) seconds (3-5 seconds are also acceptable) and go back to its initial state.

Whenever the caller or the callee wants to end the conversation, (s)he can do so by setting the value of the *startEnd* input to 1 (one) and pressing the *callerEnter* or *calleeEnter* input (depending on the conversation ending entity). At this point, your circuitry should calculate the cost of the conversation. If both the caller and the callee are from the same city (i.e., if
both of the area codes are 0 (zero) or if both of them are 1 (one)), then each transferred character will cost 1 (one) TL. Otherwise, if the caller and the callee are from different cities, then each transferred character will cost 2 (two) TL. Then, your circuitry will show the price of the conversation on the SSDs. Thereafter, when the caller sets the value of the \textit{startEnd} input to 1 (one) and presses the \textit{callerEnter} input, your circuitry should go to the IDLE state.

**Two-Sided Telephone Conversation (10%)**

In this mode, both the caller and the callee can send characters to each other. The conversation will start with the caller sending characters to the callee. The caller will continue to send new characters until (s)he sends the special character ‘-‘. When the caller sends ‘-‘ to the callee, it will be the callee’s turn to send characters. Likewise, the callee can continuously send new characters to the caller until (s)he sends the special character ‘-‘. Whenever the turn changes, previously transmitted characters are erased from the SSD output.

As the caller sends characters to the callee, the SSD output values should shift from right to left; while as the callee sends characters to the caller, the SSD output values should shift from left to right.

**Visualized Calling Process (5%)**

In this mode of the circuitry design, the LEDs will simulate the phone ringing process. They will illuminate one after another, from left to right, until the callee either answers or rejects the call. In order for the illuminated LED to be seen clearly, your circuit should wait approximately 2 (two) seconds (1-3 seconds are also acceptable) at each illuminated LED.

**Timeout (5%)**

In this mode of the design, the call will drop if there is no response from the callee after approximately 8 (eight) seconds (6-10 seconds are also acceptable). If the callee does not either answer or reject the call, then your circuitry will display “nAnS” on the SSDs for approximately 4 (four) seconds (3-5 seconds are also acceptable) and then go back to its initial state.

**Recall: caller can repeat his/her call (5%)**

In this mode of the design, the caller can repeat his/her call when either the callee rejects the call or the call drops due to no answer. If the caller wants to recall the callee, then (s)he will set the value of the \textit{startEnd} input to 0 (zero) and press the \textit{callerEnter} input. At this point, your circuitry will use the previously entered area codes. In other words, the caller will not re-enter the required area codes. Besides, these area code values will also be shown on the SSDs, again. Please note that the recall operation should take place before the circuitry gets back to its initial state.
Outputs

There will be 2 (two) different types of outputs in your circuitry: LED outputs and SSD outputs.

LED outputs

- In all of the modes, LD[7:4] are used to show the current state of the circuit
- When the caller calls the callee in Visualized Calling Process mode
  - LD[3:0]: illuminated one after another, from left to right, starting with LD2
- During data transmission in Two-Sided Telephone Conversation
  - LD1 is illuminated when the callee sends characters to the caller
  - LD0 is illuminated when the caller sends characters to the callee

7-segment displays (SSDs)

- When the caller initiates a call by pressing the callerEnter input
  - AN3: The value to be selected for the area code of the caller (0 or 1)
- When the caller finishes entering his/her own area code
  - AN3: areaCode of the caller
  - AN2: ‘-’
  - AN1: The value to be selected for the area code of the callee (0 or 1)
- When the caller finishes entering the area code of the callee
  - AN3: areaCode of the caller
  - AN2: ‘-’
  - AN1: areaCode of the callee
- When the callee rejects the call
  - AN3: ‘B’
  - AN2: ‘U’
  - AN1: ‘S’
  - AN0: ‘Y’
- As the caller enters the characters
  - AN0: current character entered by the caller
  - AN1, AN2 and AN3: previous three (3) characters entered by the caller
- When the caller/callee ends the conversation
  - AN3, AN2, AN1 and AN0: Price of the conversation in hexadecimal
    - You should use lower-case characters for 0xB and 0xD (i.e., b and d)
- When the callee enters the characters in Two-Sided Telephone Conversation mode
  - AN3: current character entered by the callee
  - AN2, AN1 and AN0: previous three (3) characters entered by the callee
• When the call drops in Timeout mode
  o AN3: ‘n’
  o AN2: ‘A’
  o AN1: ‘n’
  o AN0: ‘S’

• When the caller repeats his/her call in Recall mode
  o AN3: areaCode of the caller
  o AN2: ‘-’
  o AN1: areaCode of the callee

Bonus

The characters will be transferred among the caller and the callee in an encrypted form. You will use the Caesar Cipher, in which each letter in a plaintext is replaced by another letter that is at some fixed number of positions further down the alphabet. Your alphabet will be “ACEHLSU” and the fixed number will be 2 (two).

For instance, if the caller wants to send the character ‘A’ to the callee, then (s)he will set the value of the letter input to “001” and when (s)he presses the callerEnter input, ‘E’ will be seen on the relevant SSD. The reason behind this is that the character ‘E’ comes after 2 (two) places down from the character ‘A’ in our alphabet. As another example, if the callee wants to send the character ‘U’ to the caller, then (s)he will set the value of the letter input to “111” and when (s)he presses the calleeEnter input, ‘C’ will be seen on the relevant SSD.

The special character ‘-’ will not be encrypted. Hence, whenever the caller/callee enters this character, the other party will have the turn to send characters.

In order for the conversation to be encrypted, either the caller or the callee should press to BTN2. The conversation might start in plain and then caller or callee may decide to continue with encrypted character transmission, or they might start using encryption at the beginning by pressing the corresponding push button (BTN1 or BTN2). While the conversation is transmitted in an encrypted form, the caller or the callee can go back to transmitting the characters in plaintext by pressing BTN2 again.

This bonus will be worth 10 points. Please note that you can get this bonus if and only if your circuitry includes all of the modes that are described above.
Additional Work as a Makeup for a Missed Lab
(or poorly performed one)

If you missed a lab or received a poor grade from any, you can do this part as a makeup for the missed lab or improve your poor grade. This time, your design should allow the caller to enter a phone number after defining the area codes.

- Phone numbers are 4 (four) digit \textbf{decimal} numbers.
- Each digit should be seen on the SSDs (first digit at AN3 and last digit at AN0)
- The caller will select the phone number by using SW[7:4] and set it by setting the value of the \textit{startEnd} input to 0 (zero) and pressing the \textit{callerEnter} input.
- If the caller tries to set a non-decimal number for a specific digit, then ‘-‘ should be seen on the relevant SSD output. This value should be treated as an invalid input and your circuitry should wait for a valid input before inputting a new value to set the next digit.
- Before a digit is set, the relevant SSD output should flash with 1Hz frequency.
- The digits that are already set should be seen on the relevant SSD outputs.
- If your circuitry includes the \textit{Recall} mode, then you are not expected to show the phone number during the recall process.

It would be better for you to first implement the simplest version of the project and make sure that this version is working correctly before further implementing the bonus and/or the additional work.
module telephoneConversation (clock, reset, caller, callee, areaCode, startEnd, letter, display, leds);

    input clock, reset, caller, callee;
    input areaCode; // 0: Istanbul, 1: Ankara
    input startEnd; // 0: start, 1: end
    input [2:0] letter; // a character (from the code) of 3 bits
    output reg [27:0] display; // 4 displays of each 7 bits
    output reg [7:0] leds; // represents the correctness of the guess

    reg [3:0] current_state; // current state of the circuitry
    reg [3:0] next_state; // next state of the circuitry
    reg [14:0] characterCount; // number of characters transferred
    reg [12:0] transitionCounter; // counter to wait in the states
    ... // additional registers

    // sequential part – state transitions
    always @ (posedge clk or posedge rst)
    begin
        ... // your code goes here
    end

    // combinational part – next state definitions
    always @ (*)
    begin
        ... // your code goes here
    end

    // sequential part – control registers
    always @ (posedge clk or posedge rst)
    begin
        ... // your code goes here
    end

    // sequential part – outputs
    always @ (posedge clk or posedge rst)
    begin
        ... // your code goes here
    end

    ... // additional always statements
endmodule

Please note that this template is only an example. You can add more registers, change the bit-size of the given registers, initialize the values for the given registers, define parameters and extend the number of always statements as your design requires.
Details on Simulation Demo and Final Demo

During the simulation demos (see “Project Plan and Deadlines” section for the demos and their schedules), you are only expected to show the functionality (i.e., the correct transitions between the states) of your Finite State Machine (FSM). You do not need to add any other parts to your design during the simulation demo. Besides, the simulation demo should be performed on the simplest version of the final project description, i.e., it should NOT include the bonus and/or the additional work parts.

However, in the final implementation of your circuit, you will need some extra modules which we provide you under “Resources ➔ Term Project”. These are clock divider, debouncer and seven segment driver modules. There are comments in the modules about the units the modules implement. You can use these files just as verilog or you can create a symbol from the verilog files and add them to a schematic file; you are free to choose.

- Clock divider module generates a clock signal with a period of 50 ms, from a 25 MHz input clock.
- Debouncer circuit gets the input from a push button and generates a one clock pulse output.
- Seven segment driver module, drives the segments.

In your FSM design, think of your clock input period as 50 ms. Assume that the enter signals for the caller/callee are one clock pulse (debouncer circuit will provide that) and you can ignore the case of two inputs coming at the same time (e.g., two buttons are pushed at the same time).

In ssd.v, the given module uses the UN-divided clock, while in debouncer.v, the given module uses the divided clock. Besides, your FSM should also use the divided clock.

A demo schedule will be organized before the demo dates and will be announced through suCourse. If you want to demonstrate your design before the deadlines, you can do so by scheduling an appointment with your TA. Besides, if you want to check your work, you can come to the lab on the days that will be announced.

Please do not forget these facts:

- Each group will have 20 minutes to present their project and answer the questions we will be asking. Be prepared for any question we might ask about your project (design, coding, etc.). Every group member should be able to answer these questions.
- You must bring a hard copy (i.e. printed copy) of your latest ASM chart.
- Any group member not attending the demo without proper excuse (e.g. doctor’s report, existing final exam) will cause in the decrease of the grade of the whole group.
- For the final (implementation) demo, you must submit your work and your report through the assignment "Final Project" before its deadline. We will be checking the correctness of the submitted design.
- Each group must be present at the door of the lab at least 10 minutes prior to their scheduled time.
- No group should enter the door until they are invited by one of the TAs.
- If you are not able to demonstrate your work on time, you will not be given extra time.
Simulation Scenario

Following is the simulation scenario that you will perform your demo on. You **MUST** create a test-bench file that corresponds to the given scenario below and check the correctness of the results before coming to your demo. Otherwise, you will not have (or will **NOT** be given) time to complete the requirements.

1. If you implemented *One-Sided Telephone Conversation* mode
   
   a. Reset the phone
   b. Caller opens the phone (use BTN0 to go the relevant state)
   c. Caller set the caller’s area code as 1
   d. Caller sets the callee’s area code as 1
   e. Callee answers the phone
   f. Caller sends “ALE” message to callee
   g. Caller ends the call and the circuit shows the price
   h. Caller closes the phone (use BTN0 to go the the IDLE state)
   i. Caller opens the phone for a new call
   j. Caller sets the caller’s area code as 1
   k. Caller sets the callee’s area code as 0
   l. Callee rejects the call

2. If you implemented *Two-Sided Telephone Conversation* mode

   a. Reset the phone
   b. Caller opens the phone (use BTN0 to go the relevant state)
   c. Caller set the caller’s area code as 1
   d. Caller sets the callee’s area code as 1
   e. Callee answers the phone
   f. Caller sends “LES-” message to callee
   g. Callee sends “ULA-” message to caller
   h. Caller sends “AA-” message to callee
   i. Caller ends the call and the circuit shows the price
   j. Caller closes the phone (use BTN0 to go the the IDLE state)
   k. Caller opens the phone for a new call
   l. Caller sets the caller’s area code as 1
   m. Caller sets the callee’s area code as 0
   n. Callee rejects the call
3. If you implemented *Timeout* mode

   a. Reset the phone
   b. Caller opens the phone (use BTN0 to go the relevant state)
   c. Caller set the caller’s area code as 1
   d. Caller sets the callee’s area code as 1
   e. Callee answers the phone
   f. Caller sends “LES-” message to callee
   g. Callee sends “ULA-” message to caller
   h. Caller sends “AA-” message to callee
   i. Caller ends the call and the circuit shows the price
   j. Caller closes the phone (use BTN0 to go the the IDLE state)
   k. Caller opens the phone for a new call
   l. Caller sets the caller’s area code as 1
   m. Caller sets the callee’s area code as 0
   n. Callee rejects the call
   o. Caller opens the phone for a new call
   p. Caller sets the caller’s area code as 1
   q. Caller sets the callee’s area code as 0

4. If you implemented *Recall* mode

   a. Reset the phone
   b. Caller opens the phone (use BTN0 to go the relevant state)
   c. Caller sets the caller’s area code as 1
   d. Caller sets the callee’s area code as 0
   e. Callee rejects the call
   f. Caller recalls the callee and callee does not respond (timeout)
   g. Caller recalls the callee
   h. Callee answers the phone
   i. Caller sends “LES-” message to callee
   j. Callee sends “ULA-” message to caller
   k. Caller sends “AA-” message to callee
   l. Caller ends the call and the circuit shows the price
   m. Caller closes the phone (use BTN0 to go the the IDLE state)
   n. Caller opens the phone for a new call
Project Plan and Deadlines

You must follow a project plan and demonstrate that you meet a specific deadline by submitting your work. Each work item in the project plan will be graded separately. The project plan and grade of each work item is as follows:

1. **REQUIREMENTS:** Project requirements are given to the students.
   *Dec 16, 2013*

2. **ASM CHART:** ASM Chart is submitted in *hardcopy* to Duygu.
   (You can learn the comments made on your design and take your ASM charts back, during the office hour of Duygu, in the same day)
   *Dec 23, 2013  09:40*

3. **SIMULATION:** ISim simulation is demonstrated to the lab assistants.
   *Jan 02-03, 2014*

4. **REPORT:** A project report is submitted via *suCourse* as described in the document named *Submission Guidance for the Lab Assignments.pdf*.
   *Jan 08, 2014  23:55*

5. **DEMO:** The circuit is realized on FPGA and a demonstration must be made.
   *Jan 09-10, 2014*

**Note that these deadlines are hard, and there will be no additional time.**

Notes:

- You can work with your lab partner in this project.
- You can use HDL language. (Recommended)
- The first two groups that finish the project earliest will be awarded by five (5) bonus points in the final exam.

Some Tips: Before writing your project on verilog, you can try to write some small examples to warm up. This will help you to complete your task quicker and painless. Also there some sites that you can find some examples to study Verilog. One of them is: http://www.asic-world.com/verilog/veritut.html. These sites might help you to understand combinational, sequential circuits and state diagrams.