Multiple Connection Telephone System with Voice Messaging

Rumen Hristov, Alan Medina | 6.111 Project Proposal | Fall 2015

Introduction

We propose building a two-way telephone system. Nowadays, almost everybody have a mobile phone, but we can still utilize the convenience of simple landline phones. They are very convenient for communication from room to room in the same building. Every phone will be an FPGA with attached headphones. Our system will be able to record and received voice messages and call specific phones connected to our FPGA.

Project Overview

We will use several FPGAs, which will be connected with a wire to transmit data and simulate a phone call. One person will be able to dial a FPGA from the other and the other person can pick up and start a conversation. If the call does not succeed then the caller will hear the pre-recorded greeting message of the person who he is calling. After that he will have the option to leave a message. Later the other person can listen to this message and decide whether to save or delete it.

Our project closely follows the systems used in the traditional land line telephones and our design will face similar challenges. We will use several labkits, which will be connected with a single bidirectional data transmitting wire. This connection needs to tolerate noise, because otherwise the quality of the audio will be impacted in a negative way. The other challenging component of the project is the voice message recording. We need to store and read from the flash memory for the labkit and send the data that we read either to the headphones or to the other FPGA.

We don’t need any extra components for our project than the one that currently exist in the lab.

Functional Design

Before going into more technical details, we want to understand the capabilities of our phone and how it will transitions between the different tasks the phone needs to do. At a high level, our project can be easily explained with a state machine. The phone will change between different states, then each of the different states will enable the rest of the phone to perform the proper functions. Figure 1 shows how the states will transition in our main state machine. The phone will mostly be in IDLE, the default state, and will be able to transition to the other states, such as calling, receiving a call, or listening to saved messages.

Some other important parts of this diagram are MSG and PRE. MSG is the voice message that can be sent or received in the case that the phone is not answered. PRE is a pre-recorded greeting that will come from the memory. This is the message played to prompt a voice message (e.g. “Please leave a message after the beep”).

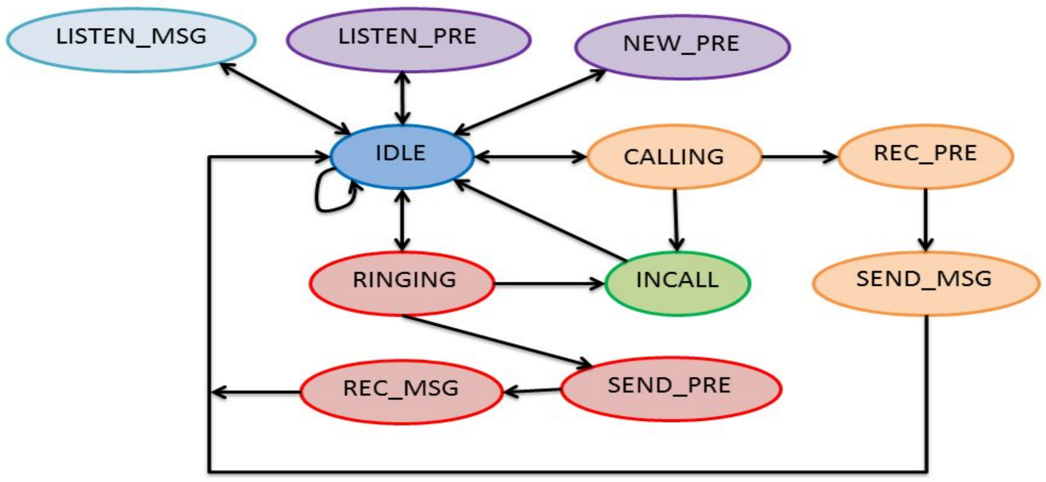

Figure 1: Finite State Machine for Phone System

Calling

When making a call, you would move down the orange path. If the person answers, the call begins. If the person doesn’t answer, you are led down a set of states that will allow you to listen to the other FPGA’s PRE and leave a MSG.

Ringing

When there is an incoming call, you can progress down the red path. First thing you can do is answer the call to begin your conversation. If you don’t answer, a timer will expire and your PRE is sent. You are then ready to receive a MSG that you can store into memory.

Listen to Messages

In the light blue state, you will be able to listen to messages that other people have left you while you were gone. There will be a flashing indicator light to let you know when you have a message waiting and you will be able to cycle through the stored messages.

Changing Pre-recorded Message

You will be able to control PRE in the purple states. You can listen to the current PRE that is saved in memory, or you can create your own personalized message and store it in memory.

Technical Design

The finite state machine is the main component of our design, but there are other technical challenges that we need to face in order to implement our system.

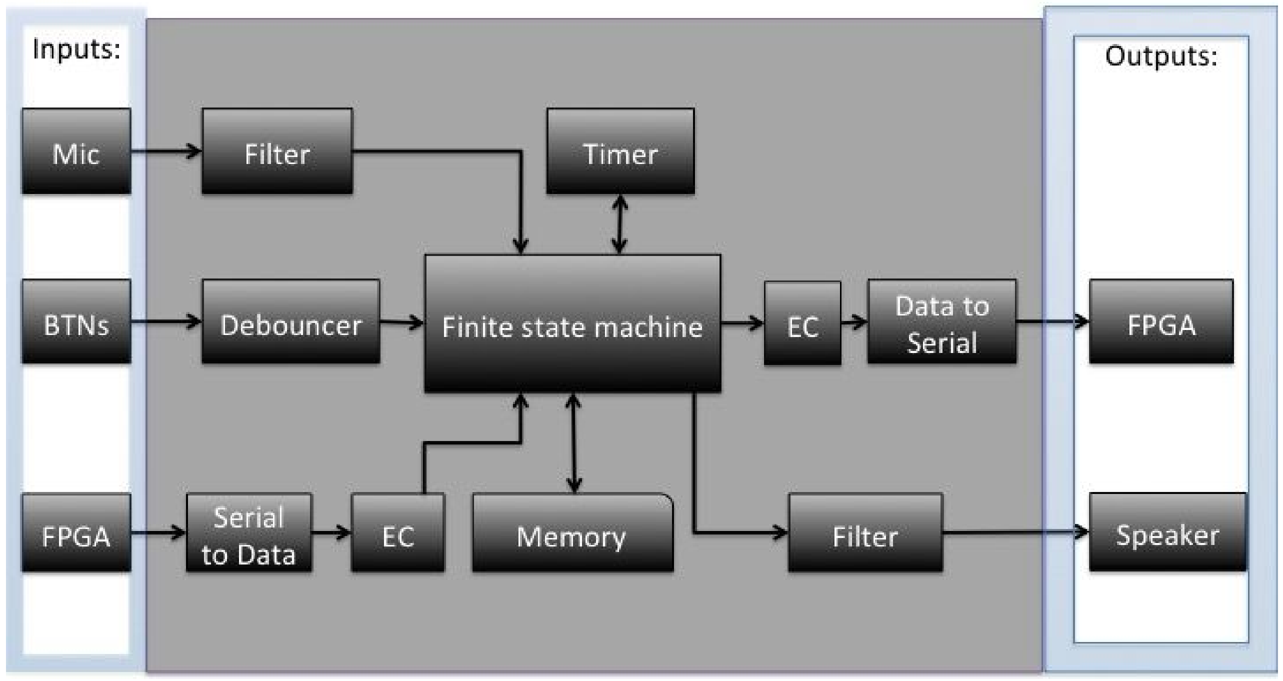

Figure 2. Block Diagram of the Telephone System.

Inputs

Microphone

We will receive data from the Microphone in three different occasions. If we are in a middle of a call or we are leaving a voice message we need to send the data to the other FPGA. If we are recording a our pre-recorded voice message we need to save the sound to memory. The FSM will decide what to do with this data. Before manipulating the sound, we will down-sample it and pass it through a low pass filter to remove aliasing. We also want to maintain good audio quality, so we will use 12 bits from the ac97 instead of just 8, as in Lab 5.

Buttons

We will use several buttons on the labkit for our project. We will need buttons to indicate:

- start of a call
- picking up a call
- recording a greeting message
- listening to saved voice messages

The signal from these buttons will be debounced and passed to the FSM.

FPGA

Communication between FPGAs will probably be one of the more challenging parts of our project, luckily, we will not be using any hardware outside the labkits. This means we do not need to worry about any standards, and we can make our own protocol that will meet our needs. We need a protocol that will be able to send and receive enough data for audio in real time. We also want our protocol to be scalable, so that we can add more FPGAs with as little changes to the system as possible.

The protocol we decided will be using a single wire to connect the FPGAs. All the the connected FPGAs will wait for a start sequence to know when there is incoming data. Once one FPGA starts sending data, the others must receive the data and can not send any data until the channel is clear.

The data being sent will be in the form of packets. Each packet will contain three parts:

- **address** [2:0] : Each FPGA will have a unique address. If the packet address matches that FPGA's address, then it acts on that packet. Otherwise the packet is ignored.
- **header** [2:0] : This tells the receiver what kind of data is being sent. For example, we need to be able to distinguish between audio data for a conversation vs. the data for a message.
- **data** [11:0] : This is the 12 bit audio data.

This adds up to an 18 bit packet. A few more bits will be added to this for error correction before it is serialized and sent across a single wire.

Timer

We want to limit the length of an unanswered phone call and send back the pre-recorded greeting after some period of time. Also we can't have arbitrarily long greetings so we will also limit them to a minute. In order to be able to enforce the time we will need a Timer module, which will be responsible for managing the time. It will take as an input a number of seconds and will activate a signal after the period has elapsed.

Error correction

As we said earlier, we will use error correction to ensure the correct transmission of the data from one FPGA to the other. The simplest form of error correction is Automatic Repeat Request. We can use some error detection mechanism (a parity bit for example) and then if we detect an error we will ask for a retransmission.

If needed, we can apply more complicated schemes: Reed-Solomon is the most widespread error correction algorithm. We can send additional bits to each of the packets that we transmit. With the help of the extra information we will be able to detect errors and if there aren't too many errors we will be able to correct them.

Memory

We need to use the flash memory to store the personal greeting and the voice messages, which are left by the caller. We will have a memory module from which we can read or write depending on the state of the FSM. We will store the audio in a down-sampled(6KHz) so that we can store more information. Using the 128 Mbits of memory available to use, we can store up to 28 messages and one personal greeting, each up to a minute in length.

Timeline and Testing

In the first week of the project we will build a simple prototype of a telephone system. The prototype will consist of just two FPGAs that are connected with each other and we transmit audio from one to the other.

In the second week we will implement the pre-recorded greetings and voice messages. Also we will start work on the error correction algorithms so that we can get a better audio quality.

In the third week we will finalize the details around the voice messages - allowing recording multiple messages, make sure we do not overfill the memory and so on. We will also implement small details as a LED light, which is blinking if we have a voice message left or a specific ringing tone when are getting a call.

In the final week we will make sure that everything is working properly. We will try implementing some of our stretch goals if time permits.

	11/9	11/16	11/23	11/30
Building State Machine	🟦			
Simple Communication	🟥			
Testing SM		🟦		
Memory Modules		🟦		
Error Correction		🟥		
Serializing		🟥		
Begin Integration		🟪		
Testing Memory/Erasing			🟦	
Test Communication/Congestion			🟥	
Integration/More Testing			🟪	
Buffer Week				🟪
Stretch Goals				🟪

Stretch Goals

If we finish the project early, we have several stretch goals in mind. We can try modifying our system to work on three or more FPGAs connected serially. We may need to introduce changes to our communication protocol as the channel becomes more congested. It will become more likely for multiple FPGAs to try to send data at the same time. We also need to prevent FPGAs from controlling the channel and not letting other FPGAs send data.

After that we can explore what happens when we significantly increase the distance between the two or three FPGAs that we have connected. We can test whether there is a lot of noise and desynchronization in the system. If there are such problems, we can try finding a solutions and make the design act as a real telephone system.

Lastly, we would like to use a display. We can start simply by showing the current state on the screen or showing how many messages you have waiting. We can also use a display to bring in new features, such as caller ID, depending on the amount of time we can devote to this.