Dual Tone Multi-Frequency Dialer

Final Project Report
December 15, 2003
E155

Don Lee and Min Shim

Abstract:
Simulating a touch-tone telephone, the dual tone multi-frequency (DTMF) dialer will be able to emit the dial tones of a telephone through one speaker to dial numbers on any conventional phone. The FPGA will receive a debounced signal from a four-by-four keypad, which will in turn be used to simulate a triangle-wave with the corresponding frequency of the number pressed. At the same time, the PIC microcontroller will be receiving the numbers being pressed and storing them into memory (FLASH program memory) for both redial and store/speed-dial functions. (Whenever the redial button is pressed, the previously dialed numbers will be played on the speaker and whenever the speed-dial button is pressed, the stored number (currently, only one set of numbers can be stored) will be played. Thus, there will be three main functions of the DTMF dialer: dial, store/speed-dial, and redial phone numbers. Whenever redial or speed dial is pressed, the previously saved numbers will be played. In our final results, the DTMF dialer was able to call numbers; however, it did not have the redial and speed-dial functions.

Introduction

The tones of a telephone are called dual tone multi-frequencies and are created by adding two sine waves together. On the keypad of a phone, each row and column represents a certain frequency. The rows correspond to the low frequencies and the columns correspond to the high frequencies.

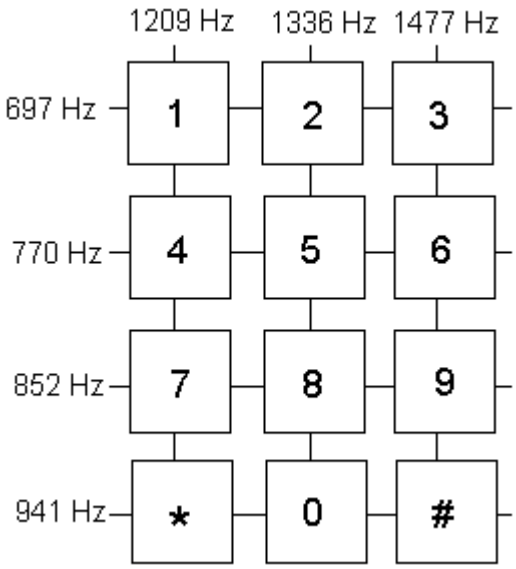

Fig. 1.1 Frequency Set-Up

As seen in figure 1.1, the rows have frequencies ranging from 770 Hz to 941 Hz (low frequencies) and the columns have frequencies ranging from 1209 Hz to 1477 Hz (high frequencies). When a button is pressed, the row frequency and the column frequency are played at the same time, creating the tone that is heard on the phone. For example, if the number five is pressed, 770 Hz and 1336 Hz will be played.

For this project, the following block diagram illustrates the relationship between the keypad, FPGA, speaker, and PIC.

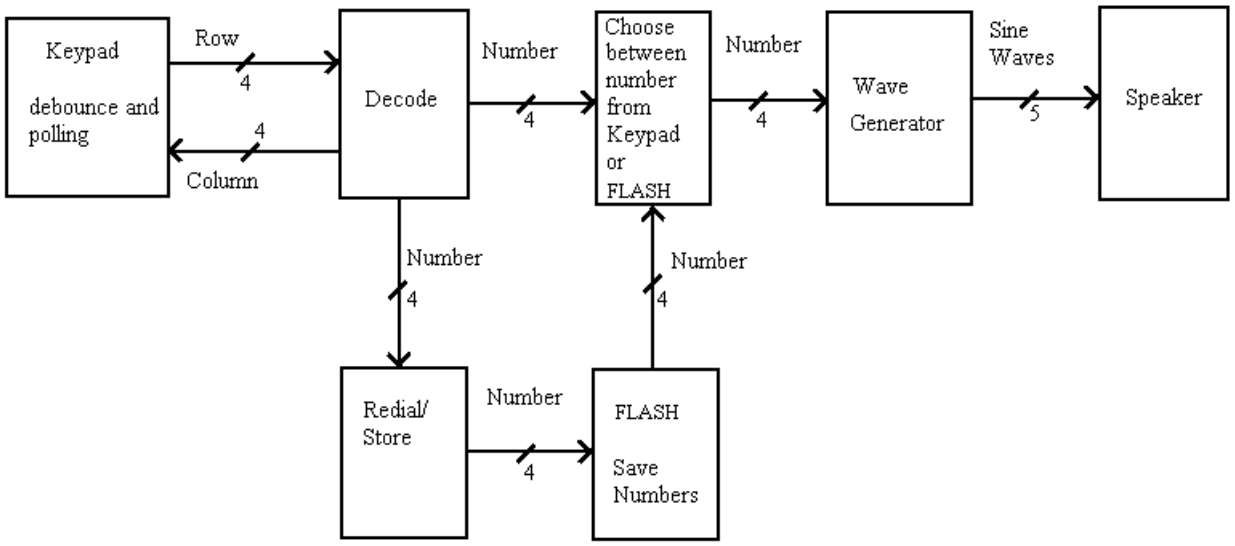

Fig. 1.2 Block Diagram

The keypad utilizes scanning and polling to obtain which button is pressed. The signal is then sent to the FPGA, where it is decoded and generated into a triangle wave to be played through a speaker. Also, the FPGA receives data from the PIC (numbers from memory, either for redial or speed-dial) to be generated and played. The PIC is used to store the numbers that are needed for speed-dial or redial. Every time a number is pressed on the keypad, it is saved in FLASH program memory for either redial or speed-dial. Also on the keypad are four extra buttons, for redial, store start, (store stop,) and speed-dial along with a separate button next to the keypad, for start and stop (pick-up and hang-up).

Schematics

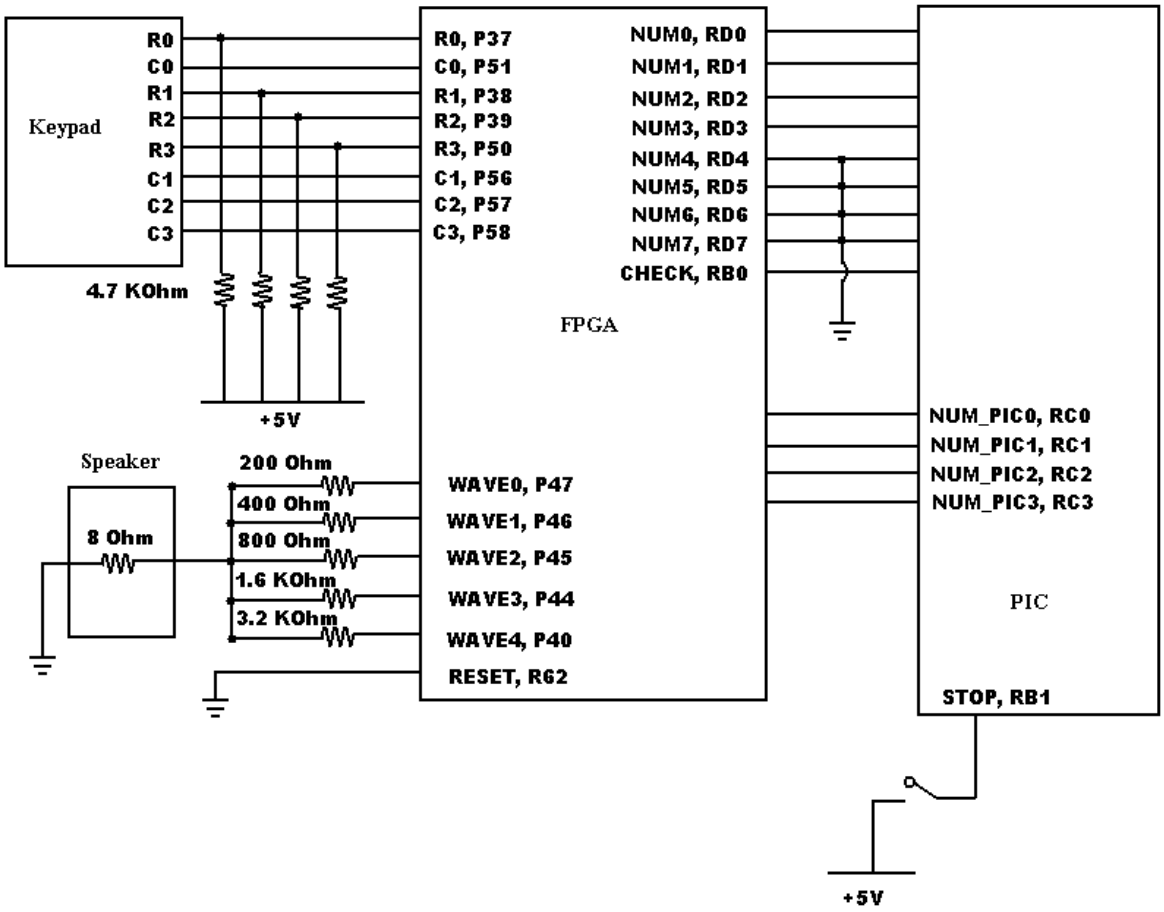

Fig. 2.1 Schematic of the Circuit Used

All the rows of the keypad receive +5V through 4.7 KOhm resistors to prevent sending too much current to the keypad. Thus, the keypad is always high.

PORTD[7:0] is used as an output from the FPGA (binary representation of the number pressed) and an input to the PIC, while PORTC[3:0] is used as an output from the PIC (again, binary representation of a number) and an input to the FPGA. The PIC also has an input from a switch to PORTB (bit 1, RB1) that will be set high when a switch is pressed.

Using five resistors, a D/A converter is created to convert the digital signal sent by WAVES[4:0] to analog, so that it can be played through the 8 ohm speaker. Since the most

significant bit (MSB) should have the largest value, i.e. the greatest current, the lowest resistance is used for that bit. All resistance values are then multiplied by 2 as it goes from MSB to LSB. The reason behind multiplying by 2 is to reflect how each bit represents a multiple of 2.

Microcontroller

1. Basic Function

In this project, the basic function of the microcontroller is to store the telephone numbers for either redial or speed-dial. For this saving method, FLASH program memory is used and the numbers for redial are stored from register 0800 on and for store/speed-dial, 0600 on. Up to seven numbers can be redialed or stored/speed-dialed. For redial purposes, if the phone is not in store mode, every number that is pressed is saved into memory. If the store button is pressed, then the next (up to) seven numbers will be saved for speed-dial. The microcontroller also sends the saved numbers back to the FPGA to be converted into triangle waves.

2. Inputs and Outputs

The microcontroller has two inputs and one output: PORTD and PORTB are inputs, while PORTC is an output. PORTD contains an eight bit binary hex number, which represents what button on the keypad was pressed. Note that the bits RD7-RD4 are grounded, so that they are always Low. We used two bits of PORTB (bit 0 and bit 1) to check if the start/stop button is pressed and whether or not the keypad has been released. The start/stop button is a button separate from the keypad, which functions as the pick-up and hang-up actions of a telephone (bit 1). To check and see if the keypad is released from a previous press (just incase the button is pressed indefinitely) bit 0 receives &row, which will be High when none of the buttons are pressed on the keypad and Low if a button is pressed. The output, PORTC consists of four bits, which represent the numbers 0 to F, and will be sent to the FPGA. From the FPGA, the binary number from the PIC will be converted into a triangle wave of the corresponding frequency.

3. Key Algorithms

number_check

Number_check is the main part of the code. From here, the code is looped until a button on the keypad or the start/stop button is pressed. If a button is pressed, the corresponding check and branch is initiated and the code branches to the appropriate part of code.

table_write

Both the redial save and store functions call on this algorithm. Before the call to table_write, the counter and pointer variables are cleared or set to the appropriate address. When it is called, the PIC begins to write data into the FLASH program memory, starting from register 0800 or 0600 depending on whether or not the store button was pushed. After the write, PORTB bit 0 is checked to see if the button was held down, so that it will not continuously write the same number over and over. Then a counter is incremented, to count how many numbers have been pushed and the algorithm returns to its last position in the code

table_read

Again, both the redial save and store functions call on this algorithm and again, the table pointers are cleared and set to either 0800 or 0600. According to the size of the counter, the table_read algorithm reads the table of values, starting from either 0800 or 0600 and ending when the counter has been decremented to zero. After each value is read, it is set to the output PORTC, to be sent to the FPGA where it will be played on the speaker and a delay is called, so that all the sounds are not played at once (since one cycle in the PIC is extremely fast). This algorithm is looped until the counter is decremented to zero.

delay

A basic delay algorithm where cycles are wasted so that there is a delay of some predetermined time.

FPGA

1. Counter

Counter provides a slow clock for the other modules. Counter is a 13-bit counter and has two outputs: newclk and slowclk. The clock cycle for this project was set to 2MHz and is too fast to sample the state of the keyboard, since a button can be pressed for a long time. Moreover, 2 different numbers cannot be pushed faster than a 5 ms interval, so a slower clock rate is needed to debounce the signal from the keypad. To achieve this, the MSB of the 13-bit counter was used as newclk, which provides an approximate 4.096 ms period. Slowclk is used in Wave Generator to generate sine waves, which will be explained more thoroughly in the Wave Generator section.

2. Scanner

Scanner takes the inputs of clk, newclk, reset, and row and outputs c (columns) and d, the number corresponding to what has been pushed on the keypad. In scanner, a finite state machine was used for polling (refer to the figure 3.1). Each column is used as a state. All rows are connected to a 5V source with 4.7 kOhm, which makes all rows High by default. When a button is pressed, one of rows will become Low and the corresponding column can be found through the FSM. All columns are set to High except the one to poll and each state has a corresponding value of the column. For example, if the FSM is at its first state where it polls the first column, the state and c (column) will have a value of 0111.

To decode the numbers pressed, logic for each number was found, in terms of row and column values. To debounce the signal, newclk was used so that the output d, would only receive the new value of s, a register used to hold a decoded number, every period of 4.096 ms. At the same time, to distinguish between a continuous push from a new push, a register k was

used as a check bit. If k becomes High only when all the rows are High, i.e. when a button is either not pushed or released, d receives the new values of s. If k is cleared, d gets the new value of s.

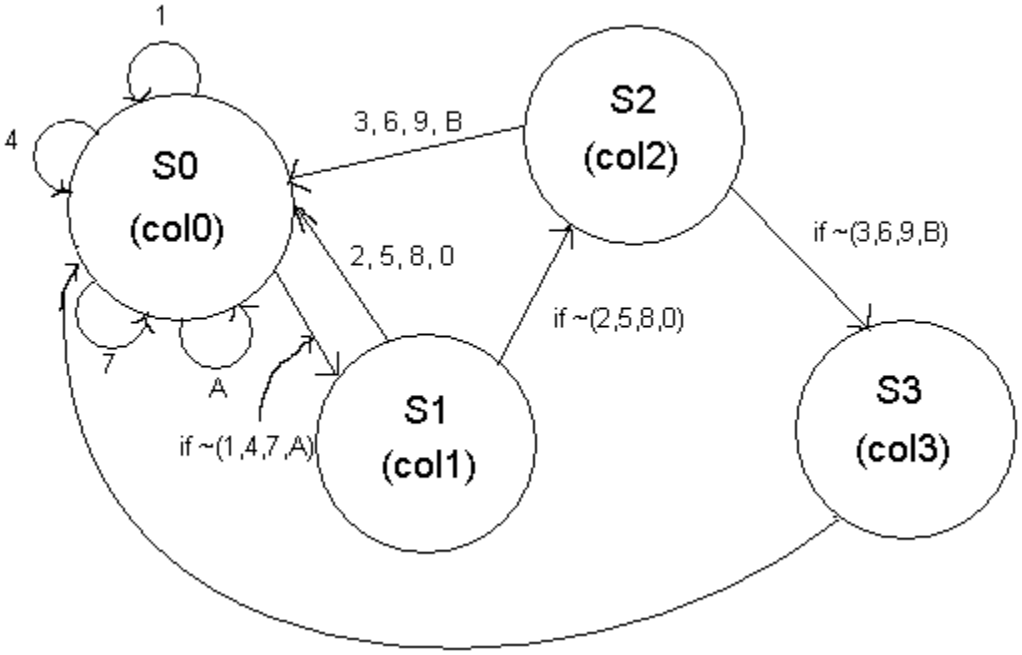

Fig. 3.1 FSM for the Keypad

3. Choose

This module has inputs clk, fpga and pic and an output of wave. fpga represents a number coming from the FPGA and pic a number from the PIC. When the redial or speed-dial button is pressed, the FPGA needs to get data from the PIC as an input to HighFreq and LowFreq, to play dial tones of previously stored numbers. Button C on the keypad is assigned to Redial and E to Store. When the button pressed is either C or E then this module switches the input source to the PIC so that the stored numbers can be dialed. Wave is an output which will be either a number from the FPGA or the PIC.

4. HighFreq and LowFreq

HighFreq stands for High Frequency and LowFreq for Low Frequency. These modules are used to generate different frequencies needed for each dial tones.

	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz		0	

Table 1. Numbers on Keypad and Corresponding Frequencies

Each number has two different frequencies: column frequencies are a group of high frequencies and row frequencies are a group of low frequencies. HighFreq and LowFreq have almost identical code except for the numbers of counters needed to achieve the desired frequencies, so only HighFreq will be explained.

HighFreq has inputs slowclk, reset, rows, and number and an output of High, which will create a triangle wave with the desired frequency. To create a triangle wave, a table of 16 values of a triangle wave with one period is used. Triangle waves of different frequencies can be generated by varying how often these values are sampled. If slower values are sampled, lower frequencies are generated. The sampling frequencies needed to generate the desired frequencies are found by the following step: The number of cycles wasted to slow the 2 MHz clock cycle to, say, 1209 Hz, is calculated by dividing 2 MHz by 1209 Hz, then by dividing by 2. The reason why it is divided by 2 is because slowclk (a 2-bit counter) is used to reduce number of bits needed to represent the sampling cycles. And the sampling frequency to generate a triangle wave is found by dividing the resultant cycles by 16, since there are 16 values from a triangle wave sampled into a table.

For each number, freq is the required sampling cycles for a corresponding frequency. Every rising edge of slowclk, counter is incremented. When this counter reaches the number of

sampling cycles, another counter, cycle is incremented. Cycle represents the indices of values in a triangle wave table. To summarize, for every sampling cycle, values from the table is sampled.

5. WaveGen

WaveGen stands for Wave Generator. It has inputs slowclk, clk, High, Low, row, and number and outputs of wave, check and d. To generate a triangle wave of two different frequencies in Wave Generator, for every slowclk, High and Low values from the frequency functions are added. To cover overflowing numbers, wave, the result of addiction of High and Low, has 5 bits. Check is a check bit that is sent to the PIC to debounce a signal. Check is set High when all rows are high and Low otherwise. Output d is the same as the output number from scanner except that it is set to hexadecimal number A after a button is released. This is to prevent the PIC from saving the same number several times. The use of Check and the purpose of d will be further explained in Microcontroller section.

6. Dialer

Dialer is the hierarchy module for all the other modules. Dialer calls all the functions in the order listed.

Results

The dual tone multi frequency phone dialer dials the numbers pressed when the phone receiver is placed close to the speaker. It saves the most recent numbers pressed for redial as well as for store. We stated in the initial proposal that it would be able to dial numbers when redial or speed-dial is pressed. However, playing (dialing) numbers saved in the PIC was not functioning in the final results.

One of most difficult parts of the design was to save and read numbers correctly from the FLASH program memory. For saving, we observed that every button pressed is constantly delayed. For example, if one is pressed, zero is saved as default, and when the next number is pressed, one will be saved. This resulted in not saving the very last number and to solve the problem, a dummy button without any value in the PIC, i.e. A or B had to be pushed to capture all numbers. It also caused a problem for the pointer when data was read. The lower bit of table pointer has to be set to 01 instead of 00 when the reading started. We spent most of our time trying to figure out how to save correctly and thus, we did not get to look more closely as to how redial and speed dial had to be done. To check these functions mentioned, we put LED's to check whether PORTA, the output port, to see if it was receiving the right values and we were confident that it did. However, since the saved numbers did not correctly reflect the numbers pressed due to delay problem, redial or speed-dial did not playback the right numbers.

Another difficulty we faced was trying to amplify the sound. We tried several Op-Amp circuits as well as a potentiometer coupled with a transistor. Some amplification worked, such as the potentiometer and transister, but it added a lot of noise to our signals and a phone receiver would not pick up the signals. As a result, we ended up putting the phone receiver very close to the speaker to dial.

This project could have been improved if we managed our time better. We spent most of the given time trying to figure out how the EEPROM and Flash program memory worked and neglected other parts of the project, which should have been more important. It could also have helped us if we researched more prior to the project. Having little knowledge of the PIC and how the FPGA and PIC communicate cost us a lot of time, resulting in a project that did not meet our initial proposal.

References

[1] E155 Microprocessor-Based Systems Lab Manual, Fall 2003

[2] DTMF. (This site is used to obtain frequencies.)

http://www.boondog.com/%5Ctutorials%5Cdtmf%5Cdtmf.htm

[3] Generating DTMF tones using soundcard

http://www.hut.fi/~then/mytexts/dtmf_generation.html

Parts List

No new parts were used for this project.

Appendix A: Verilog Module

A-1. Top-module: Dialer

```
module Dialer(clk,reset,row,pic,col,wave,d,check,High,Low);
        input clk;
        input reset;
        input [3:0] row;
        input [3:0] pic;
        output [3:0] col;
        output [4:0] wave;
        output [3:0] d;
        output check;
        output [4:0] High;
        output [4:0] Low;

         wire newclk;
         wire slowclk;
         wire [3:0] number;
         wire [3:0] waveIn;
         wire [4:0] High;
         wire [4:0] Low;
         wire [3:0] s;

        //counter outputs two different slower clocks
         counter count(clk,reset,newclk,slowclk);

        //scanner decodes numbers pressed
         Scanner number(clk,newclk,reset,col,row,number,s);

        //Choose chooses input to High- and LowFreq between FPGA and PIC
         Choose inputChoose(clk,number,pic,waveIn);

        //generates triangle waves for high frequency groups
         HighFreq highfreqs(slowclk,reset,row,waveIn,High);

        //generates triangle waves for low frequency groups
         LowFreq lowfreqs(slowclk,reset,row,waveIn,Low);

        //add high and low frequencies to generate final waves and some check bits
        //to be used in PIC
         WaveGen waves(slowclk,clk,High,Low,wave,check,row,number,d);

endmodule
```

A-2. Counter

```
module counter(clk,reset,newclk,slowclk);
        input clk;
        input reset;
        output newclk;
        output slowclk;

         reg [12:0] q;

        //q is a 13-bit counter
        always @(posedge clk or posedge reset)
                if (reset) q <= 13'b0;
                else q <= q+1;

        //new clock is generated by slowing down by 2^13 bits
        assign newclk = q[12];

        //slowclk is generated by slowing down by 2^2 bits
        assign slowclk = q[1];

endmodule
```

A-3. Scanner

```
module Scanner(clk,newclk,reset,c,r,d);
        input clk;
        input newclk;
        input reset;
        input [3:0] r;
        output [3:0] d;
        output [3:0] c;

        reg [3:0] state;
        reg [3:0] nextstate;
        reg [3:0] s;
        reg [3:0] d;
        reg k;


         //parameter values are set to be equal to bits of c corresponding to its state
         parameter S0 = 4'b1110;
         parameter S1 = 4'b1101;
         parameter S2 = 4'b1011;
         parameter S3 = 4'b0111;
```

```
 parameter S4 = 4'b0000;

//unless reset, states gets next state
 always @(posedge clk or posedge reset)
        if(reset) state <= S0;
        else      state <= nextstate;

//assign column to have the same value as state
assign c = state;

 //state register for FSM
 always @(state or r)
        case(state)
                S0: begin
                        //state transition only when none of row is LOW
                        //stays at the same state when one of row turns LOW to
                        //keep s value constant until next key is pressed

                        if(~(r[3]&r[2]&r[1]&r[0])) nextstate <= S0;
                        else                       nextstate <= S1;
                        end

                S1: begin
                        if(~(r[3]&r[2]&r[1]&r[0]))  nextstate <= S1;
                        else                        nextstate <= S2;
                        end

                S2: begin
                         if(~(r[3]&r[2]&r[1]&r[0])) nextstate <= S2;
                        else                        nextstate <= S3;
                        end

                S3: begin
                         if(~(r[3]&r[2]&r[1]&r[0])) nextstate <= S3;
                        else                        nextstate <= S0;
                        end
                default: nextstate <= S0;
        endcase

//output using logics in terms of columns and rows
assign s[3] = (~r[1]&~c[1]) | (~r[1]&~c[2]) | (~c[0]&~r[0]) | (~c[2]&~r[0]) |
              (~c[3]&~r[3]) | (~c[3]&~r[2]) | (~c[3]&~r[1]) | (~c[3]&~r[0]);
assign s[2] = (~c[0]&~r[2]) | (~c[1]&~r[2]) | (~c[2]&~r[2]) | (~c[0]&~r[1]) |
              (~c[3]&~r[3]) |  (~c[3]&~r[2]) | (~c[3]&~r[1]) | (~c[3]&~r[0]);
assign s[1] = (~c[1]&~r[3]) | (~c[2]&~r[3]) | (~c[2]&~r[2]) | (~c[0]&~r[1]) |
              (~c[0]&~r[0]) | (~c[2]&~r[0]) | (~c[3]&~r[1]) | (~c[3]&~r[0]);
```

```
        assign s[0] = (~c[0]&~r[3]) | (~c[2]&~r[3]) | (~c[1]&~r[2]) | (~c[0]&~r[1]) |
                        (~c[2]&~r[1]) | (~c[2]&~r[0]) | (~c[3]&~r[2]) | (~c[3]&~r[0]);

        //debouncing signals; d does not get a new value until a button is released
        //previously
        always @(posedge newclk)
                if((r[3]&r[2]&r[1]&r[0])) k <= 1;
                else if(~(r[3]&r[2]&r[1]&r[0]) && k)
                        begin
                        d <= s;
                        k <= 0;
                        end

endmodule
```

A-4. Choose

```
module Choose(clk,fpga,pic,wave);
         input clk;
        input [3:0] fpga;
        input [3:0] pic;
        output [3:0] wave;

         reg [3:0] wave;

        //if C(redial) or Speed-Dial(E) is pressed, input comes from PIC
         always @(posedge clk)
                if(fpga == 4'b1100 | fpga == 4'b1110)      wave <= pic;
                else                                      wave <= fpga;

endmodule
```

A-5. HighFreq

```
module HighFreq(slowclk,reset,row,num,High);
        input slowclk;
        input reset;
        input [3:0] row;
        input [3:0] num;
        output [4:0] High;


         reg [4:0] High;
         reg [4:0] counter;
```

```
reg [4:0] freq;
reg [3:0] cycle;
reg counter_clr;


//clock cycle should be set at 2MHz
always @(num)
        case(num)
                //when 0 is pressed, frequency of 1336Hz
                4'b0000: freq <= 5'b10101;
                //when 1 is pressed, frequency of 1209Hz
                4'b0001: freq <= 5'b11000;
                //when 2 is pressed, frequency of 1336Hz
                4'b0010: freq <= 5'b10101;
                //when 3 is pressed, frequency of 1477Hz

                4'b0011: freq <= 5'b10011;
                //when 4 is pressed, frequency of 1209Hz
                4'b0100: freq <= 5'b11000;
                //when 5 is pressed, frequency of 1336Hz
                4'b0101: freq <= 5'b10101;
                //when 6 is pressed, frequency of 1477Hz
                4'b0110: freq <= 5'b10011;
                //when 7 is pressed, frequency of 1209Hz
                4'b0111: freq <= 5'b11000;
                //when 8 is pressed, frequency of 1336Hz
                4'b1000: freq <= 5'b10101;
                //when 9 is pressed, frequency of 1477Hz
                4'b1001: freq <= 5'b10011;
                default: freq <= 5'b0;
         endcase

always @(posedge slowclk or posedge reset)
        if(reset)                       //clear at reset
        begin
        counter <= 5'b0;
        cycle <= 4'b0;
        counter_clr <= 0;
        end

        else if(counter_clr)
        begin
        counter <= 5'b0;                //reset counter when cycle is incremented
        counter_clr <= 0;
        end
```

```
        //if a button is released, clear counter
        else if(&row)                   counter <= 5'b0;


        //if counter reaches number of cycles needed for frequencies, cycle is
        //incremented and counter_clr is set High to clear counter
        else if(counter == freq)
                begin
                cycle <= cycle + 1;
                counter_clr <= 1;
                end

        //if not anything above, increment counter
        else                    counter <= counter + 1;


 //table of sine values sampling 16 points per period
 //for convention, values are sampled from triangle wave since it should still yield
 //almsot identical shape due to time delay between two values
always @(cycle)
        case(cycle)
                // 11, i = 0, where i is index
                4'b0000: High <= 5'b01011;
                // 12, i = 1
                4'b0001: High <= 5'b01100;
                // 13, i = 2
                4'b0010: High <= 5'b01101;
                //14, i = 3
                4'b0011: High <= 5'b01110;
                //15, i = 4
                4'b0100: High <= 5'b01111;
                //14, i = 5
                4'b0101: High <= 5'b01110;
                //13, i = 6
                4'b0110: High <= 5'b01101;
                //12, i = 7
                4'b0111: High <= 5'b01100;
                //11, i = 8
                4'b1000: High <= 5'b01011;
                //10, i = 9
                4'b1001: High <= 5'b01010;
                //9, i = 10
                4'b1010: High <= 5'b01001;
                //8, i = 11
                4'b1011: High <= 5'b01000;
                //7, i = 12
```

```
                4'b1100: High <= 5'b00111;
                //8, i = 13
                4'b1101: High <= 5'b01000;
                //9, i = 14
                4'b1110: High <= 5'b01001;
                //10, i = 15
                4'b1111: High <= 5'b01010;
                default: High <= 5'b00000;
        endcase


endmodule
```

A-6. LowFreq

```
module LowFreq(slowclk,reset,row,num,Low);
        input slowclk;
        input reset;
        input [3:0] row;
        input [3:0] num;
        output [4:0] Low;

         reg [4:0] Low;
         reg [5:0] counter;
         reg [5:0] freq;
         reg [3:0] cycle;
         reg counter_clr;

         //clock cycle should be set at 2MHz
         always @(num)
                case(num)
                        //when 0 is pressed, frequency of 941Hz
                        4'b0000: freq <= 6'b011111;
                        //when 1 is pressed, frequency of 697Hz
                        4'b0001: freq <= 6'b101100;
                        //when 2 is pressed, frequency of 697Hz
                        4'b0010: freq <= 6'b101100;
                        //when 3 is pressed, frequency of 697Hz
                        4'b0011: freq <= 6'b101100;
                        //when 4 is pressed, frequency of 770Hz
                        4'b0100: freq <= 6'b100111;
                        //when 5 is pressed, frequency of 770Hz
                        4'b0101: freq <= 6'b100111;
                        //when 6 is pressed, frequency of 770Hz
                        4'b0110: freq <= 6'b100111;
```

```
                //when 7 is pressed, frequency of 852Hz
                4'b0111: freq <= 6'b100011;
                //when 8 is pressed, frequency of 852Hz
                4'b1000: freq <= 6'b100011;
                //when 9 is pressed, frequency of 852Hz
                4'b1001: freq <= 6'b100011;
                default: freq <= 6'b0;
        endcase

always @(posedge slowclk or posedge reset)
        if(reset)                       //clear at reset
        begin
        counter <= 5'b0;
        cycle <= 4'b0;
        counter_clr <= 0;
        end

        else if(counter_clr)
        begin
        counter <= 5'b0;                //reset counter when cycle is incremented
        counter_clr <= 0;
        end

        //if a button is released, clear counter
        else if(&row)                           counter <= 5'b0;


        //if counter reaches number of cycles needed for frequencies, cycle is
        //incremented and counter_clr is set High to clear counter
        else if(counter == freq)
                begin
                cycle <= cycle + 1;
                counter_clr <= 1;
                end

        //if not anything above, increment counter
        else                            counter <= counter + 1;


 //table of sine values sampling 16 points per period
 //for convention, values are sampled from triangle wave since it should still yield
 //almsot identical shape due to time delay between two values
always @(cycle)
        case(cycle)
                // 11, i = 0, where i is index
                4'b0000: Low <= 5'b01011;
```

```
                // 12, i = 1
                4'b0001: Low <= 5'b01100;
                // 13, i = 2
                4'b0010: Low <= 5'b01101;
                //14, i = 3
                4'b0011: Low <= 5'b01110;
                //15, i = 4
                4'b0100: Low <= 5'b01111;
                //14, i = 5
                4'b0101: Low <= 5'b01110;
                //13, i = 6
                4'b0110: Low <= 5'b01101;
                //12, i = 7
                4'b0111: Low <= 5'b01100;
                //11, i = 8
                4'b1000: Low <= 5'b01011;
                //10, i = 9
                4'b1001: Low <= 5'b01010;
                //9, i = 10
                4'b1010: Low <= 5'b01001;
                //8, i = 11
                4'b1011: Low <= 5'b01000;
                //7, i = 12
                4'b1100: Low <= 5'b00111;
                //8, i = 13
                4'b1101: Low <= 5'b01000;
                //9, i = 14
                4'b1110: Low <= 5'b01001;
                //10, i = 15
                4'b1111: Low <= 5'b01010;
                default: Low <= 5'b00000;
        endcase


endmodule
```

A-7. WaveGen

```
module WaveGen(slowclk,clk,High,Low,wave,check,row,number,d);
        input [4:0] High;
        input [4:0] Low;
        input clk;
        input slowclk;
        input [3:0] row;
        output [4:0] wave;
```

```
        input [3:0] number;
        output [3:0] d;
        output check;


         reg [4:0] wave;
         reg [3:0] d;
         reg check;

        //every time when waves from HighFreq and LowFreq are generated, they are
added to
        //generate waves required for dial tone
         always @(posedge slowclk)
                wave <= High + Low;

        //check bit used in PIC
        //check is set High when all rows are high, i.e. when no button is pressed
        always @(posedge clk)
                        check <= &row;

        //when d is sent to PIC
        //if a button is released, d receives A which has no value in PIC
        always @(posedge slowclk)
                if(check)               d <= 4'b1010;
                else                    d <= number;
endmodule
```

Appendix B: flash.asm

```
;flash.asm
;Written 12/05/03 by dhlee@hmc.edu and mshim@hmc.edu
;this program saves a set of numbers from FPGA and put it in Flash program memory
;for redial-purpose, table pointer starts from 0800 and for speed-dial purpose, pointer
starts from 0600
;since we are saving only maximum of 7 numbers, pointers being close do not affect
each other.


;use the 18f452 PIC microprocessor
 LIST p=18f452
 include "p18f452.inc"


;allocate variables

counter         EQU 0x08
counter0        EQU 0x09
counter1        EQU 0x0A
counter2        EQU 0x0B
stop_check      EQU 0x10
store_check     EQU 0x11
count           EQU 0x12
count1          EQU 0x13
count2          EQU 0x14
wait_counter    EQU 0x15



        org 0x000

main
                movlw 0xCF
                movwf TRISD             ;set PortD as input
                movlw 0x03
                movwf TRISB             ;set PortB as input
                clrf TRISC              ;set PortA as output
                clrf INTCON             ;disable all interrupts

;number_check checks for each number
number_check
                movlw h'0C'
                subwf PORTD,0
                bz redial               ;button C is for redial
```

```
                movlw h'0D'
                subwf PORTD,0
                bz store                ;button D is for store function
                movlw 0Eh
                subwf PORTD,0
                bz speed                ;button E is for speed dial
                movlw 03h
                subwf PORTB,0           ;if stop button is pushed, loop to stop
                bz stop
                movlw 02h
                subwf PORTD,0
                bz numbers              ;other than that, loop to save numbers for
                                        ;redial later
                movlw 03h
                subwf PORTD,0
                bz numbers
                movlw 04h
                subwf PORTD,0
                bz numbers
                movlw 05h
                subwf PORTD,0
                bz numbers
                movlw 06h
                subwf PORTD,0
                bz numbers
                movlw 07h
                subwf PORTD,0
                bz numbers
                movlw 08h
                subwf PORTD,0
                bz numbers
                movlw 09h
                subwf PORTD,0
                bz numbers
                movlw 00h
                subwf PORTD,0
                bz numbers
                movlw 01h
                subwf PORTD,0
                bz numbers
                bra number_check        ;if nothing is pushed, keep looping until PortD
                                        ;has a valid push

numbers
                movlw 01h
```

```
                subwf store_check,0     ;if store was pushed previously, go to store
loop to keep saving
                bz store_two
                movlw 01h
                cpfseq stop_check       ;if stop was pressed previously, it means a new
set of numbers started
                bra numbers_save
numbers_initiate
                clrf counter            ;then clear counter and stop_check
                clrf stop_check
                clrf TBLPTRU
                movlw 08h               ;pointer starts from 0800
                movwf TBLPTRH
                clrf TBLPTRL
numbers_save
                movlw 08h
                movwf TBLPTRH           ;just to make sure table pointer is pointint at
                                        ;08xx
                call table_write        ;call subroutine table_write
                movff counter, counter1 ;move counter to counter1; this was used
                                        ;because both redial and store share the same
                                        ;subroutine
                bra number_check        ;after done saving, go back to number_check
                                        ;and wait for next press

redial
                clrf TBLPTRU            ;when redial button is pressed
                movlw 08h               ;start reading from 0800
                movwf TBLPTRH
                clrf TBLPTRL
                movff counter1, counter2 ;move counter1 from numbers_save to
                                        ;counter2, which will be used in table_read
                call table_read         ;subroutine table_read
                bra number_check        ;go back to number_check


store
                clrf counter            ;when store button is pressed
                clrf TBLPTRU
                movlw 06h               ;start from 0600
                movwf TBLPTRH
                clrf TBLPTRL
                clrf stop_check         ;clear stop_check if it was set to 01
                clrf counter0
                movlw 01h
```

```
                movwf store_check       ;once store is pressed, store_check is 01 so
                                        ;that next number comes back to store loop
                bra number_check        ;go back to number_check for next number
                                        ;press
store_two                               ;at next number press, if store_check is set,
                                        ;branched to this loop
                call table_write        ;call table_write
                movff counter, counter0 ;move counter to counter0 to save number of
                                        ;counters separately from redial
                                        ;counter indicates how many numbers are
                                        ;pressed in a set
                bra number_check        ;wait for next number

speed
                clrf TBLPTRU            ;when speed dial is pressed
                movlw 06h
                movwf TBLPTRH
                clrf TBLPTRL
                movff counter0, counter2 ;move saved counter2 to counter0
                call table_read         ;call table_read
                bra number_check        ;then go back to number_check for next option

stop
                movlw 01h               ;if stop button is pressed
                movwf stop_check        ;stop check is set to be 01
                clrf store_check        ;clear store_check since stop button means the
                                        ;end of a set
                bra number_check        ;then go back to number_check


;subroutine that writes numbers to Flash program memory
table_write
                movf PORTD, WREG
                movwf TABLAT            ;move pressed number to TABLAT to be saved
                TBLWT*+                 ;temporary write and increment pointer
                bsf EECON1,EEPGD        ;access program memory
                bcf EECON1,CFGS
                bsf EECON1,WREN         ;enables write
                bcf INTCON, GIE         ;disables interrupt
                movlw 55h
                movwf EECON2
                movlw h'AA'
                movwf EECON2            ;required steps to write to program memory
                bsf EECON1,WR           ;start writing
write_check2
                btfsc EECON1, WR
```

```
                bra write_check2        ;check until writing is done
                bcf EECON1,WREN         ;when it's done, disable write
wait_write
                movlw 00h
                subwf PORTB,0
                bz wait_write           ;wait until a button is pressed to return
                incf counter            ;increment counter to save how many
                                        ;numbers are pressed
                return

;subroutine that reads saved numbers from program memory
table_read
                TBLRD*+                 ;read and increment pointer
                movf TABLAT,W
                movwf PORTC             ;move read data to PORTC, output
                call delay              ;delay between numbers so that a set of
                                        ;numbers can be dialed
                decf counter2           ;decrement counter2, it reads data until
                                        ;reaches last number saved
                movlw 00h
                cpfseq counter2
                bra table_read          ;if counter2 is not zero, in other words, if
                                        ;all of saved numbers are not read
                                        ;go back to table_read until done
                return


;cause delay by wasting cycles
delay
        clrf count2
delay2
        nop
        nop                             ;wasting cycles
        nop
        nop
        nop
        movff count2, WREG
        sublw h'01FF'
        bz finish
        incf count2
        bra delay2
finish
        return

        end
```