Recipedia

Digital Cookbook

Kevin Quinn, Tim Spens, Koto Norose, Shawn Rhoades

http://www.cs.utah.edu/~tspens/cs3992/recipedia.html

Introduction

Have you ever been in your kitchen cooking something out of a cookbook, and found your hands too busy to hold up the cookbook? You spend wasted time putting down your cooking materials to pick up your book and look again at the next step in the instructions. Often, when you set down the book it closes and you have to once again find your page and place, which results in more time wasted. Our creation, Recipedia, is a digital cookbook for use in a kitchen/restaurant environment. This system will provide a convenient and easy to use interface that any person preparing a meal can follow. Through the use of a touch screen, the cook will be able to input commands directly by simply using an easy to follow graphical user interface (GUI). The system will be capable of reading the recipe out loud to the user to provide added convenience. See figure 1 below for a system block diagram.

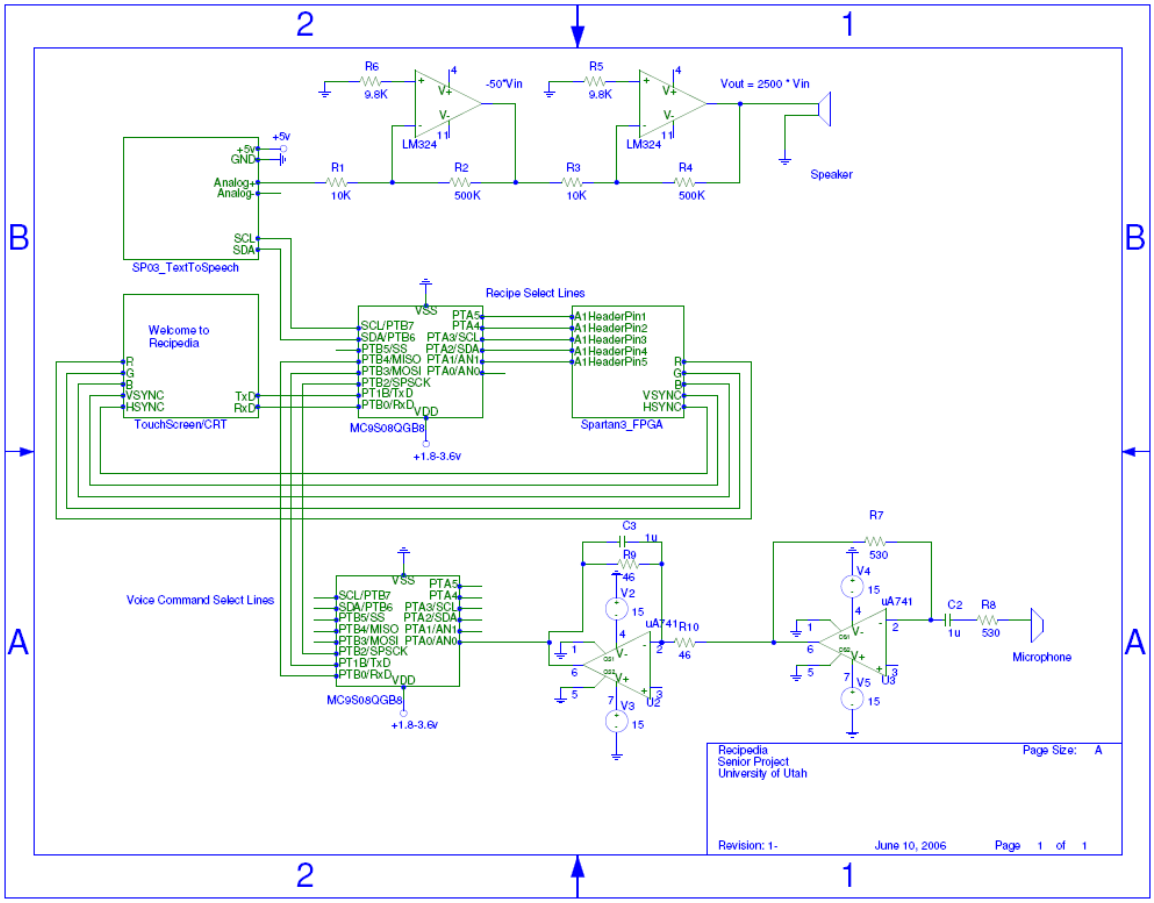


Figure 1. Block diagram of complete system

Project Functionality

We designed Recipedia to be as user friendly as possible. The first touch screen loaded on the GUI interface is referred to as HOME, this will list the different categories

available in your cookbook. Each of the categories are selectable by using the touch screen. The default categories included in Recipedia are Dinner and Deserts.

Once you have selected your food category you will be taken to a list of recipes available under that category. Under Dinner for instance you will find recipes for Golden Chicken and Curry Chicken etc. At any time you can return to home by pressing the Home button below. Each menu page will be announced as you open it (so as you choose Dinner, Recipedia will say Dinner).

Once a recipe has been selected Recipedia will begin reading the recipe aloud. The first line of the recipe will read and then wait for you to press "NEXT". After pressing "NEXT" the next line in the recipe will read. If at any time you do not understand what is being said, or need Recipedia to repeat a line, you can press "BACK" to go back a line in the recipe. Once you have reached the end of your recipe, you will want to press "HOME" to return the home page to select your next recipe.

Technical Specifications

Microcontroller C code
Environment:
We used CodeWarrior for HC08 V5.1 with the HCS08QGB8. The all code was written in C.

Code:
There main parts of the code are: Serial touch screen driver, buttons and navigation, text to speech interface, and an interface to the Spartan 3 FPGA.

Serial Touch Screen driver:
This driver was written from the HCS08 Quick Reference Guide page 64 – 68, MC9S08QG8 Datasheet page 191 – 210, examples from the Freescale Web site http://www.freescale.com and the ELO Touch Screen Driver manual http://www.elotouch.com/files/manuals/smartset.zip .

The touch screen runs at a baud rate of 9600, 8 bits, 1 stop bit and no hardware handshaking. On the HCS08 chip we used the SCI interrupt on received data buffer full to get the touch packets. Each touch packet is 10 bytes indicating the X and Y coordinates of the touch, initial touch, held touch and touch released. Once we received the complete touch packet a global flag was set in the SCI interrupt handler to signal to the mail loop that a new touch was ready to be processed.

Buttons and Navigation:
The main loop of the code is waiting for new input from the touch screen in a pulling manner. If the touch was a valid button press it continues on otherwise does nothing. Button presses also depend on which screen/location we are currently displaying. The

locations are Home, Dinner, Dessert and Recipe. Each is defined as a structure containing strings of the recipes or lines of the recipe.

If the current location is Home, there are two options: Dinner or Dessert. The main function changes the location according to recipe selections. If the current location is Dinner or Dessert recipes are displayed by title and are selectable. If the current location is Recipe, there are three options: Back, Next, and Home. If Back is pressed, the previous line of the recipe is spoken. If at the first line, it speaks the first line again. If Next is pressed, the next line of the Recipe is spoken. If at the last line, it repeats the last line again. If Home is pressed, the location is changed to Home, Home is spoken, and the display changes to Home.

The locations of the buttons were found by using a serial terminal program where we would manually touch the areas where our buttons were. We then took note of what the touch packets were and recorded that data.

Text to Speech Interface:
The text to speech interface is done over an IIC bus. The HCS08 chip has an IIC interface and interrupt when the IIC data register is written to. This was one way communication only. The microcontroller sends up to 80 characters to the Winbond WTS701 text to speech chip and then sends a command to flush that buffer. The Winbond chip takes each word and breaks it into phonemes then it uses a look up table to match each phonemes of the word to a sound. This method was used to speak each recipe line one line at a time.

Spartan 3 FPGA Interface:
The microcontroller used 5 GPIO pins to signal to the FPGA to display a BMP of the current location requested (this will be explained in more detail in the VGA design section below).

Problems Encountered:
One of the most time consuming problems we ran into with the microcontroller was stack overflow. The problem arouse when we noticed our navigation software was not working. While debugging this we found that we were getting corrupt data. It took us a while to figure out that we were blowing the stack and corrupting local variables. We moved our recipe structures into ROM global memory and this problem was gone. We ran into this problem late in the semester and we would have liked to go back and make the recipe structures into the main functions scope and increase our stack size, but we just ran out of time. We had a problem with our text to speech packets not being null terminated so we were getting undesired extra words (that we didn't want "garbage") at the end of a line.

Design functionality problem:
When the navigation code was finished we had a miscommunication that the next and back buttons were to be used to go to the next or previous page of recipe lines for one

recipe. This was fixed after we discussed how the next and back buttons were intended to function.

VGA Design

The FPGA was used to generate VGA signals to drive a monitor at 640 x 480 at about 60Hz. The recipes were also stored on a 512K SRAM. The recipes were stored as 2 bit BMPs. We only used 2 bits per color to save on memory space. In order to save a BMP in 2 bit format we used Photoshop to save as 4 bit (this is the lowest bit savable) then we wrote a program to convert the 4 bit BMP to a 2 bit BMP. One thing we learned while working on the VGA generator was that the raw BMP format is flipped vertically, when we first got a recognizable image displayed on the monitor it was flipped vertically.

Calculations for a 25Mhz pixel clock and 640 x 480:
The VGA signal timing was done by following an example at http://www.xess.com/appnotes/vga.pdf this example used a different pixel clock rate, resolution and memory interface (SDRAM). It took a while to get these numbers correct.

Pixels per line:
One complete video line is 31.77 us
31.77us * 25Mhz - 1 = 793

Lines per frame:
(16.784ms / 31.77us) – 1 = 527

Horizontal sync:
Start of sync .94us * 25Mhz + 640 + 1 = 664
Sync period 3.77us * 25Mhz + 1 = 95
End of sync 664 + 95 = 759

Vertical sync:
Start of sync .45ms * 25Mhz + 480 = 491
Sync period .064ms * 25Mhz + 1 = 2
End of sync 491 + 2 = 493

Blanking:
Video was blanked outside of visible region 640 x 480.
assign blank = (hCount >= 640 || vCount >= 480) ? 1:0;

Problems Encountered:
The VGA generator took a wile to get it working correctly. The major problem that we ran into was that our Flash memory was not fast enough for the VGA generator. Once we found this problem we changed our Verilog so that the current image being displayed was read from SRAM. The only other problem we encountered here was getting the above calculations correct from the XESS VGA generator example.

Voice Recognition

The Fast Fourier Transform allows users to obtain the spectral makeup of an audio signal, obtain the decibels of its various frequencies, or obtain the intensity of its various frequencies. The difference between them then depends upon one of a couple of equations that take the real and imaginary components of the FFT, and return either the intensity or decibel levels to be used in the result. The code takes both the real and imaginary components of the FFT result, and returns the intensity and decibels.

The FFT uses the audio signal as its real component, and uses a NULL pointer for its imaginary component indicating that the imaginary data does not exist. Upon its return, the FFT will return both the real and imaginary data components based upon the data given as the real component. The signal is mirrored with the return samples so that 0-FFT_LEN/2 contains the data, and FFT_LEN/2 to FFT_LEN contains a reverse of the data.

Once the FFT is finished with the captured audio signal a comparison is possible with pre-recorded samplings of audio. Recordings of the words “Next” and “Back” would be pre-recorded and after obtaining their audio signal and other required information you could compare future samplings that were captured while the program is running to see if there is ever a match with our pre-recorded values. If there ever was a match, we could send the signal to the FPGA just as we do when the “Next” or “Back” buttons are pressed.

The code for the Fourier transforms that we did not get integrated is listed in the Appendix.

Voice Frequency Filter Design:

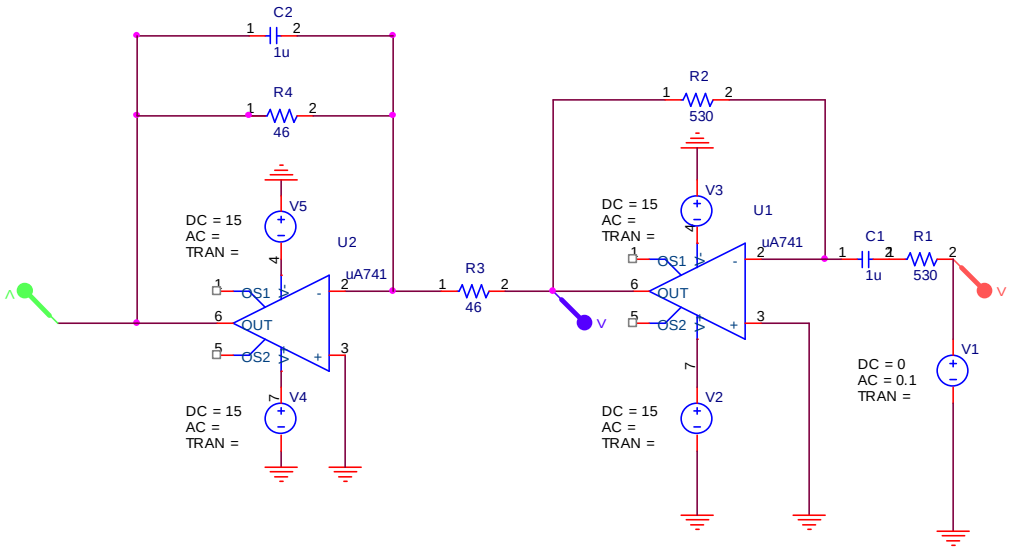


Figure 2: Schematic of voice frequency filter

From right to left: High pass filter, Low pass filter
Note: High frequencies are filtered first and then lower frequencies because the low pass filter will filter out the noise that is generated by the high pass filter.

Calculations:
The frequency spectrum of the human voice is about 300 Hz to 3400 Hz.

I set C1 to 1 uF.
FrequencyLow (r / s) = 300 * 2 * pi = 1 / (R1 * C1) from text ECE2100 page 113.
Solve for R1 = 530 Ohm
Similarly,
FrequencyHigh (r / s) = 3400 * 2 * pi = 1 / (R4 * C2)
Solve for R4 = 46 Ohm

Gain:
High pass filter: High frequency gain = -R2 / R1
Low pass filter: DC gain = -R4 / R3

Result:

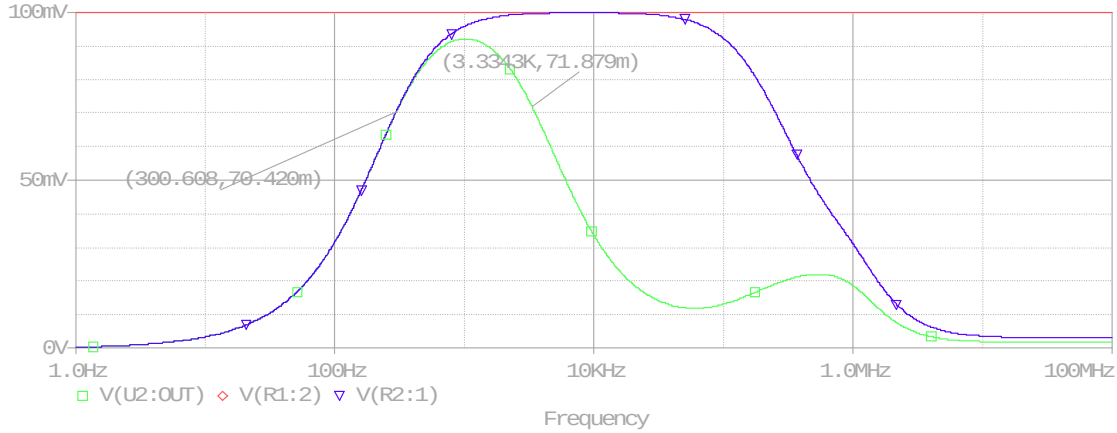


Figure 3: The AC sweep voltage

The voltage values at the frequency 300 Hz and 3400 Hz are at the 3dB points (20 * log(0.707) = -3 dB) as expected.

Recipedia Power Supply Design Procedures and Testing

The design of our power supply was based on the needs of our microcontroller and its voltage thresholds for correct operation. Our controller, the Motorola MC9S08, requires between 1.8 – 3.6 V in order to operate correctly. Therefore, we decided to design a power supply capable of providing a steady direct current (DC) voltage supply of 3.3 volts. To meet this requirement, we used the basic building blocks for a standard AC-DC power supply which includes a transformer, a rectifier circuit, filter, and a voltage regulator. Also, we designed our supply to provide up to approximately 1 amp for any

load. This was accomplished by choosing a transformer capable of providing a maximum current, with a load, of 1.2 amps. Table 1 below illustrates the components we chose for the design of our circuit. Figure 4 is our circuit schematic for our power supply.

Table 1. Components

Component Name (Reference)	Value
Transformer (T1)	12.6 V/1.2 amp
Capacitor (C1)	Electrolytic 1000µF/25V
Capacitor (C2)	Ceramic 100nF
Capacitor (C3)	Electrolytic 100µF/50V
Capacitor (C4)	Ceramic 100nF
Adjustable Voltage Regulator IC (U1)	LM 350 T
Resistor (R1)	240 Ω
Resistor (R2)	400 Ω
Bridge Rectifier (D1)	NA

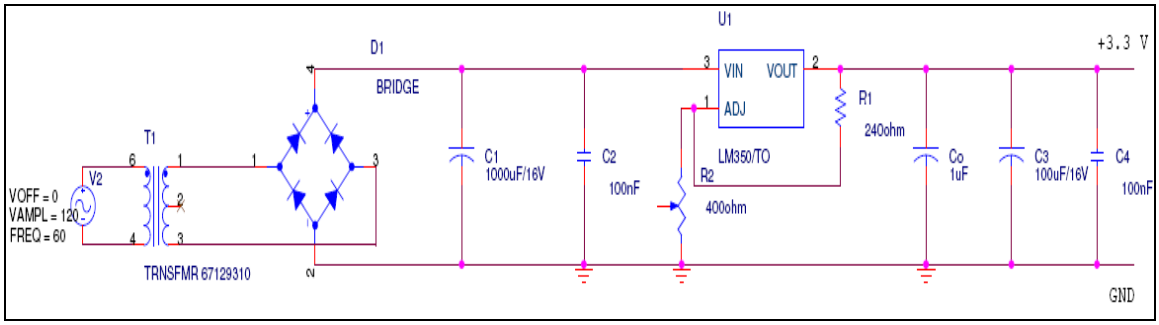


Figure 4. 3.3 V Power Supply Schematic

Using the Adjustable Voltage Regulator:
The LM350T regulator IC we are using requires adjusting according to our desired output voltage. By using the datasheet for this component, we were able to determine at what value resistor R2 would be necessary to produce an output of 3.3 volts. Below is the schematic for the LM350T as well as the solution to its equation to produce R2.

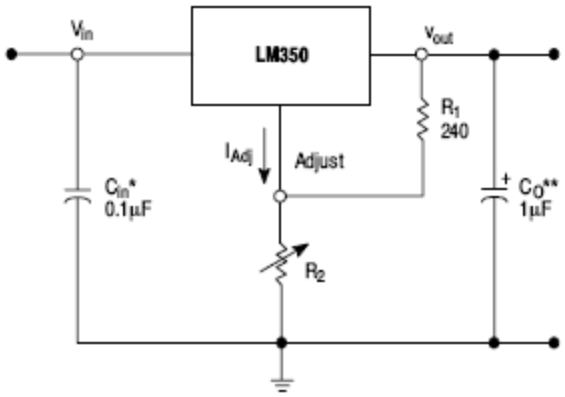


Figure 5. LM350T Adjustable Voltage Regulator Integrated Circuit

Find Value for R2:

The equation for V_{out} is,

$$V_{out} = 1.25\ V\left(1 + \frac{R_2}{R_1}\right) + I_{Adj}R_2$$

We can remove the term I_{Adj} from the equation because it is controlled to less that 100 μA and can be ignored in most applications. So that gives us,

$V_{out} = 1.25\ V * (1 + R_2/R_1)$

Now we can solve for R_2, since we know what output voltage we desire and R_1 is always set to 240 Ω. That gives us the following,

$3.3\ V = 1.25\ V * (1 + R_2/240\ \Omega)$.

Distribute the 1.25 V through,

$3.3\ V = 5.2083 * 10^{-3} * (R_2 + 240\ \Omega)$

Now divide both sides,

$3.3\ V / 5.2083 * 10^{-3} = 5.2083 * 10^{-3} * (R_2 + 240\ \Omega) / 5.2083 * 10^{-3}$

This result is

$633.6 = (R_2 + 240\ \Omega)$.

Subtract both sides by 240 Ohms,

$633.6 - 240 = (R_2 + 240\ \Omega) - 240$

This yields our solution,

$393.6\ \Omega = R_2$.

Testing Procedures:

Once a correct value for the resistor R_2 was found, prototyping of the circuit could be completed through the use of a solder-less breadboard. Using the schematic above in figure 4, the circuit was built. Before using the supply directly with the microprocessor however, we needed to conduct tests to ensure that the circuit was correct and producing our desired output voltage. We began by measuring the output voltage of the circuit with a standard voltmeter. This produced an output voltage of approximately 3.4 V. Once this measurement was taken, we connected a common LED and resistor to provide a small load for the circuit and also to be used as a signal that the supply was on. We also monitored that the circuit component temperatures were not becoming unstable. Once this step was completed we used an oscilloscope in the lab to check the value of the output voltage and to monitor the stability of the output. From the scope we received a steady DC voltage value of approximately 3.45 volts. Now that our circuit produced valid results we conducted tests to ensure that the circuit could maintain and tolerate a substantial amount of time operating. All tests mentioned above proved successful. We were now confident that our circuit would be a reliable and efficient source of power for our microprocessor.

High-Gain Amplifier:

A design of a high-gain amplifier was also conducted in order to amplify the signal outputted from our SP03 Text-to-Speech Module and inputted into a standard 8 Ohm speaker. Although the design and construction of this circuit was completed it was not implemented in the final project due to voltage requirements and time limitations. For this circuit we used a design consisting of two inverting op amp stages which produces a voltage output of 2500 times the input voltage. This design was replicated from figure 11.42 in the text Embedded Micro- computer Systems by Jonathan Valvano. Figure 6 below contains this schematic.

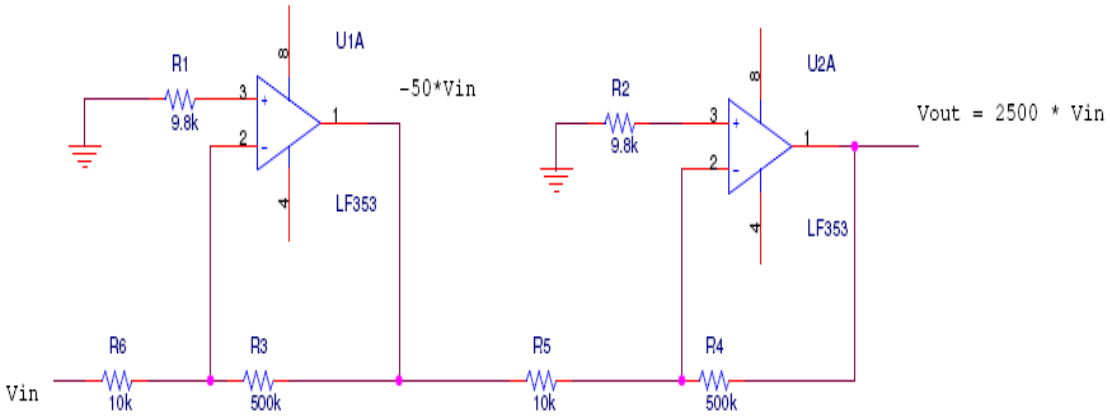


Figure 6. High-Gain Amplifier

After the circuit was built according to the schematic above, we began testing through the use of an oscilloscope and a wave generator. A sinusoidal wave input was produced using the wave generator and inputted into the V_{in} of the amplifier. This produced the waveform in figure 7.

Using the gain equation

Gain = Vout / Vin,

We produce

Gain = 84.38 V / 215.6 mV

Therefore,

Gain = 391.37

Now that our circuit was operating, we tested it in conjunction with a speaker and input wave. This produced a varying sound wave at the speaker in relation to the value of the input wave frequency. Hence, at higher frequencies the output would result in a higher pitch. Although our circuit was tested and designed, we were not able to use it in our final design. This is due to the use of a LM348N dip package containing four operational amplifiers. These particular amplifiers require an operational voltage of 16 volts to operate correctly. Because of this requirement we were not able to use our amplifier for our project due to the lack of a viable 16 volt power supply and time limitations.

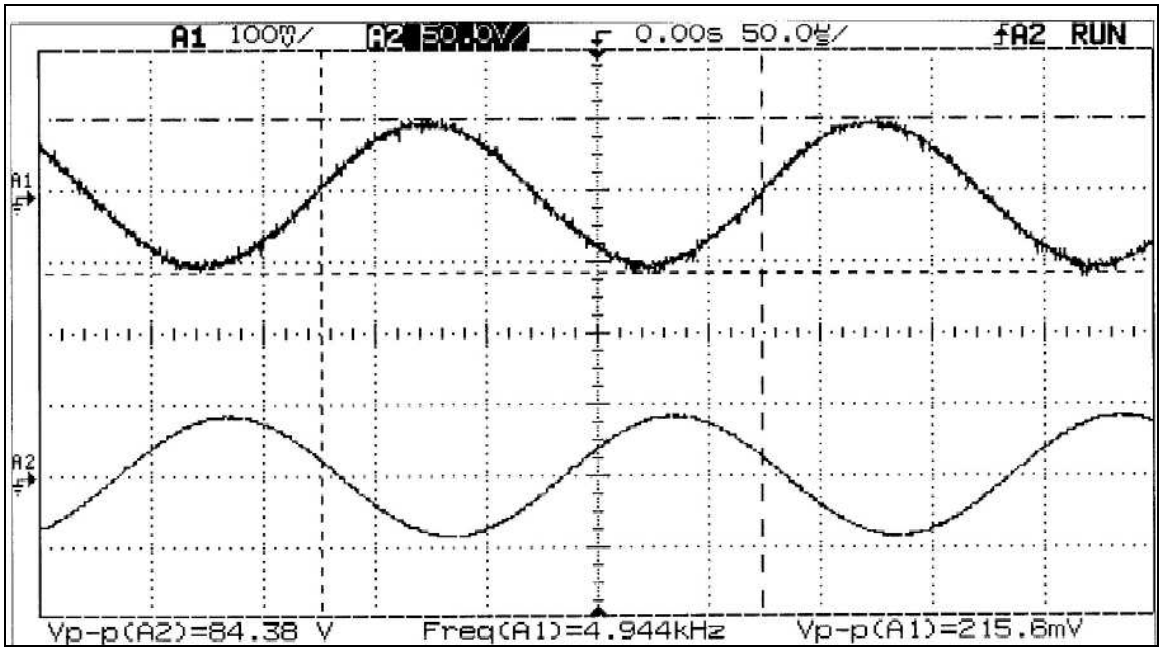


Figure 7. Oscilloscope Capture of Amplifier

Problems Encountered:

The majority of the problems we faced with the power supply were caused by component operating specifications. With the amplifier it was necessary to provide a voltage of ±16 V for the op amps to function correctly. However because the laboratory did not provide a power supply capable of producing this voltage and we did not have time to design a viable power supply and we were not able to implement our amplifier in the final design

of our project. Our original design of the power supply also consisted of a 5 volt supply, not a 3.3 volt supply. The microprocessor we chose for our project required operating voltages between 1.8 – 3.6 volts. This resulted in having to design a supply that included an adjustable voltage regulator capable of producing these voltages. Once this was implemented our power supply functioned correctly as per specification.

Future Work

One major enhancement that we would have liked to do was to have a Compact Flash card to store all the recipe text and images. This could be done by writing a IDE host controller on the FPGA to access the data on the CF card. This future work would allow the user to update and add recipes to the system from their PC. Recipe CF cards could also be sold and could be inserted into the system easily. Another major enhancement that we just didn't have time to finish would have been simple voice command recognition for "Next", "Back" and "Home" commands. This way once a user has selected a recipe she/he can say any of the commands to navigate through the recipe lines. Other useful additions would be a timer and clock.

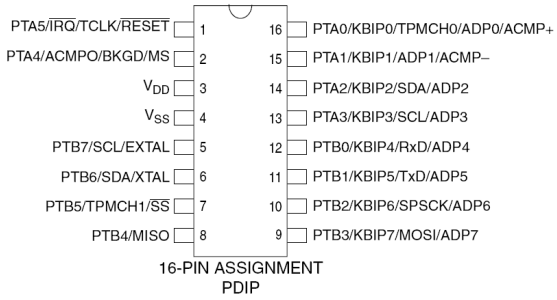
Conclusion

Overall, this has been a very challenging and educational learning experience. We have definitely run into some problems along the way, as mentioned in the technical specifications, some of our major problems we encountered were stack overflow issues, where we were getting corrupt data because the size and order of our C structures**.** We solved this problem late into the semester and we were not able to have the size and number of recipes we had initially intended. Time also was a problem for us when it came to implementing our filter, amp and voice command recognition. We all learned different things from this project but as a team we definitely have gained a much better understanding of microcontrollers, how to program them and their limitations and capabilities. We have learned a lot about how touch screens, voice frequency filters and how to make use of their technologies in a system.

Bill of Materials

Microcontrollers

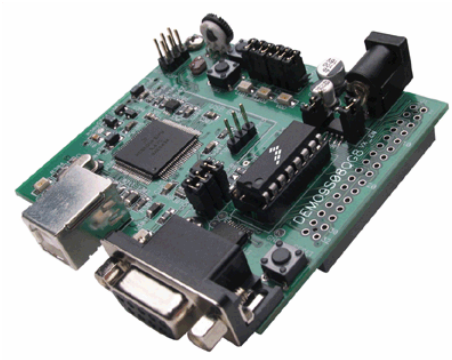
• Primary Vendor: Freescale
• Model #: HCS08
• Part #: HCS08QG8
• Unit Cost: Free sample (5)
• Quantity: 5
• Total Cost: $0.00



16-PIN ASSIGNMENT
PDIP

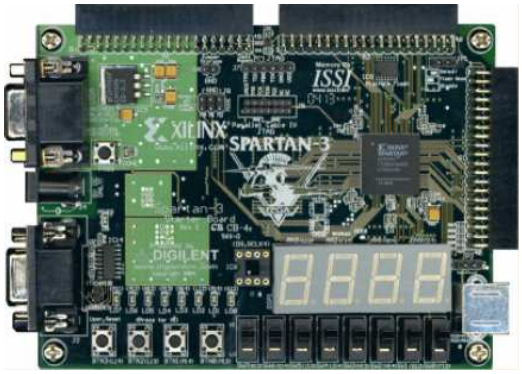
Demo Board (Programmer)

• Primary Vendor: Freescale
• Model #: MC9S08QH8
• Part # DEMO9s08QG8
• Unit Cost: $50.00
• Quantity: 1
• Total Cost: $50.00 + $6.00 Shipping



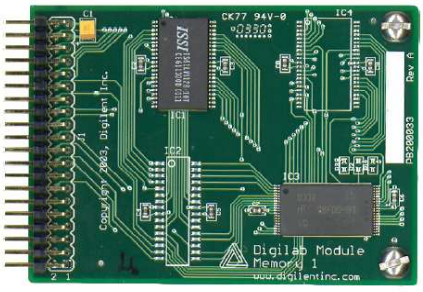
FPGA

• Primary Vendor: www.digilentinc.com
• Model #: Spatan3 starter kit
• Part #: XCS3S200
• Unit Cost: $99.00 (own 2)
• Quantity: 1
• Total Cost: $99.00 + shipping

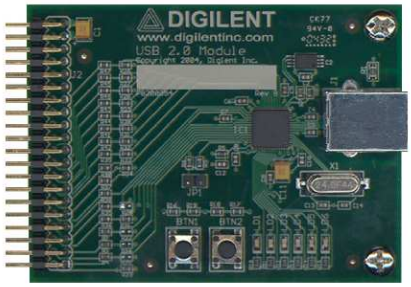


Memory Module

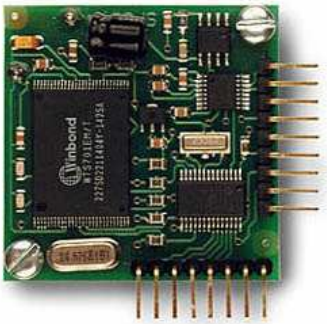
• Primary Vendor: www.digilentinc.com
• Model #: MEMC1
• Part #: MEMC1
• Unit Cost: $47.95
• Quantity: 1
• Total Cost: $47.95 + $6.00 shipping



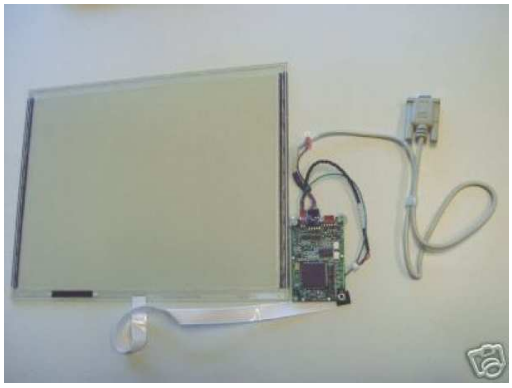
USB2 Module
• Primary Vendor: www.digilentinc.com
• Model #: USB2
• Part #: USB2
• Unit Cost: $48.95
• Quantity: 1
• Total Cost: $48.95 + $6.00 shipping



SP03 Text to Speech
• Primary Vendor: www.hobbyengineering.com
• Model #: SPO3
• Part #: R184SP03
• Lead Time: 2 weeks (In-stock)
• Unit Cost: $99.00
• Quantity: 1
• Total Cost: $99.00 + Shipping



Touch Screen
• Primary Vendor: eBay
• Model #: SCN-AT
• Part #: E274
• Lead Time: None (Purchased)
• Unit Cost: $56.00
• Quantity: 1
• Total Cost: $70.00 ($56.00 + 14.00 shipping)



Miscellaneous Analog Components
• Primary Vendor: UofU Digital Lab
• Total Cost: $10.00

System Enclosure
For our system's enclosure we constructed a large wooden box structure, with a front window for our monitor screen and a removable lid. We installed Formica on the sides, front, and lid to give the continuous feel of a kitchen countertop.

References

HCS08 Quick Reference Guide
www.Freescale.com

HCS08 Reference Manual
www.Freescale.com

MC9S08QG8 Data Sheet
www.Freescale.com

ELO Touch Screen Manual
www.elotouch.com
written by ELO

ELO Driver Manual
www.elotouch.com
written by ELO

Embedded Microcomputer Systems:
Real time interfacing, by Jonathan W. Valvano

VGA information
www.xcess.org
written by Xcess

C code for serial drivers
http://www.cs.utah.edu/~tspens/cs3992/files/serial_c.gif

Additional FFT resources include:
http://www.fftw.org/links.html

Good Verilog tutorials
http://www.engr.sjsu.edu/crabill/

Appendix

HCS08 code:
main.c

```
#include <hidef.h>
#include <string.h>
#include "derivative.h"
#include "recipe.h"


screen_place_s screen_loc;         // used to update FPGA bmp
recipe_line_s * CURR = NULL;
byte ReceivedBuffer[MAX_BUFFER_SIZE];
byte IIC_DATA[80];

byte ln = 255;

byte recipeNumber = 0;
byte LOC = HOME;
byte input = 0;           // input flag
byte buffer_index = 0;    // before correcting data
byte done = 1;            // main job's flag

byte num = 0;
byte ans = 0;
byte z = 0;               //global counter

//----Touch screen coordinates-------------------------------------------------------------
----------
byte start = 0;
byte  x0 = 0;
byte  x1 = 0;
byte  y0 = 0;
byte  y1 = 0;

//----IIC Variables ------------------------------------------------------------------------
----------
byte IIC_STEP = IIC_READY_STATUS;
byte IIC_DATA_DIRECTION = 0;      // 1 Transmit, 0 Read
byte IIC_LENGTH = 1;
byte IIC_COUNTER = 0;
byte SP03_ADDRESS = 0xC4;

//#pragma DATA_SEG DATA1
recipe_line_s Home;
//#pragma DATA_SEG DATA2
recipe_line_s Dinner;
recipe_line_s Dessert;
//recipe_line_s Breakfast;

//----Recipes ------------------------------------------------------------------------------
--------
recipe_line_s Curry_Chicken;
recipe_line_s Golden_Chicken;
//recipe_line_s Chicken_Cordon_Blue;

recipe_line_s Almond_Torte;
//recipe_line_s Raspberry_Chocolate_Cake;
//recipe_line_s Pumpkin_Chocolate_Chip_Cookies;

//recipe_line_s Pancakes;
//recipe_line_s Waffles;
//recipe_line_s Omelettes;


//----Initializations ----------------------------------------------------------------------
----------
void configureIIC(void);
void initializeSCI(void);
```

```
void init_screen_loc(void);
void init_flags(void);
void null_all(recipe_line_s * recipe);

//----SP03 Functions ------------------------------------------------------------------
----------
void welcome(void);
void speakSP03(byte,byte*);
void speakPredefined(byte);
/*Predefined phrases
    1 welcome to Recipedia the digital cook book
    2 dinner
    3 dessert
    4 breakfast
    5 back
    6 home
    7 next
*/
void speechHeader(byte);
void speechFooter(byte);
void speakLine(byte*,byte);
void clearPending(void);
void initializeTransmit(void);

//----Navigation Functions ------------------------------------------------------------
----------
void doHOMEthings(byte m, byte b, byte md);
void doDINNERthings(byte m, byte b, byte md);
void doDESSERTthings(byte m, byte b, byte md);
void doBREAKFASTthings(byte m, byte b, byte md);
void doRECIPEthings(byte m, byte b, byte md);

byte MidPressed(void);
byte BotPressed(void);

byte num_chars(byte * c);                        // counts the number of charachters in
a recipe line

//----Configure FPGA interface --------------------------------------------------------
----------
void configFPGA(void);
void updateFPGA(void);

// This delay function can be used to disable (ADC) command recognition on the other uc
void delay(unsigned int cycle);

//----Recipies just for testing-------------------------------------------------------
----------
//byte testPhrase[22] = "This is a test phrase";  // Extra byte for zero terminator
//byte *testPhrasePtr = testPhrase;


//----Main loop -----------------------------------------------------------------------
----------
void main(void) {

  Home.lines[0] = "Dinner";
  Home.lines[1] = "Dessert";
  Home.lines[2] = "Breakfast";

  Dinner.lines[0] = "Curry Chicken";
        Dinner.lines[1] = "Golden Chicken";
        //Dinner.lines[2] = "Chicken Cordon Blue";

        Dessert.lines[0] = "Allmund Torte";
        //Dessert.lines[1] = "Raspberry Chocolate Cake";
        //Dessert.lines[2] = "Pumpkin Chocolate Chip Cookies";

        //Breakfast.lines[0] = "Pancakes";
        //Breakfast.lines[1] = "Waffles";
        //Breakfast.lines[2] = "Omelettes";
```

```
//----Dinners -------------------------------------------------------------
  Curry_Chicken.lines[0] = "1 box of ehs & B Golden Curry blocks";
        Curry_Chicken.lines[1] = "1 pound cut chicken or shrimp";
        Curry_Chicken.lines[2] = "1 chopped onion";
        Curry_Chicken.lines[3] = "3 sliced or 2 cups baby carrots";
        Curry_Chicken.lines[4] = "1 or 2 potatoes cut into squares";
        Curry_Chicken.lines[5] = "Saute meat and onions for 10 minutes";
        Curry_Chicken.lines[6] = "Add 3 cups water and bring to a boil";
        Curry_Chicken.lines[7] = "Break curry blocks and add";
        Curry_Chicken.lines[8] = "Add carrots and potatoes and boil on low for 30
minutes";
        Curry_Chicken.lines[9] = "Serve over rice";

  Golden_Chicken.lines[0] = "4 chicken breasts cut into bite sized pieces";
        Golden_Chicken.lines[1] = "2 cans of Golden Mushroom Soup";
        Golden_Chicken.lines[2] = "2 cups sliced or baby carrots";
  Golden_Chicken.lines[3] = "1 medium onion chopped";
        Golden_Chicken.lines[4] = "A pinch of nutmeg";
        Golden_Chicken.lines[5] = "About one half cup chicken broth";
        Golden_Chicken.lines[6] = "Saute the chicken and the onion for 10 minutes";
        Golden_Chicken.lines[7] = "Add all, and cook for 30 minutes on medium heat";
        Golden_Chicken.lines[8] = NULL;
        Golden_Chicken.lines[9] = NULL;
        /*
        Chicken_Cordon_Blue.lines[0] = "Chicken Cordon Blue 1";
        Chicken_Cordon_Blue.lines[1] = "Chicken Cordon Blue 2";
        Chicken_Cordon_Blue.lines[2] = "Chicken Cordon Blue 3";
        Chicken_Cordon_Blue.lines[3] = "Chicken Cordon Blue 4";
        Chicken_Cordon_Blue.lines[4] = "Chicken Cordon Blue 5";
        Chicken_Cordon_Blue.lines[5] = "Chicken Cordon Blue 6";
        Chicken_Cordon_Blue.lines[6] = "Chicken Cordon Blue 7";
        Chicken_Cordon_Blue.lines[7] = "Chicken Cordon Blue 8";
        Chicken_Cordon_Blue.lines[8] = "Chicken Cordon Blue 9";
        Chicken_Cordon_Blue.lines[9] = "Chicken Cordon Blue 10";
        */
//----Desserts -------------------------------------------------------------
  Almond_Torte.lines[0] = "1 cup sugar";
  Almond_Torte.lines[1] = "1 cup flour";
  Almond_Torte.lines[2] = "Mix flour and sugar";
  Almond_Torte.lines[3] = "Add 2 eggs";
  Almond_Torte.lines[4] = "Add 1 teaspoon almond extract";
  Almond_Torte.lines[5] = "Mix well and pour into a foil lined 9 inch pie pan";
  Almond_Torte.lines[6] = "Sprinkle sliced almonds on top and bake at 375 degrees for 20
minutes";
  Almond_Torte.lines[7] = NULL;
  Almond_Torte.lines[8] = NULL;
  Almond_Torte.lines[9] = NULL;
        /*
  Raspberry_Chocolate_Cake.lines[0] = "Raspberry Chocolate Cake 1";
  Raspberry_Chocolate_Cake.lines[1] = "Raspberry Chocolate Cake 2";
  Raspberry_Chocolate_Cake.lines[2] = "Raspberry Chocolate Cake 3";
  Raspberry_Chocolate_Cake.lines[3] = "Raspberry Chocolate Cake 4";
  Raspberry_Chocolate_Cake.lines[4] = "Raspberry Chocolate Cake 5";
  Raspberry_Chocolate_Cake.lines[5] = "Raspberry Chocolate Cake 6";
  Raspberry_Chocolate_Cake.lines[6] = "Raspberry Chocolate Cake 7";
  Raspberry_Chocolate_Cake.lines[7] = "Raspberry Chocolate Cake 8";
  Raspberry_Chocolate_Cake.lines[8] = "Raspberry Chocolate Cake 9";
  Raspberry_Chocolate_Cake.lines[9] = "Raspberry Chocolate Cake 10";

  Pumpkin_Chocolate_Chip_Cookies.lines[0] = "Pumpkin Chocolate Chip Cookies 1";
  Pumpkin_Chocolate_Chip_Cookies.lines[1] = "Pumpkin Chocolate Chip Cookies 2";
  Pumpkin_Chocolate_Chip_Cookies.lines[2] = "Pumpkin Chocolate Chip Cookies 3";
  Pumpkin_Chocolate_Chip_Cookies.lines[3] = "Pumpkin Chocolate Chip Cookies 4";
  Pumpkin_Chocolate_Chip_Cookies.lines[4] = "Pumpkin Chocolate Chip Cookies 5";
  Pumpkin_Chocolate_Chip_Cookies.lines[5] = "Pumpkin Chocolate Chip Cookies 6";
  Pumpkin_Chocolate_Chip_Cookies.lines[6] = "Pumpkin Chocolate Chip Cookies 7";
  Pumpkin_Chocolate_Chip_Cookies.lines[7] = "Pumpkin Chocolate Chip Cookies 8";
  Pumpkin_Chocolate_Chip_Cookies.lines[8] = "Pumpkin Chocolate Chip Cookies 9";
  Pumpkin_Chocolate_Chip_Cookies.lines[9] = "Pumpkin Chocolate Chip Cookies 10";
```

```
//----Breakfasts -------------------------------------------------------
  Pancakes.lines[0] = "Pancakes 1";
  Pancakes.lines[1] = "Pancakes 2";
  Pancakes.lines[2] = "Pancakes 3";
  Pancakes.lines[3] = "Pancakes 4";
  Pancakes.lines[4] = "Pancakes 5";
  Pancakes.lines[5] = "Pancakes 6";
  Pancakes.lines[6] = "Pancakes 7";
  Pancakes.lines[7] = "Pancakes 8";
  Pancakes.lines[8] = "Pancakes 9";
  Pancakes.lines[9] = "Pancakes 10";

  Waffles.lines[0] = "Waffles 1";
  Waffles.lines[1] = "Waffles 2";
  Waffles.lines[2] = "Waffles 3";
  Waffles.lines[3] = "Waffles 4";
  Waffles.lines[4] = "Waffles 5";
  Waffles.lines[5] = "Waffles 6";
  Waffles.lines[6] = "Waffles 7";
  Waffles.lines[7] = "Waffles 8";
  Waffles.lines[8] = "Waffles 9";
  Waffles.lines[9] = "Waffles 10";

  Omelettes.lines[0] = "Omelettes 1";
  Omelettes.lines[1] = "Omelettes 2";
  Omelettes.lines[2] = "Omelettes 3";
  Omelettes.lines[3] = "Omelettes 4";
  Omelettes.lines[4] = "Omelettes 5";
  Omelettes.lines[5] = "Omelettes 6";
  Omelettes.lines[6] = "Omelettes 7";
  Omelettes.lines[7] = "Omelettes 8";
  Omelettes.lines[8] = "Omelettes 9";
  Omelettes.lines[9] = "Omelettes 10";
  */

  EnableInterrupts;
  LOC = HOME;
       CURR = NULL;

  //Turn on a green led indicating program is running
  //PTB5 will probably need to be used also as the indicator to disable the
  //ADC on the other uc we could use PTB5 as a red led indicating when the
  //text-to-speech is speaking.  We could make a timer interrupt every second
  //so we could blink the led
  PTBDD_PTBDD5 = 1;
  PTBD_PTBD5 = 1;

  initializeSCI();
  configureIIC();
  configFPGA();
       init_screen_loc();
  init_flags();

  // speaks predefined phrase 1, "Welcome to recipedia, the digital cook book!"
  // this is already done when the text-to-speech chip is powered on
  //welcome();

  // call this function to speak a line of a recipe by passing in a pointer to a recipe
line
  // and the number of charachters in that line
  //speakSP03(testPhrasePtr,22);
  // these functions are not necessary if we run out of code space

  //loop here forever
  while(1){
    //wait for touch on touch-screen

    if((buffer_index >= MAX_BUFFER_SIZE) && !input && (ReceivedBuffer[2] == 4)){

      x0 = ReceivedBuffer[4];  //1
```

```
      x1 = ReceivedBuffer[3];  //0
      y0 = ReceivedBuffer[6];  //3
      y1 = ReceivedBuffer[5];  //2

      // Middle button pressed?  This is one of the 'round' buttons
      ans = MidPressed();
      if(ans){
        screen_loc.round_buttons = ans;
        screen_loc.button_event = ROUND_CHANGED;
        input = 1;
      }

      // Bottom button pressed?  This is one of the 'square' buttons
      else{
        ans = BotPressed();
        if(ans){
          screen_loc.square_buttons = ans;
          screen_loc.button_event = SQUARE_CHANGED;
          input = 1;
        }else{
                            buffer_index = 0;
                            start = 0;
                    }
      }
    }

    //wait for input or done, done meaning we executed a valid button press.
               if(input && done){
                       done = 0;
                       if(LOC == HOME){
                               doHOMEthings(screen_loc.round_buttons,
screen_loc.square_buttons, screen_loc.button_event);
                       }else if (LOC == DINNER){
                               doDINNERthings(screen_loc.round_buttons,
screen_loc.square_buttons, screen_loc.button_event);
                       }else if (LOC == DESSERT){
                               doDESSERTthings(screen_loc.round_buttons,
screen_loc.square_buttons, screen_loc.button_event);
                       }else if (LOC == BREAKFAST){
                               doBREAKFASTthings(screen_loc.round_buttons,
screen_loc.square_buttons, screen_loc.button_event);
                       }else if(LOC == RECIPE){
                               doRECIPEthings(screen_loc.round_buttons,
screen_loc.square_buttons, screen_loc.button_event);
                       }
                }
          __RESET_WATCHDOG(); // feeds the dog
       }//end while
       return;
}

//----Initializations ---------------------------------------------------------------------
----------
// Function to configure the IIC module.
void configureIIC(){
  SOPT2_IICPS = 1;       // Use PTB6(SDA) and PTB7(SCL)
  IICC_IICEN = 1;        // Enable IIC
  IICC_IICIE = 1;
  IICA = SP03_ADDRESS;   // IIC Address
  IICF = 0x4B;           // Set IIC frequency 100kbps
  IIC_STEP = IIC_READY_STATUS;
}

void initializeSCI(){
  PTBPE_PTBPE0 = 0;       //dissable pull up on PTB0
  SCIBDH = 0x00;
  SCIBDL = 0x1A;          // The baud rate is 9600bps

  SCIC1  = 0x00;          // SCIC1: LOOPS=0,SCISWAI=0,RSRC=0,M=0,WAKE=0,ILT=0,PE=0,PT=0
  SCIC2  = 0x2C;          // SCIC2: TIE=0,TCIE=0,RIE=1,ILIE=0,TE=1,RE=1,RWU=0,SBK=0 //
  SCIC3  = 0x00;          // SCIC3: R8=0,T8=0,TXDIR=0,TXINV=0,ORIE=0,NEIE=0,FEIE=0,PEIE=0
```

```
  z = 0;
  while(z<10){
    ReceivedBuffer[z] = 0;
    z++;
  }
}

void init_screen_loc(){
       screen_loc.round_buttons = DEFAULT;
       screen_loc.square_buttons = DEFAULT;
       screen_loc.button_event = DEFAULT;
}

void init_flags(){
  done = 1;
       input = 0;
       buffer_index = 0;
  start = 0;
}

void null_all(recipe_line_s * recipe){
  for(z = 0; z < 10; z++){
    (recipe->lines[z]) = NULL;
  }
}



//----Check press location -------------------------------------------------------------
----------
byte MidPressed(void){
  byte xrange = 0;
  //---------first check the MIDDLE value
  if((x0 >= 0x03) && (x0 <= 0x05)){
        xrange = 1;
  }

  if(xrange){
    //--------- for MIDDLE0
    if((y0 >= 0x0A) && (y0 <= 0x0C)){
      return MIDDLE0;
    }
    //------------- for MIDDLE1
    if((y0 >= 0x08) && (y0 <= 0x0A)){
      return MIDDLE1;
    }
    //----------------for MIDDLE2
    if((y0 >= 0x06) && (y0 <= 0x08)){
      return MIDDLE2;
    }
  }
  return 0;
}

//----Check press location -------------------------------------------------------------
----------
byte BotPressed(void){
  byte xrange = 0;
  byte yrange = 0;

  //-----------next check the BOTTOM value
  if((y0 >= 0x02) && (y0 <= 0x04)){
    yrange = 1;
  }

  if(yrange){
   //--------- for BOTTOM0
    if((x0 >= 0x03) && (x0 <= 0x06)){
      return BOTTOM0;
    }
```

```
    //--------- for BOTTOM1
    if((x0 >= 0x07) && (x0 <= 0x0A)){
      return BOTTOM1;
    }
    //--------- for BOTTOM2
    if((x0 >= 0x0A) && (x0 <= 0x0E)){
      return BOTTOM2;
    }
  }//end of yrange for BOTTOM
  return 0;
}

//----Screen/Navigation functions ------------------------------------------------------
----------
void doHOMEthings(byte m, byte b, byte md){
       //byte i=0;
       if(md == ROUND_CHANGED){
              if(m == MIDDLE0){
         //display Dinner Menu
         recipeNumber = 1;
         updateFPGA();
              LOC = DINNER;
              speakPredefined(2);
              }
              else if(m == MIDDLE1){
         //display Dessert Menu
         recipeNumber = 2;
         updateFPGA();
              LOC = DESSERT;
              speakPredefined(3);
              }
              else if(m == MIDDLE2){
         //display Breakfast Menu, not yet implemented on FPGA due to memory constraints
         recipeNumber = 3;
         //updateFPGA();
              //LOC = BREAKFAST;
              //speakPredefined(4);
              }
       }
       init_screen_loc();
       init_flags();
}

void doDINNERthings(byte m, byte b, byte md){
       if(md == ROUND_CHANGED){
              LOC = RECIPE;
              if(m == MIDDLE0){
                //display Curry Chicken
                recipeNumber = 3;
                updateFPGA();
                     CURR = &Curry_Chicken;
                     if(CURR != NULL){
        num = num_chars(Dinner.lines[0]);
        speakSP03(num, Dinner.lines[0]);
      }
              }
              else if(m == MIDDLE1){
                //display Golden Chicken
                recipeNumber = 4;
                updateFPGA();
                     CURR = &Golden_Chicken;
                     if(CURR != NULL){
        num = num_chars(Dinner.lines[1]);
        speakSP03(num, Dinner.lines[1]);
      }
              }
              else if(m == MIDDLE2){
                //display Chicken Cordon Blue, not yet implemented on FPGA due to memory
constraints
                recipeNumber = 5;
      //updateFPGA();
```

```
                        //CURR = &Chicken_Cordon_Blue;
                        //if(CURR != NULL){
      //  num = num_chars();
      //  speakSP03(num, CURR->lines[ln]);
      //}
                }
        }
        else if(md == SQUARE_CHANGED){
                if(b == BOTTOM1){

                  recipeNumber = 0;
                  updateFPGA();
                        LOC = HOME;
                        CURR = NULL;
                        speakPredefined(6);
                        ln = 255;
                }
        }
        init_screen_loc();
        init_flags();
}

void doDESSERTthings(byte m, byte b, byte md){
        if(md == ROUND_CHANGED){
                LOC = RECIPE;
                if(m == MIDDLE0){

                          recipeNumber = 5;
                          updateFPGA();
                                CURR = &Almond_Torte;
                        if(CURR != NULL){
          num = num_chars(Dessert.lines[0]);
          speakSP03(num, Dessert.lines[0]);
         }
                }
                else if(m ==   MIDDLE1){

                          recipeNumber = 2;
                          //updateFPGA();
                                //CURR = &Raspberry_Chocolate_Cake;
                                //if(CURR != NULL){
        //  num = num_chars();
        //  speakSP03(num, CURR->lines[ln]);
        //}
                }
                else if(m == MIDDLE2){

                          recipeNumber = 2;
                          //updateFPGA();
                                //CURR = &Pumpkin_Chocolate_Chip_Cookies;
                                //if(CURR != NULL){
        //  num = num_chars();
        //  speakSP03(num, CURR->lines[ln]);
        //}
                }
        }
        else if(md == SQUARE_CHANGED){
                if(b == BOTTOM1){
                  recipeNumber = 0;
                  updateFPGA();
                        LOC = HOME;
                        CURR = NULL;
                        speakPredefined(6);
                        ln = 255;
                }
        }
        init_screen_loc();
        init_flags();
}

void doBREAKFASTthings(byte m, byte b, byte md){
```

```
  //recipeNumber = 0;
  //updateFPGA();
  LOC = HOME;
  CURR = NULL;
}
/*byte i=0;
        if(md == ROUND_CHANGED){
                LOC = RECIPE;
                switch (m){
                        case MIDDLE0:
                                CURR = &Pancakes;
                                i=speak_line(&Pancakes);
                                break;
                        case MIDDLE1:
                                CURR = &Waffles;
                                i=speak_line(&Waffles);
                                break;
                        case MIDDLE2:
                                CURR = &Omelettes;
                                i=speak_line(&Omelettes);
                                break;
                        default:
                                break;
                }
        }
        else if(md == SQUARE_CHANGED){
                switch(b){
                        case BOTTOM1:
                                LOC = HOME;
                                CURR = NULL;
                                speakPredefined(6);
                                //i=speak_line(&Home);
                                break;
                        default:
                                break;
                }
        }
        init_screen_loc();
        done = 1;
        input = 0;
  */

void doRECIPEthings(byte m, byte b, byte md){
        if(md == SQUARE_CHANGED){
        if(b == BOTTOM0){
       ln--;
       if(ln >= 0 && ln < 10){
         if(CURR != NULL){
             num = num_chars(CURR->lines[ln]);
           speakSP03(num, CURR->lines[ln]);
         }
       }
       else {
          ln = 0;
          num = num_chars(CURR->lines[ln]);
          speakSP03(num, CURR->lines[ln]);
       }
        }
        else if(b == BOTTOM1){
       recipeNumber = 0;
       updateFPGA();
        LOC = HOME;
        CURR = NULL;
        speakPredefined(6);
        ln = 255;
        }
        else if(b == BOTTOM2){
       ln++;
       if(ln <= 9 && (CURR->lines[ln] != NULL)) {
         if(CURR != NULL){
           num = num_chars(CURR->lines[ln]);
```

```
          speakSP03(num, CURR->lines[ln]);
         }
       }
       else{
         ln = ln--;
         num = num_chars(CURR->lines[ln]);
         speakSP03(num, CURR->lines[ln]);
       }
        }
        }
        init_screen_loc();
        init_flags();
}

//----SP03 and IIC functions ----------------------------------------------------------
----------
//function to count the number of characters in one line of a recipe
byte num_chars(byte * c){
  for(z = 0; *c != '\0'; z++){
    c++;
  }
  /*
  z = 0;
  while(((CURR->lines[ln])[z]) != 0){
    z++;
  }
  */
  z++;
  return z;
}

void welcome(){
  speakPredefined(1);
  while(IIC_STEP>IIC_READY_STATUS)__RESET_WATCHDOG(); // wait for memory to be read
  speechFooter(2);
  while(IIC_STEP>IIC_READY_STATUS)__RESET_WATCHDOG(); // wait for memory to be read
}

void speakSP03(byte numberOfBytes,byte * recipePtr){
  speechHeader(5);
  while(IIC_STEP>IIC_READY_STATUS)__RESET_WATCHDOG(); // wait for memory to be read
  speakLine(recipePtr,numberOfBytes);
  while(IIC_STEP>IIC_READY_STATUS)__RESET_WATCHDOG(); // wait for memory to be read
  speechFooter(2);
  while(IIC_STEP>IIC_READY_STATUS)__RESET_WATCHDOG(); // wait for memory to be read

}

// Functions to speak Predefined or General phrase
void speakPredefined(byte predefined){
  clearPending();

        IIC_DATA[0]=0x00;
        IIC_DATA[1]=predefined;   // Predefined Phrase

        IIC_LENGTH = 2;
  initializeTransmit();
}

void speakLine(byte *testPhrasePtr,byte numberOfBytes) {
  byte i,j;
  numberOfBytes++;
  numberOfBytes++;
  clearPending();

        IIC_DATA[0]=0x00;
        IIC_DATA[1]=0x00;

  j=0;
  i=2;
        while(testPhrasePtr[j] != '\0') {
```

```
          IIC_DATA[i] = testPhrasePtr[j];
          j++;
          i++;
        }

        IIC_DATA[i]='\0';

        while(i<80) {
    IIC_DATA[i] = 0;
    i++;
  }

        IIC_LENGTH = numberOfBytes;
  initializeTransmit();
}

void speechHeader(byte numberOfBytes){
  clearPending();

        IIC_DATA[0]=0x00;
        IIC_DATA[1]=0x00;
        IIC_DATA[2]=0x00; // Volume 00=MAX
        IIC_DATA[3]=0x05; // Speed
        IIC_DATA[4]=0x03; // Pitch
        IIC_DATA[5]=0x00;

        IIC_LENGTH = numberOfBytes;
  initializeTransmit();
}

void speechFooter(byte numberOfBytes){
  clearPending();

        IIC_DATA[0]=0x00;
        IIC_DATA[1]=0x40;

        IIC_LENGTH = numberOfBytes;
  initializeTransmit();
}

// Clear any pending IIC interrupt
void clearPending(void){
  IICC_IICEN = 0;
  IICC_IICEN = 1;
  IICS;
}

// Initialize IIC transmit
void initializeTransmit(){
  IIC_COUNTER =0;
  IIC_STEP = IIC_HEADER_SENT_STATUS;
  IIC_DATA_DIRECTION = 1;

  IICC_IICIE = 1;
  IICC_MST = 0;
  IICS_SRW = 0;
  IICC_TX = 1;                                  // Select Transmit Mode
        IICC_MST = 1;                 // Select Master Mode (Send Start Bit) this sets the
BUS Busy flag
        for(z=0;z<5;z++){
          ;
        }// Small delay
  IICD = SP03_ADDRESS;                                // Send selected slave address
}

//----FPGA function to change BMP ------------------------------------------------------
----------
void updateFPGA(void) {
  //send recipie number out A0-A4
  PTAD = recipeNumber;
}
```

```
void configFPGA(){
  //this function is to set up the output pins for recipe data to FPGA
  //PTAPEX pull ups

  //output 1 input 0
  PTBDD_PTBDD3 = 1; //these two lines are for the test swtich
  PTBD_PTBD3 = 1;

  //PTAD0 out
  //PTAD1 out
  //PTAD2 out
  //PTAD3 out
  //PTAD4 out
  //PTAD5 in
  PTADD = 0x1F;     //0001 1111  direction reg
  PTAD = 0x00;      //0000 0000  data reg
}

//----Interrupts ----------------------------------------------------------------------
----------
// Interrupt handler routine to manage all the events related to the IIC module
interrupt 17 void Viic_ISR(void){
       byte Temp;
       byte dummyRead;

       Temp = IICS;              // ACK the interrupt
       IICS_IICIF = 1;
       if(IICC_MST==1){          // If we are the IIC Master

    if(IIC_STEP == IIC_HEADER_SENT_STATUS){ // Header Sent
      IICC_TX = IIC_DATA_DIRECTION;
      IIC_STEP = IIC_DATA_TRANSMISION_STATUS;
      if(IICC_TX==0){  // If we are reading data clock in first slave byte
        Temp = IICD;
        return;
      }
    }

    if(IIC_STEP == IIC_DATA_TRANSMISION_STATUS){     // If byte transmision is in
progress.
      if(IICC_TX==1){                                       // If Master is sending data
to slave

        // Send the IIC_DATA to the slave
                            IICD = IIC_DATA[IIC_COUNTER];      // Send the next byte
                            IIC_COUNTER++;
                            if(IIC_LENGTH <= IIC_COUNTER){
                              IIC_STEP=IIC_DATA_SENT_STATUS;   // Mark we are done
sending Bytes
        }
                            return;
                       // wait until last byte sent
      }
    }

    if(IIC_STEP==IIC_DATA_SENT_STATUS){         // We are done with the transmition.
           IIC_STEP=IIC_READY_STATUS;                 // Reset our status flag
                Temp = IICS;                          // ACK the interrupt
      IICS_IICIF=1;
               IICC_TX=0;
               IICS_SRW=0;
               IICC_MST=0;// Generate a stop condition
               return;
    }
       }
  else{                 //  SLAVE OPERATION
         if(IIC_STEP <= IIC_READY_STATUS){                                  // If it is
the first byte tranmited
                IIC_STEP = IIC_DATA_TRANSMISION_STATUS;
```

```
                IICC_TX = IICS_SRW;                                          // Set the
transmision reception status
                IIC_COUNTER = 1;
          // If we are receiving data read IIC1D to get free bus and get the next byte
                if(IICC_TX==0){
                        Temp = IICD;
                        return;
            }
       }
          if(IICS_TCF==1){
                if(IICC_TX == 0){         // If data is received store it on the buffer
                        //IIC_DATA[IIC_COUNTER]=IICD;
                        dummyRead = IICD;
        IIC_COUNTER++;
        return;
      }
      else{                      // Data sent by the slave
                        if(IICS_RXAK==1){     // If byte is not ACK end transmision.
                               IICC_TX = 0;
                               Temp = IICD;
                               IIC_STEP = IIC_READY_STATUS;
                               return;
                        }
                        IICD = IIC_DATA[IIC_COUNTER];
        IIC_COUNTER++;
        return;
                }

         }
  }
}

//Interrupt handler for receiving serial packets from the touch screen
interrupt 15 void Vscirx_ISR(void) {
  // if there is too much latency in this handler we will need only store the SCID into a
global
  // and let main take care of inserting it into the array
  byte Temp;
  Temp = SCIS1;   // Acknowledge Receiver Full Flag

  if((SCID == 0x55) && (!start)){
    start = 1;
  }
  if(start && (buffer_index < MAX_BUFFER_SIZE)){
    ReceivedBuffer[buffer_index] = SCID;
    buffer_index++;
  }
  if((buffer_index == MAX_BUFFER_SIZE) && (ReceivedBuffer[2] != 4)){
    buffer_index = 0;
    start = 0;
  }
}


void delay(unsigned int cycle){
  for(z = 0; z < cycle; z++){
    ;
  }
}
```

Recipe.h

```
#ifndef __RECIPE_H_
#define __RECIPE_H_

//----IIC Control defines ------------------------------------------------------------------
----------
#define MASTER
#define IIC_ERROR_STATUS 0
#define IIC_READY_STATUS 1
```

```
#define IIC_HEADER_SENT_STATUS 2
#define IIC_DATA_TRANSMISION_STATUS 3
#define IIC_DATA_SENT_STATUS 4

//----Navigation defines ---------------------------------------------------------------
----------
#define HOME 0x00
#define NEXT 0x01
#define BACK 0x02

#define DINNER 0x11
#define DESSERT 0x12
#define BREAKFAST 0x13

#define RECIPE 0x14

#define MIDDLE0 0x30
#define MIDDLE1 0x31
#define MIDDLE2 0x32

#define BOTTOM0 0x40
#define BOTTOM1 0x41
#define BOTTOM2 0x42

#define DEFAULT 0x55

#define ROUND_CHANGED 0x60    //MCHANGED
#define SQUARE_CHANGED 0x61   //BCHANGED

#define MAX_BUFFER_SIZE 10

typedef struct {
   char * lines[10];

}recipe_line_s;


typedef struct{
 char round_buttons;  //middle_val
 char square_buttons; //bottom_val
 char button_event;
}screen_place_s;

#endif
```

FFT

```
void fft_double (unsigned int p_nSamples, bool p_bInverseTransform,
    double *p_lpRealIn, double *p_lpImagIn,
    double *p_lpRealOut, double *p_lpImagOut)
{

    if(!p_lpRealIn || !p_lpRealOut || !p_lpImagOut) return;


    unsigned int NumBits;
    unsigned int i, j, k, n;
    unsigned int BlockSize, BlockEnd;

    double angle_numerator = 2.0 * PI;
    double tr, ti;

    if( !IsPowerOfTwo(p_nSamples) )
    {
        return;
    }

    if( p_bInverseTransform ) angle_numerator = -angle_numerator;

    NumBits = NumberOfBitsNeeded ( p_nSamples );
```

```
    for( i=0; i < p_nSamples; i++ )
    {
        j = ReverseBits ( i, NumBits );
        p_lpRealOut[j] = p_lpRealIn[i];
        p_lpImagOut[j] = (p_lpImagIn == NULL) ? 0.0 : p_lpImagIn[i];
    }


    BlockEnd = 1;
    for( BlockSize = 2; BlockSize <= p_nSamples; BlockSize <<= 1 )
    {
        double delta_angle = angle_numerator / (double)BlockSize;
        double sm2 = sin ( -2 * delta_angle );
        double sm1 = sin ( -delta_angle );
        double cm2 = cos ( -2 * delta_angle );
        double cm1 = cos ( -delta_angle );
        double w = 2 * cm1;
        double ar[3], ai[3];

        for( i=0; i < p_nSamples; i += BlockSize )
        {

            ar[2] = cm2;
            ar[1] = cm1;

            ai[2] = sm2;
            ai[1] = sm1;

            for ( j=i, n=0; n < BlockEnd; j++, n++ )
            {

                ar[0] = w*ar[1] - ar[2];
                ar[2] = ar[1];
                ar[1] = ar[0];

                ai[0] = w*ai[1] - ai[2];
                ai[2] = ai[1];
                ai[1] = ai[0];

                k = j + BlockEnd;
                tr = ar[0]*p_lpRealOut[k] - ai[0]*p_lpImagOut[k];
                ti = ar[0]*p_lpImagOut[k] + ai[0]*p_lpRealOut[k];

                p_lpRealOut[k] = p_lpRealOut[j] - tr;
                p_lpImagOut[k] = p_lpImagOut[j] - ti;

                p_lpRealOut[j] += tr;
                p_lpImagOut[j] += ti;

            }
        }

        BlockEnd = BlockSize;

    }


    if( p_bInverseTransform )
    {
        double denom = (double)p_nSamples;

        for ( i=0; i < p_nSamples; i++ )
        {
            p_lpRealOut[i] /= denom;
            p_lpImagOut[i] /= denom;
        }
    }

}
```

FPGA code:

main.v

```
`timescale 1ns / 1ps

module main(CLK, RESET, RGB, HSYNC, VSYNC, recipe, DATA, SW7, SW6, SW5, ADDR, CS0, OE,
WE,/*debug values*/hCount,vCount,blank,pblank,pixel,SSG,AN,LED);
	input  CLK;  //50MHz
	input  RESET;
	input  [4:0] recipe;
	input  [7:0] DATA;
	input  SW7;
	input  SW6;
	input  SW5;
	output [18:0] ADDR;
	output CS0;
	output OE;
	output WE;
	output [2:0] RGB;
	output HSYNC;
	output VSYNC;

	//debug
	output [9:0] hCount;
	output [9:0] vCount;
	output blank;
	output pblank;
	output [7:0] pixel;
	//end debug

	reg  [18:0] ADDR_O = 0;
	wire [18:0] ADDR_W;
	wire [7:0] DATA;
	reg  clk25MHz = 0;

	output [0:6] SSG;
	output [0:3] AN;
	output [0:7] LED;

	reg [0:6] SSG;
	reg [0:3] AN;
	reg [0:7] LED;
	reg clk_1Hz;
	reg [0:24] counter_50M;

	parameter      BLANK   = 7'b1111111;
	parameter      ZERO    = 7'b0000001;
	parameter      ONE     = 7'b1001111;
	parameter      TWO     = 7'b0010010;
	parameter      THREE   = 7'b0000110;
	parameter      FOUR    = 7'b1001100;
	parameter      FIVE    = 7'b0100100;
	parameter      SIX     = 7'b0100000;
	parameter      SEVEN   = 7'b0001111;
	parameter      EIGHT   = 7'b0000000;
	parameter      NINE    = 7'b0000100;
	parameter   A          = 7'b0001000;
	parameter   B          = 7'b1100000;
	parameter   C          = 7'b0110001;
	parameter   D          = 7'b1000010;
	parameter   E          = 7'b0110000;
	parameter   F          = 7'b0111000;

	vga vga1(.clk(clk25MHz), .reset(RESET), .addr_i(ADDR_O), .addr_o(ADDR_W),
.rgb(RGB), .hSync(HSYNC), .vSync(VSYNC), .data(DATA), .cs1(CS0), .oe(OE), .we(WE),/*debug
values*/.hCount(hCount),.vCount(vCount),.blank(blank),.pblank(pblank),.pixel(pixel));


	assign ADDR = ADDR_W;
```

```
// clock divider for VGA timing 50MHZ/2 = 25MHZ
always @(posedge CLK)
begin
	clk25MHz <= clk25MHz^1;	// toggle the internal clk
end

// output current recipe number on 7seg
// 1Hz CLK
always @ (posedge CLK or posedge RESET)
begin
	if (RESET)
	begin
		clk_1Hz <= 0;
		counter_50M <= 0;
	end
	else
	begin
		counter_50M <= counter_50M + 1;
		if (counter_50M == 25_000_000)
		begin
			counter_50M <= 0;
			clk_1Hz <= ~clk_1Hz;
		end
	end
end


always @ (posedge clk_1Hz or posedge RESET)
begin
	if (RESET)
	begin
		LED <= 8'b11111111;	// turns on led's
	end
	else
	begin
		LED <= 8'b00000000;
		AN <= 4'b0111;		// enables 7-seg0 right most display
	end
end


always @ (recipe)
begin
	case (recipe[3:0])
		0: begin
			SSG <= ZERO;
			ADDR_O <= 19'h00000;
			end
		1: begin
			SSG <= ONE;
			ADDR_O <= 19'h12C00;
			end
		2: begin
			SSG <= TWO;
			ADDR_O <= 19'h25800;
			end
		3: begin
			SSG <= THREE;
			ADDR_O <= 19'h38400;
			end
		4: begin
			SSG <= FOUR;
			ADDR_O <= 19'h4B000;
			end
		5: begin
			SSG <= FIVE;
			ADDR_O <= 19'h5DC00;
			end
		6: SSG <= SIX;
		7: SSG <= SEVEN;
```

```
                        8: SSG <= EIGHT;
                        9: SSG <= NINE;
                        10: SSG <= A;
                        11: SSG <= B;
                        12: SSG <= C;
                        13: SSG <= D;
                        14: SSG <= E;
                        15: SSG <= F;
                        default: SSG <= BLANK;   // all segments are off
                endcase
        end

endmodule
```

vga.v

```
`timescale 1ns / 1ps

// 640x480 @ 25MHz clk
module vga(clk, reset, rgb, hSync, vSync, addr_i, addr_o, data, cs1, oe, we,/*debug
values*/hCount,vCount,blank,pblank,pixel);
        //debug
                output [9:0] hCount;
                output [9:0] vCount;
                output blank;
                output pblank;
                output [7:0] pixel;
        //end debug

        input clk;  //25MHz
        input reset;
        output [2:0] rgb;
        output hSync;
        output vSync;
        output [18:0] addr_o;
        input [18:0] addr_i;
        input [7:0] data;
        output cs1;
        output oe;
        output we;

        reg [9:0] vCount = 0;
        reg [9:0] hCount = 0;
        reg [7:0] pixel = 0;
        reg [2:0] rgb = 3'b000;
        reg vSync = 1;
        reg hSync = 1;
        wire blank;
        reg pblank;

        reg [18:0] memCount = 0;


        // horizantal pixel counter
        always @(posedge clk or posedge reset)
        begin
                if(reset)
                begin
                        hCount <= 0;
                end
                else if(memCount == 76800)    //(640*480)/4 = 76800, If this value is >
76800 the image scrolls, kinda cool
                        memCount <= 0;                                  //after a whole frame
has been drawn reset the memory "pointer"
                else
                begin                           // horiz. pixel counter rolls-over after 793
pixels
                        if(hCount<793) // 31.77*PixelCLK-1 note:this is a constant we are
going to use 25MHz
                        begin
```

```
                                hCount <= hCount+1;
                                if((hCount%4 == 0) && ~blank)  // increment memory
"pointer" every 4 counts of hCount
                                        memCount <= memCount+1;
                        end
                        else
                                hCount <= 0;
                end
        end


        // vertical line counter
        always @(posedge hSync or posedge reset)
        begin
                if(reset)
                        vCount <= 0;
                else
                begin
                        if(vCount<527) // total frame lines = 16784/31.77 - 1  note:this is
a constant
                                vCount <= vCount + 1;
                        else
                                vCount <= 0;
                end
        end


        // horizantal sync control
        always @(posedge clk or posedge reset)
        begin
                if(reset)
                        hSync <= 1;
                else
                begin
                        if(hCount>=664 && hCount<759)        // hsync start = .94*PixelCLK
<23> + ViewablePixels+?Lblank?+1
                                hSync <= 0;                                             //
hsync period = 3.77*PixelCLK+1
                        else
                                hSync <= 1;
                end
        end


        // vertical sync control
        always @(posedge hSync or posedge reset)
        begin
                if(reset)
                        vSync <= 1;
                else
                begin
                        if(vCount>=491 && vCount<493) // vsync start = .45*PixelCLK +
ViewableLines
                                vSync <= 0;                                             //
vsync period = .064*PixelCLK + 1
                        else
                                vSync <= 1;
                end
        end


        // blank video outside of visible region: visible region 640x480
        assign blank = (hCount>=640 || vCount>=480) ? 1:0;


        always @(posedge clk or posedge reset)
        begin
                if(reset)
                        pblank <= 0;
                else
                        pblank <= blank;
```

```
	end


	assign cs1 = 0;
	assign we = 1;
	assign oe = blank; //enable the RAM outputs when video is not blanked


	assign addr_o = addr_i+memCount;


	always @(posedge clk or posedge reset)
	begin
		if(reset)
			pixel <= 0;
		else
		begin
			//this is reading memory every 4 counts of hCount which is 4 pixels
			if(hCount[1:0] == 2'b00)
				pixel <= data;
			else
				pixel <= pixel[7:0]<<2; //left shift pixel register
		end
	end


	always @(posedge clk or posedge reset)
	begin
		if(reset)
			rgb <= 3'b000;
		else
		begin
			if(pblank == 1'b0)
			begin
				case(pixel[7:6]) //[1:0] only uses bottom 3 bits for color
RGB
					2'b00: rgb <= 3'b000;  //black
					2'b01: rgb <= 3'b111;  //white
					2'b10: rgb <= 3'b001;   //blue
					default: rgb <= 3'b100; //red
				endcase
			end
			else
				rgb <= 3'b000; //black
		end
	end

endmodule
```

main.ucf

```
NET "CLK"   LOC = "T9";
NET "RESET" LOC = "L14"; #Button3 active high

#NET "recipe<0>" LOC = "C10";
#NET "recipe<1>" LOC = "E10";
#NET "recipe<2>" LOC = "C11";
#NET "recipe<3>" LOC = "D11";
#NET "recipe<4>" LOC = "C12";

# switches
#NET "recipe<0>" LOC = "F12";
#NET "recipe<1>" LOC = "G12";
#NET "recipe<2>" LOC = "H14";
#NET "recipe<3>" LOC = "H13";
#NET "recipe<4>" LOC = "J14";
# B1 4,6,8,10,12
NET "recipe<0>" LOC = "C10";
NET "recipe<1>" LOC = "E10";
NET "recipe<2>" LOC = "C11";
```

```
NET "recipe<3>" LOC = "D11";
NET "recipe<4>" LOC = "C12";

NET "SW7" LOC = "K13";
NET "SW6" LOC = "K14";
NET "SW5" LOC = "J13";

# 7-segement display
NET "SSG<0>"  LOC = "E14";
NET "SSG<1>"  LOC = "G13";
NET "SSG<2>"  LOC = "N15";
NET "SSG<3>"  LOC = "P15";
NET "SSG<4>"  LOC = "R16";
NET "SSG<5>"  LOC = "F13";
NET "SSG<6>"  LOC = "N16";
#NET "SSG<7>"  LOC = "P16";
NET "LED<0>"  LOC = "K12";
NET "LED<1>"  LOC = "P14";
NET "LED<2>"  LOC = "L12";
NET "LED<3>"  LOC = "N14";
NET "LED<4>"  LOC = "P13";
NET "LED<5>"  LOC = "N12";
NET "LED<6>"  LOC = "P12";
NET "LED<7>"  LOC = "P11";
NET "AN<0>"  LOC = "D14";
NET "AN<1>"  LOC = "G14";
NET "AN<2>"  LOC = "F14";
NET "AN<3>"  LOC = "E13";

NET "RGB<0>"  LOC = "R11"; #blue
NET "RGB<1>"  LOC = "T12"; #green
NET "RGB<2>"  LOC = "R12"; #red
NET "HSYNC"   LOC = "R9";
NET "VSYNC"   LOC = "T10";

#NET "CS1" LOC = "B7"; #FLASH
NET "CS0" LOC = "A5"; #SRAM
NET "OE"  LOC = "A4";
NET "WE"  LOC = "A3";

NET "ADDR<0>"  LOC = "E6";
NET "ADDR<1>"  LOC = "C5";
NET "ADDR<2>"  LOC = "C6";
NET "ADDR<3>"  LOC = "C7";
NET "ADDR<4>"  LOC = "C8";
NET "ADDR<5>"  LOC = "C9";
NET "ADDR<6>"  LOC = "B8";
NET "ADDR<7>"  LOC = "A7";
NET "ADDR<8>"  LOC = "A9";
NET "ADDR<9>"  LOC = "A8";
NET "ADDR<10>" LOC = "A10";
NET "ADDR<11>" LOC = "B10";
NET "ADDR<12>" LOC = "B12";
NET "ADDR<13>" LOC = "B11";
NET "ADDR<14>" LOC = "B13";
NET "ADDR<15>" LOC = "A12";
NET "ADDR<16>" LOC = "B14";
NET "ADDR<17>" LOC = "A13";
NET "ADDR<18>" LOC = "D9";

NET "data<0>" LOC = "D5";
NET "data<1>" LOC = "D6";
NET "data<2>" LOC = "E7";
NET "data<3>" LOC = "D7";
NET "data<4>" LOC = "D8";
NET "data<5>" LOC = "D10";
NET "data<6>" LOC = "B4";
NET "data<7>" LOC = "B5";
```

testVGA.v

```
`timescale 1ns / 1ps
module testVGA;

	// Inputs
	reg CLK;
	reg RESET;
	reg [4:0] recipe;
	reg [7:0] DATA;

	// Outputs
	wire [5:0] RGB;
	wire HSYNC;
	wire VSYNC;
	wire [18:0] ADDR;
	wire CS0;
	wire OE;
	wire WE;
	wire [9:0] hCount;
	wire [9:0] vCount;
	wire blank;
	wire pblank;
	wire [7:0] pixel;

	// Instantiate the Unit Under Test (UUT)
	main uut (
		.CLK(CLK),
		.RESET(RESET),
		.RGB(RGB),
		.HSYNC(HSYNC),
		.VSYNC(VSYNC),
		.recipe(recipe),
		.DATA(DATA),
		.ADDR(ADDR),
		.CS0(CS0),
		.OE(OE),
		.WE(WE),
		.hCount(hCount),
		.vCount(vCount),
		.blank(blank),
		.pblank(pblank),
		.pixel(pixel)
	);

	always
	begin
		#1
		CLK=CLK^1;
	end


	initial begin
		// Initialize Inputs
		CLK = 0;
		RESET = 0;
		recipe = 0;
		DATA = 0;

		// Wait 100 ns for global reset to finish
		#100
      $display("\nStarting tests...");
      RESET = 0;
		#10
		while(1)
		begin
			RESET = 0;
			#10;
		end
	end
```

```
endmodule
```