Color Recognizer

Final Project Report
December 11, 2009
E155
Benyue Liu and Raquel Robinson

Abstract

Color is an essential feature of an object. Certain circumstances, for example, lost of sight, color blindless, poorly illuminated environment, or obstructed view, would restrict people's perception of color. This project aims at providing a basic solution to this problem, through a color recognizer. A webcam is interfaced with MATLAB and used to capture the image. MATLAB transmits color information to the microcontroller via a USART connection. The microcontroller identifies color and sends appropriate combination of allophones to the SpeakJet Chip. At the same time, the microcontroller generates a zero-padded 3-bit encoding, which is passed to FPGA via parallel connection. The FPGA utilizes a finite state machine to display the name of the color in a LCD. The device is partially functional. However, the finicky webcam introduces unstable RGB space color values. In the future, a Hue color space can be utilized to recognize the color more reliably.

Introduction

For people cannot perceive color correctly, a device that recognizes colors and verbally and visually communicates the color would be useful. Inspired by this subject, this project has designed a device to recognize the color of an object and implement it in the hardware. The goal of the project is to verbally communicate the color through a speech chip, and visually display the name of the color in a text based LCD.

SpeakJet Speech chip is controlled by the microcontroller and an LCD interfaces with FPGA. A built-in webcam in a laptop is used to capture the image through MATLAB commands. Color RGB values are sent to the microcontroller by serial communication by Bluetooth. The microcontroller recognizes the color and sends the color encoding to both SpeakJet Speech Chip and FPGA. FPGA will interpret the color and send coding to the display. Figure 1 is the overall flow diagram.

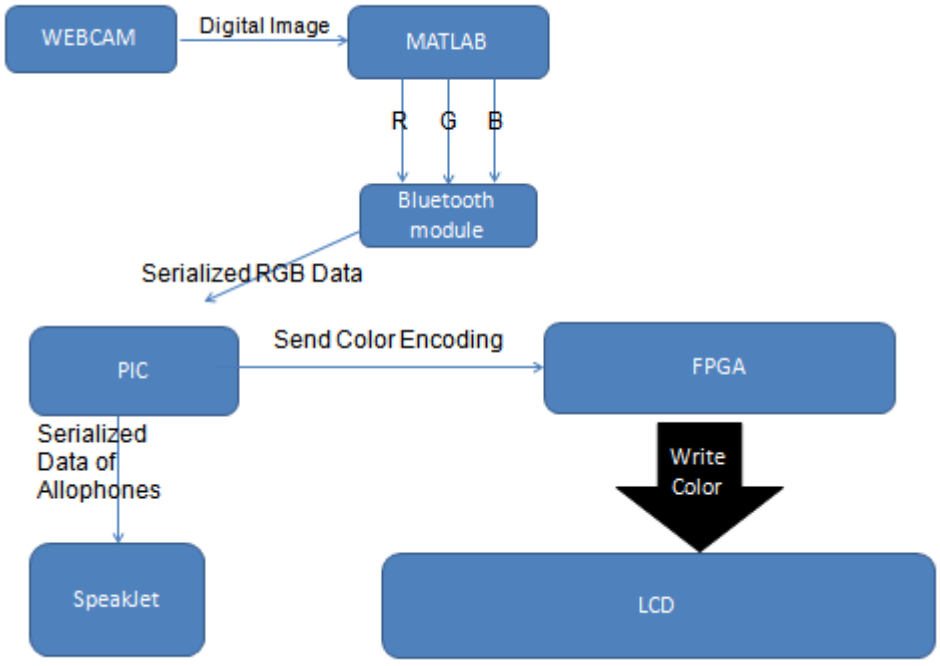


Figure 1. Overall Flow Diagram

New Hardware

The project utilizes two pieces of hardware not used previously in E155. These are the SpeakJet speech chip (see Figure 2) and the Crystalfontz (CFAH2002A-NYG-JP) LCD (see Figure 3).

Figure 2: Picture of Speakjet chip

The SpeakJet chip contains 72 preconfigured allophones which can be combined to produce complete English words. These allophones can be found on page 8 of the SpeakJet's datasheet. There are two mean of controlling the SpeakJet: 1) Simultaneous logic changes on any one of its eight event input lines; or 2) by a serial line of data. This project utilizes the second method. Data is passed to the SpeakJet from via a serial data line, using the built in USART on the PIC microcontroller.

The SpeakJet is connected to the PIC by a single serial connection on PORTC. The transceiver pin on the PIC is directly tied to the serial input pin of the speech chip (RCX). The SpeakJet has a default baud rate of 9600 bauds, and a default configuration of 8 data bits, no parity and 1 stop bit. Consequently the PIC is configured and initialized accordingly to ensure that a successful interface is achieved (see function *void main void* in Appendix V).

The PIC also needs to receive data serially from the Bluetooth module (BlueSmirf) which has a similar configuration to the SpeakJet but which utilizes a different baud rate of 115.2kbauds. Consequently, SPGRH is toggled between 10 and 129 when talking to the Bluetooth module versus talking to the SpeakJet respectively.

Another area of concern with the SpeakJet is it outputs only a small current of 25ma. This is insufficient to power an 8 ohm speaker. This can be resolved by using the Op Amp LM386N audio amplifier to amplify the sound.

The LCD we used in the project is a text based Crystalfontz (CFAH2002A-NYG-JP) LCD. It can display 2 lines and 20 characters. The pin assignment can be found in Appendix I.

Figure 3: LCD Picture.

The LCD's instruction table can be found in Appendix III. When writing instruction, many instructions need time for it to finish execution. When programming the FPGA, the execution time has to be taken into consideration. Before writing letter to the display, it has to be initialized first. The initialization sequence can be found in Appendix IV.

Writing data to a LCD requires sending the 8-bit encoding. Each letter encoding can be found in the data sheet (Appendix II).

When putting together LCD on the breadboard, pin 2 needs 5 volts, whereas Pin 3 is a variable voltage. The project connects it to ground to provide contrast.

MATLAB Design

MATLAB is used to interface with webcam because it is difficult to design a USB protocol for PIC within a short time. MATLAB's Image Data Acquisition Toolbox is used to interface with webcam and manipulate the data. A function capture_sPort.m (Appendix VII) is written to open the video object, preview the video, wait for certain time for the webcam to stabilize and get a snapshot. It also writes to the PIC using fprint. A timer is set to trigger the function continuously.

The serial port is setup before calling the function capturedTimer_sPort.m(Appendix VII). After examining the channel of the Bluetooth, the serial port object is set up with this specific outgoing port and a baudrate of 9600.

The RGB values are calculated using the average value of the pixels in the middle of the image. The RGB values are sent as characters with a carriage return in the end. This is crucial because the microcontroller is set to terminate receiving data until a carriage return is entered.

Microcontroller Design

The overall function of the PIC in this project is to obtain and store the image data sent serially from MATLAB via the Bluetooth module, translate this ASCI data into integer values corresponding to R G B values, use these R G B values to identify the correct color, and send the appropriate allophones and encoding to the SpeakJet and FPGA respectively.

The PIC completes the following sequence to achieve this function:

1) Initialization and configuration of the USART in *void main (void)*. The variables TXSTA, SPBRG, TRISC, RCSTA are given the appropriate values (see Appendix V)

2) In *char getcharserial(void)* the PIC waits for receive a character from the serial port and then returns it. Interrupt flag, RCIF, indicates when data is ready and can be returned from RCREG.

3) The PIC then utilizes the function *void getstrserial(char *origstring)* to store the entirety of the data sent from MATLAB via the serial port as a string. It achieves this by calling *getcharserial()* until a carriage return is detected.

4) The string Origstring is then parsed and converted in *void split(void)* into three integers which corresponding to the R G B values which contain the color information.

5) The function *int color_identifier(void)* essentially utilizes the integer R G B values to identify the correct color. It does this by checking what preset range combination these values fall into. Upon identifying the correct color, the baud rate is set to the SpeakJet default of 9600 bauds and a series of printf statements is then used to send the correct allophone combination of the color word to the SpeakJet. Printf writes data to the serial transceiver port on the USART. At the same time the PIC generates a zero padded 3 bit encoding specific to the color identified and passes this to the FPGA by writing this encoding to the LSB's of PORTA which the FPGA captures as an input.

FPGA Design

In this project, FPGA is used to receive data from the PIC microcontroller and send the 8-bit letter encoding of the color name to the text-based Crystalfontz (CFAH2002A-NYG-JP) LCD.

In the FPGA design, a slow clock is generated to accommodate the slow execution time (Appendix VI). At every slow clock edge, the color encoding is sent to the FPGA. Getcolor.vhd (Appendix VI) translates the color encoding to 48-bit color encoding using the datasheet (Appendix II). The 48 bit represents 6 characters. In this project, six colors are blue, red, black, yellow, green and white. The most number of letters in the spelling is yellow, so the letter encoding is set to 8*6=48 bits, with empty spaces appended after the spelling if they are not 6 characters long. The main LCD finite state machine includes the initialization stages and writing the sequence of the 6 letter encoding. The state transition diagram can be found in (Figure 4). The verilog code can be found in (Appendix VI).

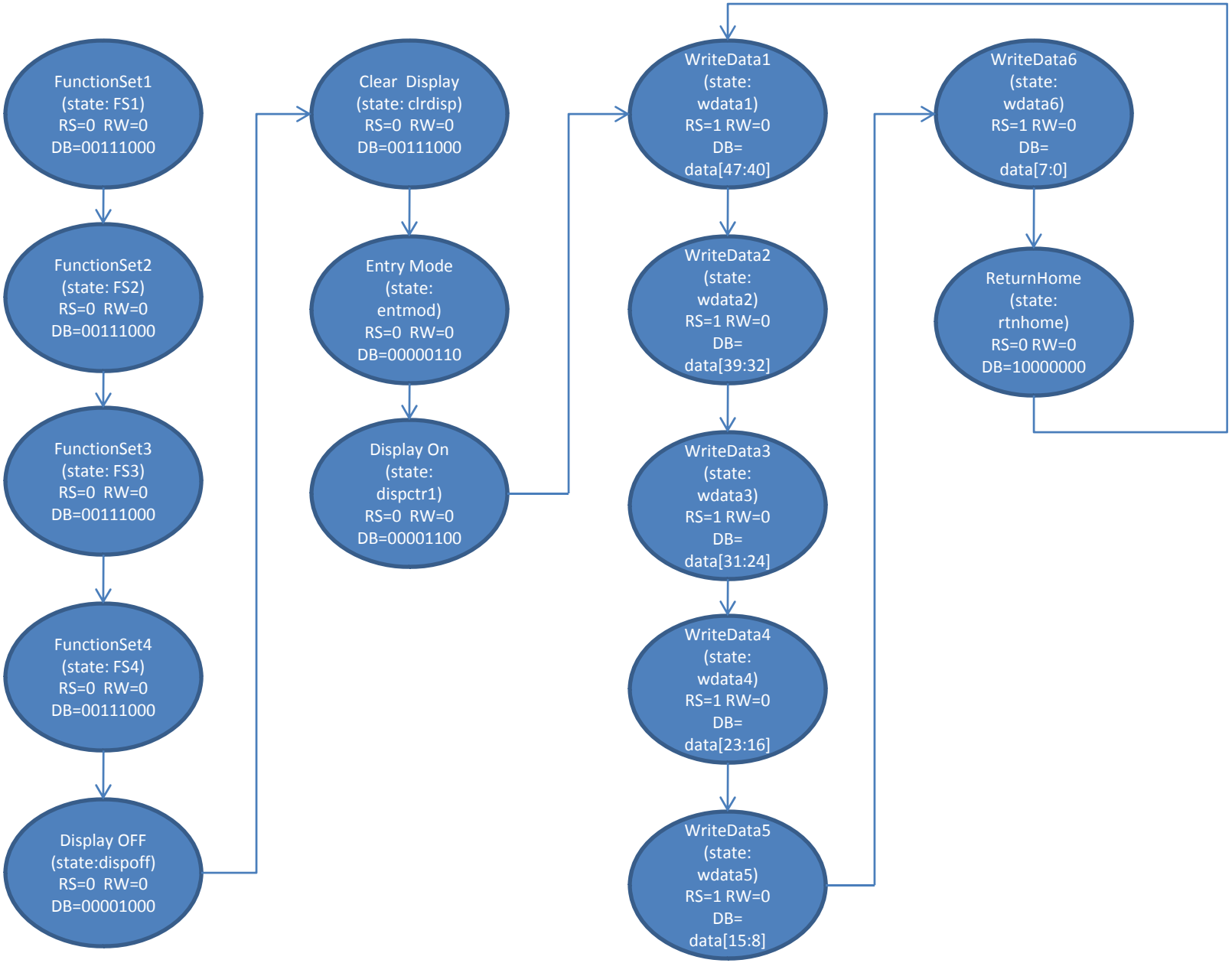


Figure 4. State Transition Diagram for FPGA LCDstatemachine.v module(Appendix VI)

Overall Schematic

Figure 5 shows the overall schematic.

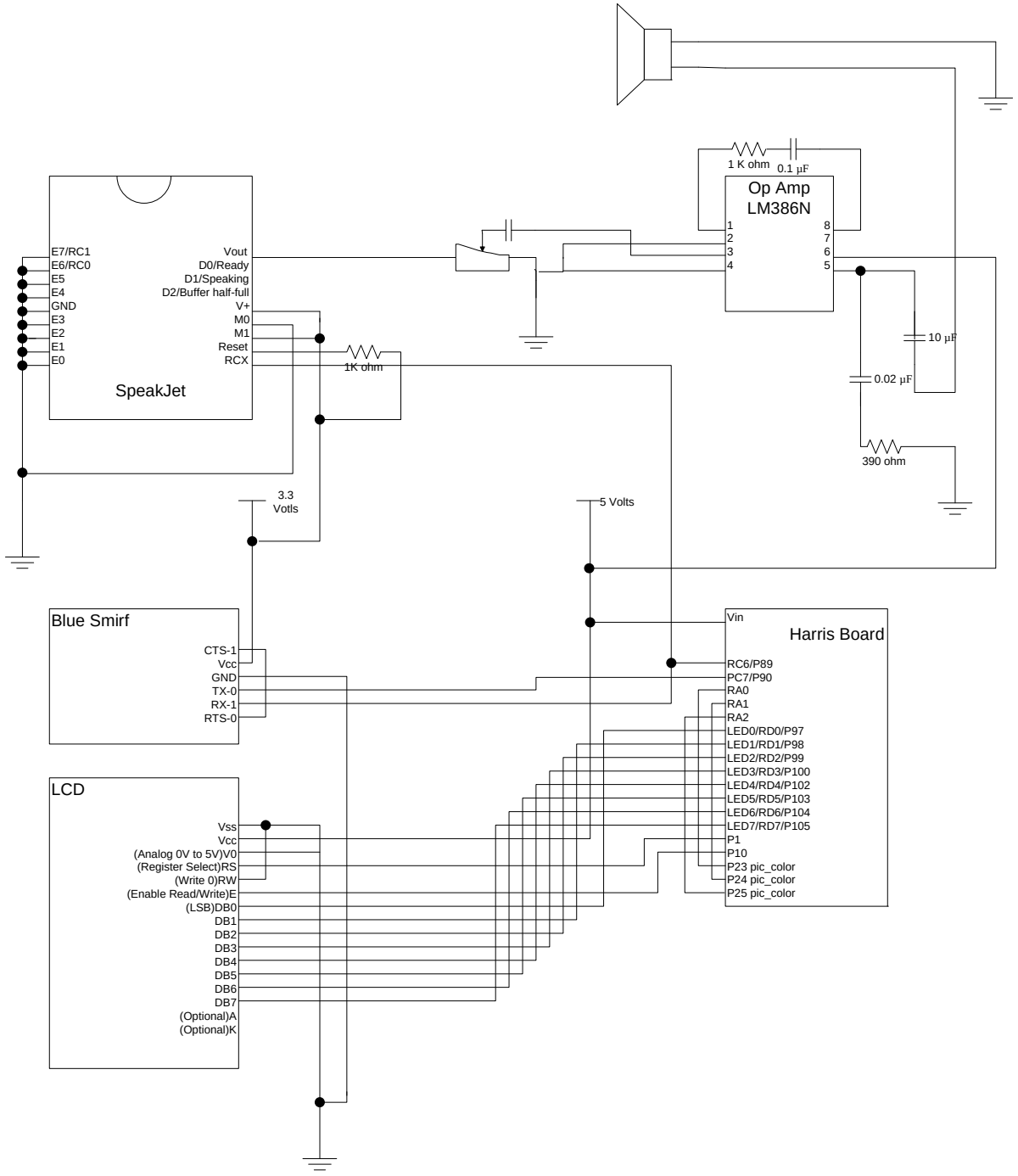


Figure 5. Overall Schematic of the System.

Results

The initial goal of this project was to build a device that would recognize color using solely a webcam, MATLAB, the Bluetooth Module the FPGA and the Speakjet chip. The plan was to utilize MATLAB to capture image from a webcam, transmit this RGB image data to the PIC via the BlueSMiRF wireless module, identify the color and send image encoding from the PIC to the FPGA. The FPGA would then generate the correct allophone combination and serially transmit this to the RCX pin on the SpeakJet. We were able to successfully allow MATLAB to capture the image using a webcam and sends information to the PIC via the Bluetooth Module. However our initial attempt at creating a Universal Serial Asynchronous Transmitter (USAT) on the FPGA proved unsuccessful. To achieve this we needed to generate a slowed clock on the FPGA that would match the default baud rate (9600 bauds) of the Speakjet. We would then have to use a state machine and output logic to organize the data in 10 bit packets (8 data bits and 1 start and 1 stop bit) which the FPGA would then send to the SpeakJet. Though this sounds relatively basic in theory we ran into some issues in our implementation. Consequently, we decided to instead utilize the FPGA to write the color on an LCD screen and have the PIC send the correct allophone combination to the SpeakJet since it has a USART readily available. Implementing the LCD proved to be difficult as well. Before anything can be written to the screen it has to be initialized as discussed in the FPGA section above. The difficulty with this was primarily due to timing and wiring. The LCD executes the 8 bit instruction passed to it on the negative edge of its slowed clock input and consequently it is essential for the data to be passed at the positive clock edge so that the is stable by the negative edge. After redefining our scope we were able to complete our design with marginal failures in functionality due to a underperforming digital camera.

Our final design achieves the following: MATLAB successfully sends color information (RGB values) to the PIC. The PIC successfully acquires these values and uses them to identify the color using ranges specified in the code. The correct allophone combination of color identified is successfully passed to the SpeakJet and a correct zero padded 3 bit encoding is passed to the FPGA. If the PIC cannot find a match for the values received it sends out the message “LO” to the SpeakJet. When the FPGA receives the right encoding successfully write the color on the display. However, at the moment, the reliability of the device is very low. The image acquisition toolbox on MATLAB seems to be aliasing the color that the webcam actually picks up. At the moment the camera is finicky and sees different colors based on the lighting of the surroundings. Consequently at times it confuses lighter colors such as yellow, green and white.

Future Work

To fix our current reliability issues, we believe that instead of using RGB color model values as the primary means of color characterization hue saturation brightness (HSB) color model could be used. This is a much more accurate means of differentiating color and is unaffected by light intensity. This can be done by simply having MATLAB return HSB values instead of RGB values and manipulating our PIC code to check HSB ranges instead as well. Once this issue is corrected identifying the colors of specific objects within it background can be investigated. It should then be possible to explore adding a memory chip to this device to interface with the PIC so that identifying object shapes and pattern matching can be explored.

Parts List

PART	SOURCE	VENDOR	PRICE
SpeakJet *Natural Speech & Complex Sound Synthesizer*	E155 Cabinet	Magnevation	FREE
OP AMP LM386N	E155 Cabinet		FREE
LCD (CFAH2002A-NYG-JP)	E155 Cabinet	Crystalfontz	FREE
Speaker	E155 Cabinet		FREE
BlueSMiRF wireless module	E155 Cabinet		FREE

Appendix I. Interface Pin Functions

Pin No.	Symbol	Level	Description
1	V_{SS}	0V	Ground
2	V_{DD}	5.0V	Supply Voltage for logic
3	VO	(Variable)	Operating voltage for LCD
4	RS	H/L	H: DATA, L: Instruction code
5	R/W	H/L	H: Read(MPU→ Module) L: Write(MPU→ Module)
6	E	H,H→ L	Chip enable signal
7	DB0	H/L	Data bit 0
8	DB1	H/L	Data bit 1
9	DB2	H/L	Data bit 2
10	DB3	H/L	Data bit 3
11	DB4	H/L	Data bit 4
12	DB5	H/L	Data bit 5
13	DB6	H/L	Data bit 6
14	DB7	H/L	Data bit 7
15	A	-	LED +
16	K	-	LED -

Appendix II.Character Generator ROM Pattern

Table.2

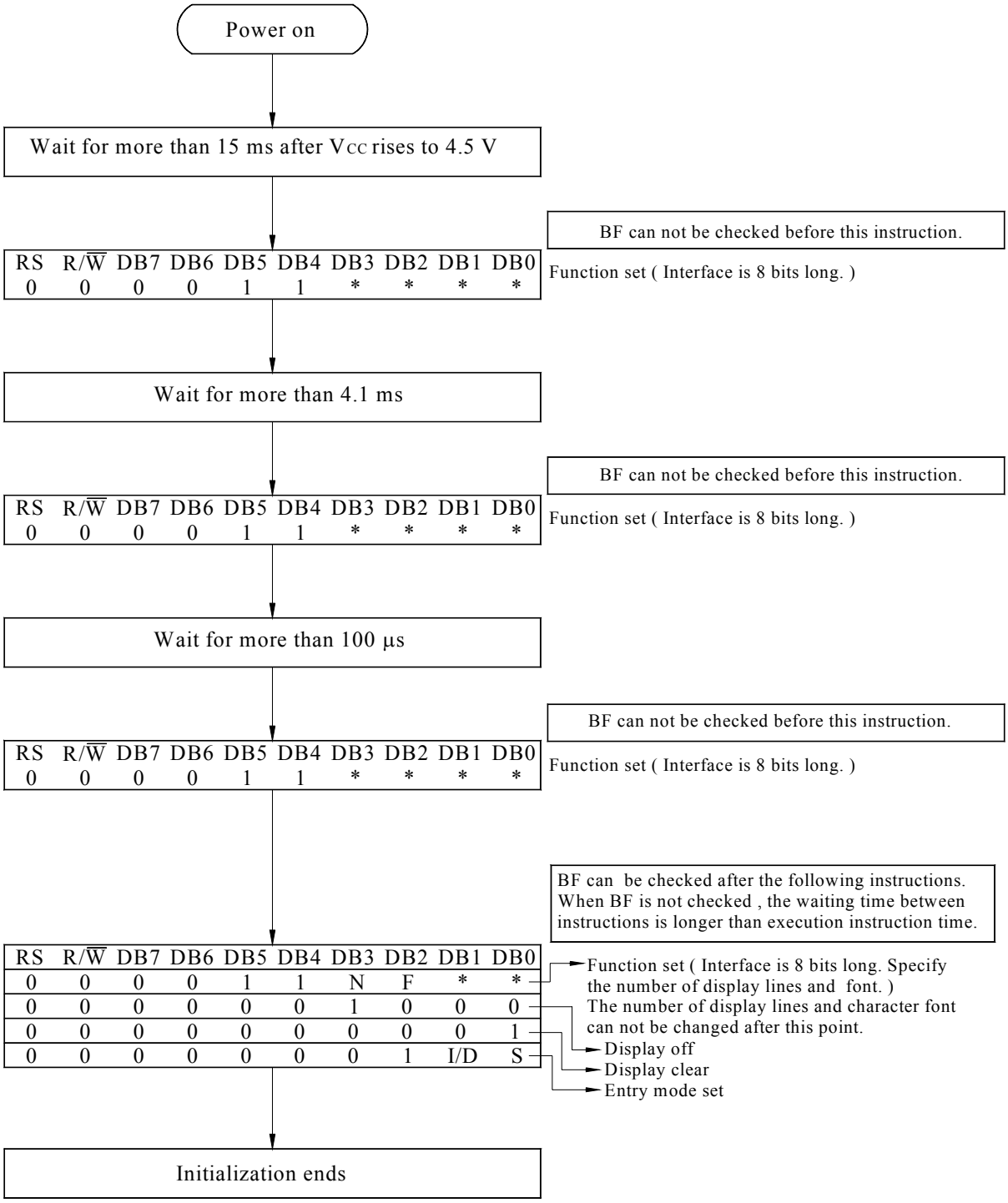
Upper 4 bit / Lower 4 bit	LLLL	LLLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHLH	HHHL	HHHH
LLLL	CG RAM (1)															
LLLH	(2)															
LLHL	(3)															
LLHH	(4)															
LHLL	(5)															
LHLH	(6)															
LHHL	(7)															
LHHH	(8)															
HLLL	(1)															
HLLH	(2)															
HLHL	(3)															
HLHH	(4)															
HHLL	(5)															
HHLH	(6)															
HHHL	(7)															
HHHH	(8)															

Appendix III.Instruction Table

Instruction	Instruction Code										Description	Execution time (fosc=270Khz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write “00H” to DDRAM and set DDRAM address to “00H” from AC	1.53ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to “00H” from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.53ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction and enable the shift of entire display.	39μ s
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor (C), and blinking of cursor (B) on/off control bit.	39μ s
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shift control bit, and the direction, without changing of DDRAM data.	39μ s
Function Set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL:8-bit/4-bit), numbers of display line (N:2-line/1-line)and, display font type (F:5×11 dots/5×8 dots)	39μ s
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter.	39μ s
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter.	39μ s
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0μ s
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43μ s
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43μ s

* ” - ” : don’t care

Appendix IV.Initializing of LCM



8-Bit Ineterface

Appendix V. PIC Microcontroller Code

```
//Color recognizer. c
// Raquel Robinson and Benyue Liu (Emma)
// 12/06/09
// Color recognizer
// Color allophones for reference
//char red[3]={0x94,0xB0,0x00};
//char green[4]= {0xAB,0x94,0x80,0x00};
//char yellow[4]={0x9E, 0x92,0x89,0x00};
//char black[5]={ 0xAB,0x91,0x84,0xC5,0x00};
//char blue[4]={ 0xAb,0x91,0x8B,0x00};
//char white[4]={ 0xB9,0x9D,0xBF,0x00};
//char none[2]= {0x00,0x00};
#include <p18f4520.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// Global Variables
char origstring[15] ; // stores the string sent over the serial port
int a;              //  R value (first number)
int b;              //  G value (second number)
int c;              //  B value (third number)
int color;          // stores the one hot 8 bit encoding for received data for testing

// Fuction Prototypes
void main(void);
char getcharserial(void);
void getstrserial (char *origstring);
int color_indentifier(void);
void split(void);

 char getcharserial(void)
//waits to receive a character from the serial port, then it returns it
{
   while (PIR1bits.RCIF == 0)
   {
   }
      if (PIR1bits.RCIF == 1)
      {
         return RCREG;
         PIR1bits.RCIF =0;
      }

}

void getstrserial(char *origstring)
```

```
//receives a string over the serial port by callin getcharserial
//until a carriage return is received
{
   char input;
   int i;
   i=0;
   input = getcharserial();

   while ( input != 0x0D)
   {
      origstring[i] = input;
       i++;
      input = getcharserial();
   }
      origstring[i] = NULL;
}

void split(void)
// splits up the preceeding string and coverts it into 3 integers
//corresponding to R, G, B values
{
   int i;
   i= 0;
   a= 0;
   b= 0;
   c= 0;

/*  the upcoming loop will go through the string origstring one char at at time
 if the char is not a space it will  check if it is between 0 and 9,
 if it is then it will multiply int a by 10 and add the char-48(convert it
  from ASCI to dec) to int a after it hits a space it will then jump to do the
same thing for the second integer int b*/

   while (!(origstring[i] >47 && origstring[i] <58))
         {
            i++;
         }
   while (origstring[i] != 32 && origstring[i] >47 && origstring[i]<58 )
         {
            a = (a*10) + (origstring[i]-48);
             i++;
          }
            i++;
   while (origstring[i] != 32 && origstring[i] >47 && origstring[i]<58 )
         {
            b = (b*10) + (origstring[i]-48);
             i++;
```

```
          }
            i++;
  while (origstring[i] != 32 && origstring[i] >47 && origstring[i]<58 )
        {
           c = (c*10) + (origstring[i]-48);
            i++;
          }
            i=0;
}

/*The following function checks the RGB values from the serial port sets the baudrate to 9600 bauds
and
 sends the right allophones as serial output to the speech chip writes the encoding to PORTA*/
int color_identifier(void){
   if (a <=250 && a>=190 && b>=80 && b<=110 && c<=120 && c>=80)   //red RGB Value Upper and
Lower Bounds
      {
         color=  0b00000010;
         PORTA = 0b00000010;
  SPBRG = 129;
         printf("%c", 0x94);
         printf("%c", 0x96);
         printf("%c", 0xB0);
         printf("%c", 0x0E);
         printf("%c", 0xAE);
         printf("%c", 0x00);
         return color;
      }

   else if (a<=120  && a>=95 && b>=130 && b<=150 && c<=110 && c>=80) //Green RGB Value Upper
and Lower Bounds
      {
         color = 0b00000110;
         PORTA = 0b00000110;
         SPBRG = 129;

  printf("%c", 0xAB);
  printf("%c", 0x94);
  printf("%c", 0x80);
  printf("%c", 0x00);
         return color;
      }

   else if (a<=190 && a>=110 && b>=110 && b<=190 && c<=100  && c>=70)
//Yellow GRB Value Upper and Lower Bounds
      {
         color=  0b00000001;
         PORTA = 0b00000001;
```

```
        SPBRG = 129;


        printf("%c", 0x9E);
        printf("%c", 0x92);
        printf("%c", 0x89);
        printf("%c", 0x00);
        return color;
      }
   else if (a<=50  && a>=0 && b>=0 && b<=50 && c<=50 && c>=0)
//black RGB Value Upper and Lower Bounds
     {
        color=  0b00000100;
        PORTA = 0b00000100;
        SPBRG = 129;
        printf("%c", 0xAB);
        printf("%c", 0x91);
        printf("%c", 0x84);
        printf("%c", 0xC5);
        printf("%c", 0x00);

        return color;
      }


   else if (a<=185  && a>=140 && b>=150 && b<=190 && c<=160 && c>=120) //white RGB Value Upper
and Lower Bounds
     {
        color = 0b00000101;
        PORTA = 0b00000101;
          SPBRG = 129;
          printf("%c", 0xB9);
          printf("%c", 0x9D);
          printf("%c", 0xBF);
          printf("%c", 0x00);
          return color;
      }
   else
     {
        color = 0b00000000;
        PORTA = 0b00000000;
        SPBRG = 129;

        printf("%c", 0x92);
        printf("%c", 0x92);
        printf("%c", 0x92);
        printf("%c", 0x00);
```

```
        return color;
      }
}



void main (void){


 TRISA = 0b00000000; // Initializes PORTA as an output port
   PORTA = 0b00000000; // PORTA has value 0
   TXSTA = 0b00100110; //sets up the  the transmit status and control register
   TRISC = 0b10000000; // configure rc6/tx and rc7/rx set bit 7 and clear bit6.
      /*
      bit 7    0   CSRC:  Dont care
      bit 6    0   TX9: 8 bit transmition
      bit 5    1   TXEN: Transmit enabled
      bit 4    0   SYNC: Asynchronous mode enable receiver
      bit 3    0   UNIMPLEMENTED: Dont care
      bit 2    1   BRGH:   High speed baud rate
      bit 1    1   TRMT:  TSR full
      bit 0    0   TX9D:   Dont care

      */

   RCSTA = 0b10010000;  //0x90;// configure receive status and control register
      /*
      bit 7    1   SPEN: Serial Port Enabled
      bit 6    0    RX9: 8 bit reception
      bit 5    0   SREN: DONT CARE
      bit 4    1   CREN: Asynchronous mode enable receiver
      bit 3    0   ADDEN: Dont care
      bit 2    0   FERR:   Dont care
      bit 1    0   ODERR:  Dont care
      bit 0    0   RX9D   Dont care

      */
 SPBRG = 0b00001010; // sets the baudrate to receive from bluesmirf 0x0A

  while(1){

   getstrserial (origstring);
   split();
   color_identifier();
   SPBRG= 0x0A; // reset the buadrate to get data from bluesmirf

   }
}
```

Appendix VI. FPGA Modules

```
module mainlcd(
        input clk,
        input reset,
        input [2:0]pic_color,
        output [7:0]instr,
        output RS,
        output RW,
        output lcdclk);

        wire [47:0]lcdcolor;
        LCDclock newclk(clk, reset, lcdclk);
         getcolor testcolor ( pic_color, clk, reset, lcdcolor);
        LCDstatemachine test(lcdclk, reset, lcdcolor, RS, RW,instr);
endmodule
```

```
module LCDclock(
	input clk,
	input reset,
	output lcdclk );

	reg [18:0] count;
	always @(posedge clk or posedge reset)
	if (reset)
		count<=1'b0;
	else
		count <= count+1'b1;
	assign lcdclk = count[18];
endmodule
```

```
module  getcolor(
input [2:0]pic_color,
input clk,
input reset,
output reg [47:0] lcdcolor
   );

          parameter none   = 3'b000;
          parameter yellow = 3'b001;
          parameter red    = 3'b010;
          parameter blue   = 3'b011;
          parameter black  = 3'b100;
          parameter white  = 6'b101;
          parameter green  = 6'b110;
          always@(*)
          case(pic_color)
          yellow:  if(~reset)lcdcolor <=
48'b0101_1001_0100_0101_0100_1100_0100_1100_0100_1111_0101_0111;
          red:     if(~reset)lcdcolor <=
48'b0101_0010_0100_0101_0100_0100_0001_0000_0001_0000_0001_0000;
          blue:    if(~reset)lcdcolor <=
48'b0100_0010_0100_1100_0101_0101_0100_0101_0001_0000_0001_0000;
          black:   if(~reset)lcdcolor <=
48'b0100_0010_0100_1100_0100_0001_0100_0011_0100_1011_0001_0000;
          white:   if(~reset)lcdcolor <=
48'b0101_0111_0100_1000_0100_1001_0101_0100_0100_0101_0001_0000;
          green:   if(~reset)lcdcolor <=
48'b0100_0111_0101_0010_0100_0101_0100_0101_0100_1110_0001_0000;
          none:    if(~reset)lcdcolor <=
48'b0001_0000_0001_0000_0001_0000_0001_0000_0001_0000_0001_0000;
          default: if(~reset)lcdcolor <=
48'b0001_0000_0001_0000_0001_0000_0001_0000_0001_0000_0001_0000;//none

          endcase
endmodule
```

```
module LCDstatemachine(
input clk,
input reset,
input [47:0]data,
output reg RS,
output reg RW,
output reg[7:0]instr
   );
        reg [3:0] state, nextstate;


        parameter FS1           = 4'b0000;
        parameter FS2           = 4'b0001;
        parameter FS3           = 4'b0010;
        parameter FS4           = 4'b0011;
        parameter clrdisp = 4'b0100;
        parameter dispctrl= 4'b0101;
        parameter entmod  = 4'b0110;
        parameter wdata1  = 4'b0111; //W
        parameter wdata2  = 4'b1000; //O
        parameter wdata3  = 4'b1001;// L
        parameter wdata4  = 4'b1010;// L
        parameter wdata5  = 4'b1011;// E
        parameter wdata6  = 4'b1100; //Y
        parameter rtnhome = 4'b1101;
        parameter dispoff = 4'b1110;


        // state register
always@(posedge clk)
        if (reset) state <=FS1;
        else    state<=nextstate;

always @(*)
        case(state)

                FS1:
                if (clk)
                begin
                 RW <= 1'b0;
                 RS <= 1'b0;
                 instr <= 8'b00111000;
                 nextstate <= FS2;
                end


                FS2:
                if (clk)
```

```
begin
 RW <= 1'b0;
 RS <= 1'b0;
 instr <= 8'b00111000;
 nextstate <= FS3;
 end

FS3:
if (clk)
begin
 RW <= 1'b0;
 RS <= 1'b0;
 instr <= 8'b00111000;
 nextstate <= FS4;
end

FS4:
if (clk)
begin
 RW <= 1'b0;
 RS <= 1'b0;
 instr <= 8'b00111000;
 nextstate <= dispoff;
end

dispoff:
if (clk)
begin
 RW<= 1'b0;
 RS<= 1'b0;
 instr<=8'b00001000;
 nextstate<=clrdisp;
end

clrdisp:
if (clk)
begin
 RW <= 1'b0;
 RS <= 1'b0;
 instr <=8'b00000001;
 nextstate <= entmod;
end

entmod:
if (clk)
begin
 RW <= 1'b0;
 RS <= 1'b0;
```

```
 instr <= 8'b00000110;
 nextstate <=dispctrl;
end

dispctrl:
if (clk)
begin
 RW <= 1'b0;
 RS <= 1'b0;
 instr <=8'b00001100;
 nextstate <= wdata1;
 end



wdata1:
if (clk)
begin
 RW <= 1'b0;
 RS <= 1'b1;
 instr <= data[47:40];
 nextstate <=wdata2;
end

wdata2:
if (clk)
begin
 RW <= 1'b0;
 RS <= 1'b1;
 instr <= data[39:32];
 nextstate <=wdata3;
 end

wdata3:
if (clk)
begin
 RW <= 1'b0;
 RS <= 1'b1;
 instr <= data[31:24];
 nextstate <=wdata4;
 end

wdata4:
if (clk)
begin
 RW <= 1'b0;
 RS <= 1'b1;
 instr <= data[23:16];
```

```
                    nextstate <=wdata5;
                    end

                   wdata5:
                   if (clk)
                   begin
                    RW <= 1'b0;
                    RS <= 1'b1;
                    instr <= data[15:8];
                    nextstate <=wdata6;
                    end

                   wdata6:
                   if (clk)
                   begin
                    RW <= 1'b0;
                    RS <= 1'b1;
                    instr <= data[7:0];
                    nextstate <=rtnhome;
                    end

                   rtnhome:
                   if (clk)
                   begin
                    RW <=1'b0;
                    RS <=1'b0;
                    instr <= 8'b10000000;
                    nextstate <=wdata1;
                   end

                   default:
                    if (clk)
                    nextstate <= FS1;
endcase
endmodule
```

Appendix VII. Matlab Code

```
function [R,G,B]=capture_sPort (sPort)
% this program requires setting up a serial port called sPort with Baudrate
% at 9600. and the serial port should be connected using command fopen.

  %open video object
vid = videoinput ('winvideo',1);

% set the image space rgb
set (vid,'ReturnedColorSpace','rgb');
preview (vid);
% wait for the camera to stablize.
   pause(10);
% gettiing a snapshot of the video
snapshot=getsnapshot (vid);

stoppreview (vid);
delete (vid);
clear vid;

imshow (snapshot);
% get the Red Value
R=round (mean (mean (snapshot (470:490,630:650,1))));
% get the Green Value
G=round (mean (mean (snapshot (470:490,630:650,2))));
% get the Blue Value
B=round (mean (mean (snapshot (470:490,630:650,3))));

% send the R,G,B value to the microcontroller
% change R to a string
Rcolorinfo= int2str (R)
% change G to a string
Gcolorinfo= int2str (G)
% change B to a string
Bcolorinfo= int2str (B)
% combine the RGB with space in between and carriage return at the end.
%Carriage return is important for the microcontroller to realize it is the
% end of the message.
   colorinfo=[Rcolorinfo char (32) Gcolorinfo char (32) Bcolorinfo char (13)]
% sending it to microcontroller
fprintf (sPort,'% c',colorinfo);
```

```
function CaptureTimer_sPort ()

t=timer;
% set the timer to run capture.m
set (t,'TimerFcn','[R,G,B]=capture_sPort (sPort)');

% set the timer running at a fixedRate
set (t,'ExecutionMode','fixedRate');
start (t);
```