A Display System For Current Location and Message

Final Project Report
12/13/18
E155

Teddy Dubno Peter Johnson

Abstract:
As students who live in close-knit communities with all of the members living in close proximity, students at Harvey Mudd College may often find themselves going to a fellow students room to find them. However, sometimes students are not in there room, either they are busy somewhere else or the cant respond to everyone trying to find out where they are. This project prototypes a door mounted information system consisting of an IOS application, a webserver, a VGA Display driver and VGA LCD display. The user types a message on the IOS application the app gets the users locations, and sends the message and location to the web server which process the information, sends it to the VGA driver on the FPGA which displays the message and location on the display. This system is functional and ready to be installed on any student who may want it.

Introduction

A large problem at a huge school like Mudd is not knowing where friends are at any given time. One might try walking to their room and knock, but no one answers. Where could they be? Sometimes those friends will be busy and unable to respond. One would have to wait and hope to see them later. Now instead imagine the scenario where the friend is using our messaging and display system. On their door is a small display. All you really need to provide curious folks with is your location and a quick message. For example, it could say "Taking a test in Parsons." This tool provides quick access to information depending on when the user wants to provide it to the world.

Our project will consist of a system to display messages and location data sent from an iphone. We will extract the location of that iphone, as well as a text message created by the user through through a simple iOS app which utilizes Apple's location services data and then send that data to the pi web server via an HTTP request (The request will not be sent continuously-location is checked only when the app is open and the request is only sent when the user presses a specific button.) From there, the pi will do some preprocessing on the strings and pass these strings of data via Serial Peripheral Interface (SPI) to the Field Programmable Gate Array (FPGA) which will process this data to make it in a format such that it can be displayed on a Video Graphics Array (VGA) compatible screen (the FPGA will be a VGA driver.) The interaction of all these systems is shown in figure 1.

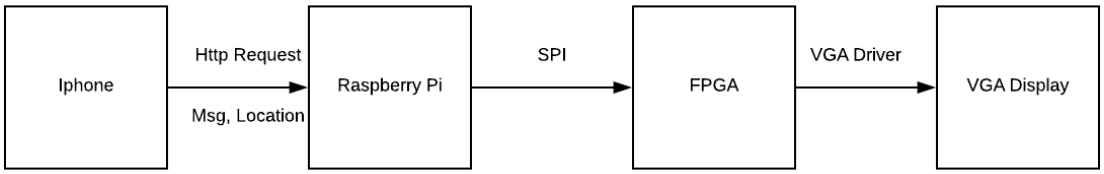

Figure 1*: Block Diagram*.

New Hardware

This project made use of two new pieces of hardware; an iPhone, and a monitor.
We ended up using both a Cathode Ray Tube (CRT) monitor and an Liquid Crystal Display (LCD) display as the screen for our text. Both of these monitors are simple to use, and function in identical ways. You only need to power them and then hook up the VGA cable to begin to display images. No other setup is necessary.

The key to this functionality is the VGA graphics standard for video display controllers. VGA was designed for CRT monitors with a 640x480 pixel display resolution. The information is passed via a 15 pin connector using analog voltages (RGB values range from 0-0.7V with higher value meaning brighter.) In a CRT, an electron gun scans across the pixels of the screen from left to right updating each pixel in a row. There is an electron beam to turn on each color on the display Red, Green, and Blue. Different colors can be produced by varying the intensity of each of the three colors relative to one another. At the end of each row, the electron gun must turn off for a period of time as the gun moves to the beginning of the next row on the left side of the screen. After all of the rows have been scanned by the gun, it must once again turn off as it returns to the upper left corner to repeat the process.

This process repeats about 60 times per second (60 Hz) to give the illusion of a steady screen to human eyes. In actuality, the monitor refreshes a complete screen at 59.94 Hz, and the screen is actually 525 rows of 800 pixels (the pixels outside the 640x480 are black and not visible.) The pixel clock operates at 25.175 MHz meaning each pixel is 39.72 ns. A scanline or row is 800 pixel clocks long. There are 48 in the “back porch” which are not displayed, then 640 of them are the active pixels, then 16 in the front porch. The last 96 of the 800 is when the scan gun is to be moved to the next row. The timing of this movement is done using an Hsync signal which is active low, only going low for thos 96 pixel clocks, and remaining high for the porches and the active pixels. The vertical timing is very similar. The main difference is that now the timing units are rows/scanlines instead of pixel clocks. The back porch is 32 rows, there are 480 active rows, and the front porch is 11 rows. Vsync is only low for a count of 2 rows as it returns to the top. LCD displays on the other hand doesnt function at all like this, but are backwards compatible with the VGA signals.

The Iphone is another new piece of hardware to work with, however, controlling this new hardware is more akin to working on a micro controller utilizing the programming language swift. The Application starts, and before letting the user begin, requests access to

location data on the iphone, if the user gives permission, the app then begins location services and starts updating location data. The application has a label a text box and a button, if the user presses the button the application takes the message from the text box, if it is empty it takes no message, capitalizes all of the letters, strips out white space replaces it with [. Then, the application takes the current location data, looks up where on mudd's campus that latitude and longitude correspond to, format that string as above, then sends that performs an async http get request with message and location as parameters. If the user closes the application the application stops updating location services to conserve power, if the app is turned back on the location services are re-enabled.

Schematic

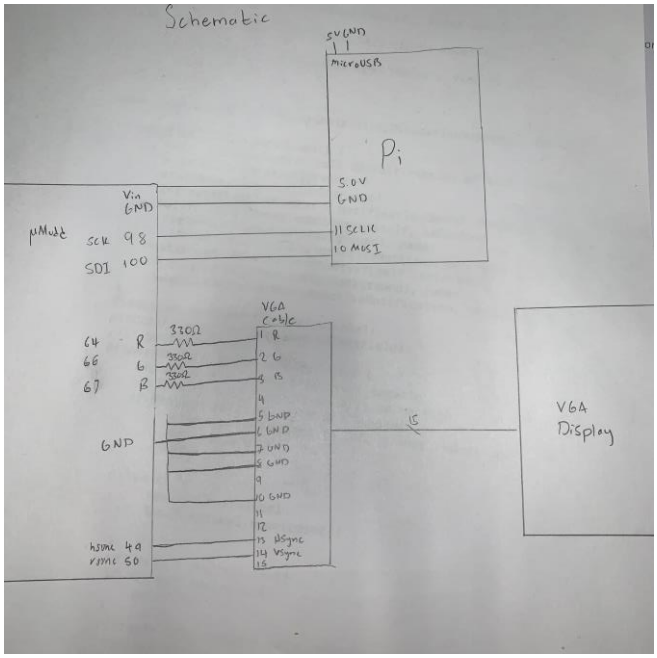

Figure 2*: System Schematic*.

The schematic for the bread board as seen in figure 2 is fairly straightforward, we have 5 volts and Gnd, from the Pi into the vin and ground on the FPGA, we also connect pin 11 (sclk), and pin 10 (Mosi) on the Pi, to pin 98(SCK) and pin 100(SDI) on the FPGA respectively. Between the VGA driver on the FPGA and the VGA adapter connected to VGA display, we have pin 64 (r) pin 66 (g) and and pin 67 (b) on the FPGA connected to pin 1 (r), pin 2 (g), pin 3 (b) on the vga adapter respectively. Pin 5,6,7,8,10 on the vga adapter all all connected to ground, pin 49 hsync and 50 vsync to pin 13 and 14 hsync and vsync respectively.

Microcontroller Design

We utilized the microcontroller on the raspberry pi, because it is easily programmable, has an operating system to help manage the web server and updating the code more easily. The Pi is running an apache2 web server running cgi-bin, this allows the web server to execute code but in the appropriate directory /usr/lib/cgi-bin/ this allows the utilization of the Pi's I/O pins remotely over the internet. The function of the micro controller in this project is to receive http requests from the Iphone parse the request to get the location and message, format them for the FPGA and send the information to the FPGA. The program ran by cgi-bin on request, gets the query string from the request, goes through the string and loads the message into its own buffer the size of the max length message which is padded by the encoding of space so that if the the message is less than max length it will display white space to the right of the message. Next this same process is done for the Location, the message and location buffers are then passed over spi in reverse, utilizing the EasyPIO library provided, to the FPGA to be stored.

FPGA Design

The FPGA will be handling a large portion of the project, it will be taking in messages from the Pi over SPI, interpreting that data as relevant character information, then driving the LCD Display by sending the correct digital and analog signals over FPGA. The FPGA is well suited for this task as it can quickly do the logic needed to figure out every line of every frame of the desired image. The FPGA will additionally be in charge of correctly handling color and font sizing concerns from a hardware perspective. For an overview of the FPGA system, see figure 3 which details the main blocks and where signals are being passed.

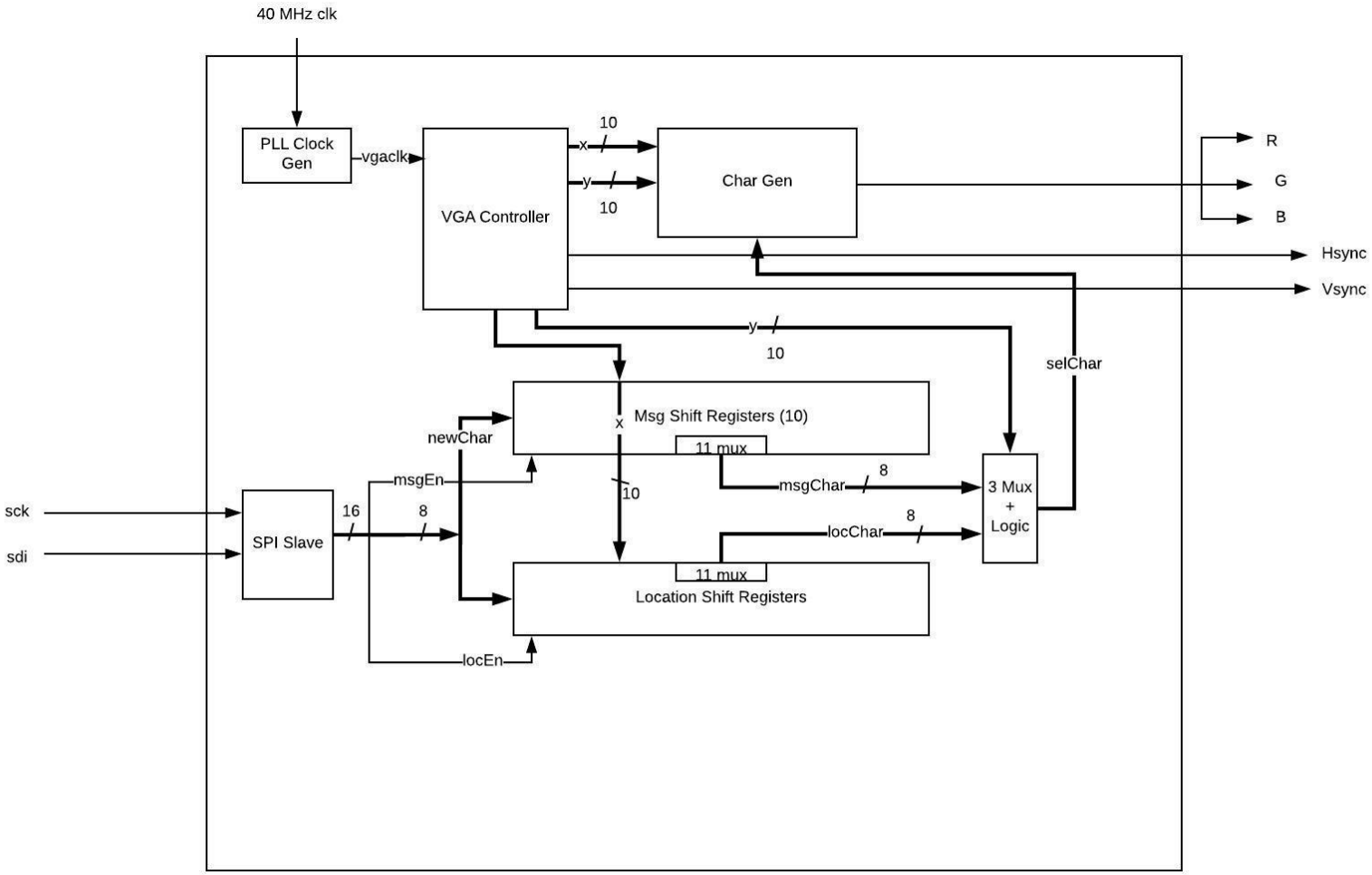

Figure 3*: FPGA Block Diagram*.

The first thing to consider in the VGA driver is the generation of timing signals. Without these, nothing else is possible. The FPGA takes in a 40 MHz clock from the Mudd Pi board. The pixel clock (vgaclk) of 25.175 MHz is generated using a phase locked loop created using the function megawizard instead of a clock divider. The VGA controller is responsible

for creating the Hsync and Vsync signals. It does so, by using the vgaclk to count through rows and columns. Depending on the row or column, the hsync and vsync will be set low. If not, they remain high. This section was written using functional SystemVerilog, but implies adders and registers for the counters, as well as muxes, comparators and several gates to do the logic based on the counting. The next module of importance is the video generation. The main module for this is the character generator ROM (Read only Memory.) It reads in an array of 32 by 32 bitmaps of characters from A-Z (ASCII 65-90), as well as the bitmap for a space in ASCII 91 ("[".) The chargenrom module was passed the character to be displayed, as well as the 6 lower bits of the current x and y signals (you only need bits corresponding to the 32 indices of the rows or columns of the character bitmap) . The output was whether the pixel corresponding to the x and y was on. It did this by grabbing a 32 bit line of the character based on the y bits combined with the ascii value-65 (since the first value in the memory was A which is 65.) To select the pixel from the row, the x offset is subtracted from 31 to get the index (the left most column in ROM is most significant, but it will be drawn in the least significant x position.) Once the pixel has been determined to be on or off, the red, green, and blue signals all receive this value of on or off. This 3.3V digital output is converted to 0.7V using a voltage divider. 330Ω in R1, with the 75Ω R2 built into our VGA cable.

The next set of modules deal with providing the specific character signal at the right time to the chargenrom. The first of these is the SPI interface. This is a slave module with the Pi function as the master. There is no need for the FPGA to send data to the Pi, so there is only an sck signal and an sdi (serial data in) signal. On each rising edge of sck, a bit is shifted into the 16 bit received register. To manage when a signal is finished being sent, the SPI module computes a done bit by counting how many times sck has gone high (how many bits have been shifted in.) Another key component in the FPGA design is the module that connects the SPI result with the shift registers such that only full chars are pushed into a register and only one char moves at a time. We accomplish this by taking the done bit outputted by spi when it receives a full char and control signal (16 bits). For each time the done bit goes high we make a signal that first sets the hasntdone to signal low, then sets the clkdone signal high. Then on the next clk cycle clkdone is set low. Hasntdone will be set to high again on the neck clk cycle that the done bit goes low. This way we have a module that works as a synchronizer from the inconsistent sclk to clk.

Once a message has been shifted in, the character that it contain will go into a shift register for either the message characters or the location characters. This occurs on the positive edge of the clk, but only if the clkdone bit is high and the enable bit corresponding to message or location is also high. These shift registers contain 15 characters of 8 bits each. Once the registers have all 15 bits shifted in, the next step is to select which character to use. Each character register corresponds to an index on the screen. Based on the x value at

the time, a character is selected from the message shift register and the location shift registers. The last step in selecting the character is to multiplex between those two characters and a space depending on the y value.

Results

This project was a successful attempt that yielded the desired results. A user is able to display their current location and a personalized message on a LCD VGA display at their leisure, and control it appropriately. The most difficult part of this process was figuring out how to move and store the characters that were coming in from spi so that they would only update when the SPI was done sending a full char and the control signals. We fixed this using a small system that generates a single pulse both during clk when the SPI was done and the pulse had not been sent yet. This needed to happen because the done on the SPI would go high for longer than a clock cycle so the char might get shifted too many registers down and chars could get lost. Our final project was almost identical to what we set out to accomplish with our initial proposal. We were instructed to remove the name of the display from the proposal, but that is the hardware that we still use. Additionally the initial proposal listed an internet connected device, instead of just saying an iphone, we fixed this and just utilized the IPhone xr and IPhone 6s.

References

[1] Harris, David M, and Sarah l Harris. *Digital Design and Computer Architecture*. Elsevier Science & Technology, 2015.

Parts

We only used parts available in the MicroP's lab and the stockroom.

Part	**Source**
VGA Breakout Board	Stock Room
LCD Display	MicroP's Lab

Appendices

Appendix I IOS application code

```
//
//  ViewController.swift
//  MicroProcessors
//
//  Created by Teddy Dubno tdubno@hmc.edu, Peter Johnson pjohnson@hmc.edu
on 11/15/18.
//  Copyright (c) 2018 Teddy Dubno. All rights reserved.
//
//Code That controls interaction from the Main view of application.
import UIKit
import MapKit

class ViewController: UIViewController,CLLocationManagerDelegate {

    override func viewDidLoad() {
        super.viewDidLoad()//call default version of view loading
        enableBasicLocationServices()
        startReceivingLocationChanges()
        let notificationCenter = NotificationCenter.default
        notificationCenter.addObserver(self, selector:
#selector(appMovedToBackground), name:
UIApplication.willResignActiveNotification, object: nil)
        notificationCenter.addObserver(self ,selector:
#selector(willReturnFromBackground), name:
UIApplication.didBecomeActiveNotification, object: nil)
    }
    @IBOutlet weak var textbox: UILabel!
    @IBOutlet weak var Message: UITextField!
    //if button pressed
    @IBAction func doit(_ sender: Any) {
        var local = whereAmI()//current locatio
        let length = 15 //length of allowed message
        if Message.text?.count ?? 0 > length{
            textbox.text="Message to long please reduce length to
\(String(length))"
            return
        }
        else {
            textbox.text=local
            local = local.uppercased()
```

```
        }
        var mess = Message.text!
        if Message.text==nil{
            print("oh")
        }
        //create http request parse message and format for fpga, remove
space replace with [.
        var urls = "http://134.173.200.227/cgi-
bin/final?l="+local.replacingOccurrences(of: " ", with: "[")
        urls+="&m="
        if mess==""{
            mess="No_Message"
        }
        urls+=mess.uppercased().replacingOccurrences(of: " ", with: "[")
        let url = URL(string: urls)!
        let urlSession = URLSession.shared
        let getRequest = URLRequest(url: url)
        //run async http request, drop data that comes back after error.
        let task = urlSession.dataTask(with: getRequest as URLRequest,
completionHandler: { data, response, error in

            guard error == nil else {
                return
            }

            guard let data = data else {
                return
            }

            do {
                print("done")
            }
        })

        task.resume()
    }
    var curlocation: CLLocation!
    let locationManager = CLLocationManager()
    //check if user has allowed location services, get permission if not.
    func enableBasicLocationServices() {
        locationManager.delegate = self
```

```
        switch CLLocationManager.authorizationStatus() {
        case .notDetermined:
            // Request basic authorization if no permissions have been
attempted
            locationManager.requestWhenInUseAuthorization()
            break

        case .restricted, .denied:
            // Disable location features
            break

        case .authorizedWhenInUse, .authorizedAlways:
            // Enable location features
            break
        }
    }

    //startRecievingLocationChanges, if Location permission is provided
    func startReceivingLocationChanges() {
        let authorizationStatus = CLLocationManager.authorizationStatus()
        if authorizationStatus != .authorizedWhenInUse &&
authorizationStatus != .authorizedAlways {
            //Hopefully this wont happen
            return
        }
        //check is service is avalible on iphone
        if !CLLocationManager.locationServicesEnabled() {
            // Location services is not available.
            return
        }
        // Configure and start the service.
        locationManager.desiredAccuracy = kCLLocationAccuracyBest
        locationManager.distanceFilter = 1.0  // In meters more than
possible but we are hopeful.
        locationManager.delegate = self//this is the file that should have
access to information
        locationManager.startUpdatingLocation()//start running
    }
    //processes location info
    func locationManager(_ manager: CLLocationManager,  didUpdateLocations
locations: [CLLocation]) {
        let lastLocation = locations.last!
        curlocation=lastLocation
```

```
    }

    //stops updating location for battery saving
    func stopPlease(){
        locationManager.stopUpdatingLocation()
    }
    //function that returns current location on campus.
    func whereAmI()-> String{
        //hard coded coordinates
        //uses dynamic ranges and pattern matching
        let shan = (34.106217 ... 34.106680, -117.711235 ... -117.710647)
        let acend = (34.105448 ... 34.106803, -117.713118 ... -117.711407)
        let west = (34.105573 ... 34.105965, -117.709160 ... -117.708668)
        let linde = (34.105692 ... 34.106221, -117.705825 ... -117.705226)
        let hoch = (34.105543 ...  34.105958, -117.710137 ... -117.709303)
        let pa = (34.106272 ... 34.106695, -117.712352 ... -117.711597)
        let campus = (34.104309 ... 34.106934, -117.713208 ... -117.703786)
        let scripps = (34.102645 ... 34.105328, -117.713141 ... -
117.707208)
        let cmc = (34.099209 ... 34.102609, -117.711440 ... -117.704992)
        //start case statement
        switch (curlocation.coordinate.latitude,
curlocation.coordinate.longitude)
        {
        case (shan.0,shan.1):return "At Shan"
        case (west.0,west.1):return "At West"
        case (linde.0,linde.1): return "At Linde"
        case (hoch.0,hoch.1):return "At the Hoch"
        case (pa.0,pa.1): return "In Parsons"
        case (acend.0,acend.1): return "At AcEnd"
        case (campus.0,campus.1):return "On Campus"
        case (scripps.0,scripps.1):return "At Scripps"
        case (cmc.0,cmc.1):return "At Cmc"
        default: return "unmapped region"
        }
    }
    //if you leave app stop location to save battery
    @objc func appMovedToBackground() {
        stopPlease()
    }
    //if the app comes back start location again.
    @objc func willReturnFromBackground(){
```

```
            startReceivingLocationChanges()
        }
    }
```

Appendix II Micro controller cgi-bin C code

```
//Final.C
//Written By tdubno@hmc.edu and pjohnson@hmc.edu
//Used as a by cgi-bin in web server for final MicroPs  project
#include "EasyPIO.h"//Provided code for using raspberry Pi I/O Pins
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define FREQ 244000  //Frequency of SPI
#define LENG 15
int main(int argc, char ** argv)
{
    int num=0;
    //Get data passed to webserver if possible
    char* p = getenv("QUERY_STRING");
    if(p==NULL){
    }
    else{
        int i;
        //Init Messege and locaion array we use [ as space because it is
char after Z
        char mess[] =
{'[','[','[','[','[','[','[','[','[','[','[','[','[','[','['};
        char local[] =
{'[','[','[','[','[','[','[','[','[','[','[','[','[','[','['};
        int partOfQuery=0;
    //Needs properly formatted requests, no error prevention for now.
        for(i=2; p[i]!='\0'; i++){
            if(p[i]=='&'){//switching parameters in query string.
                   partOfQuery=1;
               i+=2;
            }
            else if (partOfQuery) {
                mess[partOfQuery-1]=p[i];
                partOfQuery++;
            }
            else {
                local[i-2]=p[i];
            }
        }
        pioInit();
```

```
    spiInit(FREQ,0);//init PI I/O and spi
    for(i=LENG-1; i>=0; i--){
        short toSend=0b1000000000000000;//formatting 16 bit
        toSend |= mess[i];
        spiSendReceive16(toSend);//send formatted char
    }
    for(i=LENG-1; i>=0; i--){
        short toSend=0b0000000000000000;
        toSend |= local[i];
        spiSendReceive16(toSend);
    }
}
return 0;
}
```

Apendix III FPGA Verilog modules

```
////////////////////////////////////////////
// VGA.sv
// HMC E155 15 November 2018
// pjohnson@g.hmc.edu @tdubno@g.hmc.edu
// For MicroP's Final Project
// Implements VGA driver that takes in the message to be displayed
through spi
// Msg component displays on one line, location component displays on
another line
////////////////////////////////////////////


////////////////////////////////////////////
// VGA
// Top level module with SPI interface and VGA core
////////////////////////////////////////////
module VGA(input  logic clk,              /From 40 MHz oscillator
           input  logic sck,                    //From Pi
           input  logic sdi,                    //From Pi
            output logic hsync, vsync,         //To VGA cable
            output logic r,g,b);                //To vga cable

      //intermdiate logic
    logic [15:0] received;
    logic [7:0] char;
    logic vgaclk,done,clkdone,msgEn;

      //instantiate spi module
    vga_spi spi(sck,sdi,done,received);
      assign char = received[7:0];            //2nd half of short is the
char to be displayed
      assign msgEn = received[15];            //Most significant bit is
control bit

      //instantiate vga driver module
      vga_core vga(clk,done,char,msgEn,clkdone,vgaclk,hsync,vsync,r,g,b);

endmodule
```

```
//////////////////////////////////////////
// vga_spi
//   SPI slave interface is receive only
//   Instead of receiving a done bit, it just counts how many cycles of sck
have occured
//////////////////////////////////////////
module vga_spi(input logic sck,          //From master
                           input logic sdi,        //From master (is sdi)
                           output logic done,      //done bit generated
by counting
                           output logic [15:0] q); // Data received

      //intermediate logic
      logic [6:0] counter;

      //spi shift register
      always_ff @(posedge sck)
            q <= {q[14:0], sdi};

      //done bit counter register
      always_ff@(posedge sck)
            if (counter == 16)
                  counter <=1;
            else counter <= counter +1;

      //done when 16 bits have been received
      assign done = (counter==16);

endmodule

//////////////////////////////////////////
// vga_core
//    vga driver implementation
//////////////////////////////////////////
module vga_core(input logic clk,
                           input logic done,
                           input logic [7:0] char,
                           input logic msgEn,
                           output logic clkdone,
                           output logic vgaclk,
                           output logic hsync, vsync,
                           output logic r,g,b); //RGB values will be a
single intensity
```

```
	//intermediate variables
	logic [9:0] x, y;		//x and ycounter
	logic [7:0] ch;			//character to be displayed

	// Using the wizard, PLL to create the 25.175 MHz VGA pixel clock
	// Screen is 800 clocks wide by 525 tall, but only 640 x 480 used
	// HSync = vgaclk/800 = 31.470 kHz
	// Vsync = Hsync / 525 = 59.94 Hz (~60 Hz refresh rate)
	//Hsync and Vsync dont have regular duty cycles, so have to do weird
things
	PLLVGA PLLVGA_inst(.inclk0 ( clk ),.c0 ( vgaclk ));

	//instantiate character select module
	charSel msgShft(clk,done,clkdone,msgEn,char,x,y,ch);

	// Generate monitor timing signals
	vgaController vgaCont(vgaclk, hsync, vsync, x, y);

	// Module to determine whether pixels are on
	videoGen videoGen(x, y,ch, r, g, b);

endmodule

////////////////////////////////////////////
// vgaController
//  vga implementation
////////////////////////////////////////////
module vgaController #(parameter HBP = 10'd48,          //H back porch
                       HACTIVE = 10'd640,  //Horizontal active section
                       HFP = 10'd16,              //Horizontal front porch
                       HSYN = 10'd96,             //H sync time
                       HMAX = HACTIVE + HFP + HSYN + HBP,
                       VBP = 10'd32,
                       VACTIVE = 10'd480,
                       VFP = 10'd11,
                       VSYN = 10'd2,
                       VMAX = VACTIVE + VFP + VSYN + VBP)
                                    (input logic vgaclk,
                                     output logic hsync, vsync,
                                     output logic [9:0] x, y); //10
bits for up to 1024 values
	// counters for horizontal and vertical positions
	always @(posedge vgaclk) begin
```

```
            x++;                                    //vga_clk iterates through
pixels
            if (x == HMAX) begin    //If at last pixel, then we start over
                  x = 0;
                  y++;                              //Increment row
                  if (y == VMAX)    //If last row, then start over
                        y = 0;
                  end
            end

      // Compute sync signals (active low (just not the output) Refer to
pictures in Ch 9 of digital design
      assign hsync = ~(x >= HBP+ HACTIVE + HFP & x < HMAX);
      assign vsync = ~(y >= VBP+ VACTIVE + VFP & y < VMAX);

endmodule

////////////////////////////////////////////
// videogen
//  vga implementation
////////////////////////////////////////////
module videoGen(input logic [9:0] x, y,
                              input logic [7:0] ch,
                              output logic r,g,b);
      logic pixelOn;
      // Given y position, choose a character to display
      // then look up the pixel value from the character ROM
      // and display it in red or blue.Also draw a green rectangle.
      chargenrom charrom_inst(ch, x[5:0], y[5:0], pixelOn);

      assign r =pixelOn;
      assign g =pixelOn;
      assign b =pixelOn;

endmodule

////////////////////////////////////////////
// chargenrom
//  vga implementation
////////////////////////////////////////////
module chargenrom(input logic [7:0] ch, //ascii value
                                    input logic [5:0] xoff, yoff,
                                    output logic pixelOn);
```

```
	logic [31:0] charrom[26623:0]; // character generator ROM only need 6
bits
	logic [31:0] line; // a line read from the ROM

	// Initialize ROM with characters from text file
	initial
		$readmemb("char.txt", charrom);

	// Index into ROM to find line of character
	assign line = charrom[yoff + {ch-65,5'b00000}]; // Subtract 65
because A

                        // is entry 0
	// Reverse order of bits when picking
	//Left most col in ROM is most significant
	//Drawn in least significant x position
	assign pixelOn = line[5'd31-xoff];

endmodule

module megaflop(input logic clk,
                              input logic done,
                              input logic en,
                              input logic [7:0] ch,
                              output logic [7:0]
q0,q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,q12,q13,q14 );
       always_ff@(posedge clk)
            if(done&en)
                  begin
                              q0 <= ch;
                              q1 <= q0;
                              q2 <= q1;
                              q3 <= q2;
                              q4 <= q3;
                              q5 <= q4;
                              q6 <= q5;
                              q7 <= q6;
                              q8 <= q7;
                              q9 <= q8;
                              q10<= q9;
                              q11 <= q10;
                              q12 <= q11;
                              q13 <= q12;
```

```
                        q14 <= q13;
                end
endmodule

module charSel #(parameter CHW = 10'd32,
                           CHH = 10'd32,
                           HBP = 10'd48,          //H back porch
                           HACTIVE = 10'd640,  //Horizontal active section
                           HFP = 10'd16,          //Horizontal front porch
                           HSYN = 10'd96,         //H sync time
                           HMAX = HACTIVE + HFP + HSYN + HBP,
                           VBP = 10'd32,
                           VACTIVE = 10'd480,
                           VFP = 10'd11,
                           VSYN = 10'd2,
                          VMAX = VACTIVE + VFP + VSYN + VBP)
                          (input logic clk,
                           input logic done,
                           output logic clkdone,
                           input logic msgEn,
                           input logic [7:0] char,
                           input logic [9:0] x,y,
                           output logic [7:0] ch);

        logic [7:0] m0,m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14;
        logic [7:0] l0,l1,l2,l3,l4,l5,l6,l7,l8,l9,l10,l11,l12,l13,l14;
        logic hasntdone;
        always_ff@(posedge clk)
         if (~done)
                begin
                clkdone<=0;
                hasntdone<=1;
                end
            else if (hasntdone&done)
            begin
                hasntdone<=0;
                clkdone<=1;
            end
            else clkdone<=0;

        megaflop
msg_flop(clk,clkdone,msgEn,char,m0,m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m
13,m14);
```

```
        megaflop
loc_flop(clk,clkdone,~msgEn,char,l0,l1,l2,l3,l4,l5,l6,l7,l8,l9,l10,l11,l12,
l13,l14);


        logic[7:0] chm,chl;
        charLocX
locxm(x,m0,m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,chm);

        charLocX
locxl(x,l0,l1,l2,l3,l4,l5,l6,l7,l8,l9,l10,l11,l12,l13,l14,chl);
        charLocY locy(y,chm,chl,ch);

endmodule

module charLocX #(parameter CHW = 10'd32,
                                                CHH = 10'd32,
                                                HBP = 10'd48,
//H back porch
                                                HACTIVE = 10'd640,
//Horizontal active section
                                                HFP = 10'd16,
//Horizontal front porch
                                                HSYN = 10'd96,
//H sync time
                                                HMAX = HACTIVE + HFP +
HSYN + HBP,
                                                VBP = 10'd32,
                                                VACTIVE = 10'd480,
                                                VFP = 10'd11,
                                                VSYN = 10'd2,
                                                VMAX = VACTIVE + VFP +
VSYN + VBP)

                        (input logic [9:0] x,
                         input logic [7:0]
m0,m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,
                         output logic [7:0] ch);

        always_comb
            if          (0<x&x<3*CHW )          ch=00;
```

```
            else if((4*CHW<x) & (x<(5*CHW-1))) ch=m0; //Dont do <= this
gets rid of overlap
            else if((5*CHW<x) & (x<(6*CHW-1))) ch=m1;
            else if((6*CHW<x) & (x<(7*CHW-1))) ch=m2;
            else if((7*CHW<x) & (x<(8*CHW-1))) ch=m3;
            else if((8*CHW<x) & (x<(9*CHW-1))) ch=m4;
            else if((9*CHW<x) & (x<(10*CHW-1))) ch=m5;
            else if((10*CHW<x) & (x<(11*CHW-1))) ch=m6;
            else if((11*CHW<x) & (x<(12*CHW-1))) ch=m7;
            else if((12*CHW<x) & (x<(13*CHW-1))) ch=m8;
            else if((13*CHW<x) & (x<(14*CHW-1))) ch=m9;
            else if((14*CHW<x) & (x<(15*CHW-1))) ch=m10;
            else if((15*CHW<x) & (x<(16*CHW-1))) ch=m11;
            else if((16*CHW<x) & (x<(17*CHW-1))) ch=m12;
            else if((17*CHW<x) & (x<(18*CHW-1))) ch=m13;
            else if((18*CHW<x) & (x<(19*CHW-1))) ch=m14;
            else ch=00;
endmodule

module charLocY #(parameter CHW = 10'd32,
                                                    CHH = 10'd32,
                                                    HBP = 10'd48,
//H back porch
                                                    HACTIVE = 10'd640,
//Horizontal active section
                                                    HFP = 10'd16,
//Horizontal front porch
                                                    HSYN = 10'd96,
//H sync time
                                                    HMAX = HACTIVE + HFP +
HSYN + HBP,
                                                    VBP = 10'd32,
                                                    VACTIVE = 10'd480,
                                                    VFP = 10'd11,
                                                    VSYN = 10'd2,
                                                    VMAX = VACTIVE + VFP +
VSYN + VBP)
                        (input logic [9:0] y,
                         input logic [7:0] chm, chl,
                         output logic [7:0] ch);

        always_comb
            if          (0<y&y<3*CHH )          ch=00;
```

```
            else if((4*CHH<=y) & (y<(5*CHH))) ch=chm;
            else if((8*CHH<=y) & (y<(9*CHH))) ch=chl;
            else ch=00;
endmodule
```

Appendix IV array of ones and zeros that makeup 32x 32 bit font

```
//A
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000011000000000000000
00000000000000111100000000000000
00000000000000111100000000000000
00000000000001111100000000000000
00000000000001101110000000000000
00000000000011100110000000000000
00000000000011000111000000000000
00000000000011000011000000000000
00000000000110000011000000000000
00000000000110000011100000000000
00000000001110000001100000000000
00000000001100000001110000000000
00000000001100000001110000000000
00000000011000000000110000000000
00000000011000000000111000000000
00000001110000000000011000000000
00000001100000000000011100000000
00000001100000000000011100000000
00000011111111111111111100000000
00000011111111111111111110000000
00000111000000000000001110000000
00000110000000000000000111000000
00000110000000000000000111000000
00001110000000000000000011000000
00001100000000000000000011100000
00011100000000000000000001100000
00011100000000000000000001110000
```

```
00001100000000000000000001110000
00000000000000000000000000000000
00000000000000000000000000000000
//B
00000000000000000000000000000000
00000000000000000000000000000000
00000000111111111000000000000000
00000000111111111111000000000000
00000000111111111111110000000000
00000000110000000001111000000000
00000000110000000000111000000000
00000000110000000000011100000000
00000000110000000000011100000000
00000000110000000000011100000000
00000000110000000000011100000000
00000000110000000000011100000000
00000000110000000000011100000000
00000000110000000000111000000000
00000000110000000001111000000000
00000000111111111111110000000000
00000000111111111111100000000000
00000000111111111111110000000000
00000000110000000000111100000000
00000000110000000000011110000000
00000000110000000000001110000000
00000000110000000000001111000000
00000000110000000000001111000000
00000000110000000000000111000000
00000000110000000000000111000000
00000000110000000000000011000000
00000000110000000000000011000000
00000000110000000000000111000000
00000000110000000000000111000000
00000000110000000000001111000000
00000000110000000000011110000000
00000000111111111111111100000000
00000000111111111111111000000000
00000000111111111111000000000000
00000000000000000000000000000000
00000000000000000000000000000000
//C
00000000000000000000000000000000
00000000000000000000000000000000
```

```
00000000000000000000000000000000
00000000000011111111100000000000
00000000001111111111111000000000
00000000011111000000111110000000
00000000111000000000011110000000
00000001110000000000001110000000
00000011100000000000000111000000
00000011100000000000000000000000
00000011000000000000000000000000
00000111000000000000000000000000
00000111000000000000000000000000
00000111000000000000000000000000
00000110000000000000000000000000
00000110000000000000000000000000
00000110000000000000000000000000
00000110000000000000000000000000
00000110000000000000000000000000
00000111000000000000000000000000
00000111000000000000000000000000
00000111000000000000000000000000
00000110000000000000000000000000
00000111000000000000000000000000
00000111000000000000000111000000
00000011100000000000001110000000
00000011110000000000001110000000
00000001111000000000111100000000
00000000011111000011111000000000
00000000000111111111110000000000
00000000000001111110000000000000
00000000000000000000000000000000
//D
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000001111111110000000000000000
00000011111111111111000000000000
00000011100001111111100000000000
00000011100000000001111000000000
00000011100000000000111110000000
```

```
00000011100000000000001110000000
00000011100000000000000111000000
00000011100000000000000111000000
00000011100000000000000011100000
00000011100000000000000011100000
00000011100000000000000001100000
00000011100000000000000001100000
00000011100000000000000001110000
00000011100000000000000001110000
00000011100000000000000001110000
00000011100000000000000001100000
00000011100000000000000001100000
00000011100000000000000001100000
00000011100000000000000011100000
00000011100000000000000011100000
00000011100000000000000111000000
00000011100000000000000111000000
00000011100000000000001110000000
00000011100000000000111100000000
00000011100000000011111000000000
00000011111111111111110000000000
00000011111111111110000000000000
00000000000000000000000000000000
00000000000000000000000000000000
//E
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000111111111111111100000000
00000000111111111111111100000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
```

```
00000000111000000000000000000000
00000000111111111111111000000000
00000000111111111111111000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111000000000000000000000
00000000111111111111111110000000
00000000111111111111111110000000
00000000000000000000000000000000
00000000000000000000000000000000
//F
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000111111111111111110000000
00000000111111111111111110000000
00000000111000000000000000000000
00000000110000000000000000000000
00000000110000000000000000000000
00000000110000000000000000000000
00000000110000000000000000000000
00000000110000000000000000000000
00000000110000000000000000000000
00000000110000000000000000000000
00000000110000000000000000000000
00000000111111111111111100000000
00000000111111111111111100000000
00000000111000000000000000000000
00000000110000000000000000000000
00000000110000000000000000000000
```

```
00000000011000000000000000000000
00000000011000000000000000000000
00000000011000000000000000000000
00000000011000000000000000000000
00000000011000000000000000000000
00000000011000000000000000000000
00000000011000000000000000000000
00000000011000000000000000000000
00000000011000000000000000000000
00000000011000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
//G
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000001111111110000000000
00000000000111111111111000000000
00000000001111000001111110000000
00000000011100000000001110000000
00000000111000000000000111000000
00000001110000000000000011000000
00000001110000000000000011000000
00000001100000000000000000000000
00000011100000000000000000000000
00000011000000000000000000000000
00000011000000000000000000000000
00000011000000000000000000000000
00000111000000000000000000000000
00000111000000000011111111000000
00000111000000000111111111100000
00000111000000000111111111100000
00000011000000000000000011100000
00000011000000000000000011100000
00000111000000000000000011100000
00000111000000000000000011100000
00000011100000000000000011100000
00000011100000000000000011100000
00000011100000000000000011100000
```

```
00000000111100000000000011100000
00000000011110000000000111100000
00000000001111110000111111000000
00000000000011111111111100000000
00000000000000111111100000000000
00000000000000000000000000000000
//H
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000011000000000000000010000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011111111111111111111000000
00000011111111111111111111000000
00000011100000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000011000000000000000111000000
00000000000000000000000000000000
00000000000000000000000000000000
```

```
//I
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
//J
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000100000000000000
00000000000000001110000000000000
```

```
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00001100000000001100000000000000
00001110000000001100000000000000
00001110000000011100000000000000
00001110000000111000000000000000
00000111100001111000000000000000
00000011111001111100000000000000
00000001111111110000000000000000
00000000111111000000000000000000
00000000000000000000000000000000
//K
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000110000000000011100000000
00000000110000000000111000000000
00000000110000000001110000000000
00000000110000000011110000000000
00000000110000000111100000000000
00000000110000001111000000000000
00000000110000011110000000000000
```

```
00000000110000111000000000000000
00000000110001110000000000000000
00000000110011100000000000000000
00000000110111000000000000000000
00000000111111000000000000000000
00000000111110000000000000000000
00000000110111000000000000000000
00000000110011100000000000000000
00000000110001110000000000000000
00000000110000111000000000000000
00000000110000011100000000000000
00000000110000001110000000000000
00000000110000000111000000000000
00000000110000000011100000000000
00000000110000000001110000000000
00000000110000000000111000000000
00000000110000000000011100000000
00000000110000000000001110000000
00000000000000000000000000000000
00000000000000000000000000000000
//L
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
```

```
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001110000000000000000000
00000000001111111111111111000000
00000000001111111111111111000000
00000000000000000000000000000000
00000000000000000000000000000000
//M
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00111100000000000000000000111100
00111100000000000000000000111100
00111110000000000000000001111100
00111110000000000000000001111100
00110110000000000000000011001100
00110011000000000000000011001100
00110011000000000000000111001100
00110011100000000000000110001100
00110001100000000000000110001100
00110001110000000000001100001100
00110000110000000000001100001100
00110000110000000000011100001100
00110000111000000000011000001100
00110000011000000000111000001100
00110000001110000000110000001100
00110000001100000001110000001100
00110000001110000001110000001100
00110000000110000001100000001100
00110000000110000011100000001100
00110000000111000011000000001100
```

```
00110000000011000111000000001100
00110000000011100110000000001100
00110000000011011110000000001100
00110000000011111110000000001100
00110000000011111100000000001100
00110000000001111100000000001100
00110000000001110000000000001100
00000000000000000000000000000000
00000000000000000000000000000000
//N
00000000000000000000000000000000
00000000000000000000000000000000
00000011000000000000000010000000
00000011100000000000000011000000
00000011110000000000000011000000
00000011110000000000000011000000
00000011111000000000000011000000
00000011011100000000000011000000
00000011001100000000000011000000
00000011001110000000000011000000
00000011000110000000000011000000
00000011000111000000000011000000
00000011000011100000000011000000
00000011000001100000000011000000
00000011000001110000000011000000
00000011000000110000000011000000
00000011000000111000000011000000
00000011000000011000000011000000
00000011000000001100000011000000
00000011000000001110000011000000
00000011000000000111000011000000
00000011000000000011000011000000
00000011000000000111000011000000
00000011000000000111000011000000
00000011000000000011111111000000
00000011000000000000111111000000
00000011000000000000111111000000
00000011000000000000011111000000
```

```
00000011000000000000000111000000
00000000000000000000000000000000
00000000000000000000000000000000
//O
00000000000000000000000000000000
00000000000000000000000000000000
00000000000011111111000000000000
00000000011111111111110000000000
00000001111110000011111000000000
00000001111100000000111100000000
00000011111000000000001110000000
00000111110000000000000111000000
00000111100000000000000111000000
00000111000000000000000011100000
00001110000000000000000011100000
00001110000000000000000001100000
00001100000000000000000001110000
00001100000000000000000001110000
00001100000000000000000001110000
00001100000000000000000001110000
00001100000000000000000001110000
00001100000000000000000001110000
00001110000000000000000001100000
00001110000000000000000011100000
00000110000000000000000011100000
00000111000000000000000111000000
00000111100000000000000111000000
00000011110000000000001110000000
00000001111000000000011110000000
00000000111110000001111000000000
00000000011111111111110000000000
00000000000111111111000000000000
00000000000000000000000000000000
00000000000000000000000000000000
//P
00000000000000000000000000000000
00000000000000000000000000000000
```

```
00000000111111110000000000000000
00000001111111111111100000000000
00000001111111111111111000000000
00000001110000000000111100000000
00000001110000000000011100000000
00000001110000000000001110000000
00000001110000000000001110000000
00000001110000000000000110000000
00000001110000000000000110000000
00000001110000000000000110000000
00000001110000000000001110000000
00000001110000000000001110000000
00000001110000000000111100000000
00000001110000000011111000000000
00000001111111111111110000000000
00000001111111111100000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
//Q
00000000000000000000000000000000
00000000000000000000000000000000
00000000000111111110000000000000
00000000011111111111100000000000
00000000111111000011111000000000
00000001110000000000111100000000
00000001110000000000011110000000
00000011100000000000001110000000
```

```
00000111000000000000000111000000
00001110000000000000000111000000
00001110000000000000000011100000
00001100000000000000000011100000
00011100000000000000000011100000
00011100000000000000000001100000
00011100000000000000000001100000
00011100000000000000000001100000
00011100000000000000000001100000
00011100000000000000000001100000
00011100000000000000000001100000
00011100000000000000000011100000
00001100000000000000000011100000
00001110000000000000000011100000
00001110000000000000000111000000
00000111000000000000000111000000
00000111000000000000001110000000
00000011100000000000011100000000
00000001110000000000111100000000
00000000111100000011111000000000
00000000011111111111111111000000
00000000001111111100001111100000
00000000000000000000000001111000
00000000000000000000000000111000
//R
00000000000000000000000000000000
00000000000000000000000000000000
00000000111111111000000000000000
00000000111111111111110000000000
00000000111111111111111100000000
00000000110000000000011110000000
00000000110000000000001110000000
00000000110000000000000111000000
00000000110000000000000111000000
00000000110000000000000111000000
00000000110000000000000111000000
00000000110000000000000111000000
00000000110000000000000110000000
00000000110000000000001110000000
00000000110000000000011100000000
```

```
00000000111111111111111000000000
00000000111111111111100000000000
00000000111111111111110000000000
00000000110000000001111000000000
00000000110000000000011100000000
00000000110000000000001100000000
00000000110000000000011110000000
00000000110000000000000110000000
00000000110000000000000110000000
00000000110000000000000110000000
00000000110000000000000111000000
00000000110000000000000111000000
00000000110000000000000011000000
00000000110000000000000011000000
00000000110000000000000011000000
00000000110000000000000011000000
00000000000000000000000000000000
00000000000000000000000000000000
//S
00000000000000000000000000000000
00000000000000000000000000000000
00000000000011111100000000000000
00000000001111111111000000000000
00000000111100001111100000000000
00000001110000000011110000000000
00000001110000000001110000000000
00000001110000000000110000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001110000000000000000000000
00000001111000000000000000000000
00000001111100000000000000000000
00000000111111000000000000000000
00000000011111110000000000000000
00000000000111111110000000000000
00000000000001111111000000000000
00000000000000011111100000000000
00000000000000000111100000000000
00000000000000000001110000000000
```

```
00000000000000000000011100000000
00000000000000000000011100000000
00000000000000000000001100000000
00000011000000000000001100000000
00000011000000000000011100000000
00000011100000000000011100000000
00000011110000000000111000000000
00000001111000000001111000000000
00000000111111111111110000000000
00000000001111111111000000000000
00000000000000000000000000000000
00000000000000000000000000000000
//T
00000000000000000000000000000000
00000000000000000000000000000000
00000011111111111111111110000000
00000011111111111111111110000000
00000011111111111111111110000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
```

```
00000000000001110000000000000000
00000000000001110000000000000000
00000000000001110000000000000000
00000000000001100000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
//U
00000000000000000000000000000000
00000000000000000000000000000000
00000010000000000000000010000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011000000000000000110000000
00000011100000000000001110000000
00000011100000000000001110000000
00000011100000000000011100000000
00000011110000000000111100000000
00000001111100000001111100000000
00000000111111111111110000000000
00000000001111111110000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

```
//V
00000000000000000000000000000000
00000000000000000000000000000000
00000110000000000000000001000000
00000110000000000000000011000000
00000111000000000000000011000000
00000111000000000000000111000000
00000011000000000000000110000000
00000011100000000000000110000000
00000001100000000000001110000000
00000001100000000000001100000000
00000001110000000000001100000000
00000000110000000000011100000000
00000000111000000000011000000000
00000000111000000000011000000000
00000000011000000000111000000000
00000000011100000000110000000000
00000000011100000000110000000000
00000000001100000001100000000000
00000000001110000001100000000000
00000000000110000001100000000000
00000000000110000011000000000000
00000000000111000011000000000000
00000000000110001110000000000000
00000000000110001100000000000000
00000000000111001100000000000000
00000000000011011000000000000000
00000000000011111000000000000000
00000000000001111000000000000000
00000000000001110000000000000000
00000000000001110000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
//W
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

```
00000000000000000000000000000000
01000000000001110000000000000110
01100000000001110000000000000110
01100000000001111000000000000110
01100000000001101100000000001100
01110000000001101100000000001100
00110000000011001100000000001100
00110000000010001100000000001000
00110000000011000110000000011000
00011000000011000011000000011000
00011000000010000011000000011000
00011000000110000011000000010000
00011000000110000001100000110000
00001100000100000001100000110000
00001100001100000001100000110000
00001100001100000000110000100000
00001100001000000000110001100000
00000110011000000000110001100000
00000110011000000000011001100000
00000110010000000000011001000000
00000110110000000000011011000000
00000011110000000000001111000000
00000011110000000000001110000000
00000011100000000000001110000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
//X
00000000000000000000000000000000
00000000000000000000000000000000
00000110000000000000000010000000
00000111000000000000001110000000
00000011100000000000001110000000
00000001110000000000011100000000
00000001110000000000111000000000
00000000111000000000110000000000
00000000011100000001110000000000
00000000011100000011100000000000
00000000001110000011000000000000
```

```
00000000000111000111000000000000
00000000000111001110000000000000
00000000000011111100000000000000
00000000000001111000000000000000
00000000000001111000000000000000
00000000000001111000000000000000
00000000000011111100000000000000
00000000000011001110000000000000
00000000000111001110000000000000
00000000001110000111000000000000
00000000011100000011100000000000
00000000111000000011100000000000
00000001110000000001110000000000
00000011100000000000111000000000
00000111000000000000011100000000
00001110000000000000001110000000
00000000000000000000000000000000
00000000000000000000000000000000
//Y
00000000000000000000000000000000
00000000000000000000000000000000
00000110000000000000000001000000
00000111000000000000000111000000
00000011100000000000000110000000
00000011100000000000001110000000
00000001110000000000001100000000
00000000111000000000011000000000
00000000111000000000111000000000
00000000011100000001110000000000
00000000001110000011100000000000
00000000000111000011000000000000
00000000000111100111000000000000
00000000000011101110000000000000
00000000000001111100000000000000
00000000000001111100000000000000
```

```
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000000011000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
//Z
00000000000000000000000000000000
00000000000000000000000000000000
00000000111111111111111110000000
00000001111111111111111111000000
00000001111111111111111111000000
00000000000000000000001110000000
00000000000000000000001110000000
00000000000000000000011100000000
00000000000000000000111000000000
00000000000000000000111000000000
00000000000000000001110000000000
00000000000000000001100000000000
00000000000000000011100000000000
00000000000000000111000000000000
00000000000000001110000000000000
00000000000000001110000000000000
00000000000000011100000000000000
00000000000000111000000000000000
00000000000000111000000000000000
00000000000001110000000000000000
00000000000001110000000000000000
00000000000011100000000000000000
```

```
00000000001110000000000000000000
00000000001110000000000000000000
00000000011100000000000000000000
00000000011000000000000000000000
00000000111111111111111110000000
00000000111111111111111111000000
00000000111111111111111111000000
00000000000000000000000000000000
00000000000000000000000000000000
//
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```