CDA 4203/4203L
Spring 2015
Computer System Design

Final Project Report
Due by 5pm, Thursday, 30th April

Today's Date:	4/28/15		
Team Member Names:	Sean Murphy	Austin Matthew	Jesse Walton
Team Member U #'s:	U49850246	U89904116	U89823440
Work Distribution	Briefly explain each team member's contribution. Sean Murphy – Memory, Integration, bug fixes Austin Matthew – LCD, menu system Jesse Walton – Audio, Report		

Feedback: Your feedback is extremely important to improve the mini-project for future course offerings.

Total Number of Person Hours Spent:	*Estimate the number of hours spent by each team member and add the three numbers.* Sean Murphy: 2 + 2 + 6 + 8 + 14 + 12 = 44 Austin Matthew: 2 + 2 + 6 + 8 + 14 + 12 = 44 Jesse Walton: 2 + 2 + 6 + 8 + 12 + 12 = 42 Total: 44 + 44 + 42 = 130 hours
Exercise Difficulty: (Easy, Average, Hard)	Hard
Issues You ran into:	*List all problem/issues you faced while doing this project.* *(please use bulletized list)* ● *Audio codec – typo* ● *Verilog (General) – implementation and testing* ● *LCD – timing issues, logic*
Any Suggestions to improve this project:	*Better resources and training on debugging workflow* *Introduce components in prior labs.* *Training on best practices of how to use new components*

Overview

Describe the overall functionality of the system. Describe any and all relevant interfaces. Maximum 1 page.

The system uses the ram, ac97, buttons and lcd to implement an audio recorder that can record and play back 8 messages at up to 4 minutes each. This is accomplished by using a top level Verilog module to handle IO and to connect all sub-modules together.

Powering On: When the system is turned on, the user is greeted with a welcome message that remains on the screen until a button is pressed.

Top Menu: Immediately after a button is pressed, the welcome menu is removed and the top menu is shown. The menu items are: **Play**, **Record**, **Delete**, **Delete All** and **Volume.** The user can navigate to each item by pressing the navigation buttons.

Play: Selecting play brings you to a message select screen and allows you to select messages 1 through 8.

Record: Selecting record brings you to a message select screen and allows you to select messages 1 through 8. Upon selection of a message, the device will begin recording. While recording, the user may stop recording or return to top menu.

Delete: Selecting delete brings you to a message select screen and allows you to select messages 1 through 8. Selecting a message brings up a confirmation window. Confirming deletes the message from memory by resetting the timing of the recording.

Delete All: Selecting delete all brings you to a confirmation menu. Confirming this option will delete all messages from the system.

Volume: Selecting volume will bring a new window up to allow volume control using the left and right buttons.

System Block Diagram

Show a detailed diagram of the system design. This should include all major blocks and ICs, and show all datapaths and control signals.

[see diagram 1 and diagram 2]

System Functional Flow

Include a detailed flow diagram, FSMD, pseudo code, or whatever it takes to show precisely how the system works. This needs to be detailed out at the control line level, a sequence of specific events that accomplish that functionality. If you are using an FSM, for instance, show each state and which controls are asserted in that state, along with what causes each state transition. If you are using pseudo code, again show the case statements, loops, decisions, etc., that lead to each control line being asserted.

On system power up, the system goes through initialization. The lcd initializes to 4 bit mode using the lcd state machine which is setup to output the correct timings for the function and mode settings. After the initial settings are complete, the lcd displays a welcome message until a button is pressed. The AC97 codec is also initialized using the ac97 commands module. This module uses a state machine to iterate through setting all relevant settings like source and microphone. After both the lcd and ac97 are initialized, the system is ready for user input and will leave the welcome message machine when it is received, taking the user to the top menu.

During run time, the system latches button presses and passes them into a picoblaze implementation of a state machine to generate an encoded output that drives another picoblaze. The data is passed into the secondary picoblaze as an 8 bit value with the top 4 bits representing the lcd menu to display and the bottom 4 bits representing the positioning of the cursor. The second picoblaze uses this data to generate the appropriate LCD commands, letter by letter, and passes them into a final verilog module which takes the ascii values and converts them into the timing diagrams that run the lcd. The system remains responsive to button presses because of the high clock frequency of the system and the relatively low amount of audio cycles needed for recording and playback.

The ac97 always sends and receives data serially and interfaces with the memory directly for playback and recording. Each of those states are triggered within the first picoblaze, as it keeps track of the current position the user is in the system

Code for Audio Codec

Include source code for the main controller. The length of this will vary widely; depending on how much of your design's functionality is in software.

```
1   `timescale 1ns / 1ps
2
3
4   module streaming_file_system(
5       command, mem_full, fnumber, ready,
6       data_in, data_out, sync,
7       ram_clock,
8       this_reset,
9       clk_out,
10
11      // RAM hardware pins
12      hw_ram_rasn, hw_ram_casn,
13        hw_ram_wen, hw_ram_ba, hw_ram_udqs_p, hw_ram_udqs_n,
    hw_ram_ldqs_p,hw_ram_ldqs_n, hw_ram_udm, hw_ram_ldm, hw_ram_ck,
    hw_ram_ckn, hw_ram_cke, hw_ram_odt,
14        hw_ram_ad, hw_ram_dq, hw_rzq_pin, hw_zio_pin
15
16      );
17
18    parameter MESSAGE_BITS = 3;
19    parameter MEM_ADDRESS_BITS = 26;
20
21    input [2:0] command;
22    output reg mem_full;
23    input [2:0] fnumber;
24    // input [MEM_ADDRESS_BITS-MESSAGE_BITS-1:0] flen;
25    input [7:0] data_in;
26    output reg [7:0] data_out;
27    input sync;
28    input ram_clock;
29    input this_reset;
30    output reg ready;
31
32    output clk_out;
33
34
35    reg [MESSAGE_BITS-1:0] fnumber_addr;
36    reg [MEM_ADDRESS_BITS-MESSAGE_BITS-1:0] offset;
```

```
37      wire [MEM_ADDRESS_BITS-1:0] full_ram_addr;
38      assign full_ram_addr = {fnumber_addr, offset};
39
40      reg [MEM_ADDRESS_BITS-MESSAGE_BITS-1:0] fsizes [0:7];  // 8 long array
        of 23 bit numbers
41
42      wire [MEM_ADDRESS_BITS-1:0] max_ram_address;
43      reg [7:0] ram_data_in;
44      wire [7:0] ram_data_out;
45      reg ram_read_request;
46      reg ram_read_ack;
47      wire ram_rdy;
48      wire ram_rd_data_pres;
49      reg ram_write_enable;
50
51        // RAM hardware bullshit
52        output hw_ram_rasn;
53        output hw_ram_casn;
54        output hw_ram_wen;
55        output [2:0] hw_ram_ba;
56        inout hw_ram_udqs_p;
57        inout hw_ram_udqs_n;
58        inout hw_ram_ldqs_p;
59        inout hw_ram_ldqs_n;
60        output hw_ram_udm;
61        output hw_ram_ldm;
62        output hw_ram_ck;
63        output hw_ram_ckn;
64        output hw_ram_cke;
65        output hw_ram_odt;
66        output [12:0] hw_ram_ad;
67        inout [15:0] hw_ram_dq;
68        inout    hw_rzq_pin;
69        inout hw_zio_pin;
70
71        // Instantiate the RAM
72      ram_interface_wrapper myram (
73        .address(full_ram_addr),
74        .data_in(ram_data_in),
75        .write_enable(ram_write_enable),
76        .read_request(ram_read_request),
77        .read_ack(ram_read_ack),
78        .data_out(ram_data_out),
79        .reset(this_reset),
80        .clk(ram_clock),
```

```
81
82          .hw_ram_rasn(hw_ram_rasn),
83          .hw_ram_casn(hw_ram_casn),
84          .hw_ram_wen(hw_ram_wen),
85          .hw_ram_ba(hw_ram_ba),
86          .hw_ram_udqs_p(hw_ram_udqs_p),
87          .hw_ram_udqs_n(hw_ram_udqs_n),
88          .hw_ram_ldqs_p(hw_ram_ldqs_p),
89          .hw_ram_ldqs_n(hw_ram_ldqs_n),
90          .hw_ram_udm(hw_ram_udm),
91          .hw_ram_ldm(hw_ram_ldm),
92          .hw_ram_ck(hw_ram_ck),
93          .hw_ram_ckn(hw_ram_ckn),
94          .hw_ram_cke(hw_ram_cke),
95          .hw_ram_odt(hw_ram_odt),
96          .hw_ram_ad(hw_ram_ad),
97          .hw_ram_dq(hw_ram_dq),
98          .hw_rzq_pin(hw_rzq_pin),
99          .hw_zio_pin(hw_zio_pin),
100
101         .clkout(clk_out),
102         .sys_clk(clk_out),
103         .rdy(ram_rdy),
104         .rd_data_pres(ram_rd_data_pres),
105         .max_ram_address()
106       );
107
108
109        parameter c_play = 1;
110        parameter c_pause = 2;
111        parameter c_record = 3;
112        parameter c_delete = 4;
113        parameter c_delete_all = 5;
114
115
116      parameter s_wait_command  = 4'b0000;
117      parameter s_finish  = 4'b0001;
118      parameter s_delete  = 4'b0010;
119      parameter s_delete_all  = 4'b0011;
120      parameter s_play   = 4'b0100;
121      parameter s_play2  = 4'b0101;
122      parameter s_play3  = 4'b0110;
123      parameter s_record  = 4'b0111;
124      parameter s_record2  = 4'b1000;
125      parameter s_record3 = 4'b1001;
```

```
126
127     (* FSM_ENCODING="SEQUENTIAL", SAFE_IMPLEMENTATION="NO" *)
        reg [3:0] state =s_finish;
128
129     always@(posedge clk_out)
130       if (this_reset) begin
131         state <= s_wait_command;
132         ram_read_request <= 0;
133         ram_read_ack <= 0;
134         ram_write_enable <= 0;
135         fsizes[0] <= 0;
136         fsizes[1] <= 0;
137         fsizes[2] <= 0;
138         fsizes[3] <= 0;
139         fsizes[4] <= 0;
140         fsizes[5] <= 0;
141         fsizes[6] <= 0;
142         fsizes[7] <= 0;
143
144
145       end
146       else
147         (* FULL_CASE, PARALLEL_CASE *) case (state)
148           s_wait_command : begin
149             if (command == c_play) begin
150               state <= s_play;
151               offset <= 0;
152               fnumber_addr <= fnumber;
153               ready <= 0;
154             end
155
156             else if (command == c_record) begin
157               state <= s_record;
158               fnumber_addr <= fnumber;
159               offset <= 0;
160               ready <= 0;
161             end
162
163             else if (command == c_delete) begin
164               state <= s_delete;
165               fnumber_addr <= fnumber;
166               ready <= 0;
167             end
168
169             else if (command == c_delete_all) begin
```

```
170                 state <= s_delete_all;
171                 fnumber_addr <= fnumber;
172                 ready <= 0;
173               end
174
175               else begin
176                   ready <= 1;
177                   ram_read_request <= 0;
178                   ram_read_ack <= 0;
179               end
180
181           end
182
183           s_finish : begin
184               if (!command) begin
185                   mem_full <= 0;
186                   ready <= 0;
187                   state <= s_wait_command;
188               end
189           end
190
191           s_delete : begin
192               fsizes[fnumber_addr] <= 0;
193               state <= s_finish;
194           end
195
196           s_delete_all : begin
197               fsizes[0] <= 0;
198               fsizes[1] <= 0;
199               fsizes[2] <= 0;
200               fsizes[3] <= 0;
201               fsizes[4] <= 0;
202               fsizes[5] <= 0;
203               fsizes[6] <= 0;
204               fsizes[7] <= 0;
205               state <= s_finish;
206           end
207
208           s_play : begin
209               ram_read_ack <= 0;
210               if (offset == fsizes[fnumber_addr])
      // Are we at the end of of the message?
211                   state <= s_finish;
212
213               else if (!sync)
```

```
        // wait for sync to drop low again
214                 state <= s_play2;
215             end
216
217             s_play2 : begin
218                 if (sync && (command == c_play)) begin
        // Wait for the frame sync to get the next byte
219                     state <= s_play3;
220                     ram_read_request <= 1;
221                 end
222                 else if (command == c_pause)
        // If pause, just stay in this block
223                     state <= s_play2;
224                 else if (command != c_play)
        // Something other than play means finish up and return to function select
225                     state <= s_finish;
226                 else
227                     state <= s_play2;
228             end
229
230             s_play3 : begin
231                 if (!ram_rd_data_pres)
        // Wait for RAM te present the data
232                     state <= s_play3;
233                 else begin
234                     data_out <= ram_data_out;
235                     offset <= offset + 1;
236                     ram_read_request <= 0;
        // Do I need to deassert this more quickly?
237                     ram_read_ack <= 1;
238                     state <= s_play;
239                 end
240             end
241
242
243             s_record : begin
244                 if (command != c_record) begin
        // When we finish recording, write out our file size
245                     state <= s_finish;
246                     fsizes[fnumber_addr] <= offset;
247                 end
248
249                 else if (sync) begin
        // wait for frame sync to grab data and write
250                     ram_data_in <= data_in;
```

```
251                     ram_write_enable <= 1;
252                     state <= s_record2;
253                 end
254             end
255
256             s_record2 : begin
257                 ram_write_enable <= 0;
258                 offset <= offset + 1;
259                 state <= s_record3;
260             end
261
262             s_record3 : begin
263                 if (offset == 0) begin
        // praying for rollover - max message length
264                     mem_full <= 1;
265                     fsizes[fnumber_addr] <= -1;
        // please please wrap around
266                     state <= s_finish;
267                 end
268                 else if (!sync)
        // wait for sync to drop
269                     state <= s_record;
270             end
271
272         endcase
273
274     endmodule
275
```

Code for LCD Display

Include source code for the main controller. The length of this will vary widely; depending on how much of your design's functionality is in software.

// Picoblaze #1 – processes current state of system, based on button presses. Sends 8 bit encoded messages to picoblaze #2. Bits [7:4] refer to the current menu (welcome message, top, play, record, delete, delete all, message select, confirmation)

```
CONSTANT CONTROL, 00

CONSTANT READY_SIG, 07

CONSTANT MEM_FULL, 02

CONSTANT COMMAND, 03

CONSTANT FILE, 04

CONSTANT VOLUME_OUTPUT, 05

CONSTANT LCD_OUTPUT, 06


NAMEREG s0, MENU_STATE

NAMEREG s1, FILE_STATE

NAMEREG s2, TWO_STATE

NAMEREG s3, MASTER_STATE

NAMEREG s4, SEANS_CONSTANT

NAMEREG s5, VOLUME

NAMEREG s6, LCD_BUILD

NAMEREG s7, READY_REG

NAMEREG s8, COMMAND_REG

NAMEREG s9, FILE_REG

NAMEREG sA,CONTROLREG
```

```
ENABLE INTERRUPT


WELCOME:                ; WELCOME MESSAGE
        LOAD LCD_BUILD, A0
        OUTPUT LCD_BUILD, LCD_OUTPUT
        CALL WAIT_STATE
        CALL INPUT_LOADER
MAIN:                   ;MAIN MENU
        LOAD COMMAND_REG, FF
        OUTPUT COMMAND_REG, COMMAND
        CALL PUSH
MAIN_MENU_RS:
        LOAD MENU_STATE, 00
MAIN_MENU:
        LOAD MASTER_STATE, 00
        CALL WAIT_STATE
        CALL INPUT_LOADER
        COMPARE CONTROLREG, 01
        JUMP Z, UP_MAIN
        COMPARE CONTROLREG, 02
        JUMP Z, DOWN_MAIN
        COMPARE CONTROLREG, 04
        JUMP Z, BACK
        COMPARE CONTROLREG, 08
        JUMP Z, SELECT_MAIN
```

```
        JUMP MAIN_MENU


;//////////////////////////MAIN UP/DOWN//////////////////
UP_MAIN:
        COMPARE MENU_STATE, 04
        JUMP NZ, UP_GOOD
        JUMP MAIN_MENU
        UP_GOOD:
        ADD MENU_STATE, 01
        CALL PUSH
        JUMP MAIN_MENU
DOWN_MAIN:
        COMPARE MENU_STATE, 00
        JUMP NZ, DOWN_GOOD
        JUMP MAIN_MENU
        DOWN_GOOD:
        SUB MENU_STATE, 01
        CALL PUSH
        JUMP MAIN_MENU
;////////////////////////////SELECT MAIN/////////////////
SELECT_MAIN:            ;ROUTES TO MASTER SWITCH
        COMPARE MENU_STATE, 00
        JUMP Z, MASTER_PLAY
        COMPARE MENU_STATE, 01
        JUMP Z, MASTER_RECORD
```

```
        COMPARE MENU_STATE, 02
        JUMP Z, MASTER_DELETE
        COMPARE MENU_STATE, 03
        JUMP Z, MASTER_DEL_ALL
        COMPARE MENU_STATE, 04
        JUMP Z, MASTER_VOL
;///////MASTER SWITCH////////////////
MASTER_MENU:
        LOAD MASTER_STATE, 00
        CALL PUSH
        JUMP MAIN_MENU_RS
MASTER_PLAY:
        LOAD MASTER_STATE,01
        CALL PUSH
        JUMP FILE_MENU_RS
MASTER_RECORD:
        LOAD MASTER_STATE,02
        CALL PUSH
        JUMP FILE_MENU_RS
MASTER_DELETE:
        LOAD MASTER_STATE,03
        CALL PUSH;
        JUMP FILE_MENU_RS
MASTER_PLAYING:
        LOAD MASTER_STATE,04
```

```
        CALL PUSH
        JUMP TWO_OPTION_MENU_RS
MASTER_DELETING:
        LOAD MASTER_STATE,05
        CALL PUSH
        JUMP TWO_OPTION_MENU_RS
MASTER_VOL:
        LOAD MASTER_STATE,06
        CALL PUSH
        JUMP TWO_OPTION_MENU_RS
MASTER_DEL_ALL:
        LOAD MASTER_STATE,07
        CALL PUSH
        JUMP TWO_OPTION_MENU_RS
MASTER_PAUSED:
        LOAD MASTER_STATE,08
        CALL PUSH
        JUMP ONE_OPTION_MENU
MASTER_RECORDING:
        LOAD MASTER_STATE, 09
        CALL PUSH
        JUMP ONE_OPTION_MENU
```

```
;///////FILE_MENU CONTROL/////////////////////////////////
FILE_MENU_RS:
        LOAD FILE_STATE, 00
FILE_MENU:
        CALL WAIT_STATE
        CALL INPUT_LOADER
        COMPARE CONTROLREG, 01
        JUMP Z, UP_FILE
        COMPARE CONTROLREG, 02
        JUMP Z, DOWN_FILE
        COMPARE CONTROLREG, 04
        JUMP Z, BACK
        COMPARE CONTROLREG, 08
        JUMP Z, SELECT_FILE
        JUMP FILE_MENU
;////////////////////////FILE UP/DOWN/////////////////////////////////////
UP_FILE:
        COMPARE FILE_STATE, 07
        JUMP NZ, UP_FILE_GOOD           ;if file number isnt max
        JUMP FILE_MENU                  ;otherwise return
UP_FILE_GOOD:
        ADD FILE_STATE, 01
        CALL PUSH
```

```
        JUMP FILE_MENU


DOWN_FILE:
        COMPARE FILE_STATE, 00
        JUMP NZ, DOWN_FILE_GOOD         ;if file number isnt min
        JUMP FILE_MENU                  ;other wise return
DOWN_FILE_GOOD:
        SUB FILE_STATE, 01
        CALL PUSH
        JUMP FILE_MENU
;/////////////////SELECT_FILE////////////////////////
SELECT_FILE:
        COMPARE MASTER_STATE, 01
        JUMP Z, PLAY_MESSAGE
        COMPARE MASTER_STATE, 02
        JUMP Z, RECORD_MESSAGE
        COMPARE MASTER_STATE, 03
        JUMP Z, MASTER_DELETING
;/////////////////////SKIP//////////////////////////
SKIP:
        COMPARE FILE_STATE, 07
        JUMP Z,WRAP
        ADD FILE_STATE,01
        WRAP:
        LOAD FILE_STATE, 00
```

```
        JUMP PLAY_MESSAGE


;//////////////////////PLAY//////////////////////////
PLAY_MESSAGE:
        LOAD COMMAND_REG, 01
        CALL SEND_COMMAND_PRD
        JUMP MASTER_PLAYING
;//////////////////////RECORD//////////////////////////
RECORD_MESSAGE:
        LOAD COMMAND_REG, 03
        CALL SEND_COMMAND_PRD
        JUMP MASTER_RECORDING
;//////////////////////DELETE//////////////////////////
DELETE_MESSAGE:
        LOAD COMMAND_REG, 04
        CALL SEND_COMMAND_PRD
        JUMP BACK
;//////////////////////TWO OPTION MENU///for pause/skip, yes/no ,vol up/vol down
TWO_OPTION_MENU_RS:
        LOAD TWO_STATE,00
TWO_OPTION_MENU:
        CALL WAIT_STATE
        CALL INPUT_LOADER
        COMPARE CONTROLREG, 01
        JUMP Z, TWO_OPT_FLIP
```

```
        COMPARE CONTROLREG, 02
        JUMP Z, TWO_OPT_FLIP
        COMPARE CONTROLREG, 04
        JUMP Z, BACK
        COMPARE CONTROLREG, 08
        JUMP Z, SELECT_TWO_OPT
        JUMP TWO_OPTION_MENU
;//////////////////////TWO OPT UP/DOWN///////////////
TWO_OPT_FLIP:
        COMPARE TWO_STATE, 00
        JUMP Z, TWO_STATE_UP
        JUMP Z, TWO_STATE_DOWN
TWO_STATE_UP:
        LOAD TWO_STATE,01
        CALL PUSH
        JUMP TWO_OPTION_MENU
TWO_STATE_DOWN:
        LOAD TWO_STATE,00
        CALL PUSH
        JUMP TWO_OPTION_MENU
;//////////////////////TWO OPT SELECT/////////////////
SELECT_TWO_OPT:
        COMPARE MASTER_STATE,04 ; playing
        JUMP Z, PAUSE_SKIP
        COMPARE MASTER_STATE,05 ; deleting
```

```
        JUMP Z, YES_NO
        COMPARE MASTER_STATE,06 ; volume
        JUMP Z, UP_DOWN
        COMPARE MASTER_STATE,07 ; deleting all
        JUMP Z, YES_NO_ALL
PAUSE_SKIP:
        COMPARE TWO_STATE,00
        JUMP Z, PAUSE
        COMPARE TWO_STATE,01
        JUMP Z, SKIP
YES_NO:
        COMPARE TWO_STATE,00
        JUMP Z, BACK
        COMPARE TWO_STATE,01
        JUMP Z, DELETE_MESSAGE
YES_NO_ALL:
        COMPARE TWO_STATE,00
        JUMP Z, BACK
        COMPARE TWO_STATE,01
        JUMP Z, DELETE_ALL
UP_DOWN:
        COMPARE TWO_STATE,00
        JUMP Z, VOL_UP
        COMPARE TWO_STATE,01
        JUMP Z, VOL_DOWN
```

```
;//////////////////////VOL UP/DOWN////////////////////////////
VOL_UP:
        COMPARE VOLUME, 1F
        JUMP NZ, UP_VOL
        JUMP ONE_OPTION_MENU
VOL_DOWN:
        COMPARE VOLUME,00
        JUMP NZ, DOWN_VOL
        JUMP ONE_OPTION_MENU
UP_VOL:
        ADD VOLUME,01
        OUTPUT VOLUME, VOLUME_OUTPUT
        JUMP TWO_OPTION_MENU
        DOWN_VOL:
        SUB VOLUME,01
        OUTPUT VOLUME, VOLUME_OUTPUT
        JUMP TWO_OPTION_MENU
;//////////////////////PAUSE/////////////////////////////////////
PAUSE:
        LOAD COMMAND_REG, 02
        OUTPUT COMMAND_REG, COMMAND
        JUMP MASTER_PAUSED
;//////////////////////RESUME_MESSAGE//////////////////////////////
```

```
RESUME_MESSAGE:
        LOAD COMMAND_REG, 01
        OUTPUT COMMAND_REG, COMMAND
        JUMP MASTER_PLAYING
;////////////////////DELETE_ALL/////////////////////////
DELETE_ALL:
        LOAD COMMAND_REG, 05
        CALL SEND_COMMAND_PRD
        JUMP BACK
;///////////////////////ONE_OPTION////////////////////////
ONE_OPTION_MENU:
        CALL WAIT_STATE
        CALL INPUT_LOADER
        COMPARE CONTROLREG,08
        JUMP Z, ONE_OPT_SEL
        JUMP ONE_OPTION_MENU
ONE_OPT_SEL:
        COMPARE MASTER_STATE,08
        JUMP Z, RESUME_MESSAGE ;RESUME!!!
        COMPARE MASTER_STATE,09
        JUMP Z, STOP
        JUMP ONE_OPTION_MENU
;/////////////////STOP!!!!!//////////////
STOP:
        LOAD SEANS_CONSTANT,00
```

```
        OUTPUT SEANS_CONSTANT, COMMAND
        JUMP BACK
;////////////////////////BACK!!!!///////////////
BACK:           ;ONE BACK TO RULE THEM ALL!!!!
        JUMP MASTER_MENU
;//////////////////SEND_COMMAND  P/R/D  ///////////////////////////////////
SEND_COMMAND_PRD:
        OUTPUT FILE_STATE,FILE
        LOAD SEANS_CONSTANT,00
        OUTPUT SEANS_CONSTANT, COMMAND
        READY_WAIT:
        INPUT READY_REG,READY_SIG
        COMPARE READY_REG, 01
        JUMP NZ, READY_WAIT
        OUTPUT COMMAND_REG, COMMAND
;///////////WAIT STATE//////////////////////////
WAIT_STATE:
        INPUT CONTROLREG, CONTROL
        COMPARE CONTROLREG, 00        ;wait state
        JUMP NZ, WAIT_STATE
        RETURN
;////////INPUT LOADER//////////////
INPUT_LOADER:
        INPUT CONTROLREG, CONTROL ;ADD USB SHIT HERE!!!!!
        COMPARE CONTROLREG, 00
```

```
        JUMP Z, INPUT_LOADER
        RETURN
;//////////LCD_PUSH//////////////
PUSH:
        LOAD LCD_BUILD, MASTER_STATE
        SL0 LCD_BUILD
        SL0 LCD_BUILD
        SL0 LCD_BUILD
        SL0 LCD_BUILD
        COMPARE MASTER_STATE,00
        JUMP Z, MAIN_PUSH
        COMPARE MASTER_STATE,01
        JUMP Z, FILE_PUSH
        COMPARE MASTER_STATE,02
        JUMP Z, FILE_PUSH
        COMPARE MASTER_STATE,03
        JUMP Z, FILE_PUSH
        COMPARE MASTER_STATE,04
        JUMP Z, TWO_PUSH
        COMPARE MASTER_STATE,05
        JUMP Z, TWO_PUSH
        COMPARE MASTER_STATE,06
        JUMP Z, TWO_PUSH
        COMPARE MASTER_STATE,07
        JUMP Z, TWO_PUSH
```

```
        COMPARE MASTER_STATE,08
        JUMP Z, ONE_PUSH
        COMPARE MASTER_STATE,09
        JUMP Z, ONE_PUSH
MAIN_PUSH:
        ADD LCD_BUILD, MENU_STATE
        JUMP PUSH_PUSH
FILE_PUSH:
        ADD LCD_BUILD, FILE_STATE
        JUMP PUSH_PUSH
TWO_PUSH:
        ADD LCD_BUILD, TWO_STATE
        JUMP PUSH_PUSH
ONE_PUSH:
        ADD LCD_BUILD, 00
        JUMP PUSH_PUSH
PUSH_PUSH:
        OUTPUT LCD_BUILD, LCD_OUTPUT
        RETURN
        JUMP MAIN_MENU
```

```
ISR:
        LOAD LCD_BUILD, B0
        OUTPUT LCD_BUILD, LCD_OUTPUT
        LOAD COMMAND_REG, 00
        OUTPUT COMMAND_REG, COMMAND
        CALL WAIT_STATE
        CALL INPUT_LOADER
        RETURNI ENABLE

ADDRESS 3FF
JUMP ISR
```

// Picoblaze #2 – contains the characters for the various lcd messages. Receives input from picoblaze #1 and outputs data to lcd module (not lcd, yet).

```
CONSTANT ENTERCODE, 01

CONSTANT COMMAND,  04

CONSTANT EXITCODE, 02

CONSTANT READY_SIG, 03

NAMEREG s4, READY_REG

NAMEREG s0, ENTER_CODE

NAMEREG s1, EXIT_CODE

NAMEREG s2, MENU

NAMEREG s3, PLACE

NAMEREG s5, COMMAND_REG

NAMEREG s6, LAST_CODE

LOAD LAST_CODE, 00


BEGIN:

        INPUT ENTER_CODE, ENTERCODE

        COMPARE LAST_CODE, ENTER_CODE

        JUMP Z, BEGIN

        LOAD LAST_CODE, ENTER_CODE


        LOAD MENU, ENTER_CODE

        LOAD PLACE, ENTER_CODE

        AND PLACE,0F
```

```
SR0 MENU
SR0 MENU
SR0 MENU
SR0 MENU

COMPARE MENU, 00
JUMP Z, MAIN_MENU
COMPARE MENU, 01
JUMP Z, FILE_MENU
COMPARE MENU, 02
JUMP Z, FILE_MENU
COMPARE MENU, 03
JUMP Z, FILE_MENU
COMPARE MENU, 04
JUMP Z, PLAY_MENU
COMPARE MENU, 05
JUMP Z, DEL_MENU
COMPARE MENU, 06
JUMP Z, VOL_MENU
COMPARE MENU, 07
JUMP Z, DEL_MENU
COMPARE MENU, 08
JUMP Z, PAUSE_MENU
COMPARE MENU, 09
```

```
        JUMP Z, RECORD_MENU
        COMPARE MENU,0A
        JUMP Z, WELCOME
        COMPARE MENU,0B
        JUMP Z, MEM_FULL
MAIN_MENU:
        COMPARE PLACE, 00
        CALL Z, MAIN1
        JUMP Z, S_1
        COMPARE PLACE, 01
        CALL Z, MAIN1
        JUMP Z, S_2
        COMPARE PLACE, 02
        CALL Z, MAIN1
        JUMP Z, S_3
        COMPARE PLACE, 03
        CALL Z, MAIN2
        JUMP Z, S_1
        COMPARE PLACE, 04
        CALL Z, MAIN2
        JUMP Z, S_3
FILE_MENU:
        COMPARE PLACE, 00
        CALL Z, FILE1_4
        JUMP Z, S_1
```

```
        COMPARE PLACE, 01
        CALL Z, FILE1_4
        JUMP Z, S_3
        COMPARE PLACE, 02
        CALL Z, FILE2_4
        JUMP Z,S_1
        COMPARE PLACE, 03
        CALL Z, FILE2_4
        JUMP Z,S_3
        COMPARE PLACE, 04
        CALL Z,FILE3_4
        JUMP Z,S_1
        COMPARE PLACE, 05
        CALL Z, FILE3_4
        JUMP Z,S_3
        COMPARE PLACE, 06
        CALL Z, FILE4_4
        JUMP Z,S_1
        COMPARE PLACE, 07
        CALL Z, FILE4_4
        JUMP Z,S_3
PLAY_MENU:
        CALL PAUSE_SKIP
        COMPARE PLACE, 00
        JUMP Z, S_1             ;PAUSE
```

```
        JUMP NZ, S_3            ;SKIP
DEL_MENU:
        CALL CAN_CON
        COMPARE PLACE,00
        JUMP Z, S_1             ;NO
        JUMP NZ, S_3            ;YES
VOL_MENU:
        CALL UP_DOWN
        COMPARE PLACE, 00
        JUMP Z, S_1;up
        JUMP NZ, S_3;DOWN
PAUSE_MENU:
        CALL RESUME
        JUMP BEGIN
RECORD_MENU:
        CALL STOP
        JUMP BEGIN
MAIN1:
        CALL UPLINE
        CALL PRINT_MAIN_LEFT
        CALL DROPLINE
        RETURN
MAIN2:
        CALL  UPLINE
        CALL PRINT_MAIN_RIGHT
```

```
        CALL DROPLINE
        RETURN
FILE1_4:
        CALL UPLINE
        CALL PRINTFILE1_2
        CALL DROPLINE
        RETURN
FILE2_4:
        CALL UPLINE
        CALL PRINTFILE3_4
        CALL DROPLINE
        RETURN
FILE3_4:
        CALL UPLINE
        CALL PRINTFILE5_6
        CALL DROPLINE
        RETURN
FILE4_4:
        CALL UPLINE
        CALL PRINTFILE7_8
        CALL DROPLINE
        RETURN
PAUSE_SKIP:
        CALL UPLINE
        CALL PRINT_PAUSE_SKIP
```

```
        CALL DROPLINE
        RETURN
RESUME:
        CALL UPLINE
        CALL PRINT_RESUME
        CALL DROPLINE
        RETURN
STOP:
        CALL UPLINE
        CALL PRINT_STOP
        RETURN

UP_DOWN:
        CALL UPLINE
        CALL PRINT_UP_DOWN
        RETURN
CAN_CON:
        CALL UPLINE
        CALL PRINT_CAN_CON
        RETURN
WELCOME:
        CALL UPLINE
        CALL PRINT_WELCOME
        RETURN
MEM_FULL:
```

```
        CALL UPLINE
        CALL PRINT_MEM_FULL
        RETURN
S_1:
        LOAD EXIT_CODE,3E
        CALL PRINTFILE
        LOAD EXIT_CODE,7C
        CALL PRINTFILE
        LOAD EXIT_CODE,7C
        CALL PRINTFILE
        LOAD EXIT_CODE,3C
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
```

```
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        JUMP BEGIN
S_2:
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE

        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,3E
        CALL PRINTFILE
        LOAD EXIT_CODE,7C
```

```
        CALL PRINTFILE
        LOAD EXIT_CODE,7C
        CALL PRINTFILE
        LOAD EXIT_CODE,3C
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        JUMP BEGIN
S_3:
        LOAD EXIT_CODE,20
        CALL PRINTFILE
```

```
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE

LOAD EXIT_CODE,3E
CALL PRINTFILE
LOAD EXIT_CODE,7C
CALL PRINTFILE
LOAD EXIT_CODE,7C
CALL PRINTFILE
LOAD EXIT_CODE,3C
CALL PRINTFILE
```

```
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        JUMP BEGIN
DROPLINE:
        INPUT READY_REG, READY_SIG
        COMPARE READY_REG, 01
        JUMP NZ, DROPLINE
        LOAD COMMAND_REG,01
        OUTPUT COMMAND_REG, COMMAND
        RETURN
UPLINE:
        INPUT READY_REG, READY_SIG
        COMPARE READY_REG, 01
        JUMP NZ, UPLINE
        LOAD COMMAND_REG,00
        OUTPUT COMMAND_REG, COMMAND
        RETURN
PRINT_MAIN_LEFT:
        LOAD EXIT_CODE,50
        CALL PRINTFILE
```

```
LOAD EXIT_CODE,4c

CALL PRINTFILE

LOAD EXIT_CODE,41

CALL PRINTFILE

LOAD EXIT_CODE,59

CALL PRINTFILE

LOAD EXIT_CODE,20

CALL PRINTFILE

LOAD EXIT_CODE,52

CALL PRINTFILE

LOAD EXIT_CODE,45

CALL PRINTFILE

LOAD EXIT_CODE,43

CALL PRINTFILE

LOAD EXIT_CODE,20

CALL PRINTFILE

LOAD EXIT_CODE,44

CALL PRINTFILE

LOAD EXIT_CODE,45

CALL PRINTFILE

LOAD EXIT_CODE,4c

CALL PRINTFILE

LOAD EXIT_CODE,20

CALL PRINTFILE

LOAD EXIT_CODE,2d
```

```
        CALL PRINTFILE
        LOAD EXIT_CODE,3e
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        RETURN
PRINT_MAIN_RIGHT:
        LOAD EXIT_CODE,3c
        CALL PRINTFILE
        LOAD EXIT_CODE,2d
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,44
        CALL PRINTFILE
        LOAD EXIT_CODE,45
        CALL PRINTFILE
        LOAD EXIT_CODE,4c
        CALL PRINTFILE
        LOAD EXIT_CODE,4c
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,56
        CALL PRINTFILE
```

```
        LOAD EXIT_CODE,4f
        CALL PRINTFILE
        LOAD EXIT_CODE,4c
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,3e
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        RETURN
PRINTFILE1_2:
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,46
        CALL PRINTFILE
        LOAD EXIT_CODE,49
        CALL PRINTFILE
        LOAD EXIT_CODE,4c
```

```
        CALL PRINTFILE
        LOAD EXIT_CODE,45
        CALL PRINTFILE
        LOAD EXIT_CODE,31
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,46
        CALL PRINTFILE
        LOAD EXIT_CODE,49
        CALL PRINTFILE
        LOAD EXIT_CODE,4c
        CALL PRINTFILE
        LOAD EXIT_CODE,45
        CALL PRINTFILE
        LOAD EXIT_CODE,32
        CALL PRINTFILE
        LOAD EXIT_CODE,2d
        CALL PRINTFILE
        LOAD EXIT_CODE,3e
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        RETURN
PRINTFILE3_4:
```

```
LOAD EXIT_CODE,3c

CALL PRINTFILE

LOAD EXIT_CODE,2d

CALL PRINTFILE

LOAD EXIT_CODE,46

CALL PRINTFILE

LOAD EXIT_CODE,49

CALL PRINTFILE

LOAD EXIT_CODE,4c

CALL PRINTFILE

LOAD EXIT_CODE,45

CALL PRINTFILE

LOAD EXIT_CODE,33

CALL PRINTFILE

LOAD EXIT_CODE,20

CALL PRINTFILE

LOAD EXIT_CODE,46

CALL PRINTFILE

LOAD EXIT_CODE,49

CALL PRINTFILE

LOAD EXIT_CODE,4c

CALL PRINTFILE

LOAD EXIT_CODE,45

CALL PRINTFILE

LOAD EXIT_CODE,34
```

```
        CALL PRINTFILE
        LOAD EXIT_CODE,2d
        CALL PRINTFILE
        LOAD EXIT_CODE,3e
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        RETURN
PRINTFILE5_6:
        LOAD EXIT_CODE,3c
        CALL PRINTFILE
        LOAD EXIT_CODE,2d
        CALL PRINTFILE
        LOAD EXIT_CODE,46
        CALL PRINTFILE
        LOAD EXIT_CODE,49
        CALL PRINTFILE
        LOAD EXIT_CODE,4c
        CALL PRINTFILE
        LOAD EXIT_CODE,45
        CALL PRINTFILE
        LOAD EXIT_CODE,35
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
```

```
        LOAD EXIT_CODE,46
        CALL PRINTFILE
        LOAD EXIT_CODE,49
        CALL PRINTFILE
        LOAD EXIT_CODE,4c
        CALL PRINTFILE
        LOAD EXIT_CODE,45
        CALL PRINTFILE
        LOAD EXIT_CODE,36
        CALL PRINTFILE
        LOAD EXIT_CODE,2d
        CALL PRINTFILE
        LOAD EXIT_CODE,3e
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        RETURN
PRINTFILE7_8:
        LOAD EXIT_CODE,3c
        CALL PRINTFILE
        LOAD EXIT_CODE,2d
        CALL PRINTFILE
        LOAD EXIT_CODE,46
        CALL PRINTFILE
        LOAD EXIT_CODE,49
```

```
CALL PRINTFILE
LOAD EXIT_CODE,4c
CALL PRINTFILE
LOAD EXIT_CODE,45
CALL PRINTFILE
LOAD EXIT_CODE,37
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,46
CALL PRINTFILE
LOAD EXIT_CODE,49
CALL PRINTFILE
LOAD EXIT_CODE,4c
CALL PRINTFILE
LOAD EXIT_CODE,45
CALL PRINTFILE
LOAD EXIT_CODE,38
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,30
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
```

```
        RETURN



PRINT_UP_DOWN:
        LOAD EXIT_CODE,56
        CALL PRINTFILE
        LOAD EXIT_CODE,4f
        CALL PRINTFILE
        LOAD EXIT_CODE,4c
        CALL PRINTFILE
        LOAD EXIT_CODE,5f
        CALL PRINTFILE
        LOAD EXIT_CODE,55
        CALL PRINTFILE
        LOAD EXIT_CODE,50
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,56
        CALL PRINTFILE
        LOAD EXIT_CODE,4c
```

```
        CALL PRINTFILE
        LOAD EXIT_CODE,5f
        CALL PRINTFILE
        LOAD EXIT_CODE,44
        CALL PRINTFILE
        LOAD EXIT_CODE,4f
        CALL PRINTFILE
        LOAD EXIT_CODE,57
        CALL PRINTFILE
        LOAD EXIT_CODE,4e
        CALL PRINTFILE
        RETURN
PRINT_RESUME:
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
```

```
        LOAD EXIT_CODE,52
        CALL PRINTFILE
        LOAD EXIT_CODE,45
        CALL PRINTFILE
        LOAD EXIT_CODE,53
        CALL PRINTFILE
        LOAD EXIT_CODE,55
        CALL PRINTFILE
        LOAD EXIT_CODE,4d
        CALL PRINTFILE
        LOAD EXIT_CODE,45
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        RETURN
PRINT_CAN_CON:
        LOAD EXIT_CODE,43
        CALL PRINTFILE
        LOAD EXIT_CODE,41
```

```
CALL PRINTFILE
LOAD EXIT_CODE,4e
CALL PRINTFILE
LOAD EXIT_CODE,43
CALL PRINTFILE
LOAD EXIT_CODE,45
CALL PRINTFILE
LOAD EXIT_CODE,4c
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,43
CALL PRINTFILE
LOAD EXIT_CODE,4f
CALL PRINTFILE
LOAD EXIT_CODE,4e
CALL PRINTFILE
LOAD EXIT_CODE,46
CALL PRINTFILE
LOAD EXIT_CODE,49
CALL PRINTFILE
```

```
        LOAD EXIT_CODE,52
        CALL PRINTFILE
        LOAD EXIT_CODE,4d
        CALL PRINTFILE
        RETURN
PRINT_PAUSE_SKIP:
        LOAD EXIT_CODE,50
        CALL PRINTFILE
        LOAD EXIT_CODE,41
        CALL PRINTFILE
        LOAD EXIT_CODE,55
        CALL PRINTFILE
        LOAD EXIT_CODE,53
        CALL PRINTFILE
        LOAD EXIT_CODE,45
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
```

```
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,53
        CALL PRINTFILE
        LOAD EXIT_CODE,4B
        CALL PRINTFILE
        LOAD EXIT_CODE,49
        CALL PRINTFILE
        LOAD EXIT_CODE,50
        CALL PRINTFILE
        RETURN
PRINT_STOP:
        LOAD EXIT_CODE,53
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,54
        CALL PRINTFILE
```

```
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,4f
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,50
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        RETURN
PRINT_WELCOME:
        LOAD EXIT_CODE,20
```

```
CALL PRINTFILE
LOAD EXIT_CODE,54
CALL PRINTFILE
LOAD EXIT_CODE,48
CALL PRINTFILE
LOAD EXIT_CODE,49
CALL PRINTFILE
LOAD EXIT_CODE,53
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,57
CALL PRINTFILE
LOAD EXIT_CODE,41
CALL PRINTFILE
LOAD EXIT_CODE,53
CALL PRINTFILE
LOAD EXIT_CODE,20
CALL PRINTFILE
LOAD EXIT_CODE,45
CALL PRINTFILE
LOAD EXIT_CODE,41
CALL PRINTFILE
LOAD EXIT_CODE,53
CALL PRINTFILE
```

```
        LOAD EXIT_CODE,59
        CALL PRINTFILE
        LOAD EXIT_CODE,21
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        RETURN
PRINT_MEM_FULL:
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,4d
        CALL PRINTFILE
        LOAD EXIT_CODE,45
        CALL PRINTFILE
        LOAD EXIT_CODE,4d
        CALL PRINTFILE
        LOAD EXIT_CODE,4f
        CALL PRINTFILE
        LOAD EXIT_CODE,52
        CALL PRINTFILE
        LOAD EXIT_CODE,59
        CALL PRINTFILE
        LOAD EXIT_CODE,20
```

```
        CALL PRINTFILE
        LOAD EXIT_CODE,46
        CALL PRINTFILE
        LOAD EXIT_CODE,55
        CALL PRINTFILE
        LOAD EXIT_CODE,4c
        CALL PRINTFILE
        LOAD EXIT_CODE,4c
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        LOAD EXIT_CODE,20
        CALL PRINTFILE
        RETURN
PRINTFILE:
        INPUT READY_REG, READY_SIG
        COMPARE READY_REG, 01
        JUMP NZ, PRINTFILE
        OUTPUT EXIT_CODE,EXITCODE
        RETURN
```

Discussion and Conclusions

In this section, discuss how you arrived at your major decisions (why you put X in hardware and Y in software, how you selected the memory chips you used, etc.). Discuss any aspects of your design that you would do differently if you had it all to do over again, and why. Include any and all thoughts you have on the project, how it could have been done differently or better, etc.

If we could start over again, we probably would have divided the workflow a little bit differently. The lcd module proved to be the most difficult thing to implement for us, and instead of trying to adapt an 8 bit example code, we should have implemented our own from the beginning. The biggest thing to take from this project is to learn the data sheet of a component thoroughly before trying to use it. Almost always, we were able to implement the components quickly, but not at 100%. Resolving issues only came after a thorough understanding of the system and what was expected. As it takes a few minutes to compile and run the Verilog on the FPGA, it is worth taking the time to understand what is happening on a very detailed level instead of just trying to change things and make it work.

The design of the project initially began with using one module for each component and a top level wrapper module. The ac97 module was the easiest to create, but typos kept us busy for quite a while. Integrating the memory and testing it was very straight forward, but difficult to understand. The LCD was the biggest issue because of how the example implemented the setup and write states. Eventually, we rewrote the entire lcd driver in Verilog, and fed it ascii values via a picoblaze, which was in turn was fed by another picoblaze which took the button inputs and ran them through a state machine to determine the output values and the lcd message to display.

Overall, this project was difficult because it was very hard to work together on many parts. A majority of our project was spent with one person coding while the other two were doing side tasks or trying to help troubleshoot stuff. In the future, I'd like to learn how to better debug signals, especially for the on board components, such as the lcd display and the ac97 IO stream.