Pi Memory Game

Final Project Report
December 8, 2006
E155

Chris Acon and Andy Chin

Abstract

This memory game helps people learn the digits of pi in a challenging and addictive manner. After beginning the game, the user inputs which digit to start from. The game then displays the numbers of pi sequentially up to that starting digit. The user presses the sequence through a keypad, which transfers the input to an FPGA and eventually to a microcontroller. The microcontroller compares those input values to a stored array containing the digits of pi. Upon successfully inputting the correct progression, the game displays an additional digit to memorize. If the user is wrong at any time, the game ends, and an LCD displays the missed digit, user score, and high score.

Intro

We wanted to create a challenging and addictive game that incorporates various aspects of the Harrisboard microprocessor board that we had learned about throughout the course, as well as incorporate new components we found useful and interesting. We also needed to compromise between complexity and feasibility.

We decided to create a memory game based on the number pi, where the user attempts to memorize the digits of pi. This incorporated the use of the keypad and interface between the FPGA and the PIC. Important game information is displayed to the user using LEDs controlled from the FPGA. We also decided to use an LCD to display more information, which would be controlled directly from the PIC.

In this way, we were able to use our existing knowledge of microprocessors such as proper keypad polling, timing, and FPGA to PIC communication, as well as learn new information such as how to interface with an LCD.

A block diagram of our overall system is shown in Figure 1.

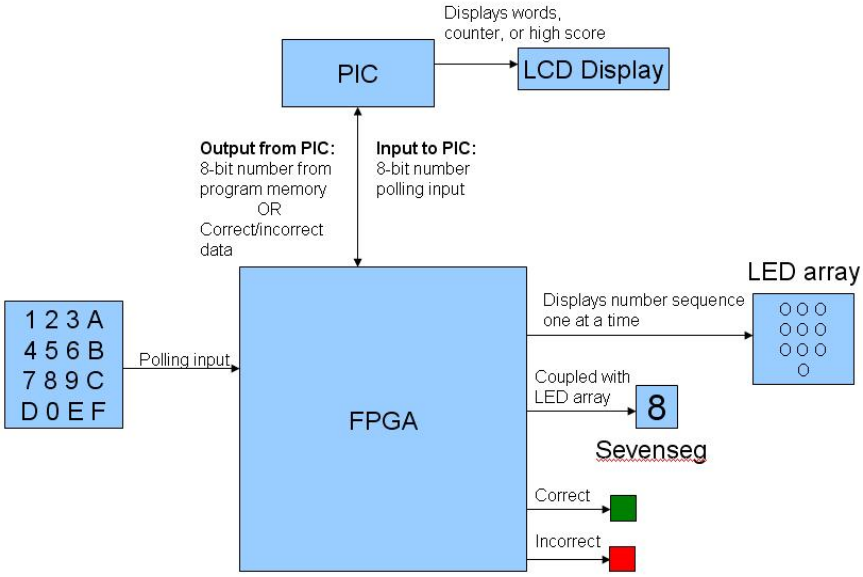


Figure 1. Overall System

The first one-hundred digits of Pi are stored on the PIC. Game queries “Start?” and “Starting Digit?” are displayed on the LCD controlled by the PIC. The FPGA polls the keypad for a user input. The polled data is sent to the PIC through the Serial Port Interface. The main game is programmed into the PIC. A number of digits of Pi are sent to the FPGA through the SPI, starting from the first digit of pi and continuing until the user specified starting digit. The FPGA displays each digit on the 7-Segment LED display as well as on the LED array where each number is represented by its corresponding position on the keypad. The user is then expected to input the displayed sequence in order. As each number is entered, the FPGA sends the data to the PIC. The PIC displays the numbers on the LCD as they are entered. The PIC also checks if the

input is correct or incorrect, and sends the signal to the FPGA, which then controls the red or green LEDs. After all digits are entered correctly, the display sequence begins again, incrementing the pi sequence by one so that the user is required to memorize an additional digit each round of play. When the user fails, the red LED is lit, and the LCD is used to display the end game sequence: "Gameover", "Score" and "HiScore". The user's score is based on the number of digits successfully memorized. The highest obtained score is stored on the PIC during its operation and displayed after each game attempt. The game then restarts.

New Hardware

For our project, we used the Crystalfontz character LCD model CFAH0802A-YYH-JP. The model number classifies the LCD. "CFA" is the brand name, Crystalfontz America, Inc. "0802" is the number of characters, in this case 8 characters x 2 lines, for a total of 16 characters possible, which we felt was sufficient for our purposes. "A-YYH-JP" refers to more specific information such as the color of the display, the polarization and temperature range, and available fonts.

Our particular LCD has 14 pins described in Figure 2.

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

Figure 2. LCD Pin Interface

Pins 1 and 2 power the LCD. Pin 3 controls the LCD contrast. For our LCD, a lower voltage corresponded to a higher contrast. Pin 4 is the Register Select (RS), and controls between selection of two register used by the LCD, the instruction register (IR) and the data register (DR). The IR is written when the LCD needs to perform certain functions such as clearing the display, shifting the cursor, toggling display options, and specifying address information. The DR writes to the display data RAM (DDRAM) or character generator RAM (CGRAM). DDRAM is used to select characters for display, while CGRAM is used to create custom characters. Pin 5 selects between Read and Write (R/W) functionality. Pin 6 controls the Enable signal (E) which is used when sending

instructions. Pins 7 through 14 are the eight data bits used for LCD instructions. Most LCDs use a very similar interface and a very similar instructions set.

The possible instructions are detailed in Figure 3.

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

* " - " : don't care

Figure 3. LCD Instruction Set

- *Clear Display* clears the LCD screen of characters.
- *Return Home* returns the cursor to the initial position at the top-left corner of the screen.
- *Entry Mode Set* toggles the automatic shift function, where the cursor either increments or decrements (I/D) after each character is displayed, (1) for increment and (0) for decrement. The SH bit is normally (0), but if set to (1), the entire display will shift after each character display, like a teleprompter.
- *Display ON/OFF Control* toggles the display on and off (D), the cursor display (C), and the cursor blinking (B). The cursor appears as a line under the cursors position, and the cursor blinking appears as a blinking black box.
- *Cursor or Display Shift* manually moves the cursor or entire display around, much like *Entry Mode Set* except a character does not need to be written. The (S/C) bit toggles between cursor shift or entire display shift, (1) for display shift and (0) for cursor shift. The (R/L) bit controls direction, (1) for right and (0) for left.
- *Function Set* is used specifically for the LCD initialization sequence, and varies between different LCDs.
- *Set CGRAM Address* is used to program up to 8 custom characters. This function was not necessary for our purposes.

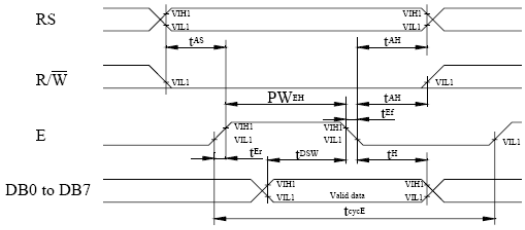
- *Set DDRAM Address* sets the cursor to a specified position. For our 8x2 display, the corresponding addresses in hexadecimal are shown in Figure 4.

1	2	3	4	5	6	7	8
00	01	02	03	04	05	06	07
40	41	42	43	44	45	46	47

Figure 4. DDRAM Addresses on 8x2 LCD Display

- *Read Busy Flag and Address* is used to read the busy flag (BF). If BF is (1), the LCD is busy in operation and cannot accept new instructions. The BF bit can be read until cleared before each instruction. In practice it is often easier to simply wait a minimum amount of time between instructions such that they are completed before the next is sent. The contents of the current address can also be read using this instruction.
- *Write Data to RAM* is used for character display. The 8-bit code sent to the data bits corresponds to predefined characters. The character codes can differ between LCDs.
- *Read Data from RAM* is used to read the contents of the internal RAM at the current address.

The Enable signal (E) is necessary for sending each instruction to the LCD. The (RS) and (R/W) bits must be set first, and then the (E) bit is set high. While (E) is high, the data bits DB0 to DB7 are set. The 8-bits are then written or read simultaneously when (E) is sent low. The timing diagram of a write operation is shown in Figure 6.



Ta=25℃, VDD=5.0± 0.5V

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	400	-	-	ns
Enable pulse width (high level)	PW_{EH}	150	-	-	ns
Enable rise/fall time	t_{Er},t_{Ef}	-	-	25	ns
Address set-up time (RS, R/W to E)	t_{AS}	30	-	-	ns
Address hold time	t_{AH}	10	-	-	ns
Data set-up time	t_{DSW}	40	-	-	ns
Data hold time	t_{H}	10	-	-	ns

Figure 5. Write Operation Timing

Exact timing values vary differ between LCDs.

When the LCD is first powered on, a specific initialization sequence must be processed to enable the LCD. The *Function Set* instruction must be sent three times after specific minimum time intervals: 15 ms after power on, 4.1 ms, then again after 100 µs. *Function Set* is then sent a fourth time, now with user specified values, then the display must turned off and cleared, and finally Entry Mode Set ends the initialization sequence. After initializing, instructions can be sent normally.

Breadboard Schematics

The pi memory game uses the Harrisboard's FPGA and PIC microcontroller to operate the LCD screen, hex display, correct/incorrect LEDs, LED array, and the keypad. These objects are arranged to make it easy for the user to see each function of the pi memory game, as shown in Figure 8:

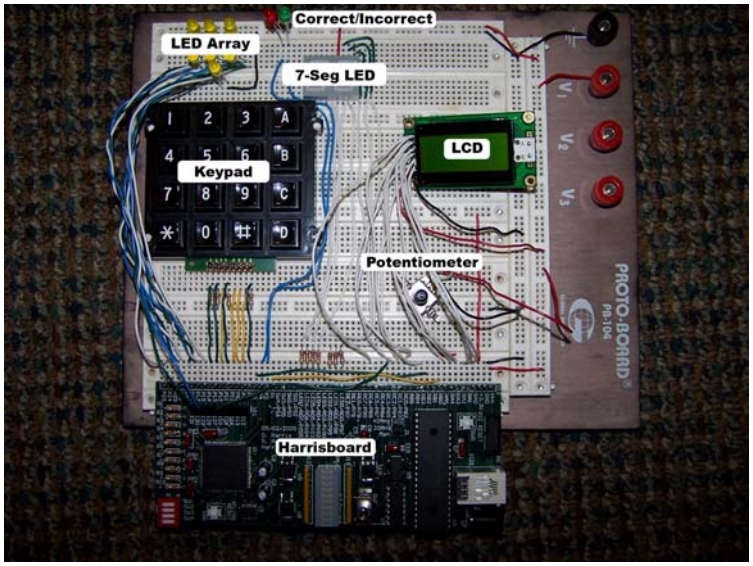


Figure 7. Physical Setup of Breadboard

The FPGA controls the keypad, LED array, correct/incorrect LEDs, and the hex display. The PIC controls the LCD through 10 different pins: RS, R/W, and DB0 through DB7. The potentiometer varies the contrast on the LCD, with 0 V for maximum contrast and 4.2 V for the minimum.

The PIC communicates to the FPGA through the serial port interface: SCK/RC3, SDO/RC4, and SDI/RC5. Since the PIC is always in master mode, a successful write to the SSPBUF register will send and receive data simultaneously through an inaccessible register. That register is then transferred to SSPBUF so that the data can be accessed. The SDI pin shifts in data to the PIC and the SDO pin shifts data out to the FPGA after each write to SSPBUF.

Detailed pin assignments are shown in Figure 9. Because so many pins are used by the FPGA and the PIC/FPGA interface, the PIC outputs to the LCD screen through selected bits of multiple registers. The PIC also has an input from pin 1 of the FPGA, the Ready

signal, to indicate that polling is complete, and an 8-bit number is ready to be shifted into the PIC. The PIC also outputs a Hold signal to pin 5 to prevent the FPGA from polling and accidentally triggering logic in the PIC. Since SCK needs to go to a global pin, it is routed to the pin 127 on the FPGA.

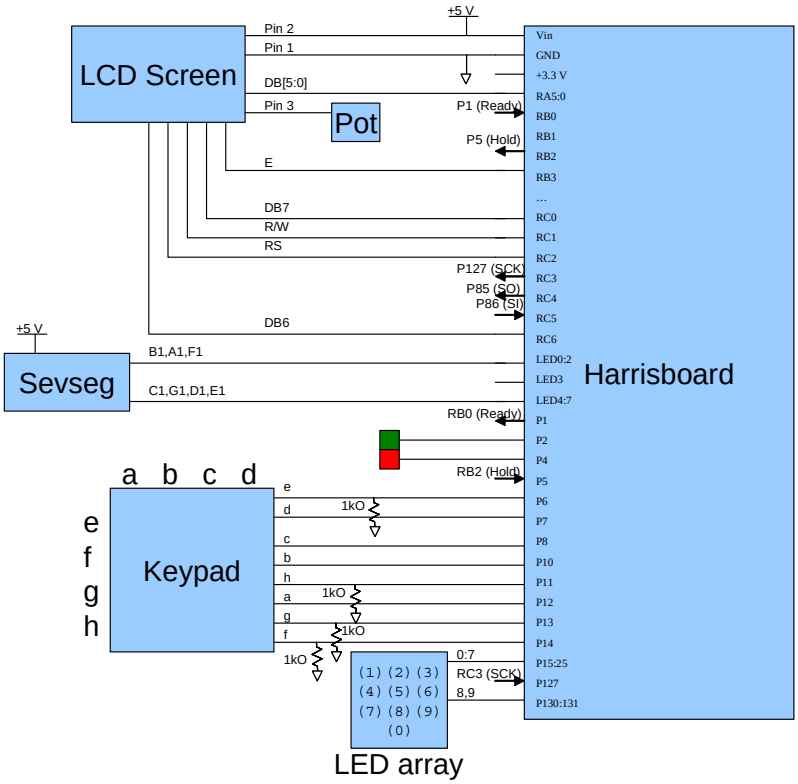


Figure 8. Pin Assignments for the Breadboard Layout

The FPGA outputs to hex display through seven pins, the LED array through ten pins, the correct/incorrect LEDs through two pins, and the polling columns of the keypad through four pins. Additionally, the FPGA has four input pins corresponding to the four rows of the keypad and the SCK input to pin 127.

Microcontroller Design

Our PIC was programmed using C, and can be viewed in Appendix B. The major software modules are:

- *main* – the main game sequence
- *isr* – the timing interrupt sequence
- *begingame* – the new game sequence which asks for start confirmation and starting digit
- *engame* – the endgame sequence displaying gameover dialogue, player score, and high score
- *poll* – waits to receive data from the FPGA
- *display* – sends a number to the FPGA for LED displays
- *LCDinitialize* – performs the LCD initialization sequence
- *LCDinstruct* – sends 10-bit instruction for LCD to corresponding PIC outputs

The main game sequence begins with *LCDinitialize* to enable the LCD screen. Then the *begingame* sequence is instantiated. The PIC outputs instructions to the LCD for initialization, and then to display specific information. The LCD first displays:

```
START?
(*)
```

The user is expected to enter (*) to begin the game, encoded as "E" in hexadecimal. The PIC awaits input from the FPGA in the *poll* module. After an input, the PIC checks SSPBUF to see if it equals "E", then continues the game. The LCD next displays:

```
START
DIGIT?
```

The user is expected to enter a number corresponding to a desired starting digit of Pi. The user can enter any number of digits, but inputs higher than 100 will be ignored, as there are only 100 digits of Pi saved in the PIC memory. When the first digit is entered, the LCD display changes to:

```
> 10
(* ends)
```

The "10" is an example number, but will be whatever the user inputs. In this case, the user will begin by attempting to memorize the first 10 digits of Pi. The user enters (*) to confirm the entry.

The Pi display sequence then commences. The LCD is not used in this sequence. The PIC outputs the digits of Pi to the FPGA starting from the first digit, 3, up to the user specified starting digit. Using timers and interrupts, each digit is displayed about 0.2 seconds except the last digit which is displayed for approximately 1 second. This allows the player to easily identify which digit was added since the previous sequence.

The game then begins the entry sequence, where the user inputs the digits of Pi previously displayed. The PIC polls until an input is received from the FPGA. This number is then checked against the correct value of Pi. A signal is output to the FPGA signaling whether the user was correct or incorrect. If correct, the number is displayed on the LCD. As each correct digit is input, it is displayed on the LCD in sequence, for example, on the user's fifth correct input, the LCD displays:

```
31415
```

The second line of the LCD is not used here. Since the LCD can only display eight characters on a line, the LCD is cleared upon entry of the ninth correct digit, and display begins again on the left starting position. The user continues until the sequence is completed, then the Pi sequence is incremented by one and the Pi display sequence begins again. The user's score is maintained to be equal to the number of digits of Pi

successfully memorized. The score increases by one after each round of play, since only one digit is added to the memorized sequence each round.

When the user fails, the incorrect signal is output to the FPGA. The ending game sequence begins. First the player is notified of failure:

```
GAMEOVER
Missed: 3
```

The "3" represents the digit the user should have entered to be correct. This is displayed for about 3 seconds, then the score is displayed:

```
Score:
08
```

The "08" is whatever the players score was, always displaying in two digits. The tens digit of the score is first extracted:

```
D = (int) score/10;
```

The score is divided by 10, and cast as an (int), then displayed in the tens position. This value is then used to extract the ones digit of the score:

```
D = score - D*10;
```

Subtracting ten times the tens digit leaves behind the ones digit for display. The user's score is displayed for about 3 seconds, then the high score is displayed:

```
HiScore:
45
```

The high score is decoded using the same algorithm, and displayed for 3 seconds before the game restarts, and the user's score is reset. The high score is also reset when the program is first run.

Since many I/O ports are already occupied by the FPGA, the 10 pins needed to control the LCD were not adjacent. Some of PORTC and PORTA are used to output the instruction signal to the LCD. The LCD instructions as they correspond to the PINS on the PIC are shown in the table below.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
RC2	RC1	RC0	RC6	RA5	RA4	RA3	RA2	RA1	RA0

Port C is configured as:

```
PORTC = ((instruct&0x380)/128 + (instruct&0x40)) | (0b10111000 &
            PORTC);
```

Here, “instruct” contains the 10-bit instruction. The last AND statement preserves the values of PORTC that do not need to be changed. Similarly, PORTA is configured as:

```
PORTA = (instruct&0x3F) | (0b11000000 & PORTA);
```

This procedure is contained in the *LCDinstruct* module.

FPGA Design

The hardware on the FPGA consists of four modules: freqdiv, polling, bitshift, and LEDarray, with the RTL schematic shown in Figure 10:

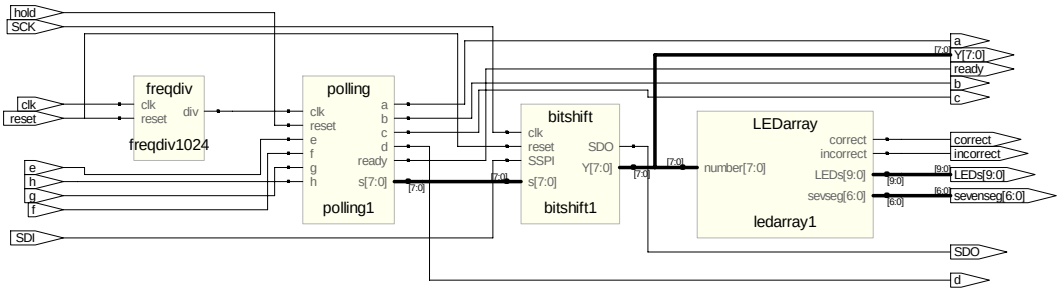


Figure 9: RTL schematic of FPGA design

Freqdiv is a 16-bit counter that divides the external clock to a reasonable polling frequency, around 1 kHz. It increments every positive edge of the 20 MHz input clock, and when bit 16 is high, freqdiv outputs high. Otherwise, it outputs low. Once bit 16 is 1, the next cycle will reset the counter to 0x0000. Since the duty cycle is irrelevant to the output frequency, this simple method was chosen over having a 50% duty cycle.

Bitshift contains two 8-bit shift registers, *ss* and *Y*, which send and receive data from the PIC, respectively. The clock for the shift registers is the SCK output from the PIC, which is routed to a global clock on the FPGA. When the PIC writes to the SSPBUF register on the PIC, it begins to shift data to and from the FPGA. The PIC produces 8 cycles of SCK, with each cycle shifting in data through SDI or shifting out data through SDO. On the FPGA, the MSB of *ss* corresponds to the SDI pin on the PIC and changes on every positive clock edge. The LSB of *Y* accepts data from the SDO pin and changes on every negative clock edge. This edge triggering achieves a synchronous interface between the PIC and the FPGA. On the first positive edge of SCK, output register *ss* becomes the same as the 8-bit character from the polling register *s*. Input register *Y* is sent to the LEDarray module for decoding and displaying.

LEDarray takes an 8-bit parallel input, designated *number*, from the *ss* register of the bitshift module and controls the LED array, correct and incorrect LEDs, and the hex display with a decoder. *Number[7:6]*, the two MSB, are unused. *Number[5]* outputs to the correct LED, *number[4]* outputs to the incorrect LED, and *number[3:0]* determines the seven output pins to the hex display and the 10 pins to the LED array. The hexadecimal value of number[3:0] lights up the corresponding LED in the array and forms the correct value on the hex display.

The most complicated portion of the FPGA is in the polling module. It takes in the divided clock from freqdiv, a reset, and four inputs from the keypad to produce an 8 bit output that is sent to bitshift. A simplified FSM is shown in Figure 11:

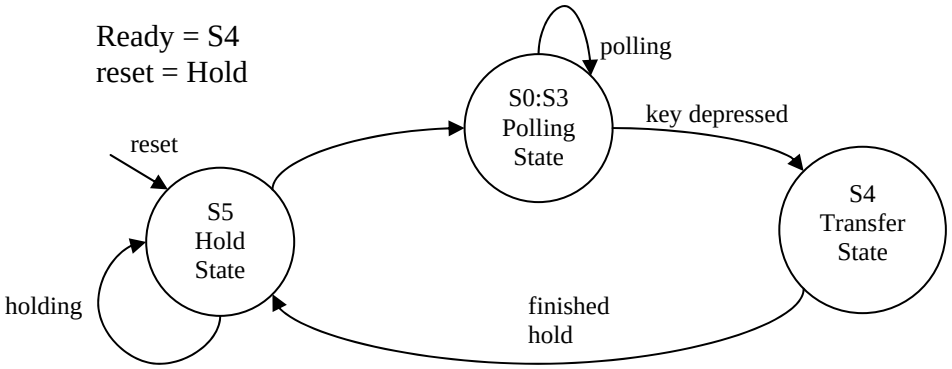


Figure 10: FSM for the polling module

The divided clock of approximately 1 kHz is suitable for human reaction times. The external clock of 20 MHz would cycle too fast through the polling, transfer, and hold states.

If the reset is on, the FSM remains at the Hold State, meaning it cannot transfer any data to the PIC. The reset input is the Hold output from the PIC and makes sure that the FPGA does not poll at inappropriate times.

S0:S3 are the Polling States in which only one column of the keypad is high. If the any of the keys are pressed, the corresponding row and column both go high at the same time, storing a decoded, 8-bit value into register *s*. That 8-bit character is sent to the module bitshift during the first positive edge of SCK. The FSM then goes into the Transfer State, in which the PIC completes the transfer and does some logic to determine if the user is correct or incorrect. Upon the next clock cycle, the FSM goes into the Hold State, which prevents any double tapping of keys that could trigger the PIC before it is ready again. Because the clock frequency of the polling module is much slower than the clock frequency of the PIC, the PIC can complete all necessary instructions before reaching the Hold State.

Results

The pi memory game works almost identically to the proposed game as detailed in the design proposal. The game starts by asking the user if they want to start and what digit to start on. The LED array and hex display light up a sequence of numbers that must be pushed in the right order to advance to the next round. Upon successful input of these numbers, the LED array and the number of displayed digits is increased by one and the user inputs an extra digit. If the user is incorrect, the LCD displays the missed number, the user score, and the high score. The game then restarts at the start screen.

There are two differences to the proposed game and the actual game. First, the actual game uses an LCD screen instead of several alphanumeric displays. This choice greatly reduces the number of wires on the breadboard, as each alphanumeric display would need around 15 wires. The LCD screen proved to be adaptable, displaying a large amount of characters and only relying on inputs from the PIC. Secondly, the game only has the PIC in master mode instead of alternating it between master and slave. It was not necessary for the PIC to alternate between master and slave, since master mode sends and receives at the same time.

One of the most difficult parts of the project was to synchronize the FPGA and the PIC. The PIC ran on a 20 MHz clock while the FPGA's polling module ran on a 1 kHz clock. The interface had to assume that in one cycle of the polling module, the PIC could run whatever code it needed to before the polling FSM advanced. Additionally, this frequency discrepancy essentially made the SCK asynchronous compared to the FPGA clock. The 8-bit registers on the FPGA had to use either positive or negative edge of SCK to synchronize data transfer into and out of the PIC.

References

Crystalfontz LCD Data Sheet:
http://www.crystalfontz.com/products/0802a/CFAH0802AYYHJP.pdf

Harrisboard Schematic:
http://www3.hmc.edu/~harris/class/e155/harrisboardsch.pdf

PIC18F452 Data Sheet:
http://www3.hmc.edu/~harris/class/e155/pic18f452.pdf

Parts List

Part	Source	Vendor Part #	Price
8x2 Yellow-Green Reflective LCD	Crystalfontz America, Inc.	CFAH0802A-YYH-JP	$17.38

Appendix A: Verilog Code

```
//////////////////////////////////////////////////////////////////////////////////
// Author:  Andy Chin
// Email:   achin@hmc.edu
// Date:    11/19/06
// Description: PI memory game FSMs
// interfaces with PIC in master mode
//////////////////////////////////////////////////////////////////////////////////
module final_ac(clk, reset, SCK, hold, e, f, g, h, SDI, a, b, c, d, SDO, Y, ready,
                                LEDs, sevenseg, correct, incorrect);
    input clk;
    input reset;
    input SCK;          // input serial clock for receiving data
    input hold;         // signal from PIC to FPGA to prevent polling
    input e;                  // e,f,g,h are polling columns
    input f;
    input g;
    input h;
    input SDI;                // serial input from PIC
    output a;                 // a,b,c,d are polling rows
    output b;
    output c;
    output d;
    output SDO;               // serial output to PIC
    output [7:0] Y;
    output ready;
    output [9:0] LEDs;        // controls 10-bit LED array
    output [6:0] sevenseg;    // controlls hex display
    output correct;
    output incorrect;

            wire [7:0] s;

            freqdiv freqdiv1024(clk,reset,div); //divides clk by 1024
            polling polling1(div, hold, e, f, g, h, a, b, c, d, s, ready);  //polls
            bitshift bitshift1(SCK, reset, SDI, s, Y, SDO); // PIC/FPGA interface
            LEDarray ledarray1(Y, LEDs, sevenseg, correct, incorrect);  // display module

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
// Author: Andy Chin
// Email: achin@hmc.edu
// Date: 9/29/06
// Description:   module freqdiv: divides clock by 2^16 = 1024
////////////////////////////////////////////////////////////////////////////////
module freqdiv(clk,reset,div);
    input clk;
    input reset;
    output div;

reg [15:0] q;

      always@ (posedge clk, posedge reset)
            if (reset | (q[15] == 1)&q[7])
                  q <= 16'd0;
            else
                  q <= q+1;

      assign div = q[15];

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
// Author: Andy Chin
// Email: achin@hmc.edu
// Date: 10/23
// Description:   module bitshift: shift data in/out of FPGA with two 8-bit shift registers
////////////////////////////////////////////////////////////////////////////////
module bitshift(clk, reset, SSPI, s, Y, SDO);
    input clk;
    input reset;
    input SSPI;
    input [7:0] s;
    output reg [7:0] Y;
    output SDO;

            reg [3:0] t;      //counter to to shift s to ss
            reg [7:0] ss;     //8-bit output shift register

      always @ (posedge clk, posedge reset)
                  //change register to s, then shift out data to PIC
                  if (reset)
                              ss <= 8'd0;
                  else if (t == 4'd0)
                              ss <= s;
                  else
                              // output 8-bit shift register
                              ss <= {ss[6:0], 1'b1};

      //shift in data to shift register to be displayed in LED array and hex display
      always @ (negedge clk, posedge reset)
            begin
                  if (reset)
                        Y <= 8'd0;
                  else
                              // input 8-bit shift register (MSB first from PIC to FPGA)
                              //Y = {Y[6:0], SSPI};

                  // counts all 8 SCK cycles
                  if (reset | t[2]&t[1]&t[0])   // reset counter at end of 8 SCK cycles
                        t = 4'd0;
                  else
                        t = t+1;
            end

      assign SDO = ss[7];     // SDO = MSB of ss

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////
// Author: Andy Chin
// Email: achin@hmc.edu
// Date: 11/20/06
// Description: module polling: polls keypad by sending one column high at a time
//          codes value into an 8-bit character, transfers and holds after
//          key is depressed
//////////////////////////////////////////////////////////////////////////////
module polling(clk, reset, e, f, g, h,  a, b, c, d, s, ready);
    input clk;
    input reset;
    input e;                            // transfer comes from the shift register module
    input f;
    input g;
    input h;
    output a;
    output b;
    output c;
    output d;
    output reg [7:0] s;                 //value of key input (0x00-0x0F)
    output ready;                       //HIGH value for data transfer

            reg [7:0] q;                //counter for 80 ms hold
            reg [2:0] state, nextstate;

            parameter S0 = 3'b000;                          //poll state 0
            parameter S1 = 3'b001;                          //poll state 1
            parameter S2 = 3'b010;                          //poll state 2
            parameter S3 = 3'b011;                          //poll state 3
            parameter S4 = 3'b100;                          //transfer state (from FPGA to PIC)
            parameter S5 = 3'b101;                          //hold state

            //state register
            always@(posedge clk, posedge reset)
                  if (reset)
                        begin
                              state <= S5;                  //resets state to S0
                        end
                  else state <= nextstate;

            //nextstate logic
            //reset->polling->transfer->hold
            always@( * )                                          //polling S0-S3
                  case (state)
                        S0:if (e|f|g|h)                           //poll mode 0
                                    nextstate = S4;
                              else
                                    nextstate = S1;
                        S1:if (e|f|g|h)                           //poll mode 1
                                    nextstate = S4;
                              else
                                    nextstate = S2;
                        S2:if (e|f|g|h)                           //poll mode 2
                                    nextstate = S4;
                              else
                                    nextstate = S3;
                        S3:if (e|f|g|h)                           //poll mode 3
```

```
                        nextstate = S4;
                  else
                        nextstate = S0;
            S4:   nextstate = S5;   //initiate transfer state from FPGA to PIC
            S5:if (q[7])            //hold state
                        nextstate = S0;
                  else
                        nextstate = S5;
            default: nextstate = S0;
      endcase

//output logic

//only one column goes HIGH at any time for polling
assign a = (state==S0);
assign b = (state==S1);
assign c = (state==S2);
assign d = (state==S3);

assign ready = (state==S4);//sends HIGH value to PIC to start transfer/receive

assign shiftreset = (state == S4);  // to reset shift registers during polling

//assigns s, 8-bit s shift reg, and counters for transfer and hold states
always@(posedge clk)
      begin

            if (reset | q[7])       // counts to ~80ms
                  q = 8'd0;         // to prevent a transfer when not ready
            else if (state == S5)
                  q = q+1;          // increments counter if in hold state


            // all combos of s are assigned
            // "if/else if" block will only trigger during polling or reset state
                    if (a&e) s = 8'h01;         // s = "1"
            else if (a&f) s = 8'h04;            // s = "4"
            else if (a&g) s = 8'h07;            // s = "7"
            else if (a&h) s = 8'h0e;            // s = "E"
            else if (b&e) s = 8'h02;            // s = "2"
            else if (b&f) s = 8'h05;            // s = "5"
            else if (b&g) s = 8'h08;            // s = "8"
            else if (b&h) s = 8'h00;            // s = "0"
            else if (c&e) s = 8'h03;            // s = "3"
            else if (c&f) s = 8'h06;            // s = "6"
            else if (c&g) s = 8'h09;            // s = "9"
            else if (c&h) s = 8'h0f;            // s = "F"
            else if (d&e) s = 8'h0a;            // s = "A"
            else if (d&f) s = 8'h0b;            // s = "B"
            else if (d&g) s = 8'h0c;            // s = "C"
            else if (d&h) s = 8'h0d;            // s = "D"

      end

endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
// Author: Andy Chin
// Email: achin@hmc.edu
// Date: 11/17/06
// Description: module LEDarray: decodes 8 bit input from PIC to LED array and hex display
//////////////////////////////////////////////////////////////////////////////////
module LEDarray(number, LEDs, sevseg, correct, incorrect);
    input [7:0] number;
    output reg [9:0] LEDs;
    output reg [6:0] sevseg;
    output reg correct;
    output reg incorrect;

       // number is 8 bits from the PIC
       // LEDs is the 10 LED array, displaying the sequence of pi
       // sevseg corresponds to a seven segment display, where
       // sevseg = 7'bABC_DEFG
       // correct tells the user if they are right
       // incorrect tells the user if they are wrong

       // number is broken down as follows:
       // number[7] = not used
       // number[6] = not used
       // number[5] = correct
       // number[4] = incorrect
       // number[3:0] = number to display on sevenseg and LED array

       // output state
       always @ ( * )
            begin
                  case(number[3:0])
                        0: begin
                                    LEDs <= 10'b00_0000_0001;          // "0" lights up
                                    sevseg <= 7'b000_0001;
                              end
                        1: begin
                                    LEDs <= 10'b00_0000_0010;          // "1" lights up
                                    sevseg <= 7'b100_1111;
                              end
                        2: begin
                                    LEDs <= 10'b00_0000_0100;          // "2" lights up
                                    sevseg <= 7'b001_0010;
                              end
                        3: begin
                                    LEDs <= 10'b00_0000_1000;          // "3" lights up
                                    sevseg <= 7'b000_0110;
                              end
                        4: begin
                                    LEDs <= 10'b00_0001_0000;          // "4" lights up
                                    sevseg <= 7'b100_1100;
                              end
                        5: begin
                                    LEDs <= 10'b00_0010_0000;          // "5" lights up
                                    sevseg <= 7'b010_0100;
                              end
                        6: begin
                                    LEDs <= 10'b00_0100_0000;          // "6" lights up
```

```
                        sevseg <= 7'b010_0000;
                    end
                7: begin
                        LEDs <= 10'b00_1000_0000;           // "7" lights up
                        sevseg <= 7'b000_1111;
                    end
                8: begin
                        LEDs <= 10'b01_0000_0000;           // "8" lights up
                        sevseg <= 7'b000_0000;
                    end
                9: begin
                        LEDs <= 10'b10_0000_0000;           // "9" lights up
                        sevseg <= 7'b000_1100;
                    end
                default
                    begin
                        LEDs <= 10'b00_0000_0000;
                        sevseg <= 7'b111_1111;
                    end
            endcase

            correct = number[5];                // indicates if user is right
            incorrect = number[4];              // indicates if user is wrong

        end

endmodule
```

Appendix B: PIC C Code

```
/*
    Author: Chris Acon
    Email: cacon@hmc.edu
    Date: 11/20/06
    Description: Pi memorizing game, interfaces with FPGA and LCD
/*

#include <p18f452.h>
#include <timers.h>

void main(void);
void isr(void);
char begingame(void);
void engame(char highscore, char score, char icorrect);
void poll(void);
void display(char number, char thigh, char tlow);
void LCDdisplay(char code);
void LCDshift(char address);
void LCDclear(void);
void LCDinitialize(void);
void wait2ms(void);
void digdisplay(char dig);
void blinkeron(void);
void blinkeroff(void);
void LCDinstruct(int instruct);
void LCDenable(void);
void LCDdisable(void);

//GLOBAL VARIABLES
char loopexit;    //used to exit while loops for interrupts, made global to pass to ISR
char highscore;   //game high score
char readbuffer;  //clears BF when (readbuffer = SSPBUF;)
char score;       //player score
char masked;      //dummy variable

#pragma code low_vector = 0x18//ISR code vector
void low_interrupt(void)
{
      _asm
            GOTO isr
      _endasm
}

#pragma code

void main(void)
{
      char piarray[100] =    {3,1,4,1,5,9,2,6,5,3,  5,8,9,7,9,3,2,3,8,4,  6,2,6,4,3,3,8,3,2,7,
                               9,5,0,2,8,8,4,1,9,7,  1,6,9,3,9,9,3,7,5,1,  0,5,8,2,0,9,7,4,9,4,
                               4,5,9,2,3,0,7,8,1,6,  4,0,6,2,8,6,2,0,8,9,  9,8,6,2,8,0,3,4,8,2,
                               5,3,4,2,1,1,7,0,6,7};//100 digits of pi
      /*3141592653 5897932384 6264338327 9502884197 1693993751
        0582097494 4592307816 4062862089 9862803482 5342117067: 100 digits / 10 per grouping
```

```
      ref:
http://3.14159265358979323846264338327950288419716939937510582097494459 2.com/index1.html*/

      char i;                 //index variable
      char j;                 //index variable
      char startdigit;//particular digit of pi
      char digit;       //particular digit of pi
      char notwrong;          //won't record high score if user is incorrect
      char correct;
      char incorrect;
      char nolights;

      highscore = 0;//to reset high score
      correct = 0x2F;
      incorrect = 0x1F;
      nolights = 0x0F;
      score = 0;//initialize score

      // initialize relevant registers
      SSPSTAT = 0x00; // <7> Input data sampled at middle of data output time
                              // <6> Data tranmistted on falling edge of SCK

      SSPCON1 = 0x20;   //    0010 0000 (master mode)
                              //    [7]         collision flag, 0
                              //    [6]         o-flow flag, 0
                              //    [5]         enable serial ports
                              //    [4]         clock idle when low
                              //    [3-0] SPI master mode, clock/4

      TRISB = 0x01;     // TRISB<0> = input for "ready" signal after FPGA input
      LATB = 0x00;      // clear PORTB
      TRISC = 0x10;     //    [5] RC5/SDO = output
                              //    [4] RC4/SDI = input
                              //    [3] RC3/SCK = output (master mode)

      //T0CON = 0x88;//no prescaler, for SIM only
      T0CON = 0x87;/*1000 0111
                              //    [7]         enables Timer0
                              //    [6]         16 bit counter
                              //    [5]         internal clock
                              //    [4]         inc on low-to-high
                              //    [3]         timer0 is prescaled
                              //    [2:0] 1:256 prescaler used*/

      INTCON = 0x00;    //    0000 0000
                              //    [7]         Disable unmasked intrs (until needed)
                              //    [6]         Disable periperhal intrs
                              //    [5]         Enable TMR0 o-flow intr
                              //    [4]         Disable ext intr
                              //    [3]         Disable portchange intr
                              //    [2]         o-flow flag, 0
                              //    [1]         ext intr flag, 0
                              //    [0]         portchange intr flag, 0

      TRISA = 0x00;//sets PORTA to output

      LCDinitialize();
```

```
startdigit = begingame();     //initialize first game

//loop to end of pi (100 digits), increment max index
for(digit = startdigit; digit<101; digit++)
{
      PORTB = 0x07;                       //prevent accidental polling while displaying
      T0CON = 0x87;                       //enable TMR0 during display "for" loop
      notwrong = 1;

      for(i = 0; i<digit; i++)            //displays digits of pi, from first digit
      {
            SSPBUF = piarray[i];          //send pi digit thru SPI

            if (i==(digit-1))                   //if index is on last digit
            {
                  display(piarray[i],0xA0,0xB3);
            }
      /*    else if (i == (digit-2))
            {
                  display(piarray[i],0xB0,0xB3);
            }
            else if(i == (digit-3))
            {
                  display(piarray[i],0xC0,0xB3);
            }
            else if ((digit-i) < 7)
            {
                  display(piarray[i],0xDC,0x5F);
            }     */
            else
            {
                  display(piarray[i],0xEC,0x5F);
            }

            display(nolights,0xFC,0xB3);  //no lights on between each digit

}     //end for, digit display

PORTB = 0x00;                       //enable polling after displaying
LCDclear();
      for(i=0; i<digit; i++)        //waits for input, sends correct/incorrect signal
      {
            poll();
            LCDshift((i%8));
            if((i%8)==0)
                  LCDclear();
            digdisplay(SSPBUF);
            if(piarray[i] != SSPBUF)            //checks if keyed value is incorrect
            {
                  SSPBUF = incorrect;
                  while(SSPSTATbits.BF == 0)    //waits until data received complete
                  {}
                  readbuffer = SSPBUF;          //reads SSPBUF to clear BF bit
                  engame(highscore, score, piarray[i]);
                  digit = begingame();          //reset game, get new starting digit
                  digit = digit-1;              //make up for increment in top for loop
                  notwrong = 0;
```

```
                        score = 0;
                        break;//resets game
                  //needs to branch to incorrect sequence, i.e. GAME OVER/HI SCORE/NEW GAME
                  }
            }

            // 1 second correct signal until next iteration of the loop
            display(correct,0xA0,0xB3);
            if (notwrong)
                  score = digit;          //increment player score if correct for round

            // Store score if higher than High Score
            if ((digit > highscore) & notwrong)
                  highscore = digit;

      }     //end for loop, increment max index
}//end main

//INTERRUPT code
#pragma interruptlow isr
void isr(void)
{
      INTCON = 0x20;//reset overflow flag, disable intr
      loopexit = 1;//to exit TMR0 while loop
}

//initialize memory game by asking to start and for user input on starting digit
char begingame()
{
      char startdigit;
      char lastinput;
      char keycount;//counter for key presses

      startdigit = 0;

      // LCD: display "START?", "(*)"
      blinkeroff();
      LCDdisplay(0x53);//display "S"
      LCDshift(0x01);//sets DDRAM address to 0x01
      LCDdisplay(0x54);//display "T"
      LCDshift(0x02);//sets DDRAM address to 0x02
      LCDdisplay(0x41);//display "A"
      LCDshift(0x03);//sets DDRAM address to 0x03
      LCDdisplay(0x52);//display "R"
      LCDshift(0x04);//sets DDRAM address to 0x04
      LCDdisplay(0x54);//display "T"
      LCDshift(0x05);//sets DDRAM address to 0x05
      LCDdisplay(0x3F);//display "?"
      LCDshift(0x40);//sets DDRAM address to 0x40
      LCDdisplay(0x28);//display "("
      LCDshift(0x41);//sets DDRAM address to 0x41
      LCDdisplay(0x2A);//display "*"
      LCDshift(0x42);//sets DDRAM address to 0x42
      LCDdisplay(0x29);//display ")"

      // polls keypad until (*) key is pressed (encoded as "E")
      PORTB = 0x00;                       //enable polling to start game
```

```
while(1)                          // keeps checking user input until "*" is pressed
{
      poll();

      if(SSPBUF == 0x0E)          //checks if keyed value is correct
      {
            break;
      }
}

// LCD: display "START DIGIT?"
LCDclear();
LCDdisplay(0x53);//display "S"
LCDshift(0x01);
LCDdisplay(0x54);//display "T"
LCDshift(0x02);
LCDdisplay(0x41);//display "A"
LCDshift(0x03);
LCDdisplay(0x52);//display "R"
LCDshift(0x04);
LCDdisplay(0x54);//display "T"
LCDshift(0x40);
LCDdisplay(0x44);//display "D"
LCDshift(0x41);
LCDdisplay(0x49);//display "I"
LCDshift(0x42);
LCDdisplay(0x47);//display "G"
LCDshift(0x43);
LCDdisplay(0x49);//display "I"
LCDshift(0x44);
LCDdisplay(0x54);//display "T"
LCDshift(0x45);
LCDdisplay(0x3F);//display "?"

keycount = 0;

// polls keypad for starting digit until * is pressed
while(1)
{
      poll();

      if(SSPBUF != 0x0E)                    //checks if keyed value is not *
      {
            startdigit = 10*startdigit + SSPBUF;
            if(keycount == 0) {
                  LCDclear();
                  blinkeron();
                  LCDshift(0x00);//begin display “>”,”(* ends)”
                  LCDdisplay(0x3E);//display ">"
                  LCDshift(0x40);
                  LCDdisplay(0x28);//display "("
                  LCDshift(0x41);
                  LCDdisplay(0x2A);//display "*"
                  LCDshift(0x43);
                  LCDdisplay(0x65);//display "e"
                  LCDshift(0x44);
```

```
                        LCDdisplay(0x6E);//display "n"
                        LCDshift(0x45);
                        LCDdisplay(0x64);//display "d"
                        LCDshift(0x46);
                        LCDdisplay(0x73);//display "s"
                        LCDshift(0x47);
                        LCDdisplay(0x29);//display ")"

                  }
                  LCDshift(0x02+keycount);
                  digdisplay(SSPBUF);
                  keycount++;
            }//end if
            else                        //keyed value is *, output current startdigit
            {
                  if (startdigit == 0)
                  {
                        startdigit = 1;
                  }
                  break;
            }

      }

      LCDclear();

      return(startdigit);
}

//displays game over, user score, high score
void engame(char highscore, char score, char icorrect)
{
      char D;//score variable
      char i;//index
      int j;//index

      // LCD: display "GAMEOVER"
      LCDclear();
      blinkeroff();
      LCDdisplay(0x47);//display "G"
      LCDshift(0x01);
      LCDdisplay(0x41);//display "A"
      LCDshift(0x02);
      LCDdisplay(0x4D);//display "M"
      LCDshift(0x03);
      LCDdisplay(0x45);//display "E"
      LCDshift(0x04);
      LCDdisplay(0x4F);//display "O"
      LCDshift(0x05);
      LCDdisplay(0x56);//display "V"
      LCDshift(0x06);
      LCDdisplay(0x45);//display "E"
      LCDshift(0x07);
      LCDdisplay(0x52);//display "R"

      LCDshift(0x40);
      LCDdisplay(0x4D);//display "M"
```

```
LCDshift(0x41);
LCDdisplay(0x69);//display "i"
LCDshift(0x42);
LCDdisplay(0x73);//display "s"
LCDshift(0x43);
LCDdisplay(0x73);//display "s"
LCDshift(0x44);
LCDdisplay(0x65);//display "e"
LCDshift(0x45);
LCDdisplay(0x64);//display "d"
LCDshift(0x46);
LCDdisplay(0x3A);//display ":"
LCDshift(0x47);
digdisplay(icorrect);

for (j=0; j<3; j++)
{
      TMR0H = 0xB3;
      TMR0L = 0xB4;                        //TMR0 set for until intr
      INTCON = 0xA0;
      loopexit = 0;                        //to enter while loop on next line
      while(loopexit == 0)                 //loop until interrupt (isr sets loopexit = 1)
      {}
}

// LCD: display "Score:" + player's score
LCDclear();
LCDdisplay(0x53);//display "S"
LCDshift(0x01);
LCDdisplay(0x63);//display "c"
LCDshift(0x02);
LCDdisplay(0x6F);//display "o"
LCDshift(0x03);
LCDdisplay(0x72);//display "r"
LCDshift(0x04);
LCDdisplay(0x65);//display "e"
LCDshift(0x05);
LCDdisplay(0x3A);//display ":"
for(i=0;i<2;i++) {
      if(i==0) {
            LCDshift(0x40);
            D = (int) score/10;//for tens digit of score
      }
      else {
            LCDshift(0x41);
            D = score - D*10;//ones digit of score
      }
      digdisplay(D);
}//end for

//pause for display of "Score:"

for (j=0; j<3; j++)
{
      TMR0H = 0xB3;
      TMR0L = 0xB4;                 //TMR0 set for until intr
      INTCON = 0xA0;
```

```
            loopexit = 0;                 //to enter while loop on next line
            while(loopexit == 0)          //loop until interrupt (isr sets loopexit = 1)
            {}
      }

      // LCD: display "HiScore" + high score
      LCDclear();
      LCDdisplay(0x48);//display "H"
      LCDshift(0x01);
      LCDdisplay(0x69);//display "i"
      LCDshift(0x02);
      LCDdisplay(0x53);//display "S"
      LCDshift(0x03);
      LCDdisplay(0x63);//display "c"
      LCDshift(0x04);
      LCDdisplay(0x6F);//display "o"
      LCDshift(0x05);
      LCDdisplay(0x72);//display "r"
      LCDshift(0x06);
      LCDdisplay(0x65);//display "e"
      LCDshift(0x07);
      LCDdisplay(0x3A);//display ":"

      for(i=0;i<2;i++) {
            if(i==0) {
                  LCDshift(0x40);
                  D = (int) highscore/10;//tens digit of hiscore
            }
            else {
                  LCDshift(0x41);
                  D = highscore - D*10;//ones digit of hiscore
            }
            digdisplay(D);
      }//end for

      //pause for display of "HiScore:"
      for (j=0; j<3; j++)
      {
            TMR0H = 0xB3;
            TMR0L = 0xB4;                 //TMR0 set for until intr
            INTCON = 0xA0;
            loopexit = 0;                 //to enter while loop on next line
            while(loopexit == 0)          //loop until interrupt (isr sets loopexit = 1)
            {}
      }

      LCDclear();
}

void poll()
{
      while (PORTBbits.RB0 == 0)          //wait for user input
      {}
      SSPBUF = 0x0F;                      //write dummy data to receive key input from FPGA
      while(SSPSTATbits.BF == 0)          //waits until transmission complete
      {}
      while (PORTBbits.RB0 == 1)          //wait until FPGA goes into hold mode
```

```
      {}
}

void display(char number, char thigh, char tlow)
{
            SSPBUF = number;              //correct = 00101111
            while(SSPSTATbits.BF == 0)
            {}
            readbuffer = SSPBUF;          //reads SSPBUF to clear BF bit
            TMR0H = thigh;
            TMR0L = tlow;                 //TMR0 set for until intr
            INTCON = 0xA0;
            loopexit = 0;                 //to enter while loop on next line
            while(loopexit == 0)          //loop until interrupt (isr sets loopexit = 1)
            {}
}

void LCDdisplay(char code)
{
      //instruction 10_####_#### display character
      LCDinstruct(0x200 + code);
}

void LCDshift(char address)
{
      //instruction 00_1###_#### display character
      LCDinstruct(address + 0x80);
}

void LCDclear(void)
{

      //instruction 00_0000_0001 clears display
      LCDinstruct(1);

      //instruction 00_0000_001X returns home, address 0x00
      LCDinstruct(2);
}

void LCDinitialize(void)
{
            TMR0H = 0xB3;                       // 1 s delay
            TMR0L = 0xB4;
            INTCON = 0xA0;
            loopexit = 0;                       //to enter while loop on next line
            while(loopexit == 0)                //loop until interrupt (isr sets loopexit = 1)
            {}

      //instruction 00_0011_10XX sets functions for initializing
      LCDinstruct(0x38);

            TMR0H = 0xFE;                       // 15 ms delay
            TMR0L = 0xDB;
            INTCON = 0xA0;
            loopexit = 0;                       //to enter while loop on next line
            while(loopexit == 0)                //loop until interrupt (isr sets loopexit = 1)
            {}
```

```
	LCDenable();
	LCDdisable();

		TMR0H = 0xFE;			// 15 ms delay
		TMR0L = 0xDB;
		INTCON = 0xA0;
		loopexit = 0;			//to enter while loop on next line
		while(loopexit == 0)		//loop until interrupt (isr sets loopexit = 1)
		{}

	LCDenable();
	LCDdisable();

	wait2ms();

	LCDenable();
	LCDdisable();

	wait2ms();

	//instruction 00_0000_1000 display off
	LCDinstruct(0x08);

	//instruction 00_0000_0001 clears display
	LCDinstruct(0x01);

	//00_0000_0110 entry mode set
	LCDinstruct(0x06);

	blinkeron();
}

void wait2ms(void)
{
	TMR0H = 0xFE;			// 2 ms delay (max delay on an LCD instruction)
	TMR0L = 0xDB;
	INTCON = 0xA0;
	loopexit = 0;			//to enter while loop on next line
	while(loopexit == 0)		//loop until interrupt (isr sets loopexit = 1)
	{}
}
 void digdisplay(char dig)//displays polled digit
{
	switch (dig) {
		case 0: LCDdisplay(0x30);//display "0"
			break;
		case 1: LCDdisplay(0x31);//display "1"
			break;
		case 2: LCDdisplay(0x32);//display "2"
			break;
		case 3: LCDdisplay(0x33);//display "3"
			break;
		case 4: LCDdisplay(0x34);//display "4"
			break;
		case 5: LCDdisplay(0x35);//display "5"
			break;
```

```
            case 6: LCDdisplay(0x36);//display "6"
                        break;
            case 7: LCDdisplay(0x37);//display "7"
                        break;
            case 8: LCDdisplay(0x38);//display "8"
                        break;
            case 9: LCDdisplay(0x39);//display "9"
                        break;
            default: LCDdisplay(0x30);//display "0"
                        break;
      }//end case
}

void blinkeron(void)
{
      //instruction 00_0000_1101 display on
      LCDinstruct(0x0D);
}

void blinkeroff(void)
{
      //instruction 00_0000_1100 display on
      LCDinstruct(0x0C);
}

void LCDenable(void)
{
      //enable bit RB3 = 1, remaining PORTB unchanged
      PORTB = 0b00001000 | (0b11110111 & PORTB);
}

void LCDdisable(void)
{
      //enable bit RB3 = 0, remaining PORTB unchanged
      PORTB = 0b11110111 & PORTB;
}

void LCDinstruct(int instruct)
{
      //send 10 byte instruction to LCD
      //instruct[9:0]
      //    =[RS,R/W,DB7,DB6,DB5,DB4,DB3,DB2,DB1,DB0]
      //    =[RC2,RC1,RC0,RC6,RA5,RA4,RA3,RA2,RA1,RA0]

      //change PORTC values RC0,RC1,RC2,RC6, remaining PORTC unchanged
      PORTC = ((instruct&0x380)/128 + (instruct&0x40)) | (0b10111000 & PORTC);

      LCDenable();

      //change PORTA values [5:0], remaining PORTA unchanged
      PORTA = (instruct&0x3F) | (0b11000000 & PORTA);

      LCDdisable();

      wait2ms();
}
```