# Programmable Controller for the Super Nintendo Entertainment System

Final Project Report
December 12, 2002
E155: Microprocessor-Based Systems

**Ryan Crabb and Galway O'Mahony**

## Abstract:

In popular fighting games for systems such as the Super Nintendo Entertainment System, powerful special moves often consist of semi-complicated sequences of button presses. This project prototypes one component of a controller adapter that would allow the game player to record and playback such button sequences. The system consists of a 128 × 64 pixel LCD screen, a Super Nintendo and its controller, an FPGA to serve as RAM, and an additional FPGA to control the RAM and LCD. The system is controlled by three buttons: one to record a sequence in the RAM, another to stop the recording process, and a third to display a recorded sequence on the LCD.

# Introduction

The Nintendo Co. released its 16-bit Super Nintendo Entertainment System (SNES) in 1991 as a successor to its hugely popular 8-bit video game console, the Nintendo Entertainment System. The SNES came with a controller that was equipped with a basic four-direction digital control pad and eight digital buttons. This controller was adequate for most games, but users quickly realized that huge advantages could be obtained with features such as turbo buttons (automatic rapid fire) and slow motion. In response to these demands, STD Entertainment released the "SN ProgramPad," a Super Nintendo controller that included a variety of these features. In many popular fighting games, a player could use a "special move" by pressing a certain sequence of buttons. The SN ProgramPad came with a number of pre-programmed button sequences, and also allowed the user to record custom sequences. When a sequence was recalled, it was automatically sent to the SNES and simultaneously displayed on a built-in LCD. In this project, we have recreated the LCD display of the ProgramPad using two FPGA's, an LCD, and an SNES with its controller. The ultimate goal of the project was to recreate the ProgramPad in its entirety, but due to time constraints this full functionality was deemed to be outside the scope of the project.

The project is essentially an add-on to the Super Nintendo controller that gives the user the ability to record a button sequence using a Xilinx Spartan XCS10 FPGA. When desired, the user may push a button that displays the sequence on an LCD. There are three modes of operation: Free Play, Data Record, and Data Playback. While in Free Play, the module in between the SNES controller and the SNES will have no effect, allowing the user to play a game as usual. While in Data Record mode, which the user can enter by pressing a "Record" button, all SNES buttons pressed will be recorded in the RAM of the FPGA. In Data Playback mode, the

recorded sequence is displayed on a LCD screen, shown as a picture of the controller with darkened areas to represent active buttons.

The system consists of an FPGA that captures the controller data and stores it in internal RAM (each CLB of the FPGA can be used as a 32 × 1 single-port RAM array), a Crystalfontz CFAG12864B-WGH-V 128 × 64 pixel graphical display module, and a second FPGA that controls the RAM and LCD screen. The Control FPGA bases the timing of its control signals on clock signals from the SNES. It sends an address and write signal to the RAM, and when necessary the RAM sends stored data back. The Control FPGA also sends control and data signals to the LCD display module. The LCD occasionally performs internal operations, during which it is inaccessible to the controller. Thus, the LCD data lines are bidirectional, allowing the FPGA to determine the internal status of the LCD with a *busy* signal. The RAM and Control FPGAs both get their button data in serial form from the SNES controller.

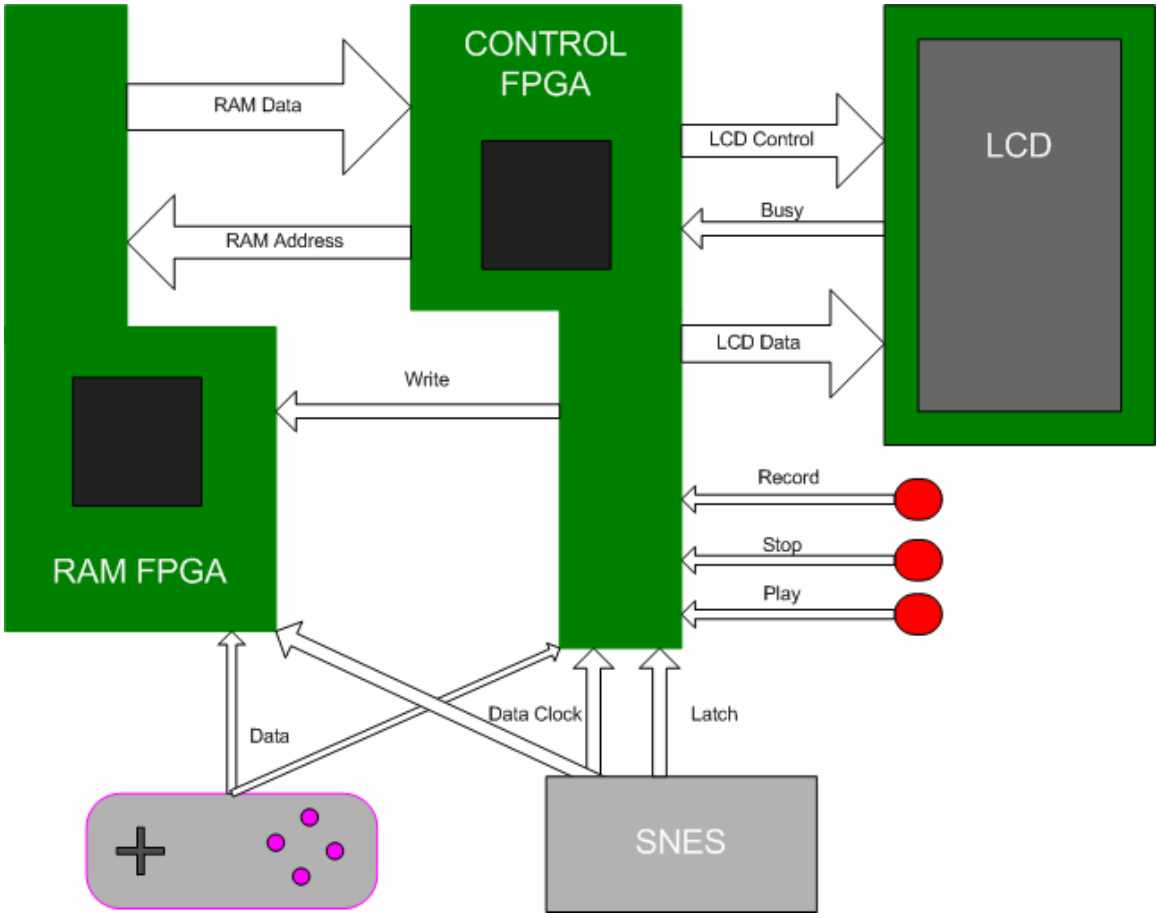


**Figure 1: This block diagram of our system shows the interaction between the physical components.**

## New Hardware

The project uses two new pieces of hardware that were not introduced in class. One is the SNES and its controller, which supply clock signals and the data, respectively, for most of the components of our system. The other new piece of hardware is the Crystalfontz graphical LCD that is used to display the button presses.

**Super Nintendo Entertainment System and Controller**

The SNES controller sends serial button data out in response to a latch signal generated by the SNES. Every 16.67msecs, the SNES sends a 12-µsec long data latch to the controller, which activates the controller's communications sequence. 6µsecs after the falling edge of the data latch, the SNES sends out a data clock with a 12µsec period that starts and ends in the logic high state. This data clock goes through 16 cycles, the first 12 of which correspond to particular buttons on the SNES controller. The SNES controller outputs the state of the appropriate button on the rising edges of the data clock (except for the first button, which is triggered by the rising edge of the data latch), and the SNES reads the button states on the falling edges of the data clock. Logic low on the data line corresponds to a button being pushed, while logic high represents the opposite. A timing diagram for the SNES-to-SNES-controller communications is shown in Figure 2. The delays between data transitions and data reads in the SNES communications protocol provide ample time for an FPGA to read the signals and perform any necessary operations on them. The main portion of the FPGA will take in the signals shown and perform the appropriate manipulations based on the current mode of operation, which is controlled by the user.

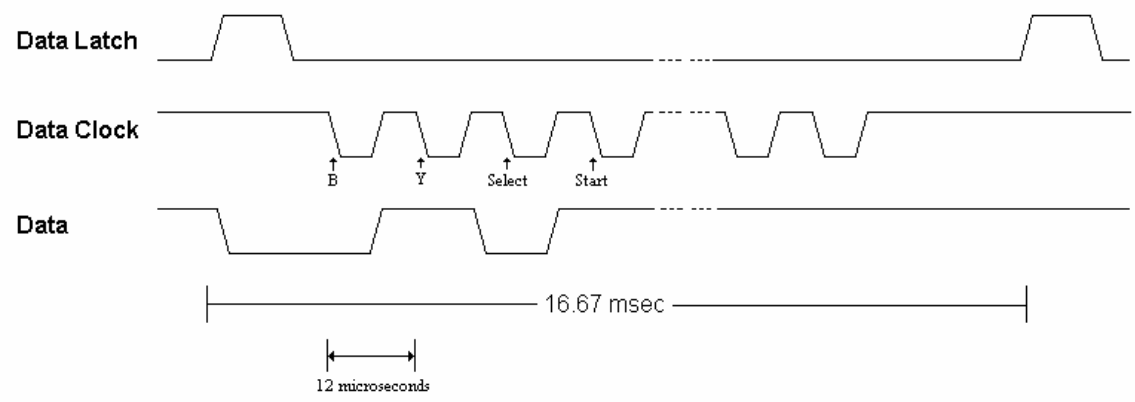


**Figure 2: SNES communications timing diagram. The SNES sends out a data latch and data clock, while the SNES controller sends out button data. As shown, buttons "B" and "Select" are pressed.**

**Crystalfontz CFAG12864B-WGH-V Graphical Display Module**

The Crystalfontz CFAG12864B-WGH-V is a graphical LCD divided into two halves, each 64 x 64 pixels. Each half of the LCD is controlled by a separate control chip, which references sets of 8 pixels by their address (column) and page (set of 8 rows). Writing to the LCD consists of first sending a 6-bit address and a 3-bit page, and then sending an 8-bit parallel data set corresponding to the 8 pixels contained in the specified column and page.

The LCD uses two control lines to select between the two halves of the display (CS1 and CS2), a screen reset signal (RST), a read/write signal (R/W), a line to specify whether the data being sent is display data or an instruction (D/I), and a clocking signal (E). Each operation performed on the LCD consists of first sending the status check pattern on the LCD control lines and checking for the busy signal (see Crystalfontz data sheet for details), sending the data, switching the E clock high, and then switching the E clock low.

The HC11 microcontroller is perfectly suited to control the LCD display. However, for this project we proposed to use only the FPGA to control our system, and we decided that we were obligated to meet the specifications we had originally proposed. In retrospect, this was a poor decision. The ability of the HC11 to execute subroutines makes interfacing with the LCD very simple. Using the FPGA, we were able to mimic subroutines using finite state machines. However, this made interfacing with the LCD a significantly more complex and considerably more difficult to debug. For details, see Appendix B.

# Schematics

The schematics of our breadboard layout are shown below. $V_{dd}$ and ground are supplied by the SNES, while the LCD screen uses an additional external power source $V_0$ for contrast

adjustment. The Record, Stop, and Play buttons are tied to $V_{dd}$ through 47k resistors and are pulled low using mechanical switches. Full pin descriptions are found in appendix B. Also, to demonstrate the RAM's functionality, we put out the RAM's output to 12 LEDs through a line with 330 Ohm resisitors.

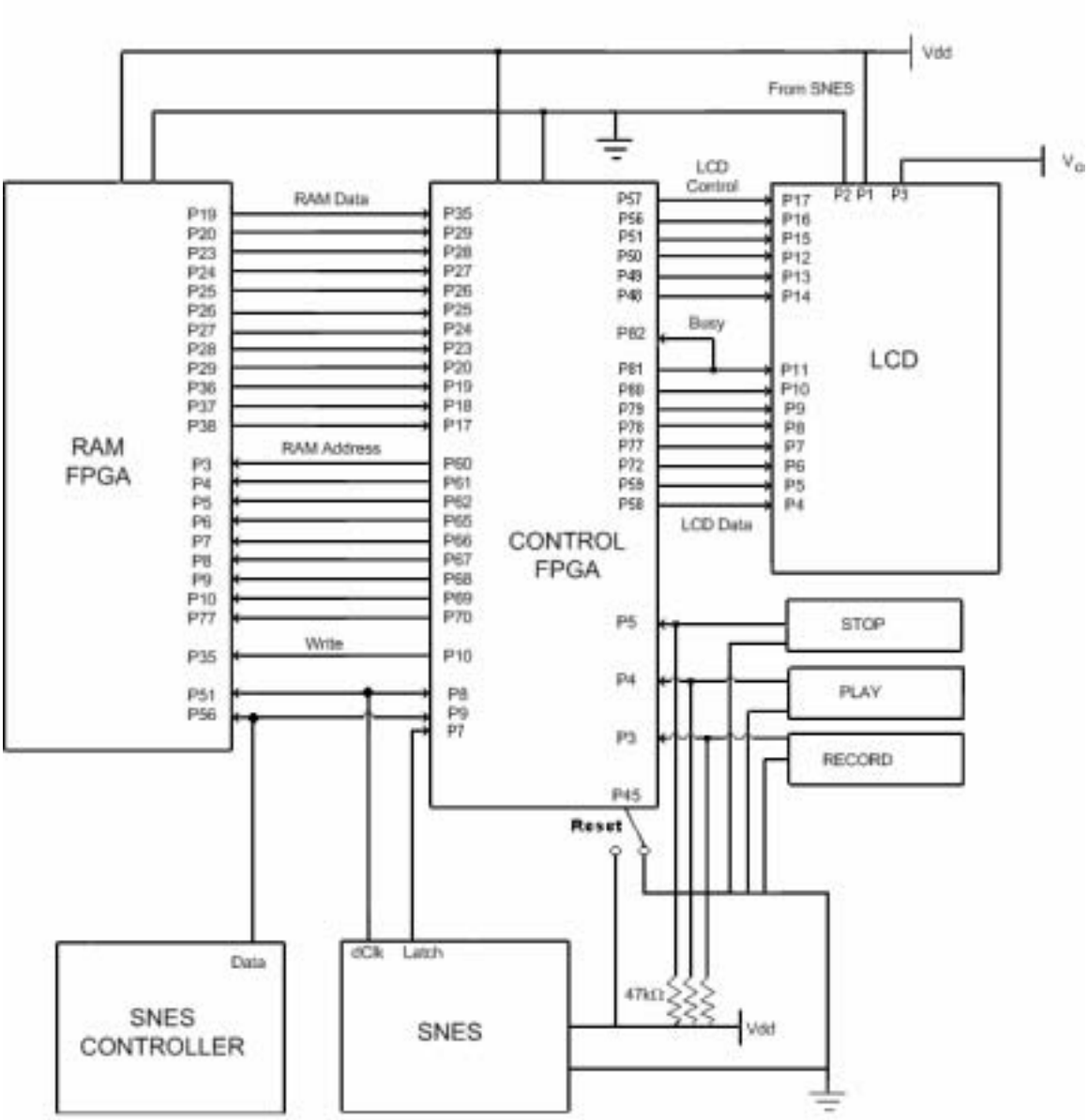


**Figure 3: Breadboard Schematics**

# Control FPGA Component

The Control FPGA essentially has three operational modes: normal operation or Free Play, Record, and Playback. On reset, the FPGA automatically enters the Free Play mode, during which nothing is displayed on the LCD screen. The user has three buttons to control the mode of operation: Record, Stop, and Play. Pressing the Record button sends the FPGA into Record Mode, during which all data from the SNES controller is stored in RAM and displayed on the LCD. This mode automatically ends after 5 seconds, or can be manually exited by pushing the Stop button. The recorded sequence can be sent to the LCD by pushing the Play button, which sends the FPGA into Playback mode. During this mode, SNES controller buttons have no effect on the FPGA.

The FPGA is divided into four primary modules: the Main Controller, the Record Module, the Playback Module, and the LCD Control. The Main Controller acts as the hub, receiving signals from the SNES and SNES controller and passing signals between the other modules. The Main Controller gets the Data Latch and Data Clock from the SNES along with Data from the SNES controller and the mode control signals from the user. It then enters the appropriate mode of operation and passes the signals to the appropriate modules.

**Main Controller**

The main module can be described as a state-machine with three states. It sends control signals to the other modules and routes the data between them based on the states. The control signals consist of *enable signals* for the record and playback modules, respectively. The data sent out includes: an *address* sent to RAM sourced from either the record or playback modules, *display data* for the LCD display module sourced from either the record module or RAM, a *final address* sent to the playback module (used to end playback), and *data clock* and *latch* signals

(originally sourced from the SNES, but synched with the main clock using flip-flops) for the record and playback modules, respectively.

The first state (and reset state) is a *wait* state or a *free* state. In this state, the play and record *enable signals* are set low, the *display data* is set all high (meaning no button data), and the *address* sent to RAM stays at zero. This state is left when either the record or play buttons are pressed.

When the record signal is triggered, the *record* state is entered. In this state, the record module's *enable signal* is set high and the *display data* and RAM *address* are set to come from the record module. When the record module sets the *complete flag*, the *final address* reached by the record module is stored in a register and the *free* state is entered again.

When the play signal is triggered, the *playback* state is entered. In this state, the play module's *enable signal* is set high, the *address* is set to come from the play module and the *display data* is set to come from the RAM. When the playback *complete flag* is set, the *free* state is once again entered.

**Record Module**

The record module consists of a 17-state finite-state-machine and a 16-bit shift register. It takes as input the (synched) *data clock* from the SNES, the serial *data* signal from the SNES controller, the signal from the stop button, the *enable signal* from the main module, and *reset*. It outputs a *write* signal to the RAM, the *button data* for the display module, an *address* for the RAM, and a *complete flag* for the main module.

In the reset state (or $0^{th}$ state), the *write* signal will be held low and the *address* will stay at zero. When the record module is enabled, it will start changing states at each negative edge of the *data clock*. The SNES sends the *data clock* in groups of exactly 16 clock cycles,

corresponding to each of the 12 buttons plus four buffer cycles. The shift register takes in the serial *data* from the SNES controller at each negative edge of the *data clock*, so that after the series is finished, bits 15-4 will hold the *button data*. In the $17^{th}$ state (corresponding to the $16^{th}$ negative clock edge) the *write* signal is set high and the *address* is incremented. When the *write* signal goes high, the main module takes the *address* value and stores it in the *final address* register (for ending playback). After about 16 milliseconds the *data clock* sequence will recommence and the FSM returns to the first state (not the $0^{th}$). If the *address* reaches 300 or the stop button is pressed, then the *complete flag* goes high. The main module will return to the free state, and the record will no longer be enabled, thereby returning it to the $0^{th}$ state. When this happens, the highest address reached will remain stored in the *final address* register, to be used by the record module.

**Playback Module**

The playback module is not much more than a counter. It takes in the (synched) *latch* signal, an *enable signal*, the *final address*, and *reset*. It outputs a RAM *address* and a *complete flag*. If *reset* is high or the *enable signal* is low, then the *address* is kept at zero. When enabled, the *address* will increment at the positive edge of the *latch*. When the *address* is equal to the *final address* then the *complete flag* goes high. When the main module receives this signal, it will return to the *free* state.

A minor flaw in this design (though one that does not affect functionality) is that if the play button is held, then the main state will be forced to remain in the *playback* state, and the *address* will continue to increment even after the *final address* has been reached. If this happens, the playback module will continue to output addresses until the *address* wraps back around to the *final address*.

**LCD Control**

The LCD control consists of two finite state machines: one which controls the button being written to the FPGA, and another which controls all operations related to writing the button to the screen.

The first FSM goes through an initialization process on reset that erases all pixels on the screen. This is necessary because the LCD powers up with all pixels turned on. After initializing, the FSM waits in an idle state until it receives a latch signal. Upon receiving the latch, the FSM begins sequencing through the SNES buttons and checking to see if they are active. If a button is pushed, the FSM triggers the display sequence and waits until it is complete. It then continues polling the different buttons. For further details, see Figure ***** in Appendix (((((;

The second FSM consists of a number of subroutines that manage the control and data lines of the LCD, and keep track of what part of the LCD is being written. Each subroutine consists of around five states that go through the LCD interface process, writing the data line, switching the E clock on and off, and waiting. The FSM starts in an initialization state. A signal from the first FSM triggers the first transition. The FSM then performs a status check, writes the address, checks the status again, writes the page, checks the status again, writes the data, checks to see if the location has reached its limit, and then either increments the location or returns to the initialization procedure. The overall flow diagram for the FSM is shown in Appendix (((((, along with diagrams of each subroutine.

Data Bit 7 is used as the busy signal during status checks. If the bit is high, then the LCD is performing internal operations and will not respond to user inputs. The LCD data lines from the FPGA are controlled using a tristate buffer. During regular states, the lines act as outputs. However, in status check routines, the FPGA stops driving the bits, and one of the lines is tapped

off to be used as the busy signal. As long as the busy signal is high, the FSM remains in the status check state.

## RAM FPGA Component

The RAM component, as would be expected, is very simple. It consists of an array of 300 flip-flops and a 16-bit shift register. It takes in as input the data clock from the SNES, the serial data from the SNES controller, the write signal from the record module and an address from the main controller. It outputs a 12-bit data bus to the main controller.

The shift register is identical to that of the record module. In fact, the only reason that the RAM has its own shift register is that it is easier to take in the single serial data than to take the 12-bit data from the Controller FPGA. After the $16^{th}$ data clock cycle, the write signal will go high, and on the positive edge of write the 12-bit register specified by the address will get bits 15-4 from the shift register. Always at the change of address, the RAM will output the data stored in the register of that new address.

## Results

The result of this project is a system that can record a sequence of SNES controller button presses for up to five seconds and output the sequence in 12-bit parallel form.

We encountered several large difficulties throughout our project that caused us to change our initial plans for the system. Firstly, we were not able to properly send all three signals between the SNES and controller (data clock, latch, and button data) through our FPGA simultaneously. In simply taking the signals as inputs and assigning them as outputs, we got cross talk between the signals. For reasons unknown to us, the clock signal would bleed into the

latch signal (only if the data line was connected as well) and prevent the control from responding properly. Because of this, we abandoned the idea of sending the signal back to the SNES and decided to focus on the LCD display. Additionally, this caused unreliability at times in simply inputting the stream correctly.

The greatest difficulty in our design process was the LCD control. In retrospect, the HC11 may have been better suited for the control of this device (and offered an additional piece of hardware). The system contained a complicated finite state machine of over 40 states to produce the signals needed by the LCD. The complexity of the state machine made debugging very difficult. We had, at one point a display module that could play button press sequences, although the drawing rate was very slow. Unfortunately, in an attempt to improve it, we lost any sort of functionality of the graphical LCD.

Our resulting product also differs from our proposal in our use of RAM. We had intended to use external RAM (Cypress Semiconductor Corporation's CY6264-70SNC, 8k × 8 SRAM) as opposed to a second FPGA. Unfortunately, we did not research the product as well as we should have. It was ordered online, and when it arrived we found the pins to be too small and inaccessible for our use. Because of our decision to use the external RAM (to make up for not sending the signal back to the SNES) was too last-minute we decided to go back to our original plan of using the FPGA for RAM.

# References

[1] CY6264-70SNC Datasheet,
http://rocky.digikey.com/WebLib/Cypress/Web%20Data/CY6264.pdf.

# Parts

| Part | Source | Vendor # | Price |
|---|---|---|---|
| 128 × 64 Graphic LCD | Crystalfontz | **CFAG12864B-WGH-V** | $37.03 |
| IC SRAM 8KX8 PD 28-SOIC | DigiKey | 428-1085-ND | $3.42 each |

## Appendix A: State Transition Diagrams

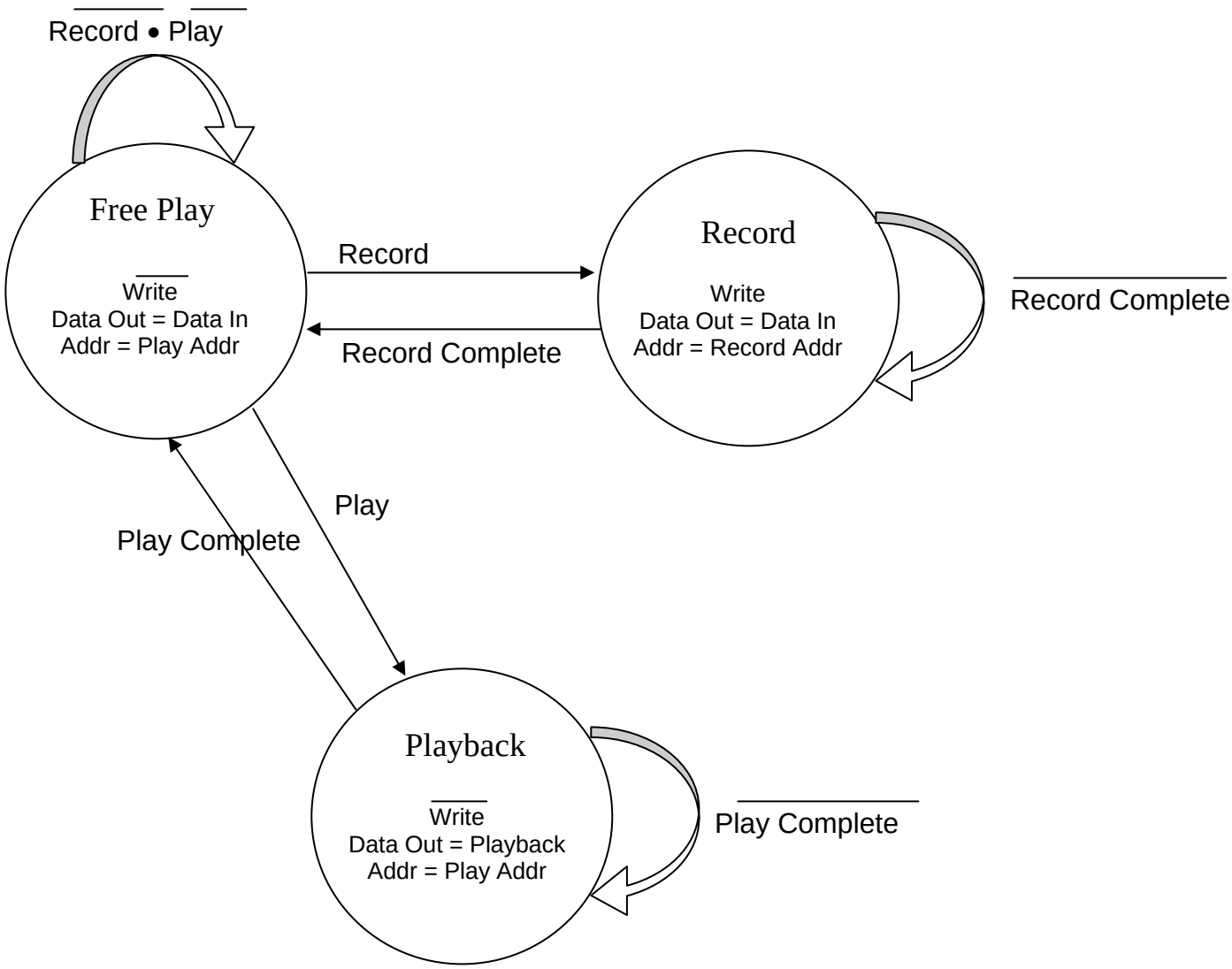


**Figure 4: Main State Transition Diagram**

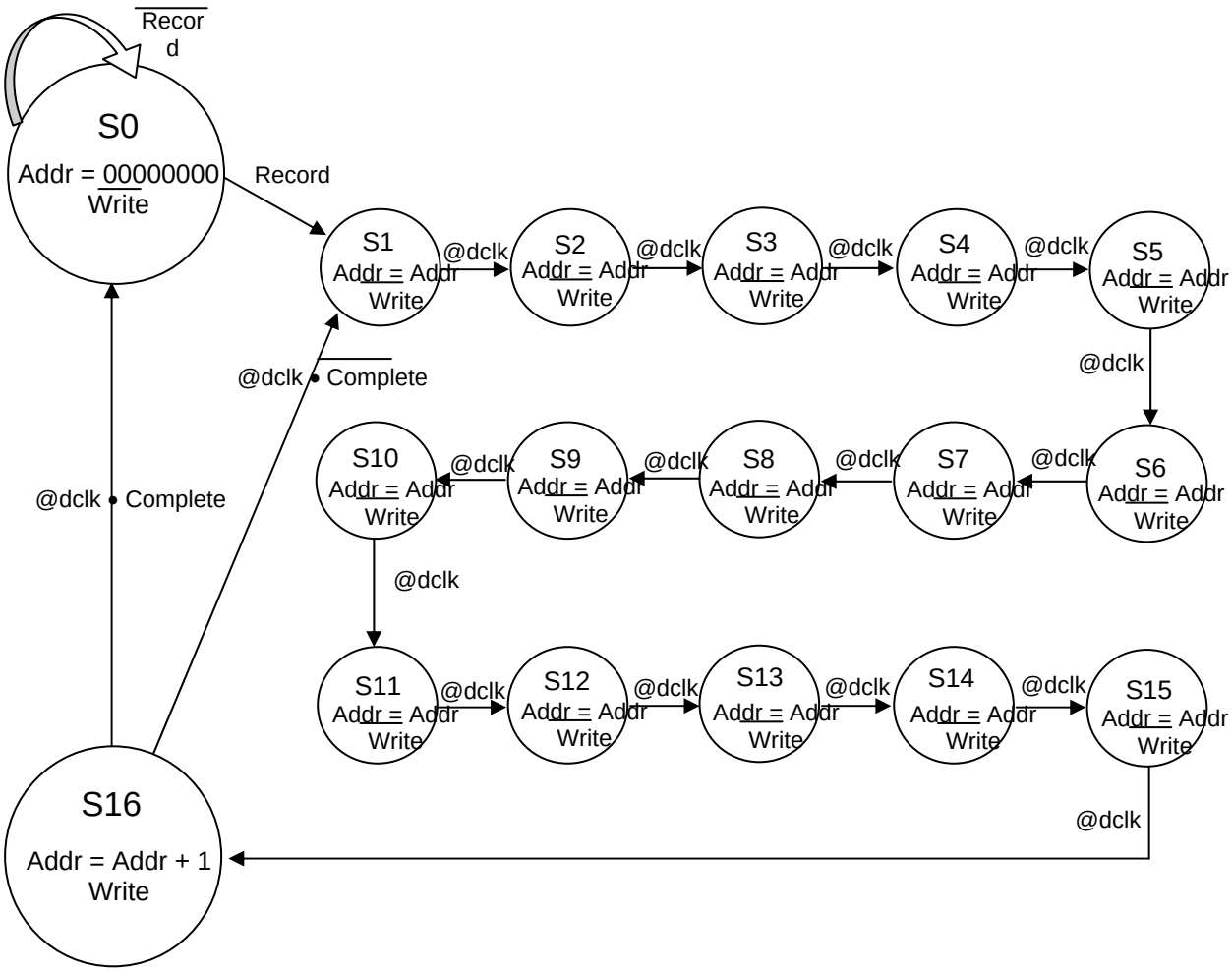


**Figure 5: Recording State Transition Diagram**

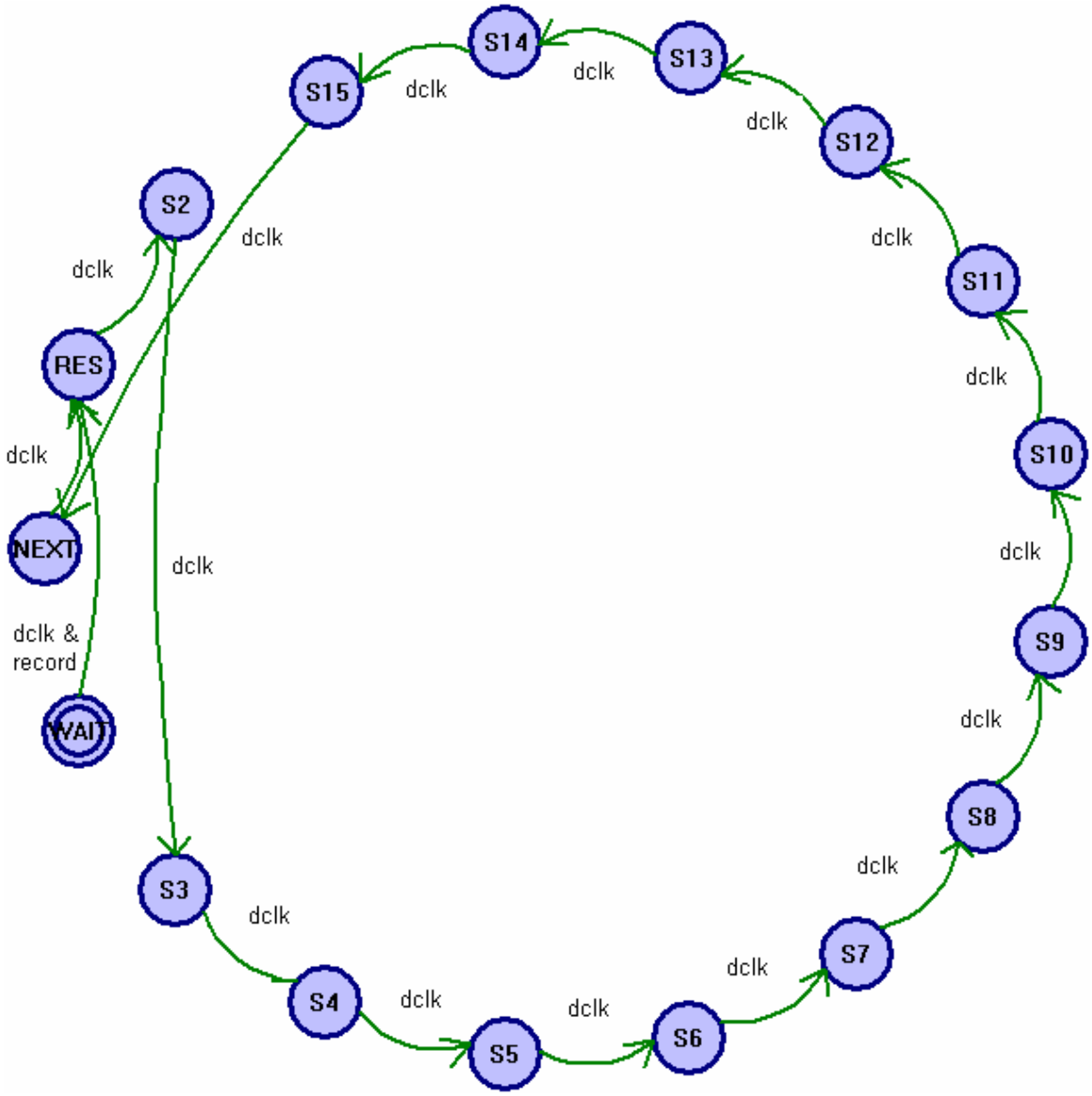


**Figure 6: Playback State Finite State Machine**

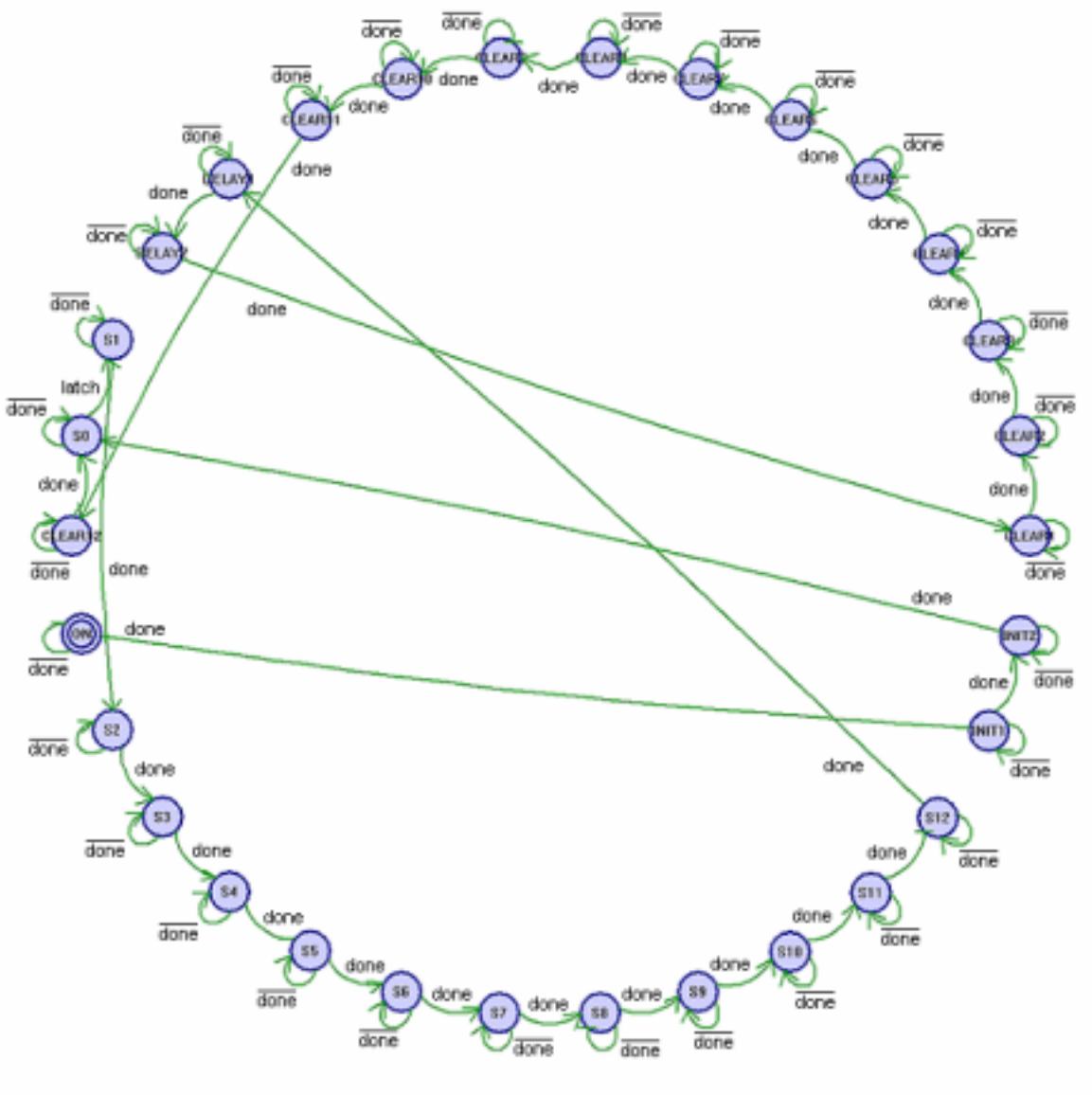


**Figure 7: Write LCD State Transition Diagram**

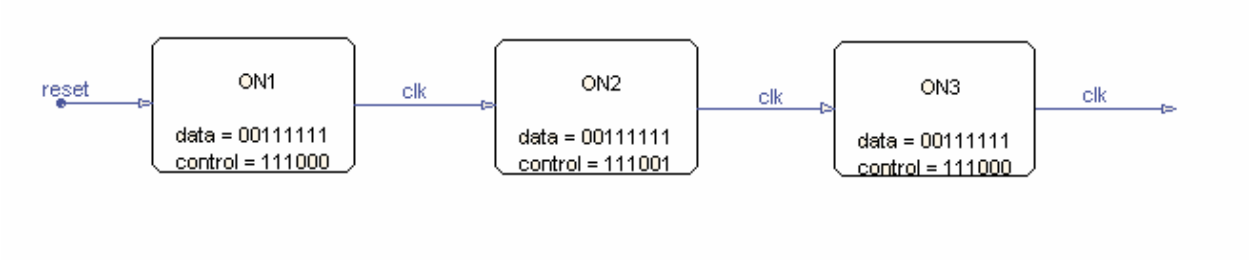


**Figure 8: Write LCD Initialization State Transition Diagram**

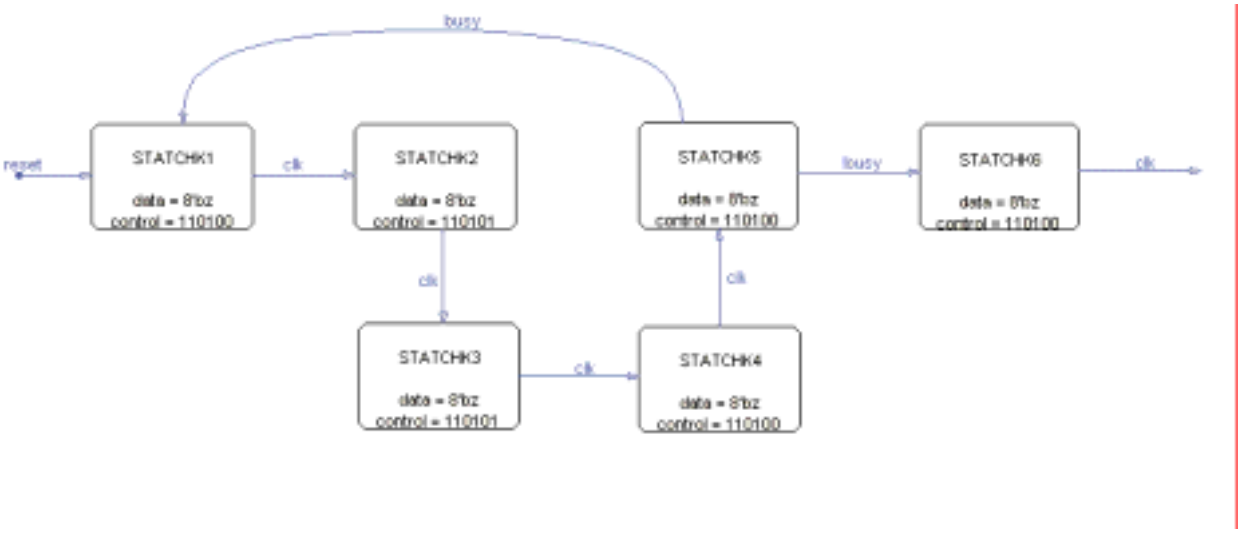


**Figure9: Write LCD Status Check State Transition Diagram**

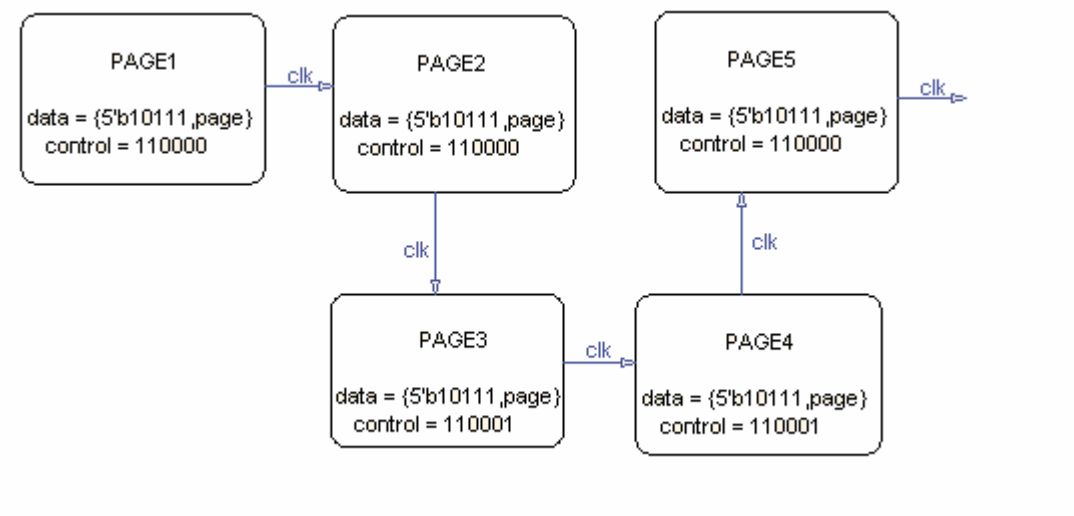


**Figure 10: Write LCD Page Information State Transition Diagram**

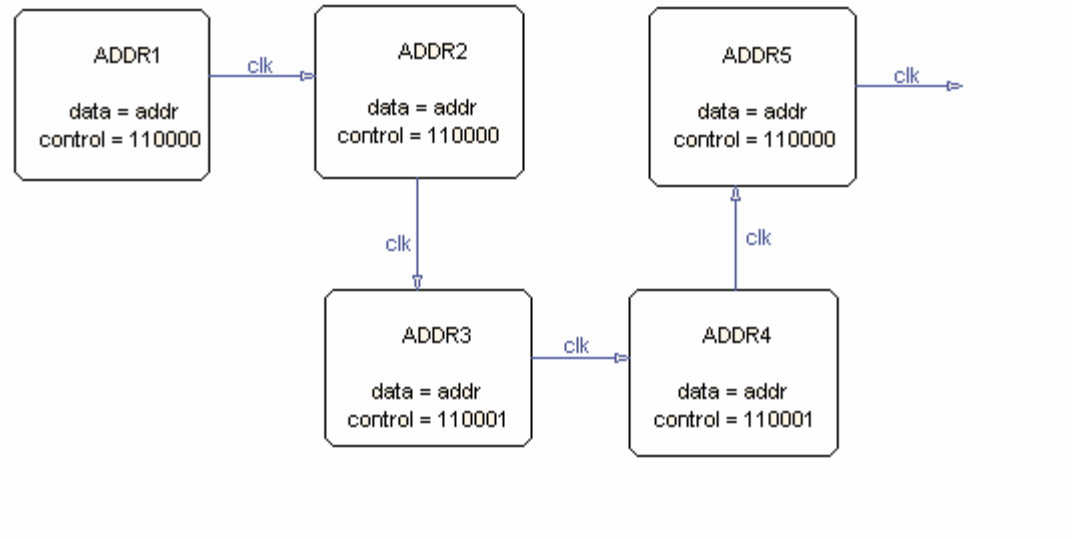


**Figure 11: Write LCD Addressing State Transition Diagram**

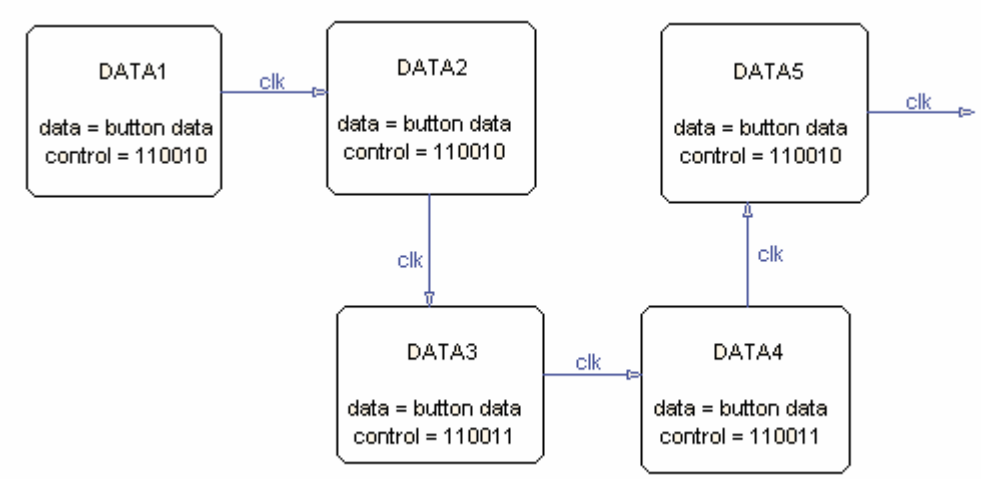


**Figure 12: Write LCD Button Data State Transition Diagram**

# Appendix B: Pin Assignments

## Controller FGPA

address[0] = P70
address[1] = P69
address[2] = P68
address[3] = P67
address[4] = P66
address[5] = P65
address[6] = P62
address[7] = P61
address[8] = P60
busy = P82
clk = P13
data_in = P9
dclk_in = P8
latch_in = P7
LCDcontrol[0] = P48
LCDcontrol[1] = P49
LCDcontrol[2] = P50
LCDcontrol[3] = P57
LCDcontrol[4] = P56
LCDcontrol[5] = P51
LCDdata[0] = P81
LCDdata[1] = P80
LCDdata[2] = P79
LCDdata[3] = P78
LCDdata[4] = P77

LCDdata[5] = P72
LCDdata[6] = P59
LCDdata[7] = P58
Play = P3
RAM_data[0] = P17
RAM_data[1] = P18
RAM_data[2] = P19
RAM_data[3] = P20
RAM_data[4] = P23
RAM_data[5] = P24
RAM_data[6] = P25
RAM_data[7] = P26
RAM_data[8] = P27
RAM_data[9] = P28
RAM_data[10] = P29
RAM_data[11] = P35
Reset = P45
startRecord = P4
stop = P5
write = P10

## RAM FPGA

address[0] = P77
address[1] = P10
address[2] = P9
address[3] = P8
address[4] = P7
address[5] = P6
address[6] = P5
address[7] = P4
address[8] = P3
data_in = P56
data_out[0] = P57
data_out[1] = P58
data_out[2] = P59
data_out[3] = P60
data_out[4] = P61
data_out[5] = P62
data_out[6] = P65
data_out[7] = P66
data_out[8] = P67
data_out[9] = P68
data_out[10] = P69
data_out[11] = P70
dclk = P51
write = P35

## Appendix C: Verilog Code

```
// main .v
// This is the main module of the Conrtrol FPGA

module
main(clk,reset,play_in,startRecord_in,stop_in,dclk_in,latch_in,data_in,address,
               RAM_data,write,busy,LCDdata,LCDcontrol,done,go,LCDstate,stat);
       input                   clk;
       input                   reset;
       input                   play_in;
       input                   startRecord_in;
       input                   stop_in;
       input                   dclk_in;
       input                   latch_in;
       input                   data_in;
       output [8:0]            address;
       input  [11:0]           RAM_data;
       output                  write;
       output                  busy;
       output [7:0]            LCDdata;
       output [5:0]            LCDcontrol;
       output                  done;
       output                  go;
       output [5:0]            LCDstate;
       output                  stat;

       wire                    play;
       wire                    startRecord;
       wire                    stop;
       wire                    dclkSync;
       wire                    latchSync;
       wire                    rec_complete;
       wire                    playcomplete;
       wire [8:0]              play_address;
       wire [8:0]              rec_address;
       wire [8:0]              RAM_address;
       wire [15:0]             rec_data;
       wire [15:0]             data_from_RAM;
       wire [11:0]             display_data;
       wire [15:0]             display_data_full;
       wire                    in_play;
       wire                    in_rec;
       wire                    free_or_play;

       reg [1:0]               state;
       reg [1:0]               nextstate;
       reg [8:0]               addrcount;

       parameter FREE   = 2'b00;
       parameter PLAY   = 2'b01;
       parameter RECORD = 2'b10;
       parameter ADDR_0 = 9'b0_0000_0000;

       // Sync the dclk
       syncFlopResetHi  dclkSF(clk,reset,dclk_in,dclkSync);
```

```
 // Sync the latch
 syncFlop   latchSF(clk,reset,latch_in,latchSync);

 // Choose the source of the RAM address
 mux2_9     addrmux(play_address,rec_address,state[1],RAM_address);

 // Choose the source of display information
 mux3_16    dispmux(data_from_RAM,rec_data,state,display_data_full);

 // Change data from 16 bits to 12 bits
 RAMpadding RAMpadding(RAM_data,data_from_RAM);

 // Playback module
 playback
playback(reset,latchSync,in_play,addrcount,play_address,playcomplete);

 // Record module
 record
record(reset,dclkSync,in_rec,data_in,stop,write,rec_data,rec_address,
            rec_complete);

 // Display module
 display    display(latchSync,reset,display_data_full,display_data);

 // LCD module
 WriteLCD
WriteLCD(clk,reset,display_data,latchSync,busy,LCDdata,LCDcontrol,done,go,
            LCDstate,stat);


// State Register

always@(posedge clk or posedge reset)
      if(reset)   state <= FREE;
      else              state <= nextstate;

// Next State Logic

always@(state or playcomplete or rec_complete or play or startRecord)
      case(state)
            FREE: if (play)               nextstate <= PLAY;
                  else if (startRecord)   nextstate <= RECORD;
                  else                    nextstate <= FREE;
            PLAY: if (playcomplete)       nextstate <= FREE;
                  else                    nextstate <= PLAY;
          RECORD: if (rec_complete)       nextstate <= FREE;
                  else                    nextstate <= RECORD;
            default:                      nextstate <= FREE;
      endcase

 // When the record module has finished, then store the last address

 always @(posedge rec_complete or posedge reset)
      if (reset)  addrcount   <= ADDR_0;
      else        addrcount   <= rec_address;

 // Output Logic
```

```
        assign in_play   = (state == PLAY);
        assign in_rec    = (state == RECORD);
        assign address   = RAM_address;
        assign play      = ~play_in;
        assign startRecord = ~startRecord_in;
        assign stop      = ~stop_in;

endmodule


// syncFlopResetHi.v
// Resetable synchronized flip-flop (reset hi)
// It is used to synchronize the data clock to the main clock

module syncFlopResetHi(clk,reset,d,q);
      input clk;
      input reset;
      input d;
      output q;

      wire a;
      wire HI;
      assign HI = 1;

      mux2 mux2(d,HI,reset,a);
      flop flop(clk,a,q);

endmodule



// sync.Flop.v
// Resetable synchronized flip-flop
// This is used to synchronize the latch

module syncFlop(clk,reset,d,q);
      input clk;
      input reset;
      input d;
      output q;

      wire a;
      wire LOW;
      assign LOW = 0;

      mux2 mux2(d,LOW,reset,a);
      flop flop(clk,a,q);

endmodule
```

```
// mux2.v
// This 2x1 mux gets used in the resetable flip-flops

module mux2(d0,d1,s,y);
      input d0;
      input d1;
      input s;
      output y;

      assign y = s ? d1 : d0; // if s=1, y=d1, else y=d0

endmodule
```

```
// flop.v
// The flip-flop is used in the resetable flip-flop

module flop(clk,d,q);
    input clk;
    input d;
    output q;

      reg q;

      always @(posedge clk) q <= d;

endmodule
```

```
// mux2_9.v
// The 2x9 mux is used in picking the RAM address source

module mux2_9(d0,d1,s,y);
      input [8:0] d0;
      input [8:0] d1;
      input s;
      output [8:0] y;

      assign y = s ? d1 : d0; // if s=1, y=d1, else y=d0

endmodule
```

```
// mux3_16.v
// The 3x16 mux is used in picking the source
// of the LCD display data

module mux3_16(d0,d1,s,y);
      input [15:0] d0;
      input [15:0] d1;
      input [1:0] s;
      output [15:0] y;

      wire [15:0] a;
      wire [15:0] b;

      parameter NODATA = 16'b1111_1111_1111_1111;

      assign a = s[0] ? d0 : NODATA;
      assign b = s[1] ? d1 : a;
      assign y = b;

endmodule



// playback.v
// The playback module gives the RAM address during play mode

module playback(reset,latch,play,addrcount,addr,complete);
      input reset;
      input latch;
      input play;
      input [8:0] addrcount;
      output [8:0] addr;
      output complete;

      reg [8:0]   addr;

      parameter   ZERO = 9'b0_0000_0000;

      always @(posedge latch or posedge reset) // Add after every latch
            if (reset)        addr <= ZERO;
            else if (~play)   addr <= ZERO;
            else              addr <= addr + 1;

      assign complete = (addr == addrcount + 1);  // Complete flag

endmodule
```

```
// record.v
// The record module gives the RAM address and write signalduring record mode,
// as well as the LCD display data.

module record(reset,dclk,record,data_in,stop,write,data_out,address,complete);
      input reset;
      input dclk;
      input record;
      input data_in;
      input stop;
      output write;
      output [15:0] data_out;
      output [8:0] address;
      output complete;


      FSM   FSM(reset,record,dclk,stop,write,address,complete);
      shift_reg   shift_reg(dclk,data_in,data_out);

endmodule



// FSM.v
// Finite state machine for record module

module FSM(reset,record,dclk,stop,write,addr,complete);
      input reset;
      input record;
      input dclk;
      input stop;
      output write;
      output [8:0] addr;
      output complete;

      wire        res;
      wire        invdclk;
      reg         write;
      reg [4:0]   state, nextstate;
      reg [8:0]   addr, nextaddr;
      reg         complete;
      wire        no_rec;

      parameter WAIT          =     5'b00000;
      parameter RES           =     5'b00001;  // State for 1st dclk
      parameter S2            =     5'b00010;
      parameter S3            =     5'b00011;
      parameter S4            =     5'b00100;
      parameter S5            =     5'b00101;
      parameter S6            =     5'b00110;
      parameter S7            =     5'b00111;
      parameter S8            =     5'b01000;
      parameter S9            =     5'b01001;
      parameter S10           =     5'b01010;
      parameter S11           =     5'b01011;
      parameter S12           =     5'b01100;
      parameter S13           =     5'b01101;
      parameter S14           =     5'b01110;
```

```
parameter S15           =     5'b01111;
parameter NEXT          =     5'b10000;  // State for 16th dclk
parameter ZERO          =     9'b0_0000_0000;  // Address 0
parameter END           =     9'b1_0010_1100;  // Final address

assign invdclk = ~dclk;
assign no_rec = ~record;


always @(posedge invdclk or posedge reset or posedge no_rec)
      if (reset|no_rec)
                  begin
                        state <= WAIT;
                        addr  <= ZERO;
                  end
      else  begin
                        state <= nextstate;
                        addr <= nextaddr;
                  end

always @(state)
      case (state)
            WAIT: begin  // Wait until record has been pressed
                              write       <= 0;
                              nextstate   <= RES;
                              nextaddr    <= addr;
                        end
             RES: begin
                              write       <= 0;
                              nextstate   <= S2;
                              nextaddr    <= addr;
                        end
              S2: begin
                              write       <= 0;
                              nextstate   <= S3;
                              nextaddr    <= addr;
                        end
              S3: begin
                              write       <= 0;
                              nextstate   <= S4;
                              nextaddr    <= addr;
                        end
              S4: begin
                              write       <= 0;
                              nextstate   <= S5;
                              nextaddr    <= addr;
                        end
              S5: begin
                              write       <= 0;
                              nextstate   <= S6;
                              nextaddr    <= addr;
                        end
              S6: begin
                              write       <= 0;
                              nextstate   <= S7;
                              nextaddr    <= addr;
                        end
              S7: begin
```

```
                        write       <= 0;
                        nextstate   <= S8;
                        nextaddr    <= addr;
                end
         S8: begin
                        write       <= 0;
                        nextstate   <= S9;
                        nextaddr    <= addr;
                end
         S9: begin
                        write       <= 0;
                        nextstate   <= S10;
                        nextaddr    <= addr;
                end
        S10: begin
                        write       <= 0;
                        nextstate   <= S11;
                        nextaddr    <= addr;
                end
        S11: begin
                        write       <= 0;
                        nextstate   <= S12;
                        nextaddr    <= addr;
                end
        S12: begin
                        write       <= 0;
                        nextstate   <= S13;
                        nextaddr    <= addr;
                end
        S13: begin
                        write       <= 0;
                        nextstate   <= S14;
                        nextaddr    <= addr;
                end
        S14: begin
                        write       <= 0;
                        nextstate   <= S15;
                        nextaddr    <= addr;
                end
        S15: begin
                        write       <= 0;
                        nextstate   <= NEXT;
                        nextaddr    <= addr;
                end
       NEXT: begin
                        write       <= 1;  // Write after last clock
                        nextstate   <= RES;
                        nextaddr    <= addr +1;
                end
      default:    begin
                              write       <= 0;
                              nextstate   <= WAIT;
                              nextaddr    <= ZERO;
                        end
endcase
```

```
      // Complete Flag logic

      always @(posedge dclk or posedge stop)
            if (stop)              complete <= 1;
            else if (addr == END)  complete <= 1;
            else                   complete <= 0;

endmodule



// shift_reg.v
// The shift register is used by both the record
// module and the RAM to take in the serial data.

module shift_reg(clk,data,dataout);
       input clk;
       input data;
       output [15:0] dataout;

       reg [15:0] shiftreg;  // 16-bit shift register

       always @(negedge clk)
       shiftreg <= {shiftreg[14:0], data};

       assign dataout = shiftreg;

endmodule



// display.v
// The display module prepares the LCD display data

module display(clk,reset,data_in,data_out);
    input         clk;
    input         reset;
    input  [15:0] data_in;
    output [11:0] data_out;

       reg [11:0] data_out;

       parameter NODATA = 12'b1111_1111_1111;  // represents no buttons

       always @(posedge clk or posedge reset)
            if (reset)  data_out <= NODATA;
            else        data_out <= data_in[15:4];

endmodule
```

```
// WriteLCD.v
// The Write LCD module creates the commands to
// control the graphical LCD

module
WriteLCD(clk,reset,bdata,latch,busy,LCDdata,LCDcontrol,done,go,state,stat);
	input		clk;
	input		reset;
	input [11:0]	bdata;
	input		latch;
	output		busy;
	output [7:0]	LCDdata;
	output [5:0]	LCDcontrol;
	output		done;
	output		go;
	output [5:0]	state;
	output		stat;

	reg	go;
	reg [4:0]  button;
	reg [5:0]  state, nextstate;

	// Module to control LCD write process for each button
	lcdFSM	lcdFSM(clk,reset,go,button,busy,LCDdata,LCDcontrol,done,stat);

	// Button encodings (chosen to make buttons on chip 1 of LCD easily
referenced (button[3]))
	parameter NONE		= 5'b00000;
	parameter B		= 5'b00001;
	parameter Y		= 5'b00010;
	parameter START		= 5'b00011;
	parameter A		= 5'b00100;
	parameter X		= 5'b00101;
	parameter R		= 5'b00110;
	parameter INITIALIZE2	= 5'b00111;
	parameter SEL		= 5'b01000;
	parameter UP		= 5'b01001;
	parameter DOWN		= 5'b01010;
	parameter LEFT		= 5'b01011;
	parameter RIGHT		= 5'b01100;
	parameter L		= 5'b01101;
	parameter INITIALIZE1	= 5'b01110;
	parameter CLEARB	= 5'b10001;
	parameter CLEARY	= 5'b10010;
	parameter CLEARSTART	= 5'b10011;
	parameter CLEARA	= 5'b10100;
	parameter CLEARX	= 5'b10101;
	parameter CLEARR	= 5'b10110;
	parameter CLEARSEL	= 5'b11000;
	parameter CLEARUP	= 5'b11001;
	parameter CLEARDOWN	= 5'b11010;
	parameter CLEARLEFT	= 5'b11011;
	parameter CLEARRIGHT	= 5'b11100;
	parameter CLEARL	= 5'b11101;
	parameter DELAY		= 5'b11110;

	// State encodings
	parameter S0		= 6'b000000;
```

```
parameter S1      = 6'b000001;
parameter S2      = 6'b000010;
parameter S3      = 6'b000011;
parameter S4      = 6'b000100;
parameter S5      = 6'b000101;
parameter S6      = 6'b000110;
parameter S7      = 6'b000111;
parameter S8      = 6'b001000;
parameter S9      = 6'b001001;
parameter S10     = 6'b001010;
parameter S11     = 6'b001011;
parameter S12     = 6'b001100;
parameter ON      = 6'b001111; // Used to turn LCD on
parameter CLEAR1  = 6'b010001; // CLEAR states used to clear each button
parameter CLEAR2  = 6'b010010; // press after writing to LCD. At first,
parameter CLEAR3  = 6'b010011; // the entire LCD screen was cleared at
parameter CLEAR4  = 6'b010100; // each latch edge. But because of
parameter CLEAR5  = 6'b010101; // how long that took, it was noticeable.
parameter CLEAR6  = 6'b010110; // Instead, each button display is cleared
parameter CLEAR7  = 6'b010111; // on the LCD immediately after it is
parameter CLEAR8  = 6'b011000; // written, which is much quicker because
parameter CLEAR9  = 6'b011001; // only small portions of the LCD must be
parameter CLEAR10 = 6'b011010; // cleared.
parameter CLEAR11 = 6'b011011;
parameter CLEAR12 = 6'b011100;
parameter CLEAR13 = 6'b011101;
parameter DELAY1  = 6'b011110;
parameter DELAY2  = 6'b011111;
parameter INIT0   = 6'b100000;
parameter INIT1   = 6'b100001; // Used to clear both sides of the LCD at
parameter INIT2   = 6'b100010; // startup (LCD initializes with every
                               // pixel darkened).

// State Register
always @(posedge clk or posedge reset)
     if (reset)  state <= ON;
     else              state <= nextstate;

// Next State Logic
always @(state or latch or done or bdata)
     case (state)
           ON:   begin
                                    nextstate   <= INIT0;
                                    button      <= NONE;
                                    go          <= 0;
                      end
           INIT0: if(latch)
                           begin
                                    nextstate    <= INIT1;
                                    button       <= NONE;
                                    go           <= 0;
                           end
                       else
                           begin
                                    nextstate    <= INIT0;
                                    button       <= NONE;
                                    go           <= 0;
                           end
```

```
INIT1: if (done)  // Clear left half of LCD
                  begin
                        nextstate    <= INIT2;
                        button       <= NONE;
                        go           <= 0;
                  end
           else
                  begin
                        nextstate    <= INIT1;
                        button       <= INITIALIZE1;
                        go           <= 1;
                  end
INIT2: if (done)  // Clear right half of LCD
                  begin
                        nextstate    <= S0;
                        button       <= NONE;
                        go           <= 0;
                  end
              else
                  begin
                        nextstate    <= INIT2;
                        button       <= INITIALIZE2;
                        go           <= 1;
                  end
S0: if (latch)    // Wait for latch before clearing LCD
            begin
                  nextstate    <= CLEAR1;
                  button       <= NONE;
                  go           <= 0;
            end
       else
            begin
                  nextstate    <= S0;
                  button       <= NONE;
                  go           <= 0;
            end
S1: if (done|bdata[11]) // Go to next state if button has been
            begin       // displayed or button is not pressed.
                  Nextstate    <= S2;
                  button       <= NONE;
                  go           <= 0;
            end
       else

            begin
                  nextstate    <= S1;
                  button       <= B;
                  go           <= 1;
          end
CLEAR1: if (done)
                   begin
                        nextstate    <= CLEAR2;
                        button       <= NONE;
                        go           <= 0;
                   end
              else
                   begin
                        nextstate    <= CLEAR1;
```

```
                              button       <= CLEARB;
                              go           <= 1;
                         end
        S2: if (done|bdata[10]) // Go to next state if button has been
                    begin       // displayed or button is not pressed.
                          Nextstate    <= S3;
                          button       <= NONE;
                          go           <= 0;
                    end
                else
                    begin
                          nextstate    <= S2;
                          button       <= Y;
                          go           <= 1;
                  end
        CLEAR2: if (done)
                         begin
                              nextstate    <= CLEAR3;
                              button       <= NONE;
                              go           <= 0;
                         end
                       else
                              begin
                              nextstate    <= CLEAR2;
                              button       <= CLEARY;
                              go           <= 1;
                         end
        S3: if (done|bdata[9])  // Go to next state if button has been
                    begin       // displayed or button is not pressed.
                          nextstate    <= S4;
                          button       <= NONE;
                          go           <= 0;
                    end
                else
                    begin
                          nextstate    <= S3;
                          button       <= SEL;
                          go           <= 1;
                    end
        CLEAR3: if (done)
                         begin
                              nextstate    <= CLEAR4;
                              button       <= NONE;
                              go           <= 0;
                         end
                       else
                              begin
                              nextstate    <= CLEAR3;
                              button       <= CLEARSEL;
                              go           <= 1;
                         end
     S4: if (done|bdata[8])     // Go to next state if button has been
                    begin
  // displayed or button is not pressed.
                         Nextstate    <= S5;
                         button       <= NONE;
                         go           <= 0;
                    end
```

```
            else
                 begin
                       nextstate    <= S4;
                       button       <= START;
                       go           <= 1;
                 end
   CLEAR4: if (done)
                        begin
                             nextstate    <= CLEAR5;
                             button       <= NONE;
                             go           <= 0;
                        end
                   else
                        begin
                             nextstate    <= CLEAR4;
                             button       <= CLEARSTART;
                             go                <= 1;
                        end
   S5: if (done|bdata[7])  // Go to next state if button has been
                 begin     // displayed or button is not pressed.
                       Nextstate    <= S6;
                       button       <= NONE;
                       go           <= 0;
                 end
            else
                 begin
                       nextstate    <= S5;
                       button       <= UP;
                       go           <= 1;
                 end
   CLEAR5: if (done)
                        begin
                             nextstate    <= CLEAR6;
                             button       <= NONE;
                             go           <= 0;
                        end
                   else
                        begin
                             nextstate    <= CLEAR5;
                             button       <= CLEARUP;
                             go                <= 1;
                        end
   S6: if (done|bdata[6])  // Go to next state if button has been
                 begin     // displayed or button is not pressed.
                       Nextstate    <= S7;
                       button       <= NONE;
                       go           <= 0;
                 end
            else
                 begin
                       nextstate <= S6;
                       button       <= DOWN;
                       go           <= 1;
                 end
CLEAR6: if (done)
                        begin
                             nextstate    <= CLEAR7;
                             button       <= NONE;
```

```
                    go          <= 0;
                end
            else
                begin
                    nextstate   <= CLEAR6;
                    button      <= CLEARDOWN;
                    go          <= 1;
                end
S7: if (done|bdata[5])
                // Go to the next state if button has been
          begin // displayed or button is not pressed.
                Nextstate   <= S8;
                button      <= NONE;
                go          <= 0;
          end
     else
          begin
                nextstate <= S7;
                button      <= LEFT;
                go          <= 1;
          end
CLEAR7: if (done)
                begin
                    nextstate   <= CLEAR8;
                    button      <= NONE;
                    go          <= 0;
                end
            else
                begin
                    nextstate   <= CLEAR7;
                    button      <= CLEARLEFT;
                    go          <= 1;
                end
S8: if (done|bdata[4])  // Go to next state if button has been
          begin       // displayed or button is not pressed.
                Nextstate   <= S9;
                button      <= NONE;
                go          <= 0;
          end
     else
          begin
                nextstate   <= S8;
                button      <= RIGHT;
                go          <= 1;
          end
CLEAR8: if (done)
                begin
                    nextstate   <= CLEAR9;
                    button      <= NONE;
                    go          <= 0;
                end
            else
                begin
                    nextstate   <= CLEAR8;
                    button      <= CLEARRIGHT;
                    go          <= 1;
                end
S9: if (done|bdata[3])  // Go to next state if button has been
```

```
            begin       // displayed or button is not pressed.
                  Nextstate    <= S10;
                  button       <= NONE;
                  go           <= 0;
            end
       else
            begin
                  nextstate    <= S9;
                  button       <= A;
                  go           <= 1;
            end
CLEAR9: if (done)
                   begin
                        nextstate    <= CLEAR10;
                        button       <= NONE;
                        go           <= 0;
                   end
              else
                   begin
                        nextstate    <= CLEAR9;
                        button       <= CLEARA;
                        go           <= 1;
                   end
S10: if (done|bdata[2]) // Go to next state if button has been
            begin       // displayed or button is not pressed.
                  Nextstate    <= S11;
                  button       <= NONE;
                  go           <= 0;
            end
       else
            begin
                  nextstate    <= S10;
                  button       <= X;
                  go           <= 1;
            end
CLEAR10: if (done)
                   begin
                        nextstate    <= CLEAR11;
                        button       <= NONE;
                        go           <= 0;
                   end
              else
                   begin
                        nextstate    <= CLEAR10;
                        button       <= CLEARX;
                        go           <= 1;
                   end
S11: if (done|bdata[1]) // Go to next state if button has been
            begin       // displayed or button is not pressed.
                  Nextstate    <= S12;
                  button       <= NONE;
                  go           <= 0;
            end
       else
            begin
                  nextstate    <= S11;
                  button       <= L;
                  go           <= 1;
```

```
            end
CLEAR11: if (done)
                  begin
                       nextstate    <= CLEAR12;
                       button       <= NONE;
                       go           <= 0;
                  end
               else
                  begin
                       nextstate    <= CLEAR11;
                       button       <= CLEARL;
                       go           <= 1;
                  end
S12: if (done|bdata[0]) // Go to next state if button has been
            begin       // displayed or button is not pressed.
                  nextstate    <= S0;
                  button       <= NONE;
                  go           <= 0;
            end
       else
            begin
                  nextstate    <= S12;
                  button       <= R;
                  go           <= 1;
            end
CLEAR12: if (done)
                  begin
                       nextstate    <= S1;
                       button       <= NONE;
                       go           <= 0;
                  end
                else
                  begin
                       nextstate    <= CLEAR12;
                       button       <= CLEARR;
                       go           <= 1;
                  end
DELAY1: if (done)
                  begin
                        nextstate   <= DELAY2;
                        button      <= NONE;
                        go          <= 0;
                  end
               else
                  begin
                        nextstate   <= DELAY;
                        button      <= DELAY;
                        go    <= 1;
                  end
DELAY2: if(done)
                  begin
                       nextstate    <= CLEAR1;
                       button       <= NONE;
                       go           <= 0;
                  end
               else
                  begin
                       nextstate    <= DELAY2;
```

```
                                        button       <= DELAY;
                                        go           <= 1;
                                    end
                    default:    begin
                                        nextstate    <= S0;
                                        button       <= NONE;
                                        go           <= 0;
                                    end
            endcase


endmodule



// lcdFSM.v
// This FSM produces the so-called sub-routines of
// the LCD display commands.

module lcdFSM(clk,reset,go,button,busy,LCDdata,LCDcontrol,done,stat);
        input clk;
        input reset;
        input go;
        input [4:0] button;
        input busy;
        output [7:0] LCDdata;
        output [5:0] LCDcontrol;
        output done;
        output stat;

        wire drive;

        reg [7:0] data, nextdata;
        reg [5:0] control, nextcontrol;
        reg [5:0] state, nextstate;
        reg [7:0] addr, nextaddr;    // Keeps track of which column is written
        reg [2:0] page, nextpage;    // Keeps track of which page is written

        // Button encodings (first side of of LCD easily referenced (button[3]))
        parameter NONE           = 5'b00000;
        parameter B               = 5'b00001;
        parameter Y              = 5'b00010;
        parameter START          = 5'b00011;
        parameter A              = 5'b00100;
        parameter X              = 5'b00101;
        parameter R              = 5'b00110;
        parameter INITIALIZE2    = 5'b00111;
        parameter SEL            = 5'b01000;
        parameter UP             = 5'b01001;
        parameter DOWN           = 5'b01010;
        parameter LEFT           = 5'b01011;
        parameter RIGHT          = 5'b01100;
        parameter L              = 5'b01101;
        parameter INITIALIZE1    = 5'b01110;
        parameter CLEARB         = 5'b10001;
        parameter CLEARY         = 5'b10010;
        parameter CLEARSTART     = 5'b10011;
        parameter CLEARA         = 5'b10100;
```

```
parameter CLEARX         = 5'b10101;
parameter CLEARR         = 5'b10110;
parameter CLEARSEL       = 5'b11000;
parameter CLEARUP        = 5'b11001;
parameter CLEARDOWN      = 5'b11010;
parameter CLEARLEFT      = 5'b11011;
parameter CLEARRIGHT     = 5'b11100;
parameter CLEARL         = 5'b11101;
parameter DELAY          = 5'b11110;

// LCD control configurations
parameter STATUSCHK1    = 6'b110100;
parameter STATUSCHK2    = 6'b101100;
parameter SETPAGE1      = 6'b110000;
parameter SETPAGE2      = 6'b101000;
parameter SETADDR1      = 6'b110000;
parameter SETADDR2      = 6'b101000;
parameter SETDATA1      = 6'b110010;
parameter SETDATA2      = 6'b101010;
parameter LCDON         = 6'b111000;

// LCD display data for each button
parameter BDATA         = 8'b1111_1111;
parameter YDATA         = 8'b1111_1111;
parameter SELDATA       = 8'b1111_1111;
parameter STARTDATA     = 8'b1111_1111;
parameter UPDATA        = 8'b1111_1111;
parameter DOWNDATA      = 8'b1111_1111;
parameter LEFTDATA      = 8'b1111_1111;
parameter RIGHTDATA     = 8'b1111_1111;
parameter ADATA         = 8'b1111_1111;
parameter XDATA         = 8'b1111_1111;
parameter LDATA         = 8'b1111_1111;
parameter RDATA         = 8'b1111_1111;
parameter CLEARDATA     = 8'b0000_0000;

// LCD initial address (column) data for each button
parameter BINIT                   = 8'b0110_0000;
parameter YINIT                   = 8'b0101_0000;
parameter SELINIT                 = 8'b0111_1000;
parameter STARTINIT               = 8'b0111_0000;
parameter UPINIT                  = 8'b0110_0000;
parameter DOWNINIT                = 8'b0110_0000;
parameter LEFTINIT                = 8'b0101_1000;
parameter RIGHTINIT               = 8'b0110_1000;
parameter AINIT                   = 8'b0111_0000;
parameter XINIT                   = 8'b0110_0000;
parameter LINIT                   = 8'b0101_1000;
parameter RINIT                   = 8'b0101_1000;
parameter INITIALIZEINIT          = 8'b0100_0000;
parameter DELAYINIT               = 8'b0100_0000;

// LCD final address (column) data for each button
parameter BFIN                    = 8'b0110_0111;
parameter YFIN                    = 8'b0101_0111;
parameter SELFIN                  = 8'b0111_1111;
parameter STARTFIN                = 8'b0111_0111;
parameter UPFIN                   = 8'b0110_0111;
```

```
parameter DOWNFIN                = 8'b0110_0111;
parameter LEFTFIN                = 8'b0101_1111;
parameter RIGHTFIN               = 8'b0110_1111;
parameter AFIN                   = 8'b0111_0111;
parameter XFIN                   = 8'b0110_0111;
parameter LFIN                   = 8'b0110_1111;
parameter RFIN                   = 8'b0110_1111;
parameter INITIALIZEFIN          = 8'b0111_1111;
parameter DELAYFIN               = 8'b0111_1111;

// LCD page (row) data for each button
parameter BPAGE        = 3'b101;
parameter YPAGE        = 3'b100;
parameter SELPAGE      = 3'b110;
parameter STARTPAGE    = 3'b110;
parameter UPPAGE       = 3'b011;
parameter DOWNPAGE     = 3'b101;
parameter LEFTPAGE     = 3'b100;
parameter RIGHTPAGE    = 3'b100;
parameter APAGE        = 3'b100;
parameter XPAGE        = 3'b011;
parameter LPAGE        = 3'b001;
parameter RPAGE        = 3'b001;
parameter DELAYPAGE    = 3'b111;

// LCD page encoding for initialization sequence
parameter INITPAGE = 3'b000;
parameter FINPAGE  = 3'b111;

// FSM State Definitions
parameter INIT      = 6'b000000;
parameter PAGE1     = 6'b000001;
parameter PAGE2     = 6'b000010;
parameter PAGE3     = 6'b000011;
parameter PAGE4     = 6'b000100;
parameter PAGE5     = 6'b000101;
parameter ADDR1     = 6'b000110;
parameter ADDR2     = 6'b000111;
parameter ADDR3     = 6'b001000;
parameter ADDR4     = 6'b001001;
parameter ADDR5     = 6'b001010;
parameter DATA1     = 6'b001011;
parameter DATA2     = 6'b001100;
parameter DATA3     = 6'b001101;
parameter DATA4     = 6'b001110;
parameter DATA5     = 6'b001111;
parameter ADDRCHK   = 6'b010000;
parameter ADDRINCR  = 6'b010001;
parameter PAGECHK   = 6'b010010;
parameter PAGEINCR  = 6'b010011;
parameter DONE      = 6'b010100;
parameter ON1       = 6'b010101;
parameter ON2       = 6'b010110;
parameter ON3       = 6'b010111;
parameter STATCHK1  = 6'b100001;   // Note: State encodings chosen
parameter STATCHK2  = 6'b100010;   // to make status check states
parameter STATCHK3  = 6'b100011;   // easily identifiable (state[5])
parameter STATCHK4  = 6'b100100;
```

```
       parameter STATCHK5  = 6'b100101;
       parameter STATCHK6  = 6'b100110;
       parameter STATCHK7  = 6'b100111;
       parameter STATCHK8  = 6'b101000;
       parameter STATCHK9  = 6'b101001;
       parameter STATCHK10 = 6'b101010;
       parameter STATCHK11 = 6'b101011;
       parameter STATCHK12 = 6'b101100;
       parameter STATCHK13 = 6'b101101;
       parameter STATCHK14 = 6'b101110;
       parameter STATCHK15 = 6'b101111;
       parameter STATCHK16 = 6'b110000;
       parameter STATCHK17 = 6'b110001;
       parameter STATCHK18 = 6'b110010;
       parameter STATCHK19 = 6'b110011;

       // State Register

       always @(posedge clk or posedge reset)
            if (reset)
                  begin
                        state       <= ON1;
                        data        <= 8'b0011_1111;
                        control     <= 6'b011000;
                        addr        <= INITIALIZEINIT;
                        page        <= INITPAGE;
                  end
            else
                  begin
                        state       <= nextstate;
                        control     <= nextcontrol;
                        addr        <= nextaddr;
                        data        <= nextdata;
                        page        <= nextpage;
                  end

       // Next State Logic

       always @(state or go or button or data or control or addr or page or
busy)
            case (state)
                  ON1:  begin
      // Delay state
                                    nextstate   <= ON2;
                                    nextdata    <= data;
                                    nextcontrol <= 6'b111001;
                                    nextaddr    <= addr;
                                    nextpage    <= page;
                              end

                  ON2: begin
      // Turn LCD display on
                                    nextstate   <= ON3;
                                    nextdata    <= data;
                                    nextcontrol <= 6'b111001;// Switch E high
                                    nextaddr    <= addr;
                                    nextpage    <= page;
                        end
```

```
            ON3: begin
// Delay state
                              nextstate   <= INIT;
                              nextdata    <= data;
                              nextcontrol <= LCDON;
                              nextaddr    <= addr;
                              nextpage    <= page;
                        end

            INIT: if (go)
                              begin
                                    nextstate   <= STATCHK1;
                                    nextdata    <= data;

                                 if(button[3])
                                          nextcontrol <= STATUSCHK1;
                                    else
                                          nextcontrol <= STATUSCHK2;

                                    case (button)
                                          B:                begin
                                                nextaddr <= BINIT;
                                                nextpage <= BPAGE;
                                                                  end
                                          CLEARB:           begin
                                                nextaddr <= BINIT;
                                                nextpage <= BPAGE;
                                                                  end
                                          Y:                begin
                                                nextaddr <= YINIT;
                                                nextpage <= YPAGE;
                                                                  end
                                          CLEARY:           begin
                                                nextaddr <= YINIT;
                                                nextpage <= YPAGE;
                                                                  end
                                          SEL:              begin
                                                nextaddr <= SELINIT;
                                                nextpage <= SELPAGE;
                                                                  end
                                          CLEARSEL:   begin
                                          nextaddr <= SELINIT;
                                          nextpage <= SELPAGE;
                                                                  end
                                          START:            begin
                                          nextaddr <= STARTINIT;
                                          nextpage <= STARTPAGE;
                                                                  end
                                          CLEARSTART: begin
                                          extaddr <= STARTINIT;
                                          nextpage <= STARTPAGE;
                                                      end
                                          UP:               begin
                                          nextaddr <= UPINIT;
                                          nextpage <= UPPAGE;
                        end
                                          CLEARUP:          begin
```

```
                nextaddr <= UPINIT;
                nextpage <= UPPAGE;
                                        end
                DOWN:             begin
                nextaddr <= DOWNINIT;
                nextpage <= DOWNPAGE;
                                        end
                CLEARDOWN:  begin
                nextaddr <= DOWNINIT;
                nextpage <= DOWNPAGE;
                                        end
                LEFT:             begin
                nextaddr <= LEFTINIT;
                nextpage <= LEFTPAGE;
                                        end
                CLEARLEFT:  begin
                nextaddr <= LEFTINIT;
                nextpage <= LEFTPAGE;
                                        end
                RIGHT:            begin

                nextaddr <= RIGHTINIT;
                nextpage <= RIGHTPAGE;
                                        end
                CLEARRIGHT: begin
                nextaddr <= RIGHTINIT;
                nextpage <= RIGHTPAGE;
                                        end
                A:                begin
                nextaddr <= AINIT;
                nextpage <= APAGE;
                                        end
                CLEARA:           begin
                      nextaddr <= AINIT;
                      nextpage <= APAGE;
                                        end
                X:                begin
                      nextaddr <= XINIT;
                      nextpage <= XPAGE;
                                        end
                CLEARX:           begin
                      nextaddr <= XINIT;
                      nextpage <= XPAGE;
                                        end
                L:                begin
                      nextaddr <= LINIT;
                      nextpage <= LPAGE;
                                        end
                CLEARL:           begin
                      nextaddr <= LINIT;
                      nextpage <= LPAGE;
                                        end
                R:                begin
                      nextaddr <= RINIT;
                      nextpage <= RPAGE;
                                         end
              CLEARR:        begin
                      nextaddr <= RINIT;
```

```
                                    nextpage <= RPAGE;
                                                end
                                    INITIALIZE1: begin
                                    nextaddr <= INITIALIZEINIT;
                                    nextpage <= INITPAGE;
                                                end
                                    INITIALIZE2: begin
                                    nextaddr <= INITIALIZEINIT;
                                    nextpage <= INITPAGE;
                                                end
                                    DELAY:              begin
                                          nextaddr <= DELAYINIT;
                                          nextpage <= DELAYPAGE;
                                                end
                                    default:     begin
                                    nextaddr <= INITIALIZEINIT;
                                    nextpage <= INITPAGE;
                                                end
                              endcase
                        end
                  else
                        begin
                              nextstate   <= INIT;
                              nextdata    <= 8'b0011_1111;
                              nextcontrol <= STATUSCHK1;
                              nextaddr    <= INITIALIZEINIT;
                              nextpage    <= INITPAGE;
                        end

      STATCHK1: begin
                              nextstate   <= STATCHK2;
                              nextdata    <= data;
                              nextaddr    <= addr;
                              nextpage    <= page;
                              nextcontrol <= control;
                         end

      STATCHK2: begin
                              nextstate   <= STATCHK3;
                              nextdata    <= data;
                              nextaddr    <= addr;
                              nextpage    <= page;

                              if(button[3])
                                    nextcontrol <= 6'b110101;
                              else
                                    nextcontrol <= 6'b101101;
                         end

      STATCHK3: begin
      // Used as a delay state to insure
                              nextstate   <= STATCHK4;
// LCD setup and hold times are not
                              nextdata    <= data;
      // violated
                              nextcontrol <= control;
                              nextaddr    <= addr;
                              nextpage    <= page;
```

```
                    end

STATCHK4: begin
                        nextstate   <= STATCHK5;
                        nextdata    <= data;
                        nextcontrol <= control;
                        nextaddr    <= addr;
                        nextpage    <= page;
                    end

STATCHK5: if(~busy) // Proceed with write if LCD is not busy
                        begin
                              nextstate   <= STATCHK6;
                              nextdata    <= data;
                              nextcontrol <= control;
                              nextaddr    <= addr;
                              nextpage    <= page;
                        end
                    else       // Repeat status check if LCD
                        begin // returns busy flag
                              nextstate   <= STATCHK1;
                        nextdata    <= data;
                              nextaddr    <= addr;
                              nextpage          <= page;

                              if(button[3])
                                    nextcontrol <= STATUSCHK1;
                              else
                                    nextcontrol <= STATUSCHK2;
                      end

STATCHK6: begin
                        nextstate   <= ADDR1;
                        nextdata    <= data;
                        nextaddr    <= addr;
                        nextpage    <= page;
                        nextcontrol <= control;
                    end

ADDR1: begin
                  nextstate   <= ADDR2;
                  nextdata    <= addr;
                  nextaddr    <= addr;
                  nextpage    <= page;

                  if(button[3])
                        nextcontrol <= SETADDR1;
                  else
                        nextcontrol <= SETADDR2;
              end

ADDR2: begin
                  nextstate   <= ADDR3;
                  nextdata    <= data;
                  nextaddr    <= addr;
                  nextcontrol <= control;
                  nextpage    <= page;
              end
```

```
ADDR3: begin
                nextstate   <= ADDR4;
                nextdata    <= data;
                nextaddr    <= addr;
                nextpage    <= page;

                if(button[3])
                      nextcontrol <= 6'b110001;
                else
                      nextcontrol <= 6'b101001;
            end

ADDR4: begin
                nextstate   <= ADDR5;
                nextdata    <= data;
                nextaddr    <= addr;
                nextpage    <= page;
                nextcontrol <= control;
            end

ADDR5: begin
                nextstate   <= STATCHK7;
                nextdata    <= data;
                nextaddr    <= addr;
                nextpage    <= page;

                if(button[3])
                      nextcontrol <= SETADDR1;
                else
                      nextcontrol <= SETADDR2;
            end

STATCHK7: begin
                      nextstate   <= STATCHK8;
                      nextdata    <= data;
                      nextaddr    <= addr;
                      nextpage    <= page;

                      if(button[3])
                            nextcontrol <= STATUSCHK1;
                      else
                            nextcontrol <= STATUSCHK2;
                 end

STATCHK8: begin
                      nextstate   <= STATCHK9;
                      nextdata    <= data;
                      nextaddr    <= addr;
                      nextpage    <= page;

                      if(button[3])
                            nextcontrol <= 6'b110101;
                      else
                            nextcontrol <= 6'b101101;
                 end
```

```
                STATCHK9: begin
                // Used as a delay state to insure
                                        nextstate   <= STATCHK10;
          // LCD setup and hold times are not
                                        nextdata    <= data;
                // violated
                                        nextcontrol <= control;
                                        nextaddr    <= addr;
                                        nextpage    <= page;
                                    end

                STATCHK10: begin
                                        nextstate   <= STATCHK11;
                                        nextdata    <= data;
                                        nextcontrol <= control;
                                        nextaddr    <= addr;
                                        nextpage    <= page;
                                     end

                STATCHK11: if(~busy)
                                        begin
                                              nextstate   <= STATCHK12;
    //Proceed with write if LCD is not busy
                                              nextdata    <= data;
                                              nextcontrol <= control;
                                              nextaddr    <= addr;
                                              nextpage    = page;
                                        end
                                    else
                                        begin
                                              nextstate   <= STATCHK7;
//Repeat status check if LCD returns busy flag
                                              nextdata    <= data;
                                              nextaddr    <= addr;
                                              nextpage    <= page;

                                              if(button[3])
                                                    nextcontrol <= STATUSCHK1;
                                              else
                                                    nextcontrol <= STATUSCHK2;
                                      end

                STATCHK12: begin
                                        nextstate   <= PAGE1;
                                        nextdata    <= data;
                                        nextaddr    <= addr;
                                        nextpage    <= page;
                                        nextcontrol <= control;
                                     end

                PAGE1: begin
                                   nextstate   <= PAGE2;
                                   nextaddr    <= addr;
                                   nextdata    <= {5'b10111, page};
                                nextpage       <= page;

                                   if(button[3])
                                        nextcontrol <= SETPAGE1;
```

```
                    else
                          nextcontrol <= SETPAGE2;
               end

    PAGE2: begin
                    nextstate   <= PAGE3;
                    nextdata    <= data;
                    nextcontrol <= control;
                    nextaddr    <= addr;
                    nextpage    <= page;
               end

    PAGE3: begin
                    nextstate   <= PAGE4;
                    nextdata    <= data;
                    nextaddr    <= addr;
                    nextpage    <= page;

                    if(button[3])
                          nextcontrol <= 6'b101001;
                    else
                          nextcontrol <= 6'b110001;
               end

    PAGE4: begin
                    nextstate   <= PAGE5;
                    nextdata    <= data;
                    nextcontrol <= control;
                    nextaddr    <= addr;
                    nextpage    <= page;
               end

    PAGE5: begin
                    nextstate   <= STATCHK13;
                    nextdata    <= data;
                    nextaddr    <= addr;
                    nextpage    <= page;

                    if(button[3])
                          nextcontrol <= SETPAGE1;
                    else
                          nextcontrol <= SETPAGE2;
               end

    STATCHK13: begin
                          nextstate   <= STATCHK14;
                          nextdata    <= data;
                          nextaddr    <= addr;
                          nextpage    <= page;

                          if(button[3])
                                nextcontrol <= STATUSCHK1;
                          else
                                nextcontrol <= STATUSCHK2;
                     end

STATCHK14: begin
                          nextstate   <= STATCHK15;
```

```
                                    nextdata    <= data;
                                    nextaddr    <= addr;
                                    nextpage    <= page;
                                    nextcontrol <= control;
                                end

            STATCHK15: begin
                                    nextstate   <= STATCHK16;
                                    nextdata    <= data;
                                    nextaddr    <= addr;
                                    nextpage    <= page;

                                    if(button[3])
                                          nextcontrol <= 6'b110101;
                                    else
                                          nextcontrol <= 6'b101101;
                               end

            STATCHK16: begin
            // Used as a delay state to insure
                                    nextstate   <= STATCHK17;
      // LCD setup and hold times are not
                                    nextdata    <= data;
            // violated
                                    nextcontrol <= control;
                                    nextaddr    <= addr;
                                    nextpage    <= page;
                          end

            STATCHK17: begin
                                    nextstate   <= STATCHK18;
                                    nextdata    <= data;
                                    nextcontrol <= control;
                                    nextaddr    <= addr;
                                    nextpage    <= page;
                                end

            STATCHK18: if(~busy)
                                    begin
                                          nextstate   <= STATCHK19;
//Proceed with write if LCD is not busy
                                          nextdata    <= data;
                                          nextcontrol <= control;
                                          nextaddr    <= addr;
                                          nextpage    <= page;
                                    end
                               else
                                    begin
                                          nextstate   <= STATCHK13;
//Repeat status check if LCD returns busy flag
                                          nextdata    <= data;
                                          nextaddr    <= addr;
                                          nextpage    <= page;

                                          if(button[3])
                                                nextcontrol <= STATUSCHK1;
                                          else
                                                nextcontrol <= STATUSCHK2;
```

```
                    end

STATCHK19: begin
                        nextstate   <= DATA1;
                        nextdata    <= data;
                        nextaddr    <= addr;
                        nextpage    <= page;
                        nextcontrol <= control;
                    end

DATA1: begin
                    nextstate   <= DATA2;
                    nextaddr    <= addr;
                    nextpage    <= page;

                    if(button[3])
                          nextcontrol <= SETDATA1;
                    else
                          nextcontrol <= SETDATA2;

                    case (button)
                          B:     nextdata <= BDATA;
                          Y:     nextdata <= YDATA;
                          SEL:   nextdata <= SELDATA;
                          START: nextdata <= STARTDATA;
                          UP:    nextdata <= UPDATA;
                          DOWN:  nextdata <= DOWNDATA;
                          LEFT:  nextdata <= LEFTDATA;
                          RIGHT: nextdata <= RIGHTDATA;
                          A:     nextdata <= ADATA;
                          X:     nextdata <= XDATA;
                          L:     nextdata <= LDATA;
                          R:     nextdata <= RDATA;
                          CLEARB: nextdata <= CLEARDATA;
                          CLEARY: nextdata <= CLEARDATA;
                          CLEARSEL:    nextdata <= CLEARDATA;
                          CLEARSTART:  nextdata <= CLEARDATA;
                          CLEARUP:     nextdata <= CLEARDATA;
                          CLEARDOWN:   nextdata <= CLEARDATA;
                          CLEARLEFT:   nextdata <= CLEARDATA;
                          CLEARRIGHT:  nextdata <= CLEARDATA;
                          CLEARA:      nextdata <= CLEARDATA;
                          CLEARX:      nextdata <= CLEARDATA;
                          CLEARL:      nextdata <= CLEARDATA;
                          CLEARR:      nextdata <= CLEARDATA;
                          INITIALIZE1: nextdata <= CLEARDATA;
                          INITIALIZE2: nextdata <= CLEARDATA;
                          DELAY:       nextdata <= CLEARDATA;
                          default:     nextdata <= CLEARDATA;
                     endcase
               end

DATA2: begin
                    nextstate   <= DATA3;
                    nextdata    <= data;
                    nextaddr    <= addr;
                    nextpage    <= page;
                    nextcontrol <= control;
```

```
            end

DATA3: begin
                nextstate   <= DATA4;
                nextdata    <= data;
                nextaddr    <= addr;
                nextpage    <= page;

                if(button[3])
                      nextcontrol <= 6'b110011;
                else
                      nextcontrol <= 6'b101011;
            end

DATA4: begin
                nextstate   <= DATA5;
                nextdata    <= data;
                nextaddr    <= addr;
                nextpage    <= page;
                nextcontrol <= control;
            end

DATA5: begin
                nextstate   <= ADDRCHK;
                nextdata    <= data;
                nextaddr    <= addr;
                nextpage    <= page;

                if(button[3])
                      nextcontrol <= SETDATA1;
                else
                      nextcontrol <= SETDATA2;
            end

ADDRCHK: begin
                nextdata    <= data;
                nextcontrol <= control;
                nextaddr    <= addr;
                nextpage    <= page;

                case (button)
                      B:    if(addr == BFIN)
                              nextstate <= DONE;
                            else
                              nextstate <= ADDRINCR;
                   CLEARB:  if(addr == BFIN)
                              nextstate <= DONE;
                                  else
                              nextstate <= ADDRINCR;
                      Y:    if(addr == YFIN)
                              nextstate <= DONE;
                            else
                              nextstate <= ADDRINCR;
                CLEARY:     if(addr == YFIN)
                              nextstate <= DONE;
                            else
                             nextstate <= ADDRINCR;
```

```
SEL:        if(addr == SELFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
CLEARSEL:   if(addr == SELFIN)
                    nextstate <= DONE;
            else
                    nextstate <= ADDRINCR;
START:      if(addr == STARTFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
CLEARSTART: if(addr == STARTFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
UP:         if(addr == UPFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
CLEARUP:    if(addr == UPFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
DOWN:       if(addr == DOWNFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
            CLEARDOWN:   if(addr == DOWNFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
LEFT:       if(addr == LEFTFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
CLEARLEFT:  if(addr == LEFTFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
RIGHT:      if(addr == RIGHTFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
CLEARRIGHT: if(addr == RIGHTFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
      A:    if(addr == AFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
CLEARA:     if(addr == AFIN)
              nextstate <= DONE;
            else
              nextstate <= ADDRINCR;
      X:    if(addr == XFIN)
              nextstate <= DONE;
```

```
                                    else
                                      nextstate <= ADDRINCR;
                        CLEARX:     if(addr == XFIN)
                                      nextstate <= DONE;
                                    else
                                      nextstate <= ADDRINCR;
                              L:    if(addr == LFIN)
                                      nextstate <= DONE;
                                    else
                                      nextstate <= ADDRINCR;
                        CLEARL:     if(addr == LFIN)
                                      nextstate <= DONE;
                                    else
                                      nextstate <= ADDRINCR;
                              R:    if(addr == RFIN)
                                      nextstate <= DONE;
                                    else
                                      nextstate <= ADDRINCR;
                        CLEARR:     if(addr == RFIN)
                                      nextstate <= DONE;
                                    else
                                      nextstate <= ADDRINCR;
                  INITIALIZE1:      if(addr == INITIALIZEFIN)
                                      nextstate <= PAGECHK;
                                    else
                                      nextstate <= ADDRINCR;
                  INITIALIZE2:      if(addr == INITIALIZEFIN)
                                      nextstate <= PAGECHK;
                                    else
                                      nextstate <= ADDRINCR;
                              DELAY:      if(addr == DELAYFIN)
                                      nextstate <= DONE;
                                    else
                                      nextstate <= ADDRINCR;
                                    default:
                                      nextstate <= DONE;
                              endcase
                        end

   ADDRINCR: begin
                              nextdata    <= data;
                              nextcontrol <= control;
                              nextaddr    <= addr+8'b00000001;
                              nextpage    <= page;
                              nextstate   <= STATCHK1;
                        end
   PAGECHK:    begin
                              nextdata    <= data;
                              nextcontrol <= control;
                              nextaddr    <= INITIALIZEINIT;
                              nextpage    <= page;

                              if (page == FINPAGE)
                                    nextstate <= DONE;
                              else  nextstate <= PAGEINCR;
                        end

PAGEINCR: begin       // Increment page each time through the
```

```
                                    // the state machine if initializing.  This
                                    // lets the initialization clear ever pixel
                                    nextstate   <= STATCHK7;
                                    nextdata    <= data;
                                    nextcontrol <= control;
                                    nextaddr    <= addr;
                                    nextpage    <= page+1;
                                     end

                    DONE: begin
                                    nextstate   <= INIT;
                                    nextdata    <= CLEARDATA;
                                    nextcontrol <= STATUSCHK1;
                                    nextaddr    <= INITIALIZEINIT;
                                    nextpage    <= page;
                                end

                    default: begin
                                         nextstate   <= INIT;
                                         nextdata    <= data;
                                         nextcontrol <= control;
                                         nextaddr    <= addr;
                                         nextpage    <= page;
                                    end
            endcase

      // Output Logic

      assign done = (state == DONE);
      assign LCDcontrol = control;
      assign drive = ~state[5];     // Set LCDdata as high impedance for status
      assign LCDdata = drive ? data : 8'bz;  // lets LCD drive busy sometimes

endmodule



module RAM(clk,dclk,address,write,data_in,data_out,led);
      input clk;
      input dclk;
      input [8:0] address;
      input write;
      input data_in;
      output [11:0] data_out;
      output [7:0] led;

      wire [15:0] data;
      reg [11:0] Mem[300:0];
      reg [11:0] internal_bus;

       // Shift register for incoming data
       shift_reg shift_reg(dclk,data_in,data);

       //initial internal_bus = Mem[address];   // initialize the output

       always @(address) internal_bus <= Mem[address];      // update the output
```

```
        always @(posedge write)
             Mem[address] <= data[15:4];         // load value into memory

        assign data_out = internal_bus;
        assign led = ~data[7:0];

endmodule
```