Final Lab Project

- Magnetic stripe card reader to LCD display
- Solution will require four (4) 22V10 chips

Given:
- Schematic
- Test fixtures

Your job:
- Design the core of the PALs

Overview of Magnetic Stripe Cards

- Commonly used in credit, debit, transportation, and gift cards
- Magnetic material (iron-ion rich) is contained in a plastic-like film
  - Stripe is 5.66 mm from edge of card and is 9.52 mm wide
  - Contains three tracks, each 2.79 mm wide
    - Tracks one and three are typically recorded at 8.27 bits per mm
    - Track two typically has a recording density of 2.95 bits per mm
- Various ISO standards define format
  - 7810, 7811, 7812, 7813, and 4909
  - Defined by each industry

See [http://en.wikipedia.org/wiki/Magnetic_stripe_card](http://en.wikipedia.org/wiki/Magnetic_stripe_card) for details

<table>
<thead>
<tr>
<th>TRACK</th>
<th>Recording Density (Bit per inch)</th>
<th>Character Configuration (Including parity bits)</th>
<th>Information Content (Including control characters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.110&quot;</td>
<td>1 VISA</td>
<td>7 bits per character 19</td>
<td>70 alphanumeric characters</td>
</tr>
<tr>
<td>0.110&quot;</td>
<td>2 KAICA</td>
<td>6 bits per character 21</td>
<td>40 numeric characters</td>
</tr>
<tr>
<td>0.110&quot;</td>
<td>3 THRIFT</td>
<td>5 bits per character 21</td>
<td>107 numeric characters</td>
</tr>
</tbody>
</table>
Overview of Magnetic Stripe Cards

- Data encoded as 7-bit characters
  - 6 bits for value (least significant bit first)
  - 1 bit for parity

Card Data Format - Track 1

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>PWN</td>
<td>Primary</td>
<td>8 bits</td>
</tr>
<tr>
<td>02</td>
<td>NAME</td>
<td>Name</td>
<td>16 bits</td>
</tr>
<tr>
<td>03</td>
<td>ADDITIONAL DATA</td>
<td>No. of characters</td>
<td>2 bits</td>
</tr>
<tr>
<td>04</td>
<td></td>
<td>Data characters</td>
<td>56 bits</td>
</tr>
<tr>
<td>05</td>
<td></td>
<td>Check character</td>
<td>1 bit</td>
</tr>
<tr>
<td>06</td>
<td></td>
<td>Verification key</td>
<td>12 bits</td>
</tr>
<tr>
<td>07</td>
<td></td>
<td>Verification Value</td>
<td>12 bits</td>
</tr>
<tr>
<td>08</td>
<td></td>
<td>Card Verification Value</td>
<td>12 bits</td>
</tr>
<tr>
<td>09</td>
<td></td>
<td>Service Code 1</td>
<td>2 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Service Code 2</td>
<td>2 bits</td>
</tr>
</tbody>
</table>

Reader serial data format

- 3 signals
  - RCP – “clock”
    - RCP only oscillates if card is moving
  - RDP – data
  - CLS – card “present” indicator
    - CLS is only active if a card is present

Decoding

- Use RCP falling transition to sample RDP only when CLS is asserted
Block diagram

- Major components
  - Reader outputs (simulation test fixture)
  - Reader buffer (logic that goes into XLA board’s FPGA)
  - LCD controller (Lab 9)
  - Reader signal decoder and serial-to-parallel converter (Lab 10)
  - LCD display (simulation test fixture)

<table>
<thead>
<tr>
<th>Simulation Model</th>
<th>YOUR CIRCUIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Card</td>
<td>Reader</td>
</tr>
<tr>
<td>Reader</td>
<td>Buffer</td>
</tr>
<tr>
<td>Reader hardware</td>
<td>XLA5 FPGA</td>
</tr>
<tr>
<td>Decoder &amp; S-to-P</td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>Display</td>
</tr>
</tbody>
</table>

LCD interface

- Eleven signal wires plus PWR/GND/Vo
  - 1 mode input
  - 1 read/write control
  - 1 enable
  - 8 data lines

**Interface Pin Connections**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC</td>
<td>Power Supply</td>
</tr>
<tr>
<td>2</td>
<td>GND</td>
<td>Power Supply</td>
</tr>
<tr>
<td>3</td>
<td>VCC</td>
<td>Power Supply</td>
</tr>
<tr>
<td>4</td>
<td>RD</td>
<td>Data Input</td>
</tr>
<tr>
<td>5</td>
<td>WR</td>
<td>Data Input</td>
</tr>
<tr>
<td>6</td>
<td>EN</td>
<td>Enable Control</td>
</tr>
<tr>
<td>7</td>
<td>DB</td>
<td>Data Bus Line</td>
</tr>
<tr>
<td>8</td>
<td>RS</td>
<td>Valid Mode</td>
</tr>
<tr>
<td>9</td>
<td>RW</td>
<td>Valid R/W Control</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Pin Layout**

*Outline Drawing*
Basic LCD operations

- Requires sequence of 4 commands on initialization
- Many more commands
  - E.g., backup cursor, blink, etc.
- Data write prints character to display

<table>
<thead>
<tr>
<th>Operation</th>
<th>RS</th>
<th>DB7...DB0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Display</td>
<td>0</td>
<td>0000 0001</td>
</tr>
<tr>
<td>Function Set</td>
<td>0</td>
<td>0011 0011</td>
</tr>
<tr>
<td>Display On</td>
<td>0</td>
<td>0000 1100</td>
</tr>
<tr>
<td>Entry Mode Set</td>
<td>0</td>
<td>0000 0110</td>
</tr>
<tr>
<td>Write Character</td>
<td>1</td>
<td>DDDD DDDD</td>
</tr>
</tbody>
</table>

ASCII codes

- Each character has a unique code
- Some codes could be used to issue commands to display
  - E.g., clear, backspace, etc.
  - These are extra credit
Block Diagram (lab 9)

Features of 22V10 PAL
Module description:

```
module lcd_control (clk, reset, write, EN, RS, CMD);
    input clk, reset;
    input write; // Write a character to the LCD
    output EN, RS; // Enable, RS signals of LCD
    output [1:0] CMD; // Index for current LCD command
    /* reg [??:??] state; */
    /* YOUR DECLARATIONS ETC. GO HERE */
    always @(posedge clk) begin
        /* YOUR SYNCHRONOUS CODE GOES HERE */
        end
endmodule
```

Module `lcd_cmd` description:

```
module lcd_cmd (RS, cmdIndex, lcdCMD);
    input RS;                  // Used to tristate the LCD CMD
    input [1:0] cmdIndex;      // Index of the command
    output [7:0] lcdCMD;       // LCD command
    /* YOUR CODE HERE */
endmodule
```

Module `tri_driver` description:

```
module tri_driver (en, from, to);
    input en;
    input [7:0] from;
    output [7:0] to;
    assign to = (en) ? from : 8'bzzzzzzzz;
endmodule
```
module lcd_tf (reset, RS, EN, RW, data);
input reset, RS, EN, RW;
input [7:0] data;
reg [2:0] resetCnt; // Counts through the reset sequence
parameter CMD0 = 8'h1, CMD1 = 8'h33, CMD2 = 8'hC, CMD3 = 8'h6;
parameter hold = 7, setup = 7;
initial begin resetCnt = 0; dataTime = 0; RSTime = 0; ENTime = 0;
end
always @(negedge reset) resetTime = $time;
always @(data) begin
  dataTime = $time;
  if (~reset && (dataTime != resetTime) && (EN==0) && (($time - ENTime) < hold)) begin
    $display("Error: Data hold time not met: %t", ($time-ENTime));
    $stop;
  end
end
always @(RS) begin
  RSTime = $time;
  if (~reset && (RSTime != resetTime) && (EN==0) && ($time - ENTime) < hold) begin
    $display("Error: RS hold time not met: %t", ($time-ENTime));
    $stop;
  end
end
always @(posedge EN) begin
  ENTime = $time;
  // Check RS setup time - there is no setup/hold wrt. data
  if ((ENTime - resetTime) && (EN==0) && (($time - ENTime) < setup)) begin
    $display("Error: EN setup time not met: %t", ($time-ENTime));
    $stop;
  end
end
always @(negedge EN) begin
  ENTime = $time;
  if (reset == 0) begin
    ENTime = $time;
    if (($time - dataTime) < setup) begin
      $display("Error: Data setup time not met: %t", ($time-dataTime));
      $stop;
    end
    if (($time - RSTime) < setup) begin
      $display("Error: RS setup time not met: %t", ($time-RSTime));
      $stop;
    end
    if (RW !== 0) begin
      $display("Error: RW should be 0");
      $stop;
    end
    if (RS === 0) begin // Writing a command
      case (resetCnt)
        0: begin // Turn on
          if (data == CMD0) begin $display("Display cleared"); resetCnt = 1;
            end
          else begin $display("Error: Invalid reset command 0");
            $stop;
          end
        end
        1: begin
          if (data == CMD1) begin $display("Display function set"); resetCnt = 2;
            end
          else begin $display("Error: Invalid reset command 1");
            $stop;
          end
        end
        2: begin
          if (data == CMD2) begin $display("Display turned on"); resetCnt = 3;
            end
          else begin $display("Error: Invalid reset command 2");
            $stop;
          end
        end
        3: begin
          if (data == CMD3) begin $display("Display entry mode set"); resetCnt = 4;
            end
          else begin $display("Error: Invalid reset command 3");
            $stop;
          end
        default: begin $display("Error: Too many reset commands");
            $stop;
          end
      endcase // case(resetCnt)
    end else if (RS === 1) begin // Writing a character
      if (resetCnt != 4) begin $display("Display has not been properly reset");
        end
      $display("Write Character: %c", data);
    end // else: !if(RS == 0)
  end // if (reset == 0)
end // always @(negedge EN)
endmodule
Block Diagram (lab 10)

One more PAL to interface reader and implement decoder

Magnetic stripe reader test fixture (1 of 2)

```verilog
module magreader_tf (reset, clk, outRCP, outRDP, outCLS);
    input reset, clk;
    output outRCP, outRDP, outCLS;
    reg outCLS, outReset;
    reg intRCP;
    assign outRCP = intRCP;
    integer delay = 0; // Generate delays
    integer count = 0; // Count bits we have sent
    parameter CLS_STATE = 0;
    // These specify the delays in terms of clock cycles
    parameter BEFORECLS = 14, // From reset to CLS asserted
        AFTERCLS = 25, // From reset to first clock
        CPHIGH = 1, CPLOW = 3, // Clock high and low time
        CLSDONE = 4; // From end of data to CLS off
    parameter BUFSIZE = 64; // Number of bits sent
    // Buffer for input data bits
    reg [BUFSIZE-1:0] data;
    // RDP output from data buffer
    assign outRDP = ~data[0];
    // States used to generate data
    parameter CLS = 0, DATA = 1, DONE = 2;
    reg [3:0] state;
    ...
```
Magnetic stripe reader test fixture (2 of 2)

```verilog
always @(posedge clk) begin
  if (reset) begin
    // Make sure the low order bits are the sentinel character.
    // data[0] is the first high bit for initialization
    data <= 64'b00000_00000_00000_00000_00000_001000_0101100_0101100_0100101_1101000_1000101_000;
    state <= CLS; // Start by asserting CLS
    delay <= 0;
    count <= 0;
    intRCP <= 1;
    outCLS <= 1;
    outReset <= 1;
    end else begin
      delay <= delay + 1;
      case (state)
        CLS: begin
          outReset <= 0;
          if (delay == BEFORECLS) begin outCLS <= 0;
            end else if (delay == AFTERCLS) begin state <= DATA; delay <= 0;
              end
            end // case: CLS
        DATA: begin
          if (delay == CPHIGH) begin intRCP <= 0;
            end else if (delay == (CPHIGH+CPLOW)) begin
              delay <= 0;
              intRCP <= 1;
              count <= count + 1;
              data <= { 1'b0, data[(BUFSIZE-1):1] };// Shift data right
              if (count == (BUFSIZE-1)) state <= DONE;
            end
          end // case: DATA
        DONE: begin
          if (~outCLS && (delay == CLSDONE)) begin outCLS <= 1;
            end
          end // case: DONE
      endcase // case(state)
    end // else: !if(Reset)
  end // always @(posedge clk)
endmodule // magreader_tf
```

Purpose of the project

- Learn how to build a realistic system
- Read data sheets
- Communicating state machines
- Deal with existing code/components