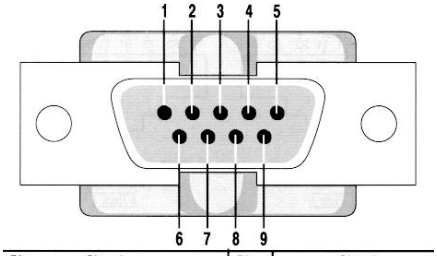
Final Project

- RS-232 serial line to LCD display
- Solution will require 4 chips on the XLA5 protoboard
  - 1 – ‘377
  - 1 – ‘244
  - 2 – 22v10 PALs
- We’ll provide everything but the core of the two PAL programs

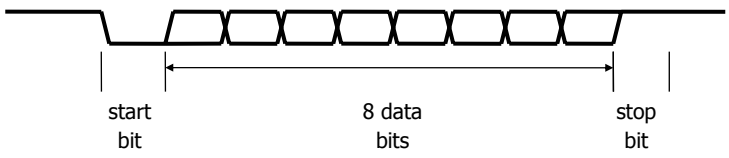
Overview of RS232

- Very established serial line communication protocol
- Originally designed for teletypes and modems
  - Point-to-point, full-duplex
  - Variable baud (bit) rates
  - Cheap 9-wire connector connectors



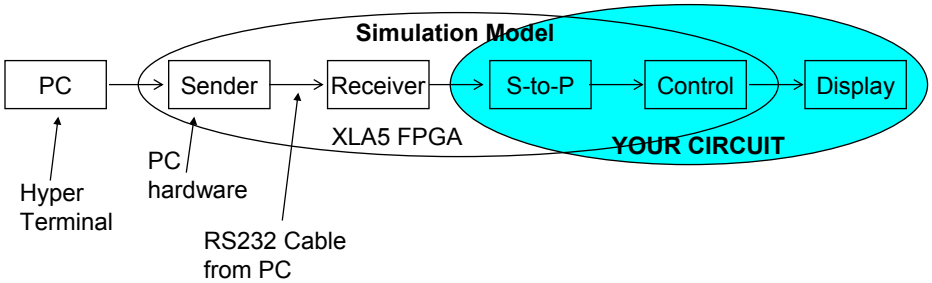
Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

RS232 serial data format

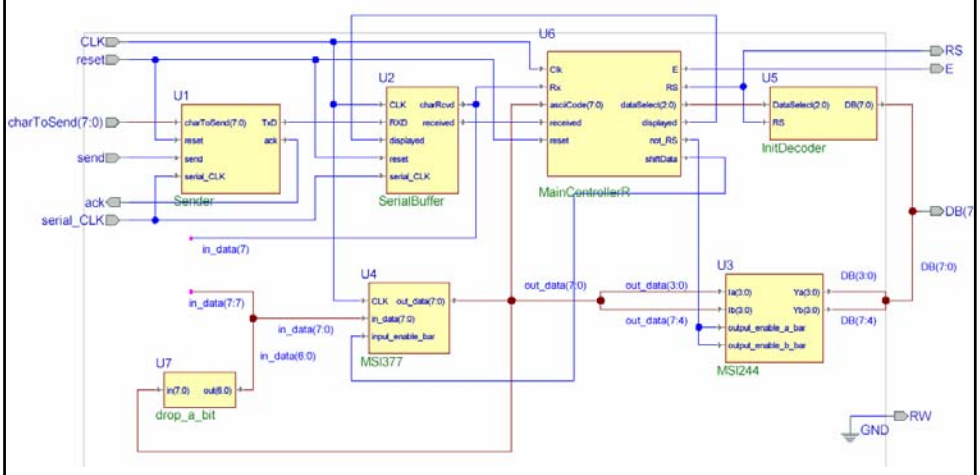


Block diagram

- Major components
  - RS232 sender (for simulation test bench)
  - RS232 receiver
  - **Serial-to-parallel converter**
  - **Main controller**
  - LCD display (no simulation model)



Simulation model

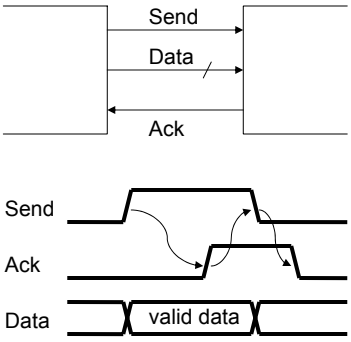


Sender

```
module Sender (serial_CLK, reset, send, charToSend, ack, TxD);
    input               serial_CLK, reset, send;
    input               [7:0] charToSend;
    output              ack, TxD;
    reg                 TxD;
    reg [7:0]           charToSendBuffer;
    reg [4:0]           bitCounter;
    reg                 sending;
    reg     [1:0]       cycleCounter;

    always @(posedge serial_CLK) begin
        if (reset) begin TxD <= 1; sending <=0; end
        else begin
                    if (send) begin
                                charToSendBuffer <= charToSend;
                                bitCounter <= 0;
                                sending <= 1;
                                cycleCounter <= 0;
                    end
                    else if (sending) begin
                                if(cycleCounter == 0) begin
                                            bitCounter <= bitCounter + 1;
                                            if (bitCounter == 0) TxD <= 0;
                                            else if (bitCounter > 0 && bitCounter <= 8)
                                                        TxD <= charToSendBuffer[bitCounter - 1];
                                            else if (bitCounter > 8)
                                                        begin TxD <= 1; sending <= 0; end
                                end
                                cycleCounter <= cycleCounter + 1;
                    end
        end
    end
    assign ack = sending;
endmodule
```

Four-cycle handshake between modules

- Don't let one get ahead of the other



Serial buffer (receiver)

```
module SerialBuffer(CLK, serial_CLK, reset, RXD, charRcvd, received, displayed);
    input  CLK;
    input  serial_CLK;
    input  reset;
    input  RXD;
    input  displayed;

    output charRcvd;
    reg    charRcvd;
    output received;
    reg    received;

    reg [3:0] bitCounter;
    reg [1:0] cycleCounter;
    reg [9:0] characterReceived;
    reg [8:0] characterToOutput;
    reg [3:0] shiftCounter;
    reg       receiving;
    reg       shifting;
    reg       done;
```

Serial buffer (receiver) – cont'd

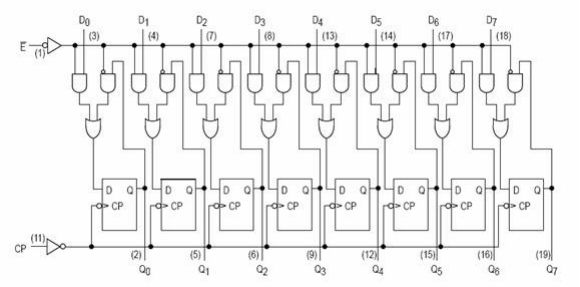
```
always @(posedge serial_CLK) begin
    if (reset) begin
          receiving <= 0;
          received <= 0;
    end
    else begin
          if (!receiving && !received && !displayed &&  RXD) ;
          if (!receiving && !received && !displayed && !RXD) begin
                    receiving <=1;
                    cycleCounter <= 0;
                    bitCounter <= 0;
          end
          if (receiving) begin
                    if (cycleCounter == 0) begin
                              characterReceived[bitCounter] <= RXD;
                              if (bitCounter < 9) bitCounter <= bitCounter + 1;
                              else begin
                                        receiving <= 0;
                                        received <= 1;
                              end
                    end
                    cycleCounter <= cycleCounter + 1;
          end
          if (received && displayed) begin
                    received <= 0;
          end
    end
end
```


Serial buffer (receiver) – cont'd

```
always @(posedge CLK) begin
    if (reset) begin
          shifting <= 0;
          done <= 0;
          charRcvd <= 1;
    end
    else if (received && !shifting && !done) begin
          characterToOutput <= characterReceived[8:0];
          shiftCounter <= 0;
          shifting <= 1;
    end
    else if (shifting && (shiftCounter < 9)) begin
          charRcvd <= characterToOutput[shiftCounter];
          shiftCounter <= shiftCounter + 1;
    end
    else if (shifting && (shiftCounter >= 9)) begin
          shifting <= 0;
          done <= 1;
          charRcvd <= 1;
    end
    else if (!received) begin
          done <= 0;
    end
end

endmodule
```

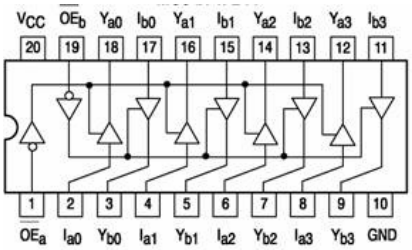
MSI'377 – 8-bit register w/ input enable

```
module MSI377 (CLK, input_enable_bar, in_data, out_data);

    input   CLK, input_enable_bar;
    input   [7:0] in_data;
    output [7:0] out_data;
    reg     [7:0] out_data;

    always @(posedge CLK) begin
          if (!input_enable_bar)
                    out_data <= in_data;
    end

Endmodule
```



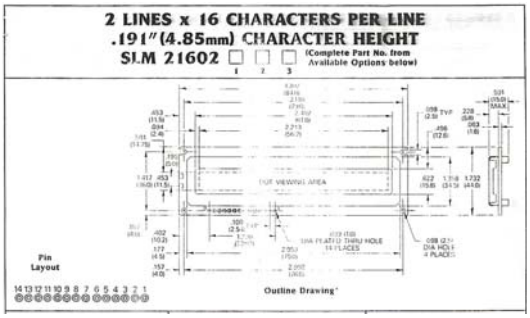
MSI'244 – 2x4-bit tri-state drivers

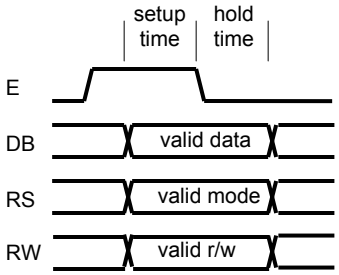
```
module MSI244 (output_enable_a_bar, output_enable_b_bar, Ia, Ib, Ya, Yb);

    input  output_enable_a_bar, output_enable_b_bar;
    input  [3:0] Ia, Ib;
    output [3:0] Ya, Yb;

    assign Ya = (output_enable_a_bar) ? 4'bzzzz : Ia;
    assign Yb = (output_enable_b_bar) ? 4'bzzzz : Ib;

Endmodule
```



LCD interface

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





Interface Pin Connections

Pin No.	Symbol	Function	
1	V_{SS}	0V	Power supply
2	V_{DD}	+5V	
3	V_O	--	
4	RS	H: Data input L: Instruction input	
5	R/W	H: Read(MPU ← LCM) L: Write(MPU → LCM)	
6	E	Enable signal	
7	DB0		Data bus line
8	DB1		
9	DB2		
10	DB3		
11	DB4		
12	DB5		
13	DB6		
14	DB7		

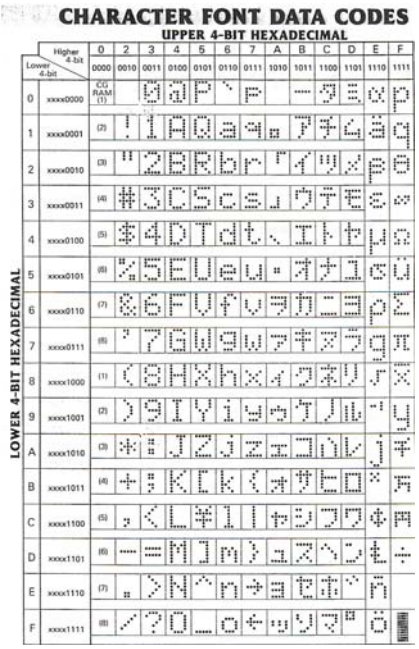
Basic LCD operations

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

Operation	RS	DB7...DB0
Clear Display	0	0000 0001
Function Set	0	0011 0011
Display On	0	0000 1100
Entry Mode Set	0	0000 0110
Write Character	1	DDDD DDDD

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



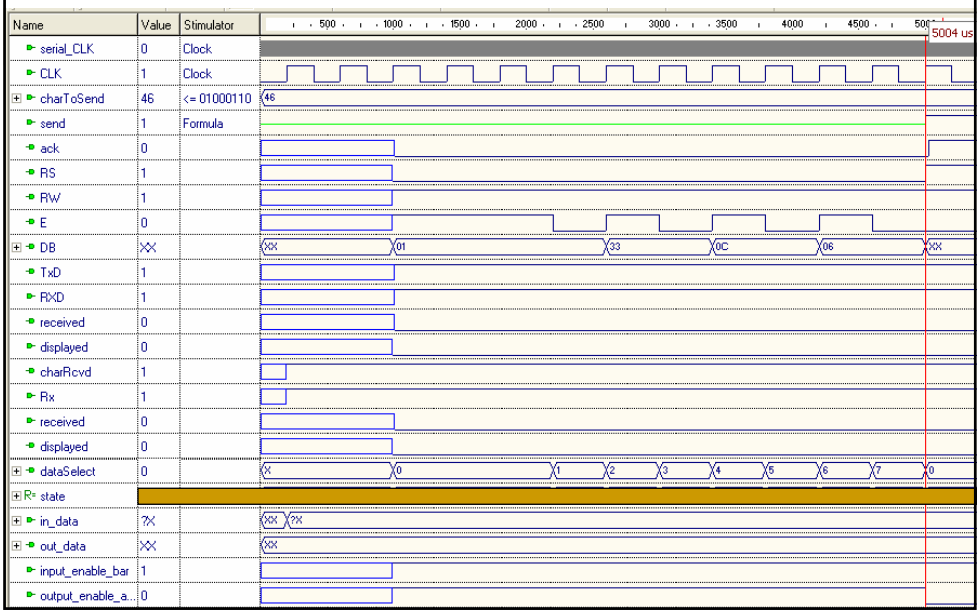
InitDecoder – command inputs to LCD

```
module InitDecoder(DataSelect, RS, DB);

   input  [2:0] DataSelect;
   input  RS;
   output [7:0] DB;

   reg    [7:0] databus;

   YOUR CODE GOES HERE

endmodule
```

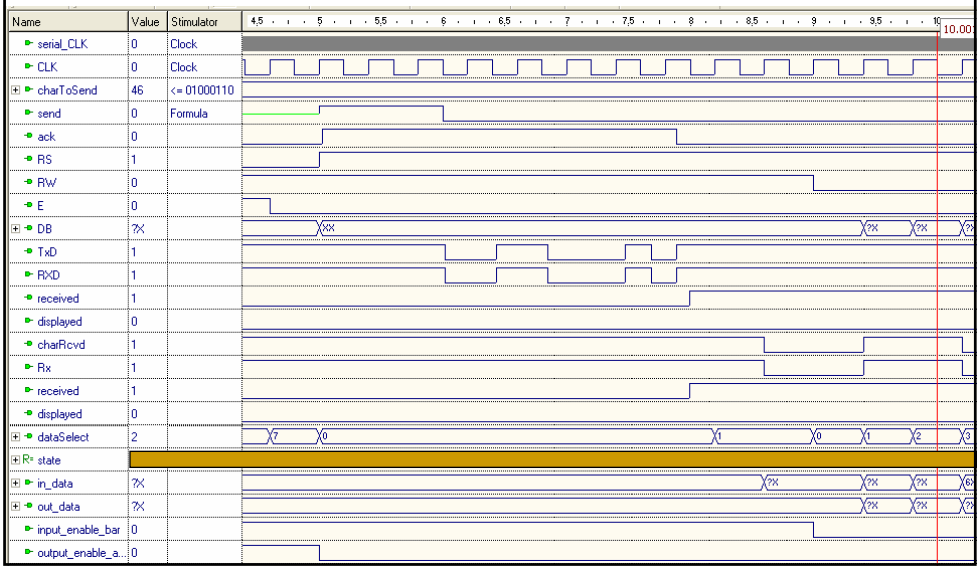
MainController – orchestrates other modules

```
module MainControllerR (reset, asciiCode, not_RS, RS, E, received,
    shiftData, Clk, Rx, dataSelect, displayed);

    input  Rx;
    input  Clk;
    input  reset;
    input  [7:0] asciiCode;
    input  received;

    output shiftData;
    output [2:0] dataSelect;
    output not_RS;
    output RS;
    output E;
    output displayed;

    YOUR CODE GOES HERE

endmodule
```

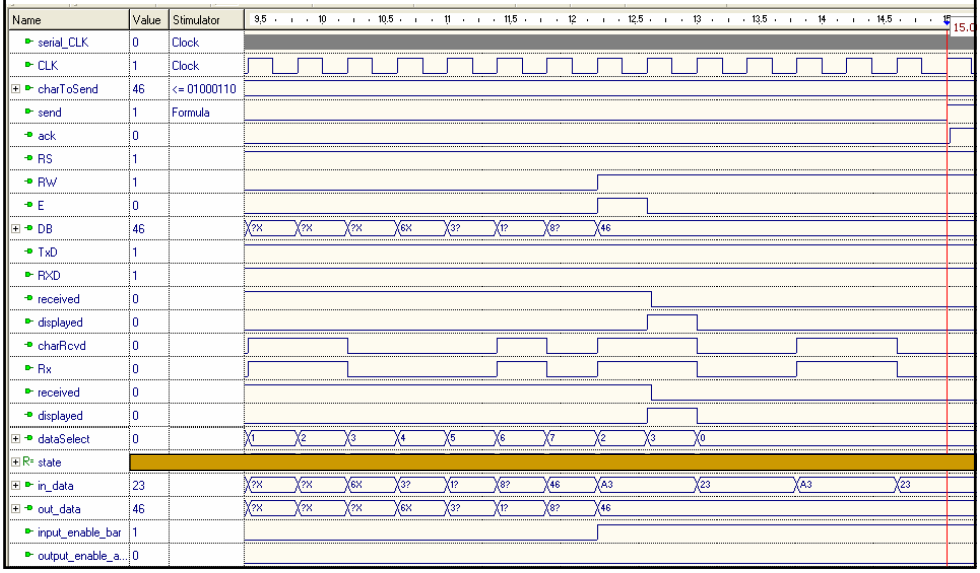

Simulation waveforms (part 1 of 5)



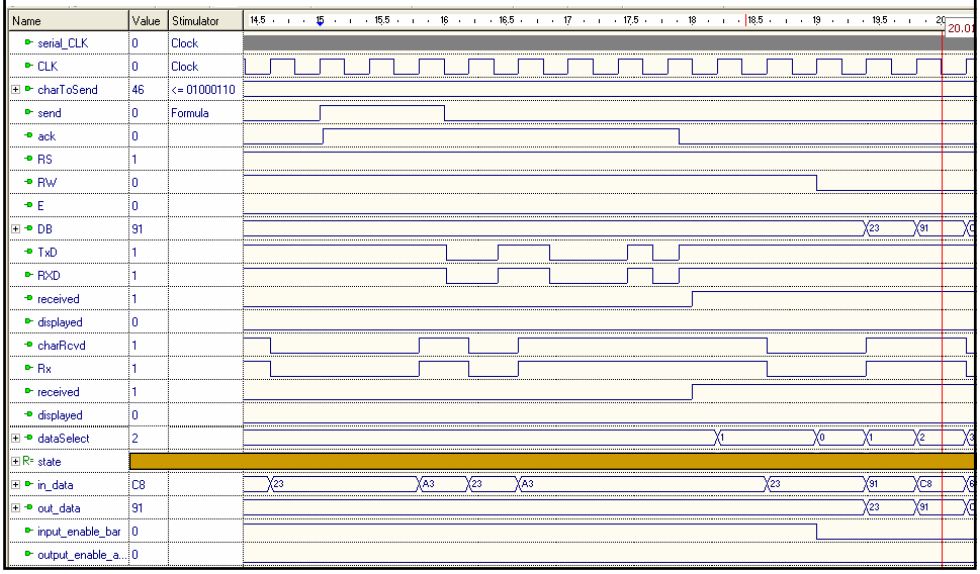
Simulation waveforms (part 2 of 5)



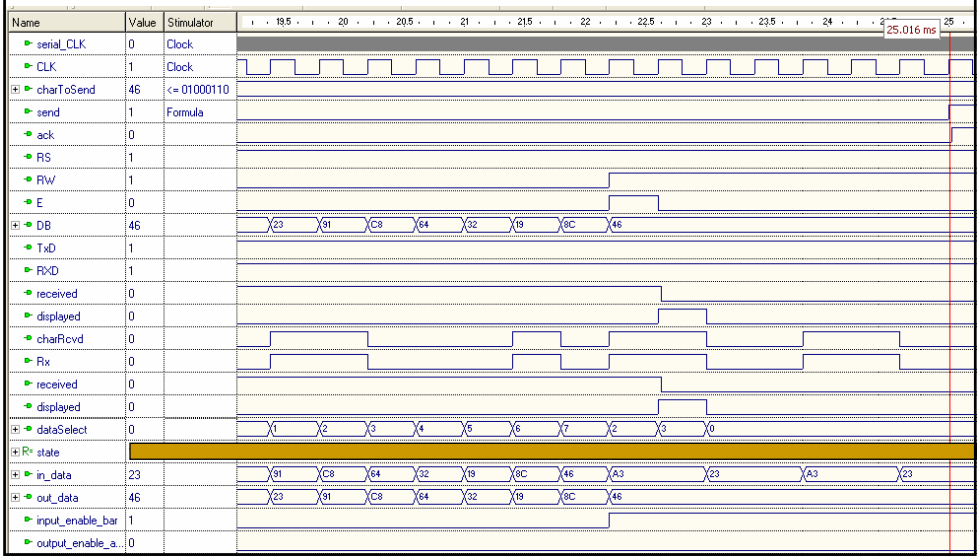
Simulation waveforms (part 3 of 5)



Simulation waveforms (part 4 of 5)



Simulation waveforms (part 5 of 5)



Purpose of the project

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