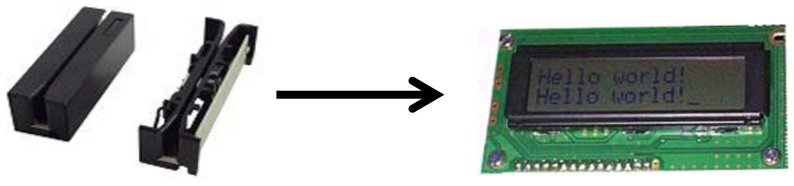
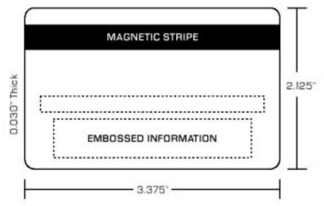
Final Lab Project

- 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

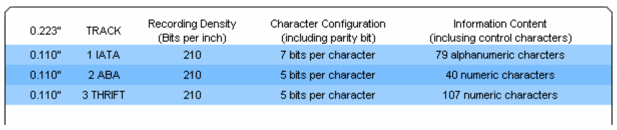


Overview of Magnetic Stripe Cards

- 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



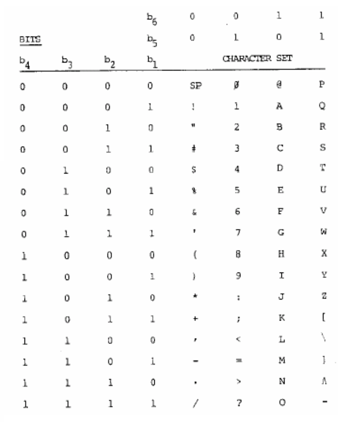
See http://en.wikipedia.org/wiki/Magnetic_stripe_card for details



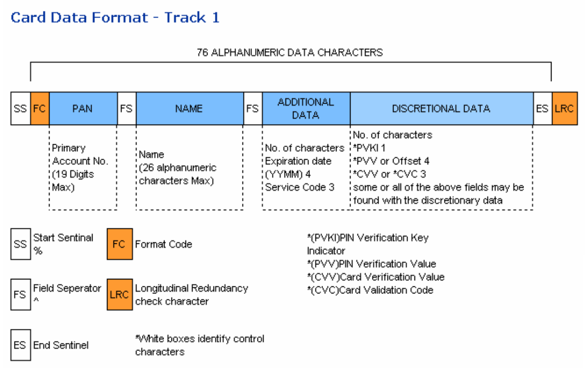
0.223"	TRACK	Recording Density (Bits per inch)	Character Configuration (including parity bit)	Information Content (inclusing control characters)
0.110"	1 IATA	210	7 bits per character	79 alphanumeric charcters
0.110"	2 ABA	210	5 bits per character	40 numeric characters
0.110"	3 THRIFT	210	5 bits per character	107 numeric characters

Overview of Magnetic Stripe Cards

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

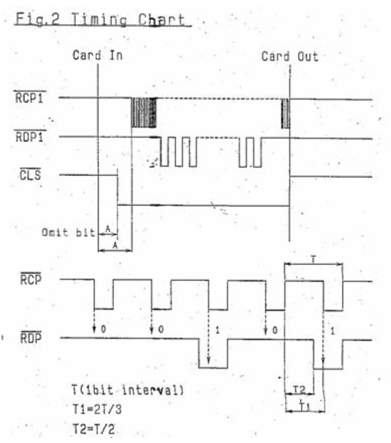


BITS			b_6	0	0	1	1
			b_5	0	1	0	1
b_4	b_3	b_2	b_1	CHARACTER SET			
0	0	0	0	SP	∅	@	P
0	0	0	1	!	1	A	Q
0	0	1	0	"	2	B	R
0	0	1	1	#	3	C	S
0	1	0	0	$	4	D	T
0	1	0	1	%	5	E	U
0	1	1	0	&	6	F	V
0	1	1	1	'	7	G	W
1	0	0	0	(	8	H	X
1	0	0	1	)	9	I	Y
1	0	1	0	*	:	J	Z
1	0	1	1	+	;	K	[
1	1	0	0	,	<	L	\
1	1	0	1	-	=	M	]
1	1	1	0	.	>	N	∧
1	1	1	1	/	?	O	-


Card Data Format - Track 1
76 ALPHANUMERIC DATA CHARACTERS
SS
FC
PAN
FS
NAME
FS
ADDITIONAL DATA
DISCRETIONAL DATA
ES
LRC
Primary Account No. (19 Digits Max)
Name (26 alphanumeric characters Max)
No. of characters Expiration date (YYMM) 4 Service Code 3
No. of characters *PVKI 1 *PVV or Offset 4 *CVV or *CVC 3 some or all of the above fields may be found with the discretional data
SS Start Sentinal %
FC Format Code
FS Field Seperator ^
LRC Longitudinal Redundancy check character
ES End Sentinel
*White boxes identify control characters
*(PVKI)PIN Verification Key Indicator
*(PVV)PIN Verification Value
*(CVV)Card Verification Value
*(CVC)Card Validation Code

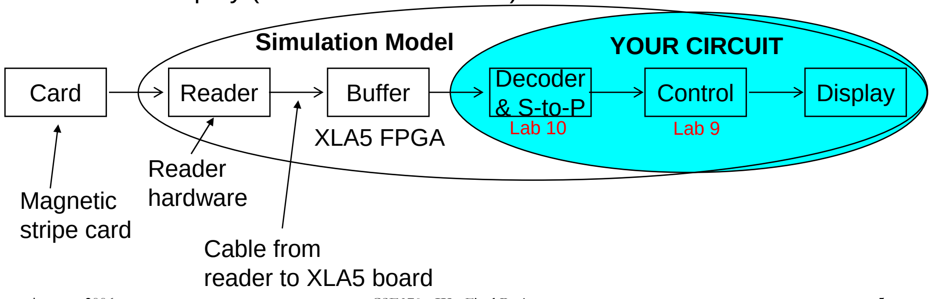
Reader serial data format

- 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

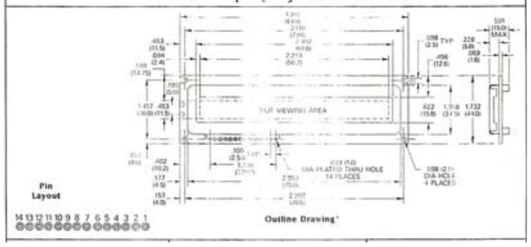
- Major components
  - Reader outputs (simulation test fixture)
  - Reader buffer (logic that goes into XLA board's FPGA)
  - **LCD controller (Lab 9)**
  - **Reader signal decoder and serial-to-parallel converter (Lab 10)**
  - LCD display (simulation test fixture)



LCD interface

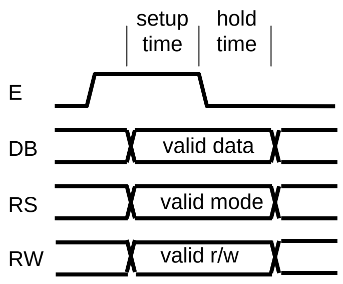
- Eleven signal wires plus PWR/GND/V_o
  - 1 mode input
  - 1 read/write control
  - 1 enable
  - 8 data lines

2 LINES x 16 CHARACTERS PER LINE
.191″(4.85mm) CHARACTER HEIGHT
SLM 21602 ☐ ☐ ☐ (Complete Part No. from Available Options below)



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		



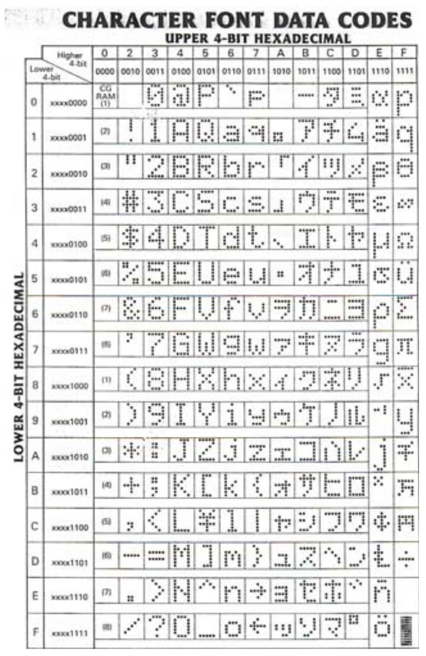
Basic LCD operations

- 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



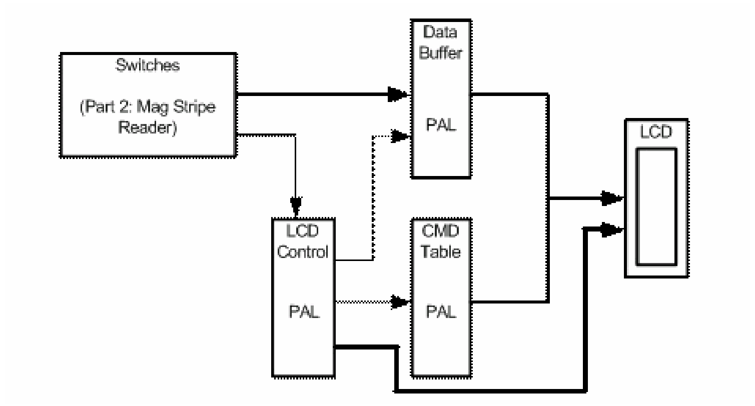
CHARACTER FONT DATA CODES

UPPER 4-BIT HEXADECIMAL

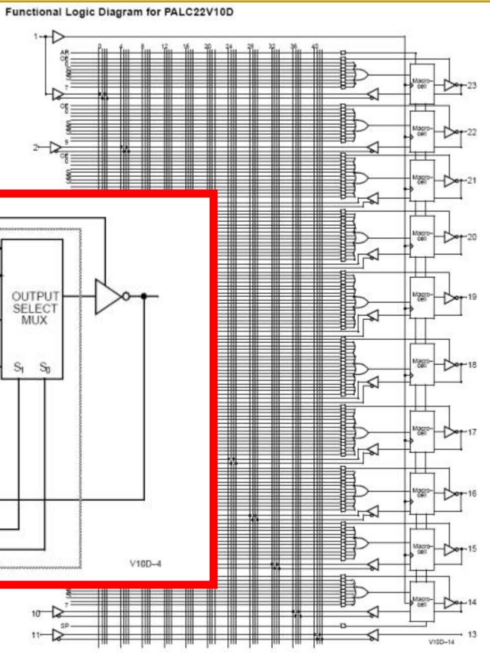
LOWER 4-BIT HEXADECIMAL

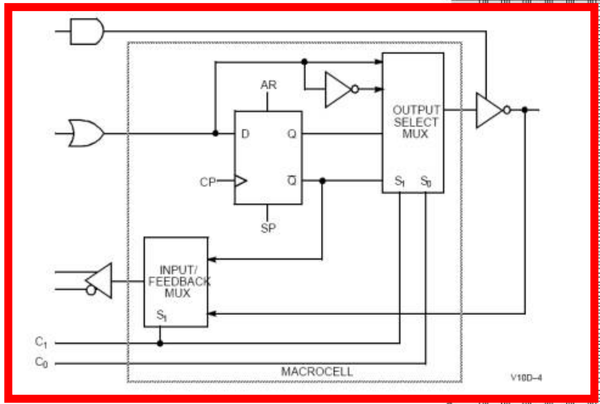
Lower 4-bit \ Higher 4-bit		0	2	3	4	5	6	7	A	B	C	D	E	F
		0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
0	xxxx0000	CG RAM (1)		0	@	P	`	p		ー	タ	ミ	α	p
1	xxxx0001	(2)	!	1	A	Q	a	q	。	ア	チ	ム	ä	q
2	xxxx0010	(3)	"	2	B	R	b	r	「	イ	ツ	メ	β	θ
3	xxxx0011	(4)	#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
4	xxxx0100	(5)	$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
5	xxxx0101	(6)	%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
6	xxxx0110	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
7	xxxx0111	(8)	'	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
8	xxxx1000	(1)	(	8	H	X	h	x	ィ	ク	ネ	リ	√	x̄
9	xxxx1001	(2)	)	9	I	Y	i	y	ゥ	ケ	ノ	ル	⁻¹	y
A	xxxx1010	(3)	*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
B	xxxx1011	(4)	+	;	K	[	k	{	ォ	サ	ヒ	ロ	ˣ	万
C	xxxx1100	(5)	,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
D	xxxx1101	(6)	-	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
E	xxxx1110	(7)	.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
F	xxxx1111	(8)	/	?	O	_	o	←	ッ	ソ	マ	゜	ö	█

Block Diagram (lab 9)



Features of 22V10 PAL





Skeleton Verilog files

```
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
```

Skeleton Verilog files (cont'd)

```
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 (en, from, to);
      input en;
      input [7:0] from;
      output [7:0] to;

      assign to = (en) ? from : 8'bzzzzzzzz;

endmodule
```

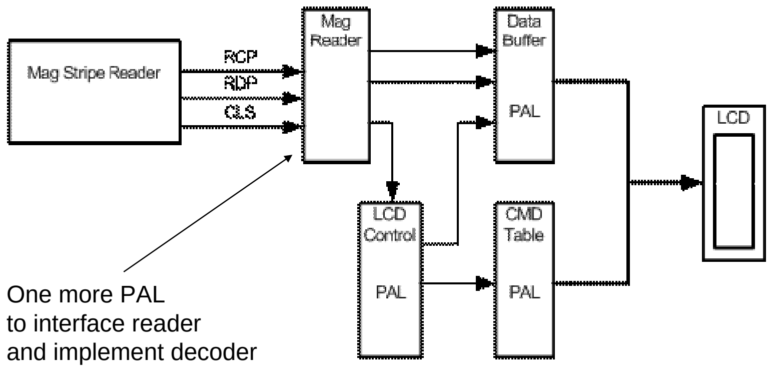
LCD test fixture (1 of 2)

```
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
   time     dataTime, RSTime, ENTime, resetTime;
   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 = $time;
      // Check RS setup time - there is no setup/hold wrt. data
      if ((ENTime != resetTime) && ($time - RSTime) < setup) begin
             $display("Error: RS setup time not met: %t", ($time-RSTime));
             $stop;
      end
   end

  . . .
```

LCD test fixture (2 of 2)

```
always @(negedge EN) begin
     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                     // First reset
              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
           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)



Magnetic stripe reader test fixture (1 of 2)

```
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 [1:0]  state;

. . .
```

Magnetic stripe reader test fixture (2 of 2)

```
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_0000001_0000001_0010000_0101100_0101100_0100101_1101000_1000101_000;
              state <= CLS;                          // Start by asserting CLS
              delay <= 0;
              count <= 0;
              outCLS <= 1;
              intRCP <= 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