The EE 109 LCD Shield

- The LCD shield is a __________________ LCD that mounts on top of the Arduino Uno.
- The shield also contains five buttons that can be used as input sources.

How Do We Use It?

- By sending it _______ (i.e. ______________ one at a time) that it will display for us
- By sending it special __________ to do things like:
  – Move the cursor to a __________________
  – _________ the screen contents
  – Upload new fonts/special characters
6.5 How Do We Communicate?

- The LCD uses a "parallel" interface (4-bits sent per transfer) to communicate with the µC (Note: µC => microcontroller).
- Data is transferred 4 bits at a time and uses 2 other signals (Register Select and Enable) to control _______ the 4-bits go and _______ the LCD should capture them.

Universe

D7 D6 D5 D4 D8 D9

Register Select

Enable

LCD

EE 109 is fun!

6.6 How Do We Communicate?

- To transfer data we send it in two groups of 4
  - First the ______ 4-bits followed by the _______ 4-bits
- RS=0 sets the destination as the command reg.
- RS=1 sets the destination as the data reg.

Universe

D7 D6 D5 D4 D8 D9

Register Select

Enable

LCD

0011

1001

Command Reg.

Data Reg.

0001

1

Display HW

6.7 Commands and Data

- LCD contains _______ registers which it uses to control its actions: Command and Data
- A Register Select (RS) signal determines which register is the destination of the data we send it (RS acts like an address selector)
  - RS = ___, info goes into the command register
  - RS = ___, info goes into the data register
- To perform operations like clear display, move cursor, turn display on or off, write the command code to the command register.
- To display characters on the screen, write the ASCII code for the character to the data register.

<table>
<thead>
<tr>
<th>Command</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear LCD</td>
<td>0x01</td>
</tr>
<tr>
<td>Cursor Home</td>
<td>0x02</td>
</tr>
<tr>
<td>Display On</td>
<td>0x0f</td>
</tr>
<tr>
<td>Display Off</td>
<td>0x08</td>
</tr>
<tr>
<td>Move cursor to top row, column i</td>
<td>0x80+i</td>
</tr>
<tr>
<td>Move cursor to bottom row, column i</td>
<td>0x00+i</td>
</tr>
</tbody>
</table>

6.8 How Do We Communicate?

- To transfer data we send it in two groups of 4
  - First the upper 4-bits followed by the lower 4-bits
- RS=0 sets the destination as the command reg.
- RS=1 sets the destination as the data reg.
Another View

• Data from the Uno is transferred by placing four bits on the data lines (PortD bits 7-4).
• The Register Select (RS) line determines whether the data goes to the LCD's "Command Register" or "Data Register"
  – RS=0 => Command Register  RS=1 => Data Register
• The Enable (E) line acts as a ________ signal telling the LCD to capture the data and examine the RS bit on the 0-1-0 transition
  – Pulse must be held at 1 for at least 230ns according to LCD datasheet

Who's Job Is It?

• So who is producing the values on the RS and Data lines and the 0-1-0 transition on the E line?
• ______!! With your _________ (setting and clearing PORT bits)

Other LCD Interface

• Other LCD devices may use
  – Only one signal (a.k.a. serial link) to communicate between the µC and LCD
    • This makes wiring easier but requires more complex software control to "serialize" the 8- or 16-bit numbers used inside the µC
  – 8-data wires plus some other control signals so they can transfer an entire byte
    • This makes writing the software somewhat easier
**Step 1**

- Mount the LCD shield on the Uno without destroying the pins
- Download the “test.hex” file and Makefile from the web site, and modify the Makefile to suite your computer.
- Run “make test” to download test program to the Uno+LCD.
- Should see a couple of lines of text on the screen.

**Step 2**

- Develop a set of functions that will abstract the process of displaying text on the LCD
  - A set of functions to perform specific tasks for a certain module is often known as an ______ (application programming interface)
  - Once the API is written it gives other application coders a nice simple interface to do high-level tasks
- Download the skeleton file and examine the functions outlines on the next slides

**LCD API Development Overview**

- Write the routines to control the LCD in layers
  - Top level routines that your code or others can use: ______________, ____________, initialize LCD, etc.
  - Mid level routines: write a byte to the command register, write a byte to the data register
  - Low level routines: controls the 4 data lines and E to transfer a nibble to a register
- Goal: Hide the ____________ about how the interface actually works from the user who only wants to put a string on the display.
Low Level Functions

- **lcd_writenibble(unsigned char x)**
  - Assumes RS is already set appropriately
  - Send four bits from ‘x’ to the LCD
    - Takes 4-bits of x and copies them to PD[7:4] (where we’ve connected the data lines of the LCD)
    - **SEE NEXT SLIDES ON COPYING BITS**
  - Produces a 0-1-0 transition on the Enable signal
  - Must be consistent with mid-level routines as to which 4 bits to send, MSB or LSB
  - Uses: logical operations (AND/OR) on the PORT bits

This will be your challenge to write in lab! We will provide the remaining API code.

Mid-Level Functions

- **lcd_writecommand(unsigned char x)**
  - Send the 8-bit byte ‘x’ to the LCD as a command
  - Set RS to 0, send data in two nibbles, delay
  - Uses: **lcd_writenibble()**

- **lcd_writedata(unsigned char x)**
  - Send the 8-bit byte ‘x’ to the LCD as data
  - Set RS to 1, send data in two nibbles, delay
  - Uses: **lcd_writenibble()**

  - Could do as one function
    - **lcd_writebyte(unsigned char x, unsigned char rs)**

High Level API Routines

- **lcd_init()**
  - Does all the steps to initialize the LCD
  - See the lab writeup and follow it **exactly as written**
    - Uses: **lcd_writenibble(), lcd_writecommand(), delays**

- **lcd_moveto(unsigned char row, unsigned char col)**
  - Moves the LCD cursor to “row” (0 or 1) and “col” (0-15)
  - Translates from row/column notation to the format the LCD uses for positioning the cursor (see lab writeup)
    - Uses: **lcd_writecommand()**

- **lcd_stringout(char *s)**
  - Writes a string of character starting at the current cursor position
    - Uses: **lcd_writedata()**

Activity: Code-Along

- Assuming the **lcd_writecommand()** and **lcd_writedata()** functions are correctly written, code the high-level functions:
  - **void lcd_stringout(char* str);**
  - **void lcd_moveto(int row, int col);**
Copying Multiple Bits

- Suppose we want to copy a portion of a variable or register into another but ______ affecting the other bits
- Example: Copy the lower 4 bits of X into the lower 4-bits of PORTB...but leave the upper 4-bits of PORTB UNAFFECTED
- Assignment ______ work since it will overwrite ALL bits of PORTB
  - PORTB = x; // changes all bits of PORTB

Copying Into a Register

- Solution...use these steps:
  1. Define a _____ that has 1's where the bits are to be copied
    #define MASKBITS 0x0f
  2. ______ those bits in the destination register using the MASK
    PORTB &= ~MASKBITS
  3. Mask the appropriate field of x and then _____ it with the destination, PORTB
    PORTB |= ___________

Do We Need Step 2...Yes!!!

- Can't we just do step 1 and 3 and OR the bits of x into PORTB
  #define MASKBITS 0x0f
  PORTB |= (x & MASKBITS);
- ____, because what if the destination (PORTB) already had some 1's where we wanted 0's to go...
- ...Just ______ wouldn't change the bits to ____
- That's why we need step 2
  2. Clear those bits in the destination register using the MASK
  PORTB &= ~MASKBITS;
Copy To Different Bit Locations

• What if the source bits are in a different location than the destination
  — Ex. Copy lower 4 bits of x to upper 4 bits of PORTB

• Step 1: Define a mask that has 1's where the bits are to be copied
  #define MASKBITS 0xf0

• Step 2: Clear those bits in the destination register using the MASK
  PORTB &= ~MASKBITS

• Step 3: ______the bits of x to align them appropriately, then perform the regular step 3
  PORTB |= (______) & MASKBITS;

Coding a Byte Transfer to the LCD

Does your code do the right thing?

• LCD lab required the program to generate an Enable (E) pulse.
• Example: The writenibble() routine controls the PB1 bit that is connected to the LCD Enable line.
  PORTB |= (1 << PB1); // Set E to 1
  PORTB &= ~(1 << PB1); // Clear E to 0
• Creates a 0→1→0 pulse to clock data/orcommands into LCD.
• But is it a pulse that will work with the LCD?
• Rumors circulated that the E pulse had to be made longer by putting a delay in the code that generated it.
• Don’t Guess. Time to read the manual, at least a little bit.
Check the LCD controller datasheet

Timing Characteristics

**RS**

VIH1 VIL1

\( t_{\text{RS}} \) \( t_{\text{VIH1}} \) \( t_{\text{VIL1}} \)

**R/W**

VIH1 VIL1

\( t_{\text{R/W}} \) \( t_{\text{VIH1}} \) \( t_{\text{VIL1}} \)

**E**

VIH1 VIL1

\( t_{\text{E}} \) \( t_{\text{PIED}} \) \( t_{\text{PW}} \)

**DB0 to DB7**

VIH1 VIL1

\( t_{\text{DB0 to DB7}} \) \( t_{\text{VIL1}} \) \( t_{\text{Valid data}} \) \( t_{\text{next}} \) \( t_{\text{cycE}} \)

Figure 27 Write Operation

Check the generated code

- Can check the code generated by the compiler to see what is happening.
- For the creation of the E pulse the compiler generated this code:
  
  \[
  \begin{align*}
  &\text{SBI \ PORTB, 1} \quad \text{; Set Bit Immediate, PORTB, bit 1} \\
  &\text{CBI \ PORTB, 1} \quad \text{; Clear Bit Immediate, PORTB, bit 1}
  \end{align*}
  \]

- According to the manual, the SBI and CBI instructions each take 2 clock cycles
- \( 16\text{MHz} \Rightarrow 62.5\text{ns/cycle}, \) so pulse will be high for 125\text{ns}

Check with the oscilloscope

Extend the pulse

- At 125\text{ns}, the E pulse it not long enough although it might work on some boards.
- Can use \_delay\_us() or \_delay\_ms() functions but these are longer than needed since the minimum delay is 1 \text{us} (=1000 ns) and we only need 230 \text{ns}
- Trick for extending the pulse by a little bit:
  
  \[
  \begin{align*}
  &\text{PORTB} \mid= (1 << \text{PB1}); \quad \text{\# Set E to 1} \\
  &\text{PORTB} \mid= (1 << \text{PB1}); \quad \text{\# Add another 125\text{ns} to the pulse} \\
  &\text{PORTB} \&= \sim(1 << \text{PB1}); \quad \text{\# Clear E to 0}
  \end{align*}
  \]
6.33 Making Things Work Together

Better looking pulse

![Graph showing a pulse with a better looking waveform.]

6.34 Making Things Work Together

Extend the pulse (geek way)

- Use the “asm” compiler directive to embed low level assembly code within the C code.
- The AVR assembly instruction “NOP” does nothing, and takes 1 cycle to do it.

```c
PORTB |= (1 << PB1); // Set E to 1
asm("nop":);        // NOP delays another 62.5ns
asm("nop":);
PORTB &= ~(1 << PB1); // Clear E to 0
```

6.35 Making Things Work Together

Don’t guess that things will work

- When working with a device, make sure you know what types of signals it needs to see
  - Voltage
  - Current
  - Polarity (does 1 mean enable/true or does 0)
  - Duration (how long the signal needs to be valid)
  - Sequence (which transitions comes first, etc.)
- Have the manufacturer’s datasheet for the device available
  - Most of it can be ignored, but some parts are critical
  - Learn how to read it
- When in doubt ➔ follow the acronym used industry-wide: RTFM (read the *!@^-ing manual)