Other I/O

- LCD display
- Flash ROM
- SPI EPROM
- Keyboard (PS/2)
- UART connectors
- DAC
- ADC

LCD Display

- 2-line, 16 character LCD display
  - 4-bit interface
  - Relatively easy to use once you have it mapped into your processor’s memory-mapped I/O
  - Send characters to it, they show up on the screen
  - Not fast!
    - Scrolling at half-second intervals is about as fast as you can go and still have a clear display

### LCD

- 2-line, 16 character LCD display
- 4-bit interface
- Relatively easy to use once you have it mapped into your processor’s memory-mapped I/O
- Send characters to it, they show up on the screen
- Not fast!
  - Scrolling at half-second intervals is about as fast as you can go and still have a clear display

### LCD Control

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>FPGA Pin</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF clk</td>
<td>M3</td>
<td>Data in EOP</td>
</tr>
<tr>
<td>SF clk</td>
<td>PE0</td>
<td>Data in EOP</td>
</tr>
<tr>
<td>SF clk</td>
<td>R0</td>
<td>Data in EOP</td>
</tr>
<tr>
<td>SF clk</td>
<td>R1</td>
<td>Data in EOP</td>
</tr>
<tr>
<td>LCD E</td>
<td>M3</td>
<td>Read/Write Enable-Pulse</td>
</tr>
<tr>
<td>LCD E</td>
<td>R0</td>
<td>Read/Write Enable-Pulse</td>
</tr>
<tr>
<td>LCD E</td>
<td>R1</td>
<td>Read/Write Enable-Pulse</td>
</tr>
<tr>
<td>LCD E</td>
<td>R2</td>
<td>Read/Write Enable-Pulse</td>
</tr>
<tr>
<td>LCD E</td>
<td>W0</td>
<td>Read/Write Enable-Pulse</td>
</tr>
<tr>
<td>LCD E</td>
<td>W1</td>
<td>Read/Write Enable-Pulse</td>
</tr>
</tbody>
</table>

- 2-line, 16 character LCD display
- 4-bit interface
- Relatively easy to use once you have it mapped into your processor’s memory-mapped I/O
- Send characters to it, they show up on the screen
- Not fast!
  - Scrolling at half-second intervals is about as fast as you can go and still have a clear display
LCD Control

- Three memory areas inside LCD
  - DD RAM – memory to hold the characters being displayed
  - Two rows of 16 characters to display
  - Also 24 extras per line that can be scrolled
  - CG ROM – Pre-defined character map
    - 192 pre-defined characters
  - CG RAM – RAM to hold 8 custom characters
    - 5x8 bit character/glyphs

Table 6.1: Character LCD Interface

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Pin Assignments</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF (D+1)</td>
<td>225</td>
<td>Data I/O CDI</td>
</tr>
<tr>
<td>SF (D-1)</td>
<td>226</td>
<td>Data I/O CDB</td>
</tr>
<tr>
<td>SF (D-0)</td>
<td>227</td>
<td>Data I/O CDB</td>
</tr>
<tr>
<td>SF (D-0)</td>
<td>228</td>
<td>Data I/O CDB</td>
</tr>
<tr>
<td>LCD CE</td>
<td>229</td>
<td>Enable/Disable Pulse</td>
</tr>
</tbody>
</table>

For example, 8’h53 = S

Also, notice the 8 CG RAM locations
- Addresses 8’h00 to 8’h07

DD RAM

- DD RAM – memory to hold the characters being displayed
  - Data written to each of these locations is the 8-bit address of a character in the CG ROM/RAM

CG ROM/RAM

- For example, 8’h53 = S
  - Note the Japanese kana characters...
  - Also, notice the 8 CG RAM locations
    - Addresses 8’h00 to 8’h07
**CG RAM**

- This example is custom character 0'h03
  - Note that there are 8 rows in custom character 3
  - So, it takes 8 writes to make a custom character
  - Row address is incremented automatically…

**Operation Overview**

- Pick an LCD screen location
  - Write an 8-bit character address to that location
  - Then it shows up on the screen
- Pick a CG RAM location
  - Write 8 bytes starting at that location
  - Now you can use that new custom character
- Do it all with just a four-bit interface…
  - Lots of little nibble writes….
Command Set

Entry Mode Set

Sets the cursor move direction and specifies whether or not to shift the display.

These operations are performed during data reads and writes.

Execution Time: 40 µs

Bit DB1: (F) Increment/Decrement

0 Auto-decrement address counter; Cursor/Blank moves to left.
1 Auto-increment address counter; Cursor/Blank moves to right.

This bit either auto-increments or auto-decrements the DE RAM and CG RAM address counter by one location, after each Write Data to CG RAM or DE RAM or Read Data from CG RAM or DE RAM command. The cursor or Blank position moves accordingly.

Bit DB0: (E) Shift

0 Writing disabled.
1 During a DE RAM write operation, shift the entire display value in the direction controlled by Bit DB1 (F). Appears as though the cursor position remains constant and the display moves.

Write Timing

Memory Mapped I/O

- Commands are sent upper-nibble first

<table>
<thead>
<tr>
<th>Function</th>
<th>Upper Nibble</th>
<th>Lower Nibble</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Display</td>
<td>0 1</td>
<td>0 1</td>
</tr>
<tr>
<td>Smart Cursor Home</td>
<td>0 0 0 1 0 0 0 0</td>
<td>0 1</td>
</tr>
<tr>
<td>Scroll Mode Set</td>
<td>0 0 0 0 1 1 0 1</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>Display On/Off</td>
<td>0 0 0 0 1 0 0 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>Cursor and Display Shift</td>
<td>0 0 0 0 1 1 0 1</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>Functional Set</td>
<td>0 0 0 0 1 1 0 1</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>Set CG RAM Address</td>
<td>0 0 0 0 1 0 0 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>Set DE RAM Address</td>
<td>0 0 0 0 1 1 0 1</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>Read/Write Reg Address</td>
<td>0 0 0 0 1 1 0 1</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>Write DE RAM</td>
<td>0 0 0 0 1 0 0 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>Write CG RAM</td>
<td>0 0 0 0 1 0 0 0</td>
<td>0 0 1 0</td>
</tr>
</tbody>
</table>

So, as a practical matter, the easiest way to deal with the LCD is to map the interface to a memory-mapped location.

- Now you can, under program control, change the values on the data and control wires

Writing to the address of the LCD Reg will update its value
Initialization

Power-On Initialization

The initialization sequence first establishes that the FPGA application wishes to use the four-bit data interface to the LCD as follows:

- Wait 15 ms or longer, although the display is generally ready when the FPGA finishes configuration. The 15 ms interval is 750,000 clock cycles at 50 MHz.
- Write $ \text{SF}_{D:11:80} = 0\text{x}3$, pulse LCD_E High for 12 clock cycles.
- Wait 4.1 ms or longer, which is 205,000 clock cycles at 50 MHz.
- Write $ \text{SF}_{D:11:80} = 0\text{x}3$, pulse LCD_E High for 12 clock cycles.
- Wait 100 μs or longer, which is 5,000 clock cycles at 50 MHz.
- Write $ \text{SF}_{D:11:80} = 0\text{x}3$, pulse LCD_E High for 12 clock cycles.
- Wait 3 μs or longer, which is 200 clock cycles at 50 MHz.
- Write $ \text{SF}_{D:11:80} = 0\text{x}2$, pulse LCD_E High for 12 clock cycles.
- Wait 30 μs or longer, which is 2,000 clock cycles at 50 MHz.

Remember, this display is SLOW compared to 50MHz!!!

Configuration

Display Configuration

After the power-on initialization is completed, the four-bit interface is now established. The next part of the sequence configures the display:

- Issue a Function Set command, $0\text{x}2$, to configure the display for operation on the Spearmint SE Starter Kit board.
- Issue an Entry Mode Set command, $0\text{x}0$, to set the display to automatically increment the address pointer.
- Issue a Display On/Off command, $0\text{x}C$, to turn the display on and disable the cursor and blinking.
- Finally, issue a Clear Display command. Allow at least 1.64 ms (32,000 clock cycles) after issuing this command.

Using the Display

To write data to the display, specify the start address, followed by one or more data values. Before writing any data, issue a Set DDRAM Address command to specify the initial 7-bit address in the DDRAM. See Figure 4.1 for DDRAM locations.

Write data to the display using a Write Data to DDRAM or DDRAM command. The 8-bit data value represents the lookup address into the CG-ROM or CG-RAM, shown in Figure 4.1. The stored bitmap in the CG-ROM or CG-RAM drives the 5 x 8 dot matrix that represents the associated character.

If the address counter is configured to auto-increment, as described earlier, the application can sequentially write multiple character codes and each character is automatically stored and displayed in the next available location.

Continuing to write characters, however, eventually falls off the end of the first display line. The additional characters do not automatically appear on the second line because the DDRAM map is not consecutive from the first line to the second.

Remember timing!

- The LCD_E enable pulse must be high for at least 230ns (12 clock cycles at 50MHz)
- The two nibbles must be separated by 1μs (50 cycles)
- Two different commands must be separated by 40μs (2000 cycles)
- But, these are easily done in an assembly language program... (as are the even longer configuration delays)
Strata Flash

- 16 MByte (8 Mword) flash ROM
  - Designed to hold configuration data for the Spartan part
  - But, can be used for general non-volatile data

Some data lines are shared with the LCD

- But, if you don’t read back from the LCD they can both work together

Writing to the Strata Flash

- Tricky!
- Luckily, there is reference design on the Xilinx web site that implements a Flash programmer
  - You can use this to load data to your board
  - See class web site in the xilinx examples directory
  - www.eng.utah.edu/~3710/xilinx-docs/examples
  - s3esk_picoblaze_nor_flash_programmer

Xilinx Flash Project
Reading from the Flash

- Not as tricky
  - But, the flash has a 75ns access time
  - So, it will take four 50MHz cycles to read data
  - Each cycle is 20ns
  - Set SF_oe and SF_ce active (low) and wait for four cycles (80ns) before grabbing return data...
  - As usual map the flash into your processor’s memory-mapped address space

Xilinx Example

```
SF_byte_read: OUTPUT s9, SF_addr_hi_port
OUTPUT s8, SF_addr_si_port
OUTPUT s7, SF_addr_lo_port
LOAD s1, 05
OUTPUT s1, SF_control_port
LOAD s1, 06
INPUT s0, SF_data_in_port
OUTPUT s1, SF_control_port
RETURN
```

Read Waveforms

Figure 9: Single Word Asynchronous Read Waveform

- 75ns
SPI Serial Flash

- 16Mbit – SPI serial protocol
- Mostly used for Xilinx configuration
  - But, you can use it for data if you want to
- You can program it using the Impact tool
- As with all Flash – reading is (relatively) easy, writing is more complex
  - In this case, reading one byte takes 40 clock ticks…

Serial Output

- Two pins: Clk and Data
  - New data presented at Data pin on every clock
  - Looks like a shift register
**SPI Serial Flash**

32 clocks before data starts coming back (runs up to 75MHz)

Then 8 more ticks to get the data (MSB first)

---

**PS/2 Keyboard Interface**

- Standard keyboard interface
  - Serial protocol similar to UART, but with its own clock
  - When you press a key, the keyboard sends a “make code” (one, sometimes two, bytes)
  - When you release the key, the keyboard sends a “break code” (two, sometimes three, bytes)
  - Collectively, these are known as “scan codes”

---

**Table 4. Instruction set**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>One-byte instruction code</th>
<th>Address bytes</th>
<th>Dummy bytes</th>
<th>Data bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREN</td>
<td>Write Enable</td>
<td>0000 0110</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WREN</td>
<td>Write Disable</td>
<td>0000 0100</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RDID</td>
<td>Read Identification</td>
<td>1001 1111</td>
<td>0</td>
<td>0</td>
<td>1 to 10</td>
</tr>
<tr>
<td>RCSTR</td>
<td>Read Status Register</td>
<td>0000 0101</td>
<td>0</td>
<td>0</td>
<td>1 to 10</td>
</tr>
<tr>
<td>WRSR</td>
<td>Write Status Register</td>
<td>0000 0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>READ</td>
<td>Read Data Bytes</td>
<td>0000 0011</td>
<td>0</td>
<td>3</td>
<td>1 to 20</td>
</tr>
<tr>
<td>FAST_READ</td>
<td>Read Data Bytes at Higher Speed</td>
<td>0000 1011</td>
<td>0</td>
<td>3</td>
<td>1 to 3</td>
</tr>
<tr>
<td>PP</td>
<td>Page Program</td>
<td>0000 0010</td>
<td>0</td>
<td>3</td>
<td>0 to 256</td>
</tr>
<tr>
<td>SE</td>
<td>Sector Erase</td>
<td>1101 1000</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>BE</td>
<td>Bulk Erase</td>
<td>1100 0111</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DP</td>
<td>Deep Power-down</td>
<td>1011 1001</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RES</td>
<td>Release from Deep Power-down,</td>
<td>1010 1011</td>
<td>0</td>
<td>3</td>
<td>1 to 10</td>
</tr>
<tr>
<td></td>
<td>and Read Electronic Signature</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1. Address bits A20 to A21 are Don’t care.
PS/2 Keyboard Interface

Codes are sent LSB first with odd parity.

Note that 11 bits are sent for each code: start, 8-data, odd parity, stop.

Scan Codes (Make Codes)

Break codes are the same, but prefixed with 0xF0 for example – Q break code is 0xF0 0x15, ← is E0 F0 74.

ASCII codes
### PS/2 Things to Keep in Mind

- When you press and hold a key, the make code is sent every 100ms or so.
- If no key is pressed, both clk and data are in their idle state.
- Probably want a PS/2 controller that grabs codes and puts them in a register that can be read by your program (memory mapped I/O).
- Probably want to set a bit that says “new code” that gets cleared when the code is read.

### PS/2 Mouse

Whenever the mouse moves it sends three bytes.

Status tells you state of buttons, sign of X and Y, and overflow for X and Y.

### UART

Two main parts:
- Connectors
- Voltage translator

You provide the UART circuit!

(See 3700 UART for details)

### UART Basics

Two main parts: Connectors Voltage translator

9600 * 8 = 76.8kHz

50MHz/651 = 76.805kHz
UART Basics

<table>
<thead>
<tr>
<th>$B_{0}$</th>
<th>$B_{1}$</th>
<th>$B_{2}$</th>
<th>$B_{3}$</th>
<th>$B_{4}$</th>
<th>$B_{5}$</th>
<th>$B_{6}$</th>
<th>$B_{7}$</th>
<th>$B_{8}$</th>
<th>$B_{9}$</th>
<th>$B_{10}$</th>
<th>$B_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>NUL</td>
<td>DLE</td>
<td>SP</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>p</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>SOH</td>
<td>DC1</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>STX</td>
<td>DC2</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>ETX</td>
<td>DC3</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>EOT</td>
<td>DC4</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>ENQ</td>
<td>NAK</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>ACK</td>
<td>STX</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>BEL</td>
<td>ET0</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>BS</td>
<td>CAN</td>
<td>8</td>
<td>H</td>
<td>Y</td>
<td>h</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>HU</td>
<td>EM</td>
<td>9</td>
<td>I</td>
<td>Z</td>
<td>i</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>LF</td>
<td>SUB</td>
<td>10</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>VT</td>
<td>ESC</td>
<td>11</td>
<td>K</td>
<td>k</td>
<td>k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>FF</td>
<td>FS</td>
<td>12</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td>\</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1101</td>
<td>CR</td>
<td>GS</td>
<td>13</td>
<td>M</td>
<td>\</td>
<td>m</td>
<td>\</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1110</td>
<td>SO</td>
<td>RS</td>
<td>14</td>
<td>N</td>
<td>a</td>
<td>n</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>SI</td>
<td>US</td>
<td>15</td>
<td>O</td>
<td>s</td>
<td>o</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use rcv-req as a flag to be read by your program?
Assert xmt-req by your program to initiate send?

Digital to Analog Converter

- Four-channel 12-bit DAC
- Serial SPI protocol - up to 50MHz
- 32-bit data format

SPI ADC

Figure 9.2: Digital-to-Analog Connection Schematics
SPI ADC

Other SPI Parts

- Remember to disable the other SPI devices...

<table>
<thead>
<tr>
<th>Signal</th>
<th>Disabled Device</th>
<th>Disable Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_SG_B</td>
<td>SPI serial Flash</td>
<td>1</td>
</tr>
<tr>
<td>AMP_CS</td>
<td>Programmable pre-amplifier</td>
<td>1</td>
</tr>
<tr>
<td>A2_CONV</td>
<td>Analog-to-Digital Converter (ADC)</td>
<td>0</td>
</tr>
<tr>
<td>SF_CE0</td>
<td>Serial Flash Parallel Flash PROM</td>
<td>1</td>
</tr>
<tr>
<td>FPGA_INIT_B</td>
<td>Platform Flash PROM</td>
<td>1</td>
</tr>
</tbody>
</table>

Analog Capture

- Programmable scaling pre-amplifier
- 14-bit ADC
- SPI interface to both of them

Analog to Digital Converter
Summary

- All I/O can be mapped into your memory space
  - You have lots of room left over in the addressable space if you use block RAMs only
  - Might need custom FSMs to actually talk to the I/O
- Control the devices under program control
  - Some memory locations will be data, some will be control
  - Writing or reading these locations will have I/O side effects
  - Remember to consider timing!
- Think about how your program will interact with I/O

Memory Map

<table>
<thead>
<tr>
<th>Word addresses</th>
<th>I/O Switches/LEDs</th>
<th>UART</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 - 3FFF</td>
<td>Top two address</td>
<td>Glyphs?</td>
</tr>
<tr>
<td></td>
<td>bits define</td>
<td></td>
</tr>
<tr>
<td></td>
<td>regions?</td>
<td></td>
</tr>
<tr>
<td>4000 - 3FFF</td>
<td>Code/Data</td>
<td></td>
</tr>
<tr>
<td>0000 - 1FFFF</td>
<td>Code/Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16k words</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(32k bytes)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Block RAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frame buffer?</td>
<td></td>
</tr>
</tbody>
</table>