Table of Contents

- User’s manual
- Functional Specification
- Block Diagram
- Protoboard Schematic
- Board Layout
- Timing Diagrams
  - VRAM
    - Timing Specifications
    - Read, Write, Refresh, Data Transfer
  - LCD
    - Timing Specifications
    - LCD Signals
  - SRAM
    - Read, Write
  - Rom
    - Read
- Memory Map
- Pin Assignment
- Quartus Design
  - Top Level Design
  - Analog Control Block
  - LCD/VRAM Control Block
  - LCD Signal Generator
  - Keypad Debouncer
  - Clock Divider
- VHDL
  - VRAM Controller State Machine
    - State machine diagram
    - vram_controller_lut.vhd
  - split.vhd, combine.vhd, combine_16_16.vhd
- Software
  - Drivers/Initialization
    - start.s
    - keypad.s, display.s, interrupts.s
    - constants.s
  - OS
    - stubfuncs.c
    - interfac.h
Digital Oscilloscope

User’s Manual

Introduction

This digital oscilloscope is an easy-to-use instrument for visualizing analog input signals. In fact, such great care has been taken to ensure ease of use that the user’s task of finding and attaching an analog voltage signal source has been removed entirely. Simply attach the oscilloscope to a computer, open up your $2500+ copy of Quartus, program the FPGA design into it, and you’re ready to view the beautiful, noise-free built-in simulated signal. Even better, there is no need to try to figure out how switch to another input signal because this oscilloscope has been specialized to display only this signal!

Hardware

The hardware devices you use to interact with and monitor the oscilloscope are described on the next page.
<table>
<thead>
<tr>
<th>Device Name</th>
<th>Device Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keypad</td>
<td>4*4 Array of 16 keys</td>
<td>Of the 16 keys, only 5 are used:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>![Keypad Image]</td>
</tr>
<tr>
<td>Display</td>
<td>LCD Panel</td>
<td>A “high” quality LCD to display the oscilloscope’s output.</td>
</tr>
<tr>
<td>Status LED</td>
<td>LED</td>
<td>A shiny red LED that lights when system has booted to let you know that initialization has completed successfully and the scope interface should be running.</td>
</tr>
</tbody>
</table>

**User Interface**

Once programmed, the system boots into Normal mode with a 100ns sampling rate and a positive slope mid-level (2.480V) trigger with no delay. x- and y-axes are displayed behind the trace, and a menu in the upper right-hand corner—demonstrated in the screenshot above—lets you change all of the scope’s settings and enter different sampling modes. To hide and show the menu, press the <Menu> key.
You can change from one highlighted menu item to the next using the <Up> and <Down> keys. To alter the highlighted configuration setting, the user presses the <Left> and <Right> keys to select a value.

The configurable settings and their values are as follows:

<table>
<thead>
<tr>
<th>Menu Title</th>
<th>Options and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Normal: Scope waits for a trigger after completing each retrace.</td>
</tr>
<tr>
<td></td>
<td>Automatic: Scope waits for a trigger after each retrace, but retriggers automatically</td>
</tr>
<tr>
<td></td>
<td>automatic without a trigger event after a delay.</td>
</tr>
<tr>
<td></td>
<td>One-Shot: Scope triggers only once, continuing to display the last trace captured.</td>
</tr>
<tr>
<td>Scale</td>
<td>Scale Axes: Display x and y axes along with trace.</td>
</tr>
<tr>
<td></td>
<td>Scale Grid: Display x-y grid along with the trace.</td>
</tr>
<tr>
<td></td>
<td>Scale Off: Display the trace only.</td>
</tr>
<tr>
<td>Sweep</td>
<td>Items are time quantities that specify the sweep rate, or the time between successive</td>
</tr>
<tr>
<td></td>
<td>samples options: 100ns, 200ns, 500ns, 1us, 2us, 5us, 10us, 20us, 50us, 100us, 200us,</td>
</tr>
<tr>
<td></td>
<td>500us, 1ms, 2ms, 5ms, 10ms, 20ms</td>
</tr>
<tr>
<td>Level</td>
<td>128 voltage quantities are given as choices for the trigger voltage level. The levels</td>
</tr>
<tr>
<td></td>
<td>are equally spaced from 0V to 5V.</td>
</tr>
<tr>
<td>Slope</td>
<td>Slope +: Scope is triggered on a positive slope (rising edge)</td>
</tr>
<tr>
<td></td>
<td>Slope -: Scope is triggered on a negative slope (falling edge)</td>
</tr>
<tr>
<td>Delay</td>
<td>Time quantities ranging between 0ms and 1ms specify the time between the trigger and</td>
</tr>
<tr>
<td></td>
<td>the start of the trace.</td>
</tr>
</tbody>
</table>

**Limitations**

If for some reason you have decided to go through the effort of finding a voltage source you want to measure, you will find that the oscilloscope has no connector to which you can attach the signal. The oscilloscope is compatible only with the built-in demonstration trace.
Digital Oscilloscope

Functional Specification

Description: The system is an interactive digital oscilloscope displaying on a 640 x 200 LCD panel. The signal is input via an analog line in, which runs through an analog to digital converter. There is a keypad to change the operation parameters. The system must be connected to a computer to program the FPGA, but can be disconnected once booted.

Inputs: Input is received through a keypad, a serial connection, and an analog input signal.

<table>
<thead>
<tr>
<th>Input Name</th>
<th>Input Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keypad</td>
<td>4*4 Array of 16 keys</td>
<td>Of the 16 keys, only 5 are used by the OS: ![Keypad Diagram]</td>
</tr>
<tr>
<td>Serial</td>
<td>RS-232 Port</td>
<td>Allows communication with the NIOS using GERMS. Downloading of sample data to the oscilloscope from the computer is not yet implemented.</td>
</tr>
<tr>
<td>Analog</td>
<td>Analog Line In</td>
<td>Analog signal line whose voltage variation with time the scope measures. This portion of the oscilloscope is not yet implemented.</td>
</tr>
</tbody>
</table>

Outputs: Output is implemented with a display, a serial connection, and a LED.

<table>
<thead>
<tr>
<th>Output Name</th>
<th>Output Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display</td>
<td>LCD Panel</td>
<td>A 640 x 200 graphics panel used to display the analog input and to present an operation and configuration menu to the user.</td>
</tr>
<tr>
<td>Serial</td>
<td>RS-232 Port</td>
<td>Allows sampled data to be uploaded to a</td>
</tr>
</tbody>
</table>
computer via a serial connection. This feature is not yet implemented.

| Status LED | LED | LED lights when system has booted to inform user that the software initialization has completed successfully and the OS should be running. |

User Interface: By default, the system boots into Normal mode with a 100ns sampling rate and a positive slope mid-level (2.480V) trigger with no delay. x- and y-axes are displayed behind the trace, and a menu in the upper right-hand corner lets the user change all of the scope’s settings and enter different sampling modes. The user can press the menu key to toggle the menu’s visibility.

The user changes from one highlighted menu item to the next using the up and down keys. To alter the highlighted configuration setting, the user presses the left and right keys to select a value.

Serial transfers will be implemented with “silent linking.” The computer will send a request to transmit data or a request to receive data, and the oscilloscope will comply without asking the user to confirm. A status message for the transfer will be displayed in the lower left-hand corner.

The menu options are as follows:

<table>
<thead>
<tr>
<th>Menu Title</th>
<th>Options and Description</th>
</tr>
</thead>
</table>
| Mode       | Normal: Scope waits for a trigger after completing each retrace.  
            | Automatic: Scope waits for a trigger after each retrace, but retriggers automatically without a trigger event after a delay.  
            | One-Shot: Scope triggers only once, continuing to display the last trace captured. |
| Scale      | Scale Axes: Display x and y axes along with trace.  
            | Scale Grid: Display x-y grid along with the trace.  
            | Scale Off: Display the trace only. |
| Sweep      | Items are time quantities that specify the sweep rate, or the time between successive samples options:  
            | 100ns, 200ns, 500ns, 1us, 2us, 5us, 10us, 20us, 50us, 100us, 200us, 500us, 1ms, 2ms, 5ms, 10ms, 20ms |
| Level      | 128 voltage quantities are given as choices for the trigger voltage level. The levels are equally spaced from 0V to 5V. |
| Slope      | Slope +: Scope is triggered on a positive slope (rising edge)  
            | Slope -: Scope is triggered on a negative slope (falling edge) |
| Delay      | Time quantities ranging between 0ms and 1ms specify the time between the trigger and the start of the trace. |
Error Handling: If a key not mapped to an interface action in the current mode of operation is pressed, then no operation occurs. No other errors are anticipated for the current implementation.
Once analog and serial input is implemented, the following error handling will apply:

- If a buffer overflow occurs on the signal input queue, “Signal Buffer Overflow” is displayed in the lower left-hand corner, and no further input is accepted. If a serial overflow occurs, “Serial Buffer Overflow” is displayed. If an error with framing, parity, a break, or some other serial error occurs, the messages “Framing Error”, “Parity Error”, “Break Error”, and “Serial Error” are displayed respectively.

Algorithms: A FFT may be implemented in the future for data analysis purposes. Currently no FFT is used.
A look up table is used to validate the user’s keypresses.

Data Structures: A circular queue will be used to buffer serial input in case bytes are received faster than the NIOS can respond. A FIFO will also be used to buffer the signal input from the analog to digital converter/analog control block.

Limitations: There is currently no analog or serial input.
Timing Diagrams

- VRAM
  - Timing Specifications
  - Read, Write, Refresh, Data Transfer
- LCD
  - Timing Specifications (1180F, 1190)
  - LCD Signals
- SRAM
  - Read, Write
- Rom
  - Read
<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>SYMBOL</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column address hold time after RAS low</td>
<td>t_{AR}</td>
<td>80ns</td>
<td></td>
</tr>
<tr>
<td>Column address setup time</td>
<td>t_{ASC}</td>
<td>0ns</td>
<td></td>
</tr>
<tr>
<td>Row address setup time</td>
<td>t_{ASR}</td>
<td>0ns</td>
<td></td>
</tr>
<tr>
<td>Access time from CAS</td>
<td>t_{CAC}</td>
<td></td>
<td>60ns</td>
</tr>
<tr>
<td>Column address hold time</td>
<td>t_{CAH}</td>
<td>20ns</td>
<td></td>
</tr>
<tr>
<td>CAS pulse width</td>
<td>t_{CAS}</td>
<td>60ns</td>
<td></td>
</tr>
<tr>
<td>DT low hold time after RAS low</td>
<td>t_{CDH}</td>
<td>40ns</td>
<td></td>
</tr>
<tr>
<td>CAS before RAS refresh hold time</td>
<td>t_{CHR}</td>
<td>25ns</td>
<td></td>
</tr>
<tr>
<td>CAS precharge time (page cycle only)</td>
<td>t_{CP}</td>
<td>50ns</td>
<td></td>
</tr>
<tr>
<td>CAS precharge time (nonpage cycle)</td>
<td>t_{CPN}</td>
<td>25ns</td>
<td></td>
</tr>
<tr>
<td>CAS high to RAS low precharge time</td>
<td>t_{CRP}</td>
<td>10ns</td>
<td></td>
</tr>
<tr>
<td>CAS hold time</td>
<td>t_{CSH}</td>
<td>120ns</td>
<td></td>
</tr>
<tr>
<td>CAS before RAS refresh setup time</td>
<td>t_{CSR}</td>
<td>10ns</td>
<td></td>
</tr>
<tr>
<td>CAS to WE delay</td>
<td>t_{CWD}</td>
<td>100ns</td>
<td></td>
</tr>
<tr>
<td>Write command to CAS lead time</td>
<td>t_{CWL}</td>
<td>40ns</td>
<td></td>
</tr>
<tr>
<td>Data-in hold time</td>
<td>t_{DH}</td>
<td>35ns</td>
<td></td>
</tr>
<tr>
<td>DT high hold time</td>
<td>t_{DHH}</td>
<td>20ns</td>
<td></td>
</tr>
<tr>
<td>Data-in hold time after RAS low</td>
<td>t_{DHR}</td>
<td>95ns</td>
<td></td>
</tr>
<tr>
<td>DT high setup time</td>
<td>t_{DHS}</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### V RAM Timing Specifications continued...

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>SYMBOL</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT low setup time</td>
<td>$t_{DLS}$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Data-in setup time</td>
<td>$t_{DS}$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DT high to CAS high delay</td>
<td>$t_{DTC}$</td>
<td>10ns</td>
<td></td>
</tr>
<tr>
<td>DT high hold time after RAS high</td>
<td>$t_{DTH}$</td>
<td>20ns</td>
<td></td>
</tr>
<tr>
<td>DT high to RAS high delay</td>
<td>$t_{DTR}$</td>
<td>10ns</td>
<td></td>
</tr>
<tr>
<td>OE pulse width</td>
<td>$t_{OE}$</td>
<td>35ns</td>
<td></td>
</tr>
<tr>
<td>Access time from OE</td>
<td>$t_{OEA}$</td>
<td>30ns</td>
<td></td>
</tr>
<tr>
<td>OE to data-in setup delay</td>
<td>$t_{OED}$</td>
<td>35ns</td>
<td></td>
</tr>
<tr>
<td>OE hold time after WE low</td>
<td>$t_{OEH}$</td>
<td>30ns</td>
<td></td>
</tr>
<tr>
<td>OE to RAS inactive setup time</td>
<td>$t_{OES}$</td>
<td>10ns</td>
<td></td>
</tr>
<tr>
<td>Output disable time from OE high</td>
<td>$t_{OEZ}$</td>
<td>0</td>
<td>30ns</td>
</tr>
<tr>
<td>Output disable time from CAS high</td>
<td>$t_{OFF}$</td>
<td>0</td>
<td>30ns</td>
</tr>
<tr>
<td>Page cycle time</td>
<td>$t_{PC}$</td>
<td>120ns</td>
<td></td>
</tr>
<tr>
<td>Access time from RAS</td>
<td>$t_{RAC}$</td>
<td>120ns</td>
<td></td>
</tr>
<tr>
<td>Row address hold time</td>
<td>$t_{RAH}$</td>
<td>15ns</td>
<td></td>
</tr>
<tr>
<td>RAS pulse width</td>
<td>$t_{RAS}$</td>
<td>120ns</td>
<td>10000ns</td>
</tr>
<tr>
<td>Random read or write cycle time</td>
<td>$t_{RC}$</td>
<td>220ns</td>
<td></td>
</tr>
<tr>
<td>RAS to CAS delay time</td>
<td>$t_{RCD}$</td>
<td>25ns</td>
<td>60ns</td>
</tr>
<tr>
<td>Read command hold time after CAS high</td>
<td>$t_{RCH}$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>SYMBOL</td>
<td>MIN</td>
<td>MAX</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>--------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Read command setup time</td>
<td>( t_{RCS} )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DT low hold time after RAS low (serial port active)</td>
<td>( t_{RDH} )</td>
<td>100ns</td>
<td></td>
</tr>
<tr>
<td>Refresh interval</td>
<td>( t_{REF} )</td>
<td>4ns</td>
<td></td>
</tr>
<tr>
<td>RAS precharge time</td>
<td>( t_{RP} )</td>
<td>90ns</td>
<td></td>
</tr>
<tr>
<td>RAS high to CAS low precharge time</td>
<td>( t_{RPC} )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Read command hold after RAS high</td>
<td>( t_{RRH} )</td>
<td>20ns</td>
<td></td>
</tr>
<tr>
<td>RAS hold time</td>
<td>( t_{RSH} )</td>
<td>60ns</td>
<td></td>
</tr>
<tr>
<td>Read-write/read-modify-write cycle time</td>
<td>( t_{RWC} )</td>
<td>300ns</td>
<td></td>
</tr>
<tr>
<td>RAS to WE delay</td>
<td>( t_{RWD} )</td>
<td>160ns</td>
<td></td>
</tr>
<tr>
<td>to be defined</td>
<td>( t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write command to RAS lead time</td>
<td>( t_{RWL} )</td>
<td>40ns</td>
<td></td>
</tr>
<tr>
<td>SC pulse width</td>
<td>( t_{SCH} )</td>
<td>10ns</td>
<td></td>
</tr>
<tr>
<td>Serial output access time from SC</td>
<td>( t_{SCA} )</td>
<td>40ns</td>
<td></td>
</tr>
<tr>
<td>Serial clock cycle time</td>
<td>( t_{SCC} )</td>
<td>40ns</td>
<td>50000ns</td>
</tr>
<tr>
<td>SC precharge time</td>
<td>( t_{SCL} )</td>
<td>10ns</td>
<td></td>
</tr>
<tr>
<td>SC high to DT high delay</td>
<td>( t_{SDD} )</td>
<td>10ns</td>
<td></td>
</tr>
<tr>
<td>SC low hold time after DT high</td>
<td>( t_{SDH} )</td>
<td>10ns</td>
<td></td>
</tr>
<tr>
<td>Serial output access time from SOE</td>
<td>( t_{SOA} )</td>
<td>35ns</td>
<td></td>
</tr>
<tr>
<td>SOE pulse width</td>
<td>( t_{SOE} )</td>
<td>15ns</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>SYMBOL</td>
<td>MIN</td>
<td>MAX</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>--------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Serial output hold time after SC high</td>
<td>$t_{SOH}$</td>
<td>10ns</td>
<td></td>
</tr>
<tr>
<td>SOE low to serial output setup delay</td>
<td>$t_{SOO}$</td>
<td>5ns</td>
<td></td>
</tr>
<tr>
<td>SOE precharge time</td>
<td>$t_{SOP}$</td>
<td>15ns</td>
<td></td>
</tr>
<tr>
<td>Serial output disable time from SOE high</td>
<td>$t_{SOZ}$</td>
<td>0</td>
<td>30ns</td>
</tr>
<tr>
<td>Rise and fall transition time</td>
<td>$t_{T}$</td>
<td>3ns</td>
<td>50ns</td>
</tr>
<tr>
<td>Write-per-bit hold time</td>
<td>$t_{WBH}$</td>
<td>20ns</td>
<td></td>
</tr>
<tr>
<td>Write-per-bit setup time</td>
<td>$t_{WBS}$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Write command hold time</td>
<td>$t_{WCH}$</td>
<td>35ns</td>
<td></td>
</tr>
<tr>
<td>Write command hold time after RAS low</td>
<td>$t_{WCR}$</td>
<td>95ns</td>
<td></td>
</tr>
<tr>
<td>Write command setup time</td>
<td>$t_{WCS}$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Write bit selection hold time</td>
<td>$t_{WH}$</td>
<td>20ns</td>
<td></td>
</tr>
<tr>
<td>Write command pulse width</td>
<td>$t_{WP}$</td>
<td>35ns</td>
<td></td>
</tr>
<tr>
<td>Write bit selection setup time</td>
<td>$t_{WS}$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Actual RAS to CAS delay</td>
<td>$t_{RCD_Actual}$</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>DEFINITION</td>
<td>DESCRIPTION</td>
<td>MIN</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td><code>t_{CLC}</code></td>
<td></td>
<td>shift clock cycle</td>
<td>166</td>
</tr>
<tr>
<td><code>t_{WCLH}</code></td>
<td></td>
<td>Shift clock &quot;H&quot; width</td>
<td>63</td>
</tr>
<tr>
<td><code>t_{WCLL}</code></td>
<td></td>
<td>Shift clock &quot;L&quot; width</td>
<td>63</td>
</tr>
<tr>
<td><code>t_{DS}</code></td>
<td></td>
<td>Data setup time</td>
<td>50</td>
</tr>
<tr>
<td><code>t_{DH}</code></td>
<td></td>
<td>Data hold time</td>
<td>30</td>
</tr>
<tr>
<td><code>t_{WECH}</code></td>
<td></td>
<td>Enable clock &quot;H&quot; width</td>
<td>100</td>
</tr>
<tr>
<td><code>t_{WECL}</code></td>
<td></td>
<td>Enable clock &quot;L&quot; width</td>
<td>100</td>
</tr>
<tr>
<td><code>t_{EDS}</code></td>
<td></td>
<td>Enable data setup time</td>
<td>50</td>
</tr>
<tr>
<td><code>t_{EDH}</code></td>
<td></td>
<td>Enable data hold time</td>
<td>20</td>
</tr>
<tr>
<td><code>t_{EDR}</code></td>
<td></td>
<td>Enable clock delay time</td>
<td>-10</td>
</tr>
<tr>
<td><code>t_{ECS}</code></td>
<td></td>
<td>Enable clock setup time</td>
<td>70</td>
</tr>
<tr>
<td><code>t_{WLPH}</code></td>
<td></td>
<td>Latch pulse &quot;H&quot; width</td>
<td>110</td>
</tr>
<tr>
<td><code>t_{WLPL}</code></td>
<td></td>
<td>Latch pulse &quot;L&quot; width</td>
<td>220</td>
</tr>
<tr>
<td><code>t_{LT}</code></td>
<td></td>
<td>Latch timing</td>
<td>100</td>
</tr>
<tr>
<td><code>t_{LH}</code></td>
<td></td>
<td>Latch hold time</td>
<td>0</td>
</tr>
<tr>
<td><code>t_{LDS}</code></td>
<td></td>
<td>Latch pulse data setup time</td>
<td>140</td>
</tr>
<tr>
<td><code>t_{LDH}</code></td>
<td></td>
<td>Latch pulse data hold time</td>
<td>50</td>
</tr>
<tr>
<td><code>t_{DFR}</code></td>
<td></td>
<td>Permissible frame signal delay</td>
<td>-500</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>DEFINITION</td>
<td>DESCRIPTION</td>
<td>MIN</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------</td>
<td>-------------------</td>
<td>-----</td>
</tr>
<tr>
<td>( t_{PD} )</td>
<td>Enable output delay</td>
<td>20</td>
<td>150</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>DEFINITION</td>
<td>DESCRIPTION</td>
<td>MIN</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------</td>
<td>---------------</td>
<td>------</td>
</tr>
<tr>
<td>t_CYL</td>
<td>Latch pulse cycle time</td>
<td>to be defined</td>
<td>400</td>
</tr>
<tr>
<td>t_WLTH</td>
<td>Latch pulse &quot;H&quot; width</td>
<td>to be defined</td>
<td>180</td>
</tr>
<tr>
<td>t_WLL</td>
<td>Latch pulse &quot;L&quot; width</td>
<td>to be defined</td>
<td>180</td>
</tr>
<tr>
<td>t_CYC</td>
<td>Shift clock cycle time</td>
<td>to be defined</td>
<td>400</td>
</tr>
<tr>
<td>t_WCLH</td>
<td>Shift clock &quot;H&quot; time</td>
<td>to be defined</td>
<td>110</td>
</tr>
<tr>
<td>t_WCLL</td>
<td>Shift clock &quot;L&quot; time</td>
<td>to be defined</td>
<td>240</td>
</tr>
<tr>
<td>t_DS</td>
<td>Data setup time</td>
<td>to be defined</td>
<td>70</td>
</tr>
<tr>
<td>t_DH</td>
<td>Data hold time</td>
<td>to be defined</td>
<td>30</td>
</tr>
<tr>
<td>t_ST</td>
<td>Data shift timing</td>
<td>to be defined</td>
<td>0</td>
</tr>
<tr>
<td>t_STH</td>
<td>Data shift hold time</td>
<td>to be defined</td>
<td>125</td>
</tr>
<tr>
<td>t_DFR</td>
<td>Permissible frame signal delay</td>
<td>to be defined</td>
<td>500</td>
</tr>
<tr>
<td>t_PD</td>
<td>Data output delay time</td>
<td>to be defined</td>
<td>30</td>
</tr>
</tbody>
</table>
LCD Timing

- clk: 20MHz
- FR: Toggles every DI
- LP/YSCL/LAT: Once every 20 ECL
- ECL: Once every 16 XSCl
- XSCl: 3.333333MHz, New frame data latched in
- Data: [3..0]
- DI: Once every 100 LP
- DT_NOW: New frame data
### Memory Map

<table>
<thead>
<tr>
<th>Module</th>
<th>Type</th>
<th>Address Range</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>rom_0</td>
<td>ROM</td>
<td>0x00000000 - 0x0001FFFF</td>
<td></td>
</tr>
<tr>
<td>ram_0</td>
<td>RAM</td>
<td>0x00020000 - 0x0003FFFF</td>
<td></td>
</tr>
<tr>
<td>unichip_memory_0</td>
<td>Legacy</td>
<td>0x00040000 - 0x0005FFFF</td>
<td></td>
</tr>
<tr>
<td>uart_0</td>
<td>UART</td>
<td>0x00060000 - 0x0007FFFF</td>
<td></td>
</tr>
<tr>
<td>keydata_pio</td>
<td>PIC (Parallel I/O)</td>
<td>0x00080000 - 0x0009FFFF</td>
<td></td>
</tr>
<tr>
<td>key_ready_pio</td>
<td>PIC (Parallel I/O)</td>
<td>0x000A0000 - 0x000BFFFF</td>
<td></td>
</tr>
<tr>
<td>analog_control_settings_data_pio</td>
<td>PIC (Parallel I/O)</td>
<td>0x000C0000 - 0x000DFFFF</td>
<td></td>
</tr>
<tr>
<td>analog_control_settings_selector_pio</td>
<td>PIC (Parallel I/O)</td>
<td>0x000E0000 - 0x000FFFFF</td>
<td></td>
</tr>
<tr>
<td>analog_control_reset_pio</td>
<td>PIC (Parallel I/O)</td>
<td>0x000F0000 - 0x0010FFFF</td>
<td></td>
</tr>
<tr>
<td>analog_fifo_read_req_pio</td>
<td>PIC (Parallel I/O)</td>
<td>0x00110000 - 0x0012FFFF</td>
<td></td>
</tr>
<tr>
<td>analog_fifo_data_in_pio</td>
<td>PIC (Parallel I/O)</td>
<td>0x00130000 - 0x0014FFFF</td>
<td></td>
</tr>
<tr>
<td>analog_fifo_full_pio</td>
<td>PIC (Parallel I/O)</td>
<td>0x00150000 - 0x0016FFFF</td>
<td></td>
</tr>
<tr>
<td>analog_fifo_empty_pio</td>
<td>PIC (Parallel I/O)</td>
<td>0x00170000 - 0x0018FFFF</td>
<td></td>
</tr>
<tr>
<td>LED_pio</td>
<td>PIC (Parallel I/O)</td>
<td>0x00190000 - 0x001AFFFF</td>
<td></td>
</tr>
<tr>
<td>wran_0</td>
<td>VRAM</td>
<td>0x001B0000 - 0x001CFFFF</td>
<td></td>
</tr>
<tr>
<td>tri_state_bridge_0</td>
<td>Avalon</td>
<td>0x001D0000 - 0x001EFFFF</td>
<td></td>
</tr>
<tr>
<td>nios_0</td>
<td>Nios Processor</td>
<td>0x001F0000 - 0x0020FFFF</td>
<td></td>
</tr>
<tr>
<td>To</td>
<td>Location</td>
<td>General Function</td>
<td>Special Function</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>CLOCK</td>
<td>PIN_27</td>
<td>Dedicated Clock</td>
<td>CLK1</td>
</tr>
<tr>
<td>RESET</td>
<td>PIN_142</td>
<td>Row I/O</td>
<td>nRS</td>
</tr>
<tr>
<td>U4OE1</td>
<td>PIN_187</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>U4OE2</td>
<td>PIN_206</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>U6OE1</td>
<td>PIN_57</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>U6OE2</td>
<td>PIN_84</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>U6DIR2</td>
<td>PIN_85</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>U7DIR1</td>
<td>PIN_87</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>U7DIR2</td>
<td>PIN_108</td>
<td>Row I/O</td>
<td></td>
</tr>
<tr>
<td>U7OE1</td>
<td>PIN_88</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>U7OE2</td>
<td>PIN_107</td>
<td>Row I/O</td>
<td></td>
</tr>
<tr>
<td>U8DIR1</td>
<td>PIN_2</td>
<td>Row I/O</td>
<td></td>
</tr>
<tr>
<td>U8DIR2</td>
<td>PIN_31</td>
<td>Row I/O</td>
<td></td>
</tr>
<tr>
<td>U8OE1</td>
<td>PIN_5</td>
<td>Row I/O</td>
<td></td>
</tr>
<tr>
<td>U8OE2</td>
<td>PIN_30</td>
<td>Row I/O</td>
<td></td>
</tr>
<tr>
<td>U9DIR1</td>
<td>PIN_32</td>
<td>Row I/O</td>
<td></td>
</tr>
<tr>
<td>U9DIR2</td>
<td>PIN_55</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>U9OE1</td>
<td>PIN_33</td>
<td>Row I/O</td>
<td></td>
</tr>
<tr>
<td>U9OE2</td>
<td>PIN_54</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>U10OE1</td>
<td>PIN_112</td>
<td>Row I/O</td>
<td></td>
</tr>
<tr>
<td>U10OE2</td>
<td>PIN_158</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[0]</td>
<td>PIN_71</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[1]</td>
<td>PIN_72</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[2]</td>
<td>PIN_73</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[3]</td>
<td>PIN_74</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[4]</td>
<td>PIN_89</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[5]</td>
<td>PIN_90</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[6]</td>
<td>PIN_91</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[7]</td>
<td>PIN_92</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[8]</td>
<td>PIN_93</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[9]</td>
<td>PIN_94</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[10]</td>
<td>PIN_96</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[12]</td>
<td>PIN_98</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[13]</td>
<td>PIN_99</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[14]</td>
<td>PIN_100</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[15]</td>
<td>PIN_101</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[16]</td>
<td>PIN_102</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[17]</td>
<td>PIN_103</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>addr[18]</td>
<td>PIN_104</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>data[0]</td>
<td>PIN_58</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>data[1]</td>
<td>PIN_59</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>data[2]</td>
<td>PIN_60</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>data[3]</td>
<td>PIN_61</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>data[4]</td>
<td>PIN_62</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>data[5]</td>
<td>PIN_63</td>
<td>Column I/O</td>
<td></td>
</tr>
<tr>
<td>data[6]</td>
<td>PIN_65</td>
<td>Column I/O</td>
<td></td>
</tr>
</tbody>
</table>
Digital Oscilloscope APEX20K Pin Assignments

<table>
<thead>
<tr>
<th>Pin Description</th>
<th>Pin Number</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>data[7]</td>
<td>PIN_66</td>
<td>Column I/O</td>
</tr>
<tr>
<td>nCS_RAM</td>
<td>PIN_37</td>
<td>Row I/O</td>
</tr>
<tr>
<td>nCS_ROM</td>
<td>PIN_34</td>
<td>Row I/O</td>
</tr>
<tr>
<td>nRD</td>
<td>PIN_67</td>
<td>Column I/O</td>
</tr>
<tr>
<td>nWR</td>
<td>PIN_68</td>
<td>Column I/O</td>
</tr>
<tr>
<td>U6DIR1</td>
<td>PIN_56</td>
<td>Column I/O</td>
</tr>
<tr>
<td>TX1</td>
<td>PIN_17</td>
<td>Row I/O</td>
</tr>
<tr>
<td>RX2</td>
<td>PIN_7</td>
<td>Row I/O</td>
</tr>
<tr>
<td>keypad_row[120]</td>
<td>PIN_120</td>
<td>Row I/O</td>
</tr>
<tr>
<td>keypad_row[122]</td>
<td>PIN_122</td>
<td>Row I/O</td>
</tr>
<tr>
<td>keypad_row[123]</td>
<td>PIN_123</td>
<td>Row I/O</td>
</tr>
<tr>
<td>keypad_row[134]</td>
<td>PIN_134</td>
<td>Row I/O</td>
</tr>
<tr>
<td>keypad_col[140]</td>
<td>PIN_140</td>
<td>Row I/O</td>
</tr>
<tr>
<td>keypad_col[144]</td>
<td>PIN_144</td>
<td>Row I/O</td>
</tr>
<tr>
<td>keypad_col[151]</td>
<td>PIN_151</td>
<td>Row I/O</td>
</tr>
<tr>
<td>keypad_col[152]</td>
<td>PIN_152</td>
<td>Row I/O</td>
</tr>
<tr>
<td>U10DIR1</td>
<td>PIN_111</td>
<td>Row I/O</td>
</tr>
<tr>
<td>U10DIR2</td>
<td>PIN_161</td>
<td>Column I/O</td>
</tr>
<tr>
<td>STATUS_LE</td>
<td>PIN_117</td>
<td>Row I/O</td>
</tr>
<tr>
<td>U4DIR1</td>
<td>PIN_179</td>
<td>Column I/O</td>
</tr>
<tr>
<td>U4DIR2</td>
<td>PIN_207</td>
<td>Column I/O</td>
</tr>
<tr>
<td>vaddr[0]</td>
<td>PIN_44</td>
<td>Row I/O</td>
</tr>
<tr>
<td>vaddr[1]</td>
<td>PIN_204</td>
<td>Column I/O</td>
</tr>
<tr>
<td>vaddr[2]</td>
<td>PIN_203</td>
<td>Column I/O</td>
</tr>
<tr>
<td>vaddr[3]</td>
<td>PIN_202</td>
<td>Column I/O</td>
</tr>
<tr>
<td>vaddr[4]</td>
<td>PIN_201</td>
<td>Column I/O</td>
</tr>
<tr>
<td>vaddr[5]</td>
<td>PIN_200</td>
<td>Column I/O</td>
</tr>
<tr>
<td>vaddr[6]</td>
<td>PIN_198</td>
<td>Column I/O</td>
</tr>
<tr>
<td>vaddr[7]</td>
<td>PIN_197</td>
<td>Column I/O</td>
</tr>
<tr>
<td>vdata[0]</td>
<td>PIN_193</td>
<td>Column I/O</td>
</tr>
<tr>
<td>vdata[1]</td>
<td>PIN_194</td>
<td>Column I/O</td>
</tr>
<tr>
<td>vdata[2]</td>
<td>PIN_195</td>
<td>Column I/O</td>
</tr>
<tr>
<td>vdata[3]</td>
<td>PIN_196</td>
<td>Column I/O</td>
</tr>
<tr>
<td>nRAS</td>
<td>PIN_45</td>
<td>Row I/O</td>
</tr>
<tr>
<td>nCAS</td>
<td>PIN_46</td>
<td>Row I/O</td>
</tr>
<tr>
<td>nDTOE</td>
<td>PIN_49</td>
<td>Row I/O</td>
</tr>
<tr>
<td>nWBWE</td>
<td>PIN_50</td>
<td>Row I/O</td>
</tr>
<tr>
<td>VRAM_SC</td>
<td>PIN_51</td>
<td>Row I/O</td>
</tr>
<tr>
<td>DI</td>
<td>PIN_38</td>
<td>Row I/O</td>
</tr>
<tr>
<td>DisplayEna</td>
<td>PIN_40</td>
<td>Row I/O</td>
</tr>
<tr>
<td>FR</td>
<td>PIN_41</td>
<td>Row I/O</td>
</tr>
<tr>
<td>LP</td>
<td>PIN_113</td>
<td>Row I/O</td>
</tr>
<tr>
<td>ECL</td>
<td>PIN_115</td>
<td>Row I/O</td>
</tr>
<tr>
<td>YSCL</td>
<td>PIN_42</td>
<td>Row I/O</td>
</tr>
</tbody>
</table>
Quartus Design

- Top Level Design
- Analog Control Block
- LCD/VRAM Control Block
- LCD Signal Generator
- Keypad Debouncer
- Clock Divider
The analog input hardware is not constructed, so take the input to be a constant zero.

Interrupt line for keypad input
Control line for the status LED.
8.2 (Serial out), 9.1 (OE, WE, LCD Control lines)
RAM and ROM Chip Select Lines
Output: Buffers 6.2 (Address/CS lines), 7 (Address)
4.2 (VRAM address)
Input: Buffers 8.1 (Serial in), 10.2 (Keypad column)
9.2 (VRAM control), 10 (LED, Keypad row)

System Clock (20MHz)
System Reset
Input lines for reading undebounced row of keypress data from the keypad
Serial input data line for NIOS terminal
8-bit Input from the ADC
Keypad row enable lines
Debounced keypresses (with key repeat), and interrupt the NIOS when one is debounced.
When requested, sample data from the ADC at a specified sample rate.
Store ADC samples until software can read and process them.
Generate signals to control the VRAM and the LCD, and provide an interface between the Avalon bus and the VRAM's bus.
Debounced key data
VCCRX2 INPUT
VCCRESET INPUT
VCCCLOCK INPUT
VCCkeypad_col[3..0] INPUT
U4OE1OUTPUT
U4OE2OUTPUT
TX1OUTPUT
U6OE1OUTPUT
U6OE2OUTPUT
U7OE1OUTPUT
U7OE2OUTPUT
U8OE1OUTPUT
U8OE2OUTPUT
U9OE1OUTPUT
U9OE2OUTPUT
U8DIR1OUTPUT
U6DIR2OUTPUT
U7DIR1OUTPUT
U7DIR2OUTPUT
U8DIR2OUTPUT
U9DIR1OUTPUT
nCS_ROMOUTPUT
nCS_RAMOUTPUT
nRDOUTPUT
nWROUTPUT
addr[18..0]OUTPUT
U6DIR1OUTPUT
keypad_row[1]OUTPUT
keypad_row[2]OUTPUT
keypad_row[3]OUTPUT
keypad_row[0]OUTPUT
STATUS_LEDOUTPUT
U10OE1OUTPUT
U10OE2OUTPUT
U10DIR2OUTPUT
U10DIR1OUTPUT
vaddr[7..0]OUTPUT
nCASOUTPUT
nRASOUTPUT
nWBWEOUTPUT
nDTOEOUTPUT
U4DIR1OUTPUT
U4DIR2OUTPUT
VRAM_SCOUTPUT
U9DIR2OUTPUT
ECLOUTPUT
LPOUTPUT
YSCLOUTPUT
FROUTPUT
DIOUTPUT
DisplayEnableOUTPUT
VCC data[7..0]BIDIR
VCC vdata[3..0]BIDIR
Analog Control Block

Save data from the ADC at a specified rate after a set delay

The settings are altered by selecting the desired setting by placing a 3-bit identifier on the selector line:

1 == Sample rate, 2 == Trigger delay, 3 == Trigger level, 4 == Trigger slope

Then the value for that setting is placed on the SETTING_DATA line.

This control block interacts with the FIFO, storing sample data on SAMPLE_CLK's rising edge by requesting that the FIFO latch in data from the SAMPlE_DATA output using its SAMPLE_REQ line.

System clock (20MHz)

Value to be used for the selected setting. 1: Set sample rate 2: Set trigger delay 3: Set trigger level 4: Set trigger slope

Only the low 7 bits are needed for trigger level/slope. Only reset trigger delay counter on first trigger event since reset (Must be on the trigger, not the sample clock!)
State machine implementing VRAM cycles.

VRAM Control Lines:
- VCCnWR INPUT
- VCCnRD INPUT
- VCCnCS INPUT
- VCCclk INPUT
- VCCnRS INPUT
- VCCADDR[18..0] INPUT
- nCASOUTPUT
- nRASOUTPUT
- nWBWEOUTPUT
- nDTOEOUTPUT
- vdata[3..0]OUTPUT
- vaddr[7..0]OUTPUT

Display Enable OUTPUT

LCD Control Lines:
- DisplayEnable OUTPUT
- (Display enabled when no reset is occurring)

Avalon Bus Ctl Line (Wait Request)

Collection of counters that implement pulses to be sent to the LCD and computes the address of the row the VRAM needs to transfer into its serial shift register.

LCD/VRAM Block
Implements reads, writes, data transfers, and refreshes for the VRAM using a finite state machine implemented in VHDL.
Generates the signals needed to drive the LCD using a block design file with counters and comparators.

Virtual Memory Access (VRAM)

System Clock (20 MHz)

Avalon Bus Ctl Line (Wait Request)
The LCD Signal Generator generates the timing signals to be sent to the LCD and the row address used during a VRAM Data Transfer.

- **XSCL** has a period of 6 clocks.
  - It is high for 3 clocks and low for 3 clocks.

- **ECL** has a period of 96 clocks.
  - It is high for 2 to 9 clocks at the start of its period.

- **DT** has a period of 960 clocks.
  - It is high for clocks 2 to 7 at its period beginning.

- **LP** has a period of 1920 clocks.
  - It is high from clocks 6 to 13 at the start of its period.

- **DI** has a period of 192000 clocks.
  - It is high for clocks 12 to 17 at the start of its period.

- **ECL** is high for clocks 2 to 9 at the start of its period.
  - It is high from clock 6 to 13 in the next period.

There are 200 rows to transfer to the LCD in total. Move to the next row each LP (so the inverted LP signal is used as a clock).

Asynchronous clear is used because the clock event will never occur upon reset. Make sure all signals change on system clock edges and prevent glitching with flip-flops.

The timing diagram shows the timings for each signal. See the LCD timing diagram for the timings that this design implements.
This design file implements a keypad debouncer. The implementation debounces a keypress for 20ms and repeats a key every subsequent 300ms if the user continues to hold it.

The debouncer supports multiple keypresses on a single row, but does not respond to simultaneous keypresses occurring on different rows. In this case, the first row with a keypress encountered during row multiplexing is the only row from which keypresses are debounced.

When a key is debounced, key_ready will pulse high for 1ms, and the encoded presses can be read from key_data[6..0].

This debouncer assumes that the clock runs at 20MHz. If it doesn’t, the debounce and key repeat times will differ from the stated values.

Key is debounced after 20ms have passed without the row nibble changing. Key will repeat after being held for 300ms, since the counter reduces (mod 300).

Divide clock by 20000 so that it cycles every 1 ms.

Detect changes in the row nibble, which will reset the debouncing counter.
Divide scale_factor by two to determine the number of counts the divided clock should be high, and how many counts it should be low. We want (scale_factor / 2) distinct counter values, so we count up to (scale_factor / 2) - 1. Known issues: if scale_factor is odd, the divided clock cycle will occur in (scale_factor - 1) ticks (rounds down to nearest even integer). A scale_factor of 1 is not allowed, as it will cause an overflow in the computation of (scale_factor / 2) - 1.
VHDL

- VRAM Controller State Machine
  - State machine diagram
  - vram_controller_lut.vhd
- Miscellaneous Helper Blocks
  - split.vhd
  - combine.vhd
  - combine_16_16.vhd
Actions executed upon state entry.

**VRAM Read Cycle**
- **READ ONE**
  - Assert Row Addr
  - nDT High
  - nCAS high
  - nWE High
- **READ TWO**
  - nRAS low
- **READ THREE**
  - Assert Col Addr
- **READ FOUR**
  - nCAS low
  - nOE low
- **READ FIVE**
  - Transfer Data
  - WR low
- **READ SIX**
  - nRAS high
  - nCAS high
  - nOE high
  - WR high
- **DT ONE**
  - Assert Row Addr
  - nDT Low
  - nCAS high
- **DT TWO**
  - nRAS low
- **DT THREE**
  - Assert Col Addr
- **DT FOUR**
  - nCAS low
- **DT FIVE**
  - nDT high

**VRAM Write Cycle**
- **WRITE ONE**
  - Assert Row Addr
  - Assert Write Mask
  - nCAS high
  - nWB high
  - nDT low
  - nRD high
- **WRITE TWO**
  - nRAS low
- **WRITE THREE**
  - Assert Input Data
  - Assert Col Addr
- **WRITE FOUR**
  - nCAS low
- **WRITE FIVE**
  - WAIT
- **WRITE SIX**
  - nRAS high
  - nCAS high
  - nWB high
  - WR low
- **REFRESH ONE**
  - nCAS low
- **REFRESH TWO**
  - nRAS low
- **REFRESH THREE**
  - nCAS high
- **REFRESH FOUR**
  - nCAS high
- **REFRESH FIVE**
  - nCAS low
- **REFRESH SIX**
  - WAIT

**VRAM Data Transfer Cycle**
- **DT ONE**
  - Assert Row Addr
  - nDT Low
  - nCAS high
- **DT TWO**
  - nRAS low
- **DT THREE**
  - Assert Col Addr
- **DT FOUR**
  - nCAS low
- **DT FIVE**
  - nDT high
-- Oscilloscope VRAM Controller
--
-- This VHDL file implements a controller for the NEC uPD41264-12 VRAM chip. The controller is implemented using a state machine whose transitions are a function of the input lines. Output signals change upon transition to a new state, and their values are looked up in a table indexed by current state number.
--
-- The controller uses a wait request line to halt the NIOS CPU until it is able to complete a read or write, since a refresh cycle or data transfer cycle could delay read/write cycles an amount unpredictable to the Avalon Tristate Bus.
--
-- See the state diagram for a visual overview of the state machine.
-- Inputs and outputs are enumerated and described in the port section below.
--
-- Revision History:
--      10/23/08    Julian Panetta  Initial Revision
--      11/16/08    Julian Panetta  Reformatted to fit within 80 characters wide
--
---------------------------------------------
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity vram_controller_lut is
  port (
    addr : in std_logic_vector(18 downto 0);  -- 16-bit address input (row & column addresses) with 3 disregarded most significant bits so it interfaces with 19-bit shared Avalon bus
    data : inout std_logic_vector(7 downto 0);  -- 8-bit data bus (data / writemask nibbles)
    nWR : in std_logic;  -- Write enable
    nRD : in std_logic;  -- Output enable
    clk : in std_logic;  -- System clock (Assumed to be 20MHz)
    nCS : in std_logic;  -- Chip select
    DT_SOON : in std_logic;  -- Data transfer needs to occur before a new cycle can be completed. ('1' == true, '0' == false)
    DT_NOW : in std_logic;  -- Data transfer needs to occur NOW ('1' == true, '0' == false)
    nRS : in std_logic;  -- reset state machine to known, IDLE state
    DT_ROW : in std_logic_vector(7 downto 0);  -- Row address to be used for the data transfer cycle. (The data transfer column is always 0.)
  );
end entity;
-- 4-bit data bus to VRAM
vdata  :  inout std_logic_vector(3 downto 0);

-- 8-bit address bus out to VRAM
vaddr  :  out std_logic_vector(7 downto 0);

-- CAS signal to VRAM
nCAS   :  out std_logic;

-- RAS signal to VRAM
nRAS   :  out std_logic;

-- nWB/nWE signal to VRAM
nWBWE  :  out std_logic;

-- nDT/nOE signal to VRAM
nDTOE  :  out std_logic;

-- Wait request signal for NIOS
WR     :  out std_logic);

end vram_controller_lut;

architecture Moore_machine of vram_controller_lut is
constant  IDLE     :  integer range 0 to 23 := 0;
-- VRAM Read Cycle States
constant  READ_ONE :  integer range 0 to 23 := 1;
constant  READ_TWO :  integer range 0 to 23 := 2;
constant  READ_THREE :  integer range 0 to 23 := 3;
constant  READ_FOUR :  integer range 0 to 23 := 4;
constant  READ_FIVE :  integer range 0 to 23 := 5;
constant  READ_SIX :  integer range 0 to 23 := 6;
-- VRAM Write Cycle States
constant  WRITE_ONE :  integer range 0 to 23 := 7;
constant  WRITE_TWO :  integer range 0 to 23 := 8;
constant  WRITE_THREE :  integer range 0 to 23 := 9;
constant  WRITE_FOUR :  integer range 0 to 23 := 10;
constant  WRITE_FIVE :  integer range 0 to 23 := 11;
constant  WRITE_SIX :  integer range 0 to 23 := 12;
-- VRAM Refresh Cycle States
constant  REFRESH_ONE :  integer range 0 to 23 := 13;
constant  REFRESH_TWO :  integer range 0 to 23 := 14;
constant  REFRESH_THREE :  integer range 0 to 23 := 15;
constant  REFRESH_FOUR :  integer range 0 to 23 := 16;
constant  REFRESH_FIVE :  integer range 0 to 23 := 17;
-- VRAM Data Transfer Cycle States
constant  DTRAN_ONE :  integer range 0 to 23 := 18;
constant  DTRAN_TWO :  integer range 0 to 23 := 19;
constant  DTRAN_THREE :  integer range 0 to 23 := 20;
constant  DTRAN_FOUR :  integer range 0 to 23 := 21;
constant  DTRAN_FIVE :  integer range 0 to 23 := 22;
constant  DTRAN_SIX :  integer range 0 to 23 := 23;

-- LUT for control line outputs. 0 => nRAS, 1 => nCAS, 2 => nWBWE, 3 => nDTOE, 4 => WR
type TABLE is array(0 to 23) of std_logic_vector(4 downto 0);
constant OUTPUT_BITS :  TABLE := (  
-- IDLE control line values
"11111",  
-- READ_* control line values
"11111", "11110", "11110", "10100", "00100", "11111","
-- WRITE_* control line values
"11011", "11010", "11010", "11000", "11000", "01111",

-- REFRESH_* control line values
"11101", "11100", "11100", "11100", "11111",

-- DTRAN_* control line values
"10111", "10110", "10110", "10100", "11100", "11111");

signal CurrentState : integer range 0 to 23; -- current state
signal NextState : integer range 0 to 23; -- next state

begin
  -- Compute the next state (function of current state and inputs) using
  -- concurrent statements
  NextState <=
    -- Transitions to IDLE
    IDLE when (nRS = '0')
    else IDLE when (CurrentState = READ_SIX or
                    CurrentState = WRITE_SIX or CurrentState = REFRESH_FIVE or
                    CurrentState = DTRAN_SIX)
    -- Idle State transitions
    else READ_ONE when (CurrentState = IDLE and nRD = '0' and
                        DT_SOON = '0' and nCS = '0')
    else WRITE_ONE when (CurrentState = IDLE and nWR = '0' and
                         DT_SOON = '0' and nCS = '0')
    else REFRESH_ONE when (CurrentState = IDLE and nCS = '1' and
                           DT_SOON = '0')
    else DTRAN_ONE when (CurrentState = IDLE and DT_NOW = '1')
    else IDE when (CurrentState = IDLE)
    -- Movement Within Cycles
    else CurrentState + 1;

  -- Upon transition into the new state, update the control line values
  -- using the output lookup table.
  output_computation : process (CurrentState)
  begin
    -- Look up and set control line values for the current state
    nRAS <= OUTPUT_BITS(CurrentState)(0);
    nCAS <= OUTPUT_BITS(CurrentState)(1);
    nWBWE <= OUTPUT_BITS(CurrentState)(2);
    nDTOE <= OUTPUT_BITS(CurrentState)(3);
    WR <= OUTPUT_BITS(CurrentState)(4);

    -- VRAM address line access
    -- Assert row address for the first and second read and write states.
    if (CurrentState = READ_ONE or CurrentState = WRITE_ONE or
        CurrentState = READ_TWO or CurrentState = WRITE_TWO) then
      vaddr <= addr(15 downto 8);
    else
      -- Assert col address for the third/fourth read and write states.
      if (CurrentState = READ_THREE or CurrentState = READ_FOUR or
          CurrentState = WRITE_THREE or CurrentState = WRITE_FOUR) then
        vaddr <= addr(7 downto 0);
      else
        -- Assert DT row address for the first and second DT states.
        if (CurrentState = DTRAN_ONE or CurrentState = DTRAN_TWO) then
          vaddr <= DT_ROW;
        else
          -- Output zero address in all other states.
          vaddr <= "00000000";
        end if;
      end if;
    end if;
  end process output_computation;
-- VRAM data bus access
-- Transfer VRAM data to Avalon Bus for the fifth read state.
if (CurrentState = READ_FIVE) then
    data <= "0000" & vdata;
else
    -- Assert write mask for the first and second write state.
    if (CurrentState = WRITE_ONE or CurrentState = WRITE_TWO) then
        vdata <= data(7 downto 4);
    else
        -- Assert graphics nibble for the third and fourth write state.
        if (CurrentState = WRITE_THREE or CurrentState = WRITE_FOUR) then
            vdata <= data(3 downto 0);
        else
            -- In all other states, the Avalon data bus and the VRAM
            -- data bus should be high impedance.
            vdata <= "ZZZZ";
            data <= "ZZZZZZZZ";
        end if;
    end if;
end if;
end if;

end process output_computation;

-- Make actual transition to the new state on rising clock edge
make_transition : process (clk)
begin
    if clk = '1' then
        -- Only change on the rising clock edge
        CurrentState <= NextState;
        -- Transition to the new state
        end if;
end process;

end Moore_machine;
-- Split
-- Splits the 16-bit input bits read from port a evenly into the 8 LSBs (placed on "low" output line) and the 8 MSBs (placed on "high" output line).
--
-- Revision History:
-- 09/05/08 Julian Panetta Initial Revision

Library ieee;
use ieee.std_logic_1164.all;

entity split is
  port
    ( a : in std_logic_vector(15 downto 0);
      low : out std_logic_vector(7 downto 0);
      high : out std_logic_vector(7 downto 0)
    );
end split;

architecture dataflow of split is
begin
  low <= a(7 downto 0);
  high <= a(15 downto 8);
end dataflow;
Library ieee;
use ieee.std_logic_1164.all;

entity combine is
   port
     (a : in std_logic_vector(1 downto 0);
b : in std_logic_vector(3 downto 0);
c : out std_logic_vector(5 downto 0)
);
end combine;

architecture dataflow of combine is
begin
c <= a & b;
end dataflow;
Library ieee;
use ieee.std_logic_1164.all;

entity combine_16_16 is
  port ( 
    a : in std_logic_vector(15 downto 0);
    b : in std_logic_vector(15 downto 0);
    c : out std_logic_vector(31 downto 0)
  );
end combine_16_16;

architecture dataflow of combine_16_16 is
begin
  c <= a & b;
end dataflow;
Software

- Drivers/Initialization (NIOS Assembly)
  - start.s
    - Initializes hardware and calls oscilloscope operating system main loop.
    - Global symbols: _start
  - keypad.s
    - Interrupt-driven driver for the keypad
    - Global symbols: install_key_handler, getkey, key_available
  - display.s
    - Code for drawing on and clearing the LCD
    - Global symbols: plot_pixel, clear_display
  - interrupts.s
    - Code for initializing interrupts
    - Global symbols: init_interrupts
  - constants.s
    - Include file holding constants used throughout the NIOS assembly files listed above.

- OS (C)
  - stubfunes.c
    - Functions emulating hardware not yet implemented
    - Global symbols: set_sample_rate, set_trigger, set_delay, start_sample, sample_done
  - interfac.h
    - Include file holding constants used throughout the oscilloscope operating system.
File: start.s
Description:
Code run when the NIOS resets. Performs hardware initialization and then
branches to Glen George's oscilloscope code.
Input:   Keypresses from the user, analog signal input (not implemented)
Output:   Output displayed
Known Bugs:   None
Revision History:
11/04/08    Julian Panetta Initial Revision
11/17/08    Julian Panetta Added more extensive documentation

.include "excalibur.s"
.include "constants.s"

.text
.global _start

#include "excalibur.s"
.include "constants.s"

.text
.global _start

void _start(void)
Description:
Code located at the NIOS' reset location. Initializes hardware and then
runs the oscilloscope OS.
Operation:
- Installs the keypad handler, initialize_interrupts, and forces interrupts
to be enabled.
- Turns on the status LED.
- Branches to the oscilloscope's main loop from which the NIOs should never
return.
Arguments: None
Return Values: None
Inputs:   Keypresses read from keypad
Outputs:   User interface displayed on LCD
Registers Destroyed: None (Registers SAVED, no supervisor/caller anyway).
Shared Variables: None
Data Structures/Algorithms: None

_start:
    SAVE    %sp, 0      ; No stack space is needed
    _BSR    install_key_handler
    NOP
    _BSR    init_interrupts
    NOP
    ; Turn on interrupts
    ; According to the NIOS Programmer's Manual, interrupts should be enabled
    ; by default and need only be enabled if explicitly disabled previously.
    ; However, I have found the interrupts don't function if they aren't
    ; enabled as done by the following two lines. These lines were copied
    ; verbatim from the Programmer's Manual.
    PFX     9
    WRCTL   %g0
    ; Turn on the LED to indicate the system has booted
    MOVI    %l1, LED_ON
MOVIA %10, na_LED_pio
PFX np_piodata
ST [%10], %11

_BSR main ; Run Glen's oscilloscope code
NOP

; This line should never be reached.
RESTRET
::: Handwritten content

File: keypad.s
Description:
Routines to allow Glen George's oscilloscope software interface with the debounced keypad.
Input: Debounced keypresses from the user (ISR run when keypress made).
Output: None.

Known Bugs: None
Revision History:
09/06/08 Julian Panetta Initial Revision
11/17/08 Julian Panetta Added documentation and replaced magic numbers with constants.

.include "excalibur.s"
.include "constants.s"

.data
key_ready: .4byte false
key_data: .4byte KEY_ILLEGAL

.text
.global install_key_handler
.global getkey
.global key_available

; Install and initialize the isr that is run when a key is debounced.
; Installs ISR.
; Clears key_ready and key_data variables.
; Clears edge capture register so previous keypresses aren't detected.

; Arguments: None
; Return Values: None
; Outputs: None
; Registers Destroyed: None (Registers SAVED/RESTORED)

install_key_handler:
SAVE %sp, 0 ; no stack space is needed
MOVIA %o0, na_key_ready_pio_irq ; select key_ready interrupt
MOVIA %o1, key_handler@h ; load ISR address
MOVIA %o2, 0 ; context (ignored)
BSR nr_installuserisr ; install our keypad isr
NOP

; Initialize key_ready and key_data words to zero
MOVIA %l2, false
MOVIA %l1, key_ready
ST [%l1], %l2 ; No key is ready initially
keypad.s

MOVIA  %l2, KEY_ILLEGAL  ; (Not needed as false == KEY_ILLEGAL)
MOVIA  %l1, key_data  ; No valid key is encoded initially
ST  [%l1], %l2

; Clear the edge capture register to ensure a keypress made
; before loading the code doesn't result in an interrupt.
MOVIA  %l1, na_key_ready_pio
MOVI  %l0, false
PFX np_pioedgecapture
ST  [%l1], %l0

RESTRET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; void key_handler(void)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   ISR that is run when a key is pressed. Simply records the presence of a key
;   and resets the interrupt event.
; Operation:
;   - Clears the edge capture register so the event doesn't fire until a new
;     keypress is made.
;   - Sets key_ready and stores debounced key data in key_data.
;
; Arguments: None
; Return Values: None
;
; Inputs:   Keys pressed by the user trigger this handler
; Outputs:  None
;
; Registers Destroyed: None (Registers SAVED/RESTORED)
; Shared Variables:
;   key_ready   -- (Set) Boolean value recording the occurrence of a keypress
;   key_data    -- (Stored) Byte recording the value of the keypress
;
; Data Structures/Algorithms: None
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
key_handler:
SAVE  %sp, 0  ; no stack space is needed
; clear the edge capture register
MOVIA  %l1, na_key_ready_pio
MOVI  %l0, 0
PFX np_pioedgecapture
ST  [%l1], %l0

MOVIA  %l0, key_ready
MOVI  %r0, true
STBS  [%l0], %r0, 0  ; Indicate a debounced key was received

MOVIA  %l0, na_keydata_pio
PFX np_piodata
LD  %l1, [%l0]

MOVIA  %l0, key_data
ST  [%l0], %l1

RESTORE
TRET  %o7

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; unsigned char key_available(void)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   Returns 0x01 if a key press is ready for reading and 0x00 if not.
; Operation:
;   - Reads key_ready and extracts bit 1 as the return value.
;
; Arguments: None
; Return Values: None
;
; Inputs: None
; Outputs: None
;
; Registers Destroyed: None (Registers SAVED/RESTORED)
; Shared Variables:
;   key_ready   -- (Read) Boolean value recording the occurrence of a keypress
;
; Data Structures/Algorithms: None

key_available:
    SAVE  %sp, 0          ; No stack space is needed
    MOVIA %o1, key_ready
    LD   %o0, [%o1]
    MOVI %o1, true
    AND  %o0, %o1        ; Check if a keypress is ready
    RESTRET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

int getkey(void)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;   Waits until a valid keypress is made (if one hasn't been already) and then
;   returns that keypress' code.

; Operation:
;   - Waits until a keypress has been made.
;   - Performs a lookup to see if that key is valid.
;   - Returns the key if it is valid, otherwise waits for another keypress.

; Arguments: None
; Return Values: None
;
; Inputs: Valid keys pressed by the user are returned by this routine.
; Outputs: None
;
; Registers Destroyed: None (Registers SAVED/RESTORED)
; Shared Variables:
;   key_ready   -- (Read) Boolean value recording the occurrence of a keypress
;   key_data    -- (Read) Byte recording the value of the keypress
;
; Data Structures/Algorithms: Look up table is used to validate keypresses

getkey:
    SAVE  %sp, 0          ; no stack space is needed
    MOVIA %l0, key_ready
    MOVIA %l3, key_data
    MOVI %l2, true
    wait:
    LD   %l1, [%l0]       ; Check key is ready
    AND  %l1, %l2
    IFS  cc_z 
    BR    wait           ; wait until a key is ready
    NOP
    MOVI %l1, false
    ST   [%l0], %l1      ; clear key_ready
LD %i0, [%l3] ; get key_data
EXT8D %i0, %l1 ; put key_data in low byte of return value

MOVIA %l4, valid_keys
validate_loop:
LD %l5, [%l4]

CMP %i0, %l5 ; valid key?
IFS cc_eq ; valid key?
BR valid ; return it.
NOP

CMPI %l5, KEY_ILLEGAL ; invalid key?
IFS cc_eq ; invalid key?
BR wait ; wait for a valid one.
NOP

BR validate_loop ; not sure yet!
ADDI %l4, 1 ; move to the next key entry
valid:
    RESTRET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; valid_keys:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;    Constant table of valid keypress values terminated by KEY_ILLEGAL.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
valid_keys:
    .word KEY_UP
    .word KEY_DOWN
    .word KEY_LEFT
    .word KEY_RIGHT
    .word KEY_MENU
    .word KEY_ILLEGAL ; KEY_ILLEGAL Terminates the list of valid keys
display.s

;---------------------------------
; File: display.s
; Description:
; Routines to set individual pixels on the LCD and clear the entire LCD.
; Input: None.
; Output: Pixels displayed on LCD.
; Known Bugs: None
; Revision History:
; 11/04/08 Julian Panetta Initial Revision
;---------------------------------

.include "excalibur.s"
.include "constants.s"

.text
.global plot_pixel
.global clear_display

;---------------------------------
; void plot_pixel(unsigned int x, unsigned int y, int p)
;---------------------------------
; Description:
; Sets the pixel at the passed (x, y) to the value p.
; The origin of the screen coordinate system is the upper left corner.
; Operation:
; - Computes the VRAM address of the nibble holding the desired pixel.
; - Adds this offset to the VRAM base address to compute the absolute address.
; - Creates a bit mask for the VRAM write so that only the desired pixel of
;   the nibble is modified.
; - Sets the write data to all 1's for p == PIXEL_BLACK and all 0's for
;   p == PIXEL_WHITE. This ensures the selected pixel is set to the correct
;   value, regardless of which of the four pixels it is.
; - Sends these computed write mask and data nibbles to the VRAM
; Arguments (After execution of SAVE):
; %i0 = x coordinate of desired pixel
; %i1 = y coordinate of desired pixel
; %i2 = p = color to paint specified pixel
; Return Values: None
; Outputs: Pixel at (x, y) painted with color p.
; (Corresponding bit in VRAM is set)
; Registers Destroyed: None (Registers SAVED/RESTORED)
; Stack depth: 0
; Shared Variables: None
; Data Structures/Algorithms: None
;---------------------------------

plot_pixel:
    SAVE   %sp, 0 ; no stack space is required

    ; Load VRAM base address
    MOVIA   %l0, na_vram_0

    ; Compute offset from VRAM base address:
    ; Offset = (VRAM_row << 8) + VRAM_column
    ;       = (y << 8) + (x >> 2)
    LSLI   %i1, 8       ; move y coordinate into the row portion of offset
    ADD    %l0, %i1
MOV   %l1, %i0    ; Get a copy of the x coordinate
LSR   %l1, 2      ; get the nibble in which the pixel appears
ADD   %10, %l1    ; Add in the column portion of the nibble offset
MOVI  %l2, 3
AND   %l0, %l2    ; get the x offset within that nibble
BGEN  %l1, 7      ; set the leftmost pixel enable bit
LSR   %l1, %i0    ; shift the pixel enable bit into the 'x' location
; within the write mask

; If the color, p, is PIXEL_WHITE, set all the bits in the graphics nibble
; to '0' (to clear the pixel). Otherwise, set all the bits in the graphics
; nibble to '1' (blacken the pixel). Assuming the PIXEL_WHITE value is 0
; and the PIXEL_BLACK value is 0x0F, p can simply be ORed into the data
; byte.
OR    %l1, %i2
FILL8 %r0, %l1    ; Fill all bytes of this word with the computed
; graphics byte so ST8D will read it in all cases
ST8D  [%l0], %r0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; void clear_display(void)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   Sets all the LCD pixels to white.
; Operation:
;   Iterates over the portions of VRAM mapped to pixels on the LCD, and paints
;   them all white 16 pixels (4 nibbles) at a time.
;
; Arguments: None
; Return Values: None

; Outputs: All pixels on the LCD are painted white.
; (All bits in VRAM that are displayed on the LCD are set to '0')
;
; Registers Destroyed: None
; Stack depth: 0
; Shared Variables: None
; Data Structures/Algorithms: None
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
clear_display:
SAVE  %sp, 0              ; no stack space is required

; Load VRAM base address
MOVIA %l0, na_vram_0
MOVIA %l3, CLEAR_PATTERN   ; Load pattern to clear 4 graphics nibbles

MOVIA %l1, SIZE_Y         ; loop over all LCD rows
MOVIA %l4, ROW_ADVANCE    ; Ammount added to address to advance row
row_loop:
    MOVIA %l2, LCD_NIBBLES_WIDE
    loop over all the nibbles in the row

col_loop:
    ST   [%l0], %l3          ; Clear 4 nibbles (16 pixels) of the LCD
    SUBI %l2, NIBBLES_PER_WRITE
    ; Subtract nibbles cleared from remaining
    IFS cc nz
    BR   col_loop            ; Continue until there are none remaining
ADDI %10, COLUMN_ADVANCE ; Move to the next set of 4 nibbles
SUBI %11, ROWS_PER_LOOP ; Subtract rows cleared from rows remaining
IFS cc_nz
BR row_loop ; Continue until there are no rows remaining
ADD %10, %14 ; Advance to the start of the next row

RESTRET
File: interrupts.s
Description:
Routine to enable interrupts for the NIOS CPU.
The only interrupts acknowledged are from the keypad, so
only the na_key_ready_pio's interrupt need be enabled.
Input: None.
Output: None.

Known Bugs: None

Revision History:
09/06/08 Julian Panetta Initial Revision

.include "excalibur.s"

.text
.global init_interrupts

Description:
Enables all the interrupts to be acknowledged by the NIOS.
This should only be called AFTER the ISRs to handle these interrupts have
already been installed.
Operation:
- Sets the key_ready pio's interrupt mask bit to '1' so that the debounced
  key bit line can interrupt the NIOS' operation.

Arguments: None
Return Values: None
Outputs: None
Registers Destroyed: None (Registers SAVED/RESTORED)
Stack depth: 0
Shared Variables: None
Data Structures/Algorithms: None

init_interrupts:
    SAVE  %sp, 0 ; No stack space is needed
    MOVIA %10, na_key_ready_pio
    MOVI %11, 1
    STP [%10, np_piointerruptmask], %11
    RESTRET
; File: constants.s
; Description:
;   Defines constants used by the keypad and display hardware drivers.
;   Also defines global software constants and constants related to the status
;   LED.
; Revision History:
;   11/17/08    Julian Panetta  Initial Revision

; Global Constants
.equ true, 1       ; Boolean true value
.equ false, 0      ; Boolean false value

; Status LED Constants
.equ LED_ON, 1     ; Value to write to LED PIO to turn on Status LED
.equ LED_OFF, 0    ; Value to write to LED PIO to turn off Status LED

; Display Constants
.equ PIXEL_BLACK, 0x0F  ; Pixel color passed to draw black pixels.
.equ PIXEL_WHITE, 0x00  ; Pixel color passed to draw white pixels.
.equ CLEAR_PATTERN, 0xF0F0F0F0 ; Word value that will clear 16 bits of VRAM

; VRAM/LCD Properties
.equ SIZE_X, 640      ; X resolution of the LCD
.equ SIZE_Y, 200      ; Y resolution of the LCD
.equ LCD_NIBBLES_WIDE, (SIZE_X / 4); Number of nibbles in VRAM per LCD row
.equ VRAM_WIDTH, 256  ; Number of nibbles in VRAM per VRAM row

; Constants used for clearing the LCD efficiently
.equ NIBBLES_PER_WRITE, 4 ; Number of nibbles cleared per write
.equ COLUMN_ADVANCE, NIBBLES_PER_WRITE ; Number of nibbles to skip over to
; advance to the next column
.equ ROWS_PER_LOOP, 1 ; Number of rows cleared per iteration
.equ ROW_ADVANCE, (VRAM_WIDTH - LCD_NIBBLES_WIDE); Number of nibbles to skip
; to advance to the next row

; Keypad Constants
.equ KEY_UP, 0x02 ; <Up>
equ KEY_DOWN, 0x22 ; <Down>
equ KEY_LEFT, 0x11 ; <Left>
equ KEY_RIGHT, 0x14 ; <Right>
equ KEY_MENU, 0x12 ; <Menu>
equ KEY_ILLEGAL, 0x00 ; Illegal Key
/* This file contains stub functions for the hardware interfacing code for the Digital Oscilloscope project. The file is meant to allow linking of the main code without necessarily having all of the low-level functions or hardware working. The functions included are:
key_available   - check if a key is available
getKey           - get a key
clear_display   - clear the display
plot_pixel      - plot a pixel
set_sample_rate - set the sample rate
set_trigger     - set the trigger level and slope
set_delay       - set the trigger delay
start_sample    - start sampling
sample_done     - sampling status

The local functions included are:
none

The locally global variable definitions included are:
none

Revision History
3/8/94   Glen George       Initial revision.
3/13/94  Glen George       Updated comments.
3/13/94  Glen George       Changed set_sample_rate to return SIZE_X.
5/9/06   Glen George       Updated start_sample stub to match the new specification.
11/4/08  Julian Panetta    Altered stub functions to implement test analog sample. Also removed stub functions for keypad/display which have real implementations.

*/

/* library include files */
/* none */

/* local include files */
#include  "interfac.h"
#include  "scopedef.h"

/* Variables used to store state of analog hardware simulator */
static long int sample_rate;
static int sample_level, sample_slope;
static long int sample_delay;
static int is_sampling;

/* Buffer to store simulated analog input data */
static char sample_data[SIZE_X];

/* sampling parameter functions */
int set_sample_rate(long int rate)
{
    sample_rate = rate;
    return  SIZE_X;
}

void  set_trigger(int level, int slope)
{
    sample_level = level;
    sample_slope = slope;
    return;
}

void  set_delay(long int delay)
{
    sample_delay = delay;
    return;
}

/* sampling functions */

void  start_sample(int auto_trigger)
{
    get_test_sample(sample_rate, SIZE_X, sample_data);
    is_sampling = 1;
}

unsigned char far  *sample_done()
{
    /* Only return NULL once per sample */
    if (!is_sampling)
        return  NULL;
    else  
        {
            is_sampling = 0;
            return  sample_data;
        }
}
This file contains the constants for interfacing between the C code and the assembly code/hardware for the Digital Oscilloscope project. This is a sample interface file to allow compilation of the .c files.

Revision History:
3/8/94   Glen George       Initial revision.
3/13/94  Glen George       Updated comments.
3/17/97  Glen George       Added constant MAX_SAMPLE_SIZE and removed KEY_UNUSED.
11/04/08 Julian Panetta    Updated to match constants of my hardware drivers.

#ifndef  __INTERFAC_H__
#define  __INTERFAC_H__

/* library include files */
/* none */

/* local include files */
/* none */

/* constants */

/* keypad constants */
#define  KEY_MENU    0x12       /* <Menu>      */
#define  KEY_UP      0x02       /* <Up>        */
#define  KEY_DOWN    0x22       /* <Down>      */
#define  KEY_LEFT    0x11       /* <Left>      */
#define  KEY_RIGHT   0x14       /* <Right>     */
#define  KEY_ILLEGAL    0       /* illegal key */

/* display constants */
#define  SIZE_X         640     /* size in the x dimension */
#define  SIZE_Y         200     /* size in the y dimension */
#define  PIXEL_WHITE      0     /* pixel off */
#define  PIXEL_BLACK    0xF     /* pixel on */

/* scope parameters */
#define  MIN_DELAY         0    /* minimum trigger delay */
#define  MAX_DELAY     50000    /* maximum trigger delay */
#define  MIN_LEVEL         0    /* minimum trigger level (in mV) */
#define  MAX_LEVEL      5000    /* maximum trigger level (in mV) */

/* sampling parameters */
#define  MAX_SAMPLE_SIZE   2400 /* maximum size of a sample (in samples) */

#endif