CSCI E-93 MEMORY SUBSYSTEM
FOR ALTERA/TERASIC DE2-70 AND DE2-115 DEVELOPMENT BOARDS

Last revised: 11/18/2014

MEMORY MAPPED I/O

The following is a summary of the available memory-mapped I/O registers and their specific function. Addresses are specified as full 21-bit addresses. The memory implementation is little-endian, so for memory operations on multi-byte words, the relevant data (*i.e.,* the status bits or character code) will always be found in the *low-order byte*. In all references, bits are numbered with bit 0 being the *least significant* bit in the byte or word, regardless of the word size.

Address **Function**

00FF00 I/O Control Register

Bit	Decimal Value	On **Read** indicates:	**Write '1'** causes:
0	1	Serial Input Ready	Serial Input Flush
1	2	Serial Output Ready	Serial Output Flush
2	4	PS/2 Input Ready	PS/2 Input Flush
3	8	LCD Output Ready	LCD Output Flush

All other bits will be '0' when read and have no effect when written with a '1', but code should not depend on the values of any other bits when read and should write all other bits as '0'.

00FF04 **Read:** Serial (RS-232) character input buffer
Write: Serial (RS-232) character output buffer

Notes

- All serial communication occurs at **9600 Baud/No Parity/8 Data Bits/1 Stop Bit/No Flow Control**. This is not configurable.

- All serial I/O is "raw"—no specific line-termination convention (*i.e.,* CR, LF, CR/LF) is enforced. The specific line-termination character(s) you receive are determined by the settings of the terminal emulator you are using, and the line-termination character(s) the terminal emulator receives are determined solely by the characters your program actually sends.

  When you press "enter" in the terminal emulator, many terminal emulators by default will send a single CR [0x0d] character (at least on Windows; *nix may differ and send a single LF [0x0a]). However many terminal emulator programs allow you to modify this behavior.

  When receiving characters, terminal emulators by default will handle CR and LF literally, meaning that a CR will return the cursor to the beginning of the current line, and an LF will advance the cursor to the next line. To get "expected" newline behavior in this mode you will need to send a CR/LF pair. However most emulators can be configured to adjust this handling in various ways, for example, to translate a single CR to a CR/LF pair.

  Clearly, when writing your programs you will need to know the expected configuration and behavior of the terminal emulator you will be using.

00FF08 **Read:** PS/2 Keyboard character input buffer
Write: LCD Character Output buffer

Notes:

- The PS/2 keyboard is only partially supported. All the alphanumeric and many common non-alphanumeric keys (e.g., Tab, Enter, Backspace) are supported. The shift key is supported, however other modifier keys (Ctrl, Alt, Caps Lock) are not. "Extended" keys such as the Function keys and cursor control keys are also unsupported. Pressing an unsupported key will typically cause some sequence of "junk" characters to be transmitted. (For those familiar with the PS/2 keyboard protocol: support essentially includes most of the keys with a single-byte make code in Keyboard Scan Codes "Set 2." Keys with multi-byte make codes are not supported.)

- Pressing "Enter" on the PS/2 keyboard will cause a single CR [0x0d] character to be transmitted. This is not configurable.

- The LCD display is implemented as a very simple line-oriented terminal. It provides 2 lines of 16 characters each. The first character written appears in the first position on the first line; subsequent characters appear in successive positions, eventually wrapping to the beginning of the second line. When the second line is complete, the display clears and output wraps back to the beginning of the first line.

- The LCD handles CR and LF characters much as a default terminal emulator does: a CR [0x0d] character causes the cursor to move to the beginning of the current line, and a LF [0x0a] character advances the cursor to the next line (without changing its column position). A backspace character moves the cursor backwards one position (but does not clear the character that was there).

MEMORY CONTROLLER HARDWARE INTERFACE

The memory controller is implemented in an entity named **memory_controller** defined in the VHDL library **cscie93**. Your top-level entity will need to instantiate and port map this entity as described below under "Your top-level entity." The following table describes the ports on the **memory_controller** entity[1]. Note that from your perspective, any ports of direction "buffer" should be regarded as outputs from the memory controller entity. Note also that **16-bit reads or writes must be made only to word-aligned (*i.e.,* even) addresses, and 32-bit reads or writes must be made only to double-word-aligned addresses (*i.e.,* the two low-order address bits must be 0)**. The results of performing a read or write to an address that is not appropriately aligned are **undefined**.

Bolded signal names indicate signals that you are likely to care about.

Port Name	Direction	Type	Description
clk50mhz	in	std_logic	The DE2 50MHz clock. This port must be mapped to a signal connected to the appropriate pin.
mem_addr	in	std_logic_vector(20 downto 0)	The **mem_addr** signal of the processor-memory interface.
mem_data_write	in	std_logic_vector(31 downto 0)	The **mem_data_write** signal of the processor-memory interface. For 16-bit and 32-bit operations, the individual bytes will be written in little-endian order.
mem_rw	in	std_logic	The **mem_rw** signal of the processor-memory interface.
mem_sixteenbit	in	std_logic	The **mem_sixteenbit** signal of the processor-memory

[1] The table presented here lists the signals used on the DE2-70. The signals used on the DE2-115 are substantially similar, but there are some minor differences in the set of signals related to the SRAM. None of these are signals you will need to use in any way, and these differences are accurately reflected in the VHDL skeleton code included later in this document.

			interface. This signal is superseded by the **mem_thirtytwobit** signal. If both are asserted, the access mode will be 32-bit. **Important:** 16-bit accesses **must** be made **only** on word-aligned (*i.e.,* even) addresses. The result of performing a 16-bit read or write on an unaligned (*i.e.,* odd) address is undefined.
mem_thirtytwobit	In	std_logic	The **mem_thirtytwobit** signal of the processor-memory interface. This signal supersedes the **mem_sixteenbit** control signal. If both are asserted, the access mode will be 32-bit. **Important:** 32-bit accesses **must** be made **only** on double-word-aligned (*i.e.,* the two low-order address bits must be 0) addresses. The result of performing a 32-bit read or write on an unaligned (e.g., odd) address is undefined.
mem_addressready	in	std_logic	The **mem_addressready** signal of the processor-memory interface.
mem_reset	in	std_logic	An active-high reset signal that can be used to reset the memory controller to its initial state.
ps2_clk	in	std_logic	The clock signal for the DE2 PS/2 keyboard interface. This port must be mapped to a signal connected to the appropriate pin.
ps2_data	in	std_logic	The data signal for the DE2 PS/2 keyboard interface. This port must be mapped to a signal connected to the appropriate pin.
clock_hold	in	std_logic	Asserting this signal will stop the clock signals generated by the memory controller.
clock_step	in	std_logic	While the **clock_hold** signal is asserted, a rising edge on this signal will cause the generated clocks to run for a single full clock cycle. This signal has no effect when the **clock_hold** signal is not asserted.
clock_divide_limit	in	std_logic_vector(19 downto 0)	An optional 20-bit counter limit that can be used to slow down the effective rate of the generated clocks. If this value is non-zero, the generated clocks will be gated to zero for this number of cycles in between clock pulses. If this value is zero (or left unmapped), the clocks are not gated.
mem_suspend	in	std_logic	If this optional signal is asserted, the memory subsystem and its clocks will continue running, but it will not respond to any memory requests.
lcd_en	out	std_logic	Part of the interface with the LCD display. This port must be mapped to a signal connected to the appropriate pin.
lcd_on	out	std_logic	Part of the interface with the LCD display. This port must be mapped to a signal connected to the appropriate pin.
lcd_rs	out	std_logic	Part of the interface with the LCD display. This port must be mapped to a signal connected to the appropriate pin.
lcd_rw	out	std_logic	Part of the interface with the LCD display. This port must be mapped to a signal connected to the

			appropriate pin.
lcd_db	inout	std_logic_vector(7 downto 0)	Part of the interface with the LCD display. This port must be mapped to a signal connected to the appropriate pin.
mem_data_read	out	std_logic_vector(31 downto 0)	The **mem_data_read** signal of the processor-memory interface. For 16-bit and 32-bit operations, the individual bytes are read in little-endian order.
mem_dataready_inv	out	std_logic	The **mem_dataready_inv** signal of the processor-memory interface.
mem_ready	out	std_logic	Diagnostic signal, should generally not be used. Indicates when the memory subsystem is waiting for a new memory request.
sysclk1	out	std_logic	The generated system clock. **You should use this clock as your main system clock**. It is a 25MHz clock on a global clock routing net. This clock responds to the **clock_hold**, **clock_step** and **clock_divide_limit** signals described above.
sysclk2	out	std_logic	A secondary system clock, 180° out-of-phase with **sysclk1**. Like **sysclk1**, this clock responds to the **clock_hold**, **clock_step** and **clock_divide_limit** signals described above.
rs232_rxd	in	std_logic	Part of the interface with the RS232 serial port. This port must be mapped to a signal connected to the appropriate pin.
rs232_txd	out	std_logic	Part of the interface with the RS232 serial port. This port must be mapped to a signal connected to the appropriate pin.
rs232_cts	out	std_logic	Part of the interface with the RS232 serial port. This port must be mapped to a signal connected to the appropriate pin.
sram_clk	out	std_logic	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
sram_dq	inout	std_logic_vector (31 downto 0)	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
sram_addr	out	std_logic_vector(18 downto 0)	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
sram_adsc_N	out	std_logic	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
sram_adsp_N	out	std_logic	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
sram_adv_N	out	std_logic	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
sram_ce1_N	out	std_logic	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
sram_ce2	out	std_logic	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
sram_ce3_N	out	std_logic	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected

			to the appropriate pin.
sram_gw_N	out	std_logic	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
sram_oe_N	out	std_logic	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
sram_we_N	out	std_logic	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
sram_be_N	out	std_logic_vector(3 downto 0)	Part of the interface with the off-chip SSRAM memory chip. This port must be mapped to a signal connected to the appropriate pin.
serial_character_ready	out	std_logic	Asserted when there is at least one character ready to be read from the serial port character buffer. Can be used to drive a level-sensitive interrupt line.
ps2_character_ready	out	std_logic	Asserted when there is at least one character ready to be read from the PS/2 keyboard character buffer. Can be used to drive a level-sensitive interrupt line.
fsmStateCode	out	std_logic_vector(4 downto 0)	Diagnostic signal, should be ignored.
ps2KbStateCode	out	std_logic_vector(3 downto 0)	Diagnostic signal, should be ignored.
serial_out_fifo_full	buffer	std_logic	Diagnostic signal, should be ignored.
serial_out_fifo_empty	buffer	std_logic	Diagnostic signal, should be ignored.
lcd_fifo_full	buffer	std_logic	Diagnostic signal, should be ignored.
lcd_fifo_empty	buffer	std_logic	Diagnostic signal, should be ignored.

YOUR QUARTUS II PROJECT

Configuring your Quartus II Project

You will receive from us a set of files that implement the memory subsystem. One you are ready to begin using the memory subsystem in your project, you will need to include all of our VHDL files in your project. You should keep all of our files in a subdirectory, rather than comingling them in your main project directory with your own files. There is also a file named **cscie93.sdc**, this file specified some clock and timing information for the Altera TimeQuest Timing Analyzer. To enable an accurate timing analysis, and avoid warnings about undefined clocks, you should include this file in your project as well.

DE2-70 Only

Once you have included our VHDL files in your project, in order to successfully compile your project you will need to add the following to your project's **.qsf** file:

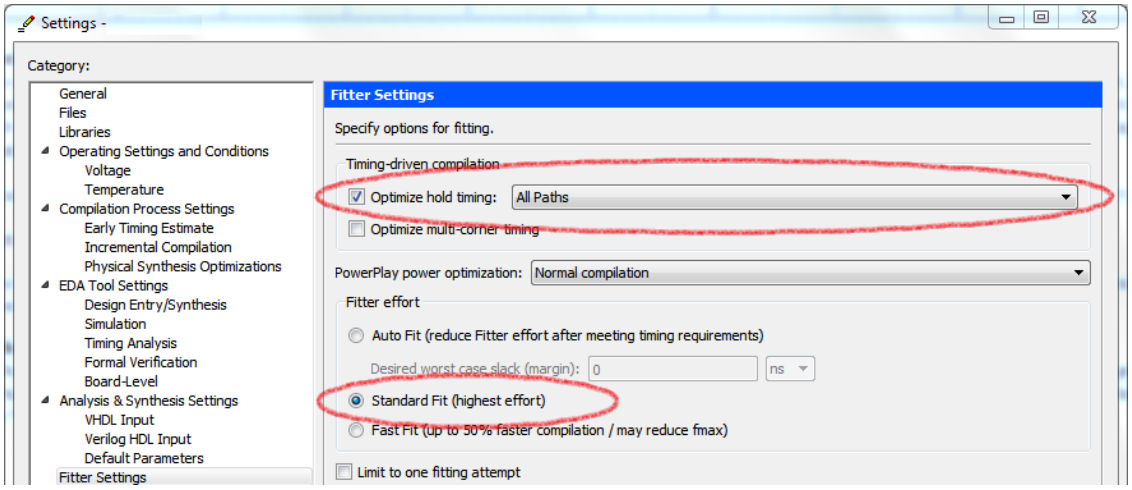
```
set_parameter -name CYCLONEII_SAFE_WRITE "\"RESTRUCTURE\""
```

This line can be added anywhere among the existing **set_global_assignment** lines. Without this line, your compilation will fail with memory configuration errors from the memory compiler.

Other suggestions

Although not required, there are several project settings that may improve your project's ability to satisfy timing requirements (at the expense of longer compilation times). If you find you are not meeting timing requirements, you may consider changing the following (both are available in the Settings dialog, in the "Fitter Settings" section, shown below):

- Make sure "Optimize hold timing" is enabled, and select "All Paths" instead of the default setting of "I/O Paths and Minimum TPD Paths"
- Change the "Fitter effort" setting to "Standard Fit (highest effort)



Your top-level entity

The following is a skeleton of what your top-level VHDL entity should look like. Your top-level entity **must** define the ports that are listed; because these ports are used to connect the memory controller to the various devices on the DE2 board that it uses, these ports **must** be assigned to specific pins. These assignments are provided to you as **chip_pin** attribute assignments in the top-level entity's architecture shells included below; please be sure you include all of these chip assignments in your top-level entity's architecture.

Important The DE2-70 and DE2-115 have **different and incompatible** pin assignments. Because of this, two separate top-level "skeleton s" are included below, and **it is essential that you use the one appropriate for the specific DE2 board you are using**. Otherwise, your design will not work, and you could risk damaging some of the on-board devices.

In addition to these required ports, your top-level entity may of course also include any additional ports that you require (e.g., to enable it to access push-buttons, slide switches, LED's and 7-segment displays). Pin assignments are not provided for these devices, you will need to add those yourself as needed.

In your top-level entity's architecture, you will need to instantiate one instance of the **cscie93.memory_controller** entity. Some of the ports on this entity must be mapped to the required top-level ports, as shown below. The remaining ports should be mapped as appropriate to ports on your CPU entity, which you will also instantiate in this architecture. Because the **memory_controller** entity defines a large number of ports, including some diagnostic signals which are not included in the sample VHDL which follows (and which do not need to be port mapped), you are **strongly encouraged** to do the port mapping using named associations (as shown) rather than positional associations.

DE2-70 Top-Level Shell File

```
library ieee;
use ieee.std_logic_1164.all;

library cscie93;

-- This file should be used for the DE2-70 board ONLY

entity YOUR_TOP_LEVEL_ENTITY_FOR_DE2_70 is
   port (
       -- CLOCK
       clk50mhz : in std_logic;
       PS/2 PORT
       ps2_clk : in std_logic;
       ps2_data : in std_logic;
       -- LCD
       lcd_en : out std_logic;
       lcd_on : out std_logic;
       lcd_rs : out std_logic;
       lcd_rw : out std_logic;
       lcd_db : inout std_logic_vector(7 downto 0);
       -- RS232
       rs232_rxd : in std_logic;
       rs232_txd : out std_logic;
       rs232_cts : out std_logic;
       -- SSRAM interface
       sram_clk : out std_logic;
       sram_dq : inout std_logic_vector (31 downto 0);
       sram_addr : out std_logic_vector(18 downto 0);
       sram_adsc_N : out std_logic;
       sram_adsp_N : out std_logic;
       sram_adv_N : out std_logic;
       sram_ce1_N : out std_logic;
       sram_ce2 : out std_logic;
       sram_ce3_N : out std_logic;
       sram_gw_N : out std_logic;
       sram_oe_N : out std_logic;
       sram_we_N : out std_logic;
       sram_be_N : out std_logic_vector(3 downto 0)
   );
end;

architecture default of YOUR_TOP_LEVEL_ENTITY is
   attribute chip_pin : string;

   attribute chip_pin of clk50mhz : signal is "AD15";

   attribute chip_pin of ps2_clk : signal is "F24";
   attribute chip_pin of ps2_data : signal is "E24";
```

```
   attribute chip_pin of lcd_on : signal is "F1";
   attribute chip_pin of lcd_en : signal is "E2";
   attribute chip_pin of lcd_rw : signal is "F3";
   attribute chip_pin of lcd_rs : signal is "F2";
   attribute chip_pin of lcd_db : signal is "B2,C3,C2,C1,D3,D2,E3,E1";

   attribute chip_pin of rs232_rxd : signal is "D21";
   attribute chip_pin of rs232_txd : signal is "E21";
   attribute chip_pin of rs232_cts : signal is "G22";

   attribute chip_pin of sram_dq : signal is
"AK8,AJ8,AK7,AK12,AH13,AJ13,AJ14,AK14,AJ16,AJ15,AH15,AJ22,AK22,AJ21,AK21,AJ20,AK23,AJ19,AK19,AH18,AJ18,AH17,AJ17,AK17,AH16,AJ12,AH12,
AK11,AJ11,AK10,AJ10,AH10";
   attribute chip_pin of sram_clk : signal is "AD7";
   attribute chip_pin of sram_addr : signal is
"AF11,AE11,AG20,AF20,AC16,AF15,AE15,AG14,AF14,AE13,AD13,AG12,AE12,AG5,AG6,AG7,AH7,AF8,AG8";
   attribute chip_pin of sram_adsc_N : signal is "AG17";
   attribute chip_pin of sram_adsp_N : signal is "AC18";
   attribute chip_pin of sram_adv_N : signal is "AD16";
   attribute chip_pin of sram_ce1_N : signal is "AH19";
   attribute chip_pin of sram_ce2 : signal is "AG19";
   attribute chip_pin of sram_ce3_N : signal is "AD22";
   attribute chip_pin of sram_gw_N : signal is "AG18";
   attribute chip_pin of sram_oe_N : signal is "AD18";
   attribute chip_pin of sram_we_N : signal is "AF18";
   attribute chip_pin of sram_be_N : signal is "AH20 , AD20,AC20, AC21";

begin

   mem : cscie93.memory_controller port map (
                clk50mhz => clk50mhz,
                mem_addr => your-signal-here,
                mem_data_write => your-signal-here,
                mem_rw => your-signal-here,
                mem_sixteenbit => your-signal-here,
                mem_thirtytwobit => your-signal-here,
                mem_addressready => your-signal-here,
                mem_reset => your-signal-here,
                ps2_clk => ps2_clk,
                ps2_data => ps2_data,
                clock_hold => your-signal-here,
                clock_step => your-signal-here,
                clock_divide_limit => your-signal-here,
                mem_suspend => your-signal-here,
                lcd_en => lcd_en,
                lcd_on => lcd_on,
                lcd_rs => lcd_rs,
                lcd_rw=> lcd_rw,
                lcd_db=> lcd_db,
                mem_data_read => your-signal-here,
                mem_dataready_inv => your-signal-here,
                sysclk1 => your-signal-here,
```

```
                sysclk2 => your-signal-here,
                rs232_rxd => rs232_rxd,
                rs232_txd => rs232_txd,
                rs232_cts => rs232_cts,
                sram_clk => sram_clk,
                sram_dq => sram_dq,
                sram_addr => sram_addr,
                sram_adsc_N => sram_adsc_N,
                sram_adsp_N => sram_adsp_N,
                sram_adv_N => sram_adv_N,
                sram_ce1_N => sram_ce1_N,
                sram_ce2 => sram_ce2,
                sram_ce3_N => sram_ce3_N,
                sram_gw_N => sram_gw_N,
                sram_oe_N => sram_oe_N,
                sram_we_N => sram_we_N,
                sram_be_N => sram_be_N,
                serial_character_ready => your-signal-here,
                ps2_character_ready => your-signal-here,
            );

 -- Instantiate your CPU entity here!
 cpu : entity work.your_cpu port map (
       . . .
       );

end;
```

DE2-70 Entity Declaration for the `memory_controller` entity

```
entity memory_controller is
    port (
        clk50mhz : in std_logic;

        mem_addr : in std_logic_vector(20 downto 0);
        mem_data_write : in std_logic_vector(31 downto 0);
        mem_rw : in std_logic;
        mem_sixteenbit : in std_logic;
        mem_thirtytwobit : in std_logic;
        mem_addressready : in std_logic;

        mem_reset : in std_logic;

        ps2_clk : in std_logic;
        ps2_data : in std_logic;

        clock_hold : in std_logic;
        clock_step : in std_logic;
        clock_divide_limit : in std_logic_vector(19 downto 0) := (others => '0');
        mem_suspend : in std_logic;

        lcd_en : out std_logic;
        lcd_on : out std_logic;
        lcd_rs : out std_logic;
        lcd_rw : out std_logic;
        lcd_db : inout std_logic_vector(7 downto 0);

        mem_data_read : out std_logic_vector(31 downto 0);
        mem_dataready_inv : out std_logic;
        mem_ready : out std_logic;

        sysclk1 : out std_logic;
        sysclk2 : out std_logic;

        rs232_rxd : in std_logic;
        rs232_txd : out std_logic;
        rs232_cts : out std_logic;

        -- SSRAM interface
        sram_clk : out std_logic;
        sram_dq : inout std_logic_vector (31 downto 0);
        sram_addr : out std_logic_vector(18 downto 0);
        sram_adsc_N : out std_logic;
        sram_adsp_N : out std_logic;
        sram_adv_N : out std_logic;
        sram_ce1_N : out std_logic;
        sram_ce2 : out std_logic;
        sram_ce3_N : out std_logic;
        sram_gw_N : out std_logic;
```

```
        sram_oe_N : out std_logic;
        sram_we_N : out std_logic;
        sram_be_N : out std_logic_vector(3 downto 0);

        -- usable as interrupts for char ready
        serial_character_ready : out std_logic;
        ps2_character_ready : out std_logic;

        --diagnostics
        fsmStateCode : out std_logic_vector(4 downto 0);
        ps2KbStateCode : out std_logic_vector(3 downto 0);
        serial_out_fifo_full : buffer std_logic;
        serial_out_fifo_empty : buffer std_logic;
        lcd_fifo_full : buffer std_logic;
        lcd_fifo_empty : buffer std_logic
    );
end;
```

DE2-115 Top-Level Shell File

```
library ieee;
use ieee.std_logic_1164.all;

library cscie93;

-- This file should be used for the DE2-115 board ONLY

entity YOUR_TOP_LEVEL_ENTITY_FOR_DE2_115 is
   port (
       -- CLOCK
       clk50mhz : in std_logic;
       -- PS/2 PORT
       ps2_clk : in std_logic;
       ps2_data : in std_logic;
       -- LCD
       lcd_en : out std_logic;
       lcd_on : out std_logic;
       lcd_rs : out std_logic;
       lcd_rw : out std_logic;
       lcd_db : inout std_logic_vector(7 downto 0);
       -- RS232
       rs232_rxd : in std_logic;
       rs232_txd : out std_logic;
       rs232_cts : out std_logic;
       -- SSRAM interface
       sram_dq : inout std_logic_vector (15 downto 0);
       sram_addr : out std_logic_vector(19 downto 0);
       sram_ce_N : out std_logic;
       sram_oe_N : out std_logic;
       sram_we_N : out std_logic;
       sram_ub_N : out std_logic;
       sram_lb_N : out std_logic
   );
end;

architecture default of YOUR_TOP_LEVEL_ENTITY is
   attribute chip_pin : string;

   attribute chip_pin of clk50mhz : signal is "Y2";

   attribute chip_pin of ps2_clk : signal is "G6";
   attribute chip_pin of ps2_data : signal is "H5";

   attribute chip_pin of lcd_on : signal is "L5";
   attribute chip_pin of lcd_en : signal is "L4";
   attribute chip_pin of lcd_rw : signal is "M1";
   attribute chip_pin of lcd_rs : signal is "M2";
   attribute chip_pin of lcd_db : signal is "M5,M3,K2,K1,K7,L2,L1,L3";
```

```
   attribute chip_pin of rs232_rxd : signal is "G12";
   attribute chip_pin of rs232_txd : signal is "G9";
   attribute chip_pin of rs232_cts : signal is "G14";

   attribute chip_pin of sram_dq : signal is "AG3,AF3,AE4,AE3,AE1,AE2,AD2,AD1,AF7,AH6,AG6,AF6,AH4,AG4,AF4,AH3";
   attribute chip_pin of sram_addr : signal is "T8,AB8,AB9,AC11,AB11,AA4,AC3,AB4, AD3, AF2, T7, AF5, AC5, AB5, AE6, AB6, AC7, AE7,
AD7, AB7";
   attribute chip_pin of sram_ce_N : signal is "AF8";
   attribute chip_pin of sram_oe_N : signal is "AD5";
   attribute chip_pin of sram_we_N : signal is "AE8";
   attribute chip_pin of sram_ub_N : signal is "AC4";
   attribute chip_pin of sram_lb_N : signal is "AD4";

begin

   mem : cscie93.memory_controller port map (
                clk50mhz => clk50mhz,
                mem_addr => your-signal-here,
                mem_data_write => your-signal-here,
                mem_rw => your-signal-here,
                mem_sixteenbit => your-signal-here,
                mem_thirtytwobit => your-signal-here,
                mem_addressready => your-signal-here,
                mem_reset => your-signal-here,
                ps2_clk => ps2_clk,
                ps2_data => ps2_data,
                clock_hold => your-signal-here,
                clock_step => your-signal-here,
                clock_divide_limit => your-signal-here,
                mem_suspend => your-signal-here,
                lcd_en => lcd_en,
                lcd_on => lcd_on,
                lcd_rs => lcd_rs,
                lcd_rw => lcd_rw,
                lcd_db => lcd_db,
                mem_data_read => your-signal-here,
                mem_dataready_inv => your-signal-here,
                sysclk1 => your-signal-here,
                sysclk2 => your-signal-here,
                rs232_rxd => rs232_rxd,
                rs232_txd => rs232_txd,
                rs232_cts => rs232_cts,
             sram_dq => sram_dq,
             sram_addr => sram_addr,
             sram_ce_N => sram_ce_N,
             sram_oe_N => sram_oe_N,
             sram_we_N => sram_we_N,
             sram_ub_N => sram_ub_N,
             sram_lb_N => sram_lb_N,
                serial_character_ready => your-signal-here,
                ps2_character_ready => your-signal-here,
             );
```

```
  -- Instantiate your CPU entity here!
  cpu : entity work.your_cpu port map (
        . . .
        );

end;
```

DE2-115 Entity Declaration for the `memory_controller` entity

```
entity memory_controller is
   port (
      clk50mhz : in std_logic;

      mem_addr : in std_logic_vector(20 downto 0);
      mem_data_write : in std_logic_vector(31 downto 0);
      mem_rw : in std_logic;
      mem_sixteenbit : in std_logic;
      mem_thirtytwobit : in std_logic;
      mem_addressready : in std_logic;

      mem_reset : in std_logic;

      ps2_clk : in std_logic;
      ps2_data : in std_logic;

      clock_hold : in std_logic;
      clock_step : in std_logic;
      clock_divide_limit : in std_logic_vector(19 downto 0) := (others => '0');
      mem_suspend : in std_logic;

      lcd_en : out std_logic;
      lcd_on : out std_logic;
      lcd_rs : out std_logic;
      lcd_rw : out std_logic;
      lcd_db : inout std_logic_vector(7 downto 0);

      mem_data_read : out std_logic_vector(31 downto 0);
      mem_dataready_inv : out std_logic;
      mem_ready : out std_logic;

      sysclk1 : out std_logic;
      sysclk2 : out std_logic;

      rs232_rxd : in std_logic;
      rs232_txd : out std_logic;
      rs232_cts : out std_logic;

      -- SSRAM interface
      sram_dq : inout std_logic_vector (15 downto 0);
      sram_addr : out std_logic_vector(19 downto 0);
      sram_ce_N : out std_logic;
      sram_oe_N : out std_logic;
      sram_we_N : out std_logic;
      sram_ub_N : out std_logic;
      sram_lb_N : out std_logic;

      -- usable as interrupts for char ready
```

```
        serial_character_ready : out std_logic;
        ps2_character_ready : out std_logic;

        --diagnostics
        fsmStateCode : out std_logic_vector(5 downto 0);
        ps2KbStateCode : out std_logic_vector(3 downto 0);
        serial_out_fifo_full : buffer std_logic;
        serial_out_fifo_empty : buffer std_logic;
        lcd_fifo_full : buffer std_logic;
        lcd_fifo_empty : buffer std_logic;
        dbg_lcd_ddram_addr : out std_logic_vector(7 downto 0)
    );
end;
```

Known issues and limitations

The PS/2 keyboard is only partially supported, as detailed above.

The LCD character output can be occasionally unreliable, with a tendency to repeat and/or skip occasional characters at certain clock rates. You are generally advised to use the serial input and output whenever possible.