



Tutorial 4: Computer Components - I/O Interface

Tutor Notes

1 Introduction

This tutorial gives you experience in interfacing I/O components to a CPU. The LCD display on the NanoBoard will be interfaced to the same CPU which was used for Tutorial 3. There are two possible methods of implementing the LCD display interface: one using a LCD controller provided by Altium and without it. You are expected to implement the first method and if you have time, implement the second.

Want to show that there is an inverse correlation between the interface hardware complexity and the complexity of the software. Using the LCD controller, the code required to access the LCD display is relatively short. Without using the LCD controller, the interface is relatively simple, yet the code is long.

As part of the preparation for this tutorial, read through the data sheet for the LCD display which can be found in the Lecture Notes Part II Section C1. Also read through the information on the Altium *LCD16X2 LCD Controller* which can be found in the Lecture Notes Part II Section B2.

The students have been encouraged in lectures to do some background reading in preparation. It would be good if you could also do some background reading to ensure that you understand the LCD display and the LCD controller.

To access the external data memory, where the LCD display will be interfaced, you need to use a special directive in C. The best way to do this is to declare a pointer to the external data memory space and assign the pointer to the first address. The pointer can then be dereferenced by treating it as a pointer to an array. An example of this is as follows:

```
__xdata volatile char *LCD = 0x0000; // Declare pointer into external
                                   // data memory starting at address 0x0000

LCD[0x0000]=0x12; // Write 0x12 to external data memory at address 0x0000
T=LCD[0x0001];   // Read external data memory at address 0x0001
```

The above has been covered in lectures.

As covered in lectures, a consideration when writing to or reading from I/O devices is that the status of the I/O has to be checked first. With the LCD display, there is a busy flag that needs to be read before writing to the display. You need to take this into account when writing your test code.

This is one of the important differences with I/O devices: the status of the device has to be checked before accessing the device. With the LCD display, there is a busy flag that needs to be checked.

More information on the actual controller used in the LCD display can be found at:

- KS0066U 16COM / 40SEG DRIVER & CONTROLLER FOR DOT MATRIX LCD
(<http://www.hantronix.com/download/ks0066u.pdf>)

The LCD display is 16 characters wide by two lines, however the internal controller assumes that there are 40 characters per line. This needs to be taken into account when writing test code for the interface without the Altium LCD controller.

It means that when you get to the end of the first line, you need to skip to position 40. Refer to the test code for the interface without using the controller where a string is sent to the display.

2 LCD Display Interface using the LCD16X2 LCD Controller

The first version of the interface uses the *LCD16X2 LCD Controller* provided by Altium. This connects directly to the LCD display. It handles the LCD initialization at power up and the writing of characters to the LCD display given the position on the screen.

The way in which the controller is connected to the LCD display is shown in Figure 6 of the *LCD16X2 LCD Controller* documentation. However, to interface the controller to the CPU also requires some extra glue logic. A block diagram showing the glue logic to the controller and the LCD display is shown in Figure 1. Note that the LCD controller re-

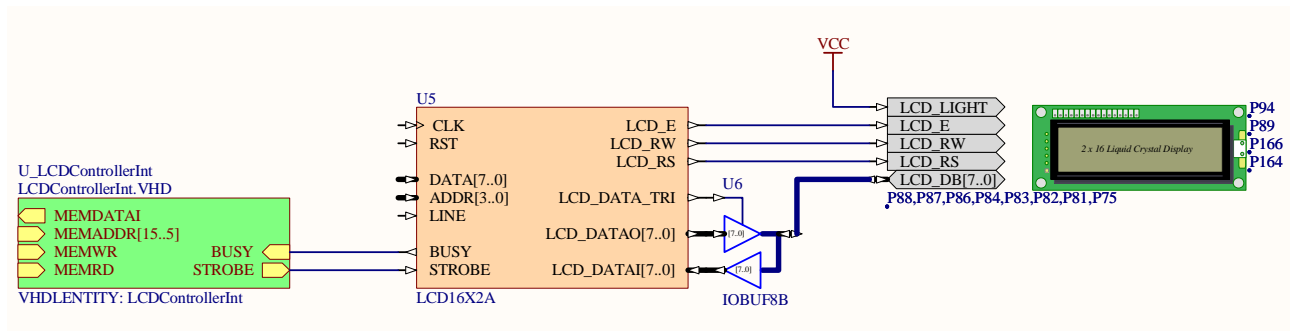


Figure 1: Interface using the LCD controller

quires 32 addresses (16 characters by two lines). The lower order address lines can connect directly to the LCD controller. The higher order address lines have to be decoded to select the correct address range. The same signals as used for the CPU can be used for **CLK** and **RST**.

The **LINE** input to the LCD controller can connect to the next highest address line. This means that the LCD controller covers 32 addresses, one for each character on the screen.

The data is read by the controller using **CLK** when **STROBE** is asserted.

The way in which the data is latched from the CPU is different to that used by, say, a memory device. This is because the LCD controller is a synchronous device with data being latched using the **CLK** signal.

There is also a **BUSY** output signal which needs to be accessible by the CPU. This could be mapped to one or all of the addresses set aside for the LCD controller. Software needs to sample this signal to check that the next character can be written.

It is actually simpler just to make the **BUSY** available for a read to any of the addresses in the LCD controller's address range.

Use the following procedure to implement the interface:

- Draw a timing diagram showing the inputs and outputs for the glue logic. Include a write cycle to the LCD display and a read of the **BUSY** flag.

The timing diagram is not shown here but it should be done first to ensure that the requirements of the design are understood and to make it easier to design the interface. Refer to the timing diagram for the simulation shown in Figure 5.

- Design the glue logic using VHDL. Place the LCD controller in the external memory address range 0x0000 to 0x001f. The busy signal should be mapped to one or all of these addresses. There are three parts to the interface:
 - Address decoding
 - Generation of the **STROBE** signal
 - Reading of the **BUSY** signal

*The VHDL code for the interface using the LCD controller is on Page 7. Note that the **BUSY** signal is returned for all addresses in the range of the LCD controller. Also, the **BUSY** signal could have been connected directly to the CPU as this is the only device on the bus and the only signal to be read. Normally don't use 'Z' for System on Chip (SoC) designs but use a multiplexer instead to choose which signal is routed back to the CPU.*

- Simulate the glue logic to ensure correct operation.

The VHDL code for the stimulus part of the test bench is on Page 8. The simulation results is shown in Figure 5. Note that the simulation shows a practical situation. The **BUSY** flag is read first then a write operation occurs and then the **BUSY** flag is read again and shown to be high.

- Implement the design on the NanoBoard using the circuit from Tutorial 3 and write suitable test code. The top level schematic is shown in Figure 2. The test code is shown on Page 11.

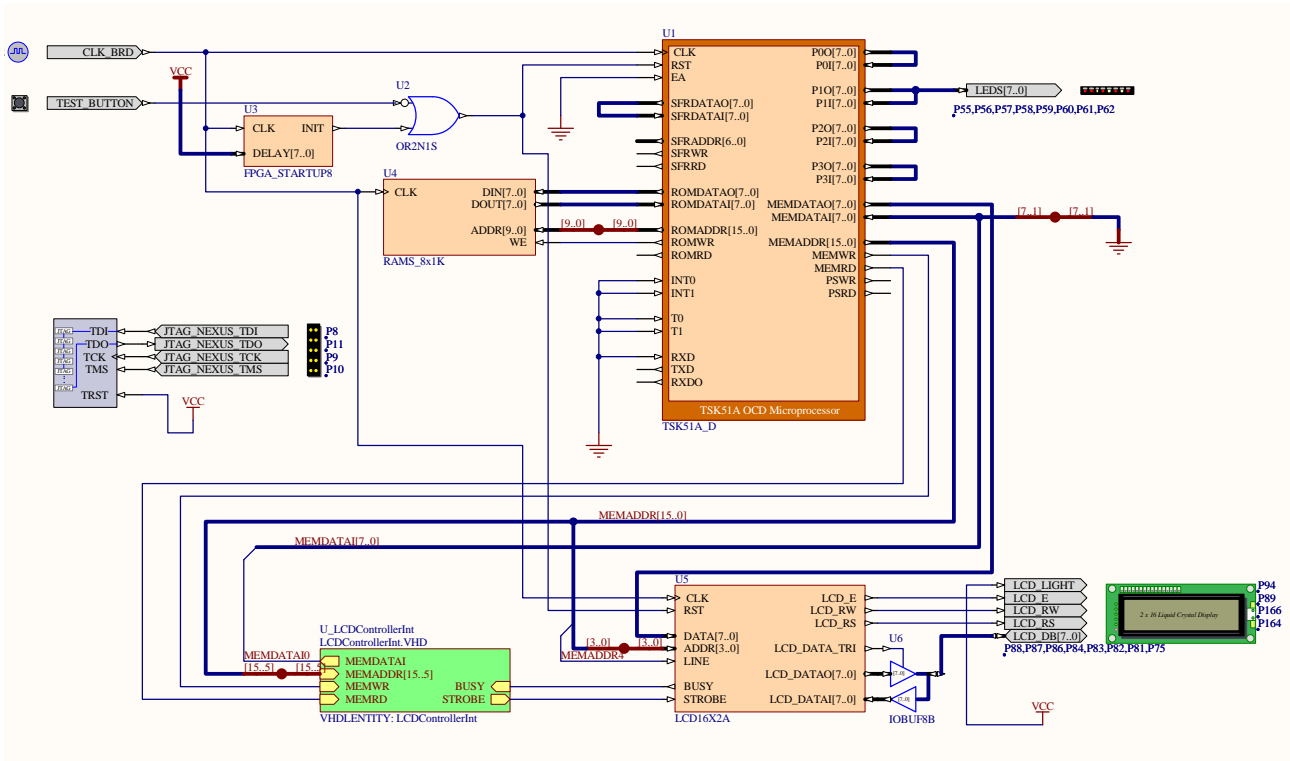


Figure 2: Top level schematic using the LCD controller

A good test to see that everything is working is to have a message flash on the display continuously. The test code does this.

3 Direct CPU Interface to the LCD Display

Attempt this task if you have time.

It is possible to connect the CPU with some glue logic to the LCD display without using the LCD controller used previously. Study the timing diagram for the CPU again, as you did for Tutorial 4, and the timing diagram for the LCD display. You should also study the table for the LCD display which describes the commands. Note that the busy flag must be checked before writing to the LCD display.

One aspect of the design that you will notice when looking at the timing diagrams is that it is not possible to directly connect the CPU to the LCD display. The **LCD_RS** and **LCD_RW** signals will have to be generated separately. That is, a register (latch) will need to be implemented to drive these signals. The **LCD_DB** and **LCD_E** signals can be generated directly from the CPU.

A block diagram for the interface is shown in Figure 3. (This is Figure 2 in the tutorial work sheet). Note that the

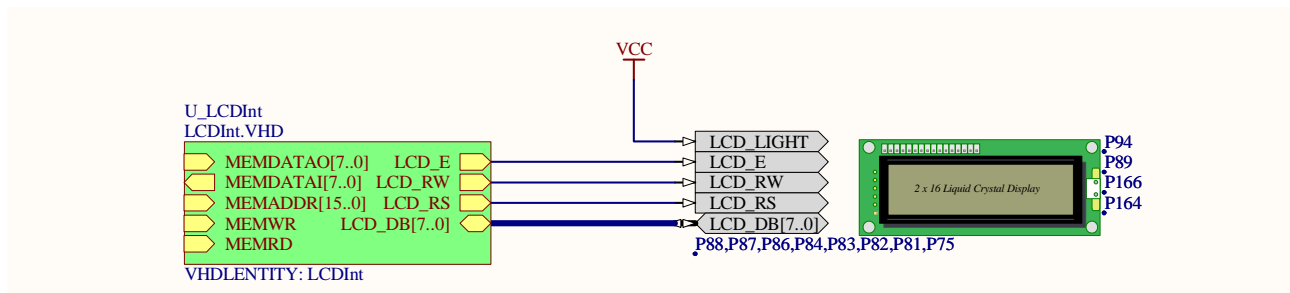


Figure 3: Direct interface between the CPU and LCD display

buffering of the data lines is done as part of the interface whereas previously the buffer was a separate component,

Use the following procedure to implement the interface:

- Confirm from the timing diagram that the CPU can't be connected directly to the LCD display. State what signals timing cannot be met.

*Have a look at the set up time for the **LCD_RW** which could be generated from **MEMRD** before the **LCD_E** signal is asserted. It would be possible to delay **LCD_E** but then the minimum pulse width would not be met. You could draw timing diagrams to help understand the timing constraints.*

*Note that **LCD_RW** could still be driven from an address line without any problems (see if any student picks this up). However, in this case it is just as easy to latch both **LCD_RS** and **LCD_RW** for consistency.*

- Draw a timing diagram for a write to the latch which sets **LCD_RS** and **LCD_RW**.

The timing diagram is not shown here but it should be done first to ensure that the requirements of the interface are understood and to make it easier to design the interface. Refer to the timing diagram for the simulation shown in Figure 6.

- Draw a timing diagram showing a read and a write cycle from the CPU to the LCD display assuming that **LCD_RS** and **LCD_RW** have been set. Show all the inputs and outputs of the LCD display interface.

See previous note.

- Design the interface using the external data address 0x0000 to access the LCD display and address 0x0001 for the latch to generate the signals **LCD_RS** and **LCD_RW**. There are four parts to the interface:
 - Address decoding
 - Data signal routing
 - Latch to generate **LCD_RS** and **LCD_RW**
 - Generation of **LCD_E** signal

The VHDL code for the interface without using the LCD controller is on Page 9.

Ensure that there are no clashes on any of the data lines.

This is done by checking the control signals going to the LCD display (**LCD_RW**) when determining whether to drive the data lines. Note that the **MEMWR** could have also been checked when writing to the LCD display, however it was not. It is assumed that software accessing the LCD display will only do a write when **LCD_RW** is low, but if it does a read, there will still be no bus clash.

- Simulate the design. This includes the operation of the latch and the signals going to the LCD display.

The VHDL code for the stimulus part of the test bench is on Page 10. The simulation results is shown in Figure 6. The simulation shows a practical situation. It simulates part of the test code where one of the LCD registers are set. The busy flag is checked first before writing to the register.

Note that the output of the latch is undefined until it is first written. In this design it does not matter that they are undefined since the **LCD_E** signal has to be asserted before the LCD display will do anything. In general it is important to take into account states at power up to make sure that the circuit will function correctly.

- Implement the design on the NanoBoard by writing a simple test program.

The top level schematic is shown in Figure 4. The test code is shown on Page 12.

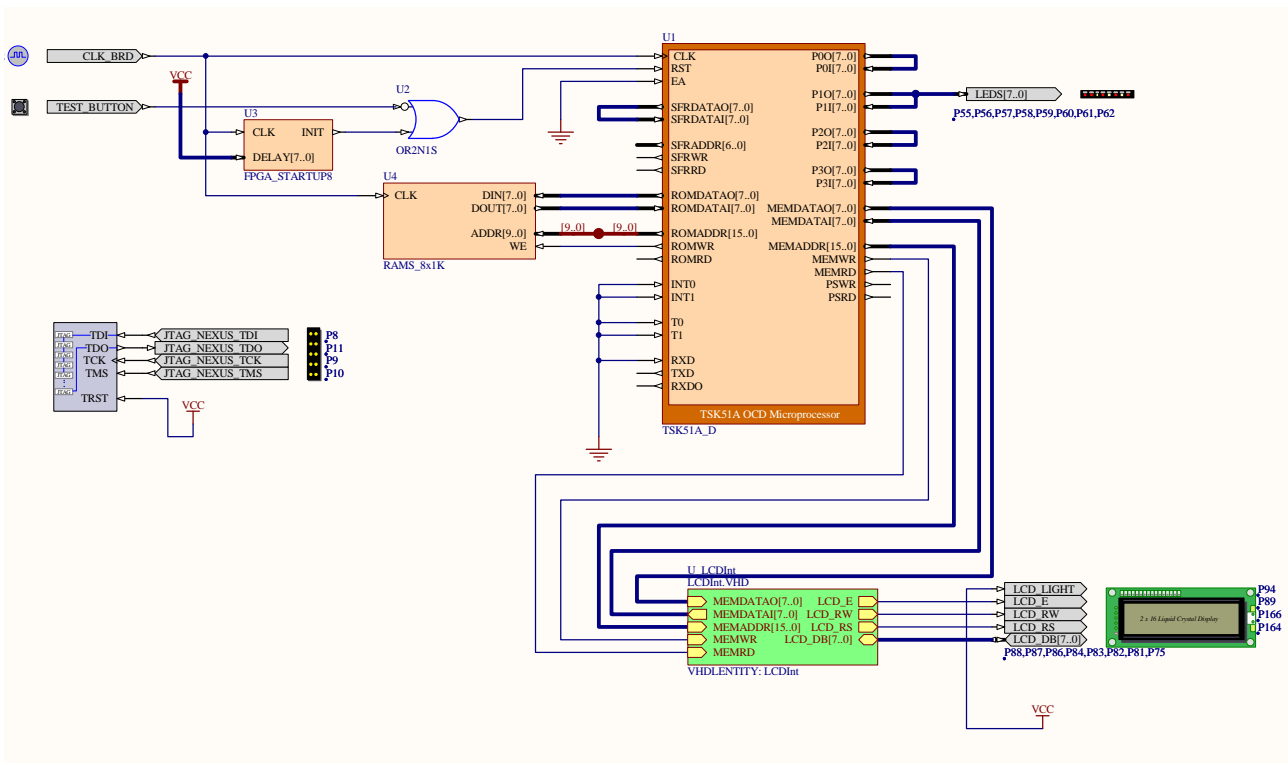


Figure 4: Top level schematic without using the LCD controller

You will have to also set the operating parameters of the LCD display at the start of your program. These can be determined from the documentation (otherwise try writing the following sequence to the instruction register: 0x01, 0x06, 0x0c, 0x38, 0x80).

You will notice that the code required to access the LCD display is much longer than that used for the first interface. It is a good example of how an intelligent interface can reduce the load on the CPU.

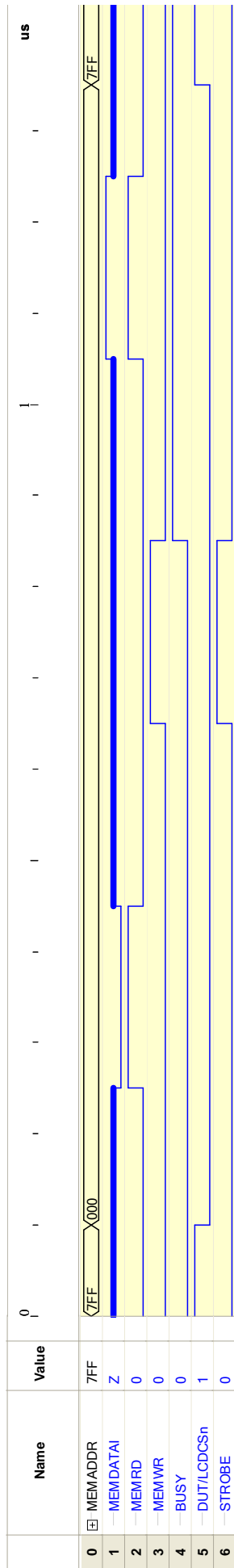


Figure 5: Simulation of the LCD interface using the LCD controller

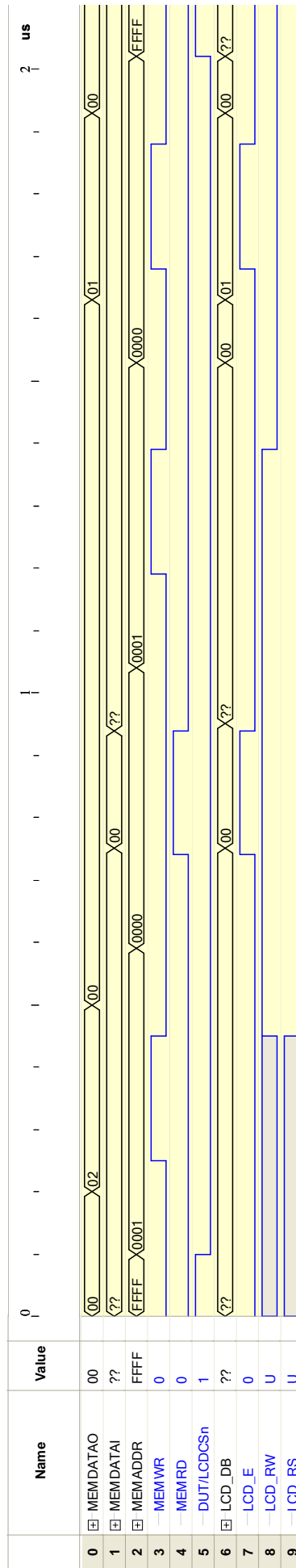


Figure 6: Simulation of the LCD interface without using the LCD controller

VHDL Code

Interface using the LCD Controller

```
— ELEC4605 Computer Engineering
— LCDControllerInt.vhd
— Peter Stepien
— 2005

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity LCDControllerInt is
  port(MEMDATAI : out std_logic;
        MEMADDR  : in  std_logic_vector(15 downto 5);
        MEMWR    : in  std_logic;
        MEMRD    : in  std_logic;
        BUSY     : in  std_logic;
        STROBE   : out std_logic);
end LCDControllerInt;

architecture behaviour of LCDControllerInt is
  signal LCDCSn : std_logic;
begin
  — Address decoding (need 32 addresses)
  — Place LCD as address 0x0000
  LCDCSn <= '0' when MEMADDR=b"0000000000" else '1';

  — Write to the LCD controller
  STROBE <= MEMWR when LCDCSn='0' else '0';

  — Read from LCD controller (the busy signal)
  MEMDATAI <= BUSY when (LCDCSn='0' and MEMRD='1') else 'Z';
end behaviour;
```

Interface using the LCD Controller Test Bench Stimulus

```
STIMULUS0: process
begin
  — insert stimulus here
  — Set initial values
  MEMADDR <= b"111111111111";
  MEMRD <= '0';
  MEMWR <= '0';
  BUSY <= '0';
  wait for 100ns;
  — Read cycle to LCD controller
  — Only reading the BUSY flag
  MEMADDR <= b"000000000000";
  wait for 150ns;
  MEMRD <= '1'; wait for 200ns;
  MEMRD <= '0'; wait for 100ns;
  — Write cycle to LCD controller
  — Note data connected directly to the controller
  MEMADDR <= b"000000000000";
  wait for 100ns;
  MEMWR <= '1'; wait for 200ns;
  MEMWR <= '0';
  BUSY <= '1'; — Assume BUSY flag set since LCD processing commant
  wait for 50ns;
  — Read cycle to LCD controller
  — Only reading the BUSY flag
  MEMADDR <= b"000000000000";
  wait for 150ns;
  MEMRD <= '1'; wait for 200ns;
  MEMRD <= '0'; wait for 100ns;
  — Finish by changing the address
  MEMADDR <= b"111111111111";
  wait for 100ns;
  wait;
end process;
```

Interface Without using the LCD Controller

```

— ELEC4605 Computer Engineering
— LCDInt.vhd
— Peter Stepien
— 2005

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity LCDInt is
  port(MEMDATAI : out std_logic_vector(7 downto 0);
        MEMDATAO : in std_logic_vector(7 downto 0);
        MEMADDR  : in std_logic_vector(15 downto 0);
        MEMWR     : in std_logic;
        MEMRD     : in std_logic;
        LCD_DB    : inout std_logic_vector(7 downto 0);
        LCD_E     : out std_logic;
        LCD_RW    : out std_logic;
        LCD_RS    : out std_logic);
end LCDInt;

architecture behaviour of LCDInt is
  signal LCDCSn : std_logic;
  signal RW, RS : std_logic;
begin
  — Latch the LCD control signals (address 0x0001)
  latch_RW_RS: process (MEMWR)
  begin
    if (MEMWR='0' and MEMWR' event) then
      if (LCDCSn='0' and MEMADDR(0)='1') then
        RW <= MEMDATAO(1);
        RS <= MEMDATAO(0);
      end if;
    end if;
  end process;

  — Address decoding (need 2 addresses) at address 0x0000
  — 0x0000 Read/write to LCD
  — 0x0001 Set RW (D1) and RS (D0)
  LCDCSn <= '0' when MEMADDR(15 downto 1)=b"000000000" else '1';

  — Write to LCD (check RW to avoid clash)
  LCD_DB <= MEMDATAO when (LCDCSn='0' and MEMADDR(0)='0' and
    RW='0') else "ZZZZZZZ";

  — Read from LCD (check RW to avoid clash)
  MEMDATAI <= LCD_DB when (LCDCSn='0' and MEMADDR(0)='0' and
    MEMRD='1' and RW='1') else "ZZZZZZZ";

  — Enable signal for both write and read
  LCD_E <= '1' when (LCDCSn='0' and MEMADDR(0)='0' and
    (MEMRD='1' or MEMWR='1')) else '0';

  — Pass through control signals
  — Note: could not use buffer since no buffer option at the schematic level
  LCD_RW <= RW;
  LCD_RS <= RS;
end behaviour;

```

Interface using the LCD Controller Test Bench Stimulus

```

STIMULUS0: process
begin
  — insert stimulus here
  — Simulate the function call write_control(0x01);
  — Set initial values
  LCD.DB <= b"ZZZZZZZZ";
  MEMDATAO <= x"00";
  MEMADDR <= x"ffff";
  MEMRD <= '0';
  MEMWR <= '0';
  wait for 100ns;
  — write_control() calls wait_busy() which calls read_control()
  — LCD[1]=2;
  — Write cycle to LCD
  MEMADDR <= x"0001";
  wait for 100ns;
  MEMDATAO <= x"02"; wait for 50ns;
  MEMWR <= '1'; wait for 200ns;
  MEMWR <= '0'; wait for 50ns;
  MEMDATAO <= x"00"; wait for 90ns;
  — return(LCD[0]);
  — Read cycle to LCD
  MEMADDR <= x"0000";
  wait for 150ns;
  MEMRD <= '1'; wait for 10ns;
  LCD.DB <= x"00"; wait for 190ns;
  MEMRD <= '0'; wait for 10ns;
  LCD.DB <= b"ZZZZZZZZ"; wait for 90ns;
  — Back to write_control()
  — LCD[1]=0;
  — Write cycle to LCD
  MEMADDR <= x"0001";
  wait for 100ns;
  MEMDATAO <= x"00"; wait for 50ns;
  MEMWR <= '1'; wait for 200ns;
  MEMWR <= '0'; wait for 50ns;
  MEMDATAO <= x"00"; wait for 90ns;
  — LCD[0]=0x01;
  — Write cycle to LCD
  MEMADDR <= x"0000";
  wait for 100ns;
  MEMDATAO <= x"01"; wait for 50ns;
  MEMWR <= '1'; wait for 200ns;
  MEMWR <= '0'; wait for 50ns;
  MEMDATAO <= x"00"; wait for 90ns;
  — Finish by changing the address
  MEMADDR <= x"ffff";
  wait for 100ns;
  wait;
end process;

```

Test Code

Using the LCD Controller

```
// ELEC4605 Computer Engineering
// Test program for first LCD interface
// Peter Stepien
// 2005

__xdata volatile char *LCD = 0x0000;

void write_string(__rom char *string)
{
    int i;

    i=0;
    while (string[i]!=0)
    {
        while ((LCD[0] & 0x01) == 1) {}
        LCD[i]=string[i];
        i++;
    }
}

void main(void)
{
    unsigned long i;

    for (;;)
    {
        write_string("0123456789ABCDEF<This is a test>");
        for (i=0;i<0xffff;i++)
        {
            __asm("nop");
        }
        write_string(".....");
        for (i=0;i<0xffff;i++)
        {
            __asm("nop");
        }
    }
}
```

Without Using the LCD Controller

```
// ELEC4605 Computer Engineering
// Test program for second LCD interface
// Peter Stepien
// 2005

__xdata volatile char *LCD = 0x0000;

void pause ()
{
    unsigned long i;
    for (i=0;i<0xffff;i++)
    {
        __asm("nop");
    }
}

char read_control ()
{
    LCD[1]=2;
    return(LCD[0]);
}

void wait_busy ()
{
    while ((read_control () & 0x80) != 0) {}
}

void write_control(char value)
{
    wait_busy ();
    LCD[1]=0;
    LCD[0]= value ;
}

void write_data(char value)
{
    wait_busy ();
    LCD[1]=1;
    LCD[0]= value ;
}

char read_data ()
{
    wait_busy ();
    LCD[1]=3;
    return(LCD[0]);
}

void lcd_init ()
{
    write_control(0x01); // Clears display
    // pause ();
    write_control(0x06); // Direction and shift
    // pause ();
    write_control(0x0c); // Display on, no cursor, no blink
    // pause ();
    write_control(0x38); // 8 bit , 2 lines
    // pause ();
    write_control(0x80); // Set DD RAM address to 0x00
}
```

```
    // pause ();
}

void write_string (_rom char *string)
{
    int i;

    write_control(0x80);
    i=0;
    while (string[i]!=0)
    {
        if (i==16)
            write_control(0x80+40);
        write_data(string[i]);
        i++;
    }
}

void main(void)
{
    lcd_init();

    for (;;)
    {
        write_string("0123456789ABCDEF<_Another_test_>");
        pause();
        write_string(".....");
        pause();
    }
}
```