

# The DSLMU Microcontroller Board Hardware Reference Manual

The DSLMU Microcontroller Board is based on an ARM 32-bit Reduced Instruction Set Computer (RISC) microcontroller and on the Xilinx Virtex-E and Xilinx Spartan-XL Field Programmable Gate Arrays (FPGAs). As ARM Limited does not manufacture its own physical silicon devices (the company is known as a “fab-less shop”), the particular microcontroller used on this board is the Atmel AT91R40008 device, one of dozens available using the ARM core from different manufacturers worldwide.

The DSLMU Microcontroller Board actually consists of two printed circuit boards connected to each other: the MU Board, which contains most of the electronics, and the Expansion Board, containing most of the peripherals and connectors. These boards were designed by staff at the University of Manchester, England, and the School of Electrical Engineering, University of New South Wales, Australia.

## Physical Layout

The DSLMU Microcontroller Board is arranged as shown in Figure 1:

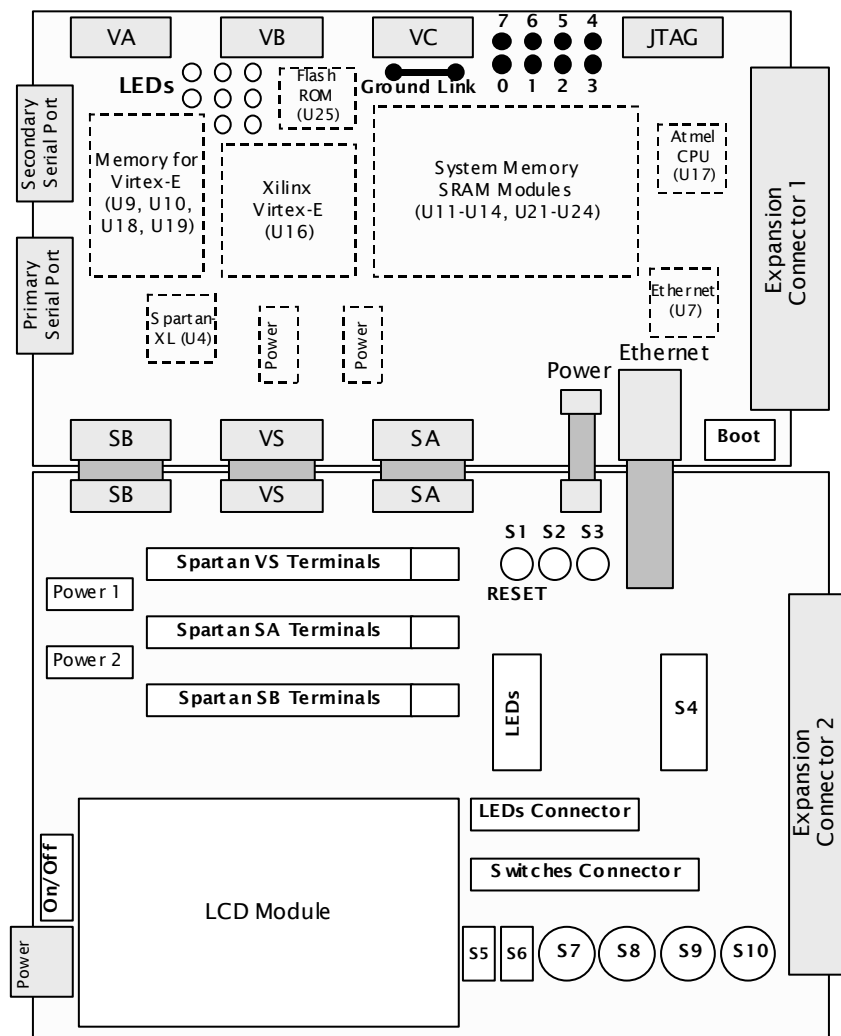


Figure 1: Physical Layout

Note, in particular, the location of the On/Off power switch and the reset button. The electronics on the printed circuit boards are quite delicate, so you should **always turn the power off when wiring up any of your own circuits!**<sup>1</sup>

The *reset button* restarts the Microcontroller Board in the case that the software you have written manages to “crash” so badly that even pressing **Stop** in the GNU Debugger or in Komodo does not help. This reset button should only be used as a last resort...

## System Block Diagram

A systems-level view of the DSLMU Microcontroller Board is shown in Figure 2. You should examine this diagram carefully to understand the operation of the board.

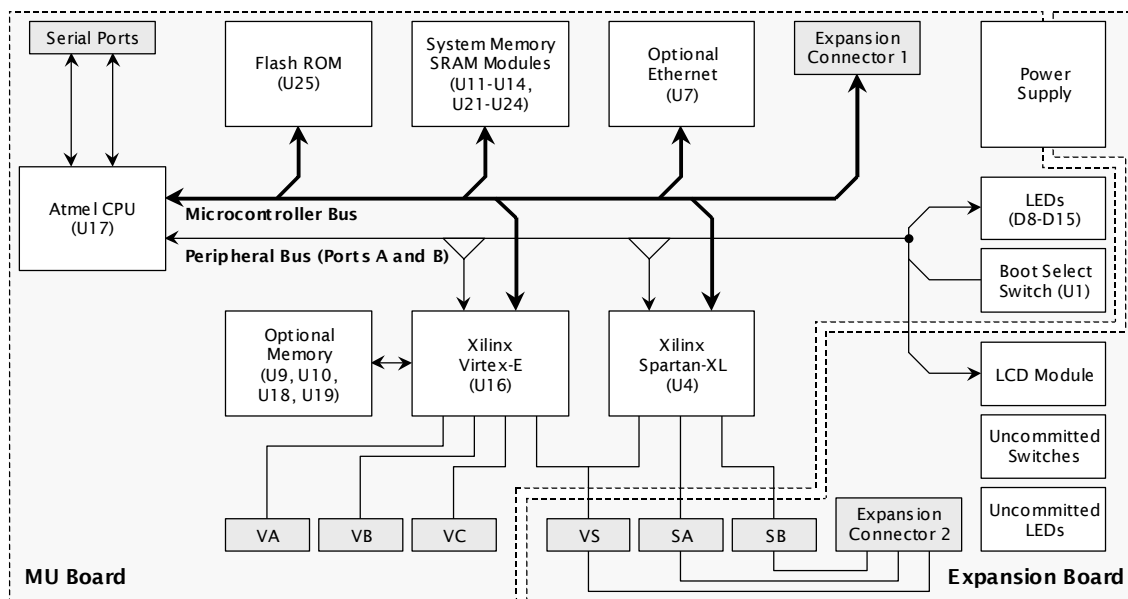


Figure 2: System Block Diagram

## The Microcontroller

The Atmel AT91R40008 microcontroller is at the heart of the DSLMU Microcontroller Board. This microcontroller uses the ARM7TDMI core from ARM Limited and implements the ARMv4T architecture. The AT91R40008 comes in a 100-pin plastic quad-flat (PQFP) package and appears as U17 in the circuit diagram. A 32MHz crystal oscillator (U3) provides the timing signal to the chip.

The microcontroller provides a bus (shown as the Microcontroller Bus in Figure 2) that consists of the address lines A0–A21, the data lines D0–D15 and various control and interrupt signals. Almost everything else on the board is connected to this bus, directly or indirectly. This means that, ultimately, it is the microcontroller’s job to control everything else on the board.

## System Memory

Connected to the Microcontroller Bus is the system memory in the form of Static RAM modules and a Flash ROM module. U11–U14 and U21–U24 provide up to eight 256K × 16-bit

<sup>1</sup> This has the unfortunate side-effect of wiping any program that you might have downloaded into the memory. If you are careful and know what you are doing, you *can* leave the power on while wiring up your own circuits. You must connect the ground (GND) wire first and leave the power (VCC) connection to last, of course, and *do* double-check everything *before* connecting the power!

memory modules for a total of up to 4 MB of read/write memory.<sup>2</sup> This read/write memory is where your programs are stored while being executed on the ARM microcontroller.

U25 provides 2 MB of read-only memory in the form of a Flash ROM. The Flash ROM is where systems-level programs are stored. The Boot Select switch helps determine which of these programs is run at start-up.

## Field Programmable Gate Arrays

The main peripherals connected to the Microcontroller Bus are the two Field Programmable Gate Arrays (FPGAs): the Xilinx Virtex-E and the Xilinx Spartan-XL. These FPGAs are designed to be reprogrammed to become any sort of peripheral that you might want (within limits, of course). For example, you could program the Spartan-XL FPGA to be a Real Time Clock (a device that keeps track of the actual date and time), or to be a latch-and-debounce circuit for input switches.

The Xilinx Spartan-XL FPGA appears as U4 in the circuit. This 100-pin chip comes from a family of devices, each of which has different numbers of programmable gates. The particular FPGA on the DSLMU Microcontroller Board is the XCS10XL-4VQ100C, an integrated circuit having up to 10,000 programmable system gates and a total of 77 programmable input/output pins. Notice that 48 pins of the FPGA are attached to connectors SA, SB and VS on the Expansion Board for input and/or output. You will be using these connectors, and consequently the Spartan-XL FPGA, in many of your experiments.

The Xilinx Virtex-E FPGA appears as U16 in the circuit. This 240-pin chip comes from its own family of rather expensive but highly functional devices. The particular FPGA used on the DSLMU Microcontroller Board is the XCV300E-6PQ240C, with up to 412,000 programmable system gates and 158 programmable input/output pins. Notice that 64 pins of this FPGA are attached to connectors VA, VB and VC on the MU Board and to connector VS on the Expansion Board.

Up to 2 MB of Static RAM can be connected to the Xilinx Virtex-E FPGA. These RAM modules are connected as two banks of 256K × 32 bits and appear as U10, U19, U9 and U18 in the circuit.

One interesting feature of the DSLMU Microcontroller Board is that both the Spartan-XL and Virtex-E FPGAs are connected to the same pins on connector VS. This means that you can do some fancy FPGA programming to make one of the FPGAs control the other, or to make both respond to the same external stimuli. However, you must be careful that you don't program both devices to drive the same pin (ie, as an output) at the same time: the result would be an electrical conflict!<sup>3</sup>

## Connected Peripherals

The DSLMU Microcontroller Board has a number of peripherals connected to the microcontroller, as shown in Figure 2. These include the serial ports, the LEDs on the MU Board, the LCD module and the optional Ethernet controller.

The MU Board has two RS232 serial port connectors, J11 and J13. These are connected to the Atmel AT91R40008 microcontroller via appropriate interface circuitry (see the circuit diagram for more details). The serial ports are capable of transmitting and receiving data at up to 115,200 baud, although without hardware flow control. Connector J11 is dedicated to communicating with the Host PC: it is used to transfer your programs to the Microcontroller Board and then to control them.

As you can see from Figure 2, the LEDs on the MU Board, the Boot Select switch and the Liquid Crystal Display (LCD) module form a “mini-bus” of seven signals, connecting these

---

<sup>2</sup> The DSLMU Microcontroller Boards in the Digital Systems Laboratory are only fitted with 512 KB of memory, appearing as U21.

<sup>3</sup> An *electrical conflict* occurs when two devices try to drive the same signal wire in different directions. For example, think through what would happen if the Virtex-E tries pulling signal VS0 low (to GND), but the Spartan-XL tries setting the same signal high (to VCC). Is the effect the same as a short-circuit? (The answer is, essentially, yes!)

peripherals to the ARM microcontroller and the Xilinx FPGAs.<sup>4</sup> This means that, with appropriate programming, any one of the microcontroller, the Spartan-XL or the Virtex-E can control what is displayed on the LCD module. If you choose to use this feature, note that it becomes *your* responsibility to avoid potential electrical conflicts!

The Cirrus Logic CS8900A device (U7 in the circuit diagram) provides a single-chip 10 Mb Ethernet connection to the DSLMU Microcontroller Board. It is connected directly to the Microcontroller Bus but is usually only directly available to systems-level programs. This device, and associated circuitry, is optional and may not be present on your board.

## Uncommitted Peripherals

The Expansion Board contains a number of uncommitted switches and LEDs. These are “uncommitted” in the sense that they are not connected to anything: you can use these peripherals in your experiments, either connecting them to the FPGA input/output pins via connectors SA, SB and VS, or to your own circuits. The switches include two debounced push-buttons, two undebounced push-buttons, two debounced toggle switches and eight undebounced DIP switches. Ten LEDs (in a bar-graph-style package) are available for use as indicators. The section “Accessing the Peripherals” on page 16 covers these peripherals in greater detail.

## Detailed Schematic Diagrams

You can find the complete schematic diagrams for both the MU Board and the Expansion Board on your CD-ROM in the *board/schem* directory. If possible, you should take the time to examine these diagrams, as they will help you to understand the operation of the board.

Reading a well-drawn circuit diagram is like reading a well-written (and well-commented!) computer program. Learning to read such diagrams will help you know how you should draw your own, making you a better engineer and electronics designer. You can also look at the original diagrams: they are an excellent example of what you should not do!

## The Boot Select Switch

The DSLMU Microcontroller Board includes some simple start-up software in the Flash ROM that is run every time the power is turned on (or the reset button is pressed). This start-up software, in turn, starts (“boots up”) one of up to sixteen different systems-level programs; the Boot Select DIP switch U1, in the bottom right-hand corner of the MU Board, is used to determine which of these programs is started. This switch is shown in Figure 3:

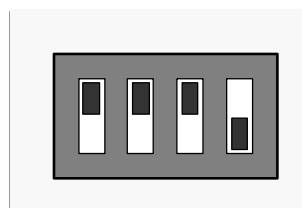


Figure 3: Boot Select Switch

**Important:** Unless your Laboratory documents ask you to do otherwise, you should *always* make sure that the Boot Select switch is set as shown in Figure 3. This selects the Komodo ARM Environment, which is discussed in the remainder of this *Hardware Reference Manual*.<sup>5</sup>

---

<sup>4</sup> The signals forming this bus, called the Peripheral Bus in Figure 2, are LC\_D4-LC\_D7, LC\_EN, LC\_RW and LC\_RS. An additional five signals (LC\_D0-LC\_D3 and LED\_EN) connect the microcontroller to the LEDs and the LCD module, but are *not* connected to the Xilinx FPGAs.

<sup>5</sup> The other switch positions are not discussed in this document; if you are interested, see *MU Board On-board Software* in the *board/doc* directory on your CD-ROM.

## Komodo ARM Environment

The Komodo ARM Environment is a systems-level program stored in the Flash ROM of the DSLMU Microcontroller Board. It is automatically run when the Boot Select switch is set to the position shown in Figure 3.

The main task of the Komodo ARM Environment is to communicate with the Host PC through the primary serial port, to download your own programs into the microcontroller's memory and to run and debug those programs. Komodo also provides you with controlled access to certain microcontroller-based peripherals and to the FPGAs, as well as, of course, to the ARM microcontroller itself.

**Important:** Much of this document only applies when the Komodo ARM Environment is running. You will need to refer to other documentation if you want to write systems-level programs; this documentation can be found on your CD-ROM in the *board/doc* directory.

## Downloading Your Program

You need to use Komodo to download your programs to the DSLMU Microcontroller Board and to debug them. Please see *An Introduction to Komodo* for more information; the following is a brief summary of that document.

To start Komodo, enter the following command line at the shell prompt:

```
kmd &
```

Once the Komodo window appears, you need to download your program to the actual hardware. To do this, click on the *second* **Browse** button in the top right-hand corner (ie, the one associated with the **Load** button, *not* the **Compile** button). A dialog box will appear that allows you to choose your executable; do so, then click OK. Back in the main window, the full file name will appear next to the **Load** button. Now, click on the **Load** button: this will download your executable to the actual board.

To download an FPGA configuration into the Xilinx Spartan-XL device, click on the **Features** button, then select the **Spartan XSC10XL** tab. You can now enter the name of your FPGA program; the relevant file will have a *.bit* extension, as generated by the Xilinx software. To download to the Xilinx Vertex-E FPGA, select the **Virtex-E XCV300E** tab instead.

## Memory Map

The Komodo ARM Environment has the memory map shown in Table 1; this is also known as its address space:

Start Address	End Address	Size	Function
0x00000000	0x003FFFFF	4 MB	Read/write memory (RAM)
0x00400000	0x0FFFFFFF	(252 MB)	(Unused)
0x10000000	0x1FFFFFFF	256 MB	Microcontroller I/O space
0x20000000	0x2FFFFFFF	256 MB	Xilinx Spartan-XL I/O space
0x30000000	0x3FFFFFFF	256 MB	Xilinx Virtex-E I/O space
0x40000000	0xFFFFFFFF	(3072 MB)	(Unused)

**Table 1: Komodo Memory Map**

There are a few points to note about the memory map shown in this table. Firstly, if your DSLMU Microcontroller Board has less than 4 MB of Static RAM modules, the end address of read/write memory will be smaller. For example, if only 512 KB of RAM is included, the end address will be 0x0007FFFF.

Secondly, although each of the memory map regions used by the three input/output spaces is 256 MB in size, most of those regions are "wasted". For example, the Microcontroller I/O

space only occupies 64 bytes out of 256 MB: talk about potential for expansion! Likewise, the Spartan-XL I/O space is only 32 bytes in size and the Virtex-E I/O space is only 64 bytes in size. **You must only access addresses that are valid in that range.**

Thirdly, you access input/output spaces in the same way that you access ordinary memory. In other words, you use the standard `ldr/ldrb/ldrh` and `str/strb/strh` ARM instructions.

Finally, over three-quarters of the memory map is empty and unused. **You should never let your program access these unused memory spaces:** doing so may cause unexpected and erroneous behaviour!

## Read/Write Memory

The read/write portion of the DSLMU Microcontroller Board's system memory appears at address `0x00000000`. It extends to the last address available in that memory; in other words, `0x003FFFFFF` if you have the full 4 MB of Static RAM installed, `0x0007FFFF` if you have only 512 KB, and so on.

All of this read/write memory is available for your use.<sup>6</sup> By default, the GNU Linker places your program at address `0x00008000`, but you can override this by using the `-T` option. For example, to place your program `filename.elf` at address `0x4000`, use the following command line:

```
arm-elf-ld -o filename.elf -Ttext=0x4000 filename.o
```

Please see the `arm-elf-ld(1)` manual page,<sup>7</sup> or refer to the *GNU Linker Reference Manual* on your CD-ROM (in the `gnutools/doc` directory), for more information.

## Microcontroller I/O Space

The Komodo ARM Environment allows you to access certain peripherals in the microcontroller's memory map. Although 256 MB of space is set aside for this purpose, starting at address `0x10000000`, only 64 bytes are defined at the start of this, and even then, most addresses within this range are classified as "Reserved for future use".

Table 2 shows the ports you can access within the Microcontroller I/O space as offsets from address `0x10000000`. These ports are all eight bits wide, and must be accessed with either the `ldrb` or the `strb` ARM instruction:

Offset	Mode	Port Name	Function
<code>0x00</code>	R/W	Port A	Bidirectional data port to LEDs, LCD, etc.
<code>0x04</code>	R/W	Port B	Control port (some bits are read only)
<code>0x08</code>	R/W	Timer	8-bit free-running 1 kHz timer
<code>0x0C</code>	R/W	Timer Compare	Allows timer interrupts to be generated
<code>0x10</code>	RO	Serial RxD	Read a byte from the serial port
<code>0x10</code>	WO	Serial TxD	Write a byte to the serial port
<code>0x14</code>	WO	Serial Status	Serial port status port
<code>0x18</code>	R/W	IRQ Status	Bitmap of currently-active interrupts
<code>0x1C</code>	R/W	IRQ Enable	Controls which interrupts are enabled
<code>0x20</code>	WO	Debug Stop	Stops program execution when written to

Table 2: Microcontoller I/O Space

<sup>6</sup> If you are wondering where the Komodo ARM Environment itself resides, the answer is that it uses an alternative address space and additional on-chip RAM that you cannot access.

<sup>7</sup> This rather cryptic notation simply means you type "`man 1 arm-elf-ld`" at the command line.

## Accessing the Ports

The best way to access the Microcontroller I/O ports from your assembly language programs is to use a base-and-offset technique with the `ldrb` and `strb` ARM instructions. For example:

```
.set    iobase, 0x10000000    ; Base of the Microcontroller I/O space
.set    portA, 0x00          ; Offset of Port A in the I/O space
.set    portB, 0x04          ; Offset of Port B in the I/O space

ldr     r2,=iobase           ; Use R2 as a base address pointer
mov     r0,#0b00010000       ; Set bit 4 and reset all other bits
strb   r0,[r2,#portB]       ; Send the data to Port B (R2 + portB)
mov     r0,#0b10100101       ; R0 = data for the LEDs
strb   r0,[r2,#portA]       ; Send the data to Port A (R2 + portA)
```

### Port A

Port A at offset 0x00 (ie, address 0x10000000) is a bidirectional data port used for a number of purposes. It is connected to the eight LEDs in the top left-hand corner of the MU Board, and to the data bus on the LCD module. These devices are explained further in the section “Accessing the Peripherals”, on page 16. In the circuit diagram, the appropriate signals from Port A are labelled LC\_D0 to LC\_D7; bit 0 of Port A is connected to LC\_D0, and so on.

The direction of Port A is determined by bit 2 of Port B: if that bit is set to 0 (the default), all eight bits of Port A are outputs. If it is set to 1, all eight bits of Port A are inputs.

Bits 4–7 of Port A are also connected to both the Xilinx Spartan-XL and Xilinx Virtex-E FPGAs and are shown in Figure 2 as part of the Peripheral Bus. With some clever programming, you can use these four bits as a general I/O interface to the FPGAs. You *must* ensure, however, that only one device writes to the LC\_D4-LC\_D7 signals at any one time. In other words, if Port A is defined to be an output, the FPGAs must define the appropriate pins as inputs. Conversely, if one of the FPGAs defines these pins as outputs, then Port A must be programmed to be an input port (and the other FPGA must also define the pins as inputs). **Failure to keep this rule can cause electrical failure!**

By referring to Figure 1, you can see that eight test-points are available between connectors VC and JTAG. These test-points (labelled TP0 to TP7 on the MU Board) correspond to bits 0–7 of Port A. You can connect an oscilloscope to these test-points (using the Ground Link as the grounding point) to investigate what your program is writing to this port.

### Port B

Port B at offset 0x04 (ie, address 0x10000004) is an 8-bit control and status port: most bits define how the peripherals attached to Port A are to be controlled. The bit definitions for this port are shown in Table 3:

Bit	Mode	Function
7	RO	Push-button switch S2 on the Expansion Board: 1 = pressed
6	RO	Push-button switch S3 on the Expansion Board: 1 = pressed
5	R/W	LCD backlight: 0 = disabled, 1 = enabled (has no effect)
4	R/W	LEDs enable: 0 = disabled, 1 = enabled
3	—	(Reserved: must be written as 0)
2	R/W	Port A direction, LC_RW: 0 = output (write), 1 = input (read)
1	R/W	LC_RS: 0 = control register, 1 = data register
0	R/W	LC_EN: 0 = disabled, 1 = enabled

**Table 3: Port B Bit Definitions**

Writing to Port B (using the `strb` ARM instruction) will set or reset the appropriate bits; it will also drive particular signals on the physical board high or low. Reading from this port (using the `ldrb` ARM instruction) will usually return the last value written to each bit.

**Bits 0–2** are used to control the LCD module and correspond to the LC\_EN, LC\_RS and LC\_RW signals respectively. Bit 2 (LC\_RW) also controls the direction of Port A: if set to zero, all eight bits of Port A become outputs. If set to one, these eight bits become inputs.

**Bit 3** is reserved for future use: you must always write 0 to this bit.

**Bit 4** is used to allow or disallow data to be written to the LEDs on the MU Board. Writing a one to this bit makes the LEDs display whatever is *currently* on Port A. Writing a zero to this bit disables the LEDs entirely (ie, turns the LEDs off).<sup>8</sup> This bit corresponds to LED\_EN in the circuit diagram.

**Bit 5** is connected to the NLCD\_BK\_LT signal on the MU Board and controls the LCD backlight. Setting or resetting this bit has no effect on your board, as the LCD does not have a backlight installed. For this reason, you should *always* set this bit to zero.

Reading **bits 6 and 7** returns the current state of push-button switches S2 and S3 on the Expansion Board. Writing to these bits has no effect. See page 18 for more information on using these switches.

## Timer Port

The Timer port at offset 0x08 (ie, address 0x10000008) provides an 8-bit timer that “runs free” at 1 kHz. The purpose of a free-running timer is to accurately measure time intervals and allow you to write time-dependant delay loops. Reading a byte from this port will return a value between 0 and 255 (0x00 and 0xFF). The value available for reading will be incremented every 1 ms (ie, 1000 times a second); when the value reaches 0xFF, it will be automatically reset to 0x00.

Writing to this port will set the timer to the value written. Please be aware that, due to internal timing issues, the first iteration of the newly-set timer may be up to -1 ms of its true time. In other words, the first increment may be made immediately, or it might only happen almost 1 ms after the timer reset. Increments after this should be accurate.

## Timer Compare Port

The Timer Compare port at offset 0x0C (ie, address 0x1000000C) is used by the internal timer to generate interrupts: if bit 0 of the IRQ Enable port is set to 1, an interrupt to the ARM microcontroller is generated every time the Timer port value equals the value stored in the Timer Compare port.

Please note that, once the interrupt is generated, the Timer port value is *not* reset to zero: that is your program’s responsibility. By setting the Timer port to zero on each interrupt, your program gains the ability to generate interrupts with intervals of between 1 ms and 255 ms,<sup>9</sup> that is, between 1000 times a second to just under 4 times a second.

## Serial RxD Port

The Serial RxD port at offset 0x10 (ie, address 0x10000010) allows you to read the user input stream from the Host PC.<sup>10</sup> Assuming bit 0 of the Serial Status port is 1, reading a value from this Serial RxD port retrieves the next byte from the user input stream. If bit 0 of the Serial Status port is zero, reading a value from this Serial RxD port returns an

---

<sup>8</sup> For those technically inclined, the reason that the LEDs only ever display what is *currently* on Port A, instead of latching values written to Port A, is that the MU Board uses a 74HC244 octal line buffer for U6 instead of a 74HC373 octal transparent latch.

<sup>9</sup> You can generate interrupts every 256 ms if you do *not* reset the Timer port, in other words, if you allow the timer to roll over.

<sup>10</sup> The *user input stream* contains the ASCII codes of key-presses typed into the Terminal window of the debugger on the Host PC. If you are wondering how the microcontroller “knows” how to read bytes from the debugger, the answer is that it is actually the Komodo ARM Environment doing all of the hard work: it translates your `ldr` instruction into the ARM instructions needed to do the job. This allows the same physical serial port cable to be used both for user input *and* for controlling your program via the debugger.

undefined value. The Serial RxD port is read-only: writing to offset 0x10 actually accesses the Serial TxD port.

An example is probably the best way to explain the Serial RxD port. Assume the user pressed the keys J N Z ENTER, in that order, in the Terminal window of the debugger.<sup>11</sup> This would place the bytes 0x4A, 0x4E, 0x5A and 0x0A into the input stream. The following code fragment would then read those four bytes:

```

.set    iobase,    0x10000000 ; Base of Microcontroller I/O space
.set    ser_RxD,  0x10       ; Serial RxD port
.set    ser_stat, 0x14       ; Serial Status port
.set    ser_Rx_rdy, 0b01     ; Test bit 0 for RxD ready status

readfour:
    bl    readbyte           ; Read first byte (0x4A = 'J')
    bl    readbyte           ; Read second byte (0x4E = 'N')
    bl    readbyte           ; Read third byte (0x5A = 'Z')
    bl    readbyte           ; Read fourth byte (0x0A = '\n')
    ...

readbyte:           ; Wait for and read a byte from the user input stream into R0;
                   ; destroys the contents of R1.

    ldr    r1,=iobase       ; R1 = base address of I/O space
rb1:
    ldrb   r0,[r1,#ser_stat] ; Read the serial port status
    tst    r0,#ser_Rx_rdy   ; Check whether a byte is ready to be read
    beq    rb1              ; (No: jump back and try again)

    ldrb   r0,[r1,#ser_RxD] ; Read the available byte into R0
    mov    pc,r1            ; and return to caller

```

## Serial TxD Port

The Serial TxD port at offset 0x10 (ie, address 0x10000010) allows you to write to the user output stream to the Host PC. You should always check that bit 1 of the Serial Status port is 1 before writing to this port. Anything written to this port (ie, written into the user output stream) will appear in the Terminal window of the debugger on the Host PC. The standard ASCII table is used to display bytes written in this way; writing the character 0x0A<sup>12</sup> will start a new line in that window.

The Serial TxD port is write-only: reading from offset 0x10 actually accesses the Serial RxD port.

## Serial Status Port

The Serial Status port at offset 0x14 (ie, address 0x10000014) indicates the status of the user input and output streams that feed the Serial RxD and Serial TxD ports respectively.

**Bit 0** indicates whether a byte is available for reading from the Serial RxD port. If this bit is one, a byte representing the user's input can be read. If this bit is zero, nothing is available to be read; in this case, reading from the Serial RxD port will return an undefined value.

**Bit 1** indicates whether the serial port transmitter (connected to the user output stream) is ready to accept a byte of data. If this bit is one, you can safely write to the Serial TxD port. If this bit is zero, writing to the Serial TxD port can cause previously-written data to be lost.

This port is read-only: writing to it has no effect.

---

<sup>11</sup> To open the Terminal window from within the Komodo debugger, click on the **Features** button, then select the **Terminal** tab. You will need to make sure that **Active** is selected.

<sup>12</sup> The character 0x0A, also known as LF, can be represented as '\n' in C and assembly language programs. '\n' is known as the new-line character in Unix and POSIX-compliant systems.

## IRQ Status

The IRQ Status port at offset 0x18 (ie, address 0x10000018) indicates whether or not peripherals are trying to interrupt the ARM microcontroller. A bit set to one in this port indicates that peripheral is generating an interrupt; a zero means it is not. The bit definitions for this port are shown in Table 4:

Bit	Mode	Function
7	R/W	Push-button switch S2 on the Expansion Board
6	R/W	Push-button switch S3 on the Expansion Board
5	R/W	Serial port transmitter ready
4	R/W	Serial port receiver ready
3	—	(Reserved)
2	R/W	Xilinx Virtex-E interrupt request
1	R/W	Xilinx Spartan-XL interrupt request
0	R/W	Timer Compare interrupt request

**Table 4: IRQ Status Port Bit Definitions**

Apart from reading this port (to see which peripheral generated the interrupt), you can also write to it to set (using a 1) or clear (using a 0) interrupt requests. Your interrupt handler should *always* write a 0 into the appropriate bit to indicate that it has handled the interrupt.

## IRQ Enable Port

The IRQ Enable port at offset 0x1C (ie, address 0x1000001C) is used by the peripheral controller to allow or disallow a peripheral from interrupting the ARM microcontroller. Reading this port returns the last value written to it.

When a bit in this port is set to 1, and that peripheral generates an interrupt (setting the corresponding IRQ Status port bit to 1), an IRQ Exception is generated by the ARM microcontroller. When the bit in this port is set to 0, no IRQ Exception is generated, even if the peripheral tries to interrupt (by setting the corresponding IRQ Status port bit to 1). The bit definitions for this port are exactly the same as for the IRQ Status port, as shown in Table 4.

Once the IRQ Exception is generated, the ARM microcontroller saves the current execution address and register flags, then jumps to the code stored at address 0x00000018. Please see page A2-19 in the *ARM Architecture Reference Manual* (page 51 in the PDF version) for more details; you can find this document in the *reference* directory on your CD-ROM.

## Debug Stop Port

The Debug Stop port at offset 0x20 (ie, address 0x10000020) is used by the Komodo ARM Environment as an interface to the debugger running on the Host PC. Writing any value to this port (using the `strb` ARM instruction) will stop your program's execution and return control to the debugger. You should only access this port when debugging—it is *not* meant to be a general-purpose method of stopping your programs!<sup>13</sup>

---

<sup>13</sup> If you are wondering how the microcontroller “knows” how to access the debugger in a remote (to it) location, the answer is that it is all an illusion: the Komodo ARM Environment performs considerable trickery on your behalf, so that a simple write to this port is translated into many ARM instructions that do what is needed. If you are interested, you can find the source code to the Komodo ARM Environment in the *board/src/monitor* directory on your CD-ROM; look at the file *imp.s* in particular.

## Xilinx Spartan-XL FPGA

One of the main peripherals on the DSLMU Microcontroller Board is the Xilinx Spartan-XL XCS10XL-4VQ100C Field Programmable Gate Array (FPGA). This 100-pin device appears as U4 in the circuit; it contains up to 10,000 programmable system gates and a total of 77 programmable input/output pins. This FPGA is designed to be reprogrammed to become any sort of peripheral that you might want (within the limits of the device, of course).

### External Connections

As you can see from Figure 2, there are three main groups of signals (“buses”) attached to the Xilinx Spartan-XL FPGA on the DSLMU Microcontroller Board.

The first group of signals is the *Microcontroller Bus*. This connection consists of 16 signals: the data lines D0–D7, the address lines A0–A4, a chip select signal (SPARTAN\_CS), a read signal (NRD\_NOE) and a write signal (NWR0\_NWE). These signals allow the ARM microcontroller, and hence your programs, to communicate with the Xilinx Spartan-XL FPGA.

The second group is the *Peripheral Bus*, which consists of seven signals: the four data lines LC\_D4–LC\_D7,<sup>14</sup> the LCD Enable signal (LC\_EN), the LCD Read/Write signal (LC\_RW) and the LCD Register Select signal (LC\_RS). This means that, if programmed correctly, the FPGA can control the LCD module on the Expansion Board. However, you *must* first disable Ports A and B in the microcontroller; **this is not possible in the current version of the Komodo ARM Environment.**

The third group consists of the 48 signals routed to the connectors SA, SB and VS on the Expansion Board. These signals are labelled SA0–SA15, SB0–SB15 and VS0–VS15 in the circuit diagram. Note that the signals VS0–VS15 are also connected to the Xilinx Virtex-E FPGA: with some fancy FPGA programming, you can use these signals to communicate between the two FPGAs. Please remember, however, that you must ensure that only one device drives any given pin (as an output) at any one time. **Breaking this rule can lead to electrical conflicts!**

A number of other signals connect the ARM microcontroller to the Xilinx Spartan-XL FPGA. However, most of these are only meaningful in systems-level programs. One signal that is important is S\_IRQ: pulling this high generates an interrupt request to the ARM microcontroller. This interrupt request is reflected in bit 1 of the IRQ Status port.

### Microcontroller Interface

As already mentioned, the Microcontroller Bus signals connect the Xilinx Spartan-XL FPGA to the ARM microcontroller. These 16 signals allow your programs to access the FPGA using standard `ldrb` and `strb` ARM instructions.

As you can see from Table 1, this FPGA can be accessed by reading from and writing to the Xilinx Spartan-XL I/O space. In fact, as only five address lines are routed to the FPGA, only 32 unique addresses are valid in this I/O space: addresses 0x20000000 to 0x2000001F. You must only access the FPGA using addresses in this range.

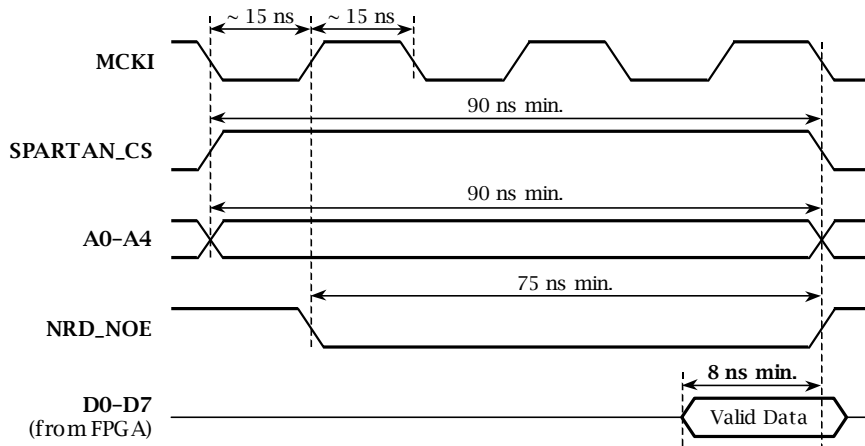
Please note that the effects of reading from or writing to valid addresses within the Xilinx Spartan-XL I/O space depend on how the FPGA was programmed, in other words, what FPGA design is currently in the device. You can make the Spartan-XL to respond in any way appropriate (or inappropriate!) by placing your own FPGA design into the device.

If you want your FPGA design to correctly interface to the ARM microcontroller, you must meet the specific timing requirements that that microcontroller places on the Xilinx Spartan-XL. This is because the Spartan-XL FPGA is just one of the devices attached to the Microcontroller Bus; each device on this bus must follow the standard Atmel AT91 read/write

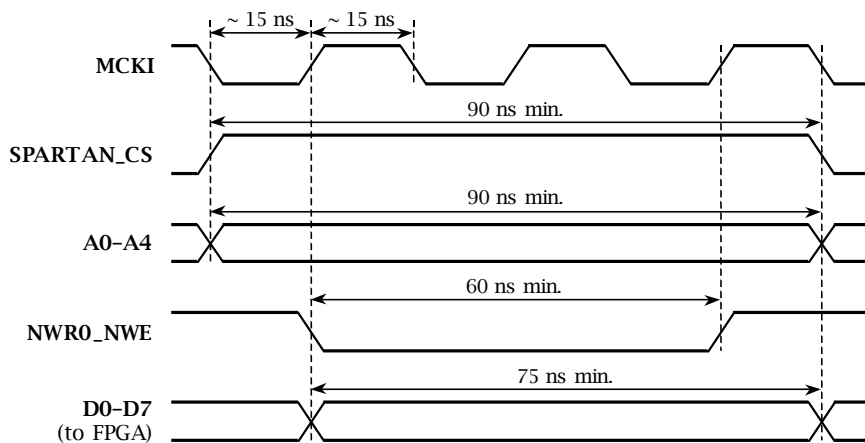
---

<sup>14</sup> Please note that the four signals LC\_D0–LC\_D3 are *not* connected to the FPGA. This means that the LCD Module must be controlled using the four-bit interface mode, not the eight-bit interface mode. See the section “Accessing the Peripherals” on page 16 for more information.

timings. This allows the ARM microcontroller to correctly access the peripherals using standard `ldr/ldrb/ldrsh/ldrh/ldrsh` and `str/strb/strh` ARM instructions. In the specific case of the Spartan-XL FPGA, the timing requirements are shown in Figure 4 (for reading from the FPGA into the microcontroller) and in Figure 5 (for writing to the FPGA from the microcontroller). MCKI is the 32 MHz system clock (generated by U3) that is connected to pad P54 on the Xilinx Spartan-XL FPGA; all timings are rounded to the nearest nanosecond.



**Figure 4: Read from Xilinx Spartan-XL FPGA Timing Diagram**



**Figure 5: Write to Xilinx Spartan-XL FPGA Timing Diagram**

Please note that the actual details of creating and implementing a design for the Spartan-XL FPGA is beyond the scope of this document.

## FPGA Pin Configuration

Table 5 shows the definitions of each *pad* (pin) on the Xilinx Spartan-XL XCS10XL-4VQ100C FPGA. You will need this information to define the input and/or output interfaces needed for your design using; how to do so is beyond the scope of this document.

A few notes are in order regarding this table. The **Pad** column indicates the pin number, as specified by the device's datasheet. Only user-programmable pads that are accessible on the DSLMU Microcontroller Board have been included in this table. The **Name** column gives the Xilinx name for this pin. The **Signal** column gives the name of the signal, as used in the MU Board circuit diagram. The **Dir** column indicates the direction in which this pin must be programmed; "Any" means that you can program this pin to be any direction you wish. The **Notes** column refers to comments following the table.

Please note that you most likely will need to refer to the datasheet for the Xilinx Spartan-XL device as you read this table; you can find this datasheet in the *board/doc/data* directory on your CD-ROM.

Pad	Name	Signal	Dir	Notes
P2	I/O, GCK1	SA0	Any	1
P3	I/O	SA1	Any	1
P4	I/O, TDI	NRD_NOE	In	2, 3
P5	I/O, TCK	SPARTAN_CS	In	2, 4
P6	I/O, TMS	NWR0_NWE	In	2, 5
P7	I/O	SA2	Any	1
P8	I/O	SA3	Any	1
P9	I/O	SA4	Any	1
P10	I/O	SA5	Any	1
P13	I/O	SA6	Any	1
P14	I/O	SA7	Any	1
P15	I/O	SA8	Any	1
P16	I/O	SA9	Any	1
P17	I/O	SA10	Any	1
P18	I/O	SA11	Any	1
P19	I/O	SA12	Any	1
P20	I/O	SA13	Any	1
P21	I/O, GCK2	SA14	Any	1
P27	I/O, GCK3	SA15	Any	1
P28	I/O, HDC	FIQ_B0	In/Out	6
P29	I/O	SB0	Any	7
P31	I/O	SB1	Any	7
P32	I/O	SB2	Any	7
P33	I/O	SB3	Any	7
P34	I/O	SB4	Any	7
P35	I/O	SB5	Any	7
P36	I/O, INIT#	NSPARTAN_INIT	In	8
P39	I/O	SB6	Any	7
P40	I/O	SB7	Any	7
P41	I/O	SB8	Any	7
P42	I/O	SB9	Any	7
P43	I/O	SB10	Any	7
P44	I/O	SB11	Any	7
P45	I/O	SB12	Any	7
P46	I/O	SB13	Any	7
P47	I/O	SB14	Any	7
P48	I/O, GCK4	SB15	Any	7
P53	I/O, D7	D7	In/Out	2, 9
P54	I/O, GCK5	MCK1	In	10
P55	I/O, D6	D6	In/Out	2, 9
P56	I/O	A4	In	2, 11
P57	I/O, D5	D5	In/Out	2, 9
P58	I/O	A1	In	2, 11
P59	I/O	A2	In	2, 11
P60	I/O	A3	In	2, 11
P61	I/O, D4	D4	In/Out	2, 9
P62	I/O	A0	In	2, 11
P65	I/O, D3	D3	In/Out	2, 9
P66	I/O	LC_D4	In, I/O	12, 13
P67	I/O	LC_D5	In, I/O	12, 13
P68	I/O, D2	D2	In/Out	2, 9
P69	I/O	LC_D6	In, I/O	12, 13
P70	I/O, D1	D1	In/Out	2, 9
P71	I/O	LC_D7	In, I/O	12, 13
P72	I/O, D0	D0	In/Out	2, 9
P73	I/O, GCK6	BAUD_CLK	In	14
P74	CCLK	SPARTAN_CCLK	In, Out	15
P76	O, TDO	S_IRQ	Out	16
P78	I/O	LC_RW	In, Out	17, 18
P79	I/O, GCK7	TIM_B1	In	19
P80	I/O, CS1	LC_RS	In, Out	17, 18
P81	I/O	LC_EN	In, Out	17, 18
P82	I/O	VS0	Any	20
P83	I/O	VS1	Any	20
P84	I/O	VS2	Any	20
P85	I/O	VS3	Any	20
P86	I/O	VS4	Any	20
P87	I/O	VS5	Any	20
P90	I/O	VS6	Any	20
P91	I/O	VS7	Any	20
P92	I/O	VS8	Any	20
P93	I/O	VS9	Any	20
P94	I/O	VS10	Any	20
P95	I/O	VS11	Any	20
P96	I/O	VS12	Any	20
P97	I/O	VS13	Any	20
P98	I/O	VS14	Any	20
P99	I/O, GCK8	VS15	Any	20

**Table 5: Xilinx Spartan-XL Pin Configuration**

**Notes on Table 5:**

1. Connected to connector SA on the Expansion Board.
2. Microcontroller Bus signal, connected to the ARM microcontroller.
3. Active-low read signal (to send data from the FPGA to the ARM microcontroller).
4. Active-high chip select signal.
5. Active-low write signal (to accept data from the ARM microcontroller to the FPGA).
6. Connected to the Fast IRQ/P12 pin on the ARM microcontroller (useful only in systems-level programs).
7. Connected to connector SB on the Expansion Board.
8. Also connected to switch S3 on the Expansion Board; Low = pressed.
9. Data signal from the ARM microcontroller.
10. 32 MHz system clock, generated by U3.
11. Address signal from the ARM microcontroller.
12. Peripheral Bus signal, connected to the LCD module and to Port A on the ARM microcontroller.

13. If the FPGA is to control the LCD module, program to be an input/output pin and disable Ports A and B in the Microcontroller I/O space (not possible in the current version of the Komodo ARM Environment). Otherwise (and by default), program to be an input-only pin.
14. Serial port baud clock from the ARM microcontroller.
15. Externally-derived clock for programming the Xilinx Spartan-XL FPGA. After configuration, this signal can only be used for read-back (useful in systems-level mode only).
16. Active-high signal connected to the IRQ 0 pin on the ARM microcontroller.
17. Peripheral Bus signal, connected to the LCD module and to Port B on the ARM microcontroller.
18. If the FPGA is to control the LCD module, program to be an output-only pin and disable Ports A and B in the Microcontroller I/O space (not possible in the current version of the Komodo ARM Environment). Otherwise (and by default), program to be input-only pin.
19. Connected to the programmable timer on the ARM microcontroller. This signal is also called TIOB2.
20. Connected to connector VS on the Expansion Board.

## Default Configuration

The Komodo ARM Environment provides a default configuration for the Xilinx Spartan-XL FPGA. This means that you can use this device from within your programs without needing to first program the FPGA. You can find the schematic diagram of this default configuration in the *board/schem* directory on your CD-ROM.

The default configuration implements six 8-bit programmable bidirectional ports that are attached to connectors SA, SB and VS on the Expansion Board. Table 6 shows the ports you can access for this configuration as offsets from address 0x20000000. These ports are all eight bits wide, and must be accessed with either the `ldr` or the `str` ARM instruction:

Offset	Mode	Port Name	Function
0x00	R/W	SA_L Data	Data input and/or output for SA0-SA7
0x01	R/W	SA_L Control	Control port for SA0-SA7
0x02	R/W	SA_H Data	Data input and/or output for SA8-SA15
0x03	R/W	SA_H Control	Control port for SA8-SA15
0x04	R/W	SB_L Data	Data input and/or output for SB0-SB7
0x05	R/W	SB_L Control	Control port for SB0-SB7
0x06	R/W	SB_H Data	Data input and/or output for SB8-SB15
0x07	R/W	SB_H Control	Control port for SB8-SB15
0x08	R/W	VS_L Data	Data input and/or output for VS0-VS7
0x09	R/W	VS_L Control	Control port for VS0-VS7
0x0A	R/W	VS_H Data	Data input and/or output for VS8-VS15
0x0B	R/W	VS_H Control	Control port for VS8-VS15

**Table 6: Default Xilinx Spartan-XL I/O Space**

As this table shows, there are really only two types of ports in this default configuration: data ports and control ports. *Data ports* are connected to the corresponding pins; these pins can be individually programmed to be inputs or outputs. *Control ports* allow you to set the direction of the corresponding data port pins.

The control ports have the following bit definitions; each bit controls the direction of the associated pin:

Bit	Port SA_L	Port SA_H	Port SB_L	Port SB_H	Port VS_L	Port VS_H
7	SA7	SA15	SB7	SB15	VS7	VS15
6	SA6	SA14	SB6	SB14	VS6	VS14
5	SA5	SA13	SB5	SB13	VS5	VS13
4	SA4	SA12	SB4	SB12	VS4	VS12
3	SA3	SA11	SB3	SB11	VS3	VS11
2	SA2	SA10	SB2	SB10	VS2	VS10
1	SA1	SA9	SB1	SB9	VS1	VS9
0	SA0	SA8	SB0	SB8	VS0	VS8

**Table 7: Mapping of Port Bits to Pins**

Writing a one to any control port bit makes the corresponding pin an input. Writing a zero makes the corresponding pin an output. Reading the control port returns the last value written to it. At reset, all pins are initialised as inputs.

The data ports have the same bit definitions as shown in Table 7; each bit is connected to its corresponding pin.

When a pin is configured as an output, writing a one to the corresponding data port bit makes the pin High (ie, sets the output to be VCC, in other words, 3.3V). Writing a zero to the bit makes the pin Low (ie, sets the output to be GND, that is, 0V). Reading this bit returns the last value written to it.

When a pin is configured as an input, reading the corresponding data port bit returns the current value of the pin: 1 is returned if the pin is High, 0 if it is Low. Writing to this bit (while defined as an input) sets the *future* value of the pin: the value the pin will become if it is ever redefined to be an output.

An example is probably the best way to illustrate how these ports can be used from within your own programs:

```
.set    spartan_base, 0x20000000 ; Base of the Spartan-XL I/O space
.set    VS_H_data,    0x0A      ; Offset to Port VS_H Data
.set    VS_H_ctrl,    0x0B      ; Offset to Port VS_H Control

ldr     r2,=spartan_base        ; Use R2 as a base address pointer
mov     r0,#0b11000111          ; R0 = value for Port VS_H Control
; VS8 = input VS12 = output
; VS9 = input VS13 = output
; VS10 = input VS14 = input
; VS11 = output VS15 = input

strb   r0,[r2,#VS_H_ctrl]      ; Set directions of VS8-VS15
mov     r0,#0b00101011         ; R0 = value for Port VS_H Data
; VS8 = High VS12 = Low
; VS9 = High VS13 = High
; VS10 = Low VS14 = Low
; VS11 = High VS15 = Low
; (Only VS11, VS12 and VS13 will be changed;
; other values are for the future)

strb   r0,[r2,#VS_H_data]      ; Set values of VS8-VS15 (VS11-VS13)
ldrb   r1,[r2,#VS_H_data]      ; R1 = current values and inputs for VS8-
; VS15 (bits 3-5, ie, VS11, VS12 and VS13,
; are previously written values, 101 in
; binary)

mov     r0,#0b00000000         ; Make VS8-VS15 all to be outputs
strb   r0,[r2,#VS_H_ctrl]      ; Now VS8-VS10 and VS14-VS15 become values
; set previously, ie, 0b00XX011. VS11-VS13
; retain their previous value of 101.
```

## Accessing the Peripherals

As already shown in Figure 1 and Figure 2, the DSLMU Microcontroller Board contains a number of connected peripherals and a number of uncommitted peripherals.

The *connected peripherals* are those connected directly to the ARM microcontroller. These include the eight LEDs on the MU Board, the LCD module and the two push-button switches S2 and S3. These peripherals can be accessed via Port A and/or Port B in the Microcontroller I/O space.

The *uncommitted peripherals* are those on the Expansion Board that are not connected to the ARM microcontroller. These include eight undebounced switches in a DIP-style package (S4), two debounced toggle switches (S5 and S6), two debounced push-buttons (S7 and S8), two undebounced push-buttons (S9 and S10) and ten LED indicators in a bar-graph-style package (D1). You can connect these peripherals to the FPGA input/output pins via connectors SA, SB and VS, or to your own circuits, as required for your experiments.

### On-board LEDs

The eight LEDs in the top left-hand corner of the MU Board are connected to the eight bits of Port A in the Microcontroller I/O space, and are controlled by bit 4 of Port B. Each LED is connected to one bit of Port A in the arrangement shown in Figure 6:

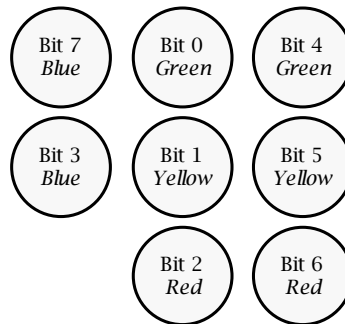


Figure 6: On-board LEDs

Writing a one to any one of these bits in Port A will turn that LED on (assuming the LEDs have been enabled). Writing a zero will turn that LED off.

Writing a one to bit 4 of Port B enables the LEDs: it makes the LEDs display whatever is *currently* on Port A. Writing a zero to this bit disables the LEDs entirely (ie, turns the LEDs off).

A typical program to display an appropriate pattern on the LEDs is:

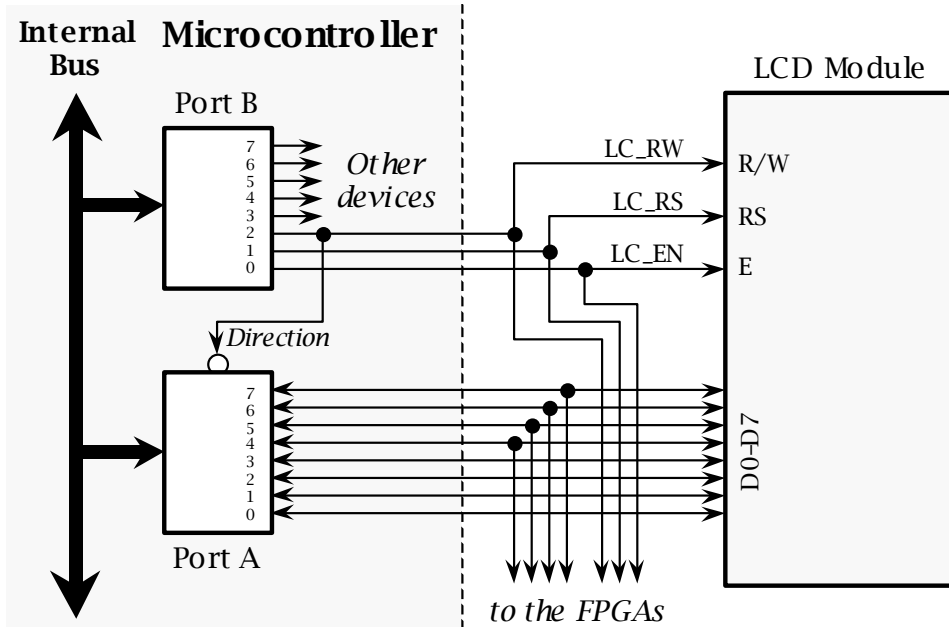
```
.set    iobase, 0x10000000    ; Base of the Microcontroller I/O space
.set    portA,  0x00         ; Offset to Port A in the I/O space
.set    portB,  0x04         ; Offset to Port B in the I/O space

ldr     r2,=iobase          ; Use R2 as a base address pointer

mov     r0,#0b00010000      ; Enable the LEDs (set bit 4)
strb   r0,[r2,#portB]      ; Send the data to Port B
mov     r0,#0b01010101      ; Turn on both green and both red LEDs
strb   r0,[r2,#portA]      ; Send the data to Port A
```

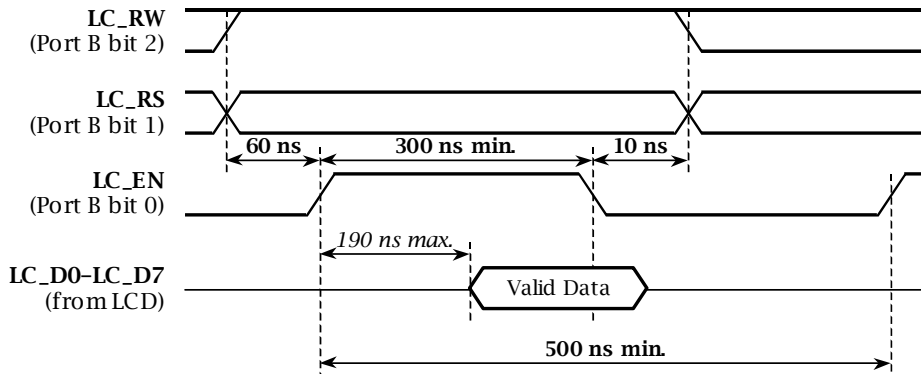
### LCD Module

The DSLMU Microcontroller Board comes with a 20-character  $\times$  4-line Liquid Crystal Display (LCD) module. This module can be controlled via the eight bits of Port A (the LCD data bus) and bits 0–2 of Port B (the LCD control signals) in the Microcontroller I/O space, and via the Peripheral Bus signals attached to the FPGAs. This arrangement is shown in Figure 7.

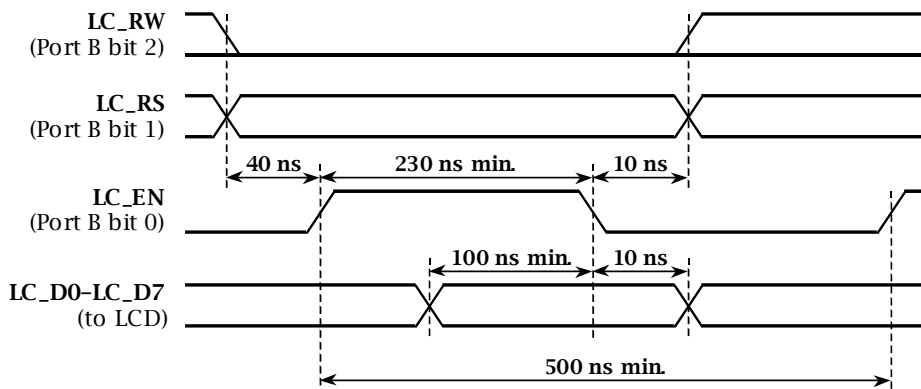


**Figure 7: Physical Interface to the LCD Module**

The LCD module is quite a complex device and supports both read and write operations. Writing to the device is accomplished by setting bit 2 of Port B (LC\_RW) to zero; reading from the device is done by setting this bit to 1. In either case, you must satisfy the requirements shown in **bold** in the timing diagrams of Figure 8 and Figure 9:



**Figure 8: Read from LCD Module Timing Diagram**



**Figure 9: Write to LCD Module Timing Diagram**

These timing requirements are more than adequately satisfied<sup>15</sup> under the Komodo ARM Environment by the following code fragments. Assume the following lines have been defined:

```
.set    iobase, 0x10000000 ; Base of Microcontroller I/O space
.set    portA, 0x00       ; Offset of Port A
.set    portB, 0x04       ; Offset of Port B
.set    LC_RW, 0b00000100 ; Bit 2: LCD Read/Write signal
.set    LC_RS, 0b00000010 ; Bit 1: LCD Register Select signal
.set    LC_EN, 0b00000001 ; Bit 0: LCD Enable signal

ldr    r2,=iobase        ; Use R2 as the base address pointer
```

Writing to the LCD module can then be done with a minimum of four `strb` ARM instructions. For example, the following code writes to the LCD Data register:

```
write_lcd_data:    ; Write data in R0 to LCD Data register; destroys R1
...
mov    r1,#(LC_RS | (! LC_RW) | (! LC_EN)) ; Set R1 to LCD signal
strb   r1,[r2,#portB] ; LC_RS but not LC_RW and not LC_EN
strb   r0,[r2,#portA] ; Write data to LCD
eor    r1,r1,#LC_EN   ; Toggle LC_EN bit of R1
strb   r1,[r2,#portB] ; This enables the LCD module
eor    r1,r1,#LC_EN   ; Toggle LC_EN again to disable the LCD module
strb   r1,[r2,#portB] ; LCD module latches the data at this point
...

```

Similarly, reading from the LCD module can be done with a minimum of three `strb` ARM instructions and one `ldrb` instruction. For example, the following code reads from the LCD Control register:

```
read_lcd_control: ; Read from LCD Control register to R0
...
mov    r1,#((! LC_RS) | LC_RW | (! LC_EN)) ; Set R1 to LCD signal
strb   r1,[r2,#portB] ; LC_RW but not LC_RS and not LC_EN
eor    r1,r1,#LC_EN   ; Toggle LC_EN bit of R1
strb   r1,[r2,#portB] ; This enables the LCD module
ldrb   r0,[r2,#portA] ; Read the data into R0
eor    r1,r1,#LC_EN   ; Toggle LC_EN again
strb   r1,[r2,#portB] ; This disables the LCD module
...

```

Please note that these routines are *not* complete by themselves: any routines that access the LCD module should always check the Busy flag before writing to it! This flag can be accessed by reading bit 7 of Port A when Port B is set to 0b00000101 (ie, LC\_RW and LC\_EN set to 1, LC\_RS set to 0). Reading 1 (0) from bit 7 of Port A indicates a the LCD controller is busy (idle).

Control and Programming the LCD module correctly and efficiently can be quite complicated and is beyond the scope of this document. You will need to consult the *Optrex Character LCD Module User's Manual* for more information on; you can find this document on your CD-ROM in the `board/doc/data` directory. A good on-line reference on controlling and programming the LCD module can be found at <http://home.iae.nl/users/pouweha/lcd/lcd.shtml>.

## Debounced Switches S2 and S3

Push-button switches S2 and S3 on the Expansion Board<sup>16</sup> are connected to bits 7 and 6 of Port B, respectively, in the Microcontroller I/O space. Reading these two bits returns the current state of the switches: if a bit is set to 1, that switch is currently being pressed. For example:

```
.set    iobase, 0x10000000 ; Base of the Microcontroller I/O space
.set    portB, 0x04       ; Offset to Port B in the I/O space
```

<sup>15</sup> The reason is that an average `mov`, `ldrb` or `strb` instruction takes approximately 13  $\mu$ s to execute under the Komodo ARM Environment. This assumes, of course, that U3 generates a 32 MHz clock.

<sup>16</sup> These two switches also appear on the MU Board, next to the eight LEDs. You can use either the switches on the Expansion Board, or those on the MU Board, as the switches form a "wired-OR" circuit.

```

.set    sw_S2,  0b10000000    ; Bit 7 = switch S2
.set    sw_S3,  0b01000000    ; Bit 6 = switch S3

ldr     r2,=iobase            ; Use R2 as a base address pointer

ldrb   r0,[r2,#portB]        ; Read the current value of Port B
tst    r0,#sw_S2             ; Is S2 being pressed?
bne    handle_S2             ; Yes, jump to handle_S2
tst    r0,#sw_S3             ; Is S3 being pressed?
bne    handle_S3             ; Yes, jump to handle_S3

```

Switches S2 and S3 can also generate interrupt requests in the ARM microcontroller: pressing one of these two switches sets the corresponding bit in the IRQ Status port in the Microcontroller I/O space: bit 7 is set for switch S2, bit 6 for S3. It is up to your programs to clear these bits, usually at the end of their interrupt service routines. Please see the description of the IRQ Status and IRQ Enable ports on page 10 for more information.

In addition to being connected to the ARM microcontroller, both switches are connected to the Field Programmable Gate Arrays. Switch S2 is connected to pad P36 on the Xilinx Spartan-XL FPGA, and switch S3 is connected to pad P123 on the Xilinx Virtex-E FPGA. When sensed by the FPGAs, both switches are active low: a Low voltage (GND, that is, 0V) indicates the switch is being pressed, a High voltage (VCC, ie, 3.3V) indicates it is not. Both S2 and S3 are debounced on the expansion board.

## DIP Switches S4

Switch S4 on the Expansion Board provides eight uncommitted on/off switches in a single package. Each of the eight switches is connected to the corresponding screw terminal connector on the Expansion Board. You can use these switches in your own circuits, or connect them to the Xilinx Spartan-XL connectors SA, SB and VS.

Each switch is *active low*: turning the switch on gives a Low output (ie, sets the output to be GND, that is, 0V), turning it off gives a High output (ie, sets the output to VCC, in other words, 3.3V). This is somewhat counter-intuitive, so please be careful!

None of eight switches comprising S4 are debounced. In fact, these switches seem particularly susceptible to contact bounce, as you will see if you connect an appropriately-configured oscilloscope!

## Debounced Toggle Switches S5 and S6

Two active-high toggle switches, S5 and S6 on the Expansion Board, are provided for your experiments. In either case, pushing the switch's actuator towards the top of the board (towards the screw terminals) turns the switch on (ie, sets the output High, to VCC). Pushing the actuator towards the bottom of the board turns the switch off.

Both S5 and S6 are debounced using standard set-reset flip-flops made from NAND gates. Please consult the Expansion Board schematic diagram for more details; you can find this document in the *board/schem* directory on your CD-ROM.

## Debounced Switches S7 and S8

Two debounced push-button switches, S7 and S8, are provided as uncommitted peripherals on the Expansion Board. These switches are active high: pressing a switch will turn it on and set the corresponding output to VCC.

Both S7 and S8 are debounced using resistor-capacitor networks passed through Schmitt trigger inverters; the debounce delay is about 70 ms. Please see the Expansion Board circuit diagram for more details.

## Undebounced Switches S9 and S10

The two push-buttons S9 and S10 on the Expansion Board are active low and not debounced: pressing one of these switches sets the corresponding output (in the screw terminal connector) to GND; releasing the switch sets the output to VCC.

**Please note that neither S9 nor S10 is debounced.** In other words, pressing one of these switches or releasing it can cause the physical switch to make or break contact multiple times before settling to the correct value. This “bouncing” gives a waveform similar to that shown in **Error! Reference source not found.**<sup>17</sup>



**Figure 10: Typical Switch Contact Bounce**

## LED Bar-graph Display

The bar-graph display D1 on the Expansion Board provides ten LEDs that you can use as on/off indicators. Connecting a screw terminal input to GND (or leaving that input unconnected) will turn the corresponding LED off; connecting the input to VCC (ie, 3.3V) will turn that LED on.<sup>18</sup> The left-most screw terminal input corresponds to the top-most LED in the bar-graph display.

## Expansion Board Connectors

The Expansion Board should be quite easy to use, as all connectors and peripherals are clearly marked on the printed circuit board itself. Thus, the following description is for your reference, for the times that you do not have the physical board in front of you. You should also consult the circuit diagram for the Expansion Board; this diagram can be found on your CD-ROM in the *board/schem* directory.

The **screw terminal connectors VS, SA and SB** allow you to connect your own circuits and/or peripherals to the Xilinx Spartan-XL FPGA signals of the same names. Thus, these connectors are attached to signals VS0-VS15, SA0-SA15 and SB0-SB15 respectively. Each screw terminal connector has a power connector next to it, allowing you to access GND and VCC (3.3V).

The **LED Array screw terminal connector** has already been discussed: this provides access to the ten LEDs in component D1.

The **Switches screw terminal connector** provides, from left to right, access to switches S5-S10 and S4 (1-8).

Two **power screw terminal connectors** are provided in the top left-hand corner of the Expansion Board. The upper connector provides access to VCC (3.3V) and GND (0V); your

---

<sup>17</sup> **Error! Reference source not found.** shows contact bounce for a generic *active-high* switch. Please remember that switches S9 and S10 are active low! In addition, contact bounce is usually only present on *either* the rising edge *or* the falling edge of the signal. Exactly what form of contact bounce occurs depends on how the switch is physically made.

<sup>18</sup> Although the LED input circuitry is designed to tolerate 5V TTL-level voltages, you should *not* connect the inputs to anything greater than 3.3V. As a rule of thumb, do *not* use the 5V or Unreg Power screw terminals in your circuits!

experiments will almost always use these two voltages. The lower connector provides access to the unregulated input voltage (typically 7-9V) and to 5V for analogue circuits. Unless specifically asked, **you should almost never use these voltages in your own circuits.**<sup>19</sup>

Please remember that **you should always turn the power off when wiring up any of your own circuits!** At the very least, you must connect the ground (GND) wire first and leave the power (VCC) connection to last. And please *do* double-check everything *before* connecting the power!

## Feedback and Credits

This document was written by John Zaitseff from the School of Electrical Engineering, University of New South Wales, with assistance from Jim Garside at the University of Manchester, and Saeid Nooshabadi, also from the School of Electrical Engineering. Please report any problems or comments that you may have regarding this document to your Laboratory Demonstrator and/or lecturer.

---

<sup>19</sup> Although the Spartan-XL inputs and the LED Array inputs should tolerate 5V TTL-level voltages, they will *not* handle the unregulated voltage. Connecting the Unreg screw terminal output to any pin will cost you a rather large sum of money...