

An Introduction to Komodo

The Komodo debugger and simulator is the low-level debugger used in the Digital Systems Laboratory. Like all debuggers, Komodo allows you to run your programs under controlled conditions. Doing this lets you see exactly what is going on in your program, helping you to remove any problems (“bugs”) that might be present.

The *examples* directory and its subdirectories on your CD-ROM contain many examples of assembly language programs that you can use to build up your debugging skills. And you are encouraged to do so, as part of your laboratory preparation! Please see the end of this document for details.

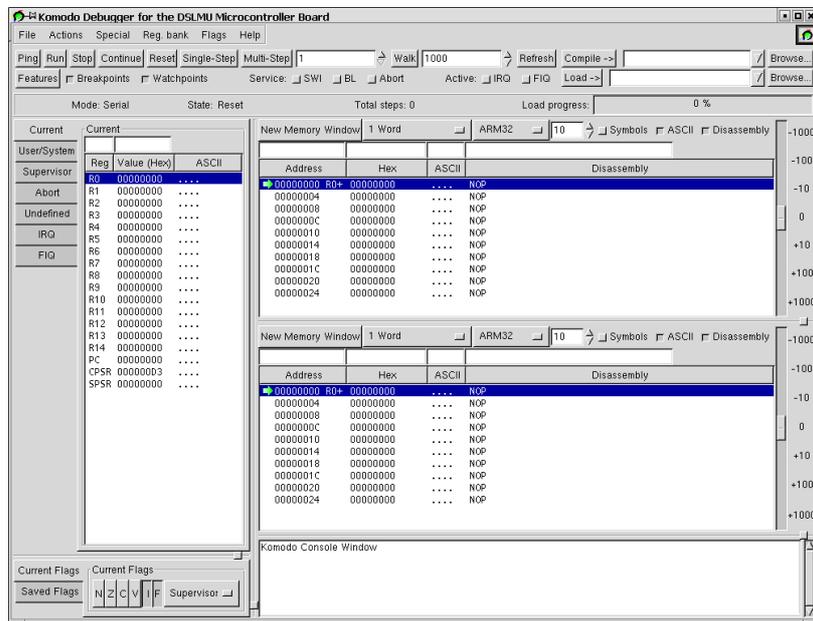
Invoking the Debugger

The Komodo debugger can operate in one of two modes: in *hardware* mode and in *emulation* mode. In hardware mode, Komodo communicates with the DSLMU Microcontroller Board via a serial cable. In emulation mode, Komodo runs some software that pretends to be a DSLMU Microcontroller Board—the software emulates the ARM microcontroller and some of the hardware that is present on the real board.

If you have a DSLMU Microcontroller Board in front of you (in other words, if you are in the Laboratory), you can start Komodo in its hardware mode by entering the following command line in the Unix shell:

kmd &

The following window should appear:



If it does not, check that the DSLMU Microcontroller Board is turned on and is ready, that the serial cable is firmly plugged in, and try again. If it still does not work, try replacing **kmd** with **kmd -v** to get some low-level debugging messages that might help diagnose the problem.

If you do not have a DSLMU Microcontroller Board in front of you, or don't need to use it, you can start Komodo in its emulation mode. To do this, enter the following Unix shell command line:

kmd -e &

Downloading Your Program

Whether you use the Komodo hardware mode or emulation mode, you need to download your ARM executable before you can begin the actual task of debugging it.

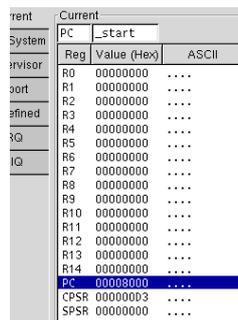
To download your executable program, click on the *second* **Browse** button in the top right-hand corner (ie, the one associated with the **Load** button, *not* the **Compile** button). A dialog box will appear that allows you to choose your executable; do so, then click **OK**. Back in the main window,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

Preparing to Run Your Program

Once you have downloaded your program to the DSLMU Microcontroller Board or to the emulator, you can prepare Komodo to run your program. There are four main steps you need to follow. These steps are also used to restart a program (ie, start running it from scratch).

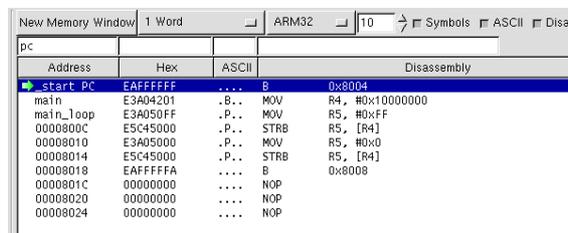
The first step is to reset the ARM microcontroller. To do this, press the **Reset** button. This initialises the PC and CPSR registers on the microcontroller.

The second step is to set the Program Counter register (PC) to the correct address. To do this, click on the PC register on the left-hand side of the window. The register name (PC) and value are copied to the edit fields above the registers. Click the left mouse button in the edit field, delete the value (using the BACKSPACE and arrow keys), type in the label **_start** and press ENTER. The list of registers will be updated to reflect the new value:



| System | Current | |
|--------|-------------|-------|
| PC | _start | |
| Reg | Value (Hex) | ASCII |
| R0 | 00000000 | ... |
| R1 | 00000000 | ... |
| R2 | 00000000 | ... |
| R3 | 00000000 | ... |
| R4 | 00000000 | ... |
| R5 | 00000000 | ... |
| R6 | 00000000 | ... |
| R7 | 00000000 | ... |
| R8 | 00000000 | ... |
| R9 | 00000000 | ... |
| R10 | 00000000 | ... |
| R11 | 00000000 | ... |
| R12 | 00000000 | ... |
| R13 | 00000000 | ... |
| R14 | 00000000 | ... |
| PC | 00008000 | ... |
| CPSR | 00000003 | ... |
| SPSR | 00000000 | ... |

The third thing to do is to get Komodo to display the portion of the program you will be running. To do this, click the left mouse button in the edit field above the **Address** label, delete anything already there, type in **pc** and press ENTER.¹ You should now see part of your program in a disassembled format; this part of the window is called the Memory panel, because it displays a portion of your program's memory:²



| Address | Hex | ASCII | Disassembly |
|-----------|----------|-------|---------------------|
| start_PC | EAFFFFF | ... | B 0x8004 |
| main | E3A04201 | .B.. | MOV R4, #0x10000000 |
| main_loop | E3A050FF | .P.. | MOV R5, #0xFF |
| 0000800C | E5C45000 | .P.. | STRB R5, [R4] |
| 00008010 | E3A05000 | .P.. | MOV R5, #0x0 |
| 00008014 | E5C45000 | .P.. | STRB R5, [R4] |
| 00008018 | EAFFFFFA | ... | B 0x8008 |
| 0000801C | 00000000 | | NOP |
| 00008020 | 00000000 | | NOP |
| 00008024 | 00000000 | | NOP |

Please note that it is the green arrow, *not* the blue highlight, that indicates the current point of execution in the Memory panel!

¹ If you like, you can enter something like **pc-8** instead: this automatically shows you the two instructions just *prior* to the location of the PC register.

² It is also called the Disassembly panel as it takes your binary executable file (*not* your source code) and disassembles it into ARM instructions.

The fourth thing you should do (although this is optional) is make sure that the **Symbols** checkbox is selected in the Memory panels, as shown in the previous screenshot. Komodo will then display any labels defined in your program instead of the address at that location.

The final (and optional) step that you might like to do is set a breakpoint at the label `exit`. Please see the section “Stopping Your Program” in this document for more information on how to do this.

Quitting the Debugger

To quit the debugger, simply select **File » Quit Program**. That is, click on the **File** menu and select **Quit Program**.

By the way, there is no need to quit the debugger because you have made some changes to your program. For example, imagine that you have found an error in the program being debugged. Simply modify the program source code in your editor window and save it, then type **make** in the Unix shell.³ You now need to download the new executable and prepare Komodo to run it, as explained previously.

Starting Your Program

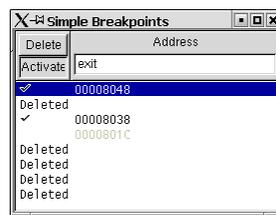
Once you have prepared Komodo to run your program, you can actually do so by pressing the **Run** button. The program will then start executing until it comes to a breakpoint.

Stopping Your Program

The whole purpose of a debugger is to control the execution of your program. In particular, Komodo allows you to stop your program at any point and then to proceed in a way most convenient to you. There are two main ways of stopping: the “abnormal” or emergency way, and the normal way.

The abnormal or emergency way to stop your program is by pressing the **Stop** button. This is especially useful if your program has “run away” from you!

The normal way to stop a program is by setting breakpoints in it. A *breakpoint*, as its name implies, breaks the program’s execution at that point. To do this, select the **Special » Simple Breakpoints** menu item, which will bring up a window similar to the following:



Select the breakpoint you wish to set from the list by clicking on it—this will highlight that line in blue. Now you can enter the address at which you want Komodo to stop. You can use labels defined in your program (as shown above) or hexadecimal numbers with or without the usual `0x` prefix.

You can use the same Simple Breakpoints window to delete breakpoints you no longer need or to temporarily disable breakpoints at which you do not wish to stop. Komodo also provides a more powerful Breakpoints window, in which you can specify a range of addresses in which Komodo will stop. You can also double-click on any line in the Memory panel to set or remove a simple breakpoint at that address.

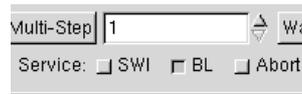
³ This assumes that you are using Makefiles to manage your project, which is highly recommended!

Stepping Through Your Program

Once your program has stopped at a breakpoint, you can let it continue running until it reaches another breakpoint. You do this by clicking on the **Continue** button or by pressing F10.

If you want to execute just the next assembly-language instruction in your program, click on the **Single-Step** button or press F7. If you want to step more than one instruction in one go, enter the number of instructions to be stepped into the edit field to the *right* of the **Multi-Step** button, then click on **Multi-Step** or press F8.

If you don't want to trace into functions (in other words, if you want to treat the whole subroutine or function call as a single assembly-language instruction), make sure that the **BL** checkbox is selected, as shown:



When this checkbox is selected, the **Single-Step** button will *not* jump into functions that are called with `b1`-type instructions. The **SWI** checkbox works in the same way.

Examining the Registers and Memory

A debugger would be pretty useless if all you could do was stop and start your program. Its power lies in the fact that you can *see* a program's state while it is stopped: in other words, you can actually see the program's registers and variables.

The Komodo debugger allows you to inspect the ARM microcontroller's registers at any time: these are all contained in the Registers panel on the left-hand side of the window. All register values are displayed in hexadecimal; if you want to see a register's value in decimal, pull out your calculator and work it out! Remember that, on the ARM microcontroller, register PC (the Program Counter) is also known as R15; register R14 is used as LR, the Link Register; and register R13 is almost always used as SP, the Stack Pointer.

You can examine a portion of your program's memory in the second Memory panel. Simply modify the edit field above the **Address** label and press ENTER; that section of memory will be displayed in hexadecimal. You can enter the address as a hexadecimal number (with or without the `0x` prefix), as a register name or as a label that you defined in your program.

Please note that there is a difference between displaying your program's variables and displaying a portion of memory: the difference is who does the interpretation. When you display a section of memory, *you* are responsible for interpreting that section correctly. Does the area of memory represent a string of bytes? Or a single 32-bit word? Or even an array of pointers? You decide!

To help you do this, Komodo allows you to display the area of memory in a variety of formats. Simply select a format from the list box above the **Address** field. The most useful formats are **1 Word** for 32-bit words, **1 Half-word** for 16-bit half-words and **1 Byte** for 8-bit bytes. Other formats can display more than one quantity on a single line.

Modifying the Registers and Memory

Komodo allows you to not only examine registers and memory, but to modify them as well. You do need to be careful, however: it is often very easy to crash your own programs when they are presented with unexpected values!

To modify one of the ARM microcontroller's registers, click on that register from the Register panel, change its value in the edit field above, then press ENTER. As usual, you can use hexadecimal numbers (with or without the `0x` prefix), register names, or labels that you have

defined in your program. Once you press ENTER, the register will be modified and the panel will be updated:

| Reg | Value (Hex) | ASCII |
|------|-------------|-------|
| R0 | 00008158 | X... |
| R1 | 000081A8 | |
| R2 | 00000014 | |
| R3 | 00000000 | |
| R4 | 00000000 | |
| R5 | 00000000 | |
| R6 | 00000000 | |
| R7 | 00000000 | |
| R8 | 00000000 | |
| R9 | 00000000 | |
| R10 | 00000000 | |
| R11 | 00000000 | |
| R12 | 00000000 | |
| R13 | 000085F8 | |
| R14 | 00000000 | |
| PC | 00008010 | |
| CPSR | 00000003 | |
| SPSR | 00000000 | |

Modifying the contents of memory is slightly trickier. First, enter the appropriate address in the **Address** edit field and press ENTER. That address will now be displayed.⁴ Next, use the TAB key to move to the **Hex** field (next to the **Address** field), then use the BACKSPACE and arrow keys as appropriate to delete any value there. Finally, you can enter the new value as a hexadecimal number, with or without the 0x prefix:

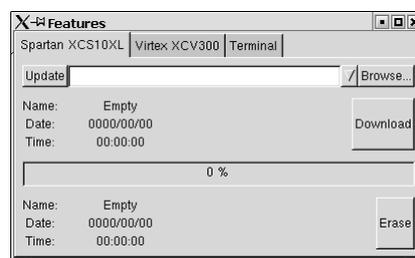
| Address | Hex | ASCII |
|----------|----------|-------|
| 00008800 | 00000000 | |
| 00008804 | 00000000 | |
| 00008808 | 2E544E4A | JNZ. |
| 0000880C | 00000000 | |
| 00008810 | 00000000 | |
| 00008814 | 00000000 | |
| 00008818 | 00000000 | |

Once you have entered the new value, press ENTER. The section of memory will be modified and the address will be automatically incremented to the next location.

Downloading an FPGA Configuration

In addition to running and debugging your programs, you can use the Komodo debugger to download an FPGA configuration into the Field Programmable Gate Arrays (FPGAs) on the DSLMU Microcontroller Board. Two FPGAs are available on this board: the Xilinx Spartan-XL XCS10XL and the Xilinx Virtex-E XCV300E.

To download a configuration into the Xilinx Spartan-XL FPGA, click on the **Features** button, then select the **Spartan XCS10XL** tab. You should see the following:



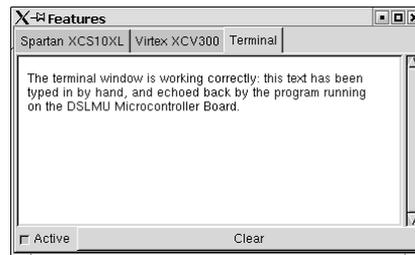
You can now click on the **Browse** button to select your FPGA program; the relevant file will have a *.bit* extension, as generated by the Xilinx FPGA Tools software. To download to the Xilinx Virtex-E FPGA, select the **Virtex-E XCV300E** tab instead. This feature is only available when connected to the actual hardware, not to the emulator.

⁴ Please note that the blue highlight might *not* reflect this new address. In general, it is best to ignore the blue highlight in Memory panels; the blue highlight *usually* reflects the last line that you selected with the mouse button.

Programming the FPGAs on the DSLMU Microcontroller Board is beyond the scope of this document. Please see the *DSLMU Microcontroller Board Hardware Reference Manual* for more information; you can find this manual in the *board/doc* directory on your CD-ROM.

Accessing the Komodo Terminal

One of the features of the software running on the DSLMU Microcontroller Board⁵ is that it allows you to write programs that communicate with the Komodo debugger. This feature is called the *terminal*, and can be accessed by clicking on the **Features** button, then selecting the **Terminal** tab. You will need to make sure that the **Active** is enabled, as shown below:⁶



Please see the description of the Serial Rx/D, Tx/D and Status ports in the *DSLMU Microcontroller Board Hardware Reference Manual* to see how you can use the terminal from within your own programs.

Example Files

As already mentioned, the *examples* directory and its subdirectories on your CD-ROM contain many examples of assembly language programs. You can use these example files to practise the debugging steps discussed in this document. And you are encouraged to do so, as “practice makes perfect”!

In particular, the *examples/intro* directory contains the following example files (amongst others); run the debugger on each and follow the suggestions given:

- | | |
|---------------------|---|
| <i>pseudo.elf</i> | Compare the source code (in <i>pseudo.s</i>) and the disassembled code in each of the subroutines. You can enter labels like <i>sub1</i> into the Memory panel to display that portion of your program. |
| <i>subr.elf</i> | Try stepping through this program with and without the BL checkbox being selected. Note how the debugger allows you to either trace into a function or to execute it as if it were a single instruction. |
| <i>jumptbl.elf</i> | Stepping through this program will help you understand jump tables. |
| <i>wordcopy.elf</i> | Trace through your program, noting how the content of the <i>src</i> array is copied to <i>dst</i> . Try modifying elements of <i>src</i> and rerunning the program. |

Happy debugging!

⁵ The on-board software is called the Komodo ARM Environment. You should be careful not to mix up the Komodo debugger (also called the front-end software) with the Komodo ARM Environment (also called the back-end or systems-level software). Although both work closely together, each performs a very different task. Please see the *DSLMU Microcontroller Board Hardware Reference Manual* on your CD-ROM for more information.

⁶ Ignore the text in the window in this screen-shot: this was transmitted by an example program running on the DSLMU Microcontroller Board.