3. HARDWARE-SOFTWARE DEBUGGER INTERFACE

3.1. INTRODUCTION

Students use the VHDL model of the 8051 microcontroller to co-simulate hardware and software in the Mentor Graphics' QSPro environment. Understanding what is going on inside the 8051 model during debugging is difficult and can made easier with tools developed using Tcl/Tk procedures built into QSPro.

This chapter discusses a hardware-software debugger GUI that displays the internal signals of the 8051 model and provides students the facility to step through their 8051 program one instruction cycle or clock cycle at a time while simultaneously simulating their hardware. Breakpoints can be set in the user's machine code to analyze intermediate results. The debugger interface also allows internal and external memory of the 8051 model to be viewed easily.

3.2. GUI DESIGN CONSIDERATIONS

Several factors were taken into consideration when designing the GUI debugger. The most important factor was that the debugger should be tied directly to the VHDL model of the 8051 microcontroller. This was needed to ensure that the debuggers' output reflects the actual operation of the 8051 microcontroller within the hardware simulation. Another important issue was that the development of the debugger should not affect the working of the 8051 VHDL model. Any changes made to the VHDL model to ensure compatibility with the debugger should not change the actual working of the 8051 microcontroller. Another factor that was important to take into consideration from a

product point of view was the portability of the software. A debugger that worked on UMRs' software environment should be able to work on any VHDL-compatible platforms to be valuable to the academic world in general.

The Tool Command Language (Tcl) along with Tool Kit (Tk) is the best choice for designing the GUI debugger [19]. Tcl, pronounced 'tickle,' is a high-level scripting language unlike C or C++ which can save considerable development time, as less code is needed for creating applications like GUIs, network simulator, text editors, real-time applications, and cross-platform applications, especially when interacting with existing software. Tcl is most useful when its rather small command set is extended. Tk, pronounced 'tee-kay,' is one such extension. Tk extends the basic Tcl language with commands to create and manipulate graphical user interfaces. In other words, Tk is an interface tool kit for Tcl. Tcl/Tk is platform independent and runs on flavors of Linux, UNIX, Windows and Macintosh systems using a Tcl/Tk interpreter. Tcl/Tk is an application language that can be embedded into existing applications and is very popular among hardware engineers.

Modelsim, the VHDL simulator used at UMR, comes with a built-in Tcl/Tk interpreter. Using the Modelsim 'when' command, VHDL signals can be copied into Tcl variables. Thus a Tcl/Tk program running inside Modelsim has access to all signals inside the VHDL model of the 8051 microcontroller. Whenever a signal changes value, the corresponding Tcl variable is automatically updated. A Tcl/Tk program can be written in an entirely event-driven manner, where a procedure is automatically run every time a Tcl variable changes. This is achieved with a command called "trace".

3.3. DESCRIPTION OF THE GUI DEBUGGER

The interface is comprised of one main window and many sub windows that can be invoked from the menu buttons on the menu bar. The main window displays the contents of frequently referenced registers like r0-r7, the accumulator, program counter, stack pointer, and data pointer. It also shows the current state of the machine cycle, the currently executing instruction, and others. The assembly-level program and associated machine code is also listed on the main interface. The menu includes menu buttons to view information associated with timers, I/O ports, serial ports, interrupts and memory. The user can click on the menu buttons to invoke sub windows that display the information. The buttons on the bottom of the interface incorporate processes like 'next instruction,' 'next clock,' 'go to breakpoint,' 'stop,' 'restart' and 'quit'. The main interface is shown in Figure 3.1 on page 15.

A detailed description of the individual components of the GUI debugger is given in the following sub sections.

3.3.1. Timers: Information about the 8051 timers, timer 0 and timer 1, can be obtained from windows brought up from the interface menu bar. The sub window, shown in Figure 3.2 on page 16, displays the contents of TCON (timer control register), TMOD (timer mode register), timer low-byte, and timer high-byte in hexadecimal format. The mode of operation of the timer, the timer overflow flag, timer run-control bit and GATE bit are also displayed. The value of these variables is automatically updated upon any change.

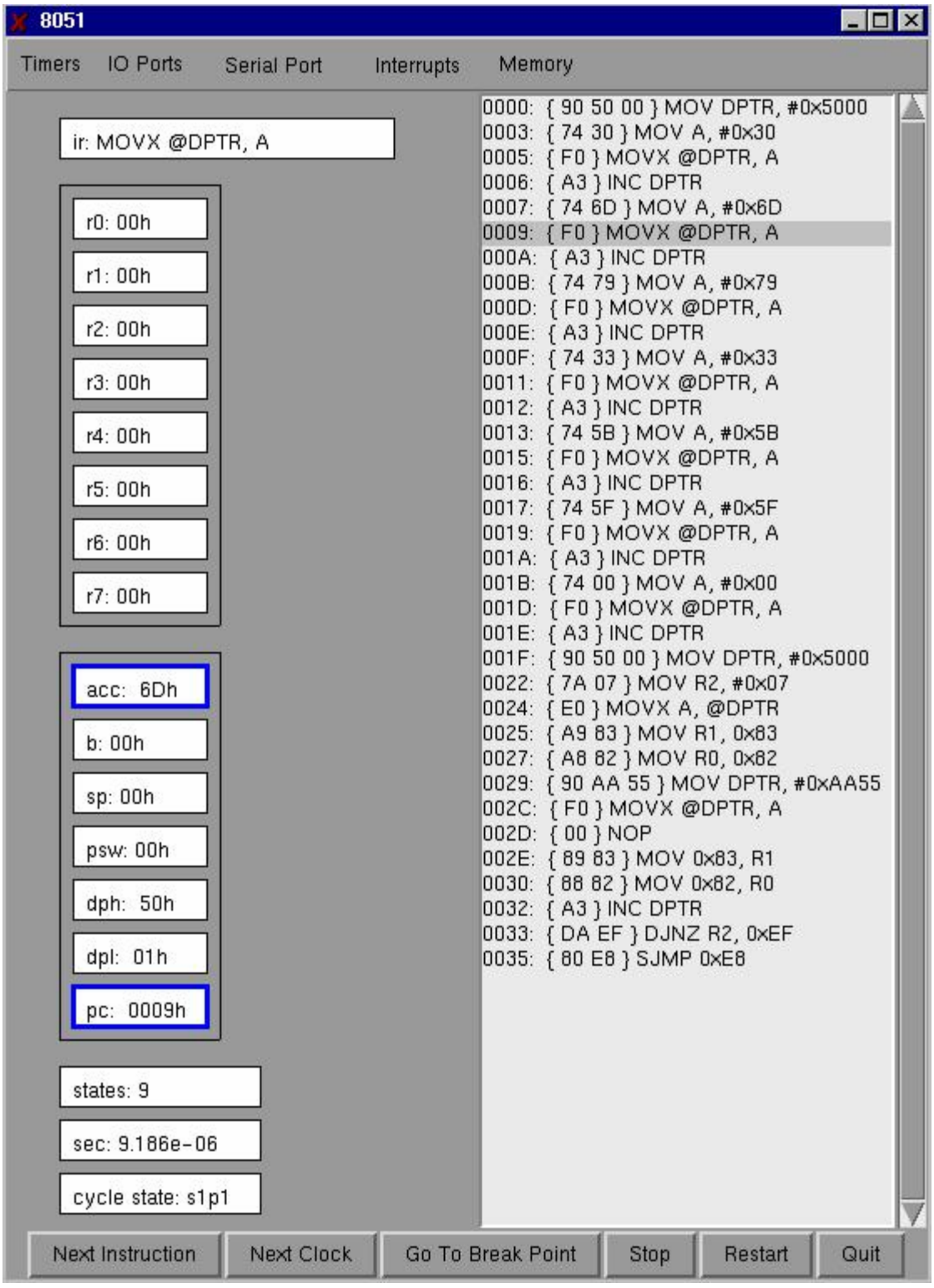

Figure 3.1. The 8051 Hardware-Software Debugger Interface

3.3.2. I/O Ports: The values of pins connected to I/O ports of the 8051 microcontroller can be displayed with the 'IO ports' menu button. A new sub window is created for each of the four ports. The windows display the 8-bit port value in binary format. Figure 3.3 shows the contents of port 0. Port values get updated automatically upon any change.

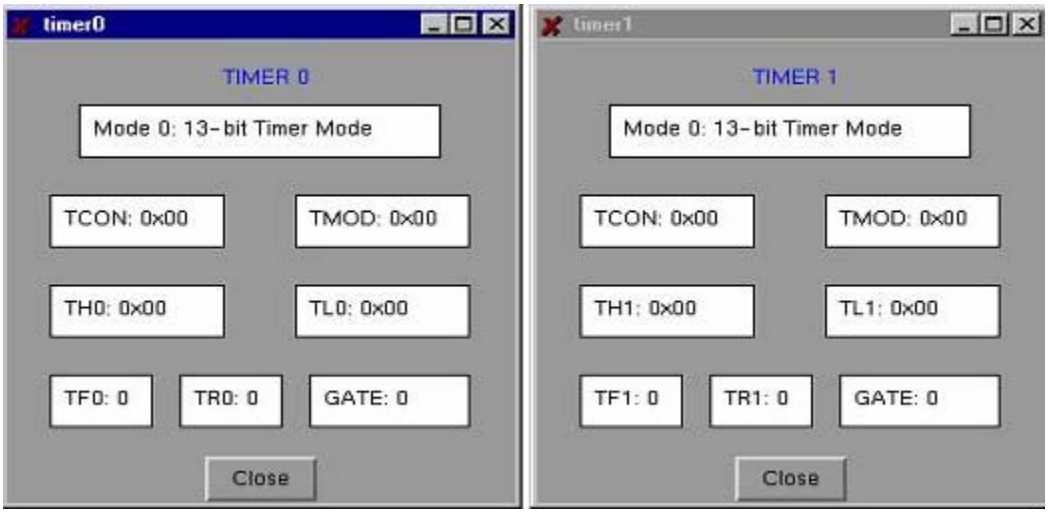

Figure 3.2. Interface windows showing the current state of the Timers

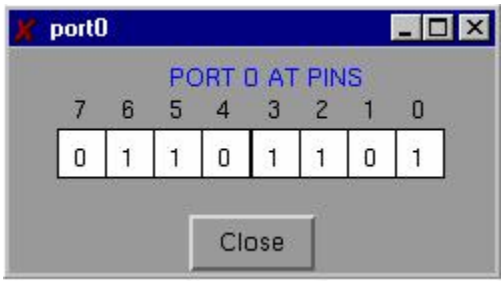

Figure 3.3 Interface window showing contents of Port 0

3.3.3. Serial Port: The 'Serial Port' menu button can be used to display information about the 8051 microcontroller serial port. Figure 3.4 shows the serial port sub window. The values of SCON (serial port control register) and SBUF (serial port buffer) are displayed in hexadecimal format. The mode of operation of the serial port is also shown depending on the SCON register. The baud rate is calculated and displayed based on the mode of the serial port. The baud rate calculation assumes a 12 MHz clock. In mode 0 and mode 2, the baud rate is fixed. For mode 1 and mode 3, baud rate is calculated using the formula shown below.

$$baud\_rate = \frac{2^{SMOD}}{32} \times \frac{clock\_frequency}{12 \times (256 - timer\_reload\_value)}$$

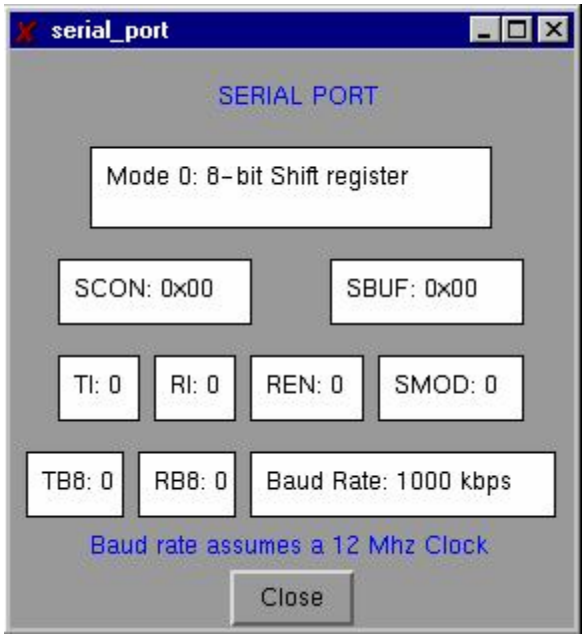

Figure 3.4. Interface window showing information about the Serial Port

The values of TI (transmit interrupt flag), RI (receive interrupt flag), REN (receive enable), SMOD (serial port mode bit), TB8 (transmit bit 8) and RB8 (receive bit 8) are also displayed. Any change in each of these variables is automatically updated.

3.3.4. Interrupts: The user can use the 'Interrupts' menu button to obtain information about the 8051 interrupts. The 8-bits of IE (interrupt enable register) and IP (interrupt parity register) are displayed as shown in Figure 3.5. Each bit of the registers is labeled for readability. The contents of these variables are automatically updated upon any change.

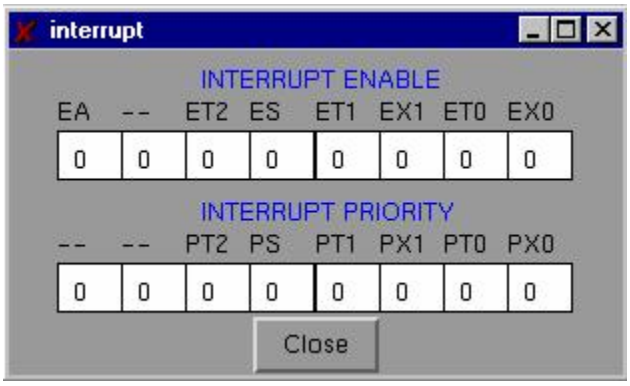

Figure 3.5. Interface window showing information about Interrupts

3.3.5. Memory: The VHDL model of the 8051 microcontroller has internal code and data memory and external memory. Contents of memory can be easily accessed from the 'Memory' menu button. The user can select any one of these memories from the drop-down menu and view its contents by specifying a range of memory locations in integer format. A detailed description of how to access each of these memories follows.

3.3.5.1. Internal code memory: The interface for accessing the internal code memory of the microcontroller is shown in Figure 3.6. The user can specify a range of memory to view its contents.

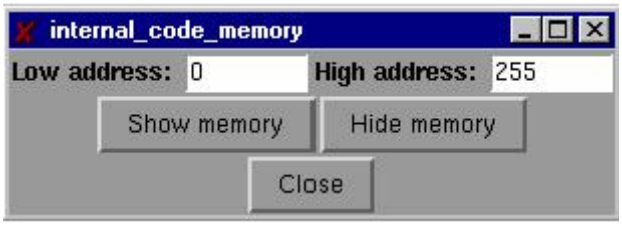

Figure 3.6. Interface window for Internal Code Memory

The 'Show Memory' button brings up the memory window as shown in Figure 3.7. The 'Hide Memory' button hides the memory window.

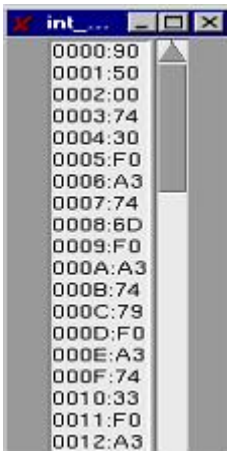

Figure 3.7. Interface window showing part of Internal Code Memory

3.3.5.2. Internal data memory: The internal data memory space of the 8051 contains 256 bytes of on on-chip RAM and an additional 128 bytes for special function registers (SFR). The SFRs are only accessible through direct addressing whereas the upper 128 bytes of RAM are accessible only through indirect addressing. The interface for accessing internal data memory, shown in Figure 3.8, provides user the option of selecting direct or indirectly accessible memories. The user can also specify a range of memory addresses.

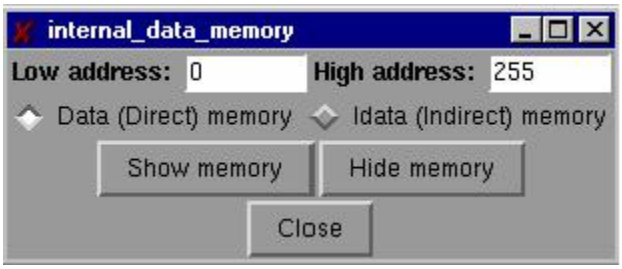

Figure 3.8. Interface window for Internal Data Memory

The memory spaces from 00H to 1FH (the register banks) and all SFRs are labeled for readability. Figure 3.9 and Figure 3.10 show the windows showing contents of internal data memory. The memory windows do not update immediately. An 'Update' button has been provided to refresh the window's contents whenever necessary. The window is not continuously updating to reduce simulation time. The interface also includes a 'Close' button, which can be used to close the window. When activated, the 'Close' button destroys the memory window and resets an internal flag to track the destruction of the window.

3.3.5.3. External memory: The external memory of the VHDL 8051 microcontroller can be accessed from the interface shown in Figure 3.11.

Figure 3.9. Direct Addressable Memory

Figure 3.10. Indirect Addressable Memory

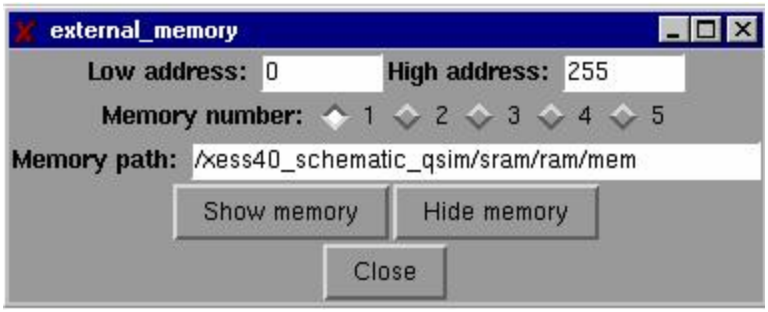

Figure 3.11. Interface window for External Memory

Most simulations of the 8051 microcontroller used in the lab include external memory. External memory is represented using a VHDL module. The user can add instances of this module to create external code and data memories. In the interface for accessing external memory, the user can provide a path to these memory modules and a range to view its contents. The path is required because the GUI may not know how many memory elements are instantiated at any one time or exactly how they have been instantiated. The user can view up to five different external memories at a time by providing the appropriate path in the interface or can view five different location ranges of the same memory. Since the memory windows do not refresh their contents automatically, an 'Update' button is provided. Figure 3.12 shows the external memory window.

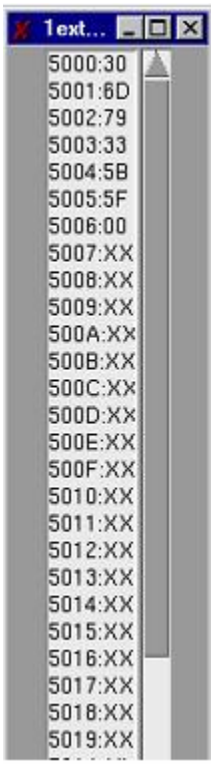

Figure 3.12. Interface window showing part of External Memory

3.3.6. Main Interface of the Debugger: The main interface of the debugger GUI displays the contents of registers frequently used in debugging like r0-r7 of bank 0, the accumulator, B register, stack pointer, program status word, low and high data pointer, and program counter in hexadecimal format. Whenever any of these registers changes it value, the box containing the register value is highlighted in blue to reflect the change.

The main window also displays the assembly-level program being executed by the 8051 microcontroller along with the machine code in the code-list box. The assembly-level program is re-constructed from the hex file provided to the VHDL 8051 microcontroller. Each instruction is preceded by its address in memory. The instruction to be executed next is highlighted in the code-list box. Users can set breakpoint in the program by selecting any instruction and hitting 'Go To Breakpoint' button to execute till that point and analyze intermediate results.

The interface also displays the total time in seconds and total number of machine cycles executed since the start of execution. It also shows the current state (s1p1, s1p2 …) of the machine cycle being executed. The next instruction to be executed is displayed in instruction register box based on the value of program counter. The main interface is shown in the Figure 3.1 on page 15.

3.3.7. Buttons: The buttons: 'Next Instruction,' 'Next Clock,' 'Go To Breakpoint,' 'Stop,' 'Restart' and 'Quit' are provided at the bottom of the interface. These buttons call Tcl procedures to perform each task. The user can step one instruction at a time with the 'Next Instruction' button. The 'Next Clock' button can be used to step through the program one clock cycle at a time. The user can also select a breakpoint in the code-list box by highlighting an instruction and can execute till that point with the

'Go To Breakpoint' button. The 'Stop' button can be used to stop execution at any point. The 'Restart' button restarts the simulation from the beginning by issuing a 'restart' command to ModelSim; it also resets contents of memory windows, internal variables and flags used in the Tcl/Tk code. The 'Quit' button is used to exit the ModelSim environment.

3.4. TESTING OF THE INTERFACE

The hardware-software interface debugger was first tested by invoking it on different assembly-level programs. Test programs were written to change the values of internal registers like TH0, SCON, IE, TMOD, SBUF and values of memory in the VHDL model. For brevity, these programs are not included in the appendix. The test programs were simulated instruction-by-instruction or clock-by-clock or by setting breakpoints using the buttons provided. The interface was checked for its correct operation by visually looking at the values of the registers and memory and comparing it with the signal values in the VHDL model. The interface reflected the true operation of the micro-controller. Changes in VHDL signals were correctly reflected on the interface.

In addition to functional testing, field-testing of the interface was done with approximately 40 undergraduate students in the 2002 Fall Semester Computer Engineering 214 class at UMR. A lab exercise was modified to use the hardware-software debugger interface and a survey was conducted to evaluate its effectiveness in the laboratory.

In one of the CpE214 laboratories students write an assembly-level program to partition the 8051's external memory into data and code segments. The data segment

holds a message that is displayed on a seven-segment display which is mapped to some external memory location. The program is responsible for initializing the message table created in the data segment, for retrieving a character from the table, and then writing the character to the seven-segment display with some short delay between consecutive characters. Before the 8051 hardware debugging GUI, students debugged the software portion of their hardware-software design using Keil μVision 2. Their hardware and software were debugged together in QSPro, but they had few tools to control the simulation and observe what was happening in the 8051. With the use of the hardware-software debugger interface, students were able to carefully debug their software portion along with their hardware (the 8031 microcontroller, the seven-segment display, external memory). They used the interface to view the internal signals of the 8051 and particular locations in memory (data segment and code segment). By simulating their program instruction-by-instruction or clock-by-clock or by setting breakpoint they had more control over the debug process.

A special-purpose Tcl/Tk interface for the seven-segment VHDL model was developed specifically for the laboratory so students could view what was being written to the seven-segment display. The interface is shown in Figure 3.13. Whenever a character is written to the seven-segment display it is highlighted in blue on the interface. Students do not have to trace the inputs to the seven-segment model to verify the function of their program.

In the second part of the experiment, students were asked to modify their program such that the data and code segments overlap. After modification students verified their design with Keil development tools. A disadvantage to simulating software without

hardware is that the software simulator has no idea what the hardware configuration is and may give incorrect results. For most students, when the design was debugged with the hardware-software debugger interface in the QSPro environment, it did not work. The debugger interface made it much easier for students to figure out the problem.

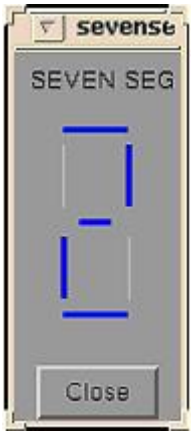

Figure 3.13. Interface window showing Seven-Segment display

Feedback from the two instructors who taught the laboratory course indicates that the design was debugged more smoothly using the interface than in previous semesters (when the interface was not available), due to the students' access to the internal registers and to internal and external memory of the 8051 microcontroller. Students were able to complete the laboratory quickly as the debugging time was reduced with the use of the interface. The laboratory includes a special problem that occurs when the students implement their design in hardware that did not occur when they simulated their design. Many more students were able to figure out the source of this problem without the instructor's help than in past semesters because of their better understanding and confidence in their design enabled by the debugger interface.

Preliminary survey results of students are encouraging. Survey results are presented in Table 3.1 with a brief summary of the questions asked and the response received. Each question had four multiple choice answers. The value in brackets represents the number of people selecting a particular choice. The survey itself is attached in Appendix A. Around nineteen people participated in the survey.

Table 3.1. Survey Statistics

Question	Response			
	A	**B**	**C**	**D**
Q.1. Helpfulness of GUI in debugging their design	Very helpful (10)	Helpful (5)	Somewhat helpful (3)	No need for a GUI (1)
Q.2. Helpfulness of GUI in debugging more complex designs	Very helpful (11)	Helpful(4)	Somewhat helpful (3)	No need for a GUI (1)
Q.3. Difficulty in debugging design without GUI	Very difficult (1)	Difficult (9)	Somewhat difficult (8)	No need for a GUI (1)
Q.4. Time spent debugging using GUI	<15 minutes (9)	15-30 minutes (6)	30-45 minutes (2)	1 hour (2)
Q.5. Helpfulness of GUI in aiding understanding of how hardware and software work together	Very helpful (3)	Helpful (9)	Somewhat helpful (5)	No need for a GUI (2)

The survey also included open-ended questions on how we might improve the GUI debugger by adding extra features and on troubles faced by the students while using the GUI debugger. One of the more common problems faced by students was the format in which they enter the memory range to be displayed in the memory interface window. Initially, the debugger was designed to takes the memory range in integer rather than hexadecimal format. Later, the code for the memory interfaces of the debugger was

modified to take the memory range in hexadecimal format. Hexadecimal format is often easier for students when handling memory addresses. This was achieved with the help of built-in functions included with the Tcl language.

Another bug identified with the debugger was a warning message which pops up after closing sub-windows without using the 'close' button provided. The message pops up because the Tcl/Tk script for the window has an active 'trace' command for the variables displayed in the sub-window. If the student closes the window using the 'close' button, the code for the GUI debugger takes care of this bug by deactivating the traces on the variables being displayed. Additional notes have been included in the procedure of the experiment to remind students to use the close button.

Apart from these minor bugs, the majority of students found the hardware-software debugger interface very helpful in debugging their design. The debugger interface also helped them to better understand how the 8051 worked and how hardware and software work together. Students felt the time spent debugging design was considerably reduced with the use of the debugger interface as they did not need to look as carefully at the traces to debug their design. The instructors teaching this lab noticed a significant improvement in students' performance and understanding of the lab compared to previous semesters.