Experiences with the UWTester in Computer Science and Engineering Education

Neil McKenzie, Carl Ebeling, Larry McMurchie, Gaetano Borriello
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
June 2, 1994

Abstract

Teaching students the complete process of circuit design, simulation, implementation, test and debug is a daunting task. Even though design description tools and circuit compilers have kept up with the increasing levels of integration found in current implementation media such as FPGAs and microcontrollers, it has become increasingly difficult for students to test and debug the complex hardware projects enabled by these modern tools and devices. In this paper we describe an inexpensive digital circuit testing environment that greatly simplifies the test and debug experience. This environment, consisting of the tester hardware and its corresponding software, enables students to experience the challenges of testing and debugging without the expense of costly commercial hardware testers or the drudgery of setting up *ad hoc* hardware test jigs. The UWTester has been in use in the Department of Computer Science and Engineering at the University of Washington for the past two years in both undergraduate and graduate hardware design courses as well as independent study and research projects.

1 Introduction

Today's digital logic implementation technologies emphasize very high levels of integration in various forms: custom VLSI chips, commodity microprocessors and microcontrollers, and field programmable logic such as FPLDs and FPGAs. Highly integrated circuits have many advantages, such as lower power, reduced circuit board area, greater functionality and lower production cost. However, these circuits require more time and money to be invested into sophisticated development tools. Introducing students to such sophisticated tools and techniques as early as possible ultimately benefits both the students and their future employers, but it places the burden on educators to have the appropriate teaching skills and adequately outfitted laboratories.

The process of circuit design involves several distinct steps. First, the designer *specifies* the circuit at the appropriate level of abstraction. A circuit can be specified in many different ways, including high-level hardware

description languages, schematics, and behavioral models. Hierarchy in the description is helpful for specifying a design in a top-down fashion. Encapsulating a part of the circuit makes it easy to replicate, much like a subroutine in software makes part of the program easy to invoke iteratively or recursively. Second, modern design tools *compile* the descriptions into a form suitable either for simulation or hardware implementation. Because simulation is less expensive than actual circuit construction, simulation is used to validate and debug a design, and ultimately reduces the amount of time needed to debug a hardware implementation. The third step is to *implement* a circuit prototype that can be tested and debugged. This final step is crucial in an educational environment. Building real hardware both motivates students and reinforces fundamental concepts.

In recent years, modern circuit design tools have been introduced to students in digital design laboratory courses. The availability of these tools has greatly increased the complexity of projects that students are able to undertake. Unfortunately, the tools available for testing and debugging have lagged behind the tools for specification, compilation, simulation and implementation. Simple projects can be constructed using breadboards, wires and TTL chips, and then tested using switches and LEDs. However, interesting projects employing FPLDs and FPGAs are often too complex to be tested and debugged in this way, due to the large amount of state they may require and the larger number of input and output signals compared with simple projects. Students can become overwhelmed getting such complex systems to work. The options available for testing and debugging complex projects are limited to commercial chip testers and *ad hoc* approaches.

Commercial chip testers can be used to perform both *functional testing* and *speed testing*. Functional testing means that we verify the I/O signals for *values* and *sequencing*, but not for precise *timing*. Although the speed testing capability of commercial testers is invaluable for testing custom chips, it is not needed for the kind of projects common to undergraduate labs. Commercial testers have two significant drawbacks. First, their cost is prohibitive for most small companies and educational laboratories. Second, commercial testers tend to focus on a model of testing known as *off-line*. Test vectors are downloaded to the tester from a host system, the test is executed, then the results are uploaded and analyzed. Usually, the test vectors are presented to the circuit prototype sequentially, with minimal or no support for conditional branching or iterative constructs. Off-line testing limits the user's interaction with the test device-under-test. Due to these drawbacks, commercial testers are seldom used in undergraduate labs.

The most common approaches to testing and debugging are *ad hoc*, in which a special-purpose environment is constructed around the project to be tested. This self-contained system can be operated interactively while being observed by a logic analyzer or oscilloscope. Constructing this environment may be expensive, difficult and time-consuming. Shortcuts are taken that result in a hard-wired and hard-to-modify test system. *Ad hoc* approaches severely limit the scope of projects that students can build.

We developed the UWTester system to fulfill the need for a convenient and inexpensive platform to test and debug complex circuit prototypes. The UWTester provides an alternative to both the expense of the commercial circuit testers and the inconvenience of the *ad hoc* approaches. Here is a list of its features:

- The software environment supports *on-line* functional testing, in contrast to the off-line testing performed by commercial testers. The host computer generates, downloads and verifies test vectors one-by-one at run-time. Computation and testing execute concurrently; to the host processor, the tester acts as a hardware subroutine. Tests employ the full power of conditional branching, subroutines and recursion. Test programs are described procedurally using a high-level programming language. Interactive programs, where the user supplies part of the data for the test at run-time, can be constructed easily.
- The software environment allows most of the mundane aspects of setting up a test to be encapsulated in a test program. For example, the programming of an FPGA can be encapsulated into a software routine that reads the FPGA program data from a file and sets the pins of the FPGA accordingly. Once this routine has been written, it can be reused whenever an FPGA needs to be programmed.
- Test set-ups can be created and broken down easily. This is important in educational laboratories because many student projects need to share the same equipment. Changing the set-up is simple and requires minimal wiring by hand. Because most of a test set-up is specified in software, its description can be edited and compiled on a separate computer from the one used to execute the test.
- The user interface is easy to learn, yet is sufficiently powerful and flexible to accommodate a wide variety of projects ranging in complexity from introductory digital design to graduate level projects. The time a student spends learning the system is amortized over multiple uses.
- The system integrates easily into the existing set of tools and equipment. Most digital logic laboratories use standard personal computer technology, either IBM-PC compatibles or Macintoshes. Because schematic capture and simulation tools are readily available on these computers, there is a strong association between the test and debug step and the specification and simulation steps. This close tie makes it possible to test a circuit prototype by using the simulation of a project that incorporates the prototype as a subcircuit.

In the following section we give an overview of the UWTester hardware architecture and implementation. Section 3 describes the high-level programming language interface. Section 4 describes a complementary interface to a schematic capture and simulation tool. Section 5 surveys uses of the UWTester in the curriculum: undergraduate education, functional chip testing, speed testing, and others. A survey of projects related to functional testing and hardware testbeds is provided in Section 6. Section 7 draws some conclusions and summarizes the current status of the project.

2 UWTester architecture and hardware implementation

The UWTester system consists of the tester unit, a Macintosh II or IBM PC compatible personal computer to which it is connected via a host interface card for either the Macintosh NuBus or the IBM PC ISA bus[1]. Figure 1 is a photograph of the UWTester unit as manufactured by Applied Precision, Inc. The lid of the tester unit has been removed to reveal the printed circuit board. The tester unit is small (about the size of a three-ring binder) and weighs only a couple of kilograms, allowing it to fit easily into a crowded undergraduate lab. The UWTester provides 128 bi-directional I/O pins to the test circuit, thereby accommodating most complex FPLDs and FPGAs. There is a 21 x 21 zero-insertion-force (ZIF) socket for testing chips. The ZIF socket permits insertion of familiar dual-inline-package (DIP) chips from 14 to 40 pins, and pin-grid-array (PGA) chips from 68 to 132 pins. Surrounding the ZIF socket is a ring of probe pins for each bi-directional signal, for connecting to external circuitry or other test equipment such as an oscilloscope, or for power and ground wires. A chip test is set up by simply connecting power and ground wires and plugging in the chip. The UWTester minimizes set-up time by mapping all the I/O signal pins in software, thereby eliminating the hand-wiring step often required in other circuit testers.

Figure 2 shows a high-level block diagram of the tester data path. Tri-state drivers control each of the 128 pins provided by the tester with two bits per driver: a data bit and a direction (enable) bit. Each test vector therefore consists of 256 bits: 128 data values and 128 enable values. Note that the enable value is specified separately for each pin and also for each test vector. This architecture allows maximum flexibility to test all groupings of bi-directional signals. During an on-line test, the host microprocessor downloads the 256 bit test vector over the host interface cable, onto the internal data bus and into the level 1 registers. Because the width of the host interface data bus is only 32 bits, it takes eight writes to issue a single test vector. All 256 bits are shifted at once from the level 1 to the level 2 registers, to strobe the entire vector onto the input pins of the test circuit. After a user-specified delay, the values of the test circuit's output pins are latched into the level 3 registers. These result values are subsequently gated onto the data bus and uploaded to the host. Off-line testing is similar, except that the data and enable values come from the test vector memory. A state machine in the tester's controller circuit sequences a block of vectors to the test circuit at a fixed rate and stores the results back into test vector memory.

For implementation, we used inexpensive FPGA technology in the form of Xilinx Logic Cell Array (LCA) chips. The data path is implemented in six Xilinx XC3020 LCAs and the controller circuit in a seventh. Each of the six data path LCAs drives 22 pins of the circuit under test and 11 bits of the internal data bus[2]. The test

[1] The UWTester hardware is described in greater detail in a set of technical reports from the University of Washington and the Proceedings of the Microelectronic System Education Conference [1-4].

[2] The internal data bus is 64 bits and the six Xilinx chips drive up to 66 bits, so there are two unused pins.

vector memory is implemented using eight 32K x 8 static RAM chips. For each test vector, each memory chip is read four times and written to twice. Hence there are 32K/6 (about 5400) test vectors available for off-line testing.

3 Programming language interface

We developed a package of low-level interface routines, implemented as a C run-time library, to support both on-line and off-line testing. Users write test programs in C and have access to all the usual C run-time support, such as text output, bit-mapped displays, and input from mice and keyboards. C compilers are widely available, inexpensive, and run on all popular processor platforms; porting the library to a new architecture is straightforward.

Figure 3 shows an example on-line test program for the 74LS163 TTL 4-bit counter chip. The tester run-time library interface is declared in the header file `tester.h` (line 7). The LS163 has a 16-pin dual in-line package footprint in the ZIF socket. The binding between the ZIF socket and this footprint is declared by the header file `DIP16.h` (line 8). The pins of the circuit under test are declared as variables in line 10. The routine `DefineSignals()` (lines 12-24) binds groups of test pins to the `Sigptr` variables. Calls to `Vdd()` and `GND()` bind groups of pins to the +5 volt power rail and ground respectively. Calls to `DefineSignal()` declare blocks of signals to be INPUT, OUTPUT or BIDIRECTIONAL. For example, line 18 assigns the four bit input bus `in` to pins 6 through 3; likewise, line 22 assigns the four bit output bus `out` to pins 11 through 14. The `load` signal is assigned using `DefineInvSignal(),` to signify that it is active-low instead of active-high. If the pins are INPUT to the chip, then an initial value can also be declared. In the subsequent execution phase, chip pin values are accessed by reading and writing these variables.

The entry point `MainTest()` begins the execution phase. There are three types of calls into the run-time library for the execution phase of the test:

- `Set()` changes the value of an INPUT signal.
- `Get()` reads the value of an OUTPUT signal.
- `Next()` sequences the tester.

Calls to `Set()` are buffered and take effect in parallel during the call to `Next()`. The values read back from `Get()` are with respect to the most recently executed `Next()` call.

In `MainTest()`, the integer variable `count` models the four-bit result from the LS163 chip. First, `count` is initialized to the value 10 (line 37), and likewise the chip has its input bus set to 10. Line 39 tells the tester

to activate the load pin which was declared active low (line 20). The subroutine `clockIt()` gives the clock pin a rising edge by setting it first low, then high; the result is that the value 10 is produced on the outputs of the chip. Line 41 verifies the result. Line 43 tells the chip to stop loading and start counting up. Lines 44 through 49 cause the chip to count up 21 times; the value of the result is verified every iteration on lines 47-48. Line 46 is necessary to model the four-bit count by masking off all but the low four bits of the result.

It is easy to modify an on-line test program and create a functionally equivalent off-line test that differs only in timing. Figure 4 shows the changes to `MainTest()` required to convert the LS163 example into an off-line test. The macros `BEGIN_OFFLINE` and `END_OFFLINE` delimit an off-line block of vectors. The statements within the off-line block are invoked using a two-pass strategy. The first pass, known as the *generate* pass, generates the block of input test vectors. The second pass, known as the *verify* pass, verifies the block of result vectors uploaded from tester memory. Between the two passes, the tester sequences the entire block of vectors off-line. There are a number of restrictions that apply to off-line testing. The size of the off-line block must fit in tester memory. Because statements within an off-line block are executed twice, the programmer must ensure that they use the same set of initial conditions and compute the same results during both passes. `Set` performs an action only during the generate phase, and `Get` returns a valid response only during the verify phase. The macros `GENERATE` and `VERIFY` delimit blocks of statements to run in their indicated phase only.

4 Schematic editor user interface

A programming language environment is one model for specifying the *behavior* of a test; alternately, a schematic can be used to specify the *structure* of a circuit as the test environment. A simulator can translate the schematic into a behavioral model of the test. A schematic may be better than a test program for describing tests for several reasons. Schematics use diagrams whereas programs use text; graphically oriented descriptions tend to be easier to understand. Schematics are already a familiar medium to most people who are designing and implementing hardware, whereas a programming language environment may be an unfamiliar abstraction. Moreover, most digital design curricula begin with schematics and only use hardware description languages in later courses. Schematics are also quite necessary if the goal is to build a stand-alone system to operate the test circuit. A schematic editor and simulator system make it easy to design a prototype of the environment for the test circuit and help debug the integration of the device into its environment as well.

We have integrated the UWTester with LogicWorks, a schematic editor and simulator package from Capilano Computing. The LogicWorks simulator interprets the schematic on-the-fly rather than using a separate compilation phase. The advantage is that incremental changes to the schematic affect the simulation immediately; as a result the environment is highly interactive. The disadvantage to interpreting the schematic is lower performance compared with using compiled code. The effective on-line testing rate is only a few tens or

hundreds of cycles per second. Nonetheless, LogicWorks has proven to be a highly effective, intuitive and widely-used tool for creating schematics as test environments for chips and subsystems. LogicWorks includes libraries of familiar building blocks such as clock drivers, logic gates, TTL chips and ALUs. For testing and debugging, there are simulated keypads, switches, logic probes and hex displays. Also, behavioral descriptions can be included inside LogicWorks, for instance in the form of a simulated FPLD or memory chip. The hardware-software boundary is defined by a symbol in the schematic that represents the *physical* circuit plugged into the UWTester. All other symbols in the schematic represent simulated components. The user declares the pins of the hardware-software boundary to be inputs, outputs or bi-directional signals. By default, the UWTester software driver expects the circuit under test to be a chip plugged into the ZIF socket. The number of pins in the boundary symbol tells the driver what chip footprint to use, from a set of pre-defined mappings corresponding to 16, 20, 24, 40, 68, 84 or 132 pin chips. These defaults can be overridden in the case of a non-standard pin mapping. In essence, specifying a different physical pin-out and/or package for the test chip requires little effort. Because the low-level software driver is the same for all chip footprints, changes are localized to the schematic view.

For LCA testing under LogicWorks, we created a separate software tool to initialize LCAs. First, the tool prompts the user to connect power and ground wires; it verifies that the wires are correctly placed. Next, the user inserts a LCA chip into the tester's ZIF socket. The tool asks the user for the name of the programming (raw bit) file generated by the Xilinx logic synthesis tools for downloading to the LCA. Once initialized, the LCA can now be used in a LogicWorks simulation session.

Figure 5 shows a LogicWorks schematic for testing a circuit implemented in an LCA. This example tests a four-bit counter circuit, analogous to the test program given in Figure 3. The UWTester ZIF socket contains a Xilinx 3020 LCA chip which has been preloaded with an LS163 circuit model. Across the top row of Figure 5 are a hex keypad (set to the value 6), a clock driver, a 74LS163 block, and a 4-bit adder block and a 4-input NOR gate at the upper right corner. In the lower right corner are a set of binary switches, a D flip-flop, and a set of binary outputs and hex outputs. The hardware-software boundary is defined by the circuit symbol labeled `lca3020`. We created the `lca3020` symbol using the LogicWorks device editor, and then attached the UWTester software driver to this symbol to specify its behavior. Because the symbol has 68 pins, the software driver maps the tester's ZIF socket to a 68-pin footprint. Except for the `lca3020` symbol, all symbols used in this example came from LogicWorks's standard libraries. The simulated LS163 circuit (upper left corner) computes the output values Q3, Q2, Q1 and Q0, and the physical LS163 circuit (in the LCA in the tester) computes the output values X3 through X0[3]. The circuitry on the right side of the diagram compares X3..X0 and Q3..Q0 by generating their difference and checking that the result is 0 by means of a NOR gate. The NOR

[3] LogicWorks connects signals by name automatically; for instance, all lines with the label `ld-` are connected together.

output is fed into a D register to eliminate transients (glitches). The binary display labeled `compare` shows the registered output. Thus, if the two 163 models compute the same output at the same time, `compare` becomes 1 (true). The two carry bits are displayed just above the two hex readouts in the lower right corner. When the combined simulation and test is first started, the two counters are counting out-of-phase as shown: the LCA value is 15 (hex F) and the simulated counter value is 9. The two counters can be brought into phase by either toggling the LD/INC switch to load both counters with the value from the hex keypad, or by toggling the CLEAR switch to clear both counters.

The combination of the UWTester and a schematic editor allows designs that comprise only part of a system to be implemented in hardware, while the remainder of the system is described by a simulation. The entire system – physical circuitry and simulated schematic – can then be operated as a single entity. LogicWorks and the UWTester together provide a framework for integrating all three levels of circuit description: behavior, structure (schematic) and physical circuitry.

5 Using the UWTester in the curriculum

The UWTester has been applied throughout our curriculum in both undergraduate and graduate courses. This section presents a survey of these varied uses. Novice users are introduced to the tester in a highly constrained environment – the fewer the options, the less opportunity there is to make mistakes. As users become more sophisticated, they are exposed to more features and must take on more responsibility for understanding the tester environment as a whole. We have gained experience with the tester at all levels of proficiency as described in the following summaries.

5.1 Introductory digital logic design

A typical laboratory exercise is to design a simple controller circuit, such as a dynamic RAM refresh controller or a traffic light controller [5]. The student describes the circuit using a state machine language. Students can verify their designs by several different means. One method is to compile the high level description into a form that can be simulated by a software logic simulator. Another method is to compile the high level description into a binary description used to program a small FPLD chip. To verify the circuit, the student places the programmed FPLD into the UWTester and runs a test program constructed by the professor or teaching assistant. The student sees a very simple and restricted interface to the UWTester. The advantage to this environment is that the student can focus on the task of defining and verifying the circuit, without worrying about the details of the tester interface.

5.2 Advanced digital logic design

Students in senior level courses in digital logic design are introduced to a more sophisticated interface to the tester. Students design a target circuit and also create the environment for testing the target circuit. One assignment from a CSE senior laboratory course is to design a Morse code decoder, implement the circuit with a Xilinx LCA, and use the UWTester and LogicWorks as the test environment. The input to the Morse code circuit is a key switch that creates a stream of dots and dashes. The student-defined circuit implemented in the LCA translates the input stream of key clicks into a sequence of ASCII character codes. The student specifies the circuit using the high level description language ABEL and compiles the design through ABEL's logic synthesis tools and the Xilinx place and route tool suite. This process converts the high-level description into an LCA programming file. To test the design, the student downloads the program bits to the LCA in the tester's socket and then starts up a LogicWorks session. When the LogicWorks simulator is activated, the proper inputs are fed into the LCA and the outputs generated by the LCA are displayed. ASCII characters are then displayed using simulated character display modules inside of LogicWorks. The UWTester also allows external circuitry to be connected, such as physical switches, LEDs, relays, keyboards and liquid crystal display (LCD) panels. Thus, the Morse code circuit from the senior lab course can be made into a stand-alone system, by using real keys and LCD instead of their LogicWorks simulation models.

Students also use the UWTester in their senior term projects. Two of these term projects illustrate the application of the tester to projects involving embedded FPGA devices. The first project is a small autonomous robot constructed from Lego bricks, motors and wheels. An LCA in the robot's controller circuit receives its programming file over a standard 4-wire RJ-11 telephone cable. One end of the phone cable is plugged into the robot, and the other end is plugged into the UWTester. After the robot's LCA is initialized, the phone cable can be disconnected to allow the robot to roam untethered. The phone cable solution permits easy integration of FPGAs into mobile systems. The second project is a small low-cost hand-held logic analyzer. Students in introductory digital design courses find sophisticated logic analyzers difficult to use and gain little leverage from them for their relatively simple projects. The hand-held logic analyzer was implemented using an LCA to trigger and sample signals and a microcontroller to control an LCD to display the waveforms. The LCA is programmed using a configuration ROM; during development, the UWTester was used to emulate this ROM. In both the robot project and the logic analyzer project, the tester greatly sped-up the development cycle, by eliminating the need for either a custom bus interface to a host computer, or configuration ROMs that would need to be continually re-programmed.

5.3 ASIC functional testing

Although LogicWorks provides a powerful, flexible front end to the UWTester, it is not usually suitable for testing custom chips. LogicWorks drives the test chip at 100 Hz or less, depending on the complexity of the

simulated test environment. In general, chips with dynamic state cannot be tested at this slow rate. There is also a practical limit on the size of a simulation that can be described in LogicWorks. Therefore it is necessary to use the more powerful programming language environment, which provides the capability for much more complex tests and much greater speed. On-line program-driven tests run at 20 to 50 kHz and have proven fast enough to test a wide range of dynamic circuits. In the cases where a hard real-time guarantee is required, the off-line capability allows up to 5400 vectors to be sequenced at approximately 0.8 MHz.

Graduate students have used the programming interface to test a wide variety of ASICs designed at the Department of Computer Science and Engineering of the University of Washington. All the ASICs were fabricated through MOSIS, a government-funded broker for chip fabrication. Table 1 shows a summary of the attributes of four of the chips and their UWTester test program code sizes. Test program sizes are computed by simply adding the sizes of all C source modules, including comments, and excluding the library modules. These test programs are simply representative samples; they are not necessarily optimized for size or capable of detecting all types of faults.

Table 1: Attributes of chips tested by the UWTester

Chip	Pad Frame	Die Size	# Transistors	Test Program Size
Apex	84 pin PGA	7mm x 9mm	60K	25K bytes
Triptych	84 pin PGA	8mm x 9mm	10K	48K bytes
SCL	132 pin PGA	8mm x 9mm	100K	28K bytes
Chaos	132 pin PGA	11mm x 11mm	100K	41K bytes

Here are detailed descriptions of the four examples. All except for SCL were developed as term projects in CSE 568, a graduate-level course for VLSI ASIC design.

Apex [6] is a highly pipelined architecture for rapid generation of 3-D curved surfaces. An Apex chip takes four 3-D points as input and generates a spline curve as output at a rate of 5 million points per second, equivalent to 50K points per frame of NTSC video (1/30 s). Using the programming language environment, we created an interactive test suite. The user selects four points on the Mac screen by moving the mouse cursor and clicking the button, and then the spline curves are displayed directly on the screen. A practically infinite number of different tests can be performed without changing the program. This test serves as an excellent demonstration of the capabilities of the tester system – user feedback is instantaneous, and the graphical representation of the results makes the testing process more tangible and easier to understand. Working Apex chips were integrated into a Macintosh NuBus card to generate 3-D surfaces on a Tektronix stereo display monitor.

Triptych [7] is a new FPGA architecture created at the University of Washington. Triptych is similar to the Xilinx LCA but yields higher circuit density than Xilinx by integrating routing and logic resources. Evaluating the Triptych prototype chip via a standard test vector approach would have been extremely complex due to two factors. First, each test requires downloading a great deal of state information. Second, the mapping from Triptych configuration to test vectors is very complex. To simplify the functional testing of Triptych, the test program presents the user with a map of Triptych's internal logical structure. To configure the Triptych chip, operations to alter its programming are specified as operations on this logical structure, and the software automatically determines the proper programming bits to generate. By using the programming interface, as well as some simple character graphics, a highly customized test environment was developed. We tested a number of Triptych prototype chips for complete functionality.

SCL [8] is a systolic cellular logic chip for performing rapid computations on two-dimensional image data. Internally, the SCL is a SIMD architecture with 64 one-bit processing elements (PEs). Input and output of the SCL consist of a 64 x 64 array of pixels, each physical PE representing 64 logical PEs. The instruction set consists of bitwise logical operators where the inputs are taken from the PE and its neighbors. The basic test program for the SCL employs a library of routines that construct and download individual instructions. The program models the instruction execution in software and compares the expected result with the chip's output. After verifying the basic functionality of a test chip, we could then download an image to the test chip and process the image in a variety of ways. The test program queries registers in each PE to obtain the current image. Partially processed images are displayed on the Macintosh screen in graphical form, to let the user examine the image at each step in processing. We verified the functionality of the architecture and then determined the subset of chips that were defect-free.

The **Chaos Router** [9] is a packet routing chip for constructing multicomputer interconnection networks in a 2-D mesh or torus topology. There are five 16-bit bi-directional routing channels per chip: north, south, east, west and the processor channel. Packets consisting of twenty 16-bit words can be injected into and ejected from any channel. Two kinds of tests have been performed: all-channel tests and loopback tests. In all-channel testing, packet traffic is randomly generated for all channels concurrently. In loopback testing, random packet traffic is injected only at the processor channel. The other channels are paired up (east-west and north-south) so that packets loop around. In essence the chip is tested as if it were a 1 x 1 torus mesh. The UWTester programming language interface made it possible to run a large number of different tests using a small test program kernel. Random traffic patterns were easy to generate using the standard C random number generator. Using this testing strategy, we were able to test the first run of prototype chips much more completely than we had been able to simulate the design. Ultimately, the test program revealed a minor logic bug in the chip design that was not detected in the original simulation of the circuit.

5.4 ASIC speed testing

The flexibility of the UWTester interface allows the user to first create a functional test and then adapt it incrementally for speed testing. Speed tests are performed by adding physical circuitry to the test chip that allow the chip to run at a faster speed than during functional testing. It may be necessary to place the test chip onto a separate board and then run a ribbon cable between it and the UWTester (i.e., as an umbilical cord). The speed test program may use a different mapping between the chip pins and the tester socket, for instance, by mapping the footprint of the tester socket to the ribbon cable connector instead of directly to the chip pins. We have performed speed tests on Triptych, SCL and Chaos. In the Chaos speed test, a separate wirewrap board is used. The board contains a free-running crystal oscillator and a multiplexor to switch between the fast crystal clock and the slow UWTester-driven clock. The tester executes the speed test by using the UWTester to inject a number of packets into the processor channel with the slow clock, switching to the fast clock for a while, then switching back to the slow clock to eject and verify the packets. There are many benefits to this strategy. The amount of additional circuitry is minimal and much easier than starting from scratch. Code re-use is very high between the functional and speed testing versions of the Chaos test program. Nearly all the differences are isolated to the pin mapping phase.

5.5 Miscellaneous uses for the UWTester

A number of graduate student projects at the University of Washington are spin-offs of UWTester technology. One PhD project was to design, simulate, construct and program a 16-processor multicomputer called Meerkat [10]. A large subcomponent of the design was too large and complicated to simulate in LogicWorks because it did not fit in available memory. The solution was to relocate part of the simulated circuit into an FPGA placed in the UWTester. The size of the software part of the Meerkat simulation was reduced enough to fit in memory and then ran successfully.

The host parallel interface card for the UWTester has been used to interface several printed circuit board projects with a Macintosh. Projects include SCAM, for evaluating and programming the SCL chip described in Section 5.3 [8], and SPAM, for evaluating and programming the Caltech Asynchronous Microprocessor [11]. The students who developed programs for both SCAM and SPAM took advantage of the C programming environment that was developed for the tester.

6 Related work

The UWTester bears many similarities with programmable testbeds such as the Universal Logic Implementer [12], and the Virtual Hardware Schematic system from Aldec Inc. The ULI requires the user to configure a set of wires via a patch panel (plugboard). The ULI system includes personality cards for various TTL chips. There is

also an RS-232 interface to a host computer. Patch panels are detachable, so that ULI stations can be shared among several students. The ULI system is used for teaching digital logic design; however it is not generally applicable to ASIC testing. Moreover, dynamic chip testing would be difficult due to the low bandwidth of the RS-232 interface. The Virtual Hardware Schematic (VHS) system is closer in spirit to the UWTester. It uses a schematic editor and simulator scheme that yields basically the same functionality as the tester with LogicWorks as the front end. Like the UWTester, VHS eliminates the need for a patch panel because all the pin mapping is done in software. Aldec also provides FPLD and FPGA support for VHS as an option at added cost. Like ULI, VHS is useable for educational purposes. The literature we have on VHS does not include any discussion of a general purpose programming language interface. As we have demonstrated in this article, a programming language interface adds flexibility. We often use the programming language interface for ASIC testing because we can easily pinpoint the parts of the test we wish to run at off-line testing rates.

7 Conclusions and current status

The UWTester is an inexpensive yet versatile system for testing chips, circuits and subsystems; it also finds use as a hardware testbed and an FPGA development environment. The UWTester is inexpensive to build; the end-user cost is an order of magnitude less than the typical "low cost" commercial chip tester. The UWTester serves a large number of students including those who are not testing custom chips; it is a system that a student can grow with, from courses on introductory digital logic to senior projects and graduate level research. The UWTester has been used in undergraduate digital design education at the University of Washington for the past two years [2]. Students have used the UWTester to construct a variety of projects using highly integrated logic, such as FPLDs, FPGAs and microcontrollers. Graduate students in Computer Science and Engineering have employed the UWTester for testing a variety of custom chips.

We believe that the successful application of the tester in the curriculum was due to three decisions. The first decision was to emphasize on-line testing rather than off-line testing. In on-line testing, the circuit under test is tightly coupled with a host computer. Test programs employ the full power of the host system, including standard high-level language support, branching, subroutines, recursion, input from keyboard and pointing devices, and output to bitmapped displays. This highly interactive form of testing provides students with a much richer experience in debugging their designs. Because changes can be made quickly and directly, the iteration cycle is much faster than the indirect approach that uses a file of test vectors. When testing devices with dynamic storage elements, the UWTester's inexpensive form of off-line testing is crucial to maintaining a fast enough clock rate to refresh the dynamic elements of the circuit.

The second important decision was to implement the UWTester using Xilinx LCAs both for the controller and the chip pin drivers. LCAs helped increase the circuit density and reduce the cost of the tester. Besides making

it possible to add features and fix design bugs long after the printed circuit boards were fabricated, LCA reprogrammability yields a much more student-friendly tester. First, LCAs consume little power and sink only a few milliamps per pin; the LCA pin drivers often prevent the user from destroying a test chip that was installed or programmed improperly. Second, the number of wires the students must install on the tester's socket is limited to power and ground connections, as the tester maps the remainder of the pins through the LCAs. This feature dramatically improves the setup and tear-down time; as a result, students are encouraged to test more often and more completely.

The third important decision was to design a straightforward programming interface to the tester, to simplify the task of integrating it in with different user environments. We have constructed and are currently using a variety of test environments, each intended for a different class of user. LogicWorks is the standard user environment for the UWTester in the undergraduate design laboratory. Students testing custom chips generally use the programming language environment which enables them to embed complex computations (mimicking those of the circuit under test) within the test program.

We are currently investigating two enhancements to the tester hardware. First, we are developing a SCSI interface for the UWTester. Macintoshes and Unix workstations have built-in SCSI ports for easy connection to external hard disks and streaming tape drives; connecting a host system to the tester would thus require only a standard SCSI cable, making the system substantially cheaper[4]. Second, we are investigating developing new pin drivers based on the Xilinx 4000 series of FPGAs. The on-chip RAM in the 4000 series LCAs will be used to apply a set of vectors in rapid succession and thus permit limited speed testing.

UWTester hardware is currently available from Applied Precision, Inc., in Mercer Island, WA. UWTester software and documentation is available for Macintosh users via anonymous FTP from `shrimp.cs.washington.edu` in the directory `/pub/MacTester`. A World Wide Web home page is available as `http://www.cs.washington.edu/homes/mckenzie/mactester.html`. The authors have also created a twenty-minute educational videotape on using the UWTester with LogicWorks for FPGA development; copies are available upon request. The authors can be contacted by e-mail using the Internet address `mckenzie@cs.washington.edu`.

Acknowledgments

We would like to acknowledge many of the students and staff at the University of Washington who have been involved with the UWTester project. Brian Eng developed an early version of the NuBus interface. Christine

[4]We did not pursue this configuration initially because SCSI has lower bandwidth and higher latency than the 32-bit parallel host interface. The on-line testing rate under SCSI is about an order of magnitude slower.

Bromfeld and Jeff Hirschberg helped develop the PC ISA bus interface. Eka Ginting helped with much of the C library development. Warren Jessop installed and maintained our development systems and networks. Stephen Lee has been supervising students and training them on the three UWTester systems in the Undergraduate Design Laboratory. A number of students developed test programs, including Scott Hauck (Triptych) and Kevin Bolding (Chaos). We also acknowledge our contacts from industry. Ian W. Jones[5] was our initial contact at Apple Computer. Don Snow, Ron Seubert, John Stewart, Andy Snow and Vince Dayton at Applied Precision Inc. helped with technology transfer and commercialization of the UWTester. Finally we acknowledge the institutions that have funded the project: Apple Computer, the National Science Foundation, the Advanced Research Projects Agency, the Washington Technology Center and Xilinx, Inc.

Bibliography

1. N. R. McKenzie, L. McMurchie and C. Ebeling. The UW MacTester: A Low-Cost Functional Tester for Interactive Testing and Debugging. Technical report UW CSE TR 92-10-08, Univ. of Washington, Dept. of CSE, Oct. 1992.
2. C. Ebeling and G. Borriello. "Establishing a Modern Digital Design Lab." *Proceedings of the 4th Microelectronic System Education Conference*. Conference Management Services, Menlo Park CA, July 1991. Also available as technical report UW CSE TR 92-08-01, Univ. of Washington, Dept. of CSE, August 1992, chapter 4.
3. C. Ebeling and N. R. McKenzie. "MacTester: A Low-Cost Functional Tester for Interactive Testing and Debugging." *Proceedings of the 3rd Microelectronic System Education Conference*. Conference Management Services, Menlo Park CA, July 1990. Also available as technical report UW CSE TR 92-08-01, Univ. of Washington, Dept. of CSE, August 1992, chapter 3.
4. N. R. McKenzie. *The UW VLSI Chip Tester*. Technical report UW CSE TR 89-12-01, Univ. of Washington, Dept. of CSE, Dec. 1989.
5. R. H. Katz. *Contemporary Logic Design*. Benjamin/Cummings, Redwood City CA, 1994, pp. 421-426.
6. R. Bedichek et al. "Rapid Low-Cost Display of Spline Surfaces." *Advanced Research in VLSI: Proceedings of the 1991 UCSC Conference*. MIT Press, Cambridge MA, March 1991, pp. 340-355.
7. S. Hauck, G. Borriello and C. Ebeling. "TRIPTYCH: An FPGA Architecture with Integrated Logic and Routing." *Proceedings of the 1992 Brown/MIT Conference on Advanced Research in VLSI and Parallel Systems*. MIT Press, Cambridge MA, March 1992.
8. R. Ling and S. L. Tanimoto. *A CMOS Chip for Systolic Cellular Logic*. Technical report UW CSE TR 90-02-05, Univ. of Washington, Dept. of CSE, Feb. 1990.
9. K. W. Bolding. *Chaotic Routing: Design and Implementation of an Adaptive Multicomputer Network Router*. PhD dissertation, Univ. of Washington, Dept. of CSE, July 1993.
10. R. Bedichek and C. P. Brown. *The Meerkat Multicomputer*. Technical report UW CSE TR 93-09-05, Univ. of Washington, Dept. of CSE, Sept. 1993.
11. A. J. Martin et al. "The Design of an Asynchronous Multiprocessor." *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference*. MIT Press, Cambridge MA, March 1989, pp. 351-373.
12. M. L. Manwaring et al. "Universal Logic Implementer: A General Purpose Tool for a Digital Logic Design Laboratory." *IEEE Transactions on Education*, Vol. 34, No. 2, May 1991.
13. Xilinx Inc. *The Programmable Logic Data Book*. 2100 Logic Drive, San Jose CA, 1993.

[5] Jones is now with Sun Microsystems Labs.

List of figures

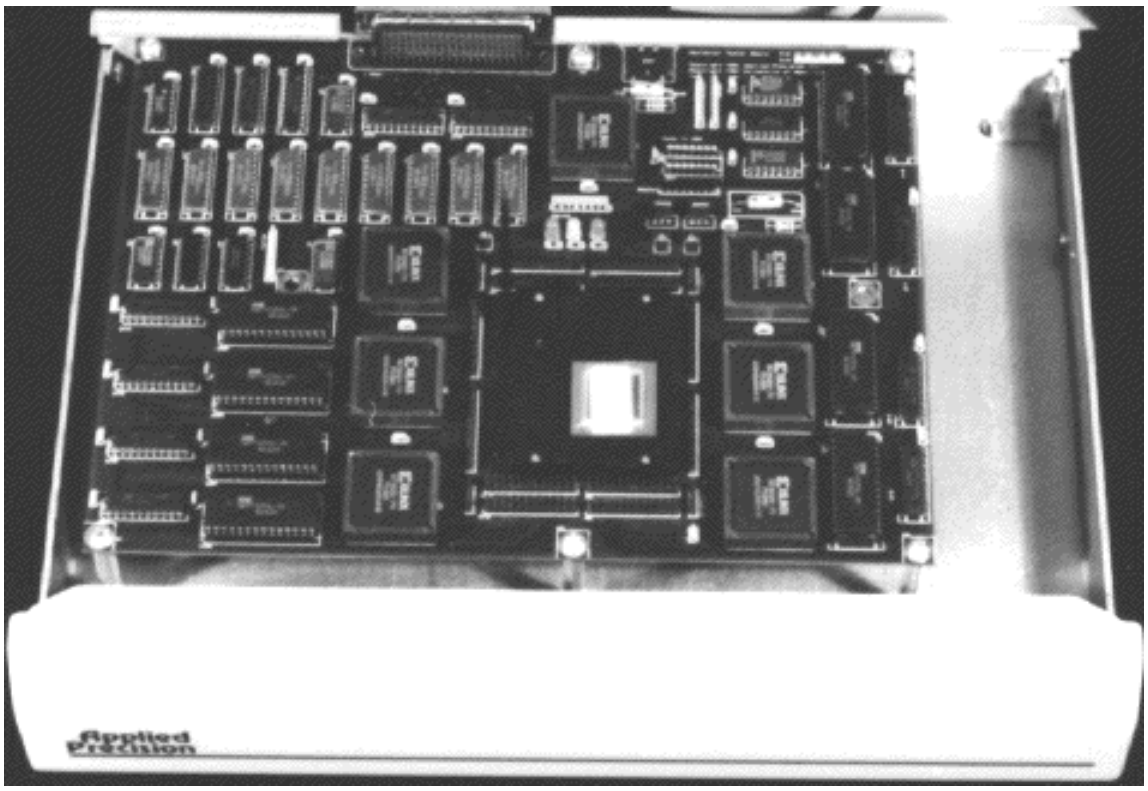

Figure 1

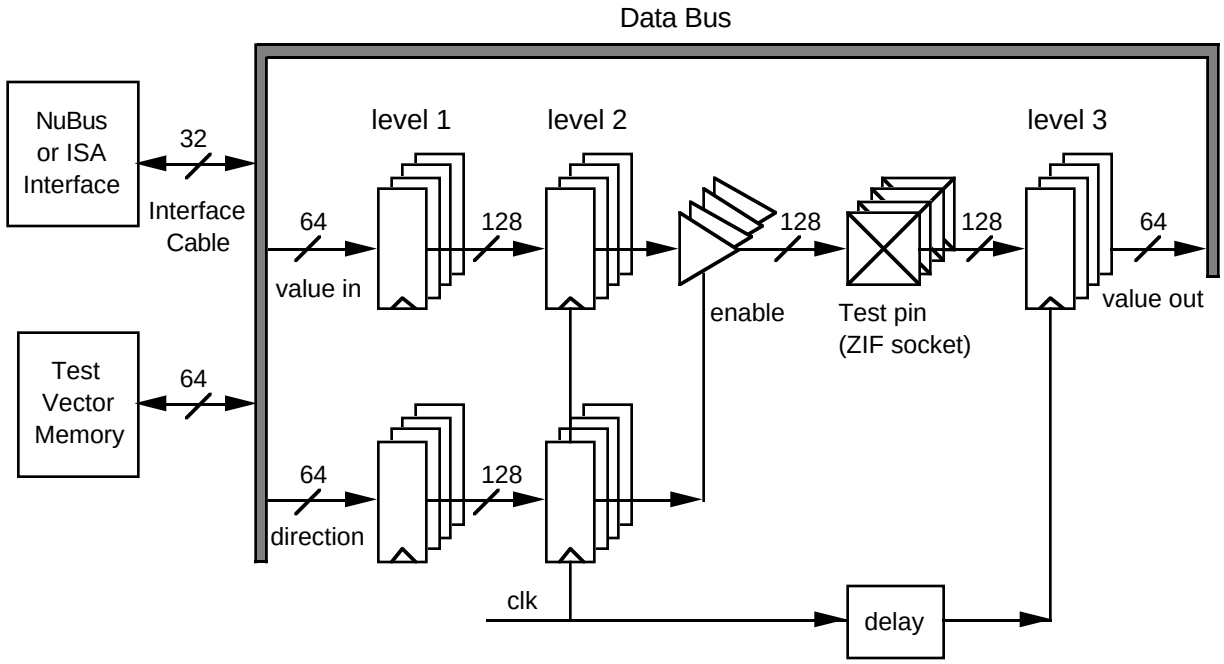
Data Bus
NuBus
or ISA
Interface
32
Interface
Cable
level 1
level 2
level 3
64
value in
128
128
enable
128
Test pin
(ZIF socket)
64
value out
Test
Vector
Memory
64
64
direction
128
clk
delay

Figure 2

```
 1      /*
 2      **  163.c
 3      **  test program for the 74LS163 counter chip
 4      */
 5
 6      #include <stdio.h>
 7      #include <tester.h>
 8      #include <DIP16.h>
 9
10      Sigptr clr, clk, in, enp, load, ent, out, rco;
11
12      void DefineSignals(void)
13      {
14          Vdd("16");      /* Indicate power pins */
15          GND("8");
16          DefineInvSignal (clr,   "1",    INPUT, 0);
17          DefineSignal    (clk,   "2",    INPUT, 0);
18          DefineSignal    (in,    "6:3",  INPUT, 10);
19          DefineSignal    (enp,   "7",    INPUT, 1);
20          DefineInvSignal (load,  "9",    INPUT, 1);
21          DefineSignal    (ent,   "10",   INPUT, 1);
22          DefineSignal    (out,   "11:14",OUTPUT, 0);
23          DefineSignal    (rco,   "15",   OUTPUT, 0);
24      }
25
26      void clockIt(void)
27      {
28          Set(clk, 0); Next();
29          Set(clk, 1); Next();
30      }
31
32      void MainTest(void)
33      {
34          short i;
35          unsigned long count;
36
37          count = 10;     /* Load counter */
38          Set(in, 10);
39          Set(load, 1);
40          clockIt();
41          printf ("Expected value %ld :: chip = %ld\n", count, Get(out));
42
43          Set(load,0);   /* Start counting up */
44          for (i=0; i<21; i++) {
45              clockIt();
46              count = (count+1) & 15;   /* lop off high bits */
47              printf ("Expected value %ld :: chip = %ld carry out = %ld\n",
48                  count, Get(out), Get(rco));
49          }
50      }
```

Figure 3

```
51      void MainTest(void)
52      {
53          short i;
54          unsigned long count;
55
56          BEGIN_OFFLINE;
57          count = 10;
58          GENERATE {
59              Set(in, 10);
60              Set(load, 1);
61          }
62          clockIt();
63          VERIFY {
64              printf ("Expected value %ld :: chip = %ld\n", count, Get(out));
65          }
66
67          GENERATE { Set(load,0); }
68          for (i=0; i<21; i++) {
69              clockIt();
70              count = (count+1) & 15;   /* lop off high bits */
71              VERIFY {
72                  printf ("Expected value %ld :: chip = %ld carry out = %ld\n",
73                  count, Get(out), Get(rco));
74              }
75          }
76          END_OFFLINE;
77      }
```

Figure 4

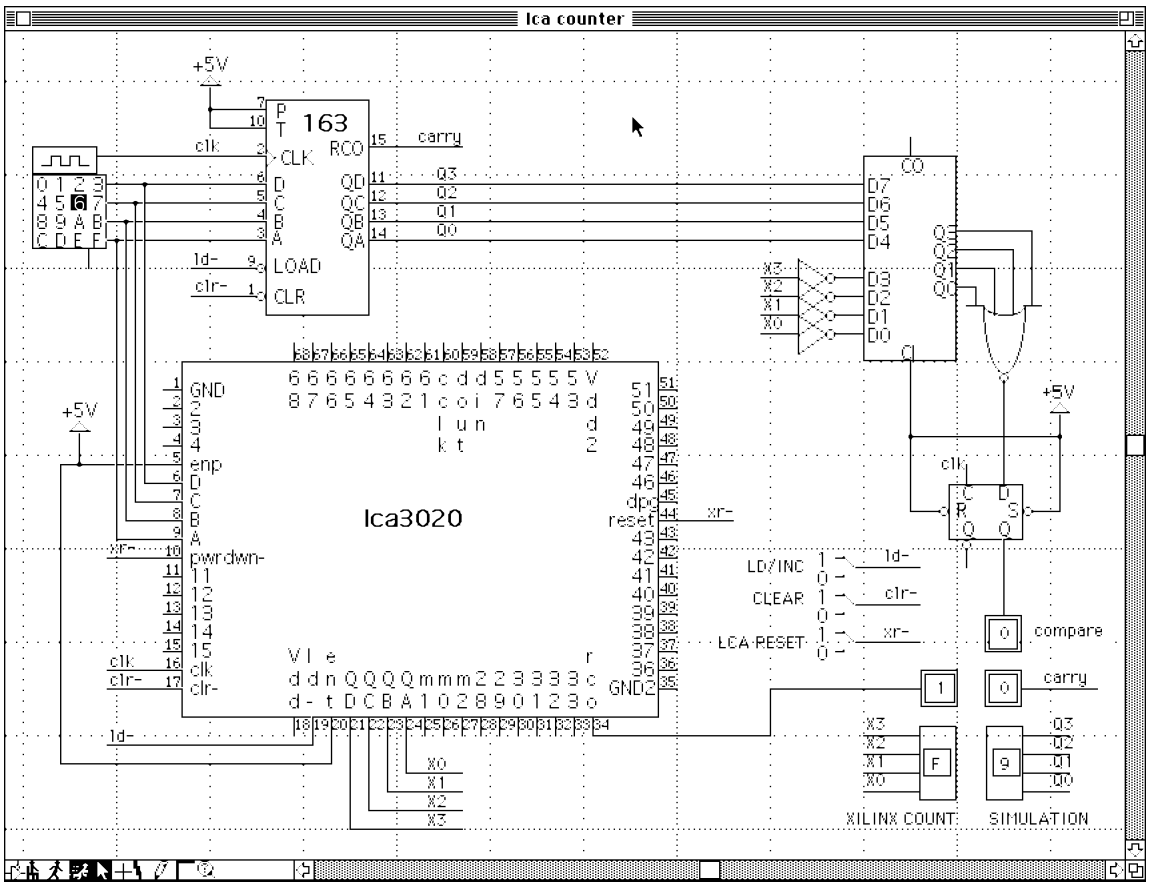
lca counter
+5V
163
CLK
RCO
carry
LOAD
CLR
ld-
clr-
clk
Q3
Q2
Q1
Q0
CO
D7
D6
D5
D4
D3
D2
D1
D0
CI
lca3020
GND
enp
pwrdwn-
reset
xr-
LD/INC
CLEAR
LCA RESET
compare
carry
XILINX COUNT
SIMULATION

Figure 5