ASIC Tester

Alfred Gifford, Ruben Gottardi, Daniel Khoury, Tyler Martinsen
University of Utah
Salt Lake City, UT 84102

***Abstract*—Our senior project is an ASIC tester meant to test chips that students in CS/ECE 5710 (or 6710) fabricate. The ultimate goal of this project to mimic and replace the University of Utah's LV500, a Tektronix logic verifier from 1989, which has failing parts and requires difficult-to-find replacement parts. Our project lacks some advanced features seen on the LV500, such as current limitation, schmooing, FTP support, but successfully replicates much of the logic verifier's core functionality.**

I. Introduction and Motivation

At the University of Utah, students enrolled in CS/ECE 5710 (or 6710) design an integrated circuit (IC) for their final project and have the option of letting MOSIS, an IC foundry, fabricate their chip. After fabrication, as per MOSIS's rules, students are required to test their chips and report their results back to MOSIS. CS/ECE 6712 teaches students how to test their chips using a device called an ASIC tester.

Presently, the University of Utah has two ASIC testers: a Verigy 93000 and an LV500. At the time that we proposed our project, the Verigy 93000 was seen as a very difficult machine to use, with several functionality issues plaguing its use. The LV500 is simpler to use and has been used in the past by students to test their fabricated chips, but the machine is over 20 years old; parts of the machine are failing, and finding replacement parts is becoming increasingly difficult with time.

ASIC testers are generally very expensive (usually around millions of dollars), and the chips that students test usually do not require the level of functionality provided by such expensive machines. Rather than require the University of Utah to continually hunt down obscure replacement parts or spend money on a new ASIC tester, we proposed to construct a low-cost ASIC tester design based primarily upon an FPGA board and a phase-locked loop (PLL) module.

(As of this writing, the Verigy 93000 appears to be very close to operational. However, there is still considerable interest among some individuals in having our project available as a substitute.)

II. Background

ASIC testers are machines that allow users to send electrical signals to a chip, sample output signals from that chip, and verify whether the signals match their expectations or not. This section primarily reviews the operation of the LV500-series ASIC testers, whose functionality we attempted to mimic in our project. In the LV500, timing waveforms (which can be thought of as internal clock signals) are configured by the user and are used to coordinate when signals are sent to and sampled from a chip. Figure 1 illustrates a so-called test cycle. Signals are sent to a chip at the rising edge of a timing waveform and sampled from the chip at the falling edge of the timing waveform. Multiple test cycles can be used concurrently to send inputs and sample outputs at different times but within the scope of a single test cycle period, as shown in Figure 2. Other important aspects of the LV500s functionality include:

* The ability to parse an external test file.
* Schmooing, where a chip is tested under varying electrical conditions (for instance, 4.3V is chosen to represent a 1 instead of 5.0V).
* A GUI allowing users to run tests.

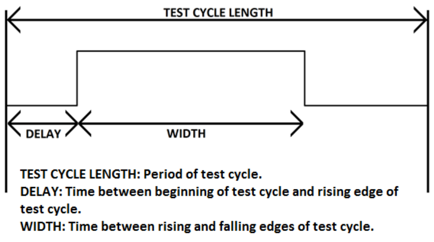

TEST CYCLE LENGTH
DELAY
WIDTH
TEST CYCLE LENGTH: Period of test cycle.
DELAY: Time between beginning of test cycle and rising edge of test cycle.
WIDTH: Time between rising and falling edges of test cycle.

Fig. 1. LV500 timing definitions.

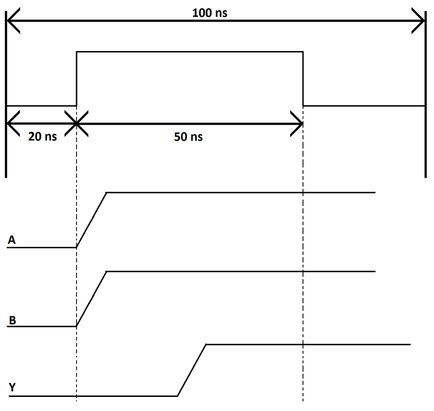


Fig. 2. Testing an AND gate using the LV500.

III. Project Implementation

The system as a whole can be divided into three sections:

* The main PCB, used for housing the ASIC.
* The FPGA systems that work to generate timing waveforms and initiate tests.

* The software used to interface with the rest of the system.

Two parts that we used and are mentioned throughout the remainder of this paper are the Numato Saturn Spartan-6 FPGA Development Board (see Figure 3) and the Digilent Atlys board (see www.digilentinc.com/atlys/ for details). Figure 4 below shows a hardware flow diagram representing our project.

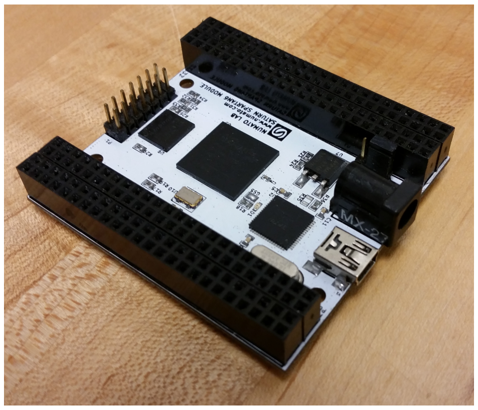


Fig. 3. Numato board.

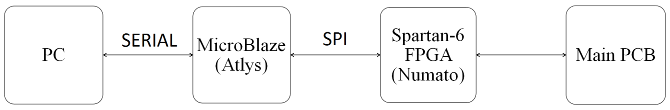


Fig. 4. Hardware flow diagram.

A. Main PCB

A fundamental part of our project is being able to easily connect a MOSIS-fabricated chip. The main PCB's purpose is to provide housing for these chips. We defined our requirements as:

* The ability to access a large number of I/O pins.
* The ability to use any pin as an input, output, power connection, and ground connection.
* Voltage leveling between the Numato board (which operates on 3.3V) and the chip under test, which operates on 5V.

1) Pin Count: The first requirement listed was constrained by the number of I/O pins on the Numato board. The Numato gave us access to 150 pins. The largest MOSIS-packaged chip that we could support consists of 139 pins, so we devoted 139 Numato pins to the main PCB and the remaining 11 pins for SPI communication.

2) PCB Design: Using jumpers and jumper wires, the user has the ability to manually wire power and ground and determine whether they wish to have an ASIC pin float or be configured to an input or output. Additionally, the user can route power out to an external adjustable voltage regulator if they wish to schmoo manually. The final PCB is shown in the image at the following page: http://www.eng.utah.edu/~dkhoury/main_pcb.jpg

3) Voltage Leveling: Figure 5 below shows the schematic of the voltage leveler we ultimately used.

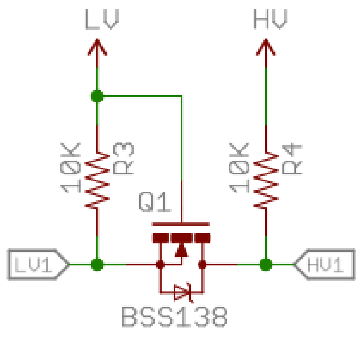


Fig. 5. Voltage leveler. Schematic from http://dlnmh9ip6v2uc.cloudfront.net/datasheets/BreakoutBoards/Logic_Level_Bidirectional.pdf.

B. FPGA: Verilog, MicroBlaze

We use a Spartan-6 FPGA to generate timing waveforms and house all of the logic required to send and sample bits to/from the ASIC at appropriate times. As a whole, the FPGA is configured as a SPI slave. The image at the following page illustrates the circuits implemented onto the board (sans the finite state machines used for control signals): http://www.eng.utah.edu/~dkhoury/verilog.png

A phase-locked loop (PLL) built into the FPGA is used to generate the timing waveforms. The PLL is a powerful piece of hardware, able to perform the following functions:

* Multiplying clocks.
* Dividing clocks.
* Allowing the frequencies, phase shifts, and duty cycles of the output clocks to be dynamically reprogrammed. (See here: http://www.xilinx.com/support/documentation/application_notes/xapp879.pdf.)

The dynamic reprogrammability is especially important. One general goal with major projects like ours is to leave the user of the system with as minimal work to do as possible. Hence, it's best if the timing waveforms are dynamically reconfigurable so that the user does not have to resynthesize the Verilog onto the FPGA for their specific needs, but can instead specify their desired configuration state through more abstracted methods.

Due to routing issues, we could not feed the 100 MHz clock signal on the Numato board directly into the PLL; we had to use a digital clock manager (DCM) as effectively a buffer. Routing issues also led us to output no more than two clock signals from the PLL.

We configured the PLL to output all clock signals at 2 MHz, with duty cycles fixed at 50 percent. (See the 'Pitfalls and Discoveries' section for our justification of this design choice.) One clock signal is designated as the "DATA" timing waveform and the other clock signal is designated as the "CLOCK" timing waveform. When testing a combinational chip, such as an inverter, the user is expected to only use the DATA waveform. When testing a sequential chip, such as a register, the user is expected to use both the DATA and CLOCK signals, and can specify the phase difference between the DATA and CLOCK signals (either 45, 90, or 135 degrees). Output bits are always sampled on the falling edge of the DATA waveform.

The FPGA accepts six one-hot chip select (CS) lines, allowing it to retrieve the following test information (note that the last three bullet points are only relevant for pins configured to be inputs to the ASIC):

* A PLL reconfiguration state. This data must only be sent if the user is testing a chip that requires two timing waveforms.
* Which pins are inputs and which pins are outputs.
* A signal to initiate a test cycle.
* The bits to be sent to the ASIC.
* The force format for each pin. We implemented all of the force formats that the LV500 supports except for RIH.
* The timing waveform (DATA or CLOCK) that each pin adheres to.

The image at this webpage (http://www.eng.utah.edu/~dkhoury/fsms.png) shows the finite state machines (FSMs) that exist in the Verilog. There are two reset signals: one to reset the DCM, and one labeled as 'GLOBAL_RST'. An FSM is required for the reset logic due to the asynchronous nature of the entire design. Otherwise, the design is not overly complicated. Generally, we wait for a SPI message and then decode it. If the SPI message instructs us to reconfigure the PLL, we trigger a FSM that does so. If the SPI message instructs us to initiate the test, through careful timing, we push all of the clocked-in bits through the double-buffered registers, allow the PLL clocks to run (they are normally blocked by the BUFGMUX modules) for about 500 nanoseconds, and then halt the test. The TEST_WAIT period ensures that we don't run two tests in a row, in case the corresponding CS line is held down for longer than anticipated.

To provide easy pin access to the Numato board, we created a shield for it. The schematic for this shield is available here (http://www.eng.utah.edu/~dkhoury/pcb_boards/FPGA_Shield.PDF) and the layout is available here (http://www.eng.utah.edu/~dkhoury/pcb_boards/fpga_shield_layout.PNG).

The MicroBlaze code implemented onto the Atlys board is, in essence, a SPI-to-serial and serial-to-SPI converter. Using the XPS SPI drivers provided by Xilinx, the MicroBlaze code's sole purpose is to extract serial messages from a PC (wherein the messages contain testing information), send analogous SPI messages to the Numato board, retrieve test results from the Numato board, and convert the test results into serial messages send over the COM port.

C. *C# and Serial Contract*

For a user to easily instruct our machine how to test a chip, we are required to take a test file as an input and send the contents to our Numato board. This process includes an application running on a host computer, the Atlys FPGA board, the Numato FPGA, and ultimately the ASIC under test on our main PCB.

The test input consists of a test file with pin settings, input/output designations, and test patterns. Erik provided us with some example .MSA test files used by the LV500 system and we adapted the template for our use. Along with our modifications we also allow a wider range of file extensions to be used (e.g. .txt).

An application we created handles the test input file by parsing the file to gather the required settings and test patterns to be tested. This application is written in C# and has the primary duties of parsing, converting, sending/receiving, and processing the returned data to determine whether or not expected results were received, using a handshaking protocol with the MicroBlaze system on the Atlys board.

Figure 6 below demonstrates a high-level diagram of the process flow.

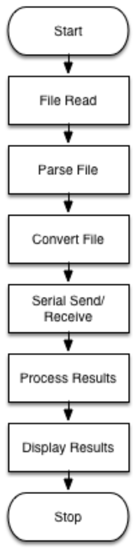
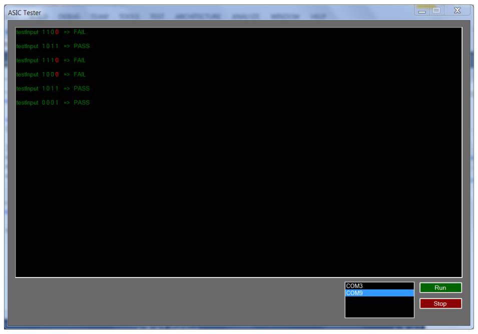


Fig. 6. C# Process Flow

The application performs the functions shown above by implementing regular expressions to parse and store data from the input file as objects. The stored objects are converted into appropriate serial calls that are then transmitted via serial to the Atlys board (which effectively serves as a SPI-serial relay). Concurrently, the application creates a thread to handle incoming transmissions from the serial SPI relay which returns acknowledgment and test result data.

When an acknowledgment is received, the application knows it is appropriate to send the next command in queue. If test result data is received, the application processes the returned data and compares the expected results against the received results, informing the user of which bits came back as expected and which bits did not.

Figure 7 below shows our GUI used to test a D flip-flop. The user specifies which COM port the MicroBlaze system is connected to, uploads a test file, and the back-end code works to execute the test. The results are promptly displayed to the user on the terminal-like screen shown in the image. Bits that were sampled incorrectly are shown in red text. (A larger picture of this image can be seen here: http://www.eng.utah.edu/~dkhoury/colored_bits.PNG.)



```
testInput 1 1 0 0  => FAIL
testInput 1 0 1 1  => PASS
testInput 1 1 1 0  => FAIL
testInput 1 0 0 0  => FAIL
testInput 1 0 1 1  => PASS
testInput 0 0 0 1  => PASS
```

Fig. 7. C# GUI.

D. Demo Chip

Because our project lacks actuators and any kind of light art, we had to consider two audiences when demoing our project: individuals familiar with electronics and individuals unfamiliar with electronics. We decided it would be optimal demonstrate at least one simple sequential circuit to the former audience and at least one simple combinational circuit to the latter audience.

MOSIS held an educational fabrication run in June 2014, and we took advantage of this opportunity by fabricating a digital chip with the following circuits:

* Inverter.
* AND gate.
* OR gate.
* 4-bit adder.
* 2-bit multiplier.
* 2-bit decoder.
* 4-bit encoder.
* 4-bit binary up-counter.
* 4-bit shift register.
* 1-bit D flip-flop.
* 4-bit ring counter.

Ultimately, in our demonstration, we demoed the inverter, AND gate, and OR gate to everybody, and the D flip-flop to people familiar with electronics.

E. Design Process

1) Main PCB: We initially assumed designing the main PCB would be trivial; surprisingly, that was not the case. The main PCB was in fact fabricated three different times. The first design had several problems. The most prominent issue was that the drill holes for the ZIF socket were too small. When designing the board in Altium, we used the datasheet for the socket to specify our hole sizes but we neglected to take into account the effect of the plating on the hole sizes. There were also some other errors (shorts between pins due to copy-and-paste errors) which would have made this board nearly unusable and very hard to debug. In the second run, we fixed the aforementioned issues and made some additions, including heavy use of silkscreen and power-indicating LEDs. After we received this board, we soldered all components onto the board and tested the circuit. It quickly became clear that the board was malfunctioning, which we later learned was due to swapping two pins on our voltage translators. By the time we decided to fabricate a third PCB, we were three weeks away from our demo date. For this reason, we decided to plaster onto the main PCB several breakout boards from Sparkfun that contain the voltage translating circuit. The final schematic be seen here (http://www.eng.utah.edu/~dkhoury/pcb_boards/ASIC_Sch.PDF) and the final layout can be seen here (http://www.eng.utah.edu/~dkhoury/pcb_boards/main_pcb_layout.PNG).

2) PLL: We initially proposed to create an ASIC tester with both functionality testing and timing testing. In the latter case, the idea is to see how quickly we can sample output bits from a chip after sending input bits to the chip before the sampled bits become incorrect in value. We believed the Spartan-6's PLL could achieve this goal for us, but testing the PLL proved otherwise for the following reasons:

* The PLL appeared to have major stability problems at all frequencies except for 2 MHz. Even at 2 MHz, the clock signals sometimes exhibit some jitter.
* The PLL will not necessarily produce the exact frequency, phase shift, and duty cycle programmed into it. Because of the way in which the PLL prioritizes certain parameters over others, specifying a frequency of 5 MHz, 37 degree phase shift, and 30 percent duty cycle might produce a clock with 5 MHz frequency, 45 degree phase shift, and 30 percent duty cycle.

Due to these issues, we reframed our project to serve only as a functionality tester.

3) Software Interfacing: The initial conception of our project was to have a keyboard-driven UI running on a Numato FPGA board. The Numato board would then send, receive, and process the data from our main PCB board to determine and present the test results to the user. Due to challenges and hardware limitations described in Appendix A: Pitfalls and Discoveries, we were forced to adapt and modify our original direction (the final product satisfied virtually all of our original objectives, but the path was very different than anticipated). Design changes forced our application to be written for and executed on a PC system. This change presented fewer hardware limitations and a wider array of software libraries. By moving our application to a host PC the FPGA board was then used primarily for a serial-SPI relay.

IV. Evaluation

The main PCB fulfilled its primary duty of providing housing to a MOSIS chip and supporting voltage leveling. Multimeter tests never revealed any issues with the main PCB, though we were unable to directly incorporate the following bonus features: *

* Schmooing abilities. (Our compromise was to provide the user with the option to send the power from the barrel-type power jack source to an external adjustable regulator, for what is basically manual schmooing.)
* Equalized trace lengths, which would have allowed us to provide the user with an estimate of the delay time between the Numato board and the ASIC. We attempted to use Altium's tools to equalize trace lengths, but the tools could only bring the traces within four to six inches of each other. (Indeed, equalized trace lengths is a surprisingly hard problem.) Since we rebranded our project as a functional tester, this bonus feature became unimportant. If users wish, they can hook up a logic analyzer to the jumpers surrounding the ZIF socket and get a rough approximation of their chip's performance.

The PLL proved to be less versatile than we had hoped for. The Verilog as a whole is able to do everything that we intended it to do, save for the lack of timing options imposed by the PLL's functional constraints. An ideal PLL would have allowed us to specify any frequency, phase shift, and duty cycle for the output clocks. In the end, we were happy that we were able to make any use of it!

The initial intent for the software side of our project was to have a GUI which would allow users to enter and change the tests we would perform via a keyboard. This GUI would ideally have allowed a user to save those tests (a feature we were considering implementing if we had time). However, because we ended up moving the GUI from the MicroBlaze system to a computer program written in C# (which can be run on any Windows system), we were able to take advantage of a PC's offerings (like saving files). We subsequently decided to focus more on loading saved tests versus developing tests in a GUI, which is generally infeasible for users to do due to the complexity of many test files. This way, a user can write a test in an MSA file with their favored IDE and not have to deal with any possible disconnects or incapabilities a GUI may provide.

The more straightfoward GUI we developed now provides a clean interface which allows a user to select the COM port they are using and the file they want to test. After selecting the file the tests are executed and the user is presented with the results formatted according to the patterns section of their code with the term 'Pass' or 'Fail' appended. Should a test fail, the user is also presented with the output bits that failed highlighted in red to help clarify where the error may exist. As of now, the test file only supports one template (in LV500-speak) – we plan to include support for multiple templates in the future.

As a whole, the project appears to work (in that a chip can be correctly tested, when using a machine like the LV500 for reference), though we've only tested simple circuits so far. We do plan to test the project further with the more complicated circuits on the demo chip (e.g. binary up-counter).

V. Conclusions

As simple as the idea of a logic verifier is, the many details behind its implementation make it a challenging project to implement. It's especially incredible to think that even with the technology of 2014, it's no trivial task to mimic a machine created 25 years earlier! We are proud that we were able to create an ASIC tester that, at the very least, can test relatively simple chips (further testing will show whether it can test more complicated chips). We accomplished most of our proposed goals and learned much along the way. For reference, Figure 8 below shows the current project as it stands. We plan to continue working on this project, as we are determined to at least provide a substitute tester for students in CS/ECE 6712. Additional work includes:

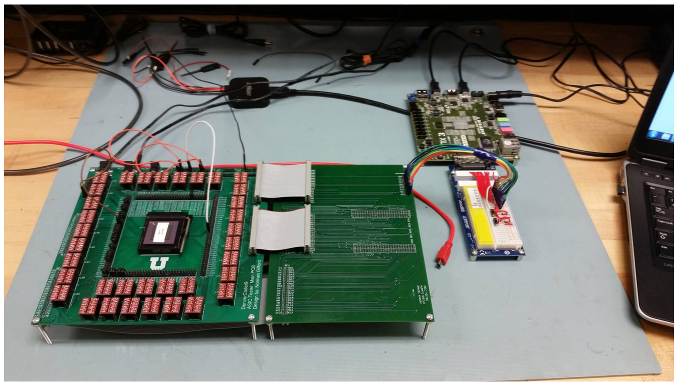


Fig. 8. Current system. Shown is the main PCB, the FPGA shield created for the Numato board, the MicroBlaze system (Atlys board), and a PC to the very far right.

* More versatility in the test files (e.g. multiple template support).
* Further testing to provide confidence in the machine.
* Finishing up documentation on how the system as a whole works.
* (Ideally) the ability to convert a PDF of a bonding diagram into the majority contents of a test file using an image-processing program.

Our website is located at http://atr.eng.utah.edu/~gottardi/MixedSignals/index.php. Note that pitfalls and discoveries is also mentioned on the 'Project Results' page.

Acknowledgment

We are thankful for all of the individuals that provided us with help throughout the course of this semester. In particular, we'd like to thank the following people:

* Brent Mellor, who allowed us to use his workplace's reflow oven.

* Erik Brunvand, who provided us with parts and guidance throughout the project.
* Anh Luong and Mike Empey, who helped us create a stencil for our main PCB.
* Travis Stroud, who provided us ribbon cables.
* Our families, who put up with our absences and will now need to get to know us again.

Appendices

A. Appendix A: Pitfalls and Discoveries

1) Numato did not provide any *.bsb files for the board we purchased. This made it virtually impossible to flash any MicroBlaze code to the board. We attempted to work with Numato on this issue but their technical support was less than helpful. In response, we moved the MicroBlaze portion of the code to an Atlys board that Erik Brunvand provided us.
2) We initially began development for the software side in Java. We had already written a parser when we attempted to add a serial connection in our code. This led to the harsh discovery that Java, being run in a virtual sandbox, does not play well with COM ports. After weeks of troubleshooting we abandoned Java and moved over to C# which provided a better connection to the COM ports and required significantly less code.
3) The initial voltage translators that we purchased and tested (part number TXB0108) ”autodetect” the direction of voltage propagation by using extremely weak internal drivers to drive voltages. Others should use this particular device with caution due to its poor driving capabilities.
4) It is hard to get even simple PCB designs correct on the first try. With the first version of our main PCB, we didn't account for how plating would affect hole sizes. On the first revision, we accidentally created a short between two pins due to a copy-and-paste error. Not until the second revision did our board work to our satisfaction.
5) The drain and source of a MOSFET are NOT interchangeable due to a parasitic diode (technically a zener diode but can generally be thought of as a unidirectional diode) that lies between those two terminals. The first revision of our main PCB failed because we mixed up the source and drain locations on the MOSFETs we used. For more information on this topic, one can research ”source-drain diode.” In general, MOSFETs should be handled as follows: *
   * Of the voltages applied to the source, drain, and body, the source should be tied to the lowest voltage (NMOS) or the highest voltage (PMOS).
   * The drain and source of a MOSFET are not necessarily interchangeable, whether the application is analog or digital.
6) The PLL code provided by Xilinx refused to implement onto the Spartan-6 FPGA on the Numato board. (The code synthesized but an odd error related to the location of the clock signal on the Numato board prevented the code from mapping successfully onto the hardware.) Several attempts to resolve the bug, including seeking help on the Xilinx forum and emailing the author of the PLL code for assistance, failed to bring about a resolution. As a desperation measure, we attempted to tie the output of a DCM to the PLL – with a few tweaks, we managed to get the PLL to implement onto the FPGA.

B. Appendix B: Bill of Materials

The Bill of Materials (BOM.pdf) can be found at http://www.eng.utah.edu/~dkhoury/BOM.pdf.

C. Appendix C: Schematics and Code

All schematics and code can be found at http://www.eng.utah.edu/~dkhoury/.