

Digital VLSI Slot Machine Project Documentation

Christopher Condrat, Heather Malko
*Department of Electrical and Computer Engineering;
 Department of Computer Science
 University of Utah, Salt Lake City, UT 84112*

Abstract—We present a design for a casino slot machine implemented as a digital VLSI chip. The design represents fully functional slot machine: user input is accepted, the reel positions are randomly computed and animated, and rewards are provided according to certain conditions. This paper also covers the testing performed on the physical chip that was fabricated from the design described in this paper, as well as the problems associated with it.

Index Terms—VLSI, Verilog, Slot Machine, graphics

I. INTRODUCTION

THIS project involves the design of a digital VLSI chip that operates as a slot machine. Slot machines are relatively simple state machines: they accept input (which includes money), generate random numbers (reel positions) and reward money based off of that number (payout). Commercial slot machines are nearly all controlled by computer chips; mechanical features like spinning reels and pull-handles exist solely for entertainment value.

At the core of slot machines the main computer chips are responsible for forcing the machines to abide by a certain “payout,” that is, the long-term percentage of wins to losses, nearly always less than 100%, but by law, usually greater than 90%. The concept of payout-percentage is a combination of winning-combination odds and the amount paid out by a winning-combination. Payout computations are the most basic part of any slot machine, and what make nearly all slot machines act the same with few exceptions. For reasons discussed later, our slot machine provides greater-than-100% payout, which means it would lose its owner money over time and therefore be a very poor slot machine for commercial purposes.

What makes a slot machine distinctive is its entertainment value, which comes in the form of graphics and sounds. The entertainment value a slot machine provides is what keeps a player playing by feeding the machine more money. With the payout computations being relatively consistent across most slot machine designs, most of the design and engineering of slot machines is in the slot machine’s ability to provide entertainment to the user. This project is no less different: most of the coding went into generating, rendering and animating graphics.

Our project’s slot machine, with few exceptions, provides the same action as a real slot machine, including a relatively high level of visual graphics and details for a single chip design. We shall show that, while the goals of this project were ambitious, the design meets those goals and has resulted in a quality implementation.

II. PROJECT DESIGN

The key design effort for this project is centered upon capturing the experience of a real slot machine. The most important aspect of any slot machine is that it, over time, loses the player money. To that end, our slot machine performs poorly: it awards more money than it consumes. This is not a real issue however, because the player can add infinite amounts of virtual money to the slot machine to play with. Despite this design “flaw” the slot machine is designed to act like a real slot machine in other ways. The project was broken down into a series of key goals for what our slot machine would accomplish:

Color graphics, pre-rendered – Key to the slot machine is that it should render a full slot machine to the screen, and in color. This chip controls no mechanical reels or other devices; the chip renders all the graphics. The graphics are rendered in 6-bits-per-pixel color (2-bits per channel), allowing for 64 colors, and also making this the only class project to render colors to the screen. Any board containing our chip and wanting to display the colors will need a pair of resistors to be attached to the 2-bits-per-color-channel red, blue and green outputs; information on these resistor specifications will be given later. The graphics are pre-rendered, and stored on chip, or on an attached EEPROM or other memory device.

Animation – A simple slot machine implementation would either just show the resulting reel values and award money, or cycle through the reel symbols one-by-one. This implementation takes the slot machine realism further by actually animating the symbols spinning on the reels. In addition, the reels are designed to reveal their values in order, left to right, in a timed fashion to extend the animation and emulate the real reel action that takes place on a slot machine.

Rule-based Payout – The slot machine still abides by rules-based payouts, for instance some symbols, the jackpot symbol

and the cherries symbol provide winnings for simply appearing anywhere with any other combination of symbols. This is in addition to different three-of-a-kind awards.

Money management – When the user wins, the money is placed into a winnings counter. If the user should decide to play again, the money should be subtracted from the winnings before the player needs to add new money.

These four goals are the driving force behind the slot machine. With these in mind some design decisions were made: the overall operation of the slot machine, the visual design of the slot machine, and the needed modules to implement the slot machine. The overall design of the slot machine, in terms of modules can be seen in Fig. 1.

Now we shall go over the different components that make up the slot machine:

A. Slot Machine Control

The slot machine’s operation is best described by the logic

machine is a finite state machine with 8 states. A basic state diagram for the control logic can be seen in Fig 2.

There are two inputs: add coin and spin. The control module holds two values, the current winnings and the value of the bet. When the add coin button is pressed, it adds one coin to the bet. A coin is subtracted from the winnings if there is any money present, or else it is assumed to have come from the player. The control only allows for three coins to be bet per spin.

When the spin button is pressed, the control sends a lock signal to a random number generator module, which latches its value, and a reel animate signal to the video controls. The control stays in a spin state until the done signal is received, meaning the reels are finished animating on the screen. The next step is to calculate the winnings. If a winning spin occurs, the amount won is added to the winnings and the control returns to the beginning and waits for a coin to be added.

There are a total of 8 symbols for each reel. The random

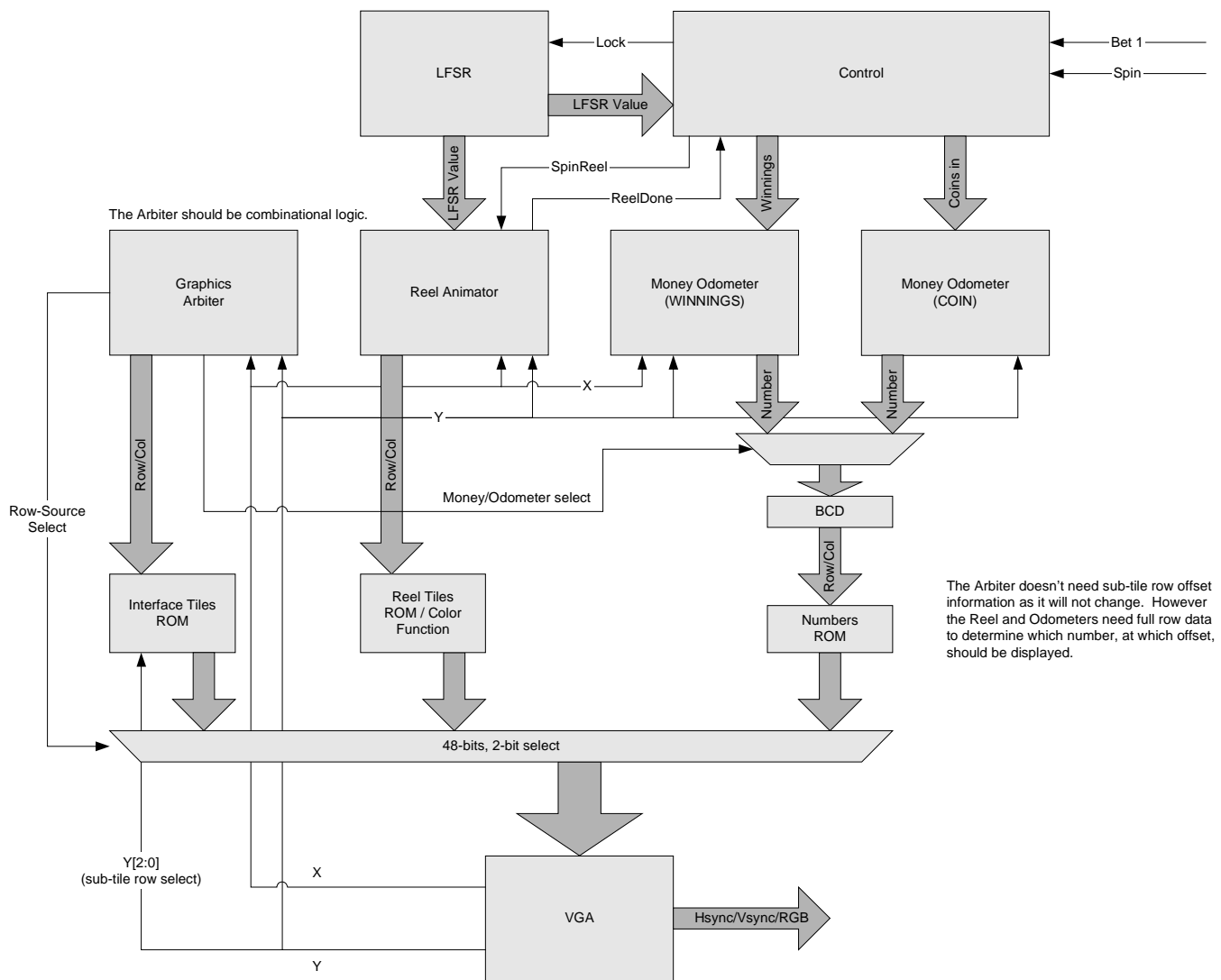


Fig. 1. A top-level block-diagram of the chip.

that controls the play action. The control module for the slot machine has 9 bits, allowing each one of the three reels uses 3

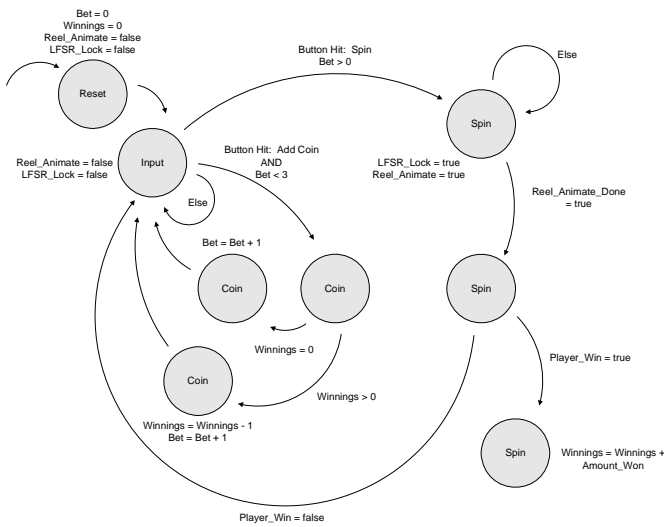


Fig. 2. The state diagram for the control logic.

of the bits from the random number. There is a payout for any 3 of a kind, and for any spin that has a jackpot or a cherry symbol. Every possible combination will occur 1 out of every 512 spins, except for the triple-jackpot, which occurs once out of every 1,024 spins. With the current payout, 59% of the spins will be a winning spin. Over time, the slot machine has a payout of 247%—a very bad thing for a commercial slot machine, but fun for the player. The table in Fig 3 shows the winnings and the odds for the different combinations.

B. Graphics

Graphics establish the look and feel of the slot machine. Great attention was paid to ensuring that the slot machine would be able to display detailed graphics to the screen. Memory restrictions were a deciding factor in determining the detail and resolution of graphics the slot machine would render, and also the screen resolution. Subsequently, the chosen resolution was 128x120, centered on a 640x480 display, meaning that all pixels needed to be quadrupled in size (2x2).

All graphics were manually drawn on a computer. The restriction on the number of colors means that graphics can only be drawn using 64-colors, or in other words, the high two bits of each color channel. Despite this limitation, the graphics are relatively high quality, as depicted in Fig 4.

Displaying graphics to the screen involves many modules working independently to determine what is drawn to the screen. The graphic system is based upon rendering “tiles”—8x8 pixel graphics—repeatedly across the screen using a tile-mapper. This mapping module is named the *graphics arbiter*, and is named such because in addition to providing the basic tile mapping for the slot machine interface, it is charged with the task of arbitrating which modules will render to the screen.

When a module is allowed to decide what is displayed to the screen, it provides row and column information to ROMs or graphic functions. These in turn pass on pixel information, as a full 8-pixel tile-row of 6-bit pixels (48-bits total) to a multiplexer, the output of which is selected by the graphics

arbiter. The output of this large multiplexer is then passed onto the VGA controller, which renders the graphics to the screen using VGA signals. The relationship of mapped data in the graphic system can be seen in Fig 5.

Finally, the VGA controller is the heart of the graphic system, and yet has the most mundane role. The controller merely provides screen coordinates to the other modules, while dumbly rendering the pixel information from the 48-bit tile row to the outside VGA connector.

C. Reel Animation

Slot machines attract users through lights, sounds, and also movement. A slot machine would be boring without reels to spin and keep the user in anticipation. The reel animator module controls reel animation. Animation comes in the form of vertical scrolling of reels, much like the mechanical versions. This requires a vertical offset to be computed for each of the reels. Each reel rotates independently of the others; however, all reels rotate at the same speed. The reel results are revealed when the reels stop. To make the slot machine animation more attractive and coordinated, reels are stopped from left-to-right, and money is only awarded (or taken) after all the reels have fully revealed.

Animation is timed upon successive frames (full screens) of animation, and therefore the reel animator only changes the reel-offset at the beginning of a frame. As a VGA screen is refreshed 60 times per second, the reel spins at 8 pixels per frame, rendering nearly a full reel rotation per second. Each reel spins a minimum of four times before revealing its symbol, extending the play action.

D. Money Odometers and BCD

The money odometers render numerical values to the screen representing both the coin wager, and the user winnings. Internally, bit vectors represent both the winnings and the coins in the machine; however, a BCD was designed to display the current winnings in base-10, to provide the player with a

Winning Combos	Odds	Winning s	Winnings per Spin
3-jackpots	0.0977%	500	0.49
3-cherries	0.1953%	200	0.39
3-3xbars	0.1953%	100	0.20
3-bars	0.1953%	50	0.10
3-\$	0.1953%	40	0.08
3-777	0.1953%	30	0.06
3-7	0.1953%	20	0.04
3-lemons	0.1953%	10	0.02
2-jackpots & 1-cherry	0.5859%	5	0.03
1-jackpot & 2-cherries	0.5859%	4	0.02
2-jackpots	3.5156%	4	0.14
2-cherries	3.5156%	2	0.07
1-jackpot & 1-cherry	7.0313%	3	0.21
1-jackpot	21.0938%	2	0.42
1-cherry	21.0938%	1	0.21

Fig 3 Winnings and odds computations



Fig. 4. A picture of the slot machine in operation.

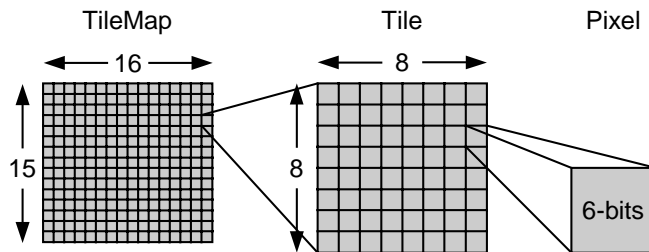


Fig. 5. The relationship of mapped data for rendering graphics.

more familiar counting scheme. The player can win up to $2^{16} - 1 = 65,536$ coins before the counter rolls over; we assumed the player would leave before that happens. Animating the money odometers would be an area of further work, and with the addition of a cash-out button (to make the odometer drop to zero), help our slot machine emulate real ones even better.

E. Graphics ROMs

The graphic ROMs contain the graphics to be rendered to the screen. They accept a row and column as input and return a full row of pixels for a tile (48-pixels) as output. For the reel ROM, the graphics turned out to be so large that it could not be held on-chip (this status is still pending), so in its place, as a potentially temporary solution, a color-generating function was added in place of a ROM. This saved a lot of room in the final design.

An alternative to an on-chip ROM is the use of an EEPROM. EEPROMs act like static RAMs for reading, allowing for relatively large amounts of data to be stored off-chip. Our EEPROM design turned out to be nearly as large as the reel ROMs, because its last-minute design used a lot of flip-flops as a buffer, and so was removed in lieu of a color-generating function. We shall see if the reel ROMs or the EEPROM controller will find their way into the fabricated design; for now a color function is used in place of the reel ROM.

F. Final Details

First, a shading module was added in between the reel ROM output and the VGA controller. This shading module shades the top and bottom of the reel graphics, by subtracting color, to give the appearance of a rounded shape. The effect is not as good as it could be because of the low number of colors, but it is still observable.

As an area of future work, multi-frame animated tiles could be added to the interface by assigning certain tiles to cycle through tiles. Such animations could be used to render blinking lights or switch the state of a winner indicator or such.

III. DESIGN APPROACH AND IMPLEMENTATION

The design approach for this project, in terms of actual chip design is centered on independent module operation; that is, the various modules that make up the entire chip should be able to be implemented without depending on other modules. This approach allows different parts of a chip to be designed in parallel, and therefore speeds up the process of implementing chips.

Another important aspect of our design approach is that we used other programming languages, in this case Matlab, to generate code for our project. A lot of the data, notably graphics, needed to be converted from raw form (bitmaps) into Verilog code, which could then be added to the chip. The use of code-generating programs also ensures that errors can be fixed on a global basis, instead of requiring a group member to hand-edit mistakes out of the individual blocks. Copies of our Matlab scripts for generating Verilog code can be found in the accompanying reference material.

A. Simulation and Verification

Slot machines are not inherently complex; they are basically a mechanized implementation of a dice-throwing game with wagers and winnings. The mechanics of a slot machine's control code can be functionally simulated in Verilog as it is merely a state-machine. Slot machines are not, however, merely made of control logic; they are also charged with entertaining the user. The "action" rendered by a slot machine is measured in human-countable time, making functional Verilog-simulations slow at best, and meaningless at worst. In our tests, rendering a single frame of graphics, using test-benches on the Icarus Verilog[1] simulator would take minutes even on very fast machines. Animations were to be measured in multiple seconds of real time, and with 60 frames per second, simulation using only Verilog proved to be a very time-consuming process. A better solution was needed.

The solution to the simulation problem came in the form of using an FPGA, a 100,000-gate Xilinx Spartan II in our case, as the “chip.” Verilog is compiled using Xilinx’s ISE FPGA compiler, and the resulting bitmap is loaded onto an FPGA. Due to space constraints, some graphics, notably the reel ROMs, needed to be removed and/or swapped out; however the resulting FPGA implementation could still provide the necessary rendition of the slot machine to verify its functionality.

The real-time simulations also enabled our group to make adjustments to the code itself. For instance, the reels should stop (and therefore reveal the symbol it lands on) starting with the left reel and ending with the right. Some combinations of symbols, due to the relative order of the symbols, would cause the reels to be rendered out of order. The solution to this problem: force two full-reel-rotations between successive reel-stops. This extends the animation, but ensures that the reels reveal themselves in order.

IV. INTEGRATION INTO A SYSTEM

The chip requires a 50Mhz clock to operate. Other clock speeds will affect the VGA controller in adverse ways and cause animation timing-problems as well.

The slot machine chip requires outside components to operate correctly. Depending on the implementation, the chip may need access to an external EEPROM for reel graphics data, or may stand alone without the EEPROM if enough chip space is available for fabrication, or if the internal color function (no symbol graphics) is used instead of reel graphics. The slot machine does, however, depend on resistor ladders to produce color output to the screen. Each 2-bit color channel can provide four combinations of output, and by passing these outputs through resistors (470Ω and 1000Ω), variable voltages can be sent out the VGA port, and therefore variable levels of color output. These, along with the synchronizing signals allow the chip to render graphics to the VGA port. Amplifiers for the signals may be required, but this requirement has not been verified.

For input, pre-debounced push-buttons should be used for

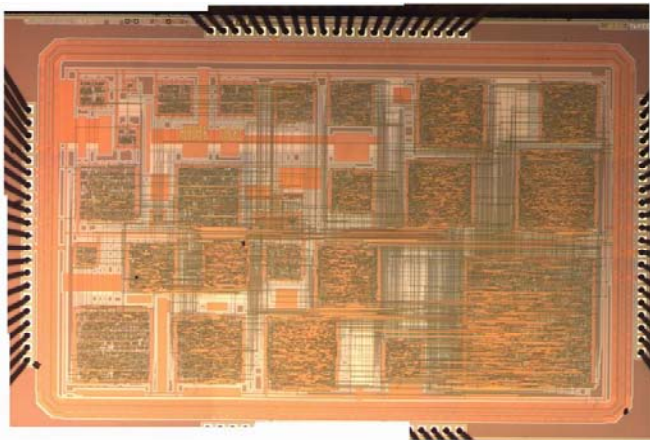


Fig. 7. Photo of the fabricated chip.

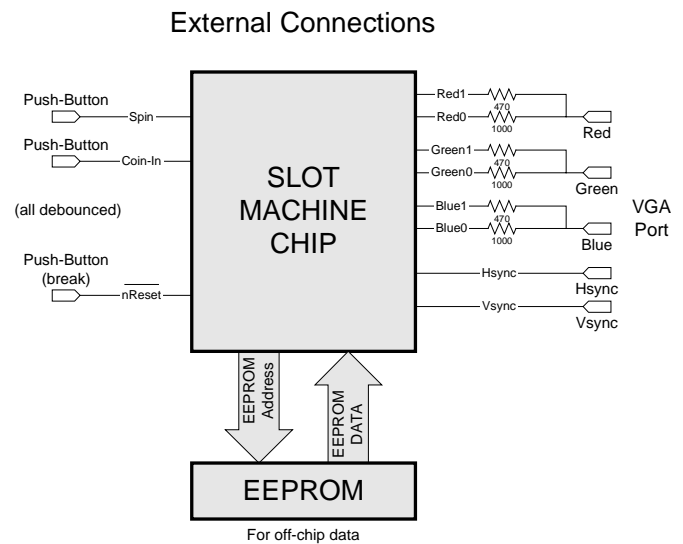


Fig. 6. How the slot machine chip would integrate into a system.

the Spin and Coin-in signals. A break-based push-button should be used for the reset signal, as it is active-low. Please consult the Fig 6 for further details.

V. FABRICATED CHIP

The design was fabricated by MOSIS circuit-fabrication service [2], which provides free design fabrication for educational purposes. This free fabrication service has two purposes: to enable students to fabricate designs and receive a physical chip as a result, and more importantly, for MOSIS, helps provide “realistic” and varied designs to help gauge the quality of the fabrication process for MOSIS. The latter purpose requires that students take a testing course to test and benchmark their chips—both for educational value, and to provide data to MOSIS on how well their fabrication operates.

Fabricated designs vary in size, and are measured in *tiny processor units* (TPUs). Arbitrary numbers of TPUs are not allowed, and designs must fit into sizes of one, two, four, and—for the first time ever in an educational fabrication run—six TPUs. Because of the massive amount of graphics data in the slot machine design, strings were pulled to allow a 6-TPU processor design to be sent to MOSIS—without a reduction of total chips returned. This also meant that no EEPROM had to be incorporated in an extra-chip design to store the graphics data.

Unfortunately, as there was no previous instance of a 6-TPU design, the pad-ring required extra modification to work with the design. Additional metal had to be laid down into the design to fit the fabrication rules as well.

The resulting fabricated chip can be seen in Fig. 7. A micro-fabbing microscope was used to photograph various parts of the chip at high resolution. The images were then stitched together using Adobe Photoshop and color-matched to provide consistent lighting—as if the chip was photographed as a whole.

VI. CHIP TESTING

The chip was tested using an ancient Tektronix LV-500 machine, built in 1989. This archaic relic from the personal-computer dark ages helps benchmark chip performance under “extreme” conditions. As the MOSIS fabrication technology

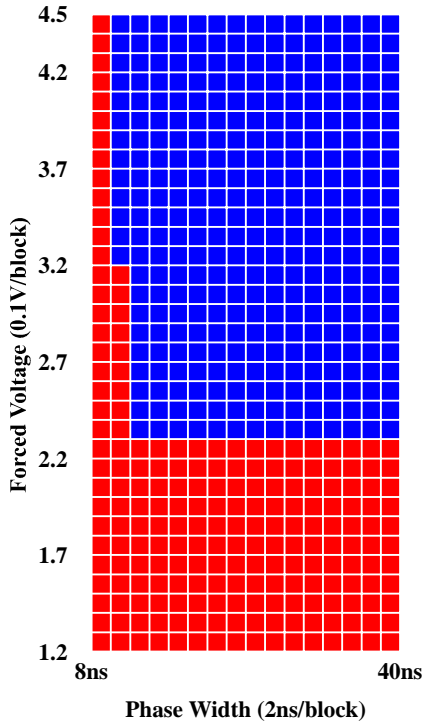


Fig. 8. Schmoo Plot

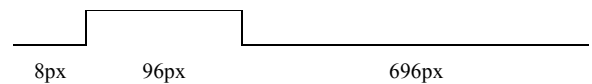
available to students is not especially modern, the chip designs have the opportunity to fail under adverse conditions where the 5V, 0.6-micron logic may not hold up.

Testing is performed using test vectors, for which the LV-500 may store up to 64,000 of. Values are forced, and then tests are made to determine whether the device has returned the correct value. The tester has no ability to automatically generate a clock, so clock signals must be coded manually into test vectors. Test vectors can either be inputted by hand at the terminal, or loaded into the machine via its onboard FTP server, which is only accessible via one or two machines on campus.

In the case of the slot machine design, the output generated by the chip comes in the form of video signals. These signals take some time before any change in the output can be detected. The earliest-arriving, that is earliest-changing, predictable signal is the horizontal synch signal (Hsync), which has a pattern:



Fig. 9. Photo of the FPGA-simulated design.



Each pixel is one clock cycle (the design ended up being clocked down to 25 Mhz for simplicity). The pattern above repeats continuously as the screen is drawn.

After a somewhat frustrating start, where the wrong voltage was applied to the design, the tester started consistently reporting that the tests passed. Next came the schmoo plots.

Schmoo-plots depict two test conditions, for instance chip-voltage and clock-speed, varied on a 2D plot, with one condition representing the horizontal axis, and the other condition representing the vertical axis. A schmoo plot helps visualize the conditions under which the chip will operate correctly.

The varying-conditions used in the testing of the fabricated chip were voltage and clock pulse-width. There were some issues involving untestable voltage values, specifically the device voltage, which would not go below the forced voltage. We were unable to find a workable solution to this issue, so instead of varying the device voltage, we varied the forcing voltage versus the phase width and plotted the graph. The resulting schmoo-plot can be seen in Fig 8.

After testing the chip using the LV-500, we determined that the MOSIS fabrication run produced a working yield of 100%. This is unsurprising seeing that this was a relatively simple design, and it wasn't really pushing the boundaries of even the

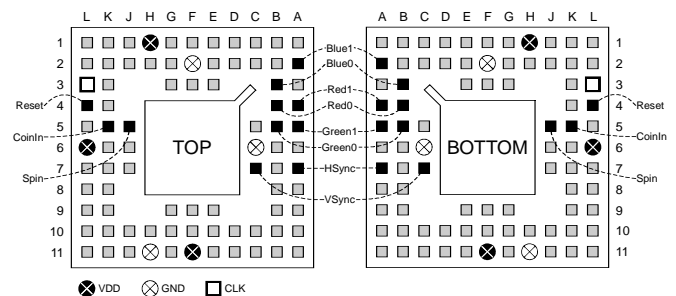


Fig. 10. Wiring diagram.

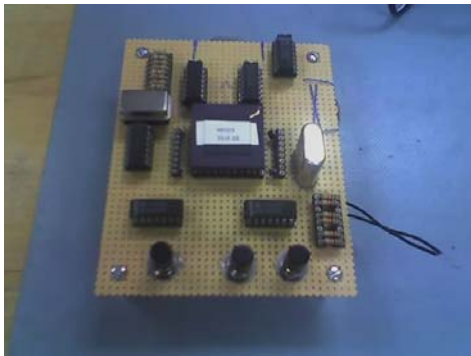


Fig. 11. Implemented Board
 age-old 0.6μ-technology. Unfortunately, we were unable to provide a qualitative assessment of the yield, as will be seen later, as we were only able (or willing) to test the Hsync signal on the LV-500. A better functional test would come in the form of a board to hold and operate the chip, for its original purpose: to play.

VII. BOARD DESIGN

The goal of the project was to produce a working *playable* slot machine, which would produce color graphics on a VGA monitor. In the design phase of the project, we were able to simulate the design on an FPGA, albeit with a limited set of graphics because of the lack of slice-resources available. With the full 6-TPU design, we were able to incorporate all the graphics that were drawn for the design, including all reel symbols, which needed to be omitted from the FPGA, in lieu of some functionally generated color “symbols.” A photo of the FPGA simulation in operation can be seen in Fig 9. Needless to say, the ability to see the design working in its full glory was something to look forward to.

The slot machine chip required that a special circuit board be designed and implemented for proper validation and verification of the fabricated chip. The basic game requires that there exists an input clock signal and multiple switches for the following input signals: reset, spin, and coin. The output of the game requires that a VGA monitor be implemented; hence, a VGA adaptor is essential. This circuit board consists

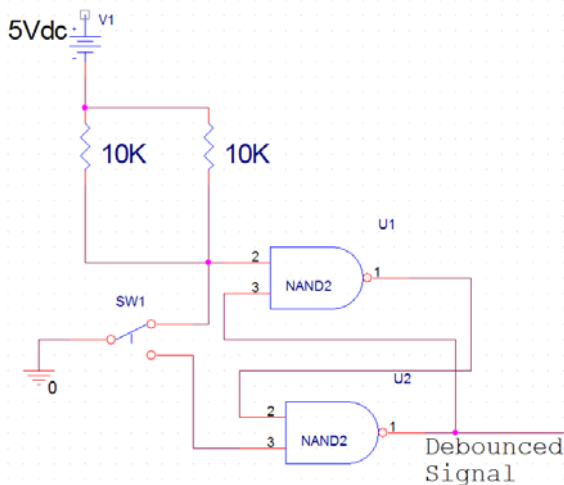


Fig. 12. Pushbutton Debouncing Circuit

of a 5-Volt power supply, a 25 MHz clock, 3 switches and their associated components, and a VGA connector and its associated components.

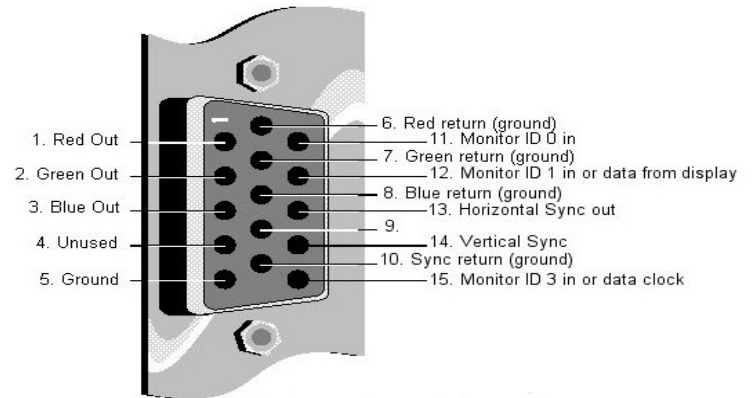


Fig. 13. VGA Connector [3]

This circuit board includes a 5-volt supply for powering and grounding the fabricated slot machine chip, the clock chip, the switches, and multiple debouncers. It was determined that the clock consist of 5Vdc HCMOS Crystal clock oscillator with features such as wide frequency range, tight symmetry option, fast rise and fall times, HCMOS and TTL compatible, tristate enable/disable options, and anti-static packaging tube. Hence

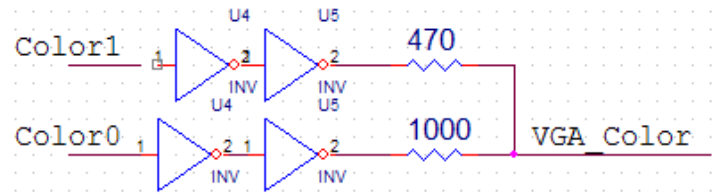


Fig. 14. VGA Buffered Signals

an Abracon xo – crystal clock oscillators with Full Size TTL/HCMOS Clock Oscillator 25MHz was selected. The final input signals were switches for the reset, spin, and coin signals. It was determined that a pushbutton debouncing circuit is necessary for these input signals for the chip. Figure 12 represents this configuration.

For the output of the vga monitor, it is necessary to examine the vga connector

It is essential that the circuit board buffers the outputs of the chip. The buffer is a single-input device, which has a gain of 1, mirroring the input at the output. It has value for impedance matching and for isolation of the input and output. Furthermore, it is essential that the Red, Green, and Blue have this configuration.

VIII. RESULTS & CONCLUSION

Our team has successfully implemented a working slot machine as a digital chip. We are happy to acknowledge that we have achieved the aims of the project with a high level of standards. Additional work could have be done on the design to make the slot machine even better; however, time, notably, space constraints dictated otherwise. Presented below are the

results of implemented circuit board and further conclusions.

The fabricated chip worked when tested, but we have not been able to make it work correctly in its functional environment. After implementing the circuit board presented above, the result was a blank monitor screen. Hence, the potential problems are improper implementation of the above design, clock issues, or an unsuccessful fabricated chip.

The first potential problem is a design flaw or a poor implementation of this design. It seems unlikely that there is a design flaw for switches since an Agilent 54622D Mixed Signal Oscilloscope was utilized for testing purposes. The simulated results match preliminary results; i.e. a switch switches properly with the bouncing circuit. Next a multi-meter seemed to vary that wire wrapped signals were properly connected and matched the design schematic. Hence, it is possible that the design was implemented poorly. We will further explore this issue.

The second potential problem is a fault clock. First, the design utilizes an Abracon XO – crystal clock oscillators with Full Size TTL/HCMOS Clock Oscillator 25MHz. Based on the Agilent 54622D Mixed Signal Oscilloscope, there is a successful 25 MHz Clock. However, it possible that the fabricated chip will not operate successfully with this component. Hence, an Agilent Function Generator was employed to emulate a clock. Based on the results, values for Hsync and Vsync were consistently at 25mV DC. Yet, it is possible that errors occurred when implementing this configuration and we plan on exploring this issue.

The third potential problem was a faulty fabricated chip. This could potentially mean that a layout design occurred or failed block layout occurred. The memory allocation resources at the University of Utah limited the amount of analog simulations. Hence the FPGA was heavily used as a test for analog simulation. Another potential problem associated with this conclusion is that the chip was not fabricated correctly. This seems highly unlikely since MOSIS is well known for their products. This final conclusion does not seem likely since the chips produced good schmo plots.

We hope to find the error of our poor simulation results and be finally able to play the slot machine in its full colorized glory. This project has been an amazing learning experience. The opportunity to design, fabricate this slot chip, and finally implement an associated circuit board was incredibly exciting and satisfying. We have form the foundation of chip design and hopefully we will use these skills in our potential future chip designs.

REFERENCES

- [1] Icarus Verilog simulator and compiler.
<http://www.icarus.com/eda/verilog/>
 - [2] The MOSIS Service
<http://www.mosis.org/>
 - [3] VGA Connector
http://www.theavguide.co.uk/AdvHTML_Upload/vgaplug3.jpg
-