

ECE363 - Project 3

Analysis of Design

Artur Poole

April 24, 2024



LEHIGH
UNIVERSITY

Contents

Executive Summary	1
1 Functional Verification	3
1.1 Functional Verification of Synthesized Design	4
1.2 Comparison of Functional Verification	4
2 Formal Verification	6
2.1 Results	6
3 Revisions	8
3.1 Understanding and Revising the Design	8
3.2 Functional Verification of Edited Original Design	9
3.3 Functional Verification of new Synthesized Design	9
3.4 Formal Verification of Revised Designs	10
Conclusion	13

Intro

The objective of this project is to gain an understanding of functional and formal verification of synthesized designs. In previous projects, a game FSM was created to manage the states of a game wherein a "player" can move through different locations to find gold or meet their maker. In the second project, this design was synthesized using Design Compiler and the Nangate OpenCellLibrary to optimize the original design using a series of constraints.

In this project, the synthesized design is functionally verified, wherein the synthesized gameFSM, under the same testing conditions, will operate functionally the same as the original design. There can be differences, as expected, due to factors such as output and input delays that constrained the original design; however, as long as the main functionalities, such as winning and losing, are produced correctly, other differences can be ignored. This report will examine a confirmation of the original design, followed by functional verification of the synthesized design, which will then be compared to the original functional verification. Lastly, the two designs will undergo formal verification, and if necessary, revisions to the original design to accommodate the synthesis process will be functionally verified against the original design.

1 Functional Verification

Firstly, conducting functional verification of the original design is essential to understand what to expect for the synthesized design. While only the game FSM (Finite-State-Machine) file is synthesized, three other files are necessary for the SoC (System-on-Chip) design implemented in this project. These files include the harness, responsible for holding the game FSM, interface, and testbench. The interface enables the testbench to 'send and receive' data from the gameFSM DUT (Design Under Test). The same three files will be used in simulating and testing both the original and synthesized game FSM files.

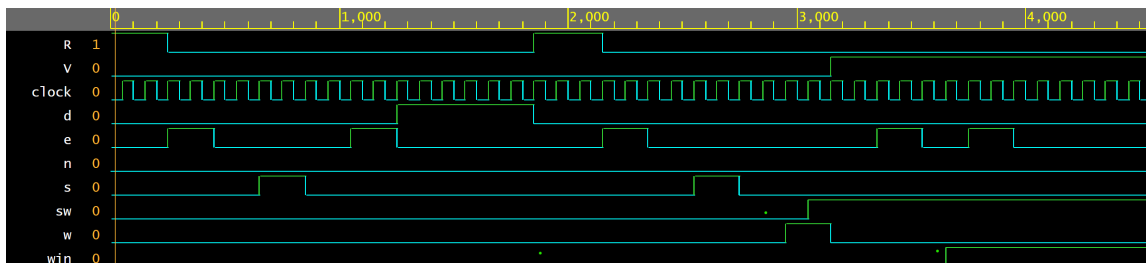


Figure 1.1: Waveform of Original Design

The testbench used tests the death and win paths. As observed in the image above, the testbench leads to the death state where 'd == 1'. This is confirmed in both the testbench code and game FSM, where moving east, south, and east will result in not having the vorpal sword when reaching the dragon's den. Next, the reset is set to high, and again, it is observed that all the signals 'd' and 'win' are both set to low or zero. The testbench then tests the win path by having the 'player' traverse to the sword stash before proceeding to the dragon's den. At the end of the image, it is observed that 'win' is set to high after traveling east, south, west, east, and east. The functional verification for the original design is accurate

to the expected results detailed in the design specification of the game FSM. Other than that, the red bars, as discussed in the intro, can occur and are not a problem unless the outputs received in the testbench are not as expected.

1.1 Functional Verification of Synthesized Design

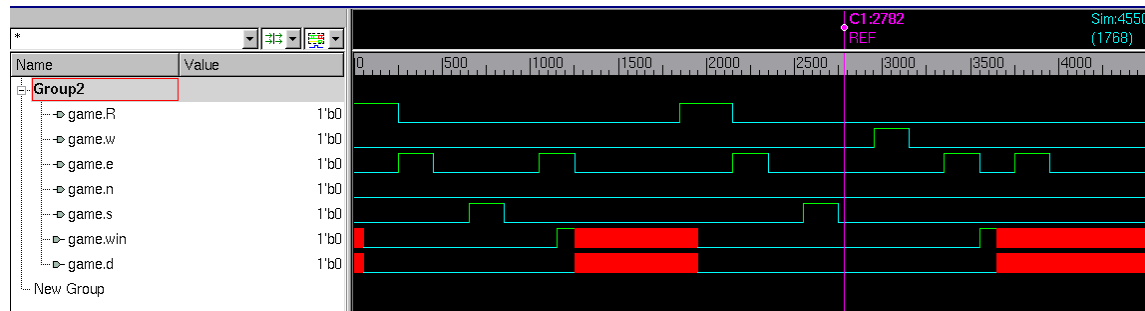


Figure 1.2: Waveform of Synthesized Design

It's evident from the image above that the synthesized design is not functioning as expected. In the first of the test cases, the death path, when the player reaches the dragon's den without the sword, the FSM outputs 'win' as high instead of 'd'. While it's clear what the issue is, it is hypothesized that the delays and other constraints may have caused a problem with the 'V' or sword output, indicating that the 'player' has the sword. However, the expected result would be the opposite, where the win state is impossible to achieve because 'V' is never HIGH or 1. Upon attempting examination of the schematic and the 'V' wire into the room FSM, the results showed a value of 'x' along with a red bar across the waveform.

1.2 Comparison of Functional Verification

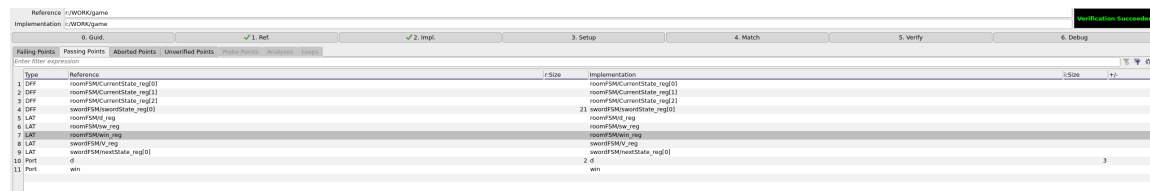
In the regular design, the outputs into the terminal produced the expected results; however, in the synthesized design, the outputs showed 'x' for both 'win' and 'd'. For 'd', the value observed is either 0 or 'x'; however, for 'win', the red bar appearing after a short time is disrupting the testbench results, although the waveform shows it to be more accurate. Overall, the new design is unsuccessful for the death path and untrustworthy for the win path, although it appears to be functioning correctly in the waveform. Also, more noticeable are the misalignment and timing

issues in the original and synthesized versions. Notice how the game.win wave ticks high at the negative edge of the 2nd to last east control. Due to timing issues in the original design, the game FSM is getting two clock cycles for the east command and thus going east twice in quick succession, hence why 'win' goes high before the testbench tells it to travel to the den where it is automatically switched to the win state. This can be fixed by changing the always blocks in the Verilog code. For this project, it meant switching the roomFSM state manager and win, d, and other output managers to run on the positive edge of the clock.

2 Formal Verification

Formal verification is conducted using Synopsys Formality, which checks that the reference (old design) and the implementation (synthesized) are functionally the same. Unlike the manual functional verification done in earlier sections, this program automatically matches components and outputs together, verifying that they behave in the same manner. This provides more information and insight into the revision process of the designs. Whereas for the synthesized design, it was difficult to determine what exactly caused the observed error, Formality can utilize matching to pinpoint where designs differ in physical and functional implementation.

2.1 Results



Type	Reference	Implementation	Size
1 DFF	roomFSMCurrentState_reg[0]	roomFSMCurrentState_reg[0]	
2 DFF	roomFSMCurrentState_reg[1]	roomFSMCurrentState_reg[1]	
3 DFF	roomFSMCurrentState_reg[2]	roomFSMCurrentState_reg[2]	
4 DFF	swordFSMCurrentState_reg[0]	swordFSMCurrentState_reg[0]	21
5 LUT	roomFSMNextReg	roomFSMNextReg	
6 LUT	roomFSMNextReg	roomFSMNextReg	
7 LUT	roomFSMNextReg	roomFSMNextReg	
8 LUT	swordFSMNextReg	swordFSMNextReg	
9 LUT	swordFSMNextReg	swordFSMNextReg	
10 Port	0	0	2
11 Port	win	win	3

Figure 2.1: Passing Points - Formality

Every point passed; however, there were two unmatched points: 'swordFSM/nextState_reg[1]' and 'swordFSM/swordState_reg[1]'. This unmatched point is mysterious, as 'nextState' and 'swordState' are represented by a single bit in Verilog but 2 bits according to the unmatched points. On the contrary, while 'room-

State' is represented by 3 bits in both the Verilog and Formality. However, 'roomFSM/nextState' is only 0-3. This could explain an error occurring with the functional verification where, when the player reaches the dragon's den without getting the sword first, they are still able to win the game. It was observed that the 'V' variable in DVE was 'x' or red on the waveform viewer.

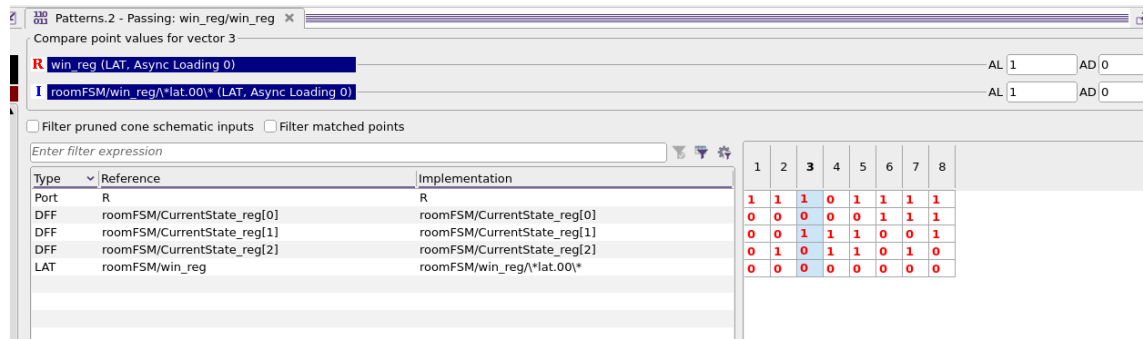


Figure 2.2: Win Point Comparison

Despite the two unmatched points, the verification was 100% successful and reported that the 11 matched points had all been verified. The image above is the pattern viewer of the win register for the reference and implementation designs. Despite it appearing as red, this confirms the expected results, as the state register is able to traverse the states, and 'win' is never set to 1. This pattern doesn't take into account other inputs such as V where the win_reg should output 1 in state 6 if V was HIGH. As we confirmed, the 'win' output works in the synthesized design but not the death output.

3 Revisions

Due to problems with the synthesized version, specifically the 'win' and 'd' outputs outputting 'X', the original file needs revisions to synthesize properly. To expedite the synthesis process, the constraints described in the previous report have been compiled into a single file, 'synthesize.tcl'. This allows for a single command in design vision, 'source synthesize.tcl', to execute and implement all the constraints instantly. All that's left to do afterwards is compile and save the various files, such as the synthesized Verilog file and the SDF.

3.1 Understanding and Revising the Design

At first glance, one might suspect an issue related to 'win' and 'd', since their outputs are red or 'X'. However, further examination of the schematic and waveforms of internal variables, such as 'V', led to the conclusion that the sword FSM was the module that produced errors. As described in the Verilog manual, 'X' or red bars may indicate an issue with continuous assignment. The FSMs developed use continuous assignment, wherein whatever input they are currently receiving will affect their instantaneous output; there is no impulse or other behavior for inputs and outputs.

The first change made to the original design file was to set any output 'variables', not states, using non-blocking assignments instead of blocking assignments. While this had no effect, it resolved about 20 warnings from the synthesis and compile functions. Since this didn't fix the issue, a realization dawned that the actual formatting may be different. While the original design worked as intended in EDAPlayground and in Visual Compiler, once synthesized, it did not.

The fix ended up being, instead of setting the output of the sword FSM, 'V', from '1'd0' and '1'd1' to '0' and '1' respectively. In Verilog, setting 'V <= 1'd0' changes

its representation to a single-bit, while 'V = 1' sets it to a register of any bit width but dynamically becomes 1 bit. In synthesis, '1d0' and '1'd1' representation gets distorted and causes issues. After ensuring the 'V' outputs were set to '0' or '1', alongside 'win' and 'd' in room FSM, the waveform performed as expected. The final change was to adjust the room FSM timing so that the room FSM would only operate on the posedge of the clock, rather than the reset/transition operating on the positive edge of the clock and the logical always block running constantly.

3.2 Functional Verification of Edited Original Design

As many changes, some unnecessary, were made to the original design for the synthesized version to work properly, the original design needs to be functionally verified again. Additionally, the print statements in the testbench confirmed that win and die states for the win and die paths worked properly.

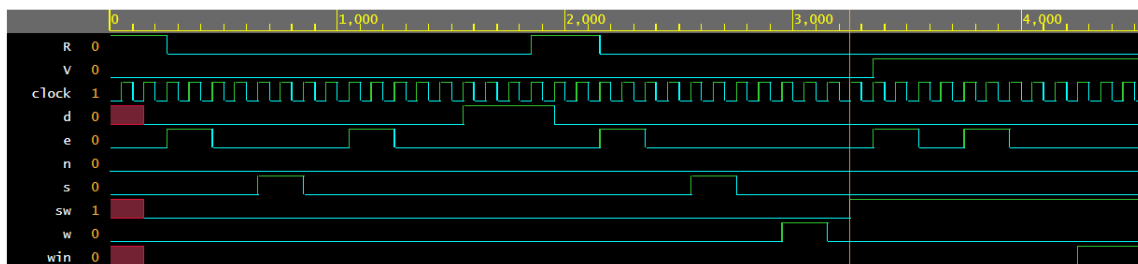


Figure 3.1: Waveform of Modified Original Design

The first 'path' is a reset where win and 'd' are expected to be zero. The second path leads the player to its death and expects an output of 1 for 'd' and 0 for 'win'. The third path is the win test path.

It's clear from the image above that, while the code has changed, there have been no changes in functionality. This design is then synthesized quickly using the .tcl file explained earlier and then compiled using VCS and the NangateOpenCell-Library to be functionally verified and compared.

3.3 Functional Verification of new Synthesized Design

Using the waveform viewer in Synopsis Design Vision Environment, its observed that the device behave functionally exactly the same, and the constraints imposed

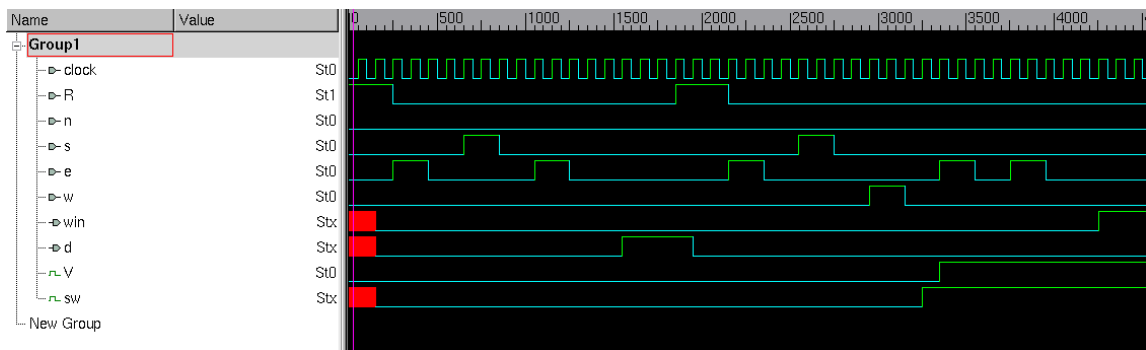


Figure 3.2: Waveform of Modified Synthesized Design

on the synthesized design do not harm the modules functionality. Upon closer examination there were no observed delay or timing differences of when certain outputs were set to high despite delay constraints on the synthesized design.

```

VCS Simulation Report
Time: 4550
CPU Time: 0.430 seconds; Data structure size: 0.1Mb
Tue Apr 23 22:03:57 2024
[awp224@ece-server03 project3_2]$ dve -full64
[awp224@ece-server03 project3_2]$ ./simv
Info: [VCS_SAVE_RESTORE_INFO] ASLR (Address Space Layout Randomization) is detected on the machine. To enable $save functionality, ASLR
will be switched off and simv re-executed.
Please use '-no_save' simv switch to avoid this.
Chronologic VCS simulator copyright 1991-2023
Contains Synopsys proprietary information.
Compiler version U-2023.03_Full64; Runtime version U-2023.03_Full64; Apr 23 22:41 2024
Interface connected
time to reset and start
reset done
d= 0, win=0
d= 1, win=0
d=0, win=1
$finish at simulation time 4550
VCS Simulation Report
Time: 4550
CPU Time: 0.460 seconds; Data structure size: 0.1Mb
Tue Apr 23 22:41:13 2024

```

Figure 3.3: ./simv Output for Synthesized Design (Same for both)

3.4 Formal Verification of Revised Designs

Although the formal verification of the original passed all the matching points, there was one item "swordState_reg[1]" that was due to initializing the swordState as a 2-bit number instead of just a single bit. This was an error that stemmed from a misunderstanding of creating multi-bit registers in verilog. This was fixed simply by defining swordState and nextSwordState as "reg" instead of "reg[1:0]"

However, in addition to the other changes described above, the new design needs to be formally verified against the revised original design.

```

***** Verification Results *****
Verification SUCCEEDED
ATTENTION: synopsys_auto_setup mode was enabled.
See Synopsys Auto Setup Summary for details.
-----
Reference design: r:/WORK/game
Implementation design: i:/WORK/game
10 Passing compare points
-----
Matched Compare Points   BBPin   Loop   BBNet   Cut   Port   DFF   LAT
TOTAL
-----
Passing (equivalent)      0       0       0       0       2       4       4
10
Failing (not equivalent)  0       0       0       0       0       0       0
0
*****
1
150 module SwordFSM( // sw output from reaching the sword stash sta

```

Figure 3.4: Formal Verification

The image shows that the formal verification passed every single point. Furthermore there were no "unmatched" points indicating that the revisions made to the design were successful. The image above is the terminal output of the verification where it expands upon what type of compare point its testing and whether they are equivalent. As observed none are failing.

Conclusion

In conclusion, this project aimed to develop an understanding of functional and formal verification of synthesized designs, specifically two FSMs managing a game. Through analysis and testing, it became evident that while the original design functioned as expected, the synthesized version exhibited discrepancies, particularly in the output signals for win and death paths.

Functional verification of the original design showcased its the specified functionality with 100% accuracy. However, the synthesized design presented challenges, with unexpected outputs observed during testing. Functional verification, combined with the error/warning logs of design vision and formality, allowed for a greater understanding of the synthesis process and how representation affects final synthesized design. Despite initial discrepancies, revisions to the original design were implemented to ensure compatibility with the synthesis process, notably adjusting output assignments to 0s and 1s and using non-blocking assignments for output variables while maintaining them for state changes.

Subsequent functional verification of the revised original design demonstrated equivalence to the original, reaffirming consistent functionality across implementations. In addition, formal verification of the revised design showed all points passing which which represents functional equivalency on a hardware level between the original and synthesized design.

In all, this project showed the role of the verification and iteration processes in ensuring the functionality of synthesized designs. Through a systematic approach encompassing both functional and formal verification, discrepancies were identified, addressed, and resolved, resulting in designs that functionally behave identically.



LEHIGH
UNIVERSITY