ECE 582/682 PROJECT

In this project, you will develop and verify a sequence detector Finite State Machine (FSM) that identifies a specific sequence of binary inputs. This FSM will be used as part of a security system to detect authorized sequences for operation activation. You will utilize Synopsys VC Formal to formally verify the safety, correctness, and robustness of your design.

The VC Formal Apps you must use. (You can use more apps if you want):

VC Formal AEP App: App for Automatically Extracted Properties.
The AEP app will extract properties from your design written in a supported HDL, such as SV, and analyze them automatically to provide results. This is done automatically, and writing properties or assertions is not necessary, but you need to specify what you want to do in the TCL file.

VC Formal FXP: Formal X-Propagation App.
The FXP app will scan your design written in a supported HDL, such as SV, for "don't cares/x's" to observe if they can propagate through the system. The app injects don't cares in designated parts of your design and monitors their propagation. This is done automatically, and writing properties or assertions is not necessary, but you need to specify what you want to do in the TCL file.

VC Formal FCA: Formal Coverage Analyzer.
The FCA App can be utilized to qualify formal property verification with coverage by identifying unreachable goals, either because of constraints or due to uncovered goals in simulation. Formal core coverage reachability analysis can identify the minimum amount of design code required for formal engines to reach the full or specified proof depths. Writing properties or assertions is not necessary, but you need to specify what you want to do in the TCL file.

VC Formal FPV: Formal Property Verification.
The FPV app can verify various design properties, such as assertions to validate design behavior, assumptions to restrict the formal verification environment, and coverage properties to track significant or expected events. Writing properties in SVA (SystemVerilog Assertions) is required for this app.

Problem 1: Specifications for the Sequence Detector FSM:

The sequence to be detected is a 12-bits binary number generated by converting your last (Rightmost) 3 digits of your PSU in a BCD (Binary Coded Decimal) format.

For example, if your ID number is: 905511564, you'll represent 564 as 12 bits using BCD. If you are working in a group of 2, one of the team members ID is sufficient. Please include the complete student ID on your project as well.

1) The FSM should detect your specific 12-bits BCD binary sequence.
2) After successful detection, the FSM should reset to an initial state to search for a new sequence.
3) Include a 'deadlock' state that the FSM enters if a specific incorrect sequence is detected (e.g., "1111"). Once in this state, the FSM should not transition to any other state, simulating a lockout condition for security purposes. It'll be stuck forever. (Even a reset cannot take it back to the initial state).
4) Introduce an 'isolation' state that is entered from the initial state with a specific input (e.g., "0000"). The isolation state represents a maintenance or diagnostic mode that requires a system reset to exit.
5) The FSM will have a negative edge-triggered asynchronous reset that returns to the initial state, ensuring the system can recover from any state, except the deadlock state.
6) Implement simple noise mitigation (Described below) by checking for the consistent presence of the current expected bit in the sequence across multiple clock cycles before transitioning to the next state.

Implementation of the noise mitigation:

To implement simple noise mitigation within your FSM, you should intermediate 'check' states between your main sequence detection states. These 'check' states will confirm the presence of the expected input over **two clock cycles** before allowing the FSM to proceed to the next state in the sequence.

Here's a simplified approach to model this:

Simplified Noise Mitigation in FSM:

1. Additional 'Check' States:

- For each state that is part of the sequence detection, introduce a corresponding 'check' state. The FSM moves to this 'check' state before it transitions to the next main state.

2. Stability Counter:

- In each 'check' state, implement a counter that increments if the expected bit remains the same as the input. If the counter reaches a threshold of 2 (e.g., the bit is stable for 2 consecutive clock cycles), the FSM transitions to the next state in the sequence.

3. Reset on Change:

- If the input changes before the counter reaches the threshold, reset the counter, and return to the previous stable state, depending on your design's needs.

Basically, if you have for example 12 states in the system, you'll introduce 12 more states for checking, resulting in 24 total states.

If your design is identical/too similar to the one used by other students, further investigation will be conducted.

Task 1: FSM Design and Implementation

- 1.1 Create a detailed FSM diagram for the sequence detector FSM, indicating all state transitions, input sequences, and corresponding outputs, including 'check' states for noise mitigation.
- 1.2 Implement the FSM in SystemVerilog. Use separate **always_comb** blocks for state transitions and output logic. Employ a **case** statement with 'x' as the default case for state transitions. Use **unique case** for output logic to ensure deterministic behavior.
- 1.3 Model noise mitigation using intermediate 'check' states as described in the project specifications.

Task 2: Formal Verification with the VC Formal AEP App.

- 2.1 Use the VC Formal AEP app to perform automatic extraction and verification of properties. Refine the FSM design or constraints to address any discrepancies found.
- 2.2 If bugs are found, you must fix them. If they can't be fixed, you must explain why. Make sure to document any bugs you find.

Task 3: Formal Verification with the VC Formal FXP App.

- 3.1. Use the VC Formal FXP app to investigate 'don't care' propagation and ensure it does not affect sequence detection accuracy.
- 3.2 If bugs are found, you must fix them. If they can't be fixed, you must explain why. Make sure to document any bugs you find.

Task 4: Comprehensive Property Verification using VC Formal FPV App.

- 4.1. Write at least 15 SystemVerilog Assertions (SVA) that cover all aspects of the FSM's design, including sequence detection, error handling, maintenance mode entry/exit, deadlock state behavior, and noise mitigation logic. Use the project specifications above (not your HDL code) to write those assertions.
    - Your assertions should thoroughly test the FSM's response to various erroneous sequences to confirm that the system transitions to and behaves correctly in the deadlock state.
    - Verify the FSM's ability to enter, function within, and exit the maintenance state according to the specified conditions, except when in the deadlock state.
- 4.2. Write any SystemVerilog Assumptions (Constraints on the inputs) needed for your design, and document all assumptions made in the design and verification process. These assumptions may include input signal characteristics, reset behavior, or system operating conditions.
- 4.3. Run the VC Formal FPV app to verify all written SVAs, ensuring that each aspect of the FSM's design is functioning as intended.
- 4.4 If bugs are found, you must fix them. If they can't be fixed, you must explain why. Make sure to document any bugs you find.

Task 5: Analyze coverage using the VC Formal FCA App.
5.1. Utilize the VC Formal FCA app to ensure coverage completeness, identifying and addressing any unreachable states or transitions.

Report requirements.

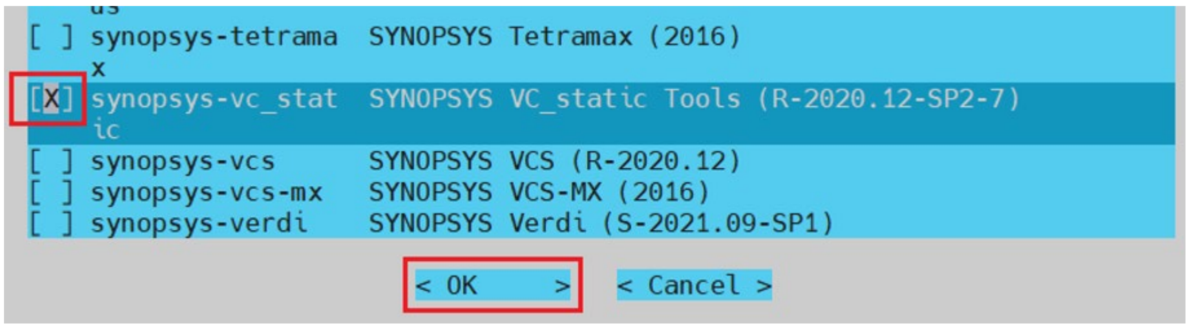
- You must include the students Complete ID numbers on this report.
- Compile detailed report that documents the design, implementation, and verification process, including the state diagram, SystemVerilog code, SVAs, screenshots from all the apps results, and findings from each verification app.
- Present a detailed analysis of any challenges encountered, how they were resolved, and any limitations or potential areas for future improvement.
- Upload 2 files. (PDF or Word document of your report, and a .zip or .rar file of all your files used in this project). Make sure to upload those on Canvas as two separate files, Don't put the pdf or word file inside the .zip file.

Supplementary instructional materials covering the AEP, FXP, FPV, and FCA applications are available on Canvas.

Software setup:
<u>This is a summary of the setup process, but there is an additional tutorial for installing MobaXterm and the PSU VPN if this is your first time installing them and you need more help.</u>

1. You will need to connect to the RedHat system remotely:
mo.ece.pdx.edu. If you are on campus, you can remote access through Ubuntu PC in the Intel lab. If you are off campus, check: https://cat.pdx.edu/services/network/vpn-services/ to install PSU VPN on your system. Then install a remote access software support x-server such as MobaXterm or putty with xming.
2. Login to mo.ece.pdx.edu with your ECE username and password.

3. Run addpkg and select **synopsys-vc_static**, and then re-log in the system.



```
[ ] synopsys-tetrama  SYNOPSYS Tetramax (2016)
    x
[X] synopsys-vc_stat  SYNOPSYS VC_static Tools (R-2020.12-SP2-7)
    ic
[ ] synopsys-vcs      SYNOPSYS VCS (R-2020.12)
[ ] synopsys-vcs-mx   SYNOPSYS VCS-MX (2016)
[ ] synopsys-verdi    SYNOPSYS Verdi (S-2021.09-SP1)

             < OK     >   < Cancel >
```

4. Create a workstation directory for your VC Formal project in your home directory, place the SystemVerilog/Verilog files you want to check in the directory, and run vcf form there. You can also run vcf -gui in GUI mode, so you can just select your files instead of using commands.
5. Read the tutorial to practice some basic commands.

Note:

1. The report should show screenshots of all the result from VC Formal before and after you attempt to fix the falsified properties.

Honor Pledge

On the cover page of your project report, please type/write down the following sentence in the underlined section below and sign/type your name under it.

"On my honor, I have neither given nor received unauthorized aid on this project."

Do NOT copy and paste the sentence, please do type/write every word in this sentence. Failure to write down this sentence and sign your name or failure to honor the content of this sentence will result in 0 points in the project.

Write the above sentence in the underlined section below:

Type or Sign Your Name here: ____________________________