

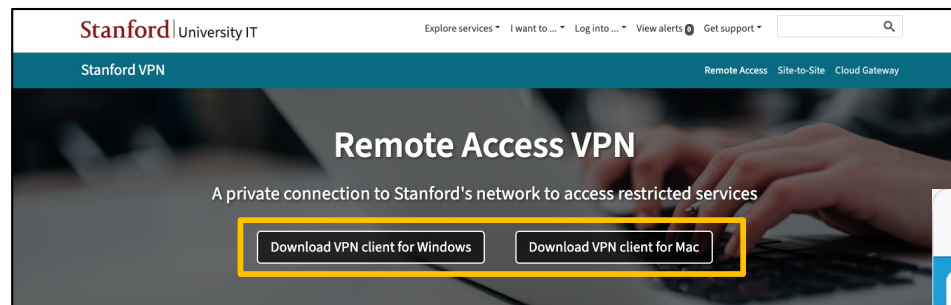


# Server Setup

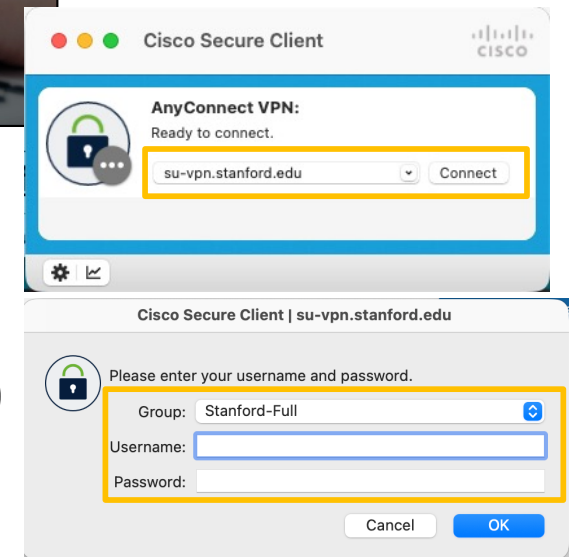
Samantha

# Connect to Stanford VPN

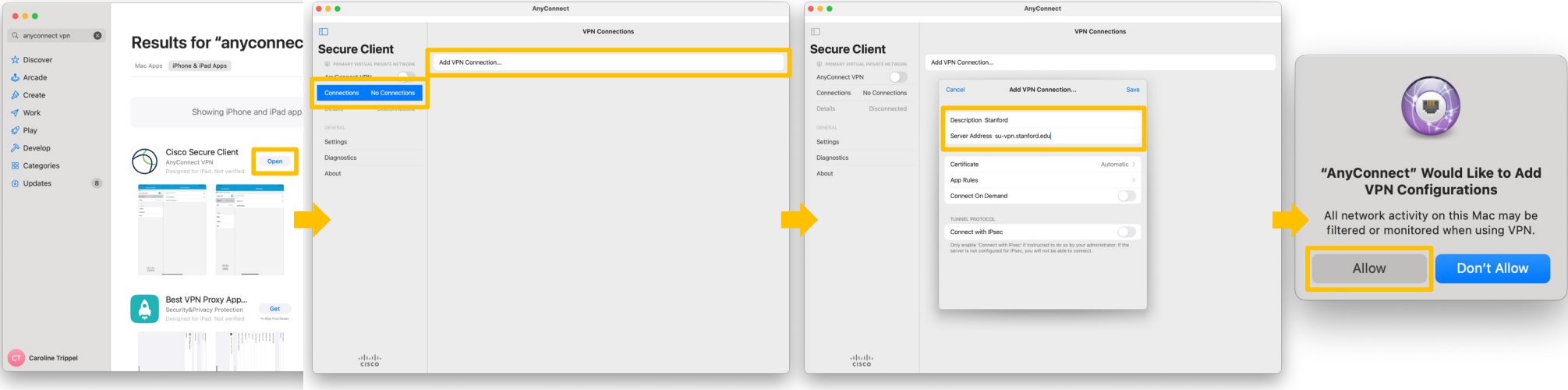
- Download client for Windows or Mac: <https://uit.stanford.edu/service/vpn>



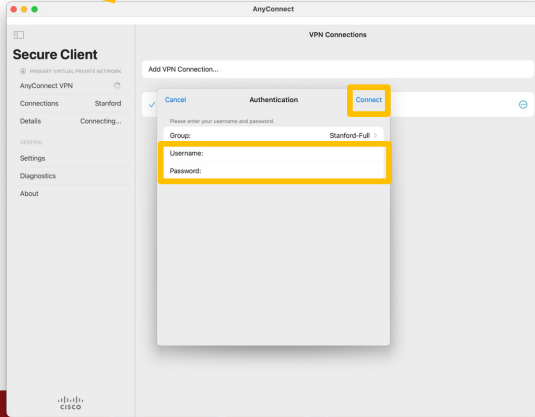
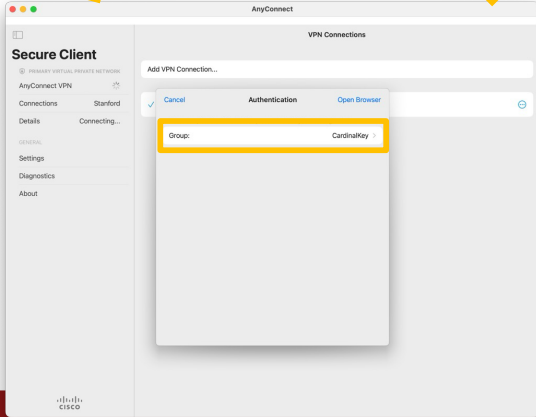
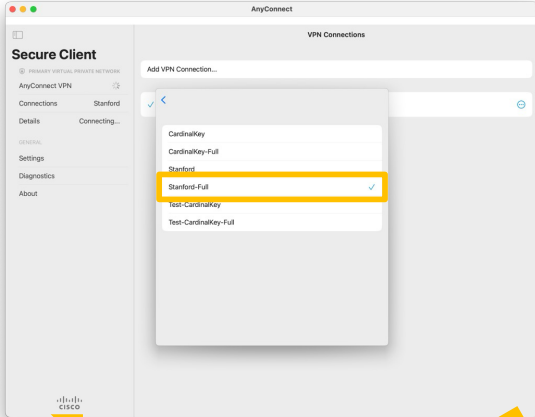
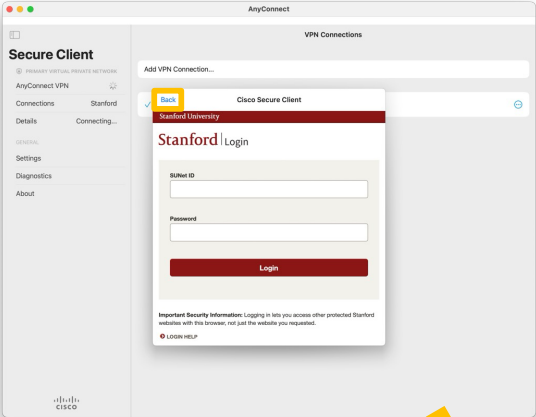
- Install VPN client on your laptop
- Connect to VPN
  - Group: Stanford-Full
  - Username & password on printed paper (red ink)
  - Select 1 for multi-factor -- we will accept



# Connect to Stanford VPN – Option 2 (Mac App Store)



# Connect to Stanford VPN – Option 2 (Mac App Store)



## SSH into Stanford Server

- After connecting to the VPN, open a terminal on your machine.
- SSH into Cafe-JG server and enter password when prompted:
  - Username & password on printed paper (black ink)

```
$ ssh -XY <username>@cafe-jg.stanford.edu
```

- Test display:

```
$ module load jaspergold  
$ jg -allow_unsupported_OS
```

- If you see the Jasper GUI open, great! You're done!
- **Else, if you get an error about the display, see next slide.**

## Debugging X11 Forwarding Issues

### On Mac:

1. Install XQuartz:  
<https://www.xquartz.org/>
2. Log out and back in (or reboot)
3. Launch XQuartz
4. Try SSH again and test display:

### On Windows:

#### Option A:

1. If you have WSL installed, try WSL.

#### Option B:

1. Install MobaXterm:  
<https://mobaxterm.mobatek.net>
2. Launch MobaXterm
3. Try SSH again and test display:

```
$ ssh -XY <username>@cafe-jg.stanford.edu  
$ module load jaspergold  
$ jg -allow_unsupported_OS
```

Ignore this pop-up (harmless)

Jasper Apps Expert System User Profile

Personal Information

Completing the user profile improves system performance since only those recommendations most relevant to your level of expertise will be displayed

What is your name? *(required)*

What is your level of experience in Formal Verification?

What is your level of experience in Jasper?

Do not show again

OK Cancel

## Working directory and environment setup

```
$ cd ~/fava_isca2026 && . ./setup_venv.sh
```



# Hardware Model Checking Quick Start

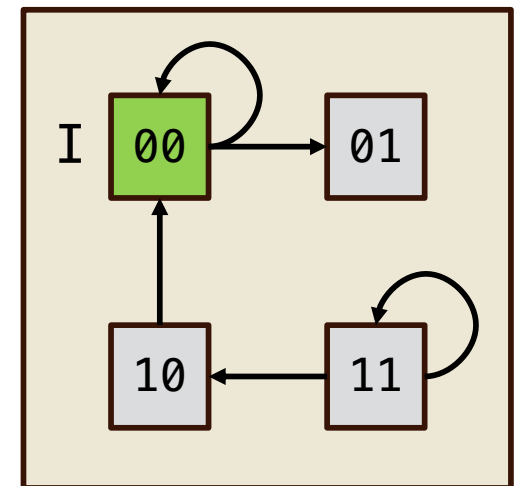
Samantha

Stanford | ENGINEERING

## Hardware Model Checking: Background

- A hardware design is modeled as a transition system, described by a tuple  $\langle S, I, T \rangle$ 
  - **S**: Set of all states
  - **I**: Set of initial states
  - **T**: Transition relation
- Example:
  - $S = \{00, 01, 10, 11\}$
  - $I = \{00\}$
  - $T = \{(00, 00), (00, 01), (10, 00), (11, 10), (11, 11)\}$

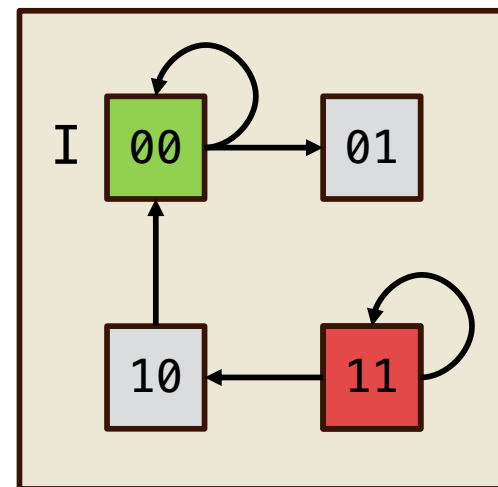
Finite state machine



## Hardware Model Checking: Background

- A property **P** describes a **set of traces** (sequences of states starting at some  $s_0 \in I$ )
  - **Assert P** – prove all reachable traces of the design belong to P
  - **Cover P** – find some reachable trace of the design that belongs to P
  - **Assume P** – consider only traces that belong to P
- Subject to any **assumes**, a **model checker** searches all traces of a design under verification (**DUV**) for:
  - One that satisfies a **cover**
    - Ex 1: cover {01} ✓  $\geq 1$  **Reachable** trace
    - Ex 2: cover {11} ✗ **Unreachable**
  - One that violates an **assert**
    - Ex 3: assert {!11} ✓ **Proven** for all traces
    - Ex 4: assert {00} ✗ **Counterexample**

Finite state machine



# Hardware Model Checking: Cadence Jasper GUI

This tutorial uses Jasper, a suite of model checkers by Cadence

The screenshot displays the Cadence Jasper GUI interface. The Design Hierarchy Browser on the left shows a tree structure with 'counter (counter)' selected. The Property Table on the right lists various properties with columns for Type, Name, Engine, Bound, Traces, Time, and Task. The Console at the bottom shows a command prompt with the text 'enter commands here' and a red arrow pointing to the input field.

Type	Name	Engine	Bound	Traces	Time	Task
Assert	counter.UNDERFLOW	Hp	2	1	0.0	<embedded>
Cover (rel...)	counter.UNDERFLOW;precondition1	PRE	1	1	0.0	<embedded>
Assume	counter.INPUT_UP_INTERFACE	?	0	0	0.0	<embedded>
Assert	counter.OVERFLOW	Hp	3	1	0.0	<embedded>
Cover (rel...)	counter.OVERFLOW;precondition1	Hp	2	1	0.0	<embedded>
Cover	counter.EVER_FULL	Hp	3	1	0.0	<embedded>

Design Hierarchy Browser

Property Table

Design Hierarchy

Property Table

Design Hierarchy Task Tree Total: 9 Filtered: 9 Selected: 0 Validity: 5:3:0:1 Run: 1:0:0:8

session\_0 session\_1 session\_2

```
%% report -summary -results -file jg_summary.txt -force ;# generate a result report
determined
```









[<embedded>] % enter commands here

console

## Hardware Model Checking: Property Information in GUI

Property Table									
No filter									
Filter on name									
Type	Name	Engine	Bound	Traces	Time	Task			
✗	Assert	counter.UNDERFLOW	Hp	2	1	0.0	<embedded>		
✓	Cover (rel...)	counter.UNDERFLOW:precondition1	PRE	1	1	0.0	<embedded>		
●	Assume	counter.INPUT_UP_INTERFACE	?		0	0.0	<embedded>		
✗	Assert	counter.OVERFLOW	Hp	3	1	0.0	<embedded>		
✓	Cover (rel...)	counter.OVERFLOW:precondition1	Hp	2	1	0.0	<embedded>		
✓	Cover	counter.EVER_FULL	Hp	3	1	0.0	<embedded>		

↑ Status/Proof results      ↑ Type of property      ↑ Property name      ↑ Evaluated time

-  Assumption
-  Unprocessed property
-  Processing property
-  Proven/Covered property
-  CEX/Reachable property
-  Undetermined property
-  Vacuous proven property
-  Error Assumptions

- When a model checker can neither find a trace in  $P/\neg P$  (for **cover/assert**) nor prove one does not exist, it emits a **bounded proof** of  $\neg P/P$ , indicating that a trace could not be found **within N transitions** (clock cycles) from any  $s_0 \in I$

## Tutorial Terminology

**DUV:** Design under verification

- For this tutorial, the DUV is a SystemVerilog hardware design

**IUV:** Instruction under verification

- A symbolic (i.e., arbitrary, to be selected by the model checker) instruction, whose encoding / program counter are contained in signals **i0** / **pc0**
- Often constrained to a certain instruction type (i.e., the opcode field of **i0** is constrained)

**e0\_hpn:** “Happened” register for event **e0**

- Assume wire **e0** is a flag that indicates whether event **e0** is happening in the current cycle (e.g., **e0** may indicate whether instruction **i0** is being fetched)
- Register **e0\_hpn** will have value 1 if wire **e0** has had value 1 (i.e., event **e0** has happened) in any previous cycle; else, **e0\_hpn** will have value 0
- These registers are *added to the design automatically* via templates

## Tutorial File Screenshot Labels

- All files are labeled with their path relative to the working directory of the section of the tutorial that they are referenced in
- Each file is labeled/outlined in color as:
  - **Input files**, for which the user of our tools is responsible during normal tool use
  - **Outputs/Intermediate outputs**, which the user does/does not need to inspect during normal tool use
  - **SystemVerilog or Murϕ Design files** (no outline)