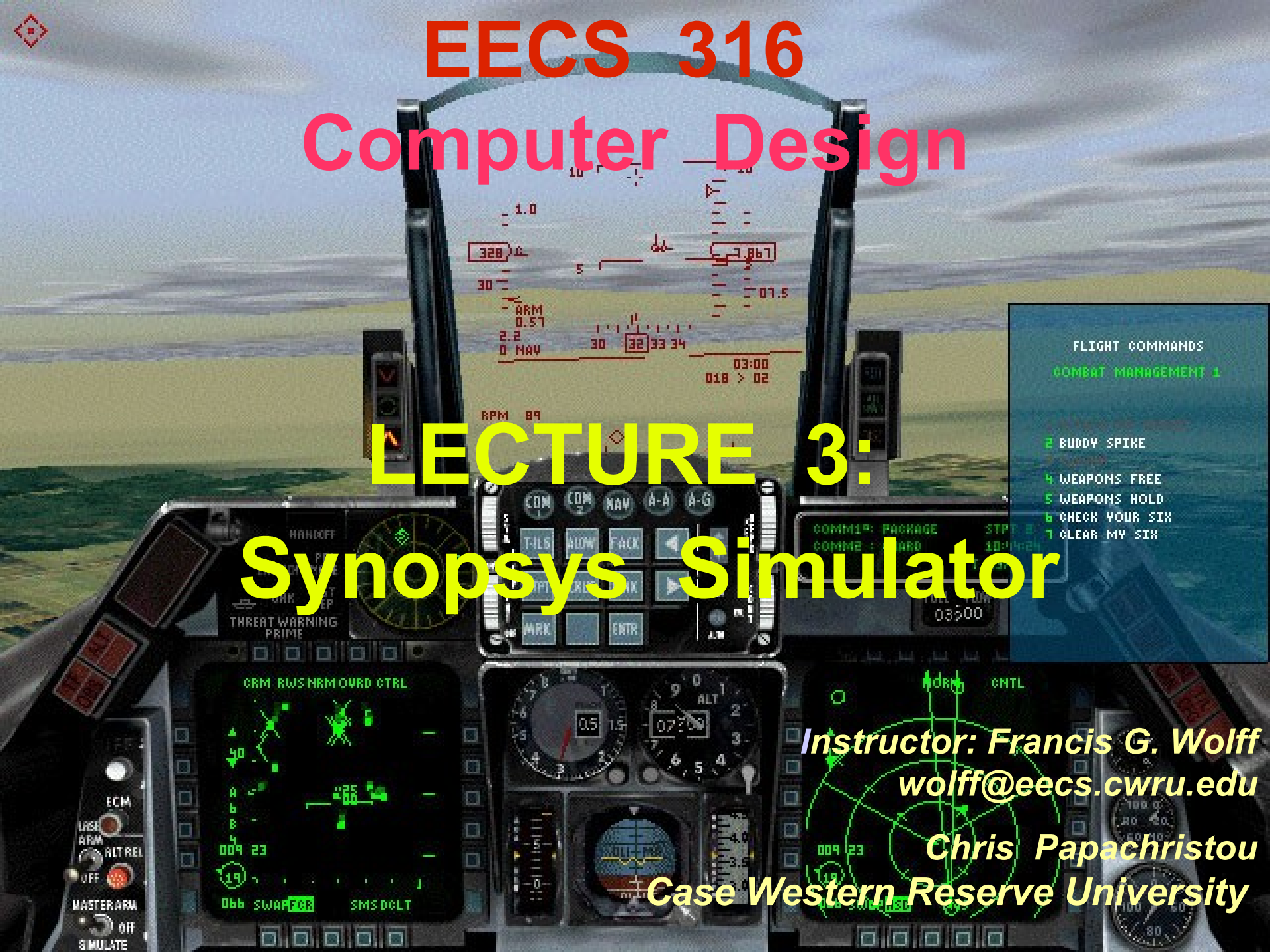
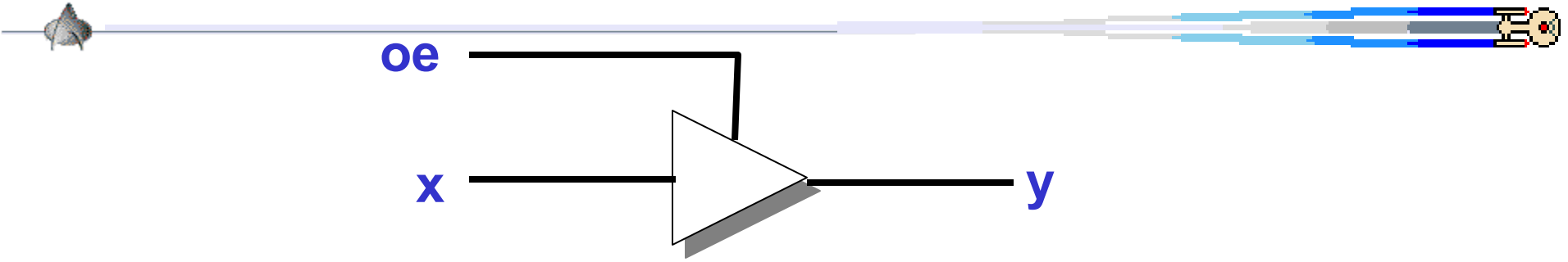

EECS 316
Computer Design
LECTURE 3:
Synopsys Simulator
Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Chris Papachristou
Case Western Reserve University

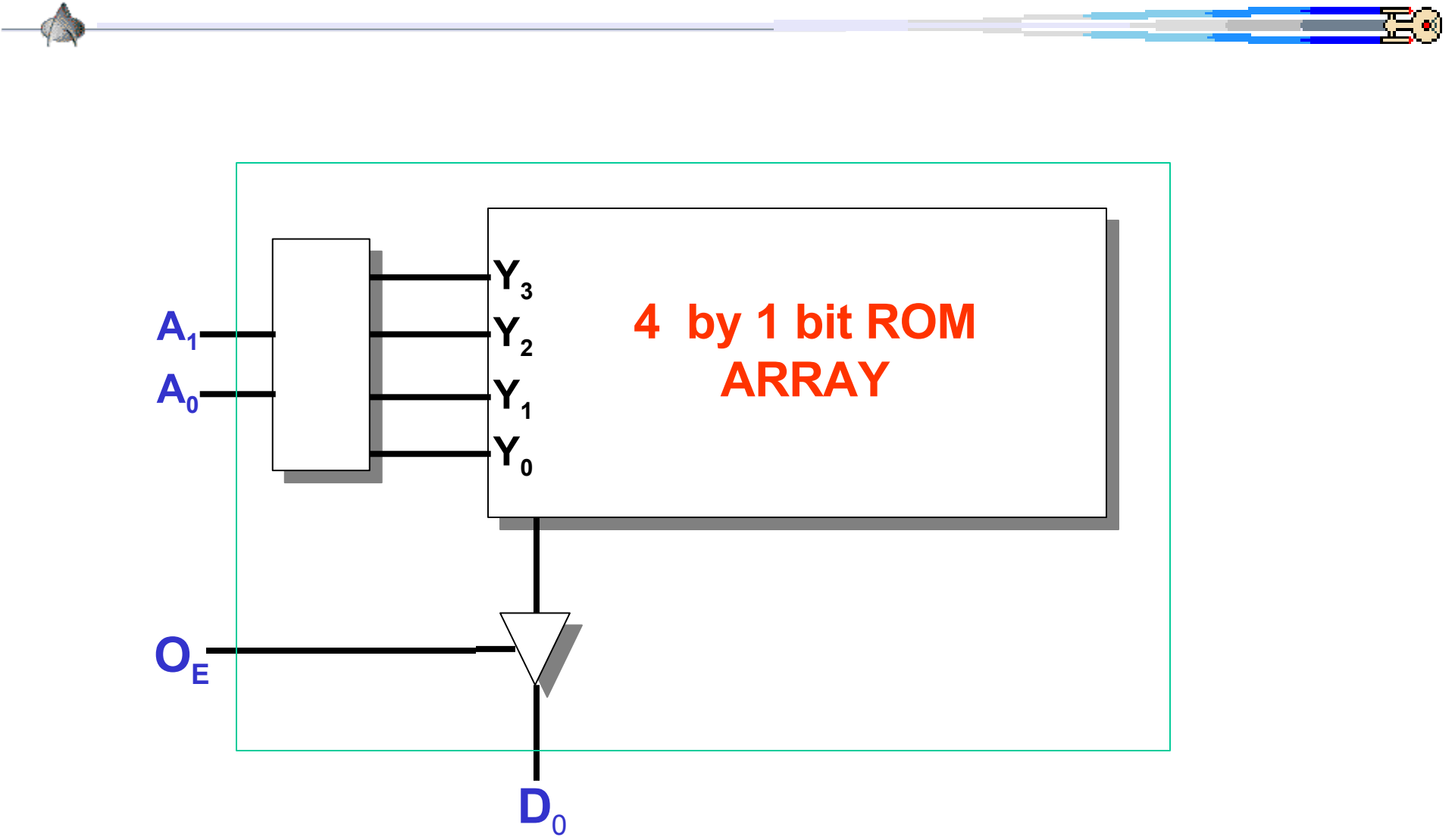
Review: Tri-State buffer




```
ENTITY TriStateBuffer IS
        PORT(x:     IN     std_logic;
             y:     OUT    std_logic;
             oe:    IN     std_logic
); END;
```

```
ARCHITECTURE Buffer3 OF TriStateBuffer IS
BEGIN

        WITH oe SELECT
        y <=    x   WHEN '1',   -- Enabled:  y <= x;
                'Z' WHEN OTHERS; -- Disabled: output a tri-state

END;
```

Review: ROM: 4 bit Read Only Memory



Review: ROM: 4 bit Read Only Memory

```
ENTITY rom_4x1 IS
        PORT(A:     IN std_logic_vector(1 downto 0);
                OE:    IN std_logic; -- Tri-State Output
                D:     OUT std_logic
); END;
```

```
ARCHITECTURE rom_4x1_arch OF rom_4x1 IS
  SIGNAL ROMout: std_logic;
BEGIN
  BufferOut: TriStateBuffer PORT MAP(ROMout, D, OE);
  WITH A SELECT
        ROMout <=   '1' WHEN "00",
                    '0' WHEN "01",
                    '0' WHEN "10",
                    '1' WHEN "11";
```

Component Instance

Component declaration name

Component Declaration/Instance relationship

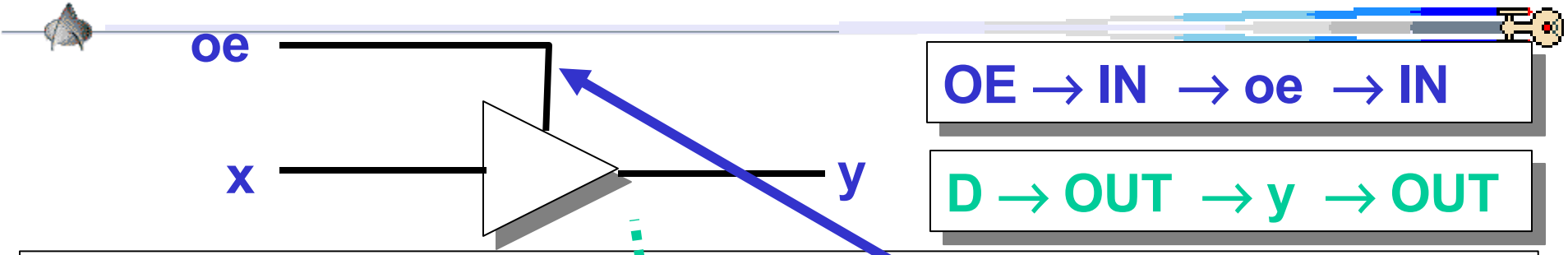
```
ARCHITECTURE rom_4x1_arch OF rom_4x1 IS

  COMPONENT TriStateBuffer
  PORT (x: IN std_logic; y: OUT std_logic, oe: IN std_logic);
  END COMPONENT;

  SIGNAL ROMout: std_logic;
BEGIN
   BufferOut: TriStateBuffer PORT MAP(ROMout, D, OE);

   WITH A SELECT
         ROMout <=   '1' WHEN "00",
                     '0' WHEN "01",
                     '0' WHEN "10",
                     '1' WHEN "11";
END;
```

Component Declaration

Colon (:) says make a Component Instance

Component Instance Name: *BufferOut*

Component Port relationship



OE → IN → oe → IN

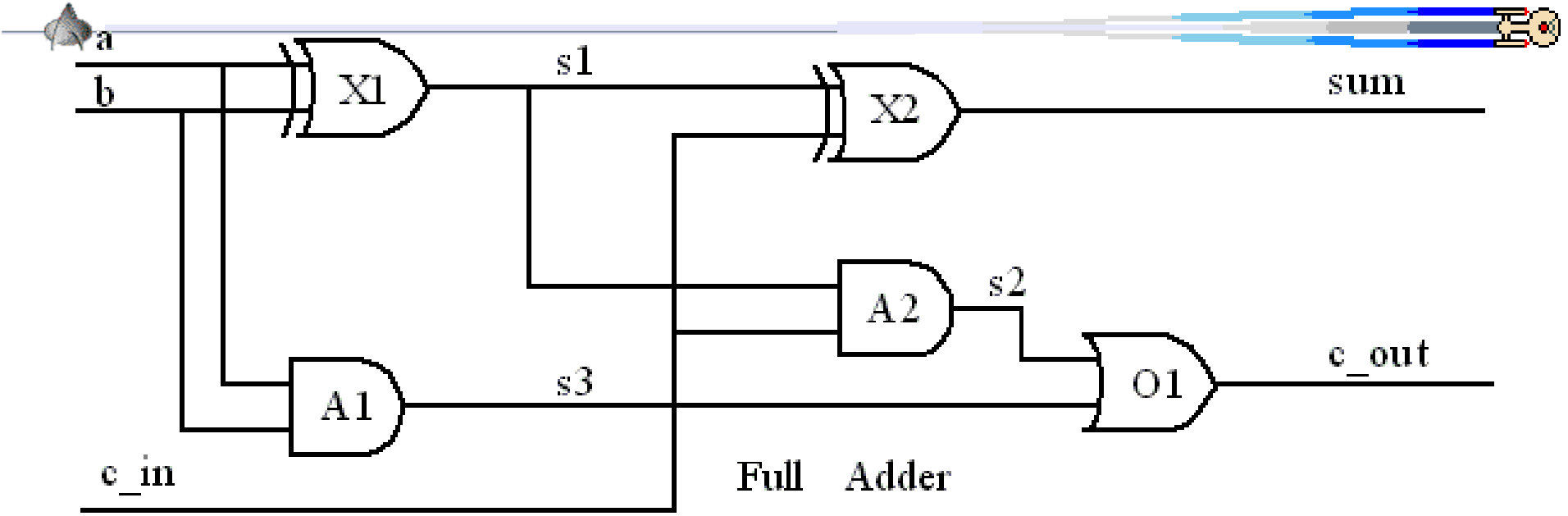
D → OUT → y → OUT

```
COMPONENT TriStateBuffer
  PORT (x: IN std_logic; y: OUT std_logic, oe: IN std_logic);
END COMPONENT;
```

```
BufferOut: TriStateBuffer      PORT MAP(ROMout, D, OE);
```

```
ENTITY rom_4x1 IS
        PORT(A:     IN std_logic_vector(1 downto 0);
                OE:     IN std_logic; -- Tri-State Output
                D:      OUT std_logic
); END;
```

Full Adder: Architecture




```
ARCHITECTURE adder_full_arch OF  adder_full  IS


BEGIN
        Sum  <= ( x XOR y ) XOR Cin;
        Cout <= ( x AND y ) OR (Cin AND (x XOR y));
END;
```

adder_full.vhd: complete file

```
LIBRARY IEEE;
use IEEE.std_logic_1164.all;
```

```
ENTITY adder_full IS
    PORT (x, y, Cin: IN std_logic; Sum, Cout: OUT std_logic
); END;
```

```
ARCHITECTURE adder_full_arch OF  adder_full  IS

BEGIN
        Sum   <= ( x XOR y ) XOR Cin;
        Cout   <= ( x AND y ) OR (Cin AND (x XOR y));
END;
```

```
CONFIGURATION  adder_full_cfg OF  adder_full IS
        FOR adder_full_arch
        END FOR;
END CONFIGURATION;
```

The Synopsys VCS Simulator

Starting Synopsys environment

- if remote: ssh host.ces.cwru.edu

  host is one of: hp04 hp05 hp06 hp07 hp08 hp09

  - note: you can have several sessions of ssh from your computer

- logon

if local: Open a console window

- Start typing within the console window
- Start the tshell: /bin/tcsh
- Source the synopsys executable paths and environment
  - source /local/eda/synopsys_setup
- Change your directory to Synopsys: cd ~/SYNOPSYS

VHDL analyzer: vhdlan

The vhdl analyzer analyzes the vhdl for syntax errors

Unix command: vhdlan –NOEVENT <filename.vhd>

- **Must be done to every vhdl file in the design**

For example:
> vhdlan –NOEVENT adder_full.vhd

Synopsys VHDL Analyzer Version ... ...

Synthesis: Debugging syntax errors

Open a ssh host terminal #1 (or ssh) window and
Enter the generic_mux.vhd using an ascii editor:

vi adder_full.vhd	
i	--i for insert mode
….	--enter code
ESC	--Escape key to exit insert mode
:w	--write the file but do not exit

Open another telnet host terminal #2 (or telnet) window and run vhdlan for syntax errors.

vhdlan –NOEVENT adder_full.vhd

Use the editor in ssh host #1 to to fix the errors then write (:w) then in ssh host #2 type !vh to reanalyze the vhdl source code until there are no more errors.

VHDL Simulator: scs

Unix command: scs <vhdl_configuration_name>

- **Starts the text based vhdl simulator**

For example:

> scs adder_full_cfg

Synopsys 1076 VHDL Simulator Version ... ...

Copyright (c) 1990-2000 by Synopsys, Inc.
ALL RIGHTS RESERVED

This program is proprietary and confidential information of Synopsys, Inc. and may be used and disclosed only as authorized in a license agreement controlling such use and disclosure.

#

Simulator command line prompt #

VHDL Simulator list components: ls

scs list command: ls [–t] [–v]

- lists the vhdl component types and data values

After reading in the adder_full.vhd design, a list will show

```
# ls
ADDER_FULL        STANDARD        ATTRIBUTES
STD_LOGIC_1164    _KERNEL


# ls –t
ADDER_FULL        COMPONENT INSTANTIATION STATEMENT
STANDARD          PACKAGE
ATTRIBUTES        PACKAGE
STD_LOGIC_1164    PACKAGE
_KERNEL           PROCESS STATEMENT
#
```

VHDL Simulator change directory: cd and pwd

scs cd command:

```
cd  <component_path>
cd  ..
pwd
```

- cd - change design hierarchy (cd .. go up a level)
- pwd - display present working directory

```
# cd   ADDER_FULL
# pwd
/ADDER_FULL
```

Alternately, using full paths
ls –t /ADDER_FULL

```
#  ls  –t
X               IN PORT       type = STD_LOGIC
Y               IN PORT       type = STD_LOGIC
CIN             IN PORT       type = STD_LOGIC
SUM             OUT PORT      type = STD_LOGIC
COUT            OUT PORT      type = STD_LOGIC
_P0             PROCESS STATEMENT
```

VHDL Simulator assign signal: assign

scs command: assign [–after <time>] <value> <signal>

- assign a value to a signal

```
# ls –v
X                'U'
Y                'U'
CIN              'U'
SUM              'U'
COUT             'U'
# assign ‘1’ X
# ls –v
X                '1'
Y                'U'
CIN              'U'
SUM              'U'
COUT             'U'
```

Alternately, using full paths
assign ‘1’ /ADDER_FULL/X

VHDL Simulator run simulation: run

scs command: run [<time nanoseconds>]

- **Use Control-C to cancel a simulation**

```
# assign   '1'   X
# assign   '1'   Y
# assign   '0'   Cin
#  ls   –v
X         '1'
Y         '1'
CIN       '0'
SUM       'U'
COUT      'U'
# run
# ls  –v
X         '1'
Y         '1'
CIN       '0'
SUM       '0'
COUT      '1'
```

This is what we would expect

VHDL Simulator include file

scs command: source [-e] <filename.scsim>

- Reads and executes **scs** commands from a file
- -e will displays the lines as it reads them in

For example, the file adder_full.scsim contains:

```
cd ADDER_FULL
assign '1' X
assign '1' Y
assign '0' Cin

ls –v >adder_full.run
run
ls –v >>adder_full.run
quit
```

> overwrite file

>> append to file

VHDL Simulator include using full path names

For example, adder_full.scsim using full path names:

```
assign  '1'  /ADDER_FULL/X
assign  '1'  /ADDER_FULL/Y
assign  '0'  /ADDER_FULL/Cin

ls  –v  /ADDER_FULL  >adder_full.run
run
ls  –v  /ADDER_FULL  >>adder_full.run
quit
```

VHDL Simulator trace

scsim command: trace <signals>

- Traces vhdl signals on GUI Synopsys Waveform Viewer
- To best view signals a time element must be added
- Use View => Full Fit in order to fully view the signals

For example,

```
cd ADDER_FULL
assign –after 5 '1' X
assign –after 5 '1' Y
assign –after 5 '0' Cin
trace X Y Cin Sum Cout
ls –v
run
ls –v
exit
```

VHDL Simulator: abstime, step, next, status

vhdlsim command: abstime

- display the current absolute simulation time so far

scsim command: step [<n steps>]

- step through each vhdl statement, default n=1

scsim command: next [n steps]

- step through each vhdl statement within current arch

scsim command: status

- show current simulation status

VHDL Simulator: where, environment, restart

scsim command: where

- displays where the process and event stacks

scsim command: environment

- displays the simulator environmental variables

scsim command: restart

- restart the simulation using all previous commands
- Clean restart: restart /dev/null

scsim command: quit

- exit the simulator environment

VHDL Simulator: help

scsim command: help [<simulator_command>]

- **simulator command help: help ls**

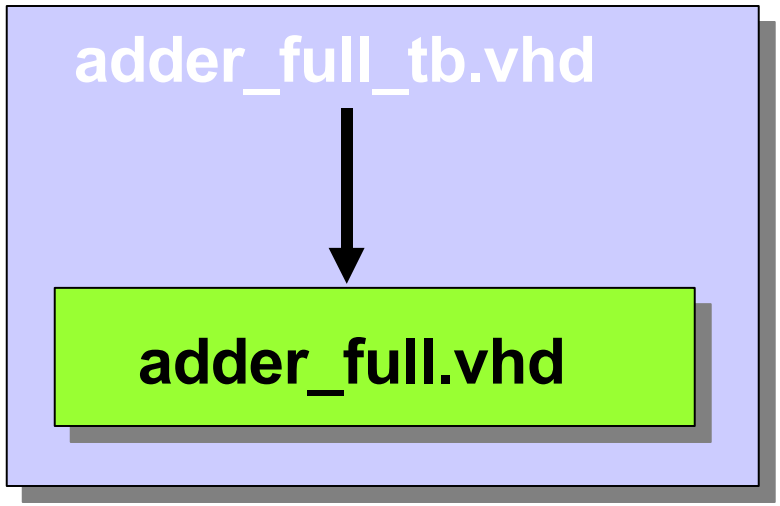
 # help step
Subject: STEP
Syntax: STEP [n]

STEP executes the next "n" lines of VHDL source code. If you omit the argument "n", it executes a single line.

STEP enters functions and procedures.

STEP does not count or stop on lines that are monitored by an OFF monitor.

adder_full_tb.vhd: full adder test bench



Stimulus Only Test Bench Entity

The output of the testbench will be observe by the digital waveform of the simulator.

```
LIBRARY IEEE;
use IEEE.std_logic_1164.all;
```

```
ENTITY adder_full_tb IS
        PORT (Sum, Cout:          OUT std_logic);
END;
```

adder_full_tb.vhd: architecture

```
ARCHITECTURE adder_full_tb_arch OF adder_full_tb IS
   COMPONENT adder_full
     PORT (x, y, Cin: IN std_logic; Sum, Cout: OUT std_logic);
   END COMPONENT;
   SIGNAL x, y, Cin: std_logic;
BEGIN
  x <= '0',  '1' after 50 ns,  '0' after 100 ns; --Test Input
  y <= '1',  '1' after 50 ns,  '0' after 100 ns;
  Cin <= '0',  '1' after 50 ns;
  UUT_ADDER: adder_full PORT MAP(x, y, Cin, Sum, Cout);
END;
```

```
CONFIGURATION  adder_full_tb_cfg OF  adder_full_tb IS
         FOR adder_full_tb_arch END FOR;
END CONFIGURATION;
```

VHDL Simulator: test bench

```
Unix> vhdlan –NOEVENT adder_full.vhd
Unix> vhdlan –NOEVENT adder_full_tb.vhd
Unix> scs adder_full_tb_cfg

# ls
ADDER_FULL_TB   STANDARD      ATTRIBUTES
STD_LOGIC_1164  _KERNEL
# cd ADDER_FULL_TB
# ls
SUM          _P0          _P2          ADDER_FULL     Y
COUT         _P1          UUT_ADDER    X              CIN
# ls –t
SUM                  OUT PORT     type = STD_LOGIC
COUT                 OUT PORT     type = STD_LOGIC
UUT_ADDER  COMPONENT INSTANTIATION ADDER_FULL
COMPONENT
X                    SIGNAL       type = STD_LOGIC
```

VHDL Simulator: run 10 ns

```
# ls  –v
SUM             'U'
COUT            'U'
X               'U'
Y               'U'
CIN             'U'
# run  10
10 NS
# ls  –v
SUM             '1'
COUT            '0'
X                '0'
Y                '1'
CIN             '0'
```

VHDL Simulator: run 60 ns (go passed 50ns)

```
# run  60
70 NS
# ls  –v
SUM              '1'
COUT             '1'
X                '1'
Y                '1'
CIN              '1’
# quit
```

VHDL Simulator GUI: scirocco

Unix command: scirocco <vhdl_configuration_name> &

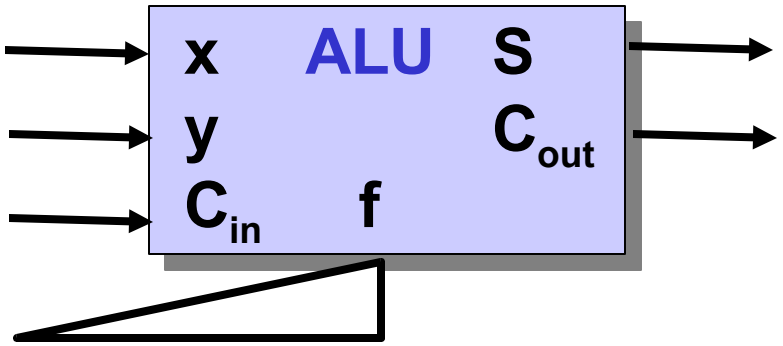
- Starts the VHDL GUI version of scs simulator
- Does everything scs does via menus

- Use the trace command to view signals
  - First mark the variable with the mouse
  - Then traces -> signals

Assignment #3 (1/3)

a) Test the vhdl code of assignment #2.3 1-bit alu and then run it using vhdlan and vhdlsim. Write a two useful test cases for each function (i.e. show one with carry and another without carry). Hand in the source files and session using the Unix script command (see next page).

function f	ALU bit operation
000	S = 0; Cout = 0
001	S = x
010	S = y; Cout =1;
011	S = Cin; Cout = x
100	S = x OR y; Cout=x;
101	S = x AND y; Cout=x;
110	(Cout, S) = x + y + Cin; (component)
111	(Cout, S) = full subtractor (component)



Assignment #3 (2/3)

1) logon…

2) /usr/bin/script assign3_a.txt

cp /local/eda/synopsys_setup $HOME

(synopsys_setup should be in your home dir)

3) source synopsys_setup

(this will create a SYNOPSYS directory at first time)

4) cd ~/SYNOPSYS

6) cat alu_bit.vhd

7) vhdlan –NOEVENT alu_bit.vhd

8) scs alu_bit_cfg

9) …test bench commands “assign”,”ls -v”, ...

10) quit

11) quit

12) lpr assign3_a.txt

Assignment #3 (3/3)

b) Write a vhdl test bench to the vhdl code of 3a 1-bit alu and then run it using vhdlan and vcs. Use the same test cases from part a. Hand in the source files and session using the Unix script command as follows:

1) logon…
2) /usr/bin/script assign3_b.txt
3) cp /local/eda/synopsys_setup $HOME
4) source synopsys_setup
5) cd ~/SYNOPSYS
6) cat alu_bit.vhd
7) cat alu_bit_tb.vhd
8) vhdlan –NOEVENT alu_bit.vhd
9) vhdlan –NOEVENT alu_bit_tb.vhd
10) scs alu_bit_tb_cfg
11) ….NO “assign” commands
12) quit
13) quit
14) lpr assign3_b.txt