EECS 470 Lab 2
Synopsys Build System

Department of Electrical Engineering and Computer Science
College of Engineering
University of Michigan

Friday, 17th January, 2014
Overview

Administrivia

Project 1

Generic Verilog Design
   Preprocessor Directives
   Parameters
   Hierarchical Design

Tools
   Make
   VCS
   Design Compiler
Homework

- Homework 2 is due Monday, 3rd February at the beginning of lecture

Projects

- Project 1 is due Tuesday, 21st January at 9:00PM

We are available to answer your questions. Office hours can be found in the course google calendar, and questions can be posted in the course Piazza.
Project 1 Administrivia

Grading

- Objective Grading
  - 70 points possible
  - Test cases automatically run

- Subjective Grading
  - 30 points possible
  - Verilog style graded by hand
Submission Script

- Run from the directory *above* the directory your project is in...
- For example:
  
  ```
  $ pwd
  /afs/umich.edu/user/w/c/wcunning/
  $ cd classes/eecs470/projects
  $ ls
  project1 project2 project3
  $ /afs/umich.edu/user/b/r/brehob/Public/470submit -p1 project1
  ```
Hierarchical Design

- Used to expand modules
  - Build a 64-bit adder out of 1-bit adders
  - Build a 4-bit and out of 2-bit ands
- No additional logic needed
Project 1 Hints

Hierarchical Design

▶ Used to expand modules
  ▶ Build a 64-bit adder out of 1-bit adders
  ▶ Build a 4-bit and out of 2-bit ands

▶ No additional logic needed

▶ Project 1 Part C and D
  ▶ Build a 4-bit priority selector out of 2-bit priority selectors
  ▶ Build a 4-bit rotating priority selector out of 2-bit priority selectors
module and2(
    input logic [1:0] a,
    output logic x
);
    assign x=a[0] & a[1];
endmodule

module and4(
    input logic [3:0] in,
    output logic out
);
    logic [1:0] tmp;
    and2 left(.a(in[1:0]),.x(tmp[0]));
    and2 right(.a(in[3:2]),.x(tmp[1]));
    and2 top(.a(tmp),.x(out));
endmodule
Project 1 Hints

```
x
and2
a[1]  a[0]

and2
a[1]  a[0]
```

```
in[1]  in[0]
```

```
out
```
Any questions about Project 1?
What is a preprocessor?

- Prepares code/designs for compilation
  - Combines included files
  - Replaces comments and whitespace

Why do I care?

- Programmable
- Generic Designs (parameterized caches anyone?)
Verilog Macros

Definitions

- Syntax: `define <NAME> <value>
  - Note: That is a tick, not an apostrophe before define
  - The tick character is found with the ~, usually above tab (on US standard layout keyboards)
- Usage: `<<NAME>>`
- Good for naming constants (can be used like wires)
- Convention dictates that macro names be in all caps
- Can also `undef <NAME>`
Verilog Macros

Flow Control
Text inside flow control is conditionally included in the compiled source file.

`ifdef Checks if something is defined
`else Normal else behavior
`endif End the if
`ifndef Checks if something is not defined
Verilog Macros by Example

```
`define DEBUG
`define LOCKED 1'b0
`define UNLOCKED 1'b1
module turnstile(
    input logic coin, push,
    input logic clock, reset
`ifdef DEBUG
    ,output logic state
`endif
);
`ifndef DEBUG
logic state;
`endif
always_comb begin
    next_state = state;
    if (state == `LOCKED && coin) next_state = `UNLOCKED;
    if (state == `UNLOCKED && push) next_state = `LOCKED;
end
always_ff @(posedge clock) begin
    if (reset) state <= #1 `LOCKED;
    else state <= #1 next_state;
end
endmodule
```
Verilog Headers

What is inclusion?

- Paste the code from the specified file where the directive is placed

Where will I use this?

- System defines separated out, e.g.
  ```
  `define ALU_OP_ADD 5'b10101
  `define DCACHE_NUM_WAYS 4'b0100
  ```
- Macro assertion functions, printing functions, etc.
- Note: headers are named with the .vh extension
Verilog Headers

Inclusion

- Syntax: `include <FILE_NAME>
- Pastes the contents of `<FILE_NAME>` wherever the include appears

Include Guards

- Inside the header...

  ```
  `ifndef __FILE_NAME_H__
  `define __FILE_NAME_H__
  ...
  `endif
  ```
Verilog Inclusion by Example

```verilog
and4.v

`include and2.v
module and4(
    input logic [3:0] in,
    output logic out
);
    logic [1:0] tmp;
    and2 left(.a(in[1:0]),.x(tmp[0]));
    and2 right(.a(in[3:2]),.x(tmp[1]));
    and2 top(.a(tmp),.x(out));
endmodule
```
and2.v

module and2(
    input logic [1:0] a,
    output logic x
);
    assign x=a[0] & a[1];
endmodule
Verilog Inclusion by Example

Better and2.v

```verilog
`ifndef __AND_2_V__
`define __AND_2_V__

module and2(
    input logic [1:0] a,
    output logic x);
    assign x=a[0] & a[1];
endmodule

`endif
```
Bibliography

Much of this information was taken from the Verilog Preprocessor paper by Wilson Snyder of Cavium Networks. It is highly recommended reading.
Parameters

What is a parameter?

- Constant defined inside a module
- Used to set module properties
- Can be overridden on instantiation

How do I use them?

- Definition
  ```
  parameter NUM_CACHE_LINES = 8;
  ```
Setting Parameters

Overriding

- Set parameters for a module on instantiation
- Allows for different versions different places
- Usage:
  
```
cache #(NUM_CACHE_LINES(4)) d_cache(...);
```

defparam

- Set parameters for a module
- Usage:
  
```
defparam dcache.NUM_CACHE_LINES = 4;
```
Macros Vs. Parameters

Macros

- Possibly globally scoped – namespace collision
- Use for modules that can change, but only have one instance
- Particularly for caches and naming arbitrary constants
- Needs the ` in usage

Parameters

- Locally scoped – no namespace collision
- Use for modules with many instances at different sizes
- Particularly for generate blocks (which are in a later lab)
- Does not need extra characters (like the `)
Array Connections

- Make a simple module a duplicate it several times
- Assume we have a module definition:
  - `one_bit_addr(a, b, cin, sum, cout);
- All ports are 1 bit, the first three inputs, last two outputs
- How do we build an eight bit adder?
module eight_bit_addr(
    input en, cin,
    input [7:0] a, b,
    output [7:0] sum,
    output cout);

    wire [6:0] carries;

    one_bit_addr a0(en, a[0], b[0], cin, sum[0], carries[0]);
    one_bit_addr a1(en, a[1], b[1], carries[0], sum[1], carries[1]);
    one_bit_addr a2(en, a[2], b[2], carries[1], sum[2], carries[2]);
    one_bit_addr a3(en, a[3], b[3], carries[2], sum[3], carries[3]);
    one_bit_addr a4(en, a[4], b[4], carries[3], sum[4], carries[4]);
    one_bit_addr a5(en, a[5], b[5], carries[4], sum[5], carries[5]);
    one_bit_addr a6(en, a[6], b[6], carries[5], sum[6], carries[6]);
    one_bit_addr a7(en, a[7], b[7], carries[6], sum[7], cout);

endmodule
The Error Prone Way

- Lots of duplicated code
- Really easy to make mistake
- Now try building a 64-bit adder..., 256?
- There is a one line substitute
The Better Way

module eight_bit_addr(
    input en, cin,
    input [7:0] a, b,
    output [7:0] sum,
    output cout);

wire [6:0] carries;

one_bit_addr addr [7:0] (  
    .en(en), .a(a), .b(b), .cin({carries,cin}),  
    .sum(sum), .cout({cout,carries})  
);
endmodule

All of the ports in the one_bit_addr module are 1 bit wide. All of the busses we pass are 8 bits wide, so each instantiation of the module will get one, except en which is only 1 bit wide and thus copied to every module.
The (Even) Better Way

```
`define ADDR_WIDTH 8
module eight_bit_addr(
    input en, cin,
    input [(`ADDR_WIDTH-1):0] a, b,
    output [(`ADDR_WIDTH-1):0] sum,
    output cout);

wire [(`ADDR_WIDTH-2):0] carries;

one_bit_addr addr [(`ADDR_WIDTH-1):0] (  
    .en(en), .a(a),.b(b),.cin({carries,cin}),  
    .sum(sum),.cout({cout,carries})  
);
endmodule
```
Array Connections: Pitfalls and Errors

What happens if a wire isn’t 1 or N bits wide?

```verilog
wire foo;
wire [3:0] bar;
wire [2:0] baz;

simple s1 [3:0](.a(foo), .b(bar), .c(baz));
```

test.v:14: error: Port expression width 3 does not match expected width 4 or 1
But still easy to accidentally promote a wire to a bus
What is Make?

- Build System
  - Automatically build an *executable* from *source* files
  - Rules for building are stored in a *Makefile"
- Essentially a script of compilation commands
- Handles dependency resolution
What is Make?

- Build System
  - Automatically build an *executable* from *source* files
  - Rules for building are stored in a *Makefile*
- Essentially a script of compilation commands
- Handles dependency resolution

Why do I care?
What is Make?

What is Make?

- Build System
  - Automatically build an executable from source files
  - Rules for building are stored in a Makefile
- Essentially a script of compilation commands
- Handles dependency resolution

Why do I care?

- simv is an executable
- Verilog files are source files
- vcs is a compilation command
Anatomy of a Makefile

- **targets** are what we want to build
- **dependencies** are what we need to build it
- **commands** are how to build it

This looks something like the following

```
target: dep1 dep2 ...
    command1
    command2
```
Lab 1 Makefile

```
simv: $(SIMFILES) $(TESTBENCH)
    $(VCS) $(TESTBENCH) $(SIMFILES) -o simv -R | tee program.out

syn_simv: $(SYNFILES) $(TESTBENCH)
    $(VCS) $(TESTBENCH) $(SYNFILES) $(LIB) -o syn_simv | tee synth.out

syn: syn_simv
    ./syn_simv | tee syn_program.out
```

What do the $(() mean?
A *macro* is a way to store custom build behavior.

```make
VCS = SW_VCS=2011.03 vcs -sverilog +vc -Mupdate -line -full64

TESTBENCH = tut_test.v
SIMFILES = tut_mod.v

simv: $(SIMFILES) $(TESTBENCH)
    $(VCS) $(TESTBENCH) $(SIMFILES) -o simv -R | tee program.out
```
Dependency Resolution

What does it mean to be a dependency?

- Dependencies are used during compilation, e.g., list of source files
- Might be intermediate or primary

How does Make resolve dependencies?

- Does the target exist?
  - **No** – Build it.
  - **Yes** – When was it built in relation to the dependencies?
    - **After** – Stop.
    - **Before** – Rebuild.
Automatic Makefile Macros

$@  The Current Target Name
$^  A List of All Dependencies
$<  The First Prerequisite
$?  A List of All Dependencies Requiring Rebuilding
Special Makefile Targets

.DEFAULT  Sets what runs when make is executed with no arguments

.PHONY  Dependencies will be built unconditionally

.PRECIOUS  Dependencies will be kept, intermediate or not

.SECONDARY  Dependencies are automatically treated as intermediates, but not deleted
Make Resources

- GNU Make Manual
  - Special Targets
  - Automatic Variables
  - Makefile Conventions
- Wikipedia - Make (Software)
Intro to VCS

What is VCS?
▶ Synopsys Verilog Compiler Simulator
▶ Builds Simulators from Verilog (structural or behavioral)
▶ We barely manage to brush the surface...

Why do we care?
▶ Knowledge is power...
▶ ...Specifically the power to debug
VCS by Example

VCS = SW_VCS=2011.03 vcs -sverilog +vc -Mupdate -line -full64

SW_VCS  CAEN specific; sets which one of the installed VCS versions is run
-sverilog Interpret designs using the SystemVerilog standard
+vc    Allow direct C code hooks in the design
-Mupdate Compile incrementally
-line   Allow interactive debugging (might need other options)
-full64 Build 64 bit executables
More VCS Options

+define  Define a `define macro
  Ex. +define+DEBUG, Ex. +define+CLK=10
+lint=all  Provide many more warnings
  -gui  Build a DVE executable for interactive waveform debugging
  -o  Name of the executable generated
  -R  Run the executable after compilation
What is synthesis?

- The process of turning Behavioral Verilog into Structural Verilog
- Using a technology library

Why do I care?

- Example of how a design might be built
- Informed guesses about...
  - Power
  - Area
  - Performance
Intro to Design Compiler and Synthesis

What is Design Compiler?

- The Synopsys synthesis tool
- Industry-leading

Why do we care?

- The tool for EECS 470
- Once again, because knowledge is power...
- ...Specifically, the power to Google
Design Compiler by Example

read_file -f sverilog [list "tut_mod.v"]
set design_name two_bit_pred
set clock_name clock
set reset_name reset
set CLK_PERIOD 6

read_file  Read through the files in the list for dependency resolution and error, perform operator replacement
design_name  Design to be synthesized
clock_name  Name of the clock net
reset_name  Name of the reset net
clk_period  Clock period ($T_{clk}$) in ns
Tools
Design Compiler

Design Compiler by Example

```
compile -map_effort medium
```

**compile** This command actually synthesizes the design. 
- **map_effort** can be medium or high.

```
redirect $chk_file { check_design }
```

**$chk_file** This command creates the *.chk file with the design warnings

```
write -hier -format verilog -output $netlist_file $design_name
```

**$netlist** Writing with -format verilog generates the *.vg file containing the structural verilog output
Design Compiler by Example

```
write -hier -format ddc -output $ddc_file $design_name
```

$ddc_file  This command generates the *.ddc file, which is included in designs where we want to synthesize a module separately. More on this in Project 2.
Design Compiler by Example

```
redirect $rep_file { report_design -nosplit }
redirect -append $rep_file { report_area }
redirect -append $rep_file { report_timing -max_paths 2 -input_pins
-nets -transition_time -nosplit }
redirect -append $rep_file { report_constraint -all_violators
-verbose -nosplit }
```

$rep_file  This command generates the *.rep file, containing the timing report for the synthesized design. More on this in Project 2.
Design Compiler by Example

```
syn_simv: $(SYNFILES) $(TESTBENCH)
    $(VCS) $(TESTBENCH) $(SYNFILES) $(LIB) -o syn_simv | tee synth.out
```

| tee | This command saves the output of the synthesis tool into a file. Errors and warnings found in this file are particularly important. |
Designs in Memory

analyze...elaborate

- Creates intermediate files
  - Specifically, templates of generated designs with parameterized sizes
- Automatically links design components to templates
- Specifically resolves several errors that you will encounter in the final project

VS.

read_file

- Simpler, single command
- No intermediate files
- No linking
- Can cause problems
Lab Assignment

- Assignment is posted to the course website as **Lab 2 Assignment**.
- If you get stuck...
  - Ask a neighbor, quietly
  - Put yourself in the help queue
- When you finish the assignment, sign up in the help queue and mark that you would like to be checked off.
- If you are unable to finish today, the assignment needs to be checked off by a GSI in office hours _before_ the next lab session.