Lec-1a-tools **SEE Tools Documentation, installation and usage:**

projects/LC3trunk/docs/README-

-Electric -Subversion -verilog -unix

SUBVERSION (version control)

Two svn repositories:

https://svn.cs.georgetown.edu/svn/projects/ (Course project materials)
and
https://svn.cs.georgetown.edu/svn/projects2/ (Course documents, your project branches)

They both use the same **username**/**password:**

250-374-developer y(&qwqsq

Copy URLs to a Web browser.
You can see the current revision of the repository and hand copy files.
You will be prompted for a couple of reasons:

(1) **certificate cannot be authenticated**: Accept as a permanent exception.
(2) **login authentication:** Use username/password.

----- Sometimes, you get the same prompts twice: Just **do everything twice**.

NEVER do SVN IMPORT or EXPORT.

Getting a local copy

```
svn co https://svn.cs.georgetown.edu/svn/projects/
```

Local name of your working copy will be "**projects**"
in your system's directory tree where you did **svn co**.

Problems? Erase your working (local) copy,

```
/bin/rm -rf projects
```
(**BUT, move and save your changed work out FIRST**)

Local rename your working copy is ok, but **only the root**:

```
mv project MyWorkingCopyOfProjects
```

For **help** with Subversion commands,

```
svn help
```

UNIX stuff you should learn/know/review

Typical commandline tools:

vi / **emacs**: editors
sh / **make**: shell commands, build dependencies
grep: pattern matching in files
sed: stream editing
awk: stream editing w/ more complexity
m4/**cpp**: pre-processors

typical shell commands:

man info ls pwd cd rm mv cp exit echo cat
mkdir rmdir alias set which whereis
jobs, ctl-z, fg, %2, &
ps -ex, kill -6 (-9)
> >> | <
tar
gzip, gunzip, compress, uncompress (.z)

Things unix

processes, login shell, child processes, environment variables, open files, stdin, stdout

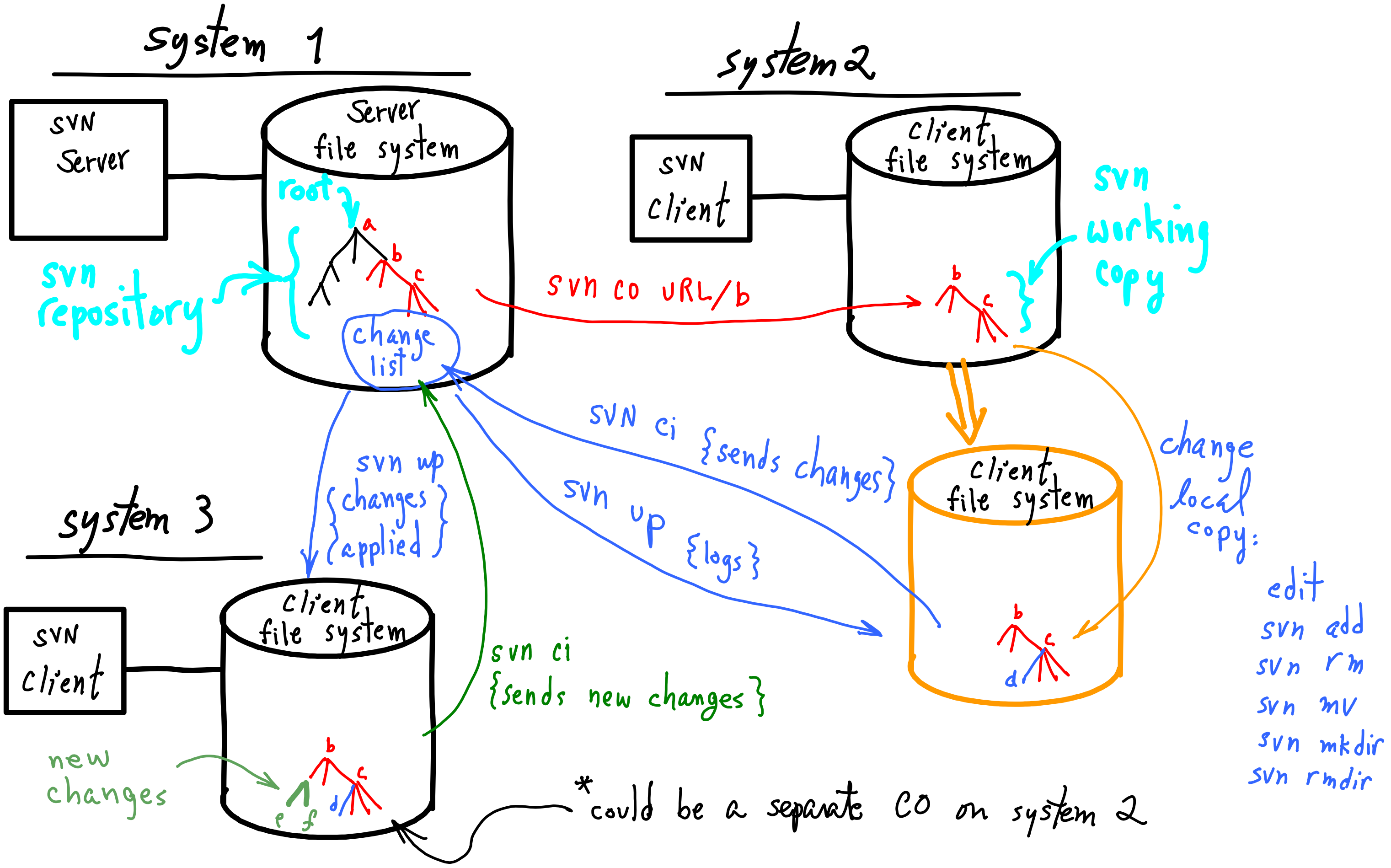
```
%> set                         #--- see all environment variables
%> echo $PATH                  #--- see PATH variable content ( w/ ":" separators for sub-strings)
%> cd                          #--- your home directory in unix/cygwin environments
%> vi .bash_profile
      export VISUAL="vi"       #--- needed for "svn ci" to edit log comments
%> ls                          #--- see files in current directory
%> cd foo; cd ..               #--- move in file system tree
%> mkdir; rmdir                #--- add/subtract sub-tree
%> rm                          #--- remove file, forever
%> pwd                         #--- see shell's idea of current position in file system
%> exit                        #--- kills current shell, returns to parent process
%> tar -xvf foo.tar            #--- unpack a tree
%> gunzip foo.tar.z            #--- uncompress a packed tree or file
%> man ls                      #--- see how to use the "ls" program
%> info ls                     #--- also see "ls" usage (more complete?)
%> alias l "ls -F"             #--- make shorthand for a command
%> ps -ex                      #--- see all running/sleeping processes
%> kill -9 12345               #--- send a signal to process 12345 that kills it
%> jobs                        #-- see current jobs that are asleep
%> ^z                          #--- put jobs to sleep (e.g., editing session), return to parent
%> fg                          #--- wake up most recently slept process
%> %2                          #-- wake up job 2
%> cat foo                     #--- dump file content to stdout
%> cat foo bar > foobar        #---send content of foo and bar to file "foobar"
%> cat foo bar | grep "who"    #--- send content to grep via stdin for subprocess
%> less foobar                 #--- see content a screenfull at a time
%> make target                 #--- read Makefile, find target, execute shell commands
%> cat foobar | sed 's/Hi/hi/' #--- stream editing, by lines
%> awk                         #-- more stream editing, by fields per line
%> m4                          #--- input stream macro expansion
%> cpp                         #--- input macro expansion, C preprocessor ("#define", e.g.)
%> rm -rf workDir              #--- destroy/remove entire local tree, including ".svn" sub-trees
%> cp foo ../bar/              #--- copy file or dir
```

SUBVERSION

Repository exists on svn server.

-- **svn co** https:URL/dir (get a "working copy" of subtree)
-- **svn ci** (send local changes to repository)
-- **svn up** (get changes from repository)
-- **svn -v log** (see **svn ci** log messages for subtree)
-- **svn help** (see list of commands and instructions)

-- SVN commands apply to **current subtree**.
-- simultaneous, mulitple working copies.
-- **svn co -r123** https:URL/dir
(checkout prior version)
-- **svn status** (check for local changes)



svn add foo #--- mark file or directory "foo" to be added to repository
svn rm foo #--- deletes foo and schedules delete from repository
svn mv foo bar #--- deletes foo and adds bar
svn status #--- see state of working copy
"?" unknown to svn, not part of repository.
"M" modified, changes will be sent at next ci
"A" will be added to repository
"D" will be deleted from repository
"!" missing locally, but in repository
"C" conflicts: edits overlap prior checked-in changes

svn copy URL/dir1 URL/dir2 #-- Start a new development branch: makes a copy of subtree, and starts new changelist.

svn merge (to join parallel trees)

Read Documentation in projects/LC3trunk/docs

Read the READMEs. Use a Web browser, download individual files:

NB--Browser DOES NOT create a local working directory or files:
you cannot
----- check-in/commit changes, **svn ci**
----- get updates, **svn up** (instead, re-download to get newest version)

====== Subversion (SVN) Clients, installation ==============

Subversion consists of **two parts, a server, and a client**. You only need a client. Most downloads will include a server, but **you do not need to setup the server**.

Is a **commandline client svn** already installed as part of your OS?
If not, **is an executable binary available?** (Rather than downloading source code and building.)

--- **Mac OSX** 10.5 and later: use the terminal app.
Get XCode (older ones are free), see Apple Developer Connection.
(See MacPorts.org to download binaries not included in XCode, if needed.)

--- **Windows: Avoid binaries for gui svn clients** on the subversion web site.
You need a **unix interface to windows** anyway for iverilog; so, you should **install cygwin**:

http://www.cygwin.com/

setup.exe ===> Lots of **selections** you can make

--- Base:	gzip, grep, sed, tar, which
--- Devel:	gdb, make, subversion
--- Editors:	emacs, vim
--- Net:	openssl

First install: **take all defaults**
rerun later: **select things to add**

CYGWIN users, SEE "A Note on Windows and Cygwin directory structure" below.

Altering SVN Tree

NO: SVN IMPORT!!!

- remove → svn rm
- move → svn mv
- add → svn add
- svn status
- svn help

NO: ADDING TEMPORARIES

- executable binaries
- Electric's .v files
- debug output?

rm: if you delete a file w/o using **svn rm**, svn will think the file is missing and will restore it when you next "svn up".

mv: if you rename a file w/o using **svn mv**, svn will think it is new (and the old one missing). NB--**svn mv** will appear as a svn Delete/Add pair.

add: if you want something to become part of your repository **svn add**.

Do **svn status** before doing **svn ci** (committ). It tells you what the next **svn ci** will do:

```
"?" file (or dir) is unknown, nothing will be done.
"M" file/dir is modified, changes will be sent.
"A" file/dir will be added to repository
"D" file/dir will be deleted from repository
"C" Conflict: you tried to commit changes that overlapped
        with other changes already committed.
```

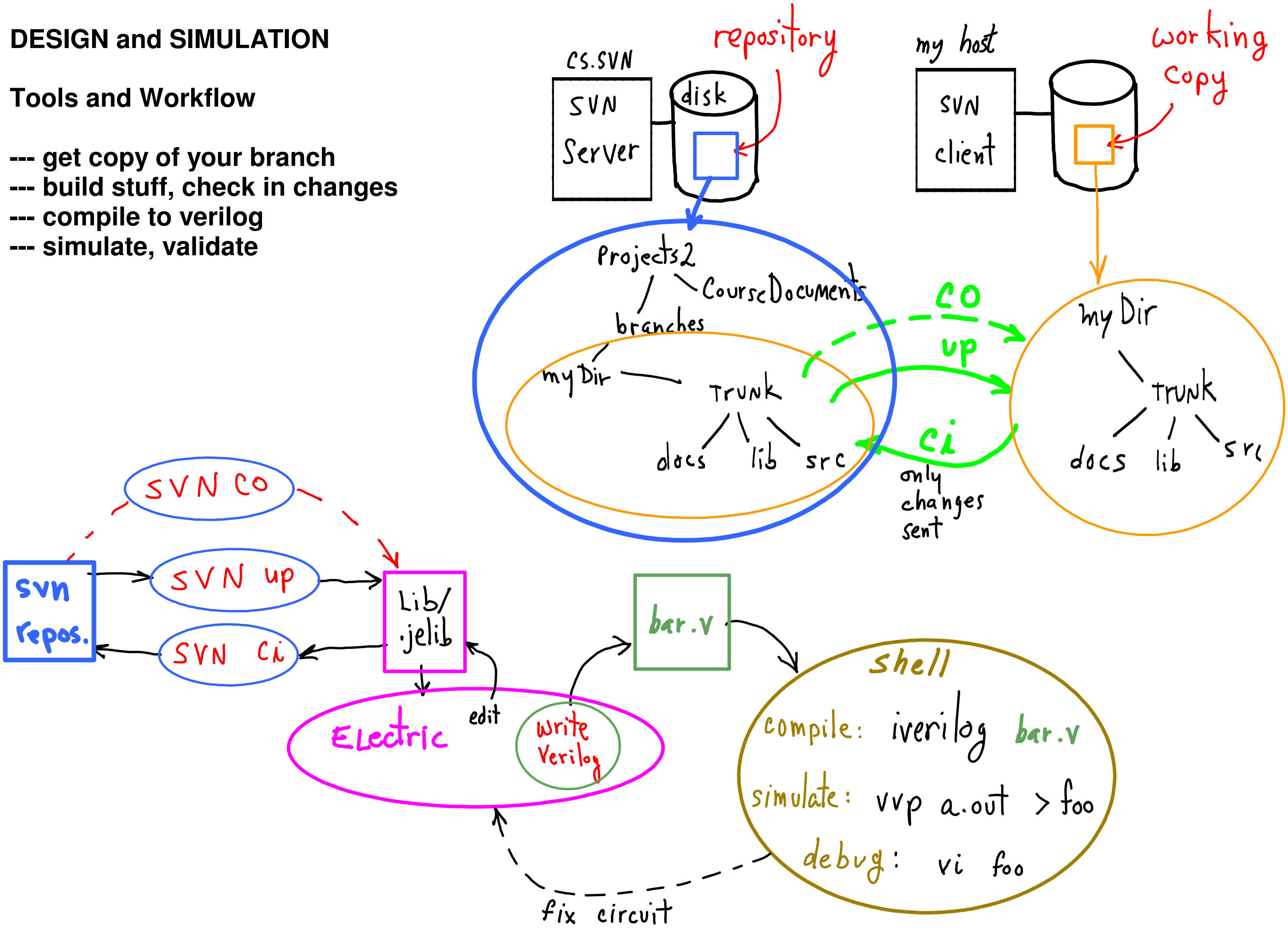
If your local copy is confused, you can completely erase it locally,
/bin/**rm** -rf myDir
then re-checkout. If you have altered files, put them in a safe place first, then do **rm**, then move them into your new working copy.

- Use Web access for downloading anything not in your own subtree of repository.
  — Safest
- Checkout tree, but never svn ci except in your subtree
  — Handy: you get updates to docs, etc.
- **myDir/.svn** — Subversion keeps track of
  ~/.subversion
  — repository address
  — authentification
  — files/dirs changes/status

DESIGN and SIMULATION

Tools and Workflow

--- get copy of your branch
--- build stuff, check in changes
--- compile to verilog
--- simulate, validate



Workflow:

--- **Electric.File.OpenLibrary** "myDir/trunk/lib/foo.jelib"
===> open lib files, then make changes.

--- **in terminal window**:

```
cd myDir
svn ci          (write good comments in commit window.)
```

--- **Electric.Tools.Simulation.WriteVerilogDeck**
====> "myDir/trunk/run/foo.v"   (create verilog file from design)

--- **in terminal window**:

```
cd myDir/trunk/run
iverilog foo.v                 (compile verilog)
vvp a.out  >  foo_output       (simulate)
vi foo_output                  (check results)
```

--- go back to Electric, revise design.

Electric

projects/LC3-tools/electricBinary.jar

projects/LC3-trunk/examples/tutorial.jelib

--- Get tutorial.jelib
use Web browser
download into your branch
svn add

--- Open tutorial.jelib
start ElectricBinary.jar
^File.OpenLibrary

--- See Documentation
Electric.LeftPanel:
^Explorer tab
^^0AAA-ReadMe{doc}
also see text boxes in schematics:
^^reg{sch}
^^regUsage{sch}

--- Create a cell
^Cell.NewCell
set cell properties:
Library[tutorial]
Name: __________
Type[schematic]

--- Place Blackbox in cell:
^Components.Schematic.{Black box}
^Components.Schematic.Misc.VerilogCode

--- Extract verilog code
^Tools.Simulation.(WriteVerilogDeck)

also, see (Place Cell)

(edit code box) Edit.Properties

Electric's names versus Verilog's names

- schematic cell vs. icon cell

Design is in a schematic cell: **foo{sch}**
Icon has its graphical design in icon cell: **foo{ic}**
Hierarchy: place icon **foo{ic}** into **bar{sch}**

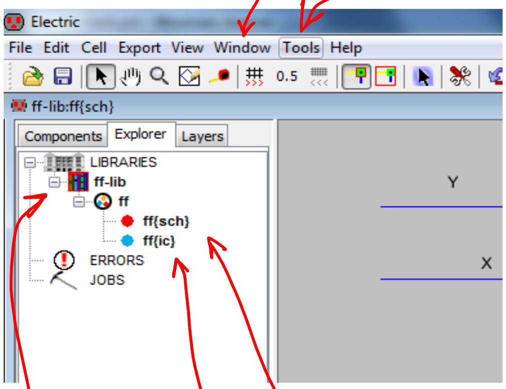
Electric		Verilog
schematic cell	↔	class/module def
schematic exports	↔	module parameters
icon instance	↔	module instance
Wires To icon exports	↔	instance's args

- Electric : schematic cell "sch" has a name.
- Verilog : module uses cell's name
- Electric : each icon instance has its own name
- Verilog : each module instance has icon instance name.
- Hierarchy : Electric's Exports = Verilog's args list (ports)

Verilog

modules = classes (except top-level module from top-level cell) ← like Java main class or C's "main()", aka "testbench" zoom

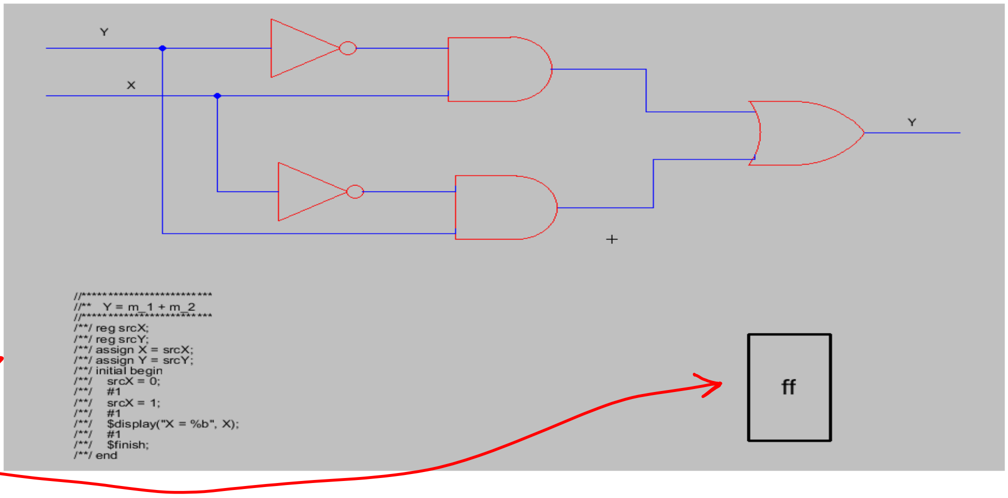
simulation (Verilog). Write Verilog Deck


Electric
File Edit Cell Export View Window Tools Help
0.5
ff-lib:ff{sch}
Components Explorer Layers
LIBRARIES
ff-lib
ff
ff{sch}
ff{ic}
ERRORS
JOBS
Y
X

library

cell {sch}

cell {ic}



```
//*************************
//**   Y = m_1 + m_2
//*************************
/**/ reg srcX;
/**/ reg srcY;
/**/ assign X = srcX;
/**/ assign Y = srcY;
/**/ initial begin
/**/    srcX = 0;
/**/    #1
/**/    srcX = 1;
/**/    #1
/**/    $display("X = %b", X);
/**/    #1
/**/    $finish;
/**/ end
```

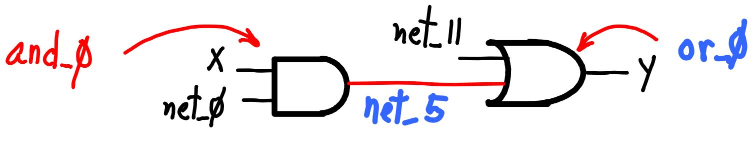
```
/* Verilog for cell 'ff{sch}' from library 'ff-lib' */
/* Created on Fri Jan 18, 2013 11:51:35 */
/* Last revised on Fri Jan 18, 2013 12:12:05 */
/* Written on Fri Jan 18, 2013 12:18:34 by Electric VLSI Design System, version 9.03 */

module ff();
  /* user-specified Verilog code */
  //*************************
  //**    Y = m_1 + m_2
  //*************************
  /**/ reg srcX;
  /**/ reg srcY;
  /**/ assign X = srcX;
  /**/ assign Y = srcY;
  /**/ initial begin
  /**/    srcX = 0;
  /**/    #1
  /**/    srcX = 1;
  /**/    #1
  /**/    $display("X = %b", X);
  /**/    #1
  /**/    $finish;
  /**/ end

  wire X, Y, and_0_yc, and_0_yt, and_2_yc, and_2_yt, buf_0_c, buf_1_c, net_0;
  wire net_11, net_5, net_6, or_0_yc, or_0_yt, pin_16_wire;

  and and_0(net_5, net_0, X);
  and and_2(net_11, net_6, Y);
  not buf_0(net_0, Y);
  not buf_1(net_6, X);
  or or_0(Y, net_11, net_5);
endmodule   /* ff */
```

Electric { Trims redundant parts.
also, produces unused wires,
sometimes?

Electric makes up instance names, if none assigned.

net_5 is a wire instance connected to output of and_0, an instance of and, and connected to input of instance or_0

class/module



- port connections:

selecting ports, placing wire

- creating ports: - characteristic [input/output]
  - name

Create Export (aka, a "port"):

--**Place a pin** into **foo{sch}** (wire/bus)
--**Select the pin**

^Export.CreateExport

--**Fill in properties** (name, input/output)

Change existing export's properties:

Select export's text (not pin)
(shows highlighted X across pin)

Edit.Properties.ObjectProperties

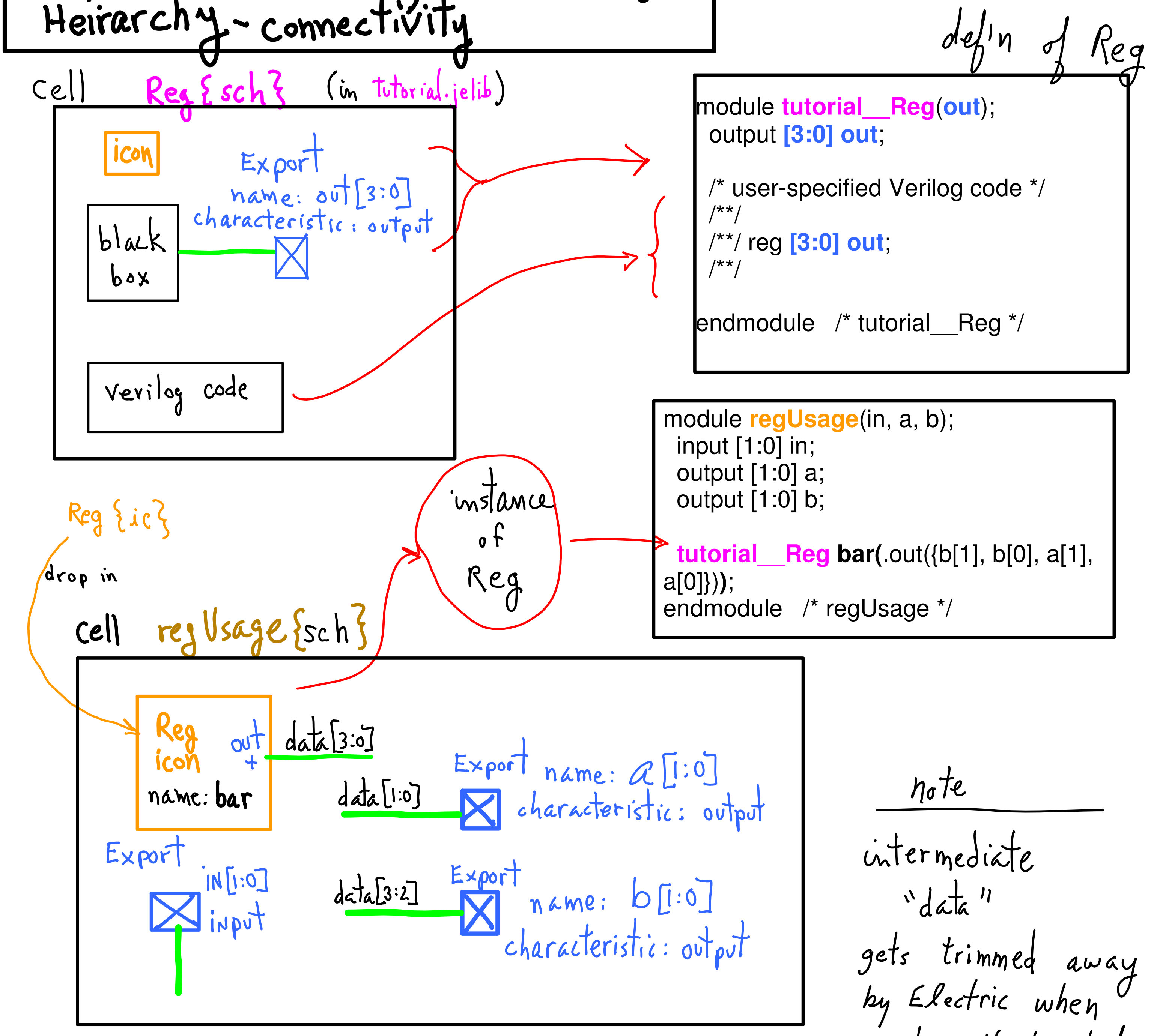
wires go from pin to pin.
Busses are collections of wires.
Busses go from bus-pin to bus-pin

Electric.Components.{wire pin}
" " .{wire}
" " .{bus pin}
" " .{bus}

- wires use pins, wire from selected pin to {cell area}^ or "+"^
- busses
  - concatenation *examples* Tutorial.jelib/RegUsage
  - sub-bus connections
  - connect via naming
  - Bus naming: foo[3:0], 4-wire bus

Verilog
- wire naming and connections in module instantiation arg list.
  - connect by position in arg list: and and_0(Y, A, B);
  - or by name: and and_1(.in0(A), .in1(B), .out(Z));
- wire, reg, input, output [designations] (size parameters)

Exports / verilog module args
Heirarchy - connectivity



```
module tutorial__Reg(out);
  output [3:0] out;

  /* user-specified Verilog code */
  /**/
  /**/ reg [3:0] out;
  /**/

endmodule   /* tutorial__Reg */
```

```
module regUsage(in, a, b);
  input [1:0] in;
  output [1:0] a;
  output [1:0] b;

  tutorial__Reg bar(.out({b[1], b[0], a[1],
a[0]}));
endmodule   /* regUsage */
```

note

intermediate "data" gets trimmed away by Electric when creating Verilog deck

in Electric	equivalent in Verilog
Reg{sch} Export, **out**[3:0], **output**	module tutorial_**Reg**(**out**) **output** [3:0] **out**
instance of Reg{ic} named **bar**	tutorial_Reg **bar**( )
bar.**out**[1:0]-to-**a**[1:0] connection	**.out**(**{** ..., **a**[1], **a**[0] **}**)
equivalent syntax	..., .out[1](a[1]), .out[0](a[0])

Bus Concatenation

The connections between levels in a hierarchy are expressed as "Exports" in Electric and as args in Verilog. Electric trims away redundant wires; so, the busses dissappeared in the Verilog code.

Seeing Verilog results

- Verilog compilation + simulation [discrete event]

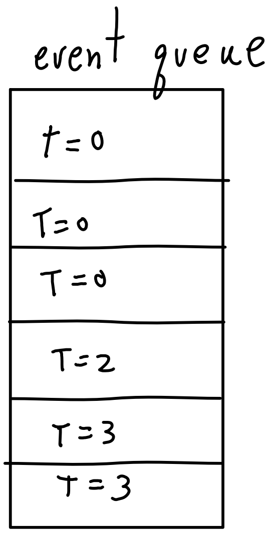
```
Electric.Tools.Simulation(Verilog).WriteVerilogDeck
iverilog foobar.v
vvp a.out > foo
make debug
```

⇒ save as foobar.v

⇒ saves simulation output

⇒ see contents of foo, foo2, foo3

or
(just look at foo w/ editor)



Least time-stamp simulation event pulled from queue, executed, new events posted to queue.

- Verilog
  - — getting something to happen —
    - initial begin . . . end
    - always begin . . . end (NB 0-delay → ∞ loop)
  - — stopping:

    ```
    $finish;
    ```

  - — Seeing what happened:

    ```
    $display("%d", $time);
    $write("hi");
    ```

    ↖ no eoln

    when? delay to see results.

  - documentation: see

    docs/verilog

    (or google "verilog tutorial")

LC3 system

- projects/trunk/
- lib/system.jelib.top{sch}

Top-level cell of LC3

- test.jelib.top_rtL_test

Top-level cell testbench has instance of system.top, a "main" for simulation.

- system.jelib.top.showRegs

a "task" can be called in verilog code.

Task def'n

task f; begin ... end endTask

Task invocation

f;

MAKE and svn up

Keeping up-to-date with CourseDocuments:

```
svn co  URL/520-2013/CourseDocuments/
```

URL=https://svn.cs.georgetown.edu/svn/projects2

Creates a working copy of the CourseDocuments subtree on your machine.

Update periodically,

```
cd 520-CourseDocuments
svn up
```

MAKE can be a handy way of keeping commands and executing them. For example, here is a possible **Makefile** (see below for notes on syntax):

```
#------------------
#-- Makefile
#----------------
URL=https://svn.cs.georgetown.edu/svn/projects2/520-2013
AUTH= --username 250-374-developer  --password 'y(&qwqsq'
doCO::
        svn co  $(URL)/CourseDocuments/     520-CourseDocuments  \
                    $(AUTH)
doUP::
        cd 520-CourseDocuments; svn up  $(AUTH)
#------ Makefile END
```

Next, use these **unix commands**,

```
make doCO
make doUP
```

I find this very handy. Also, if you are new to unix and/or make, it is a good way to get started.

Makefile syntax:

--- **Makefile targets** are "doCO" and "doUP".
Make will look for the target "doCO" in the local Makefile.

:: or : means, **do the following commands** for this target.
The next line, **the command**, **MUST start with a TAB** character.

--- Makefile **commands** are **shell commands**
Executed as if you had typed into the console.

Command must be all **on one line**.

But, if the command is long, **use `\`** at the **end of each line**.
Means: "Please ignore the end-of-line character and consider this to be all on one line."

--- Makefile **variables**

Assignment is the same as **shell syntax**
"**FOO=abc**" assigns the string "abc".

`$(FOO)` or equivalently **`${FOO}`** is replaced with the value "abc".

--- **Multiple commands** for a single target. Each command is on a separate line. E.g.,

```
doUP::
      echo "Doing an svn update"
      svn up
```

Each line forks its own shell to execute the commandline.
This will not do what you might expect:

```
doUP::
      echo "Doing an svn update"
      cd 520-CourseDocuments
      svn up  $(AUTH)
```

Forks three shells, one for each commandline.
2nd shell does **cd and exits**.
3rd shell does **not execute in 520-CourseDocuments/** .

BUT, a **";" separated list of commands is a "list command".**
Forks **one shell** to **execute the list**.

```
cd 520-CourseDocuments; svn up
```

The parent shell **executes "cd"** as a built-in **without forking a child process**.
Then **forks a process to do svn**
"svn" process **inherits the current working directory** from its parent.

A Note on Windows and Cygwin directory structures.

For Windows systems, cygwin and Windows do not agree on the shape of the directory tree of the entire file system. For Windows, the actual root is "C:\", e.g., if you are using your C: drive. Cygwin is usually installed in C:\cygwin\ with your unix home below there. To get to the Windows root, C:\, using cygwin, do this,

```
    cd /cygdrive/c/
```

Note that you have two home directories: (1) your cygwin home which is in cygwin's /home/, and your Windows home, which is probably in,

```
    /cygdrive/c/Users/
```

It can get confusing. It is best to keep your work in your unix home directory which is under /home.