ECE 4750 Linux, Git, PyMTL, Verilog Cheat Sheet

Linux Commands

`man command`	display help for given `command`
`echo "string"`	display given `string`

`echo "string" > file`	create `file`
`cat a`	display file `a`
`less a`	display file `a` with paging and search
`ls`	list contents of current working dir
`ls -la`	list contents of current working dir (verbose)
`ls A`	list contents of dir `A`
`ls *.txt`	list files with `.txt` suffix in current working dir

`pwd`	display current working dir
`mkdir A`	make dir `A` to `B`
`mkdir -p A/B`	make all dirs in path `A/B`
`cd A`	change current working dir to `A`
`cd ..`	change current working dir to parent dir
`cd ~`	change current working dir to home dir
`tree`	recursively list contents of current working dir

`cp a b`	copy file `a` to `b`
`cp -r A B`	copy dir `A` to `B`
`mv a b`	move file `a` to `b`
`mv A B`	move dir `A` to `B`
`rm a`	remove file `a`
`rm -r A`	remove dir `A`

`wget url`	download file at `url`
`grep "string" a`	search file `a` for given string
`grep -r "string" A`	recursively search files in dir `A`
`find . -name "string"`	find files named `string` in dir `.`
`tar -czvf a.tgz A`	create archive `a.tgz` of dir `A`
`tar -xzvf a.tgz`	extract archive `a.tgz`
`top`	view what is running on system

`ENVVAR="string"`	set environment variable
`echo ${ENVVAR}`	display given environment variable
`cmd > a`	redirect output of `cmd` to newly created file `a`
`cmd >> a`	redirect output of `cmd` to append to file `a`
`cmd_a && cmd_b`	execute `cmd_a` and then execute `cmd_b`
`cmd_a | cmd_b`	send output from `cmd_a` to `cmd_b`

`source setup-ece4750.sh`	source setup script for course
`quota`	check disk usage
`trash`	move file to `${HOME}/tmp/trash`

Git Commands

`help cmd`	display help on git command `cmd`
`clone url`	clone repo at given URL
`add a`	add file `a` to index
`add A`	add directory `A` to index
`commit`	commit indexed files
`commit -m "msg"`	commit indexedfiles w/ commit `msg`
`commit -a -m "msg"`	commit tracked files w/ commit `msg`
`log`	show history log of previous commits
`status`	show status of local repo
`checkout a`	revert file `a` to last commit
`checkout A`	revert dir `A` to last commit
`pull --rebase`	pull remote commits to local repository
`push`	push local commits to remote repository
`whatchanged`	show incremental changes for each commit

`xstatus`	compact status display
`xlog`	compact log display
`xadd`	add all tracked, modified files to index
`xpull`	short for `pull --rebase`

ECE 4750 Lab Management Script

`ece4750-lab-admin`

`--help`	display help
`--join-class <githubid>`	join class on GitHub
`--status`	check group status

`--make-request group-create`
create a new group, use `--status` to find group num

`--make-request group-join <groupnum>`
join group with given group num

`--make-request group-approve <netid>`
approve student with given NetID to join group

`--make-request group-leaave`
leave current group

`--cancel-request`	cancel pending request

Pending requests usually take one or two minutes to be accepted by the system, and updates to GitHub can take 5–10 minutes. Please be patient!

Remote Login to `ecelinux` Servers

```
ssh -X <netid>@ecelinux-01.ece.cornell.edu
ssh -X <netid>@ecelinux-02.ece.cornell.edu
```

GitHub and TravisCI URLs

```
http://github.com/cornell-ece4750/lab-groupXX
http://magnum.travis-ci.com
         /cornell-ece4750/lab-groupXX
```

Example Development Session

```
% source setup-ece4750.sh
% cd ${HOME}/ece4750
% git clone \
   git@github.com:cornell-brg/ece4750-tut3-pymtl tut3

% mkdir -p tut3/sim/build
% cd tut3/sim/build
% py.test ../tut3_pymtl/gcd
% py.test ../tut3_pymtl/gcd --verbose
% py.test ../tut3_pymtl/gcd -x -s --tb=long

% py.test ../tut3_pymtl/gcd/test/GcdUnitRTL_test.py
% py.test ../tut3_pymtl/gcd/test/GcdUnitRTL_test.py \
   -k basic_0x0 --dump-vcd
% gtkwave *.vcd &

% ../tut3_pymtl/gcd/gcd-sim --help
% ../tut3_pymtl/gcd/gcd-sim --impl rtl \
    --input random --stats --trace --dump-vcd
% gtkwave gcd-rtl-random.vcd &
```

ECE 4750 Linux, Git, PyMTL, Verilog Cheat Sheet

RegIncr.py

```python
#====================================
# Registered Incrementer
#====================================

from pymtl import *

class RegIncr( Model ):

  def __init__( s ):

    s.in_ = InPort  ( Bits(8) )
    s.out = OutPort ( Bits(8) )

    # Sequential logic

    s.reg_out = Wire( Bits(8) )

    @s.tick
    def block1():
      if s.reset:
        s.reg_out.next = 0
      else:
        s.reg_out.next = s.in_

    # Combinational logic

    @s.combinational
    def block2():
      s.out.value = s.reg_out + 1
```

Example Line Trace for GCD

```
   src     in     A  B  ST out  sink
 1: 0f05 > 0f:05(xx xx I )   >
 2: #      > #    (0f 05 C-)   >
 3: #      > #    (0a 05 C-)   >
 9: #      > #    (05 05 C-)   >
10: #      > #    (00 05 Cs)   >
11: #      > #    (05 00 C )   >
12: #      > #    (05 00 D )05 > 05
```

Legend for val/rdy interfaces in line traces

spaces	`!valid && ready`
`#`	`valid && !ready`
`.`	`!valid && !ready`
`msg`	`valid && ready`

RegIncr.v

```verilog
//====================================
// Registered Incrementer
//====================================

`ifndef REGINCR_REG_INCR_V
`define REGINCR_REG_INCR_V

module tut4_verilog_regincr_RegIncr
(
  input  logic       clk,
  input  logic       reset,
  input  logic [7:0] in,
  output logic [7:0] out
);

  // Sequential logic

  logic [7:0] reg_out;
  always @( posedge clk ) begin
    if ( reset )
      reg_out <= 0;
    else
      reg_out <= in;
  end

  // Combinational logic

  logic [7:0] temp_wire;
  always @(*) begin
    temp_wire = reg_out + 1;
  end

  // Combinational logic

  assign out = temp_wire;

endmodule

`endif /* REGINCR_REG_INCR_V */
```

Valid PyMTL in Concurrent Blocks

```
Bits BitStruct
| & ^ ~ and or not
== != <= < >= > << >> + -
s.signal[n], s.signal[n:m]
sext(), zext(), concat()
reduce_and(), reduce_or()
reduce_xor()
if, else, elif
```

Coding Conventions

- Try to keep lines less than 74–80 chars
- Include header comment at top of each file
- Include comments to explain code
- Use only spaces, no tabs; use two-space indentation
- Use `CamelCase` for model/module names
- Use `under_scores` for var, port, instance names
- Use `clk` and `reset` for clock and reset
- Use informative variable names
- Separate sequential from combinational logic
- FSMs have 3 blocks: state, transitions, outputs
- Datapaths use structural composition

PyMTL Specific

- Use s instead of `self`
- All wires and modules must be members of parent model (i.e., `s.wire`, `s.model`)
- Only use `.next` in `s.tick_rtl`
- Only use `.value` in `s.combinational`
- Only signals (e.g., `InPort`, `OutPort`, `Wire`) can be used to communicate between concurrent blocks
- Prefer bit operators (`|`, `&`, `^`) over `or`, `and`, `not` in boolean logic equations

Verilog Specific

- Use subdir prefix for macro and module names
- Explicitly include file dependencies
- Use include guards
- Use `ALL_CAPS` for macro names
- Use p_ prefix for parameters
- Use c_ prefix for constants
- Use localparams for local constants
- Always declare type of all ports
- Align port declarations, one per line
- Align port connections, one per line
- Align parameter config, one per line
- Avoid positional port connections & param config
- Only use <= in `always_ff`, = in `always_comb`
- Prefer ternary operator over `if`
- Prefer `case` over `casez`
- Use tasks for compact state output table

Synthesizable Verilog Keywords

`logic`
`logic [N-1:0]`
`& | ^ ^~ ~` (bitwise)
`&& || !`
`& ~& | ~| ^ ^~` (reduction)
`+ - >> << >>> == != > <= < <=`
`{}`
`{N{}}` (repeat)
`?:`
`always_ff, always_comb`
`if, else`
`case, endcase`
`begin, end`
`module, endmodule`
`input, output`
`assign`
`parameter`
`localparam`
`genvar`
`generate, endgenerate`
`generate for`
`generate if else`
`generate case`
named port connections
named parameter passing

Synthesizable Verilog Keywords with Limitations

`always`
`enum`
`struct`
`casez, endcase`
`task, endtask`
`function, endfunction`
`=` (blocking assignment)
`<=` (non-blocking assignment)
`typedef`
`packed`
`$clog2()`
`$bits()`
`$signed()`