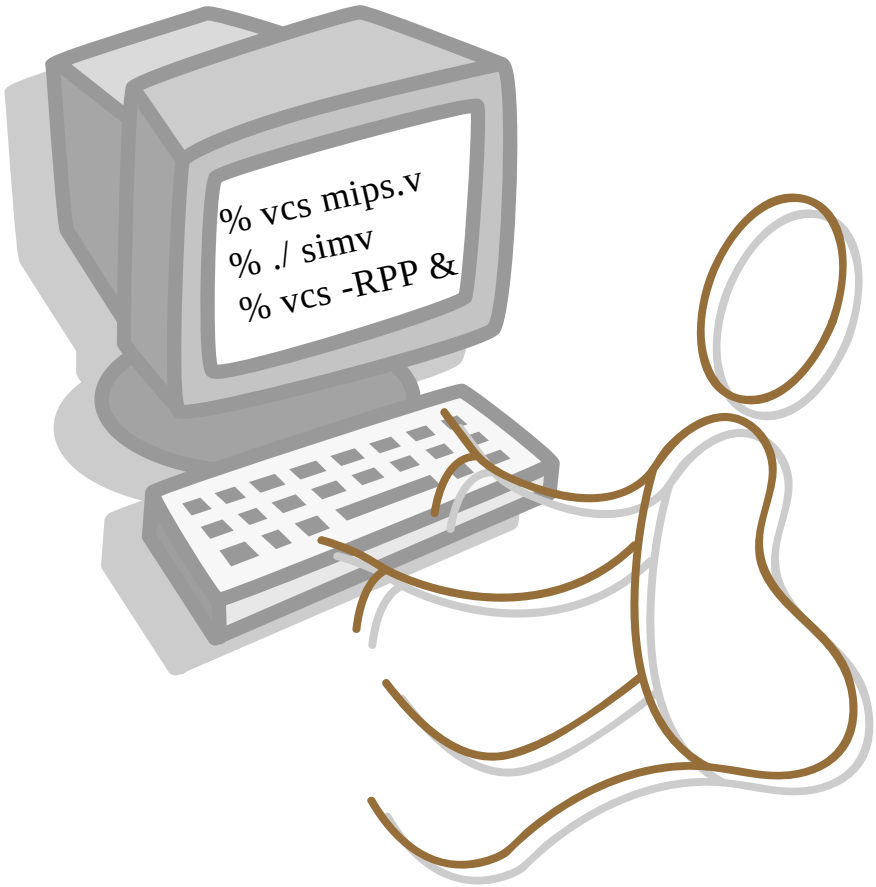
Tutorial #2

Verilog Simulation Toolflow



Tutorial Notes Courtesy of Christopher Batten

i ure y T C

A Hodgepodge Of Information

- C sour e ana e ent syste
- Bro sin a C repository ith ie s
- a efile erilo uil syste
- ritin asse ly
- sin the isasse ler
- sin tra e output instea of a efor s

Concurrent Versions System

- **central repository** ontains all erilo o e as ell as infor ation on ho han e hat an hen
- sers **checkout** a opy of the erilo o e e it it an then **commit** their o ifie ersion
- sers an see hat has han e to help tra o n u s an this allo s ultiple users to or on the sa e erilo o e at the sa e ti e
- ur repository is at it sroot ut you shoul ne er a ess the repository ire tly nstea use C o an s of the follo in for

```
% cvs <commandname>
% cvs help
```

6.884 CVS Repository

There are three pri ary types of top le el ire tories in the repository

- a ples e eryone has a ess
- n i i ual ire tories only you ha e rea rite
- ro e t ire tories e eryone has a ess

To he out the e a ples an try the out use

```
% cvs checkout examples
```

To he out your in i i ual ire tory use

```
% cvs checkout <MIT server-username>
```

CVS Basics

Common CVS commands

- `cvs checkout pname` — Checkout a working copy
- `cvs update pname` — Update working dir vs. repos
- `cvs commit [filelist]` — Commit your changes
- `cvs add [filelist]` — Add new files/dirs to repos
- `cvs diff` — See how working copy differs

Set the CVSEDITOR environment variable to whichever editor you use when writing log messages

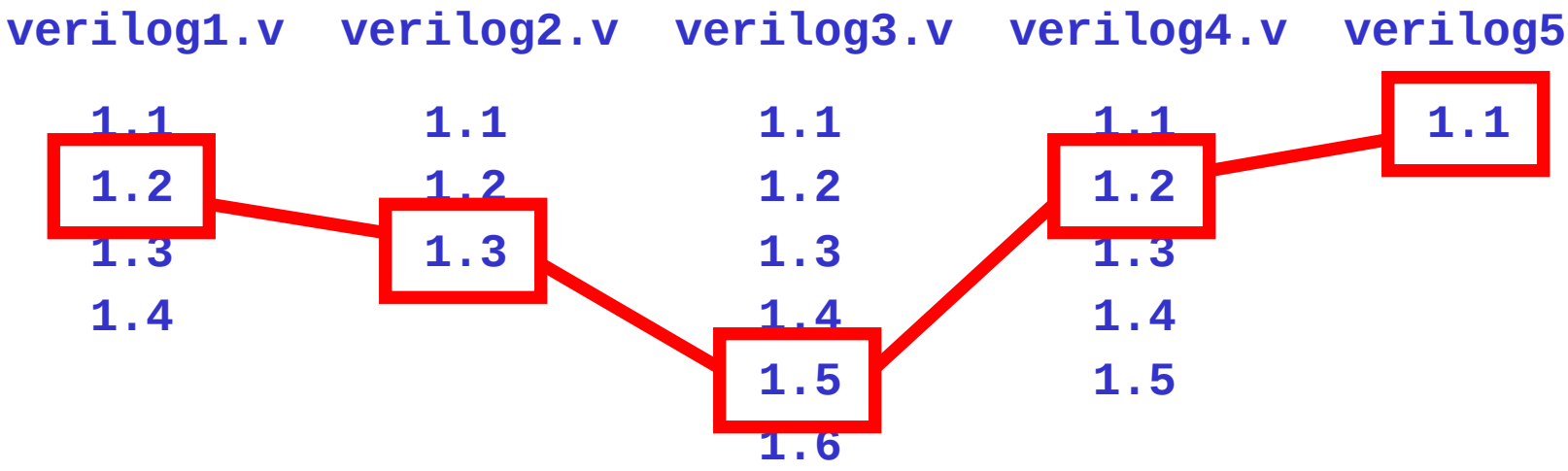
6.884 CVS Repository

se the follo in o an s to he out the ips sta e test harness test pro ra s et an then put the into your o n s ire tory

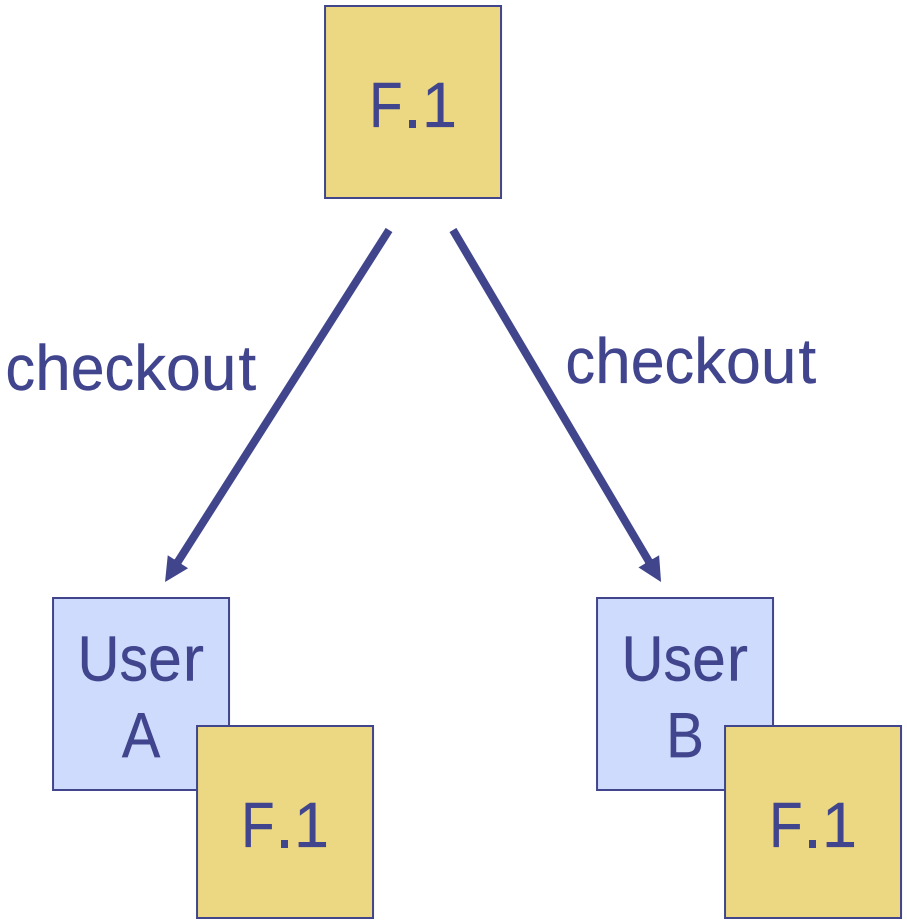
```
% cvs checkout 2005-spring/cbatten
% cd 2005-spring/cbatten
% cvs export –r HEAD examples/mips2stage
% mv examples/mips2stage .
% rm –rf examples
% find mips2stage | xargs cvs add
% <start to make your additions>
% cvs add [new files]
% cvs update
% cvs commit
```

Using CVS Tags

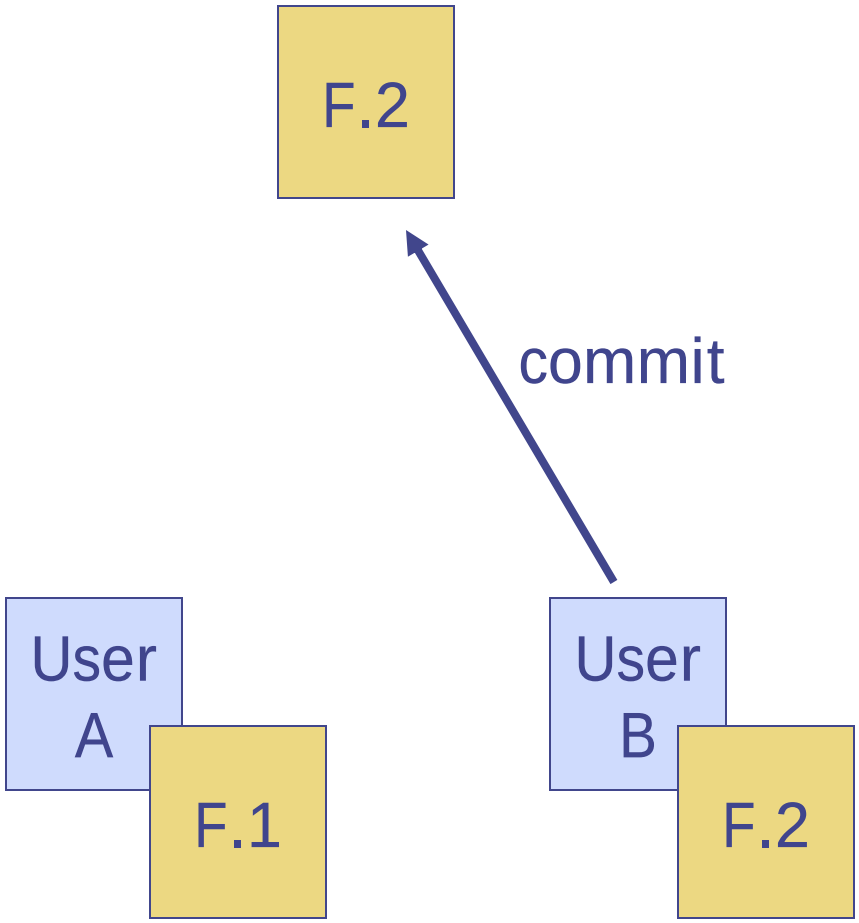
y oli ta s are a ay to ar all the files in your pro e t at a ertain point in the pro e t e elop ent ou an then later he out the hole pro e t e a tly as it e iste pre iously ith the ta



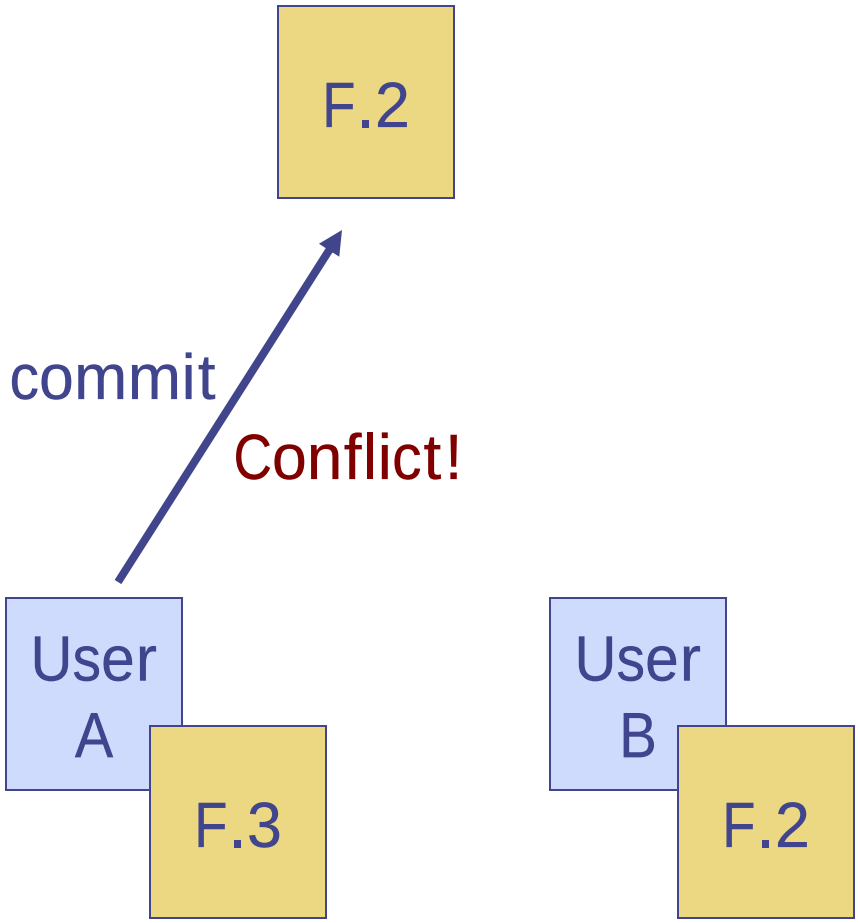
CVS – Multiple Users



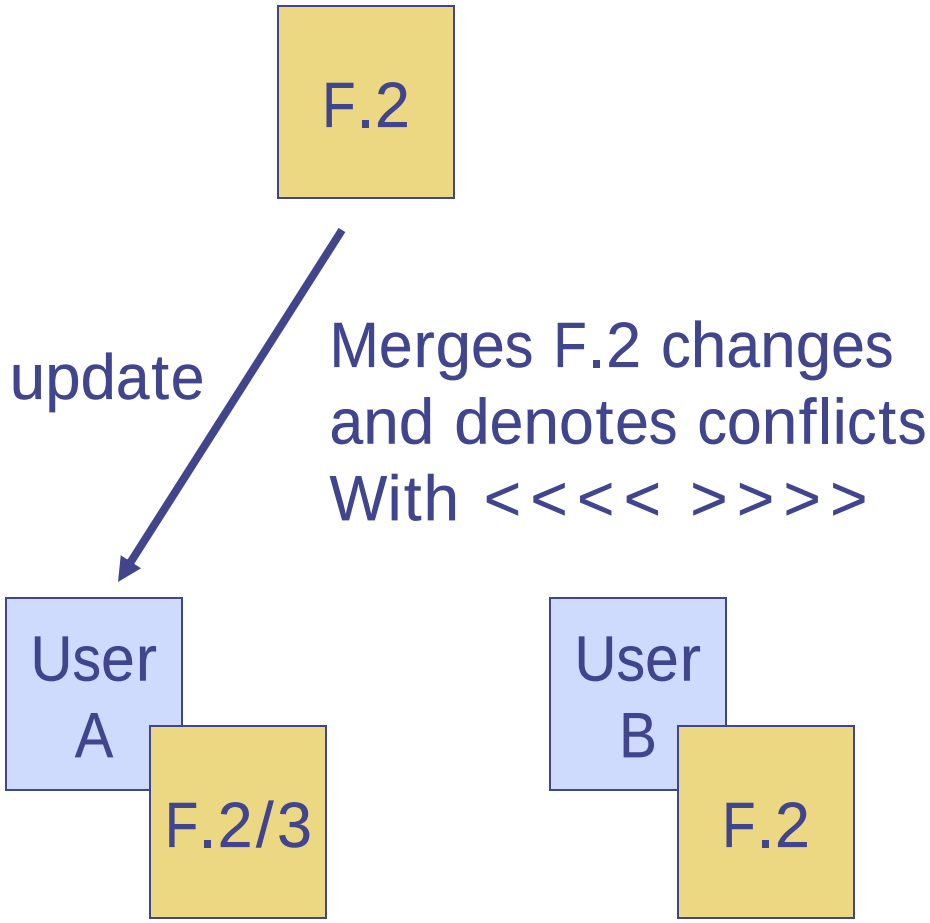
CVS – Multiple Users



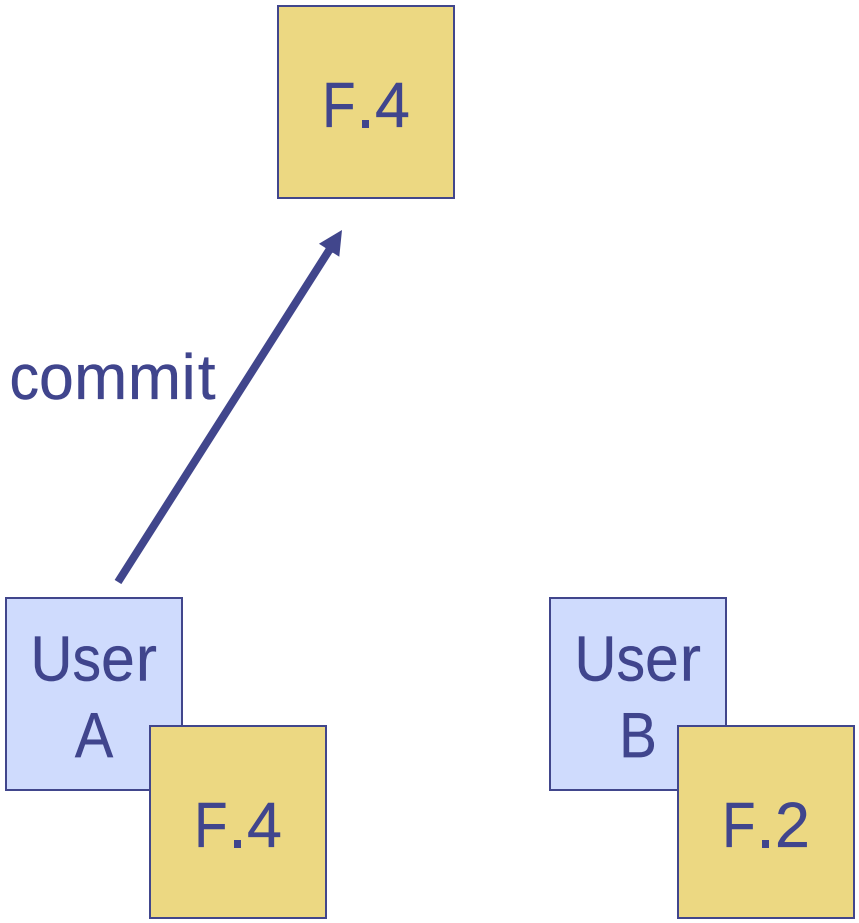
CVS – Multiple Users



CVS – Multiple Users

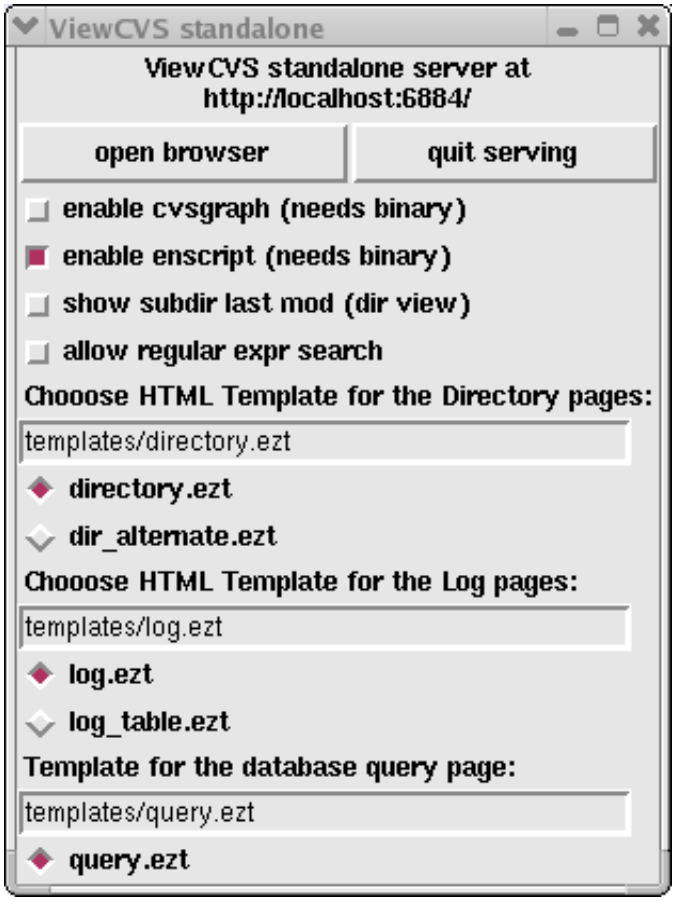


CVS – Multiple Users



Use Viewcvs for Repository Browsing

ie s is a on enient tool for ro sin the s repository throu h a e interfa e se the start ie s o an to start the ie s e ser er then point your lo al ro ser to http lo alhost



mkasic.pl → Makefiles

hy not use a efiles to start out ith

epen en y tra in is less ne essary

iffi ult to i ple ent so e operations

hy are e han in no

a efiles are ore fa iliar to any of you

epen en y tra in ill e o e ore useful ith the a ition of test inary eneration an Bluespe o pilation

Using the Makefiles

Create a uil ire tory then use onfi ure pl to reate a a efile an then use arious a e tar ets to reate arious enerate pro u ts

nli e asi pl e ill e pla in u h ore enerate pro u t ire tly in the uil ire tory

```
gcd/
 Makefile.in
 configure.pl
 verilog/
   gcd_behavioral.v
   gcd_rtl.v
 config/
   gcd_behavioral.mk
   gcd_rtl.mk
 tests/
   gcd-test.dat
```

```
% cvs checkout examples
% cd examples/gcd
% mkdir build-gcd-rtl
% cd build-gcd-rtl
% ../configure.pl ../config/gcd_rtl.mk
% make simv
% ./simv
% vcs –RPP &
```

Modifying the config/*.mk Files

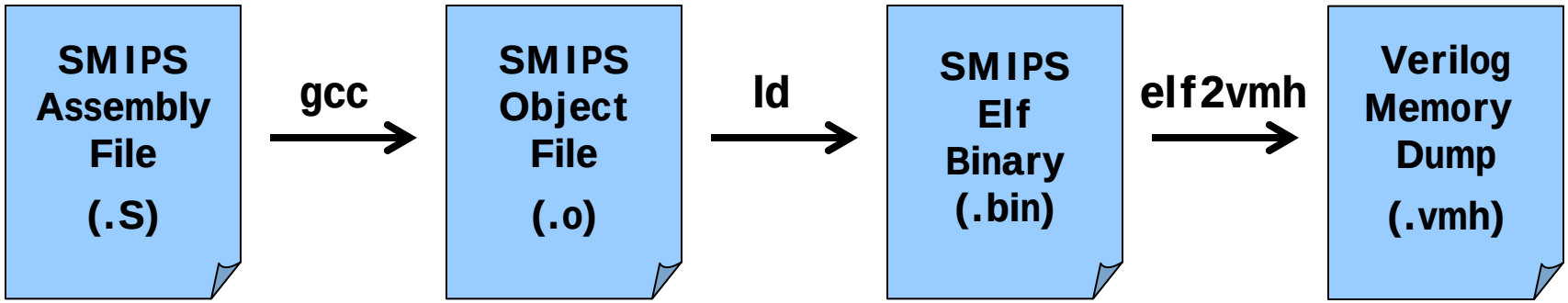
nala ous to asi pl f files e ept these use stan ar a e onstru ts

```
#====================================
# Configuration makefile module

verilog_src_dir   = verilog
verlog_toplevel   = gcd_test
verilog_srcs      = gcd_rtl.v

# vcs specific options
vcs_extra_options = -PP

# unit test options
utests_dir = tests
utests     = gcd-test.dat
```

Using the Makefiles with mips2stage

```
% cvs checkout 2005-spring/cbatten/mips2stage
% cd spring-2005/cbatten/mips2stage
% mkdir build
% cd build
% ../configure.pl ../config/mips2stage.mk
% make simv
% make self_test.bin
% make self_test.vmh
% ./simv +exe=self_test.vmh
% make run-tests
```

You can just use make run-tests and the dependency tracking will cause the simulator and the tests to be built before running the tests



Lab Checkoff Procedure

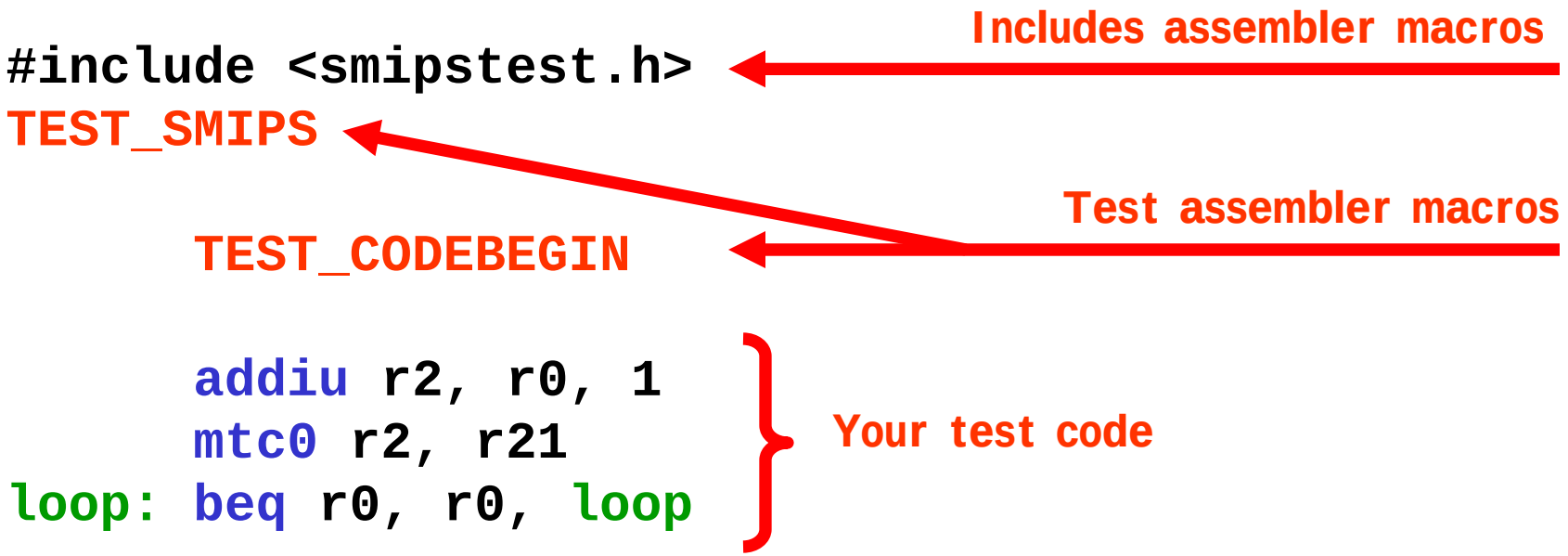
e ill e usin the follo in pro e ure to he off la so please a e sure these steps or on a lean uil

f the la finale ta oes not e ist then e ill ust he out the ost re ent ersion

```
% cvs checkout –r lab1-final \
                2005-spring/cbatten/mips2stage
% cd 2005-spring/cbatten/mips2stage
% mkdir build
% cd build
% ../configure.pl ../config/mips2stage.mk
% make simv
% make run-tests
% <run simv with some other tests>
```

Writing SMIPS Assembly

ur asse ler ta es as input asse ly o e
ith arious test a ros in the follo in for at



Writing SMIPS Assembly

ou an fin the asse ly for at for ea h instru tion in the pro essor spe ne t to the instru tion ta les

se self test as an e a ple

31 26	25 21	20 16	15 11	10 6	5 0	
opcode	rs	rt	rd	shamt	funct	R-type
opcode	rs	rt	immediate			I-type
opcode	target					J-type
Load and Store Instructions						
100011	base	dest	signed offset			LW rt, offset(rs)
101011	base	dest	signed offset			SW rt, offset(rs)
I-Type Computational Instructions						
001001	src	dest	signed immediate			ADDIU rt, rs, signed-imm.
001010	src	dest	signed immediate			SLTI rt, rs, signed-imm.
001011	src	dest	signed immediate			SLTIU rt, rs, signed-imm.
001100	src	dest	zero-ext. immediate			ANDI rt, rs, zero-ext-imm.
001101	src	dest	zero-ext. immediate			ORI rt, rs, zero-ext-imm.
001110	src	dest	zero-ext. immediate			XORI rt, rs, zero-ext-imm.
001111	00000	dest	zero-ext. immediate			LUI rt, zero-ext-imm.
R-Type Computational Instructions						
000000	00000	src	dest	shamt	000000	SLL rd, rt, shamt
000000	00000	src	dest	shamt	000010	SRL rd, rt, shamt
000000	00000	src	dest	shamt	000011	SRA rd, rt, shamt
000000	rshamt	src	dest	00000	000100	SLLV rd, rt, rs
000000	rshamt	src	dest	00000	000110	SRLV rd, rt, rs
000000	rshamt	src	dest	00000	000111	SRAV rd, rt, rs
000000	src1	src2	dest	00000	100001	ADDU rd, rs, rt
000000	src1	src2	dest	00000	100011	SUBU rd, rs, rt
000000	src1	src2	dest	00000	100100	AND rd, rs, rt
000000	src1	src2	dest	00000	100101	OR rd, rs, rt
000000	src1	src2	dest	00000	100110	XOR rd, rs, rt
000000	src1	src2	dest	00000	100111	NOR rd, rs, rt
000000	src1	src2	dest	00000	101010	SLT rd, rs, rt
000000	src1	src2	dest	00000	101011	SLTU rd, rs, rt
Jump and Branch Instructions						
000010	target					J target
000011	target					JAL target
000000	src	00000	00000	00000	001000	JR rs
000000	src	00000	dest	00000	001001	JALR rd, rs
000100	src1	src2	signed offset			BEQ rs, rt, offset
000101	src1	src2	signed offset			BNE rs, rt, offset
000110	src	00000	signed offset			BLEZ rs, offset
000111	src	00000	signed offset			BGTZ rs, offset
000001	src	00000	signed offset			BLTZ rs, offset
000001	src	00001	signed offset			BGEZ rs, offset
System Coprocessor (COP0) Instructions						
010000	00000	dest	cop0src	00000	000000	MFC0 rt, rd
010000	00100	src	cop0dest	00000	000000	MTC0 rt, rd

Writing SMIPS Assembly

ur asse ler a epts three types of re ister spe ifier for ats

```
        addiu r2, r0, 1
        mtc0 r2, r21
loop:   beq r0, r0, loop
```

```
        addiu $2, $0, 1
        mtc0 $2, $21
loop:   beq $0, $0, loop
```

```
        addiu t0, zero, 1
        mtc0 t0, $21
loop:   beq zero, zero, loop
```

} Traditional names for MIPS calling convention

Writing SMIPS Assembly (at)

The assembler reserves r for a ro e pansion and will o plain if you try an use it ithout e pli itly o erri in the assembler

```
#include <smipstest.h>
TEST_SMIPS

        TEST_CODEBEGIN

        .set noat
        addiu r1, zero, 1
        mtc0 r1, r21
loop:   beq zero, zero, loop
        .set at

        TEST_CODEEND
```

Assembler directive which tells the assembler not to use r1 (at = assembler temporary)

Writing SMIPS Assembly (reorder)

By efault the ran h elay slot is **not visible** The asse ler han les fillin the ran h elay slot unless you e pli itly ire t it not to

```
#include <smipstest.h>
TEST_SMIPS

        TEST_CODEBEGIN

         .set noreorder
        addiu r2, zero, 1
        mtc0 r2, r21
loop:   beq zero, zero, loop
        nop
         .set reorder

        TEST_CODEEND
```

Assembler directive which tells the assembler not to reorder instructions – programmer is responsible for filling in the delay slot

Use smips-objdump for Disassembly

entually the isasse le instru tions ill sho up in the h file ut it is still useful to ire tly isasse le the inary

```
#include <smipstest.h>
TEST_SMIPS

        TEST_CODEBEGIN
        addiu r2, zero, 1
        mtc0 r2, r21
loop:   beq zero, zero, loop
        TEST_CODEEND
```

```
% make simple_test.bin
% smips-objdump -D simple_test.bin
```

Examining Assembler Output

```
#include <smipstest.h>
TEST_SMIPS

      TEST_CODEBEGIN
      .set noat
      addiu r1, zero, 1
      mtc0 r1, r21
loop: beq zero, zero, loop
      .set at
      TEST_CODEEND
```

```
00001000 <__testresets>:
    1000: 40806800 mtc0 $zero,$13
    1004: 00000000 nop
    1008: 40805800 mtc0 $zero,$11
    100c: 3c1a0000 lui $k0,0x0
    1010: 8f5a1534 lw $k0,5428($k0)
    1014: 409a6000 mtc0 $k0,$12
    1018: 3c1a0000 lui $k0,0x0
    101c: 275a1400 addiu $k0,$k0,5120
    1020: 03400008 jr $k0
    1024: 42000010 rfe
00001100 <__testexcep>:
    1100: 401a6800 mfc0 $k0,$13
    1104: 00000000 nop
    <snip>

00001400 <__testcode>:
    1400: 24010001 li $at,1
    1404: 4081a800 mtc0 $at,$21

0001408 <loop>:
    1408: 1000ffff b 1408 <loop>
          ...
    141c: 3c080000 lui $t0,0x0
    1420: 8d081530 lw $t0,5424($t0)
    1424: 3c01dead lui $at,0xdead
    1428: 3421beef ori $at,$at,0xbeef
    142c: 11010003 beq $t0,$at,143c <loop+34>
          ...
    1438: 0000000d break
    143c: 24080001 li $t0,1
    1440: 4088a800 mtc0 $t0,$21
    1444: 1000ffff b 1444 <loop+3c>
```

Examining Assembler Output

```
#include <smipstest.h>
TEST_SMIPS

        TEST_CODEBEGIN
        .set noat
        addiu r1, zero, 1
        mtc0 r1, r21
loop:   beq zero, zero, loop
        .set at
        TEST_CODEEND
```

```
00001000 <__testresets>:
    1000: 40806800 mtc0 $zero,$13
    1004: 00000000 nop
    1008: 40805800 mtc0 $zero,$11
    100c: 3c1a0000 lui $k0,0x0
    1010: 8f5a1534 lw $k0,5428($k0)
    1014: 409a6000 mtc0 $k0,$12
    1018: 3c1a0000 lui $k0,0x0
    101c: 275a1400 addiu $k0,$k0,5120
    1020: 03400008 jr $k0
    1024: 42000010 rfe
00001100 <__testexcep>:
    1100: 401a6800 mfc0 $k0,$13
    1104: 00000000 nop
    <snip>

00001400 <__testcode>:
    1400: 24010001 li $at,1
    1404: 4081a800 mtc0 $at,$21

0001408 <loop>:
    1408: 1000ffff b 1408 <loop>
        ...
    141c: 3c080000 lui $t0,0x0
    1420: 8d081530 lw $t0,5424($t0)
    1424: 3c01dead lui $at,0xdead
    1428: 3421beef ori $at,$at,0xbeef
    142c: 11010003 beq $t0,$at,143c <loop+34>
        ...
    1438: 0000000d break
    143c: 24080001 li $t0,1
    1440: 4088a800 mtc0 $t0,$21
    1444: 1000ffff b 1444 <loop+3c>
```

Trace Output Instead of Waveforms

It is sometimes very useful to use $display calls from the test harness to create a cycle by cycle trace output instead of pouring through a waveform

```
integer cycle = 0;
always @( posedge clk )
begin
 #2;
 $display("CYC:%2d [pc=%x] [ireg=%x] [rd1=%x] [rd2=%x] [wd=%x] tohost=%d",
           cycle, mips.fetch_unit.pc, mips.exec_unit.ir,
           mips.exec_unit.rd1, mips.exec_unit.rd2, mips.exec_unit.wd, tohost);
  cycle = cycle + 1;
end
```

```
CYC: 0 [pc=00001000] [ireg=xxxxxxxx] [rd1=xxxxxxxx] [rd2=xxxxxxxx] [wd=00001004] tohost=  0
CYC: 1 [pc=00001004] [ireg=08000500] [rd1=00000000] [rd2=00000000] [wd=00001008] tohost=  0
CYC: 2 [pc=00001400] [ireg=00000000] [rd1=00000000] [rd2=00000000] [wd=00000000] tohost=  0
CYC: 3 [pc=00001404] [ireg=24010001] [rd1=00000000] [rd2=xxxxxxxx] [wd=00000001] tohost=  0
CYC: 4 [pc=00001408] [ireg=4081a800] [rd1=xxxxxxxx] [rd2=00000001] [wd=0000140c] tohost=  0
CYC: 5 [pc=0000140c] [ireg=1000ffff] [rd1=00000000] [rd2=00000000] [wd=00001410] tohost=  1
CYC: 6 [pc=00001408] [ireg=00000000] [rd1=00000000] [rd2=00000000] [wd=00000000] tohost=  1
```

Trace Output Instead of Waveforms

t is so eti es ery useful to use isplay alls fro the test harness to reate y le y y le tra e output instea of pourin throu h a efor s

```
#include <smipstest.h>
TEST_SMIPS
        TEST_CODEBEGIN
        .set noat
        addiu r1, zero, 1
        mtc0 r1, r21
loop:   beq zero, zero, loop
        .set at
        TEST_CODEEND
```

```
CYC: 0 [pc=00001000] [ireg=xxxxxxxx] [rd1=xxxxxxxx] [rd2=xxxxxxxx] [wd=00001004] tohost=  0
CYC: 1 [pc=00001004] [ireg=08000500] [rd1=00000000] [rd2=00000000] [wd=00001008] tohost=  0
CYC: 2 [pc=00001400] [ireg=00000000] [rd1=00000000] [rd2=00000000] [wd=00000000] tohost=  0
CYC: 3 [pc=00001404] [ireg=24010001] [rd1=00000000] [rd2=xxxxxxxx] [wd=00000001] tohost=  0
CYC: 4 [pc=00001408] [ireg=4081a800] [rd1=xxxxxxxx] [rd2=00000001] [wd=0000140c] tohost=  0
CYC: 5 [pc=0000140c] [ireg=1000ffff] [rd1=00000000] [rd2=00000000] [wd=00001410] tohost=  1
CYC: 6 [pc=00001408] [ireg=00000000] [rd1=00000000] [rd2=00000000] [wd=00000000] tohost=  1
```

Final Notes

Lab Assignment 1

on t orry a out s a e for no sin e ill e finishin settin this up this afternoon

lease rite at least one other s all test it too e a lon ti e to et the asse ly tool hain or in

ou ust erify that the he off pro e ure or s

our erilo ill e he e out auto ati ally ri ay at p

Lab Assignment 2

ynthesi e your t o sta e ips pro essor

ssi ne on ri ay an ue the follo in ri ay e

ill or on a synthesis tutorial o er the ee en an e ail it out on on ay