Homework-4

Implementing a Leon based SoC

---

1. Setting Up The Files

We will use leon2-1.0.12 for this homework. All files for this version can be found at :
/home/rishi/soc/soc.tar.gz

From your home directory proceed as follows.

```
➢ mkdir hw4
➢ cd hw4
➢ cp /home/rishi/soc/soc.tar.gz .
➢ gunzip -c soc.tar.gz | tar -xvf -
```

The LEON model has the following directory structure:

soc	top directory
soc/Makefile	top-level Makefile
soc/leon/	LEON vhdl model
soc/modelsim/	Modelsim simulator support files
soc/pmon	Boot-monitor
soc/syn	Synthesis support files
soc/tbench	LEON VHDL test bench
soc/tsource	LEON test bench (C source)
soc/aes	AES vhdl model + AMBA wrapper
soc/fir	FIR vhdl model + AMBA wrapper
soc/org_edit	Original & modified files
soc/ram_tsmc25	ARTISAN RAM models

- cd soc
- cp /home/rishi/soc/soc/modelsim.ini .
- cd tsource
- cp ram.dat ram_backup.dat
- cd ..

Editing the top-level Makefile:

- **pico Makefile**

  Go to the bottom of the Makefile where you can see this…

```
clean:
        cd leon; make clean
        cd tbench; make clean
        cd tkconfig; make clean
        cd tsource; make clean
        -rm dumpdata.txt trnscrpt transcript vsim.wlf vsim.wav
```

Instead of above it should be like this. (We don't want to delete the files of tkconfig,tsource every time).

```
clean:
        cd leon; make clean
        cd tbench; make clean
        -rm dumpdata.txt trnscrpt transcript vsim.wlf vsim.wav
```

2. Customizing the LEON

2.1 Customizing for Virtex2 Technology

In /hw4/soc directory:

- **make xconfig**

a window like below should open.

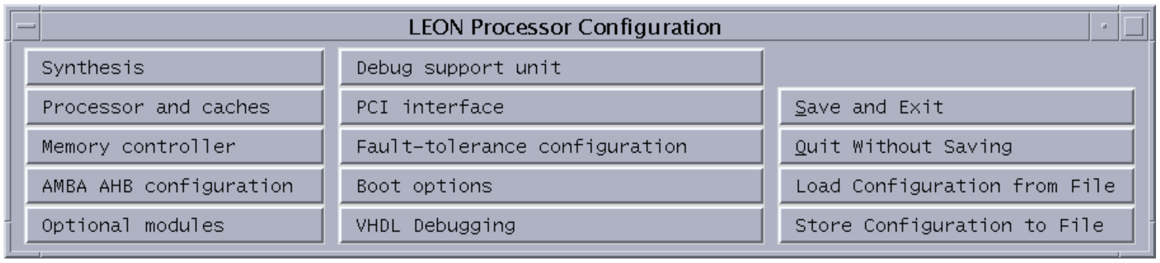
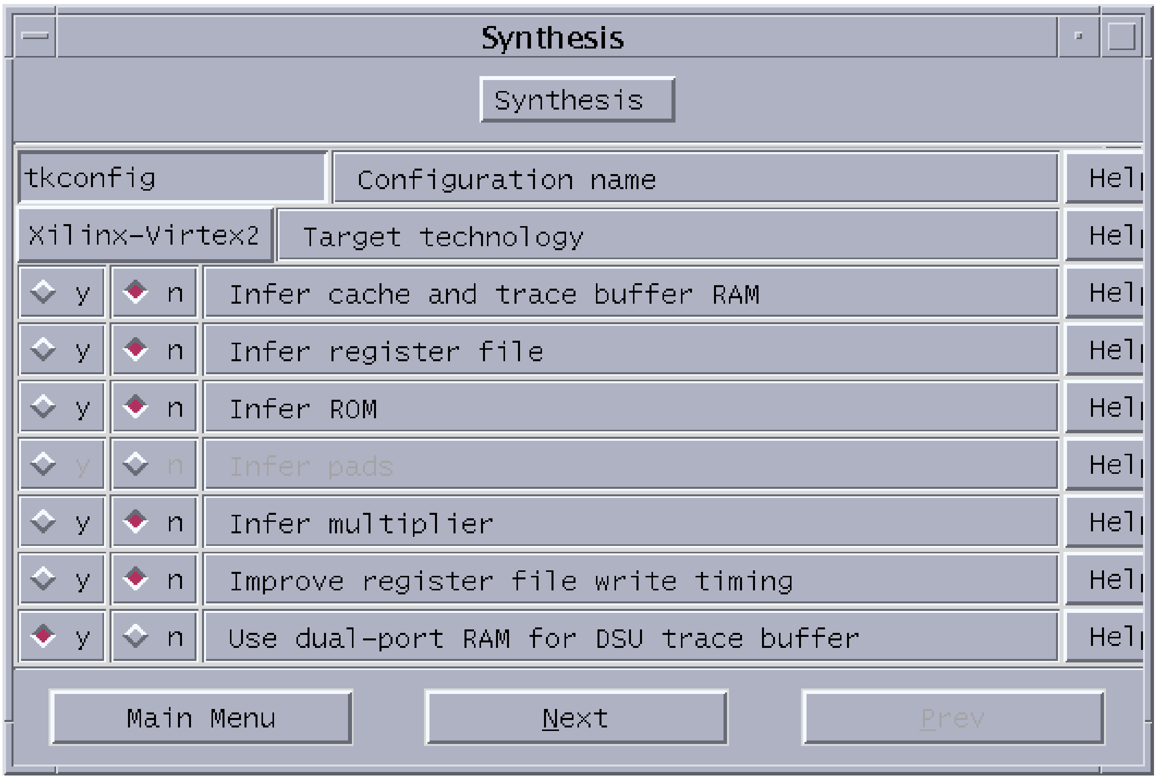


Figure1

Click on "**synthesis**" and a second window for synthesis customization will open. In that window select "**Target Technology**" to be "**Virtex2**" and you'll see all the variables below are automatically selected. For time being we'll use the default values of these variables. The window should look like this.



tkconfig	Configuration name
Xilinx-Virtex2	Target technology
y [✓ n]	Infer cache and trace buffer RAM
y [✓ n]	Infer register file
y [✓ n]	Infer ROM
y n	Infer pads
y [✓ n]	Infer multiplier
y [✓ n]	Improve register file write timing
[✓ y] n	Use dual-port RAM for DSU trace buffer

Figure2

Click on "**Main Menu**" button. In main menu select "**Boot option**" in that window select boot option to be "**Memory**" (Default is: **Memory**). Click on "**Main Menu**" button.

Press "**Save and Exit**" button.

- **make dep** (this will create a device.vhd file in leon directory which would contain your customized design)
- **mentor_old_tools**
- **make all** (this will compile all the files)

Once the LEON model has been compiled, use the TB_FUNC32 test bench to verify the behaviour of the model. **Simulation should be started in the top directory**.

- **vsim tb_func32 &**
- **run -all** (In the ModeSim window)

The output from the simulation should be similar to:

```
# *** Starting LEON system test ***
# Register file
# Cache controllers
# Interrupt controller
# UARTs
# Timers, watchdog and power-down
# Parallel I/O port
# Test completed OK, halting with failure
# ** Failure: TEST COMPLETED OK, ending with FAILURE
Simulation is halted by generating a failure.
```

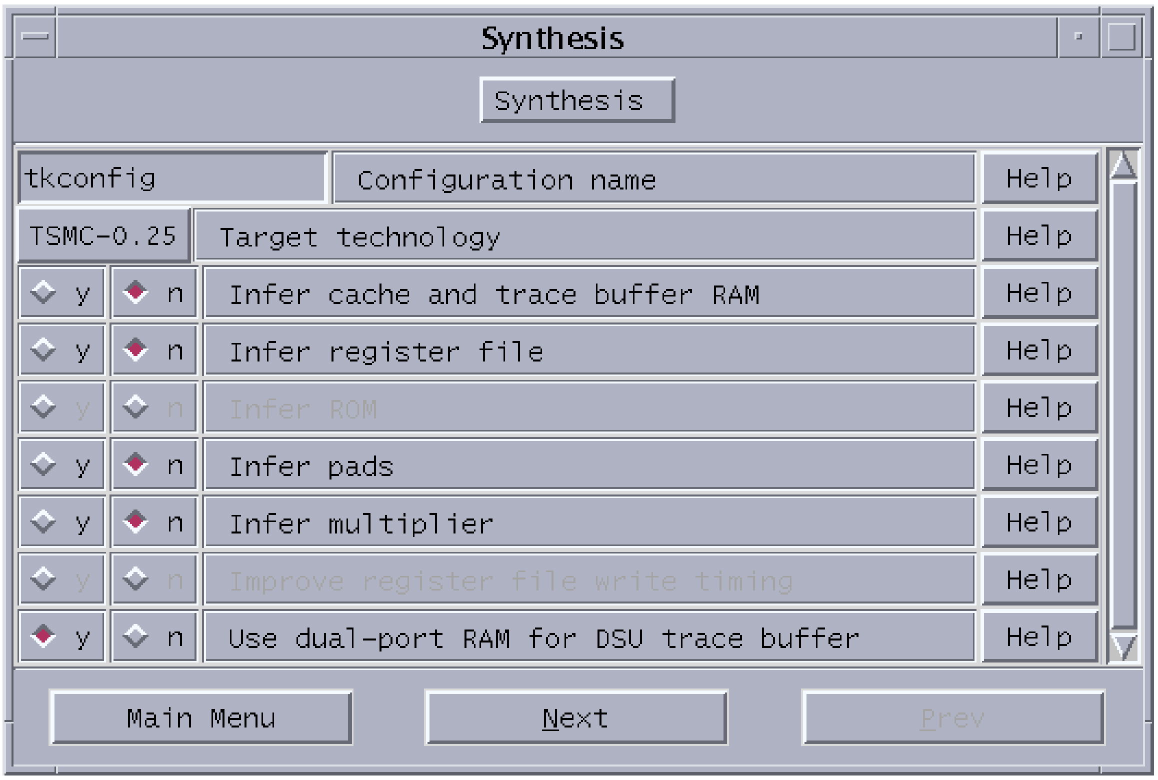
2.2 Customizing for TSMC-25 Technology

In /hw4/soc directory:

- **cd leon**
- **cp device.vhd device-virtex2.vhd**
- **cd ..**
- **make xconfig**

In "**Main Menu**" click on "**synthesis**" and a second window for synthesis customization will open. In that window select "**Target**

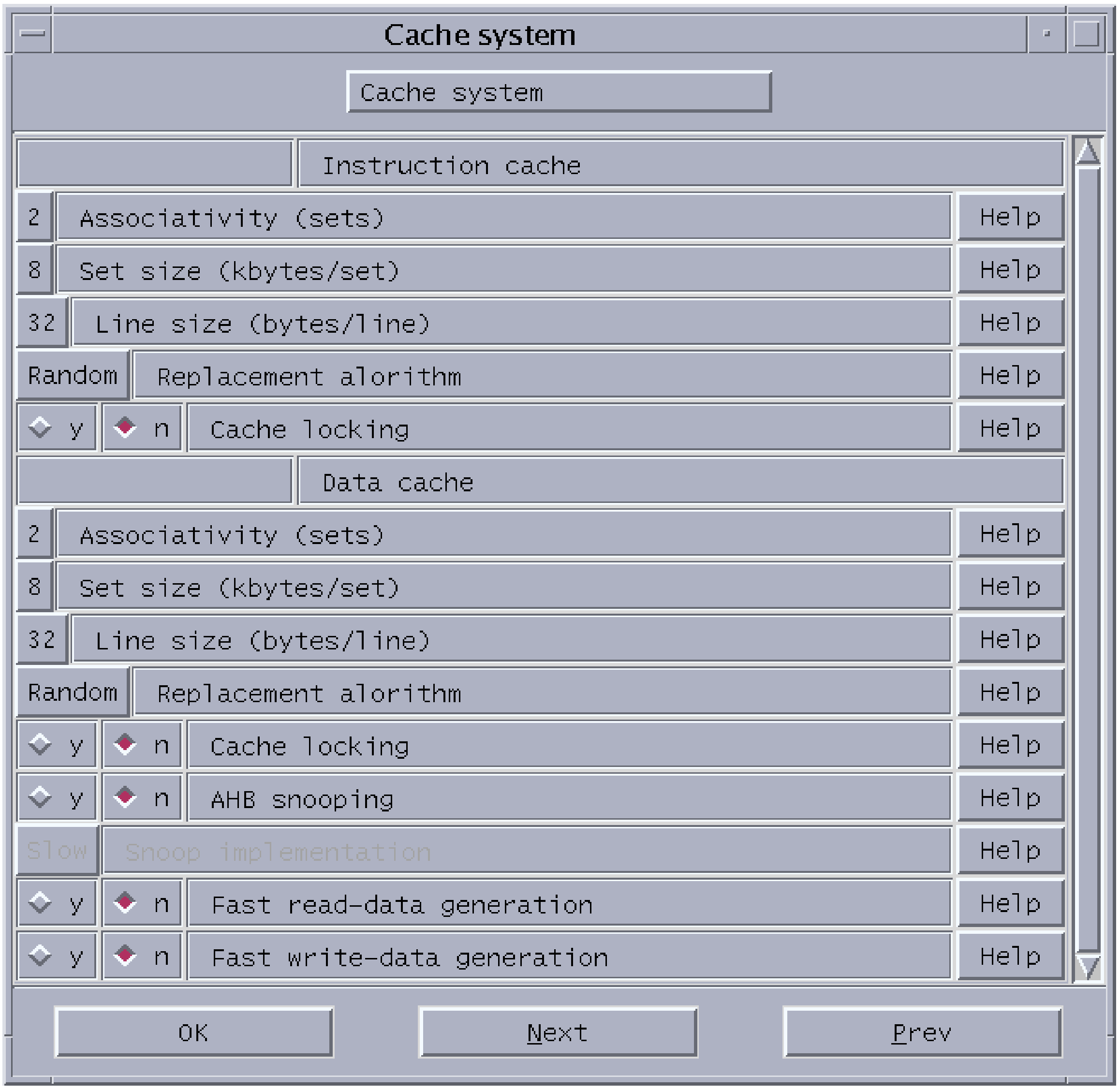
Technology” to be “**TSMC 25**” and you’ll see all the variables below are automatically selected. For time being we’ll use the default values of these variables. The window should look like this.



tkconfig	Configuration name	
TSMC-0.25	Target technology	
y [✓ n]	Infer cache and trace buffer RAM	
y [✓ n]	Infer register file	
y n	Infer ROM	
y [✓ n]	Infer pads	
y [✓ n]	Infer multiplier	
y n	Improve register file write timing	
[✓ y] n	Use dual-port RAM for DSU trace buffer	

Figure3

Click on “**Main Menu**” button. In main menu select “**Boot option**” in that window select boot option to be “**Memory**” (Default is: **Memory**). Click on “**Main Menu**” button. In main menu select “**Processor and caches**” in that window select “**cache system**” and change the “**set size**” to **8k**. Entrees should look like this. Press “**OK**” and then “**Main Menu**”.



Cache system

Instruction cache

2	Associativity (sets)	Help
8	Set size (kbytes/set)	Help
32	Line size (bytes/line)	Help
Random	Replacement alorithm	Help
y [✓ n]	Cache locking	Help

Data cache

2	Associativity (sets)	Help
8	Set size (kbytes/set)	Help
32	Line size (bytes/line)	Help
Random	Replacement alorithm	Help
y [✓ n]	Cache locking	Help
y [✓ n]	AHB snooping	Help
Slow	Snoop implementation	Help
y [✓ n]	Fast read-data generation	Help
y [✓ n]	Fast write-data generation	Help

Figure4

For now we'll use only default settings for others. Don't mess with it until you know what you are doing. Press "**Save and Exit**" button.

- **make dep** (this will create a device.vhd file in leon directory which would contain your customized design)
- **mentor_old_tools**
- **make all** (this will compile all the files)

Once the LEON model has been compiled, use the TB_FUNC32 test bench to verify the behaviour of the model. **Simulation should be started in the top directory**.

- **vsim tb_func32 &**
- **run -all** (In the ModeSim window)

The output from the simulation should be similar to:

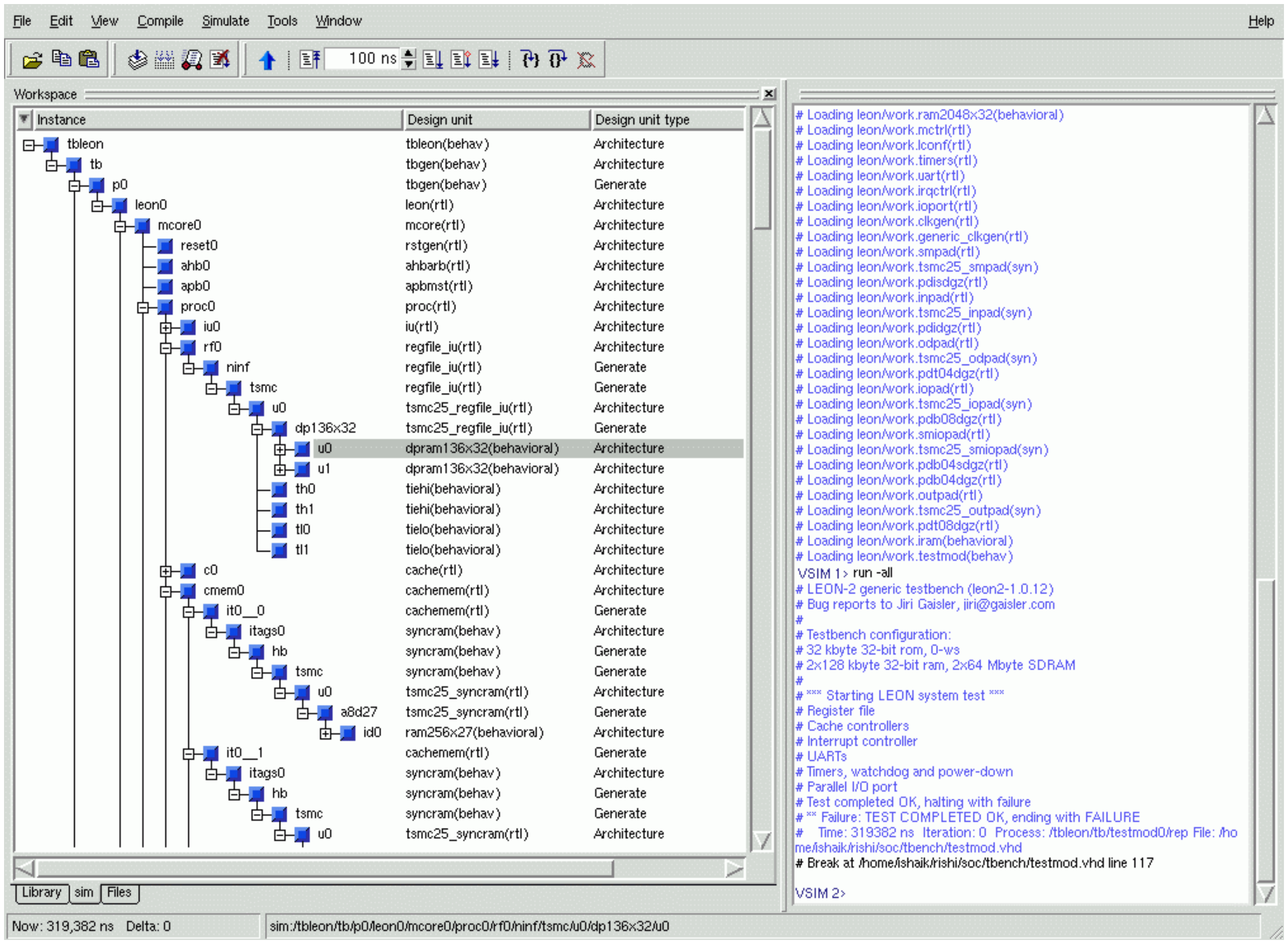
```
# *** Starting LEON system test ***
# Register file
# Cache controllers
# Interrupt controller
# UARTs
# Timers, watchdog and power-down
# Parallel I/O port
# Test completed OK, halting with failure
# ** Failure: TEST COMPLETED OK, ending with FAILURE
Simulation is halted by generating a failure.
```

3. ARTISAN Rams

To find out what size of rams we need in our design. We may have to go back one step. From top directory i.e. '**soc**' run:

- **make all**
- **vsim tb_func32&**

In Modelsim window we can see the size of the rams it is using, by going to the "**proc0**" model as shown in the figure below. As we can see we need to use DPRAM of size 136x32 and single port ram of size 256x27. We can use DPRAM instead of single port rams.



Instance	Design unit	Design unit type
tbleon	tbleon(behav)	Architecture
tb	tbgen(behav)	Architecture
p0	tbgen(behav)	Generate
leon0	leon(rtl)	Architecture
mcore0	mcore(rtl)	Architecture
reset0	rstgen(rtl)	Architecture
ahb0	ahbarb(rtl)	Architecture
apb0	apbmst(rtl)	Architecture
proc0	proc(rtl)	Architecture
iu0	iu(rtl)	Architecture
rf0	regfile_iu(rtl)	Architecture
ninf	regfile_iu(rtl)	Generate
tsmc	regfile_iu(rtl)	Generate
u0	tsmc25_regfile_iu(rtl)	Architecture
dp136x32	tsmc25_regfile_iu(rtl)	Generate
u0	dpram136x32(behavioral)	Architecture
u1	dpram136x32(behavioral)	Architecture
th0	tiehi(behavioral)	Architecture
th1	tiehi(behavioral)	Architecture
tl0	tielo(behavioral)	Architecture
tl1	tielo(behavioral)	Architecture
c0	cache(rtl)	Architecture
cmem0	cachemem(rtl)	Architecture
it0__0	cachemem(rtl)	Generate
itags0	syncram(behav)	Architecture
hb	syncram(behav)	Generate
tsmc	syncram(behav)	Generate
u0	tsmc25_syncram(rtl)	Architecture
a8d27	tsmc25_syncram(rtl)	Generate
id0	ram256x27(behavioral)	Architecture
it0__1	cachemem(rtl)	Generate
itags0	syncram(behav)	Architecture
hb	syncram(behav)	Generate
tsmc	syncram(behav)	Generate
u0	tsmc25_syncram(rtl)	Architecture

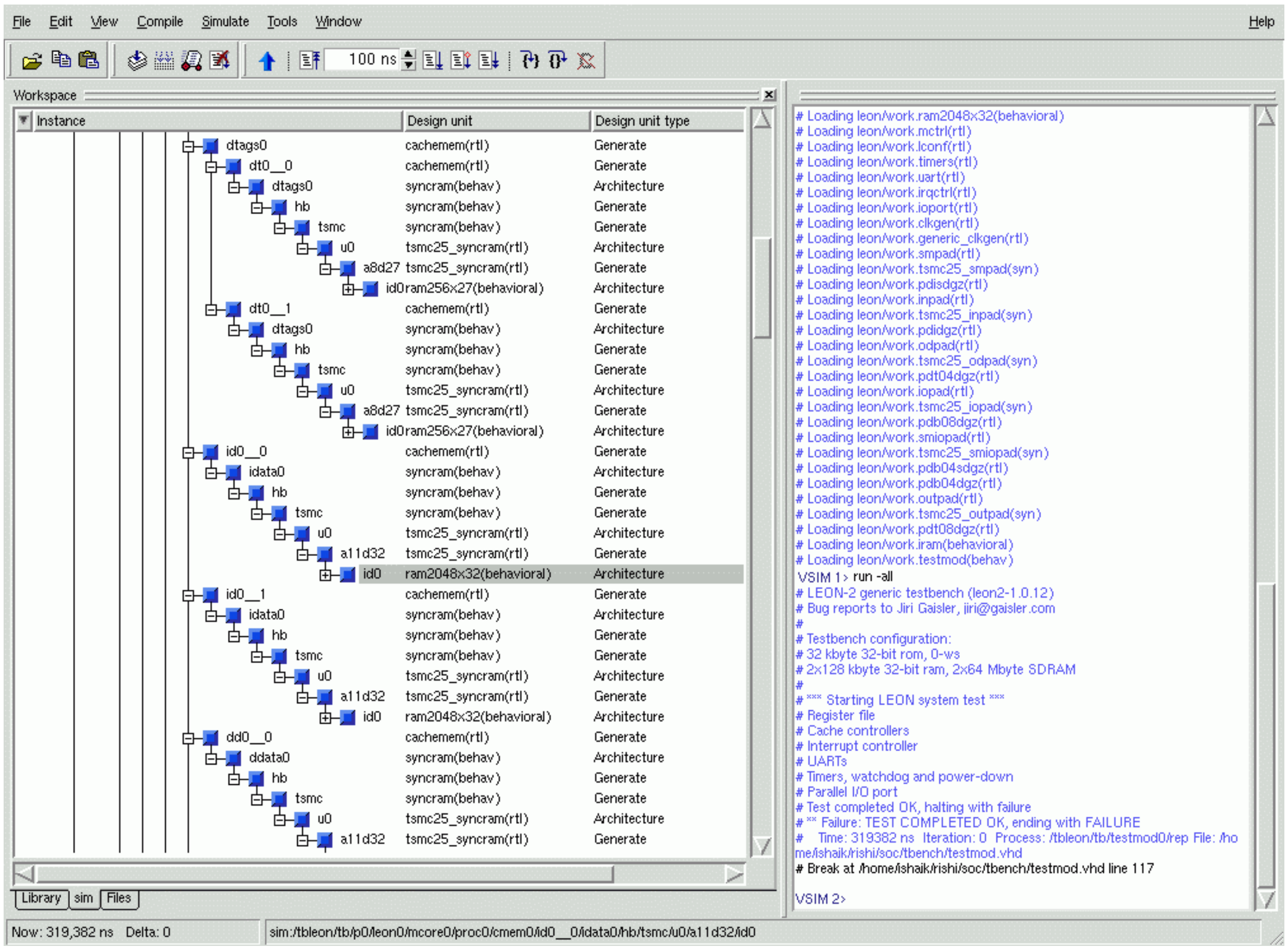
```
# Loading leon/work.ram2048x32(behavioral)
# Loading leon/work.mctrl(rtl)
# Loading leon/work.lconf(rtl)
# Loading leon/work.timers(rtl)
# Loading leon/work.uart(rtl)
# Loading leon/work.irqctrl(rtl)
# Loading leon/work.ioport(rtl)
# Loading leon/work.clkgen(rtl)
# Loading leon/work.generic_clkgen(rtl)
# Loading leon/work.smpad(rtl)
# Loading leon/work.tsmc25_smpad(syn)
# Loading leon/work.pdisdgz(rtl)
# Loading leon/work.inpad(rtl)
# Loading leon/work.tsmc25_inpad(syn)
# Loading leon/work.pdidgz(rtl)
# Loading leon/work.odpad(rtl)
# Loading leon/work.tsmc25_odpad(syn)
# Loading leon/work.pdt04dgz(rtl)
# Loading leon/work.iopad(rtl)
# Loading leon/work.tsmc25_iopad(syn)
# Loading leon/work.pdb08dgz(rtl)
# Loading leon/work.smiopad(rtl)
# Loading leon/work.tsmc25_smiopad(syn)
# Loading leon/work.pdb04sdgz(rtl)
# Loading leon/work.pdb04dgz(rtl)
# Loading leon/work.outpad(rtl)
# Loading leon/work.tsmc25_outpad(syn)
# Loading leon/work.pdt08dgz(rtl)
# Loading leon/work.iram(behavioral)
# Loading leon/work.testmod(behav)
VSIM 1> run -all
# LEON-2 generic testbench (leon2-1.0.12)
# Bug reports to Jiri Gaisler, jiri@gaisler.com
#
# Testbench configuration:
# 32 kbyte 32-bit rom, 0-ws
# 2x128 kbyte 32-bit ram, 2x64 Mbyte SDRAM
#
# *** Starting LEON system test ***
# Register file
# Cache controllers
# Interrupt controller
# UARTs
# Timers, watchdog and power-down
# Parallel I/O port
# Test completed OK, halting with failure
# ** Failure: TEST COMPLETED OK, ending with FAILURE
#    Time: 319382 ns  Iteration: 0  Process: /tbleon/tb/testmod0/rep File: /ho
me/ishaik/rishi/soc/tbench/testmod.vhd
# Break at /home/ishaik/rishi/soc/tbench/testmod.vhd line 117

VSIM 2>
```

```
Now: 319,382 ns   Delta: 0     sim:/tbleon/tb/p0/leon0/mcore0/proc0/rf0/ninf/tsmc/u0/dp136x32/u0
```

Figure5

In figure 2 below we see that we also need a single port ram of size 2048x32.We can use DPRAM instead of single port rams.



Instance	Design unit	Design unit type
dtags0	cachemem(rtl)	Generate
dt0__0	cachemem(rtl)	Generate
dtags0	syncram(behav)	Architecture
hb	syncram(behav)	Generate
tsmc	syncram(behav)	Generate
u0	tsmc25_syncram(rtl)	Architecture
a8d27	tsmc25_syncram(rtl)	Generate
id0ram256x27(behavioral)		Architecture
dt0__1	cachemem(rtl)	Generate
dtags0	syncram(behav)	Architecture
hb	syncram(behav)	Generate
tsmc	syncram(behav)	Generate
u0	tsmc25_syncram(rtl)	Architecture
a8d27	tsmc25_syncram(rtl)	Generate
id0ram256x27(behavioral)		Architecture
id0__0	cachemem(rtl)	Generate
idata0	syncram(behav)	Architecture
hb	syncram(behav)	Generate
tsmc	syncram(behav)	Generate
u0	tsmc25_syncram(rtl)	Architecture
a11d32	tsmc25_syncram(rtl)	Generate
id0	ram2048x32(behavioral)	Architecture
id0__1	cachemem(rtl)	Generate
idata0	syncram(behav)	Architecture
hb	syncram(behav)	Generate
tsmc	syncram(behav)	Generate
u0	tsmc25_syncram(rtl)	Architecture
a11d32	tsmc25_syncram(rtl)	Generate
id0	ram2048x32(behavioral)	Architecture
dd0__0	cachemem(rtl)	Generate
ddata0	syncram(behav)	Architecture
hb	syncram(behav)	Generate
tsmc	syncram(behav)	Generate
u0	tsmc25_syncram(rtl)	Architecture
a11d32	tsmc25_syncram(rtl)	Generate

```
# Loading leon/work.ram2048x32(behavioral)
# Loading leon/work.mctrl(rtl)
# Loading leon/work.lconf(rtl)
# Loading leon/work.timers(rtl)
# Loading leon/work.uart(rtl)
# Loading leon/work.irqctrl(rtl)
# Loading leon/work.ioport(rtl)
# Loading leon/work.clkgen(rtl)
# Loading leon/work.generic_clkgen(rtl)
# Loading leon/work.smpad(rtl)
# Loading leon/work.tsmc25_smpad(syn)
# Loading leon/work.pdisdgz(rtl)
# Loading leon/work.inpad(rtl)
# Loading leon/work.tsmc25_inpad(syn)
# Loading leon/work.pdidgz(rtl)
# Loading leon/work.odpad(rtl)
# Loading leon/work.tsmc25_odpad(syn)
# Loading leon/work.pdt04dgz(rtl)
# Loading leon/work.iopad(rtl)
# Loading leon/work.tsmc25_iopad(syn)
# Loading leon/work.pdb08dgz(rtl)
# Loading leon/work.smiopad(rtl)
# Loading leon/work.tsmc25_smiopad(syn)
# Loading leon/work.pdb04sdgz(rtl)
# Loading leon/work.pdb04dgz(rtl)
# Loading leon/work.outpad(rtl)
# Loading leon/work.tsmc25_outpad(syn)
# Loading leon/work.pdt08dgz(rtl)
# Loading leon/work.iram(behavioral)
# Loading leon/work.testmod(behav)
VSIM 1> run -all
# LEON-2 generic testbench (leon2-1.0.12)
# Bug reports to Jiri Gaisler, jiri@gaisler.com
#
# Testbench configuration:
# 32 kbyte 32-bit rom, 0-ws
# 2x128 kbyte 32-bit ram, 2x64 Mbyte SDRAM
#
# *** Starting LEON system test ***
# Register file
# Cache controllers
# Interrupt controller
# UARTs
# Timers, watchdog and power-down
# Parallel I/O port
# Test completed OK, halting with failure
# ** Failure: TEST COMPLETED OK, ending with FAILURE
#    Time: 319382 ns  Iteration: 0  Process: /tbleon/tb/testmod0/rep File: /ho
me/ishaik/rishi/soc/tbench/testmod.vhd
# Break at /home/ishaik/rishi/soc/tbench/testmod.vhd line 117

VSIM 2>
```

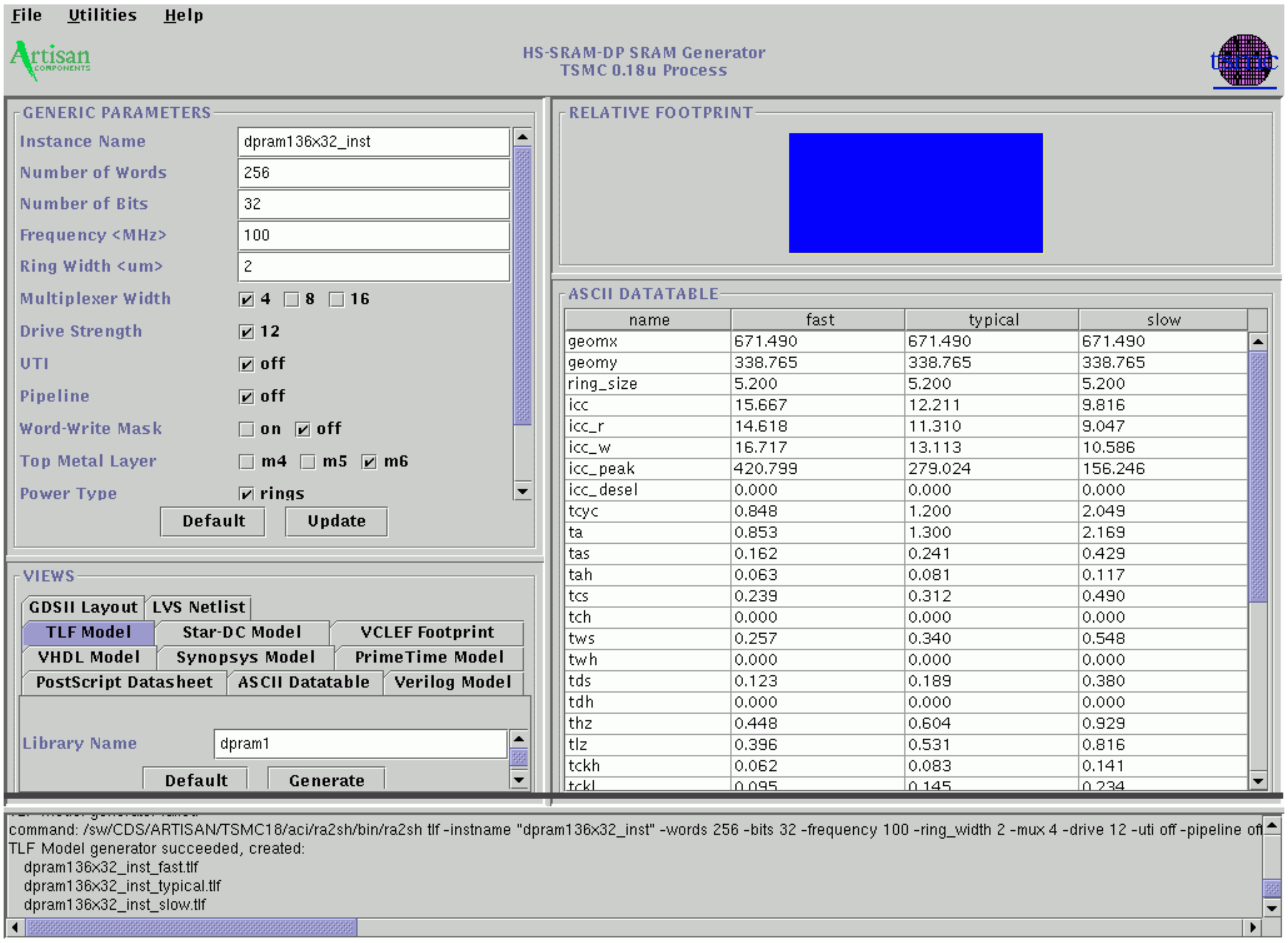
```
Now: 319,382 ns   Delta: 0
sim:/tbleon/tb/p0/leon0/mcore0/proc0/cmem0/id0__0/idata0/hb/tsmc/u0/a11d32/id0
```

Figure6

Now, exit from the Modelsim window. The next step is to generate synthesizable RTL models for RAMs. In top directory ‘**soc**’ execute following commands:

- **mkdir ram_tsmc25** (not required)
- **cd ram_tsmc25**
- **/sw/CDS/ARTISAN/TSMC18/aci/ra2sh/bin/ra2sh**

A window like this should open.



Instance Name	dpram136x32_inst
Number of Words	256
Number of Bits	32
Frequency <MHz>	100
Ring Width <um>	2
Multiplexer Width	[✓ 4] 8 16
Drive Strength	[✓ 12]
UTI	[✓ off]
Pipeline	[✓ off]
Word-Write Mask	on [✓ off]
Top Metal Layer	m4 m5 [✓ m6]
Power Type	[✓ rings]
Library Name	dpram1

name	fast	typical	slow
geomx	671.490	671.490	671.490
geomy	338.765	338.765	338.765
ring_size	5.200	5.200	5.200
icc	15.667	12.211	9.816
icc_r	14.618	11.310	9.047
icc_w	16.717	13.113	10.586
icc_peak	420.799	279.024	156.246
icc_desel	0.000	0.000	0.000
tcyc	0.848	1.200	2.049
ta	0.853	1.300	2.169
tas	0.162	0.241	0.429
tah	0.063	0.081	0.117
tcs	0.239	0.312	0.490
tch	0.000	0.000	0.000
tws	0.257	0.340	0.548
twh	0.000	0.000	0.000
tds	0.123	0.189	0.380
tdh	0.000	0.000	0.000
thz	0.448	0.604	0.929
tlz	0.396	0.531	0.816
tckh	0.062	0.083	0.141
tckl	0.095	0.145	0.234

```
command: /sw/CDS/ARTISAN/TSMC18/aci/ra2sh/bin/ra2sh tlf -instname "dpram136x32_inst" -words 256 -bits 32 -frequency 100 -ring_width 2 -mux 4 -drive 12 -uti off -pipeline off
TLF Model generator succeeded, created:
   dpram136x32_inst_fast.tlf
   dpram136x32_inst_typical.tlf
   dpram136x32_inst_slow.tlf
```

Figure7

We need to generate following views for each of our RAM design. 1. Verilog Model, 2. Synopsys Model, 3. TLF Model, 4. VCLEF footprint, 5. GDSII Layout. Following files are generated in ram_tsmc25 directory:

Model	Files genertaed
Verilog	dpram136x32_inst.v
Synopsys model	dpram136x32_inst_fast_syn.lib, dpram136x32_inst_typical_syn.lib, dpram136x32_inst_slow_syn.lib
TLF Model	dpram136x32_inst_fast.tlf, dpram136x32_inst_typical.tlf dpram136x32_inst_slow.tlf
VCLEF Footprint	dpram136x32_inst.vclef
GDSII Layout	dpram136x32_inst.gds2

For more information you can find the manual for artisan ram generator at:

/sw/CDS/ARTISAN/TSMC18/aci/ra2sh/doc/user_guide

Similar models are generated for **ram256x27** & **ram2048x32** using the information given in below table.

PARAMETERS	**DPRAM 136x32**	**RAM 256x27**	**RAM 2048x32**
Instance Name	dpram136x32_inst	ram256x27_inst	ram2048x32_inst
Number of words	256	256	2048
Number of width	32	27	32
Frequency (Mhz)	50	50	50
Multiplexer Width	4	4	8
Library Name (when needed)	DPRAM1	RAM2	RAM3

Use default values for the rest,(i.e., ring width=2 etc)

These RAMs cannot be used as it is. Wrappers are required for these blocks in order to communicate with the leon-2 processor in same fashion as the behavioral models do. These wrappers **(dpram136x32_box0.vhd, dpram136x32_box1.vhd, ram2048x32_box0.vhd & ram256x27_box0.vhd)** are provided in ram_tsmc25 directory.

4. Synthesis: Leon-2 Processor-standalone

- **cd syn**
- **cp /home/rishi/652/soc/leon2-1.0.12/syn/.synopsys_dc.setup .**
- **cp /home/rishi/652/soc/leon2-1.0.12/syn/.synopsys_vss.setup .**

We will be using 'Synopsys model' (i.e. *.lib files) of rams for synthesis purposes. In order to this, we need to add the designs in the library (i.e. *.lib files) to a database (i.e. *.db files) and then add that database format to our tsmc18 cell database.

Generating the synopsys database for RAMs. (library to database conversion).Create a new File 'lib2db.dcsh' with the following data

```
define_design_lib WORK -path WORK
read_lib ../ram_tsmc25/dpram136x32_inst_typical_syn.lib
write_lib DPRAM1 -format db –output ../ram_tsmc25/dpram136x32_inst_typical.db

read_lib ../ram_tsmc25/ram256x27_inst_typical_syn.lib
write_lib RAM2 -format db -output ../ram_tsmc25/ram256x27_inst_typical.db

read_lib ../ram_tsmc25/ram2048x32_inst_typical_syn.lib
write_lib RAM3 -format db -output ../ram_tsmc25/ram2048x32_inst_typical.db

quit
```

- **mkdir WORK**
- **synopsys_tools**
- **dc_shell -f lib2db.dcsh**

Now we need to edit **'.synopsys_dc.setup'** file to add RAM's database to our TSMC18 cell database. Students need to change the highlighted text according to their directory structure. If you are using the same file names as in the above table, no need to modify .db file names.

```
File Name: .synopsys_dc.setup
search_path = {} + search_path + /sw/CDS/ARTISAN/TSMC18/aci/sc/synopsys +
/sw/CDS/ARTISAN/TSMC18/PADS/synopsys/tpz973g_200c +
/home/tmarwah/soc/soc/ram_tsmc25
link_library = {typical.db"*"}
target_library = typical.db
symbol_library = typical.db

syntetic_library = { /sw/synopsys/libraries/syn/dw06.sldb +
/sw/synopsys/libraries/syn/dw02.sldb + /sw/synopsys/libraries/syn/dw01.sldb  }

link_library = target_library + synthetic_library + dw06.sldb + dw03.sldb +
dw02.sldb + dw01.sldb + tpz973gtc.db + dpram136x32_inst_typical.db +
ram2048x32_inst_typical.db + ram256x27_inst_typical.db
search_path = search_path + {synopsys_root + "/dw/sim_ver"}
```

In order to generate Black Box for Rams, create a new file 'ram_box.dcsh' in 'syn' directory with the following contents:

```
define_design_lib WORK -path WORK
analyze -f vhdl -library WORK  ../ram_tsmc25/ram256x27_box0.vhd
elaborate ram256x27_box0
uniquify
compile -map_effort high
write -f verilog -hierarchy -o ../leon/ram256x27_box0.v
 quit
```

- **cd syn**
- **rm –r WORK**
- **mkdir WORK**
- **synopsys_tools**
- **dc_shell –f ram_box.dcsh** (do it for each ram model)

Note# Run the above file for each ram file i.e. ram2048x32_box0.vhd, dpram136x32_box0.vhd, dpram136x32_box1.vhd. Please substitute the name of ram file in ram_box.dcsh also delete WORK directory after every run.

Now replace the LEON files so that new files use these Ram Black Boxes instead of the original behavioral models. This can be done by replacing original tech_tsmc25.vhd with tech_tsmc25-rishi.vhd.

- **cd leon**
- **cp /home/rishi/652/soc/leon2-1.0.12/leon/tech_tsmc25.vhd tech_tsmc25-rishi.vhd**
- **cp tech_tsmc25.vhd tech_tsmc25-org.vhd**
- **cp tech_tsmc25-rishi.vhd tech_tsmc25.vhd**
- **cd syn**
- **cp leon.dcsh leon-org.dcsh**
- **cp /home/tmarwah/soc/soc/syn/leon.dcsh leon.dcsh**
- **rm –r WORK**
- **mkdir WORK**
- **synopsys_tools**
- **dc_shell –f leon.dcsh > zm01.txt**

This is going to take a while. And you can keep checking the output file (zm01.txt) for errors. To get post-synthesis simulation of the Netlist:

- **cd leon**
- **cp ../syn/leon.v .**
- **cp leon.vhd cp_leon.vhd**
- **rm leon.vhd**
- **cp Makefile Makefile-vhdl**
- **cp /home/tmarwah/soc/soc/leon/Makefile_post_synth Makefile_synth**
- **cp Makefile_synth Makefile**
- **cp /home/tmarwah/soc/soc/leon/tsmc18.v .**
- **cp /home/tmarwah/soc/soc/leon/tp*.v .**
- **cp /home/tmarwah/soc/soc/leon/timescale.v .**
- **cp /home/tmarwah/soc/soc/leon/dpram512x36_inst.v .**
- **cp ../ram_tsmc25/ra*.v .**
- **cp ../ram_tsmc25/dp*.vhd .**
- **cp ../ram_tsmc25/ra*.vhd .**
- **cd ..**
- **cd tbench**

Edit the tbgen.vhd file and assign clkperiod as 50ns.

```
File Name: tbgen.vhd
Note: We need to edit the frequency( clkperiod = 50 ; ie freq =20MHz)
 entity tbgen is
  generic (
    msg1      : string := "32 kbyte 32-bit rom, 0-ws";
    msg2      : string := "2x128 kbyte 32-bit ram, 0-ws";
    pci       : boolean := false;       -- use the PCI version of leon
    pcihost   : boolean := false;       -- be PCI host
    DISASS    : integer := 0;   -- enable disassembly to stdout
    clkperiod : integer := 50;         -- system clock period
    romfile   : string := "tsource/rom.dat";  -- rom contents
    ramfile   : string := "tsource/ram.dat";  -- ram contents
    sdramfile : string := "tsource/sdram.rec";  -- sdram contents
```

- **cd ..**
- **make clean**
- **mentor_old_tools**
- **make all**
- **vsim tb_func32 &**
- **run –all** (In ModelSim window)

The output from the simulation should be similar to:

```
# *** Starting LEON system test ***
# Register file
# Cache controllers
# Interrupt controller
# UARTs
# Timers, watchdog and power-down
# Parallel I/O port
# Test completed OK, halting with failure
# ** Failure: TEST COMPLETED OK, ending with FAILURE
Simulation is halted by generating a failure.
```

5. Adding an IP Block

Adding an IP block as an AMBA bus master to Leon processor. The IP block used here is the AES block. For AES to act as a bus master, a wrapper has been created that would enable it to communicate with the AMBA buses. Copying the files of IP block into **leon** directory

- **cd leon**
- **cp ../aes/DW_ram*.vhd .**
- **cp ../aes/aes*.v .**
- **cp ../aes/controller.v .**
- **cp ../aes/topmodule.v .**
- **cp ../aes/aes*.vhd .**
- **mv cp_leon.vhd leon.vhd**

In order to include AES block, as bus master, we need to change following files: MCORE.VHD, TARGET.VHD, AMBACOMP.VHD and DEVICE.VHD. The modified files are copied in 'leon' directory.

- **rm ambacomp.vhd mcore.vhd target.vhd device.vhd Makefile**
- **cp ../org_edit/ambacomp-soc.vhd ambacomp.vhd**
- **cp ../org_edit/mcore-soc.vhd mcore.vhd**
- **cp ../org_edit/target-soc.vhd target.vhd**
- **cp /home/rishi/652/soc/SoC-Thesis/leon/device.vhd .**
- **cp /home/tmarwah/soc/soc/leon/Makefile_soc_pre .**
- **cp Makefile_soc_pre Makefile**
- **cp /home/tmarwah/soc/soc/leon/modelsim.ini .**
- **cp /home/tmarwah/soc/soc/leon/artisan_lib.vhd .**
- **cd ..**
- **cp -r /home/rishi/652/soc/leon2-1.0.12/ram_virtex2 .**
- **make clean**

Now we need to change the software for Leon so that we can program the transfer of data from registers in Leon to the memory of our amba master.

- **cd tsource**
- **rm ram.dat**
- **make clean**

- **cp leon_test.c leon_test-org.c**
- **cp ../org_edit/leon_test.c .**
- **bash**

REMEMBER, THE SPARC/RTEMS IS INSTALLED IN faster MACHINE ONLY. SO YOU HAVE TO DO THE FOLLOWING BASH PART IN faster ONLY. (THOSE WHO WORKING IN /TMP DIRECTORY, COPY THE TSOURCE DIRECTORY TO YOUR HOME DIRECTORY AND DO THE SAME).

In response to the bash prompt, please set following path:

- **export PATH=$PATH:/opt/rtems/bin**
- **make all**

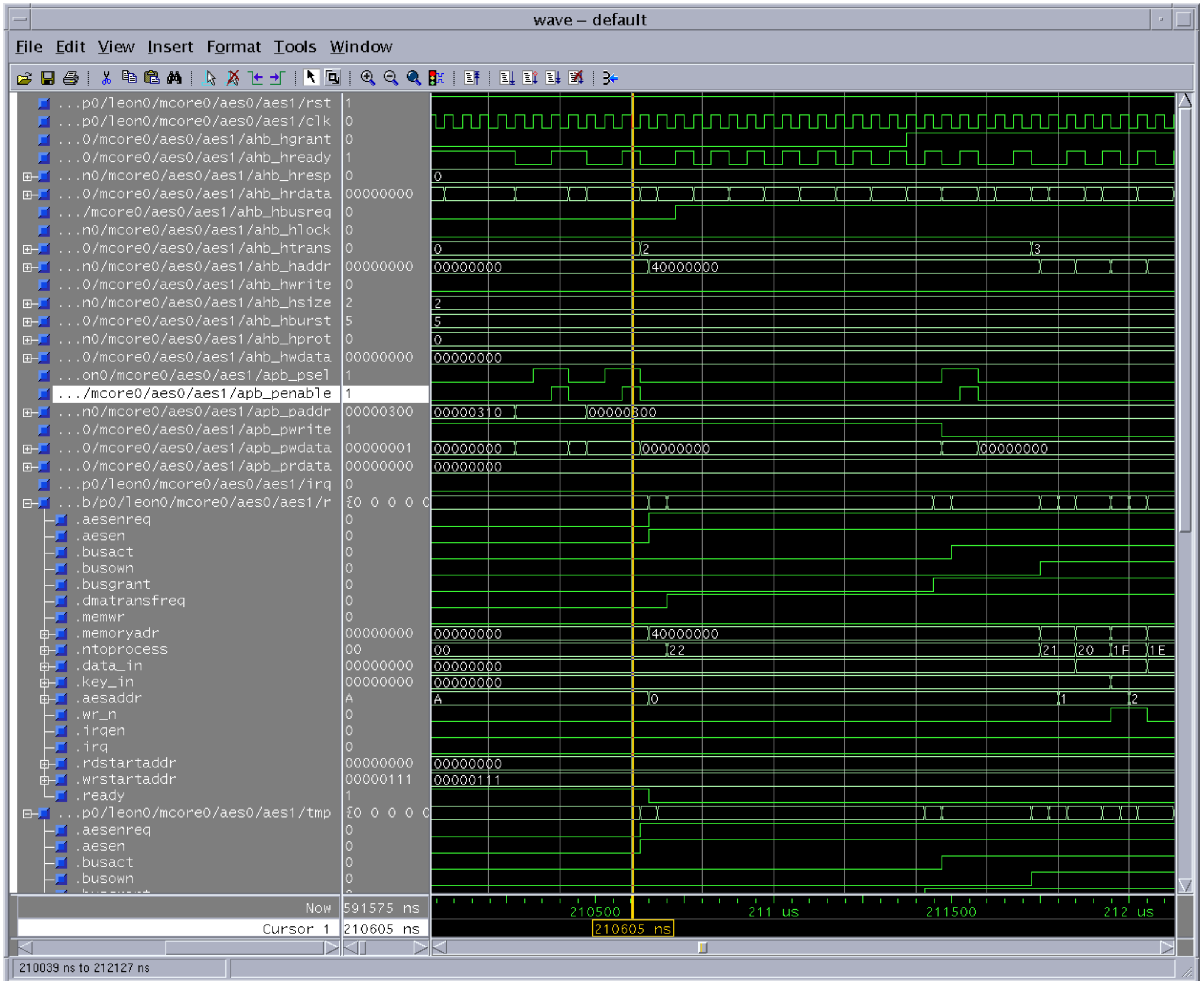
After make all, it should compile without errors.

- **exit**

Now we have set all the files and we can simulate the design. All the relevant signals can be seen by running the wave file leon1.do.

- **cd ..** (top 'soc' directory)
- **cp /home/tmarwah/soc/soc/leon1.do .**
- **mentor_old_tools**
- **make all**
- **vsim tb_func32&**
- **do leon1.do** (in the modelsim window)
- **run -all** (in the modelsim window)

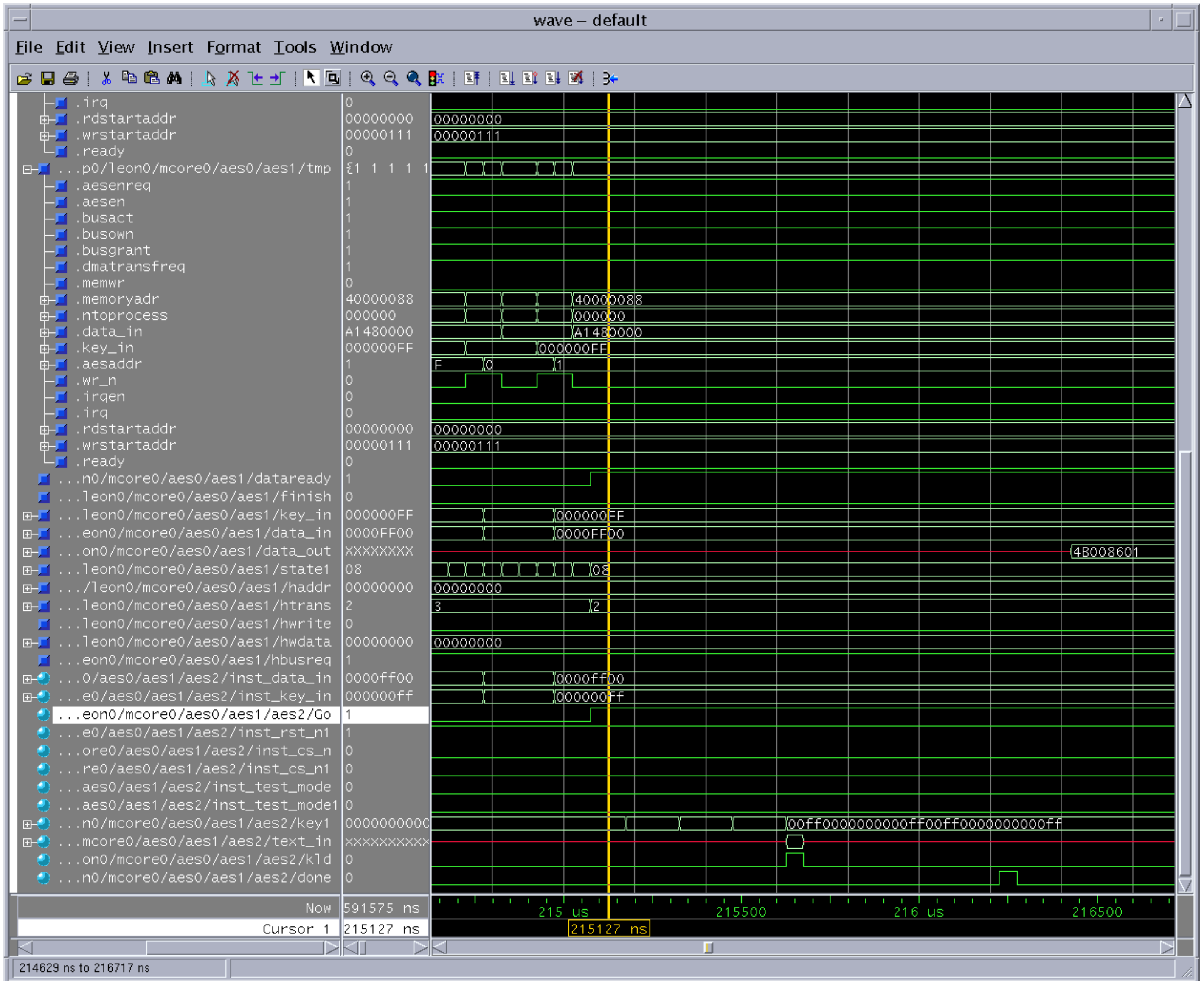
"make all" should run without any errors. If it does then all your files are set. You can see how data communication is taking place between master and leon by watching the simulation results of signals in aes_ctrl module. Finally, you will see the compilation of the top level modules. The waveforms are shown below:



```
...p0/leon0/mcore0/aes0/aes1/rst            1
...p0/leon0/mcore0/aes0/aes1/clk            0
...0/mcore0/aes0/aes1/ahb_hgrant            0
...0/mcore0/aes0/aes1/ahb_hready            1
...n0/mcore0/aes0/aes1/ahb_hresp            0
...0/mcore0/aes0/aes1/ahb_hrdata            00000000
.../mcore0/aes0/aes1/ahb_hbusreq            0
...n0/mcore0/aes0/aes1/ahb_hlock            0
...0/mcore0/aes0/aes1/ahb_htrans            0
...n0/mcore0/aes0/aes1/ahb_haddr            00000000
...0/mcore0/aes0/aes1/ahb_hwrite            0
...n0/mcore0/aes0/aes1/ahb_hsize            2
...0/mcore0/aes0/aes1/ahb_hburst            5
...n0/mcore0/aes0/aes1/ahb_hprot            0
...0/mcore0/aes0/aes1/ahb_hwdata            00000000
...on0/mcore0/aes0/aes1/apb_psel            1
.../mcore0/aes0/aes1/apb_penable            1
...n0/mcore0/aes0/aes1/apb_paddr            00000300
...0/mcore0/aes0/aes1/apb_pwrite            1
...0/mcore0/aes0/aes1/apb_pwdata            00000001
...0/mcore0/aes0/aes1/apb_prdata            00000000
...p0/leon0/mcore0/aes0/aes1/irq            0
...b/p0/leon0/mcore0/aes0/aes1/r            {0 0 0 0 0
   .aesenreq                                0
   .aesen                                   0
   .busact                                  0
   .busown                                  0
   .busgrant                                0
   .dmatransfreq                            0
   .memwr                                   0
   .memoryadr                               00000000
   .ntoprocess                              00
   .data_in                                 00000000
   .key_in                                  00000000
   .aesaddr                                 A
   .wr_n                                    0
   .irqen                                   0
   .irq                                     0
   .rdstartaddr                             00000000
   .wrstartaddr                             00000111
   .ready                                   1
...p0/leon0/mcore0/aes0/aes1/tmp            {0 0 0 0 0
   .aesenreq                                0
   .aesen                                   0
   .busact                                  0
   .busown                                  0

                                 Now        591575 ns
                            Cursor 1        210605 ns
```

210039 ns to 212127 ns



```
wave - default
File  Edit  View  Insert  Format  Tools  Window

.irq                                       0
.rdstartaddr                               00000000
.wrstartaddr                               00000111
.ready                                     0
...p0/leon0/mcore0/aes0/aes1/tmp           {1 1 1 1 1
.aesenreq                                  1
.aesen                                     1
.busact                                    1
.busown                                    1
.busgrant                                  1
.dmatransfreq                              1
.memwr                                     0
.memoryadr                                 40000088
.ntoprocess                                000000
.data_in                                   A1480000
.key_in                                    000000FF
.aesaddr                                   1
.wr_n                                      0
.irqen                                     0
.irq                                       0
.rdstartaddr                               00000000
.wrstartaddr                               00000111
.ready                                     0
...n0/mcore0/aes0/aes1/dataready           1
...leon0/mcore0/aes0/aes1/finish           0
...leon0/mcore0/aes0/aes1/key_in           000000FF
...eon0/mcore0/aes0/aes1/data_in           0000FF00
...on0/mcore0/aes0/aes1/data_out           XXXXXXXX
...leon0/mcore0/aes0/aes1/state1           08
.../leon0/mcore0/aes0/aes1/haddr           00000000
...leon0/mcore0/aes0/aes1/htrans           2
...leon0/mcore0/aes0/aes1/hwrite           0
...leon0/mcore0/aes0/aes1/hwdata           00000000
...eon0/mcore0/aes0/aes1/hbusreq           1
...0/aes0/aes1/aes2/inst_data_in           0000ff00
...e0/aes0/aes1/aes2/inst_key_in           000000ff
...eon0/mcore0/aes0/aes1/aes2/Go           1
...e0/aes0/aes1/aes2/inst_rst_n1           1
...ore0/aes0/aes1/aes2/inst_cs_n           0
...re0/aes0/aes1/aes2/inst_cs_n1           0
...aes0/aes1/aes2/inst_test_mode           0
...aes0/aes1/aes2/inst_test_mode1          0
...n0/mcore0/aes0/aes1/aes2/key1           000000000C
...mcore0/aes0/aes1/aes2/text_in           XXXXXXXXXX
...on0/mcore0/aes0/aes1/aes2/kld           0
...n0/mcore0/aes0/aes1/aes2/done           0

Now        591575 ns
Cursor 1   215127 ns

214629 ns to 216717 ns
```

6. References

- Srivastava, R., ``Development of An Open Core System-on-Chip Platform'' , M.S. Thesis, University of Tennessee, August 2004.

- AMBA documentation : http://www.gaisler.com/doc/amba.pdf

- LEON-2 processor User's Manual : http://www.gaisler.com

- Artisan Components,Inc. "Generator User Manual"