EE520 | EE620 Project 3 Assignment

Assigned: Friday, November 05, 2021

Project 3 Due: Monday, December 06, 2021, 11:59 PM

The grade for this project will be based on:

1. Completion of the EDA Tutorial 3 assignment by indicated date and time - No Exceptions!
2. Completion of the Project 3 assignment by indicated date and time - No Exceptions!
3. Live code review of arb.v and arb_test.v by student with lab instructor(s).
4. The following files uploaded to the mycourses Project 3 Report dropbox:
    a. The Project 3 Engineering Report (use template *EE520_EE620_tech_memo.doc* on mycourses), uploaded as a PDF copy to the mycourses Project 3 Report drop box (PDF only), due: Monday, December 06, 2021, 11:59 PM.
    b. arb_test.v, uploaded as arb_test.v.txt to the mycourses Project 3 Report drop box, due: Monday, December 06, 2021, 11:59 PM.
    c. arb.v, uploaded as arb.v.txt to the mycourses Project 3 Report drop box, due: Monday, December 06, 2021, 11:59 PM.
    d. <logon_ID>_test.asm, uploaded as <logon_ID>_test.asm.txt to the mycourses Project 3 Report drop box, due: Monday, December 06, 2021, 11:59 PM.
    e. The Project 3 database copy, uploaded to the mycourses Project 3 Database drop box, due: Monday, December 06, 2021, 11:59 PM.
    f. NOTE: Due to the size of the files submitted, the mycourses Project 3 Report dropbox is configured to only keep one submission, therefore if you complete 3 submissions, only the files included with the last submission are kept. However, a submission is one or more files, therefore all required Project 3 files must be uploaded simultaneously as one submission to the mycourses dropbox.

Revision History:

V14 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V13 R2	Updates for EE520 | EE620 by: Mark A. Indovina
V13 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V12 R3	Updates for EE520 | EE620 by: Mark A. Indovina
V12 R2	Memory viewer instructions and other updates for EE520 | EE620 by: Mark A. Indovina
V12 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V11 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V10 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V9 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V8 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V7 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V6 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V5 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V4 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V3 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V2 R1	Updates for EE520 | EE620 by: Mark A. Indovina
V1 R1	Original created for EE520 | EE620 by: Mark A. Indovina

TABLE OF CONTENTS

1. INTRODUCTION

The purpose of Project 3 is to introduce you to top down, system level design, including:

- Design specification
- System partitioning
- RTL design
- Verification
- Logic Synthesis
- Timing Analysis
- Hardware / Software co-design

Project 3 will require you to work with a core level RTL database and perform the following tasks:

a. Create module Memory Access Bus Arbiter (ARB) per specifications in section 4
    i. Design Memory Access Bus Arbiter (ARB) RTL
    ii. Enhance the Memory Access Bus Arbiter (ARB) test bench
    iii. Perform RTL verification
    iv. Perform logic synthesis
    v. Perform test insertion
    vi. Perform detailed timing analysis
    vii. Perform gate level verification of the scan inserted netlist
b. Insert your Memory Access Bus Arbiter (ARB) module into the DTMF Receiver RTL database
    i. Perform RTL verification
    ii. Perform logic synthesis
    iii. Perform test insertion
    iv. Perform detailed timing analysis
    v. Perform gate level verification of the scan inserted netlist
c. Create an assembly language test program per specifications in section 7 and verify program operation using the full DTMF Receiver RTL database.
d. Except for derived data (netlists, log files), all source code created will be managed with *git*.

2. DTMF Receiver RTL DATABASE DESCRIPTION

Please see the file, DTMF_Receiver.pdf, in the project / lab area on mycourses for a description of the DTMF Receiver module. This document provides of an overview of the following:

a. Overview of the DTMF Receiver Core Module
b. Descriptions of each DTMF Receiver Core Module Block
    i. Including Tiny DSP (TDSP)
c. TDSP Assembly Language Instruction Set

3. DATABASE CHECKOUT

To start, you will need to create a work area for the project. Logon to one of the compute servers and create a work area for Project 3 as follows:

```
cd ee620
mkdir proj3
cd proj3
```

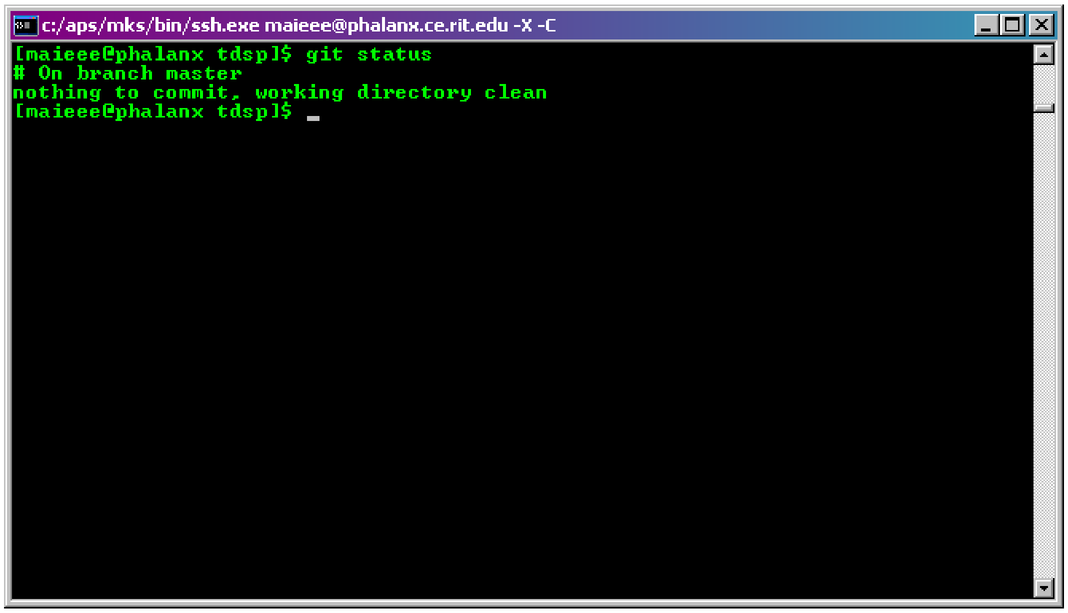
Now let’s check out the tutorial database as follows:

```
git  clone  /classes/ee620/verilog/proj3/tdsp
```

Note that this version of the command does not allow you to use *git* across a network.

Once the database is cloned, change to the repository directory and check the repository status as follows:

```
cd tdsp
git status
```



```
[maieee@phalanx tdsp]$ git status
# On branch master
nothing to commit, working directory clean
[maieee@phalanx tdsp]$ _
```

If the *git status* command does not report “nothing to commit, working directory clean”, something went wrong – ask for assistance.

Throughout Project 3 you must maintain your source code with *git*, in a nutshell you will be using the following commands daily:

- *git status* – to check the status of the repository
- *git add* – to add new files to the repository
- *git commit* – to commit your changes to the repository
- *git show* – show you repository objects
- *git log* – show you commit logs

If you get stuck, *git help* will display of list of *git* commands, and *git help <command>* will display a command manual page.

4. GENERAL GUIDLINES

The database is comprehensive and includes the core DTMF Receiver module, various versions of the TDSP, assembler, behavioral simulation model for the DSP algorithm, etc. When working with the core DTMF Receiver module, you will be working in the directory:

proj3/tdsp/dtmf+humm/

When developing the Memory Access Bus Arbiter (ARB) module, you will be working in the directory:

proj3/tdsp/arb/

Once the Memory Access Bus Arbiter (ARB) module is complete, you will copy the source RTL to:

proj3/tdsp/dtmf+humm/src/

The *TDSP assembler* is provided as a series of programs written in Perl (of high-level, general-purpose, interpreted, dynamic programming languages). Since Perl is interpreted, the programs are provided in source code form. For assembly, you will work in the directory:

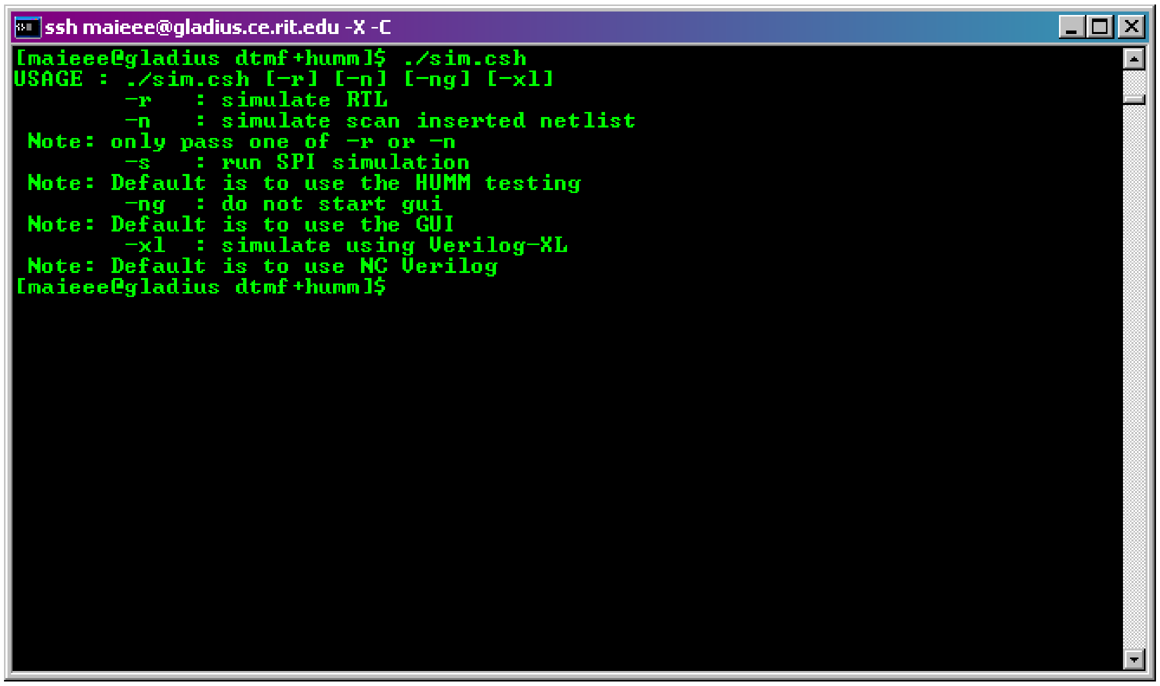
proj3/tdsp/asm/

The assembler program is *tdspasm*, reads assembler programs, *<program name>.asm*, and produces a merged listing, *<program name>.lst*, a symbol table, *<program name>.sym*, and an object file suitable for loading into the RTL database, *<program name>.obj*.

Note that the assembler is flexible in that TDSP instructions can be enabled or disabled in the RTL database to optimize the size of the TDSP. This functionality is available by using the program *getop*. If you're curious ask and we'll show you how to use these additional features.

5. VERIFICATION GUIDLINES

The database contains a simulation script, *sim.csh*, similar to the tutorial. The script options are as follows:



ssh maieee@gladius.ce.rit.edu -X -C

```
[maieee@gladius dtmf+humm]$ ./sim.csh
USAGE : ./sim.csh [-r] [-n] [-ng] [-xl]
        -r    : simulate RTL
        -n    : simulate scan inserted netlist
 Note: only pass one of -r or -n
        -s    : run SPI simulation
 Note: Default is to use the HUMM testing
        -ng   : do not start gui
 Note: Default is to use the GUI
        -xl   : simulate using Verilog-XL
 Note: Default is to use NC Verilog
[maieee@gladius dtmf+humm]$
```

Figure 1 sim.csh options

The script supports RTL and Netlist level simulation with two verification suites:

1. HUMM Test Mode
   This mode executes a memory test suite using the program found in *humm.asm*. This program is assembled and the object file is copied to the verification ROM image *tdsp/dtmf+humm/etc/rom1.txt*.

2. SPI Test Mode (**-s** option as shown in Figure 1)
   This mode executes the DTMF Receiver DSP software suite using the program found in *dtmf_tdsp.asm*. This program is assembled and the object file is copied to the verification ROM image *tdsp/dtmf+humm/etc/rom0.txt*. During this mode, data representing an audio signal is fed to the core via the SPI port.

You will run both modes to verify the DTMF Receiver database.

6. Memory Access Bus Arbiter (ARB) MODULE SPECIFICATIONS

The Memory Access Bus Arbiter (ARB) module coordinates DMA and TDSP access to the Data Sample memory. The protocol is a simple REQUEST, GRANT scheme. Note that the arbiter is biased to allow TDSP priority access if both devices request at the same time. ARB is coded as an explicit state machine. The state encodings are supplied in the include file, *include/arb.h*, and you will use these defines to code your state machine. **WARNING:** you MUST use the state encoding defines as given; you are NOT allowed to use parameters. The bus arbiter is clocked by a 25 Mhz positive edge triggered clock (clk) and is asynchronously reset with an active high reset (reset).

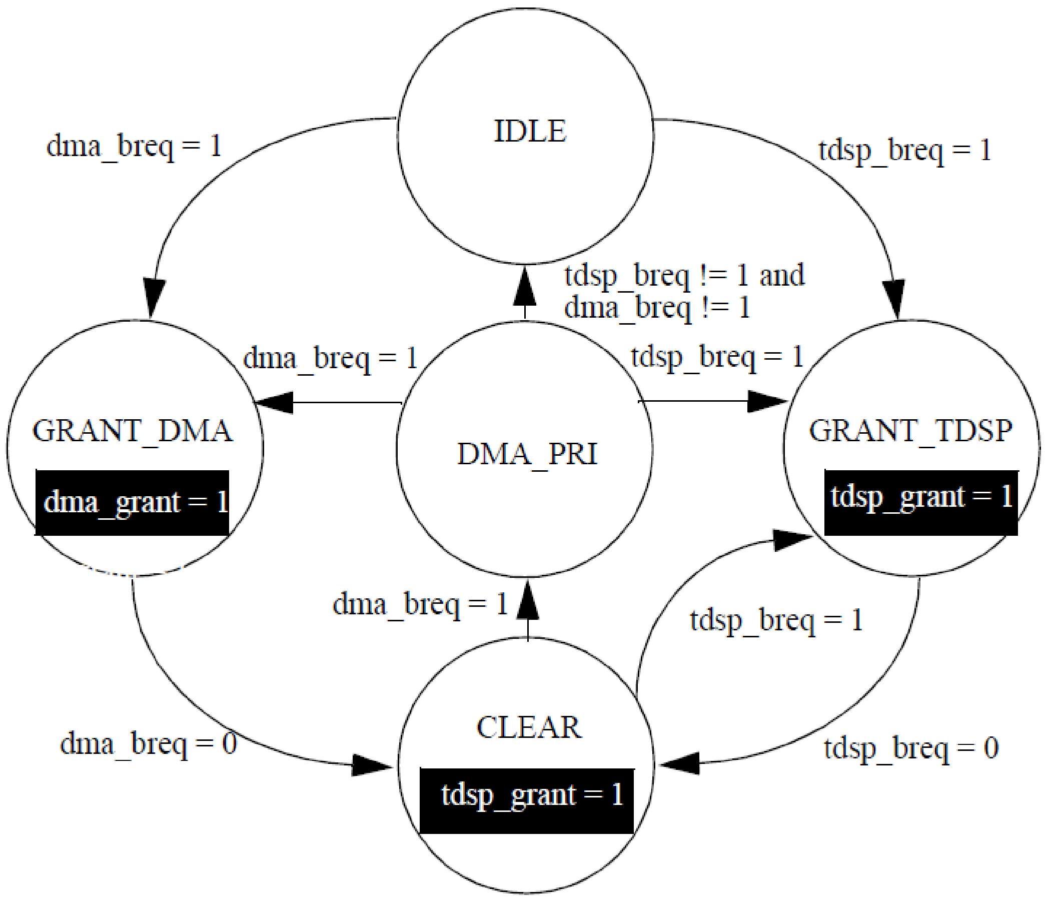


Figure 2 Memory Access Bus Arbiter (ARB) State Machine Diagram

Unless otherwise shown in Figure 2, dma_grant and tdsp_grant are 0. Also, as noted earlier, the tdsp has priority over the dma in all states except the DMA_PRI state.

When developing the Memory Access Bus Arbiter (ARB) module, you will be working in the directory:

proj3/tdsp/arb/

This work area is organized exactly like the tutorial, and should be familiar to you.
Note that the Memory Access Bus Arbiter (ARB) module RTL code will initially be created as the file:

proj3/tdsp/arb/src/arb.v

And as noted in the project requirements, you are to also enhance the testbench:

proj3/tdsp/arb/src/arb_test.v

Your enhancements are to add automatic checking to confirm that the RTL functions properly. Open the current testbench and note the following section:

```
repeat (256)
        begin
        @(posedge clk)
                dma_wait <= $random ;
                tdsp_wait <= $random ;
        fork
        dma_request ;
        tdsp_request ;
        join
        repeat (4)
                @(posedge clk) ;
        end
```

This section is used to randomly generate Memory Access Bus Arbiter (ARB) requests, but provides no thorough checking that a grant occurred other than counting grants.

You are to add logic in the testbench which will check and confirm that grants are generated by your Memory Access Bus Arbiter (ARB) logic after an appropriate number of cycles as described in the Memory Access Bus Arbiter (ARB) state diagram; late grants should report an error as follows:

- Have a 1 bit signal that flags a DMA grant error when active high
- Have a 1 bit signal that flags a TDSP grant error when active high
- When a DMA grant error occurs, report the error (including current simulation time) to the simulator console
- When a TDSP grant error occurs, report the error (including current simulation time) to the simulator console
- Using a `define add capability to conditionally compile in code to stop the simulation if a DMA grant error occurs

- Using a `define add capability to conditionally compile in code to stop the simulation if a TDSP grant error occurs

HINT: create two new tasks, dma_check and tdsp_check which could be similar in form to the current dma_request and tdsp_request; all four tasks would be fired in parallel by the fork/join. The check tasks DO NOT MAKE USE OF ANY CODE from the dma_request or task_request tasks and should follow an algorithm similar to what is show below as pseudo code for a DMA check task:

- Check if the DMA has a request asserted
  - If not, wait for the request to be asserted
- Check if the TDSP has a grant asserted
  - If so, wait for the TDSP grant to be de-asserted
- Once the TDSP grant is de-asserted, start counting clock cycles until the DMA has a grant asserted
  - If the count exceeds the maximum, generate a DMA error

Pseudo code for a TDSP check task would be similar:

- Check if the TDSP has a request asserted
  - If not, wait for the request to be asserted
- Check if the DMA has a grant asserted
  - If so, wait for the DMA grant to be de-asserted
- Once the DMA grant is de-asserted, start counting clock cycles until the TDSP has a grant asserted
  - If the count exceeds the maximum, generate a TDSP error

Note that you must confirm operation of your testbench changes by forcing the grants low using debug features built into the simulator (<u>and without changing your source code or testbench</u>) and observing that your error checking works as expected. Force example, using the debug features you can ***force*** a signal to be low causing a verification failure.

Carefully check your gate level simulation since glitches in your generated grant signals are not acceptable.

Once you are finished with the Memory Access Bus Arbiter (ARB) module, a copy your arb.v RTL code will be promoted and checked into the core database as:

proj3/tdsp/dtmf+humm/src/arb.v

It is mandatory to use *git* for source control while developing the Memory Access Bus Arbiter (ARB) – you must commit your work.

Once you have promoted a copy your arb.v RTL code to the core database, you must (this list is not exhaustive, see the submission requirements for further details): (i) run the core database RTL simulation in HUMM and SPI Test mode; (ii) synthesize the core database; (iii) collect timing and power data for the core database; (iv) run the core database gate simulation in HUMM and SPI Test mode.

7. TDSP ASSEMBLY LANGUAGE TEST PROGRAM SPECIFICATIONS

Using your assigned bit code from Project 1 & 2, you will create a TDSP assembly language test program which will meet the following specifications:

a. Variable X in data RAM located at the address which is your bit code
    i. X is initialized to 3

b. Variable Y in data RAM located at the address which is your bit code + 1
    i. Y is initialized to 4

c. Variable Z in data RAM located at the address which is your bit code + 2
    i. Z is initialized to 2

d. In a loop that executes 64 times, and where "i" is the loop counter starting at 0, compute the following:
    i. Y = 2*X + Z - 2
    ii. X = Y + 5
    iii. Y = -Y
    iv. Z = Y - i

e. Note that all variables, required and ones you create will reside in **data page 1**.

[CONTINUED NEXT PAGE]

f. You will be using the SPI Test mode of the test bench for simulation; this mode monitors the output of the RCC module and stops the simulation when a '#' key is observed.
    i. For this testbench feature to work, at the end of your program you will need will emulate pressing a '#' key by writing the following sequence to the RCC module:
        1. Write a spectrum that emulates "quiet tone[1]
        2. Delay ~170 clock cycles[2]
        3. Write a spectrum that emulates "quiet tone"
        4. Delay ~170 clock cycles
        5. Write a spectrum that emulates a '#' key
        6. Delay ~170 clock cycles
        7. Write aspectrum that emulates a '#' key
        8. Delay ~170 clock cycles
        9. Write a spectrum that emulates "quiet tone"
        10. Delay ~170 clock cycles
        11. Write a spectrum that emulates "quiet tone"
        12. Note that "delay" means a delay loop to allow the RCC (with appropriate "wait" time between spectrum writes to the RCC as discussed in iii below) hardware to process the spectrum.
        13. Note that the RCC requires a phantom write to an address to initiate processing of the applied spectrum (you might want to revisit the EDA Tutorial 3 RCC simulations).
    ii. See the code section commented 'write out calculated spectrum to "results character conversion block"' in the DTMF Receiver assembly code for an example of how to write spectrum blocks to the RCC module for analysis.
    iii. You may recall is that the RCC requires a certain amount of time to process an input spectrum. You may want to go back and review the RTL simulation waveforms from the RCC database (EDA tutorial 3) to determine how many clock cycles you need to *"wait"* between writing spectrum's to the RCC to allow the RCC block enough time to complete processing one spectrum before writing another.
        1. Hint: "*wait*" by creating a delay routine that burns instruction cycles for a predetermined amount of time.

g. To confirm the expected results of your assembly program, you must write a program in C (preferred), Perl, or Python (not Matlab) that executes the same calculations and computes the final result. Your engineering report should document the expected result, and also provide detailed evidence using waveform snapshots of the calculations on the TDSP and eventually the final computed result (which should match your expected result).

[1] "quiet tone" is a signal with no discernable audio present, the resulting spectrum has zero energy.
[2] 170 clock cycles is an estimate, you must check the operation of the RCC hardware

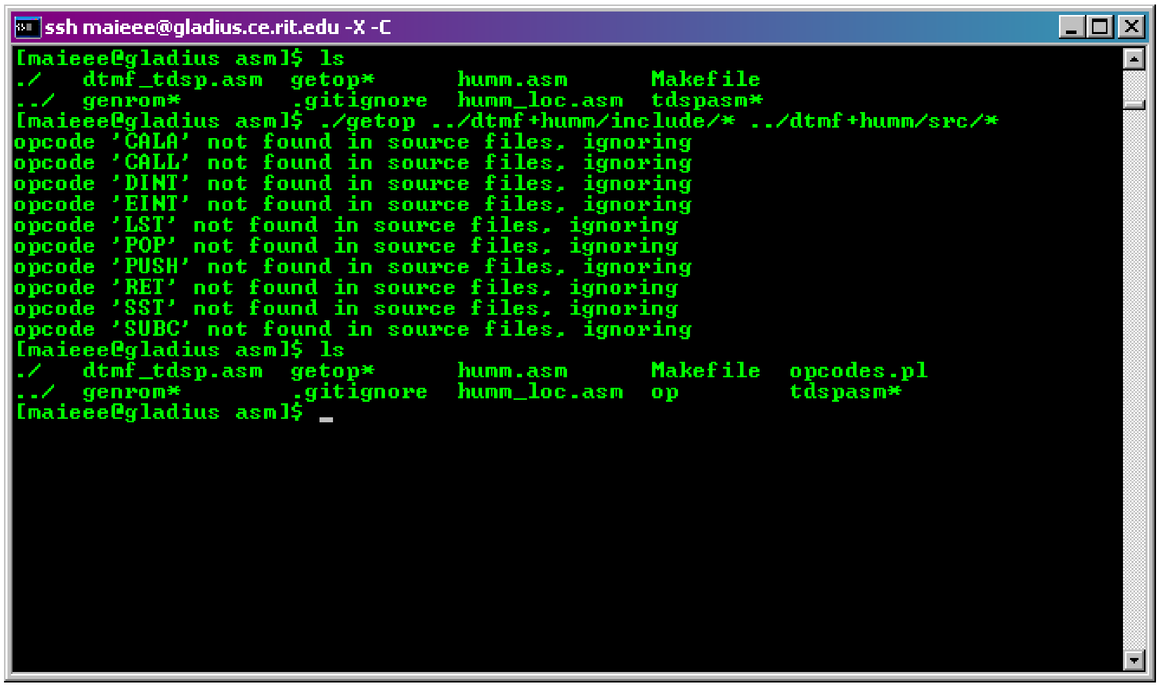
h. For a C program, you can start with the template file, test.c in the asm directory. Once you have the code written, you would compile it by executing: make test and then execute the command ./test to run your program.

Your source code will be named <logon_ID>_test.asm where (<logon_ID> is your Linux system logon ID), and to develop your assembly program, you will work in the assembler directory and use the TDSP assembler:

proj3/tdsp/asm/<logon_ID>_test.asm

To optimize the tdsp for size (silicon area), designers can drop instructions that might not be used in the end signal processing software. To support of this level of flexibility, the assembler supports a configurable instruction set. Instructions included in the tdsp are determined by parsing the tdsp RTL code with the '*getop*' program, allowing users to specifically configure the assembler to support the required instruction set.

When you checked out the database, the assembler instruction set configuration file is not included. To create this file, run the '*getop*' program in the asm directory as shown in Figure 3.



```
[maieee@gladius asm]$ ls
./   dtmf_tdsp.asm  getop*      humm.asm      Makefile
../  genrom*        .gitignore  humm_loc.asm  tdspasm*
[maieee@gladius asm]$ ./getop ../dtmf+humm/include/* ../dtmf+humm/src/*
opcode 'CALA' not found in source files, ignoring
opcode 'CALL' not found in source files, ignoring
opcode 'DINT' not found in source files, ignoring
opcode 'EINT' not found in source files, ignoring
opcode 'LST' not found in source files, ignoring
opcode 'POP' not found in source files, ignoring
opcode 'PUSH' not found in source files, ignoring
opcode 'RET' not found in source files, ignoring
opcode 'SST' not found in source files, ignoring
opcode 'SUBC' not found in source files, ignoring
[maieee@gladius asm]$ ls
./   dtmf_tdsp.asm  getop*      humm.asm      Makefile  opcodes.pl
../  genrom*        .gitignore  humm_loc.asm  op        tdspasm*
[maieee@gladius asm]$ _
```

Figure 3 Running getop program to configure TDSP instruction set

Alternatively, you can just execute 'make opcodes' to generate the assembler instruction set configuration file.

Building assembly programs is done using the utility '*make*' to determine automatically which pieces of a program need to be built, and issue the appropriate commands found in the Makefile to build them. Following the conventions found in the Makefile (specifically tabs), add a target to build your assembly source:

proj3/tdsp/asm/Makefile

To build your assembly code, you would simply type '*make*' on the command line. Note that a Makefile typically contains targets, such as '*clean*' which is used to "clean" the work area. You would invoke this target by issuing '*make clean*' on the command line.

As noted earlier, for simulation, you will use the SPI Test mode and to execute your binary within the RTL simulation environment, copy your resulting object file to *tdsp/dtmf+humm/etc/rom0.txt* before launching the simulation.

While the program executes, at a minimum you must trace the following signals to verify program operation:

a. Program Address
b. Program Data Buss
c. Program Read Strobe
d. Data Address
e. Data Buss
f. Data Read Strobe
g. Data Write Strobe
h. Accumulator (ACC)
i. Product Register (P)
j. Memory locations showing your variables in memory (see Appendix 1)

It is mandatory to use *git* for source control while developing the TDSP assembly language test program.

8. NOTES ON TDSP ASSEMBLY LANGUAGE PROGRAMMING

The TDSP assembly language instruction set and assembler syntax is described in the document, DTMF_Receiver.pdf. The document also includes block diagrams of the processor and other DTMF receiver components.

The assembler syntax is fairly common and follows a few simple rules.

Equates are used to provide a symbolic name to a numeric constant, or a Direct Memory Address (DMA).	DMA example: `pow1   =     60`

To initialize a value in memory, you would first load that value to the accumulator, and then store the value to memory	Example, initialize DMA 19 to the value 1
	
	`one  =  19 ; Equate for memory location 19`
	`lack  1    ; Load the accumulator with a`
	`           ; constant of one`
	`sacl  one  ; Store the accumulator low to`
	`           ; memory location one (19)`

The TDSP is a 16-bit machine with a 32-bit accumulator	If you need to save the entire accumulator to memory, you have to save the lower 16-bits using sacl to one memory location, and the upper 16-bits using sach to another memory location.

Direct memory addressing forms the data memory address by concatenating seven bits of the instruction word with the data page pointer. This implements a paging scheme in which each page contains 128 words and therefore the TDSP has two memory pages for DMA addressing.	Example, set the data page pointer to point to page 1:
	
	`page1  =  1  ;  Equate for memory page 1`
	`ldpk    page1`

Indirect addressing forms the data memory address from the least significant eight bits of one of the two auxiliary registers, AR0 or AR1. The auxiliary register pointer (ARP) selects the current auxiliary register for indirect address generation. The auxiliary registers can automatically post increment or post decrement in parallel with the execution of any indirect instruction to permit single-instruction-cycle manipulation of data structures in memory.	Indirect addressing syntax: * Use AR(ARP) for the instruction address *+ Use AR(ARP) for the instruction address, then post increment AR(ARP) *- Use AR(ARP) for the instruction address, then post decrement AR(ARP)

If the state of the auxiliary register pointer (ARP) is unknown, define the auxiliary register pointer

Example:

```
larp    ar0 ; Set the auxiliary register
            ; pointer to point to ar0
```

The TDSP has a multiplier, however being a machine focused on DSP, the multiplier is organized differently than traditional CPU's. The multiplier has a temporary (T) register used for one operand; the multiply result is stored in a product register

Multiply example:

```
lt  xk  ; Load multiply temporary operand
        ; with data sample, x
mpy  scale   ; Multiply
             ;   xk * scale -> p
ltp  dla1  ; Load multiply temporary
           ; operand with dla1
           ; Move last multiply product to
           ; accumulator
           ;   xk * scale -> ACC
mpy  recf1  ; Multiply
            ;   dla1 * recf1
apac    ; Add last multiply product to
        ; accumulator
        ;   ACC + recf1*dla1 -> ACC
```

To familiarize you with the assembly syntax, three assembly programs are included with the DTMF receiver database:

humm.asm	A simple test program used in testbench “humm” mode
dtmf_tdsp.asm	The DTMF receiver DSP program using the Goertzel algorithm
humm_loc.asm	A simple test program used in testbench “humm” mode

9. NOTES ON DATABASE ORGANIZATION

There is a README file and MANIFEST file at the top level of the database:

MANIFEST.txt	Baseline listing of files checked into repository
README.txt	Brief description of database organization

You might also notice there is a naming convention used for derived files such as reports and netlists. The convention is to add an extension to the name of the file as follows:

<technology><scan?>_<tool?>

Where:

<technology>	identifies the target process technology
<scan?>	optional identifier used to indicate that the design has been processed through test insertion; would typically identify what type of test protocol used, i.e. full-scan (scan), partial-scan, LSSD, etc.
<tool?>	optional identifier used to indicate what tool produced this file
<rpt?>	optional identifier used to indicate which report is contained in this file

For example, a listing of the report files generated during the synthesis, timing and power analysis would be as follows:

Design Compiler logic synthesis, pre-scan reports:
report/dc/dtmf_recvr_core_tsmc18_dc.rpt

Design Compiler test insertion, post scan reports:
report/dc/dtmf_recvr_core_tsmc18_scan_dc.rpt

PrimeTime static timing analysis reports for the test inserted netlist:
report/pt/dtmf_recvr_core_tsmc18_scan_pt_ct.rpt
report/pt/dtmf_recvr_core_tsmc18_scan_pt_rt.rpt
report/pt/dtmf_recvr_core_tsmc18_scan_pt_rclk.rpt
report/pt/dtmf_recvr_core_tsmc18_scan_pt_rac.rpt
report/pt/dtmf_recvr_core_tsmc18_scan_pt_rcv.rpt
report/pt/dtmf_recvr_core_tsmc18_scan_pt_rca.rpt
report/pt/dtmf_recvr_core_tsmc18_scan_pt_rdt.rpt

PrimePowerl power analysis report for the test inserted netlist:
report/pt/dtmf_recvr_core_tsmc18_power_pr.rpt

10. PROJECT ENGINEERING REPORT

Your Project 3 Engineering Report (use template *EE520_EE620_tech_memo.doc* on mycourses) will include the following:

1. Section 1, Introduction
2. Section 2, Memory Access Bus Arbiter (ARB) module
    a. Brief Module Description
    b. arb.v RTL Source Code
    c. arb_test.v test bench Source Code
    d. Screen Shots of RTL Level Simulation Waveforms
        i. Include screen shots of Waveforms and the simulator console to indicate operation of your testbench enhancements.
    e. Screen Shots of Netlist Level Simulation Waveforms
    f. Document the following from the logic synthesis report:
        i. Total cell area for the pre-scan netlist
        ii. Total number of cells in the pre-scan netlist
        iii. Worst case timing path for the pre-scan netlist
        iv. Power consumption (dynamic and leakage) for pre-scan netlist
        v. Total cell area for the post-scan netlist
        vi. Total number of cells in the post-scan netlist
        vii. Worst case timing path for the post-scan netlist
        viii. Power consumption (dynamic and leakage) for the post-scan netlist
        ix. DFT Test Coverage for the post scan netlist
    g. Document the following from the various timing analyzer reports:
        i. Worst case timing path for the post-scan netlist
        ii. Total timing analysis coverage for the post-scan netlist

[CONTINUED NEXT PAGE]

3. Section 3, DTMF Receiver core RTL Database including new Memory Access Bus Arbiter (ARB) module
    a. Screen Shots of RTL Level Simulation Log and Waveforms for HUMM and DTMF Receiver Verification Suites
    b. Screen Shots of Netlist Level Simulation Log and Waveforms for HUMM and DTMF Receiver Verification Suites
    c. Document the following from the logic synthesis report:
        i. Total cell area for the pre-scan netlist
        ii. Total number of cells in the pre-scan netlist
        iii. Worst case timing path for the pre-scan netlist
        iv. Power consumption (dynamic and leakage) for pre-scan netlist
        v. Total cell area for the post-scan netlist
        vi. Total number of cells in the post-scan netlist
        vii. Worst case timing path for the post-scan netlist
        viii. Power consumption (dynamic and leakage) for post-scan netlist
        ix. DFT Test Coverage for the post scan netlist
    d. Document the following from the various timing analyzer reports:
        i. Worst case timing path for the post-scan netlist
        ii. Total timing analysis coverage for the post-scan netlist
4. Section 4, TDSP Assembly Language Test Program
    a. Program Description
    b. Test Program Used to Model ASM Code
    c. Assembly Language Test Program Source Code
    d. Assembly Language Test Program Merged Listing
    e. Assembly Language Test Program Symbol Table
    f. Assembly Language Test Program Object File
    g. Any relevant simulator log files
    h. Screen Shots of Gate Level Simulation Waveforms showing program execution
5. Section 5, Summary and Conclusions
    a. Project Summary
    b. Conclusions including
        i. *What went well*
        ii. What didn't go well
        iii. *What did you learn*

[CONTINUED NEXT PAGE]

Your Project 3 engineering report should be considered an OFFICIAL ENGINEERING REPORT, similar to an engineering report you would submit to an employer, grant review committee, or peer review committee for presentation or publication. In addition to the format specified, take note of the following:

- Sections and sub-sections will be labeled.
- Sections requiring discussion such the "Introduction", "Description", "Summary", and "Conclusions" require at least one good paragraph (or two or three) of lucid text by the author (that would be you).
- Items such as screen shots, tables, included source code, etc., must be labeled with a suitable caption.
- Items that ask you to *document the following from the suitable report* means that you will extract the appropriate information from the report and document what is requested in your Project 3 engineering report. This does not mean that you just copy the tool report into your Project 3 engineering report. If you do, the tool report must be labeled with a suitable caption AND you need to annotate the tool report such that requested item is obvious to the reader. It is inappropriate to assume that the reader is intimately familiar with your work.

Take note that significant points will be deducted for gross errors in the format, and content of your Project 3 engineering report. Points will also be deducted as necessary for gross errors in punctuation, capitalization, spelling, and grammar. Take care with the mechanics of written language as you produce your Project 3 engineering report.

11. PROJECT DATABASE COPY

Once you have completed Project 3, use the script, /classes/ee620/bin/genkit.csh, to create the project database copy for uploading to mycourses. See *mai_520_620-database-upload.pdf* for more details.

12. PROJECT GRADING

As noted earlier, the grade for this project will be based on:

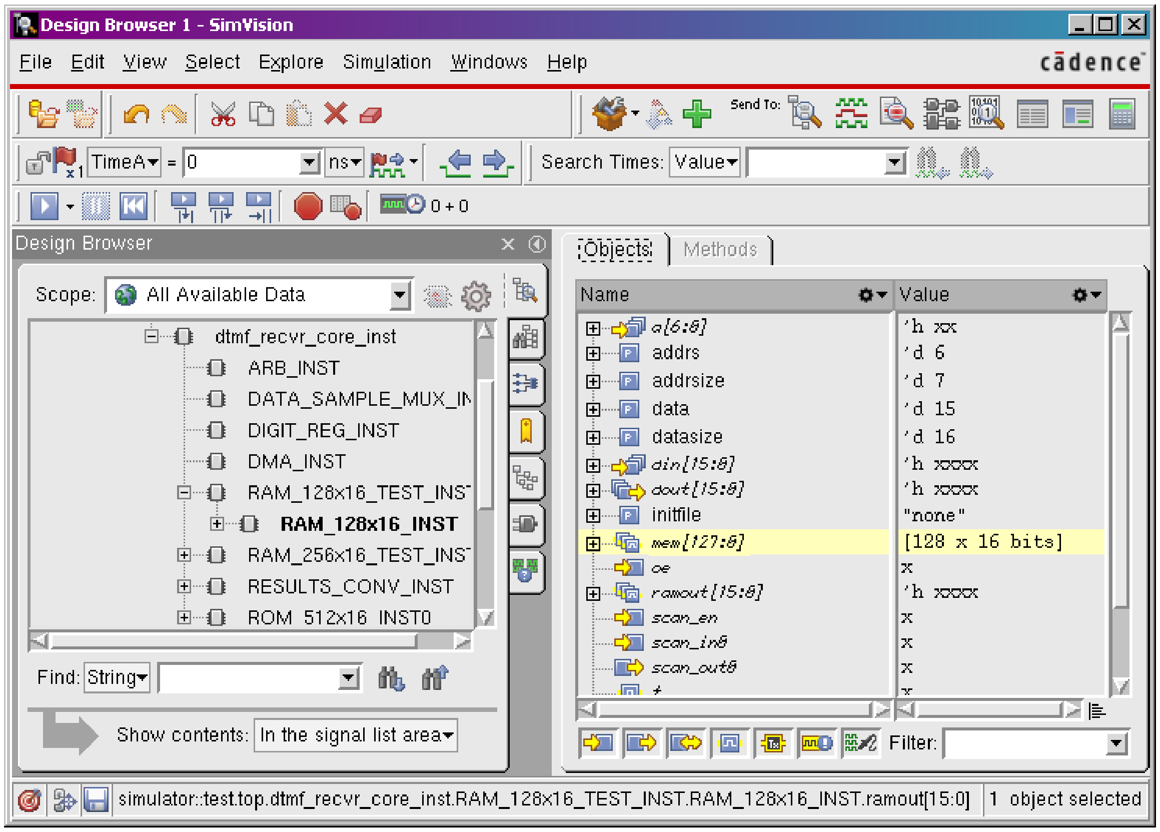
1. Completion of the EDA Tutorial 3 assignment by indicated date and time - No Exceptions!
2. Completion of the Project 3 assignment by indicated date and time - No Exceptions!
3. Live code review of arb.v and arb_test.v by student with lab instructor(s).
4. The following files uploaded to the mycourses Project 3 Report dropbox:
    a. The Project 3 engineering report, uploaded as a PDF copy to the mycourses Project 3 Report drop box (PDF only), due: Monday, December 06, 2021, 11:59 PM.
    b. arb_test.v, uploaded as arb_test.v.txt to the mycourses Project 3 Report drop box, due: Monday, December 06, 2021, 11:59 PM
    c. arb.v, uploaded as arb.v.txt to the mycourses Project 3 Report drop box, due: Monday, December 06, 2021, 11:59 PM
    d. <logon_ID>_test.asm, uploaded as <logon_ID>_test.asm.txt to the mycourses Project 3 Report drop box, due: Monday, December 06, 2021, 11:59 PM
    e. The Project 3 database copy, uploaded to the mycourses Project 3 Database drop box, due: Monday, December 06, 2021, 11:59 PM.
    f. NOTE: Due to the size of the files submitted, the mycourses Project 3 Report dropbox is configured to only keep one submission, therefore if you complete 3 submissions, only the files included with the last submission are kept. However, a submission is one or more files, therefore all required Project 3 files must be uploaded simultaneously as one submission to the mycourses dropbox.

Table 1 Project 3 Grading:

EDA Tutorial 3 Score	5%
Memory Access Bus Arbiter (ARB) Score	30%
DTMF Receiver RTL database Score	10%
TDSP Assembly Language Test Program Score	25%
Project 3 Final Report Score (Including readability, grammar, spelling, format)	25%
Overall Quality Score	5%
Late Deduction	0%
Graded Total	**100%**

APPENDIX I

To open the memory viewer, scope to the data memory:



TimeA = 0 ns | Search Times: Value | 0 + 0

Scope: All Available Data

```
dtmf_recvr_core_inst
    ARB_INST
    DATA_SAMPLE_MUX_IN
    DIGIT_REG_INST
    DMA_INST
    RAM_128x16_TEST_INST
        RAM_128x16_INST
    RAM_256x16_TEST_INST
    RESULTS_CONV_INST
    ROM_512x16_INST0
```

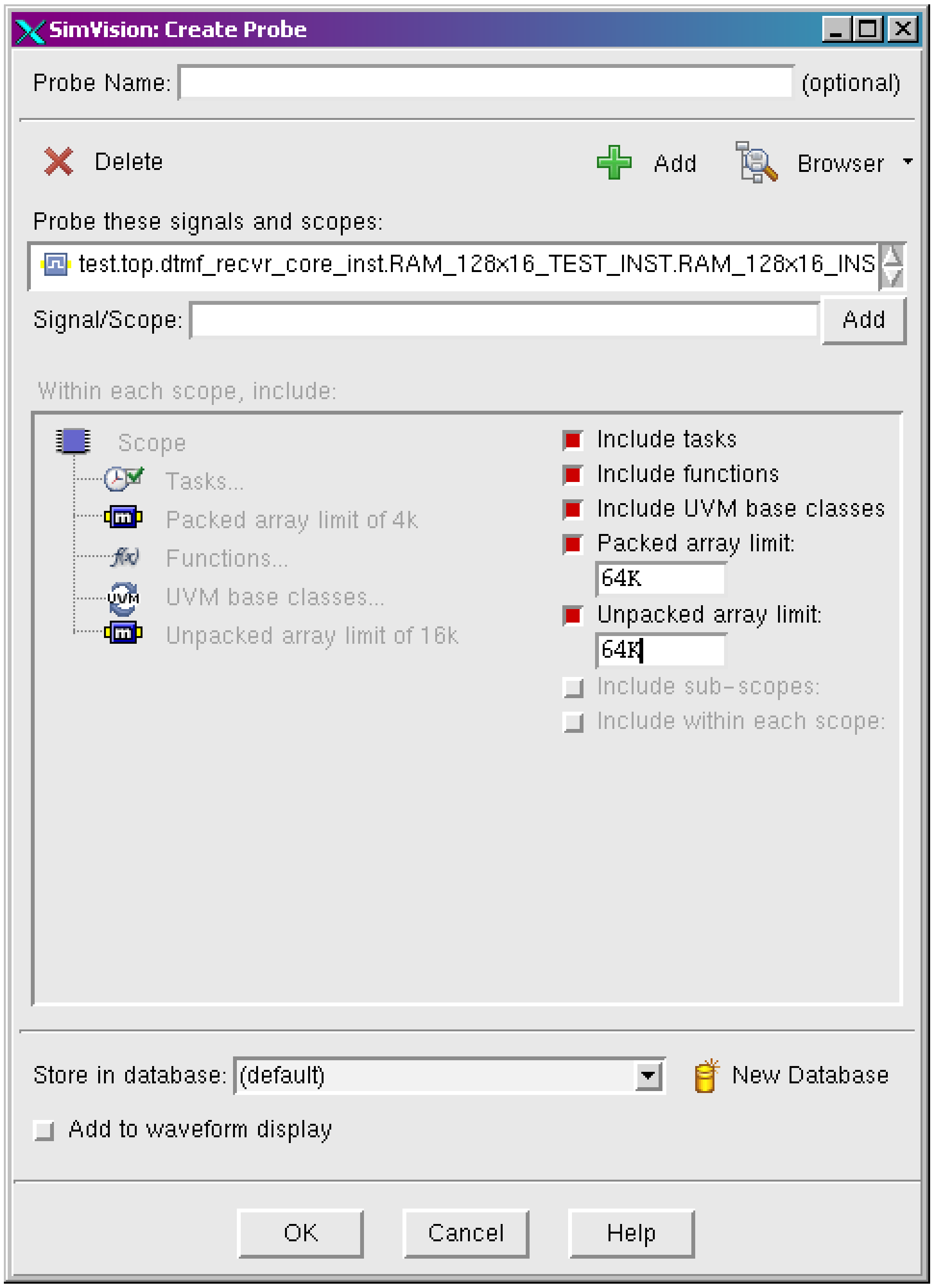
Find: String | Show contents: In the signal list area

Name	Value
a[6:0]	'h xx
addrs	'd 6
addrsize	'd 7
data	'd 15
datasize	'd 16
din[15:0]	'h xxxx
dout[15:0]	'h xxxx
initfile	"none"
mem[127:0]	[128 x 16 bits]
oe	x
ramout[15:0]	'h xxxx
scan_en	x
scan_in0	x
scan_out0	x
t	x

simulator::test.top.dtmf_recvr_core_inst.RAM_128x16_TEST_INST.RAM_128x16_INST.ramout[15:0] | 1 object selected

Figure 4 Design Browser find Data Memory

Right click on **mem** and create a probe:



SimVision: Create Probe

Probe Name: (optional)

Delete Add Browser

Probe these signals and scopes:

test.top.dtmf_recvr_core_inst.RAM_128x16_TEST_INST.RAM_128x16_INS

Signal/Scope: Add

Within each scope, include:

- [✓] Include tasks
- [✓] Include functions
- [✓] Include UVM base classes
- [✓] Packed array limit: 64K
- [✓] Unpacked array limit: 64K
- [ ] Include sub-scopes:
- [ ] Include within each scope:

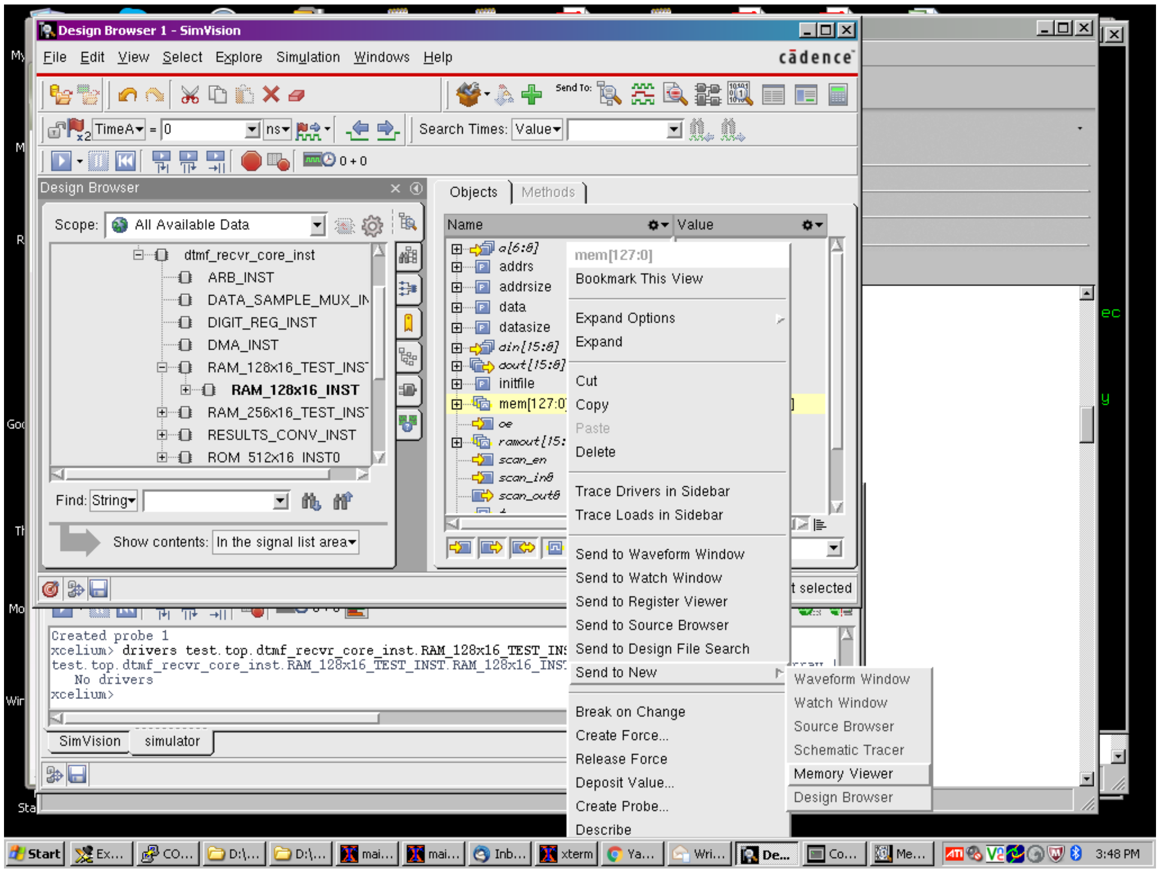
Store in database: (default) New Database

- [ ] Add to waveform display

OK Cancel Help

Figure 5 Probe Memory

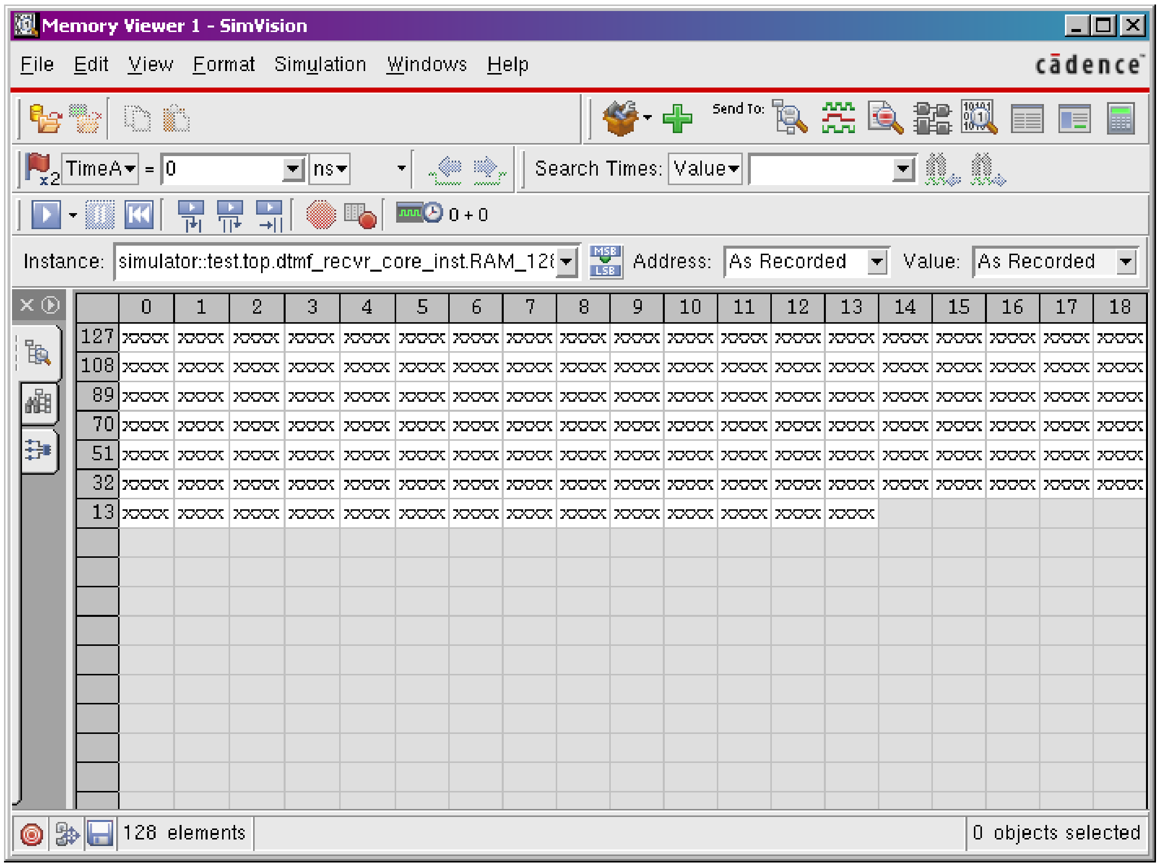
Right click on **mem** again and open a new memory viewer:



```
Created probe 1
xcelium> drivers test.top.dtmf_recvr_core_inst.RAM_128x16_TEST_INS
test.top.dtmf_recvr_core_inst.RAM_128x16_TEST_INST.RAM_128x16_INS
   No drivers
xcelium>
```

Figure 6 Open Memory Viewer

The memory viewer will open:


Memory Viewer 1 - SimVision
File Edit View Format Simulation Windows Help
cādence
Send To:

TimeA = 0 ns

Search Times: Value

0 + 0

Instance: simulator::test.top.dtmf_recvr_core_inst.RAM_12

Address: As Recorded

Value: As Recorded

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
127	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
108	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
89	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
70	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
51	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
32	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
13	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx					

128 elements

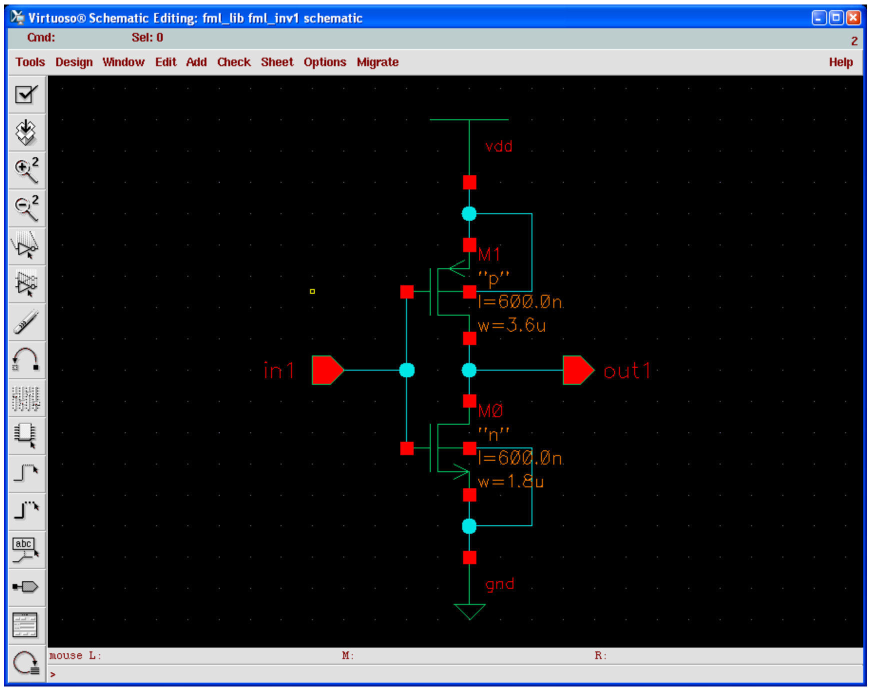
0 objects selected

Figure 7 Memory Viewer

If you open a waveform window for the testbench (right click on **test**, and send to a new waveform window), as you move the cursor in the waveform window, the memory viewer contents will track the cursor. For example, once you open the testbench (**test**) waveforms, you should be able to find the accumulator and track that as your code executes and also track the memory contents as you move the cursor.

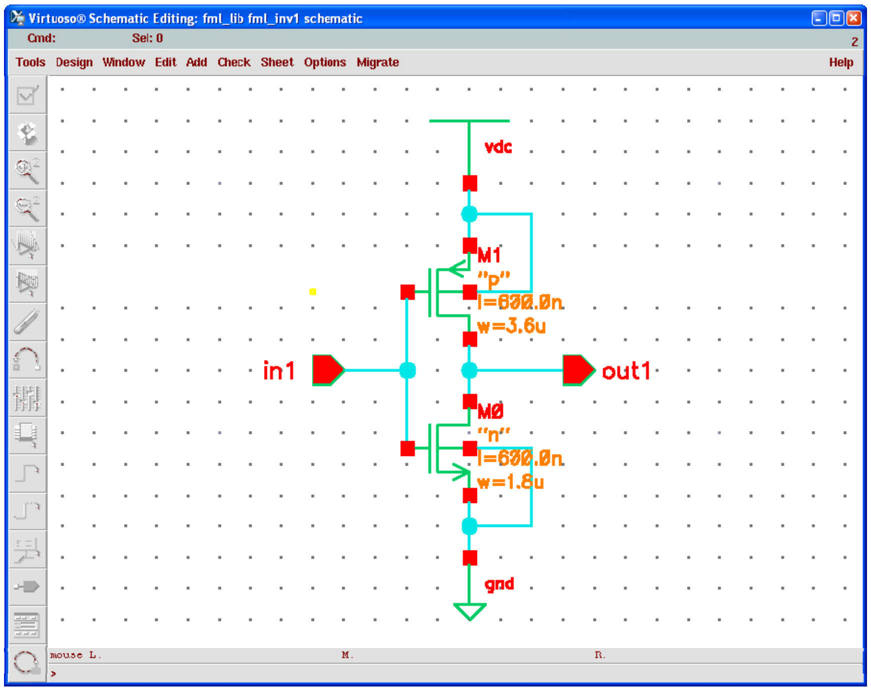
APPENDIX II

How to get rid of the black background for printing and documentation purposes:


Virtuoso® Schematic Editing: fml_lib fml_inv1 schematic
Cmd:
Sel: 0
Tools Design Window Edit Add Check Sheet Options Migrate Help
vdd
M1
"p"
l=600.0n
w=3.6u
in1
out1
M0
"n"
l=600.0n
w=1.8u
gnd
mouse L:
M:
R:

After you copy/paste the window containing the schematic in a word file, using ALT-PRT_SCR, right click on the image and select SHOW PICTURE TOOLBAR.

Once the toolbar opens, towards the right hand side there is a command SELECT TRANSPARENT COLOR. Click on it and then click anywhere on the black background in your window. See the result below.


Virtuoso® Schematic Editing: fml_lib fml_inv1 schematic
Cmd:
Sel: 0
2
Tools
Design
Window
Edit
Add
Check
Sheet
Options
Migrate
Help
vdc
M1
"p"
l=600.0n
w=3.6u
in1
out1
M0
"n"
l=600.0n
w=1.8u
gnd
mouse L.
M.
R.
>