Project 5 Cadence Encounter Test Help

18-765: Digital Systems Testing and Testable Design

For any questions or further information about how the items discussed in this document fit into Project 5, please reference *Project 5 Tutorial: Benchmark Circuit s27*, *Project 5 Scripts Help*, or the help documents within Cadence Encounter Test.

Table of Contents

Introduction

Starting Cadence Encounter Test

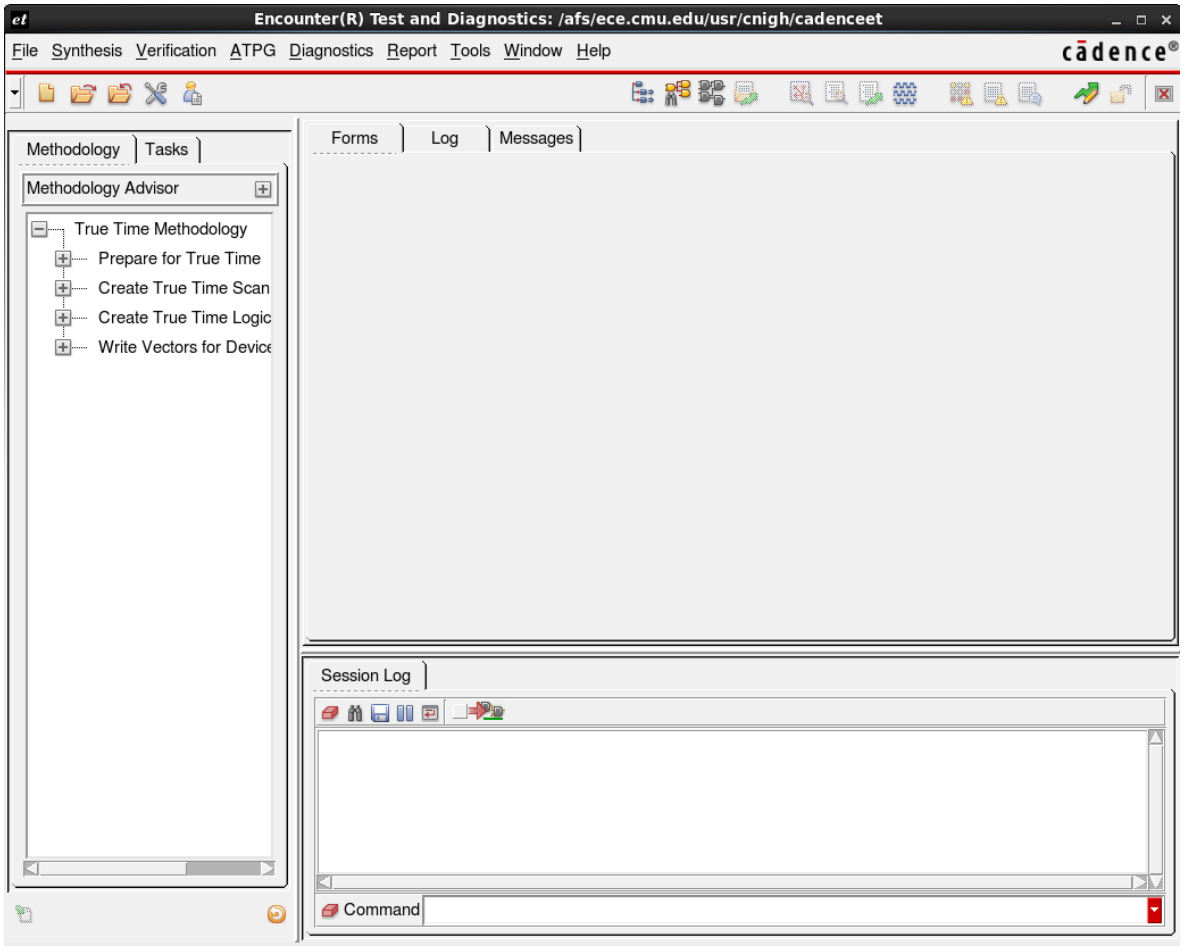
In order to use Cadence Encounter Test, the source file `tools_et` must be copied to your directory. After you have done this, source the file using the following command:

```
source tools_et
```

You may now start the Encounter Test GUI using the following command:

```
et &
```

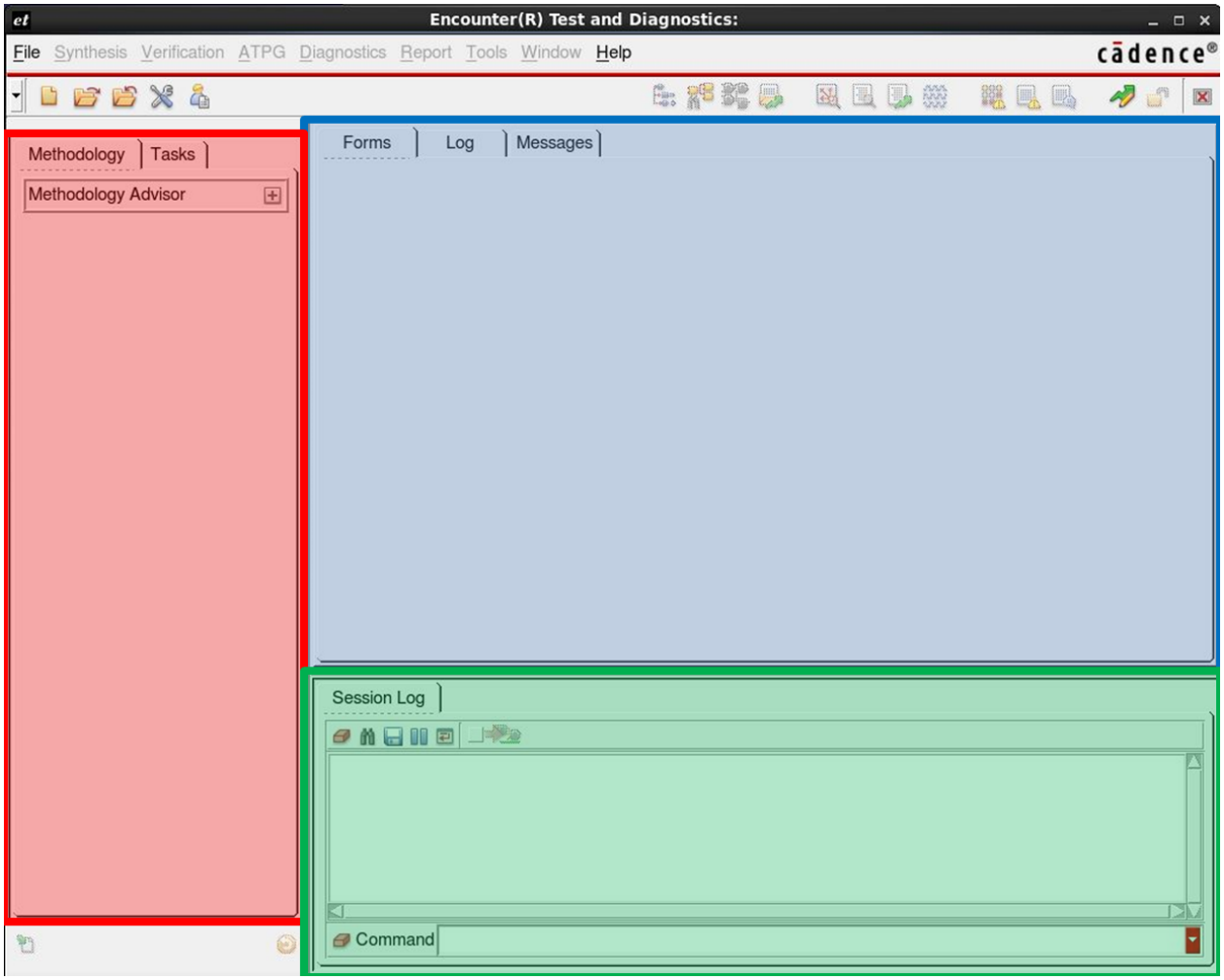
The GUI should appear, looking similar to the image shown below.



Cadence Encounter Test can also be used through command line inputs. This document is specifically for use with the GUI. If you prefer to use the command line to operate Encounter Test you can start it by the terminal command `et -c`. Please reference the Encounter Test Manual for more information about the required command line arguments.
I recommend the GUI because of the clear description of the required files and the ease of use. If you wish, there is also a section of the GUI which accepts command line instructions.

Explaining the GUI

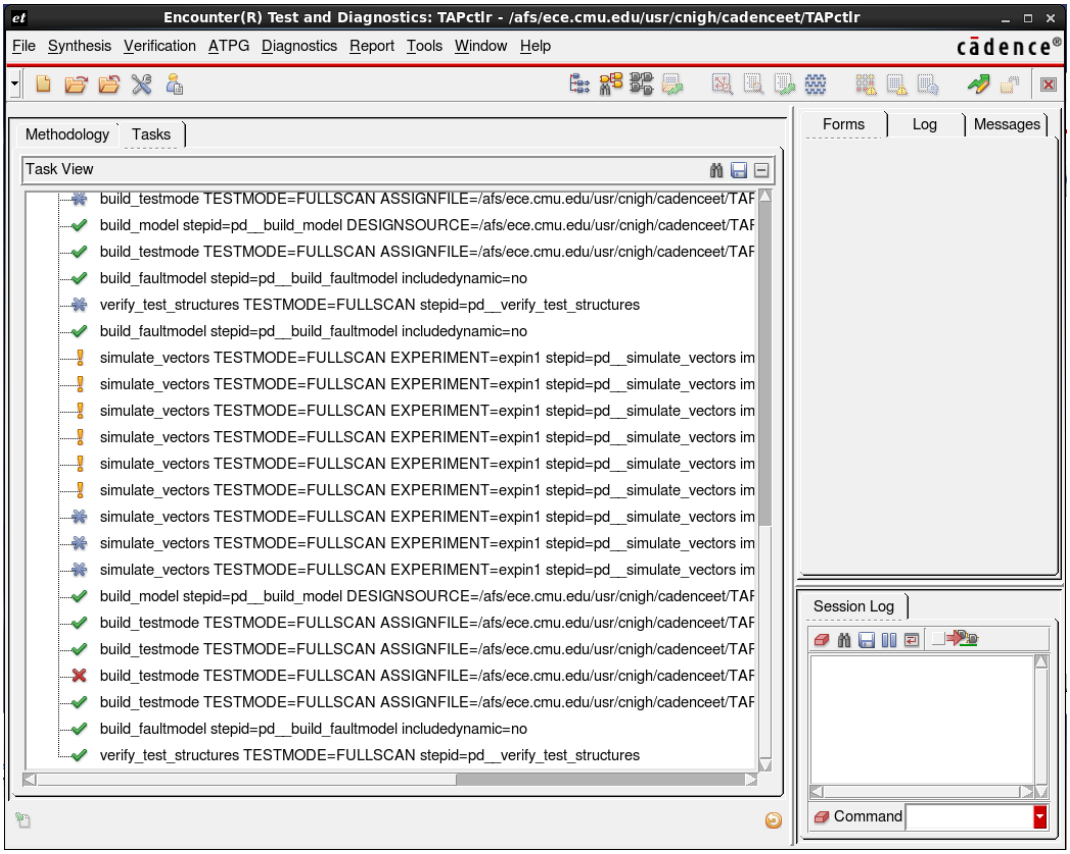
Fortunately, Encounter Test's GUI is quite intuitive and easy to use. However, understanding the different panes of the window and their different purposes can make the tool more easily accessible.



Methodology/Tasks

(Red Pane)

On the left-hand side of the GUI you will see a pane that can be used to view the Encounter Test's defined methodology and the tasks that have been previously run on the project. The *Methodology* tab can be useful as a reminder for the steps and their ordering, but many of the functions which Encounter Test is capable of performing will not be used in this project. For this reason, I recommend primarily using the *Tasks* tab.



The *Tasks* tab allows the user to see the previous actions which have been run and their arguments. The icon on the left of the text corresponds to the final status of the task:

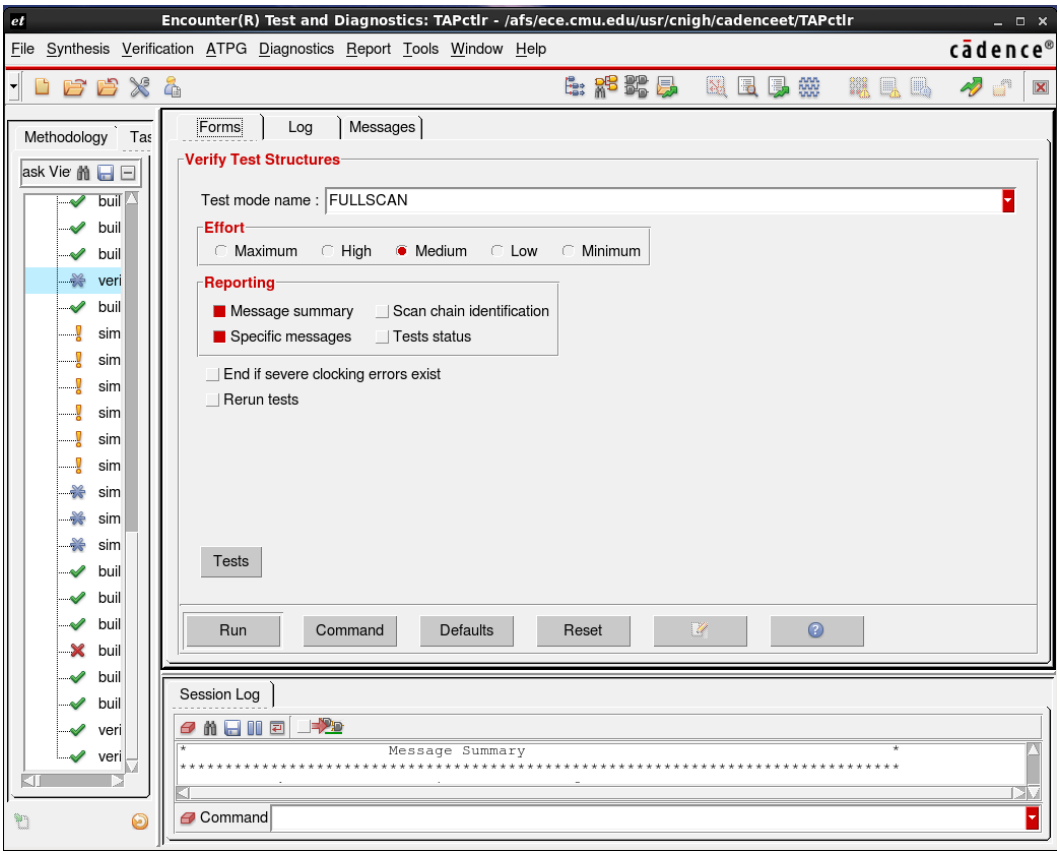
- ✔ : Completed, no warnings
- ✱ : Completed, non-severe warnings
- ! : Completed, severe warnings
- ✖ : Error(s), not completed

Selecting a task from the list will give more information in the *Forms/Log/Messages* portion of the window. This aspect of the GUI will be discussed further in the next section of the report.

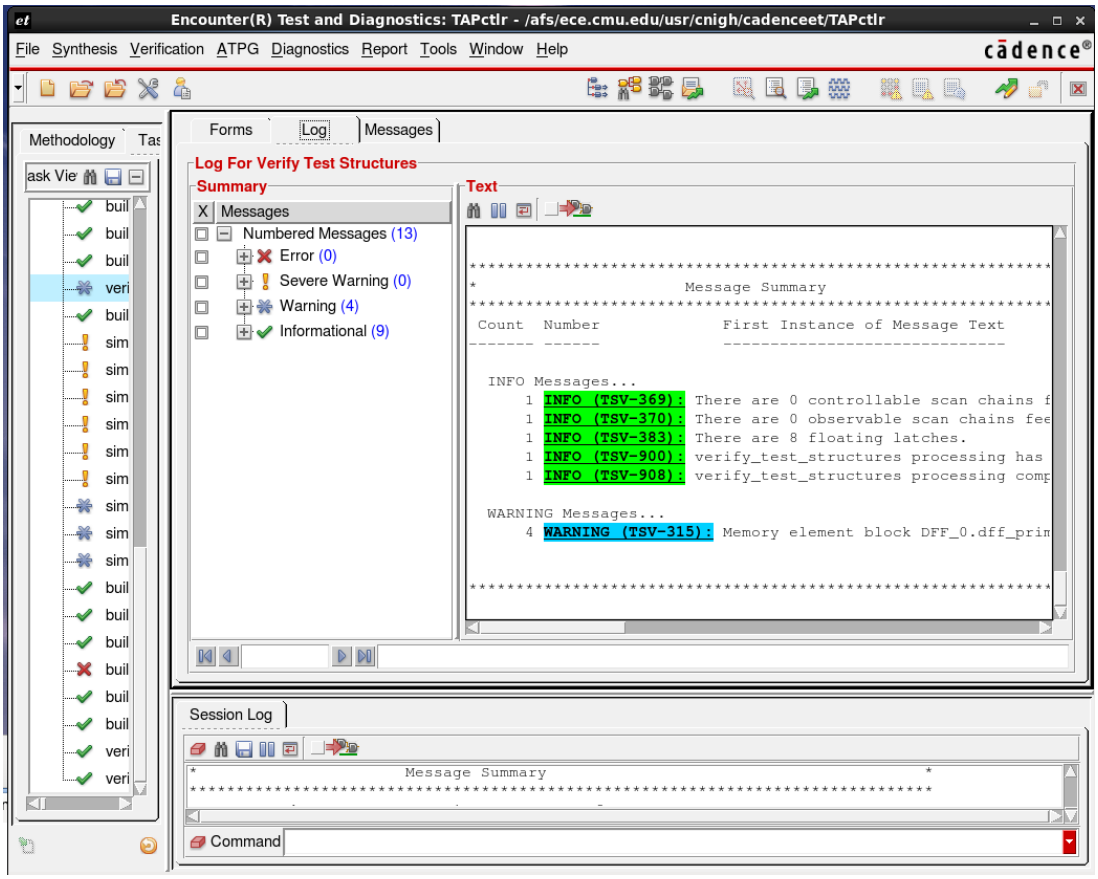
Forms/Log/Messages
(Blue Pane)

The center of the GUI is primarily used in conjunction with the *Tasks* tab discussed earlier. When a specific task is selected from the list, the information displayed by each of the three tabs will change to correspond with that specific run of the task Each of the three tabs can be useful depending on the desired action

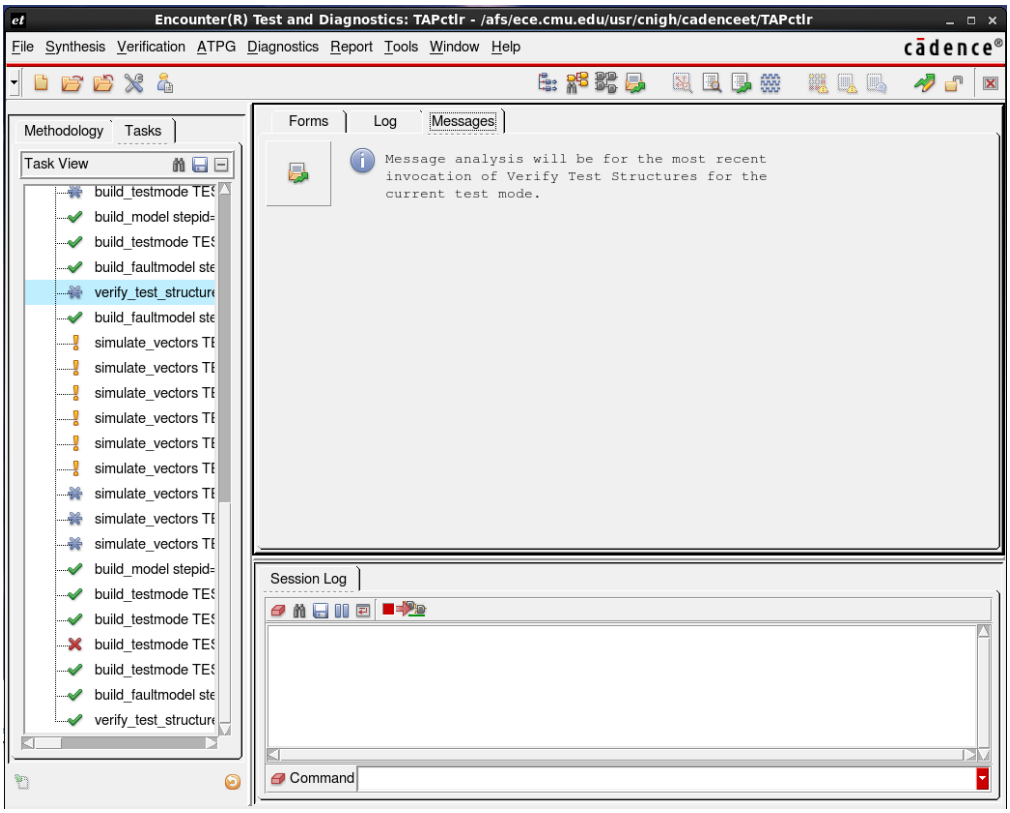
When a task has been selected, the *Forms* tab will change to allow the user to run that same task again. The user can change the inputs of the and run the action right from the main GUI screen without the need for opening other windows.



The *Log* tab allows the user to view the log information about that specific task. This is the same information that appear in the *Session Log* section after the task was being run initially. The presence of this *Log* tab means that the user can view information about tasks which were run on the same project, but during previous sessions.
This pane of the GUI also allows users to navigate through and find where specific messages (informational, warning, error) occurred. Clicking on the highlighted message will open a popup window which gives further information about the message. For warnings and errors this may include helpful portions such as an explanation or the recommended user response.

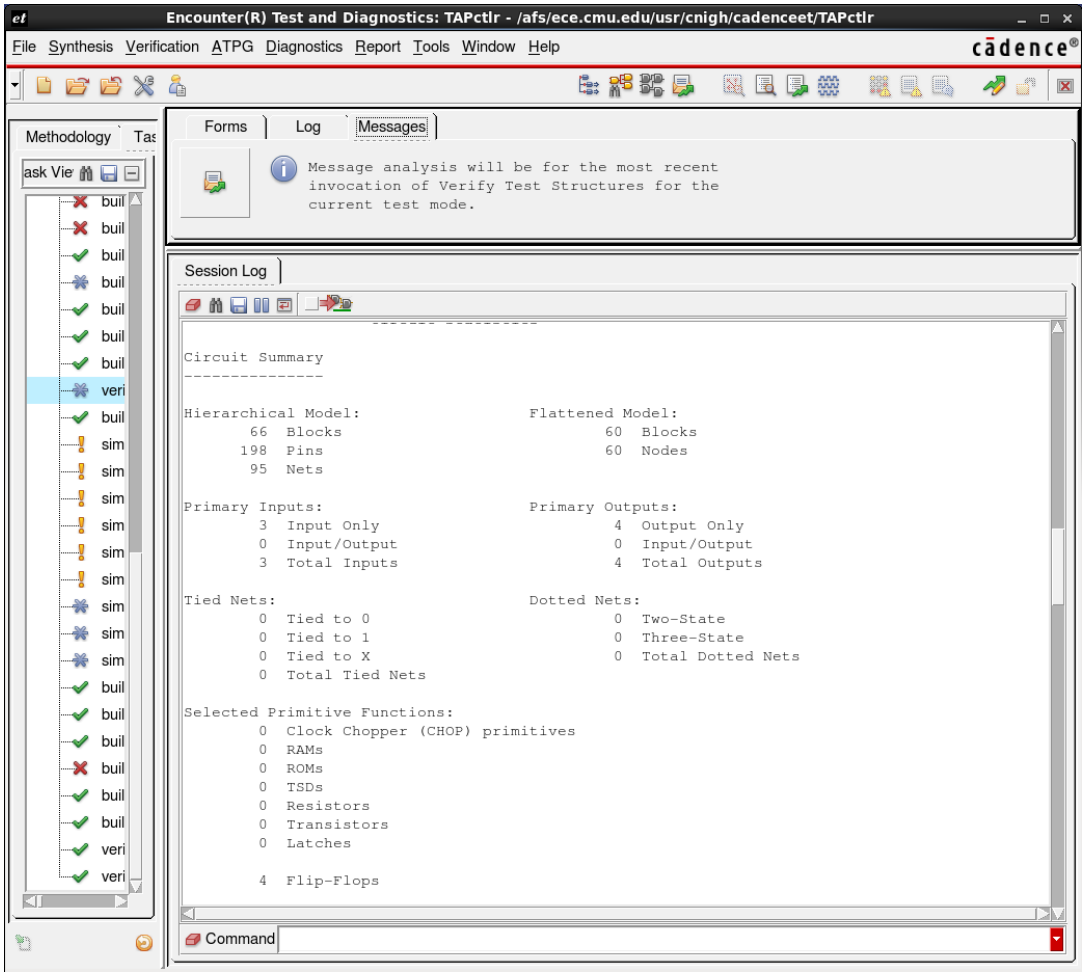


For many of the types of tasks that can be selected, you will find that the *Messages* tab has no function. In fact, of the tasks that I have explored, only *verify_test_structures* can be selected to create useful information on the *Messages* tab.
After selecting a *verify_test_structures* task, clicking on the *View Messages* button will produce a popup window with different messages produced by the task. Double clicking on one of these messages will bring up a further window, and selecting one of the messages from the resulting window and clicking *Analyze* will bring up the schematic of your circuit, often pointing to the problematic area. (If no structures appear after opening the schematic you must locate the portion of the circuit yourself through *View* on the toolbar). This can be extremely helpful when working with a large circuit that returns errors or warnings when attempting to verify the circuit's structures.



Session Log
(Green Pane)

Located at the bottom of the GUI, the *Session Log* creates a log of the tasks that Encounter Test has performed during the current session. As briefly mentioned before, this is the same information that can be viewed by the *Log* tab when a single task has been selected. The *Session Log* produces a realtime output, allowing the user to see how long each part of the running task is taking. At the end of the running of each task there will be a summary printed, showing the different messages that the task has returned. The user can then scroll up to view where these messages occurred in order to further understand their meaning.

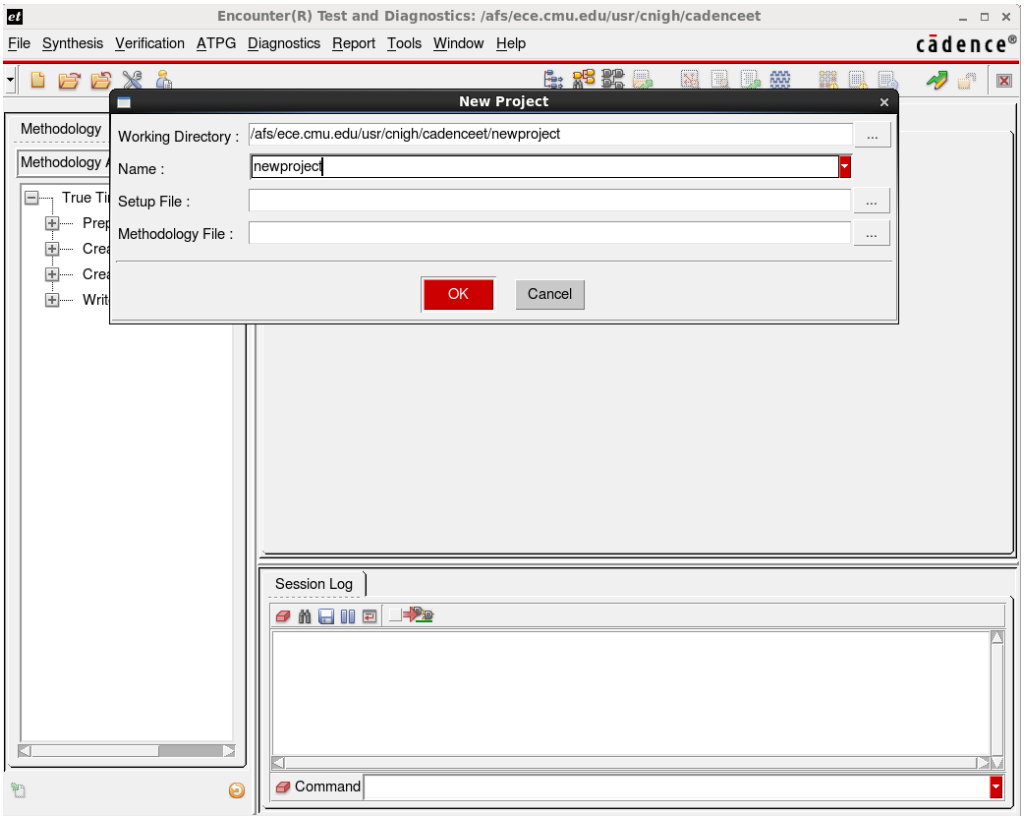


New Project

File ⇒ New ...

User Input: Path for working directory, Project name
Output: Working directory, *tbdata* and *testresults* directories within working directory

The first step required for working with Encounter Test is to create a new project. On the toolbar select *File*, then *New …* . A window will appear entitled *New Project.*
Give the path of the desired working directory for your project. Give the name for a directory which does not already exist, as Encounter Test will create one with the designated name at the designated path. Each project should have its own working directory. To help make it easy to access the correct project when searching through directories, I recommend giving your project the same name as your working directory.



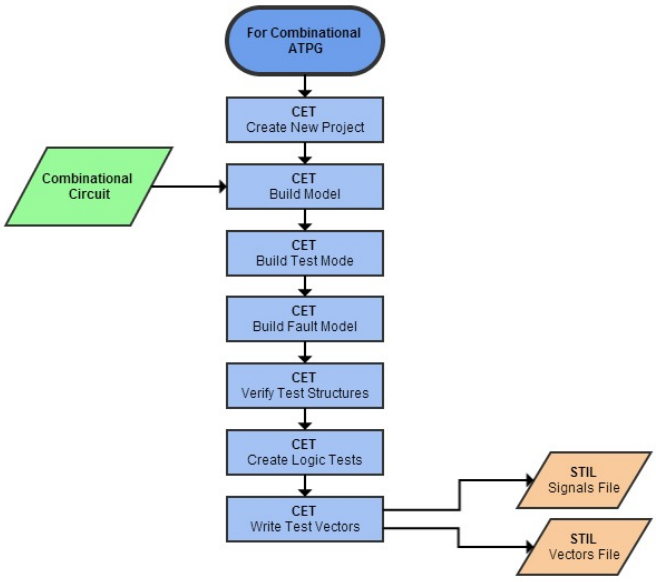
After these two fields have been filled, select *OK*. Outside of Encounter Test, if you then navigate to the defined path you will see the new working directory. Inside of your working directory you will see two new directories entitled *tbdata* and *testresults*. *tbdata* will be used exclusively by Encounter Test, and we don't need to worry about it or its contents.

After creating a brand new project *testresults* will be empty. This directory will store any file-based outputs from Encounter Test. Most notably, it will store the files which contain the test vectors created by the ATPG of Encounter Test. More detail on this will be given when writing test vectors. *testresults* will also house the log files created for all tasks.

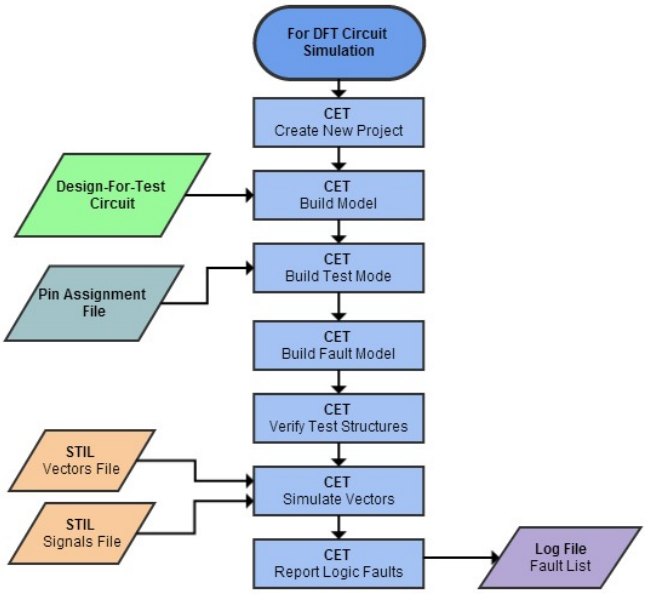
Use for Project 5

Project 5 Encounter Test Process

When completing this project, you will need to use Encounter Test to create and work with a circuit at least twice. Initially, you will need to generate the test vectors for the core circuit by modeling a purely combinational version of the circuit in Encounter Test. The flow for this portion of the project can be found below. Each process block in the chart is described in further detail in the body of this document.



You must also model your newly-designed DFT circuit in Encounter Test. The steps in this process are similar, but you will be simulating test vectors instead of creating them. The flow is shown below.



Building Circuit and Properties

In order to work with circuits in Encounter Test, the program must translate the user-created circuit to a known form. This is done through a series of steps, each of which adds something to the full model of the circuit.

After creating your project, you should add all of the files that are used by your circuit to the working directory. You may have multiple Verilog files which contain modules that are used in conjunction to make up the circuit for your project. Some files, like Verilog primitive definitions, might be used by multiple projects. I would recommend copying this file to each of the working directories so that each project will not need to look outside of its working directory in order to operate.

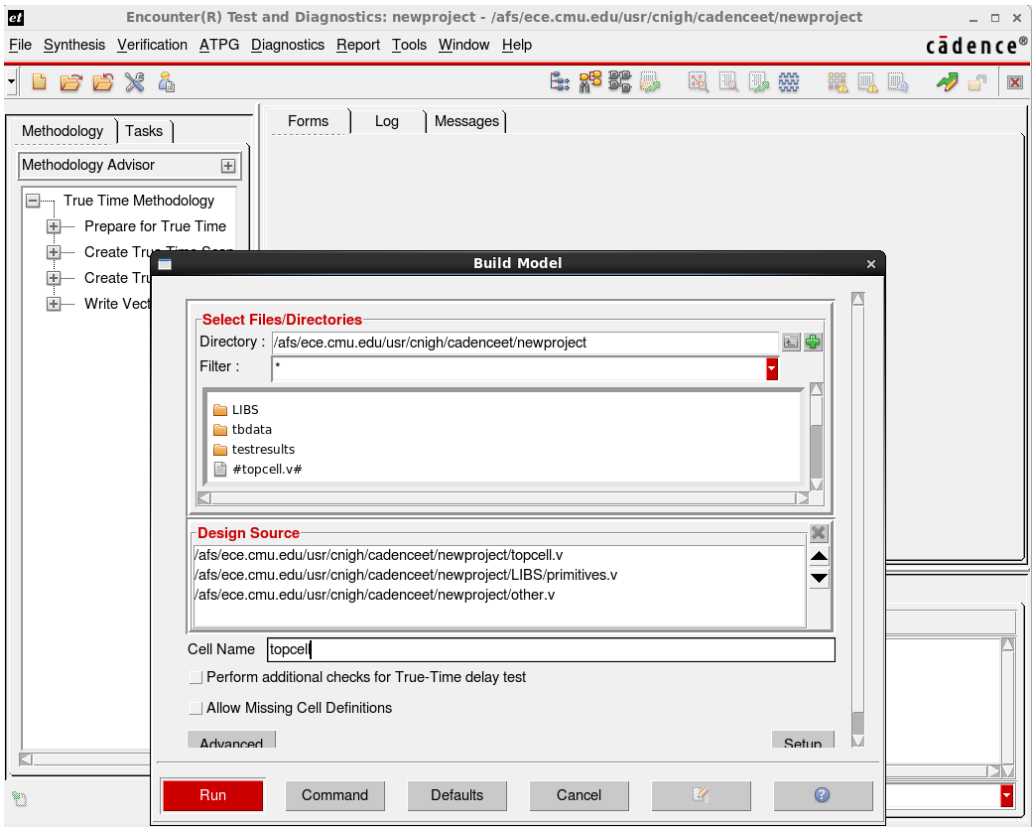
IMPORTANT: Cadence Encounter Test is only able to work with structural Verilog. It can understand the Verilog module hierarchy, but the lowest-level modules should all be in terms of structural primitives. If you attempt to use a procedural implementation for any of your Verilog modules, Encounter Test will not be able to work with that module.

Building the Model for your circuit

Verification ⇒ Build Models ⇒ Model …

User Inputs: Design Source files
Outputs: None

Building the model for your Verilog circuit allows Encounter Test to check your code, compile it, and build an actual schematic model of the circuit.
In order to access this action select *Verification* on the toolbar, then *Build Models* and *Model … .*



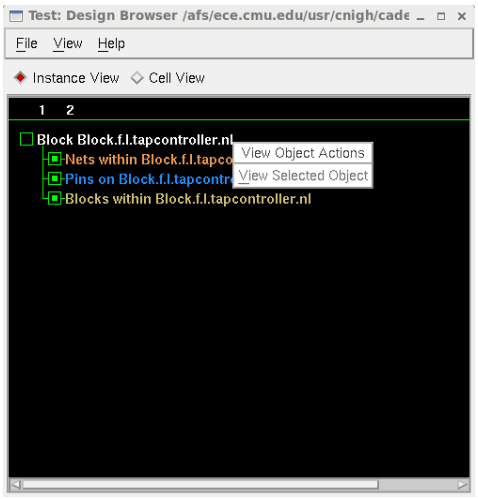
To build the model you must find all of the Verilog files that contribute to your circuit (including primitives, etc.) and add them to the *Design Source* window by clicking on them. Click on the folder icon in order to open folders within the *Select Files/Directories* window. If you have accidentally added an unwanted file to your *Design Source*, you can remove it by selecting it and clicking the X in the upper right corner of the *Design Source* window.
The *Cell Name* field can be filled with the top-cell module of your circuit, or it can be left blank and Encounter Test will attempt to automatically detect the module from your Verilog code.

Viewing Your Circuit

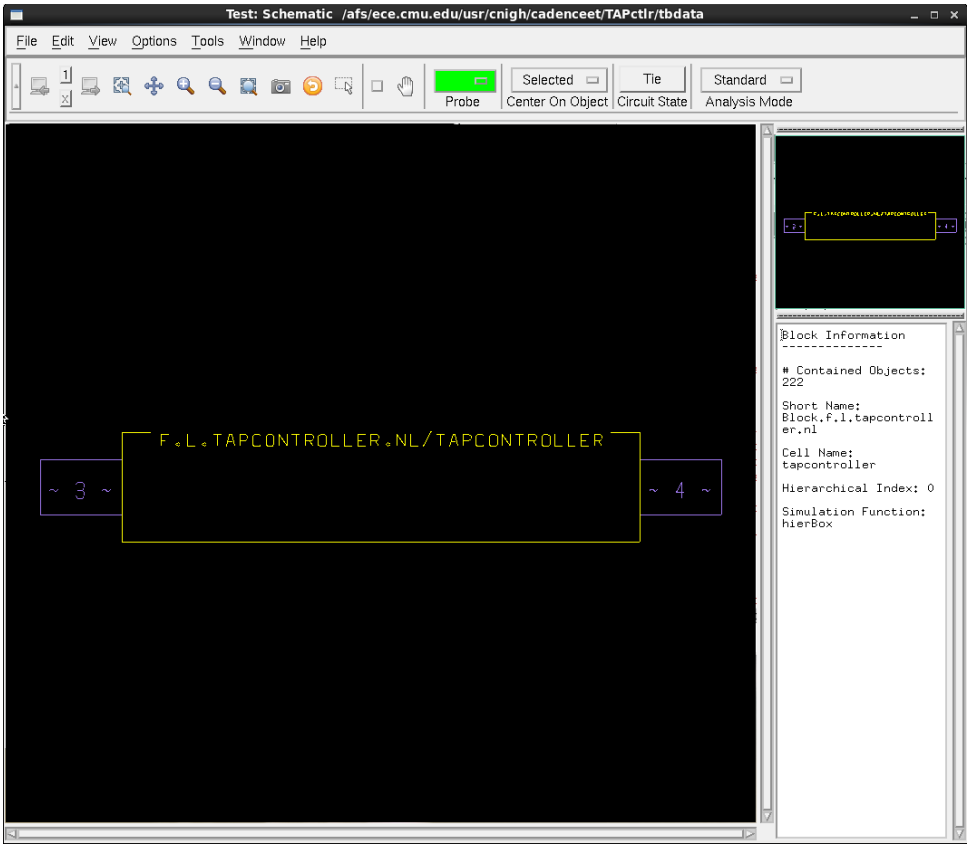
Window ⇒ View ⇒ Design Browser

User Inputs: None
Outputs: None

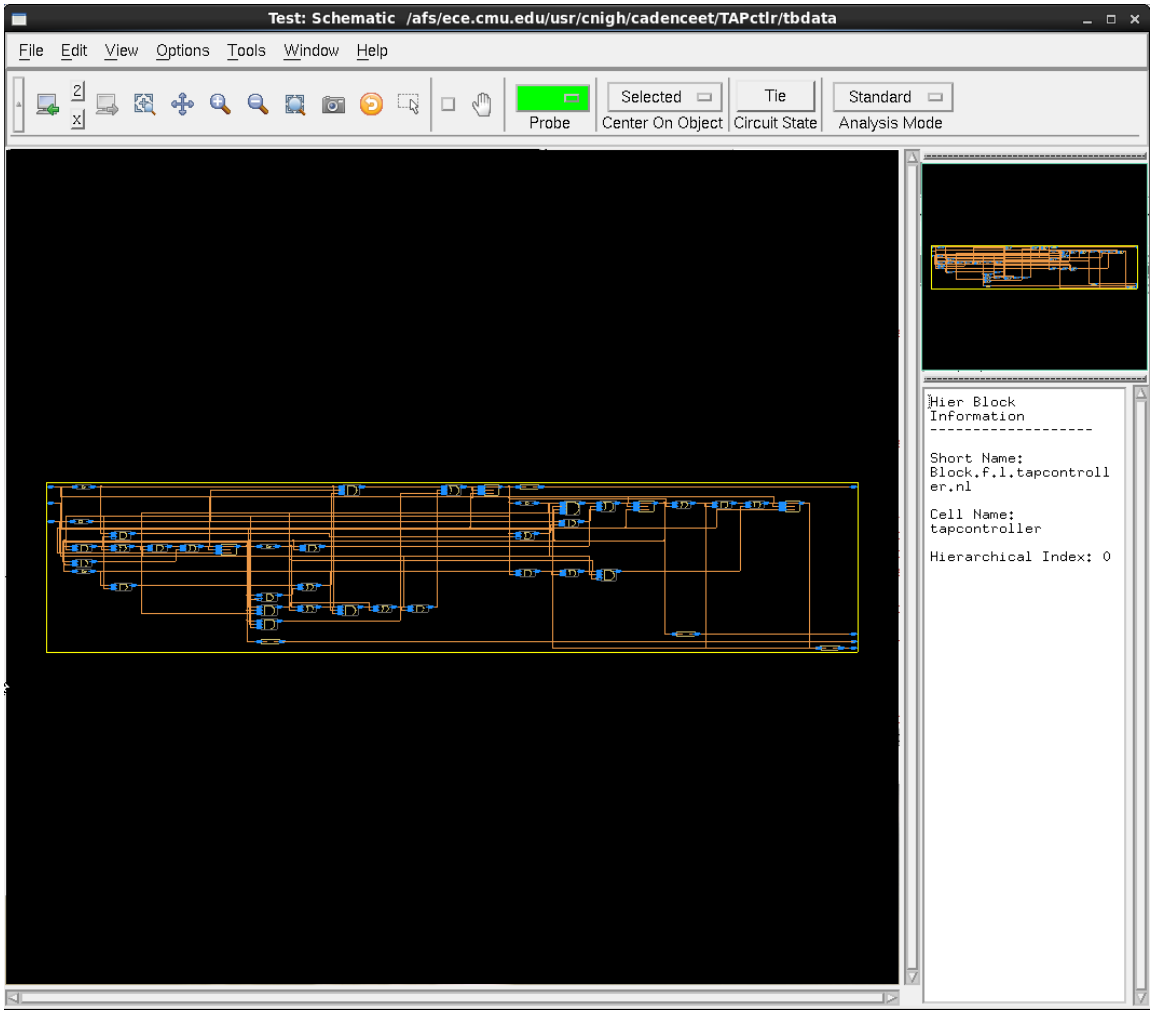
One of the benefits of using a software tool like Encounter Test is the ability to view the schematic of your circuit. It can be helpful for diagnosing problems within your circuit and ensuring that connections are properly made.
In order to view the schematic of the circuit select *Window*, then *View* and *Design Browser*. This will open the *Design Browser*, which can be used to navigate through the different elements of the circuit. An example of this can be shown below.



The different sections of blocks, nets, and pins can be expanded and collapsed by clicking the green box next to the name. In order to view the schematic of any block or pin, right click on the desired element and select *View Selected Object*.



You can *Explode* blocks or pin groups by double clicking, and this can be done to view a representation of the circuit from block-level all the way to gate-level. The schematic viewer is only capable of displaying 1500 objects at once, so it is likely best to view your circuit at a block level instead of a gate level when looking for connections between modules.



Right clicking on an element will also give you many options when exploring your circuit. Combinational logic can be simulated one bit at a time by manually applying values to its inputs, nets can be traced forwards and backwards. Inactive logic will be colored gray, allowing you to easily see when wires are not impacting the resulting circuit.

Lastly, Encounter Test will not allow the user to run tasks while the schematic viewer is open. After you have finished looking at your circuit, close the schematic window and the design browser. Then on the main Encounter Test GUI, click the lock in the upper right-hand corner. When the window with the title *Unlock GUI Resources?* appears, click *Yes* to continue. You will now be able to run any other tasks.

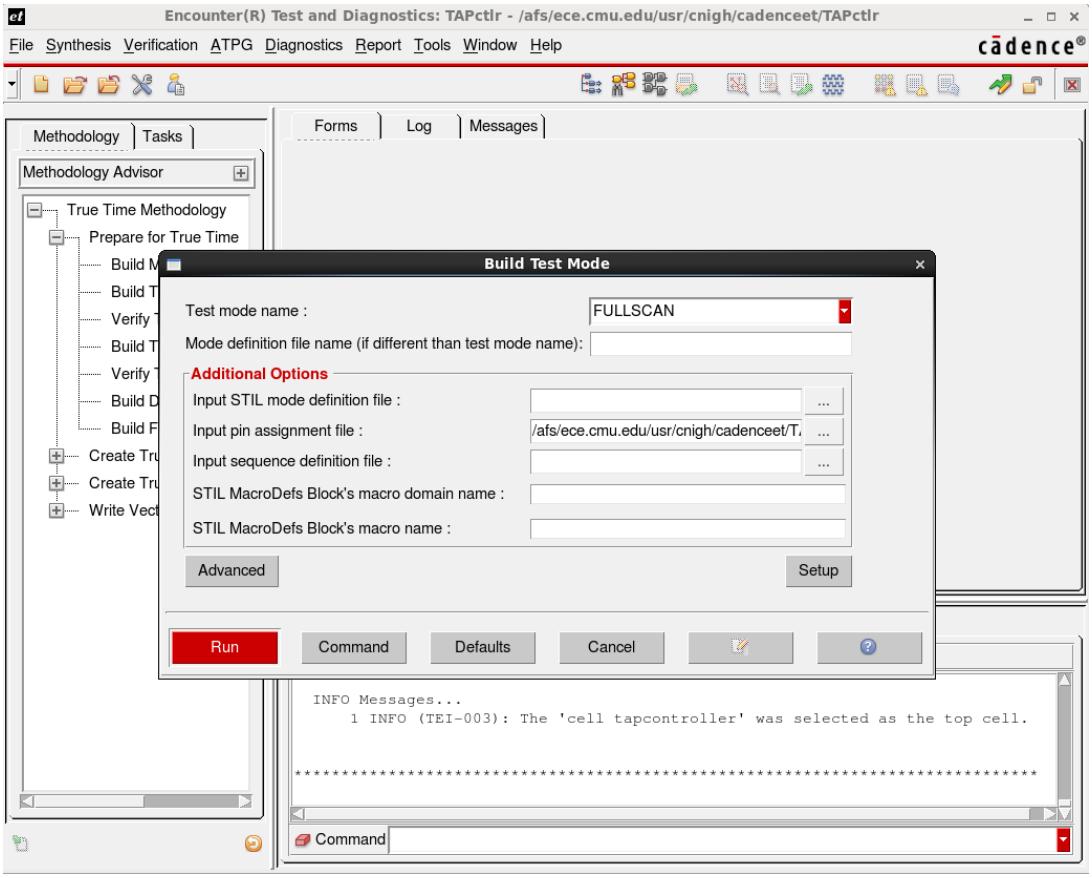
Building the Test Mode

Verification ⇒ Build Models ⇒ Test Mode …

User Inputs: Test Mode Name, Pin Assignment File
Outputs: None

The next step is building the for the circuit. Different circuits might have multiple modes in which they can be tested. Instead of using Encounter Test to create and control these test modes, we use a single test mode and alter the function of the circuit through control bits like TMS or SE.

In order to build the test mode, select *Verification* on the toolbar, then *Build Models* and *Test Mode … .*

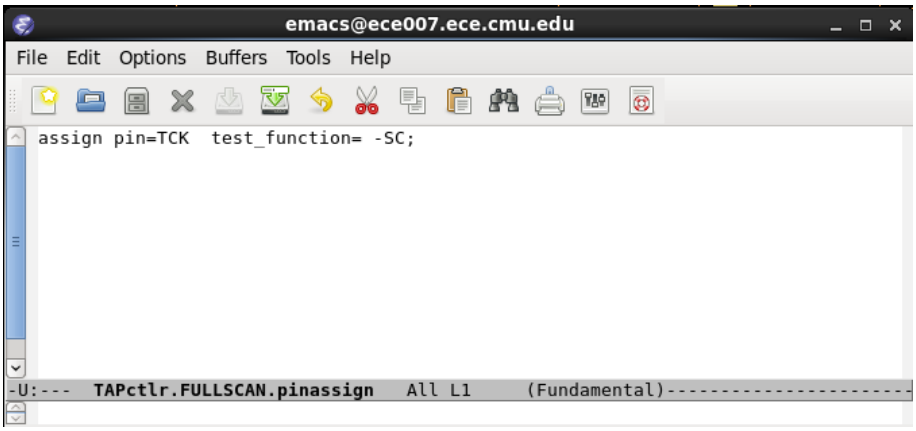


For *Test mode name* type “FULLSCAN”. Building the test mode also requires a pin assignment file in which the clocks are defined as such. This is a very simple file, but correct syntax is essential.

The file should define any clocks in the following form:

```
assign pin=<ckpinname>     test_function=<(+,-)>SC;
```

An example is shown below. The polarity for the test function should correspond to the off-position of the flip-flops in the circuit. For example, if the flip-flops on the circuit update on the rising edge then the clock driving them should have a '-' polarity for the test function.



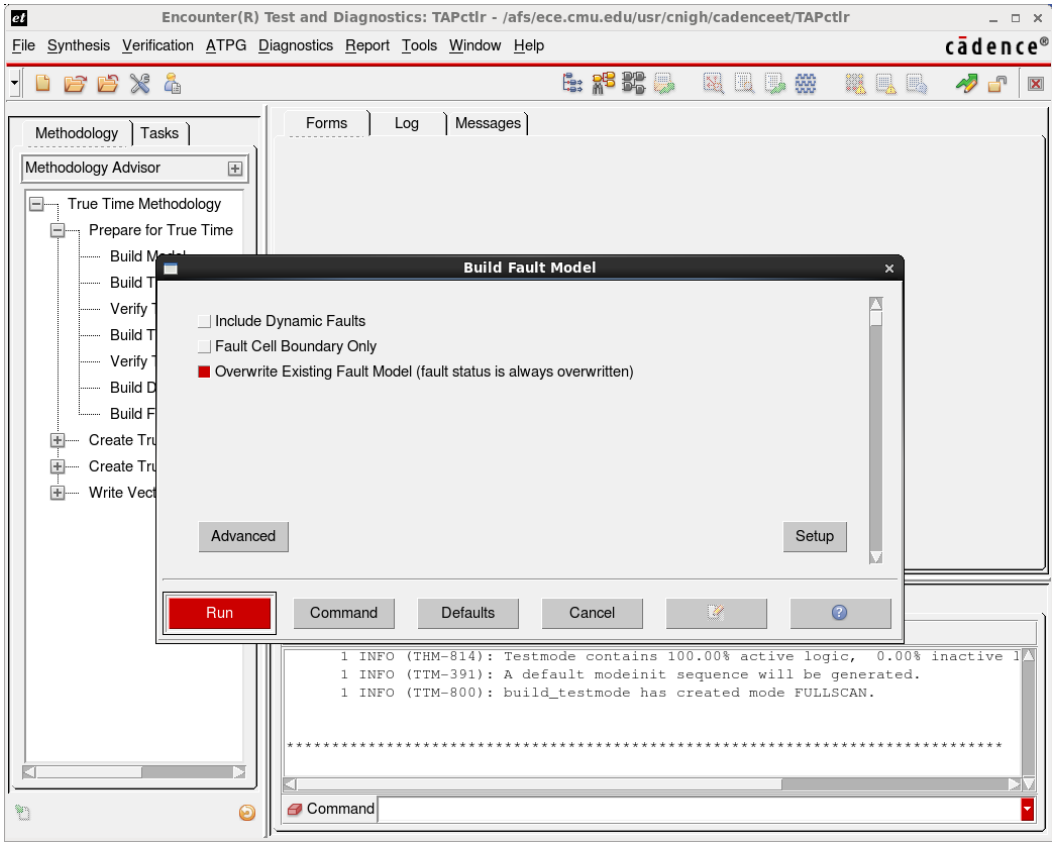
There are other possible test functions and pin definitions that Encounter Test is able to accept. For the purpose of this project, only use the SC (System Clock) test function.

Building the Fault Model

Verification ⇒ Build Models ⇒ Fault Model ...

User Inputs: None
Outputs: None

We now must build the fault model for the circuit. In this step, Encounter Test creates a list of faults which will be used when performing ATPG and simulation. To access it select *Verification* on the toolbar, then *Build Models* and *Fault Model … .* The following window should appear.



Encounter Test can perform analysis on many different fault types and models, but for this project we will only be working with static SSL faults. To avoid the creation of unwanted faults, ensure that *Include Dynamic Faults* and *Fault Cell Boundary Only* are deselected, and that *Overwrite Existing Fault Model* is selected when you wish to update the model. Press *Run* to continue.

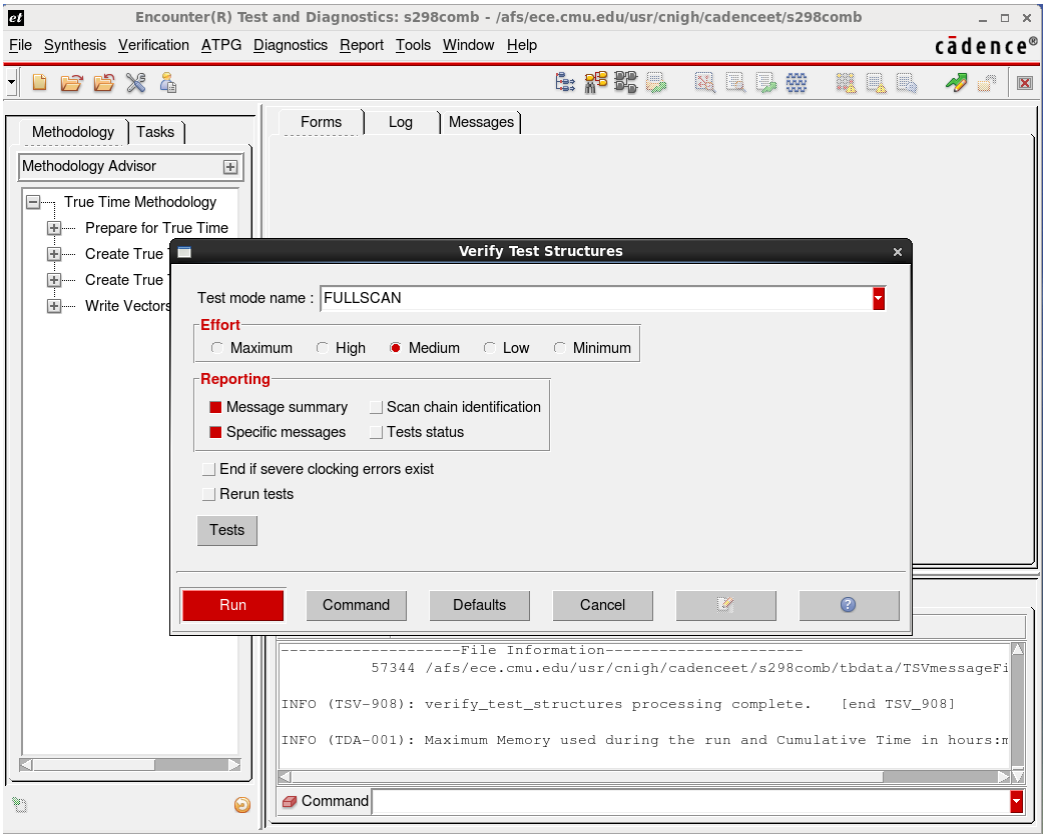
Verifying Test Structures

Verification ⇒ Verify ⇒ Test Structures …

User Inputs: Test mode name
Outputs: None

Verify Test Structures will perform a series of checks on your circuit, returning information and warnings that can help diagnose any problems. This will include checks for things like clock gating, race conditions, and internal scan chains. It is possible to skip this step, but a severe warning will be returned when creating the logic test vectors.

To access this action select *Verification* on the toolbar, then *Verify* and *Test Structures …* . The following window should appear.



Enter “FULLSCAN” for *Test mode name* to match the test mode build earlier. The rest of the options can remain the same. Select *Run* to continue.
As discussed earlier, the *Messages* tab can be very useful when identifying any warnings or errors produced by this task. Refer to the earlier section of the report for specific details on how to use this feature.

Test Pattern Generation and Use

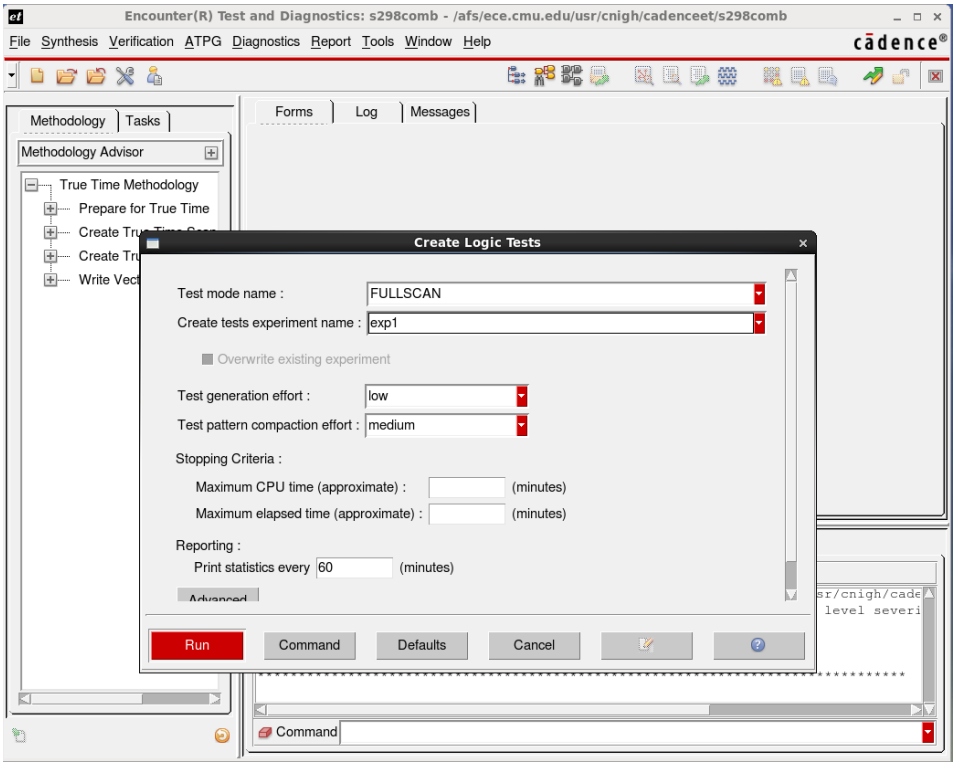
Creating Logic Tests Using ATPG

ATPG ⇒ Create Tests ⇒ Specific Static Tests ⇒ Logic …

User Inputs: Test mode name, Experiment name
Outputs: Experiment with logic test vectors

Creating the logic tests will perform the ATPG for working with a logic circuit. For the purposes of this project, this step will work best on a purely combinational circuit. The circuit that is used for this project is obviously sequential, but it can be altered to obtain valuable testing information from this step.

To access this action select *ATPG* on the toolbar, then *Create Tests*, *Specific Static Tests*, and *Logic*.



Ensure that the “FULLSCAN” test mode used earlier is entered for *Test mode name*. In the Create tests experiment name slot you will name the experiment for which the tests are being created. This experiment name will be used as the reference for the specific test sequence.

The other information can remain the same, but it may be beneficial to change some settings. If you increase the settings for effort in test generation and test pattern compaction Encounter Test may be able to minimize the size of the test set while maintaining fault coverage.

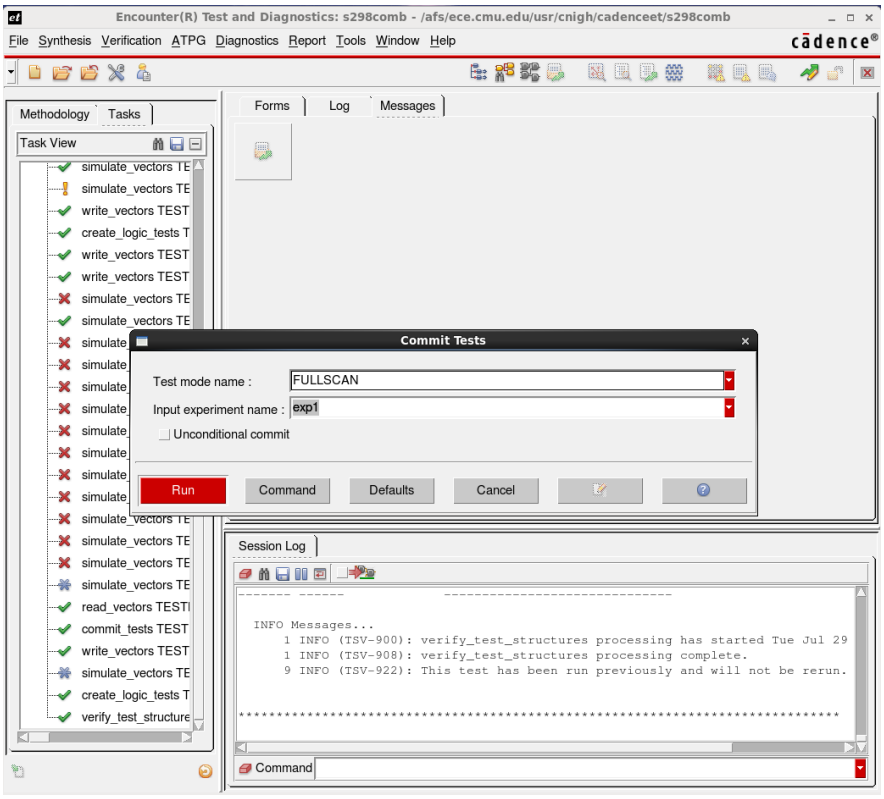
Committing Logic Tests (optional)

ATPG ⇒ Commit Tests …

User Inputs: Test mode name, Experiment name
Outputs: None

Committing tests will apply a set of test vectors to your circuit. It will return information about the fault coverage of the test sequence in the log.

To access this action select *ATPG* on the toolbar and then *Commit Tests …* . The following window should appear.



Ensure that “FULLSCAN” is placed in the *Test mode name*. Select the experiment which contains the test vectors that you wish to apply to the circuit. Press *Run* to continue.

When using manually-created test vectors, this step is not necessary. Custom test sequences can be accepted into Encounter Test and applied to the circuit through simulation, which will be discussed in the section entitled *Simulating a Circuit Using Custom Vectors*.

Writing Logic Test Vectors

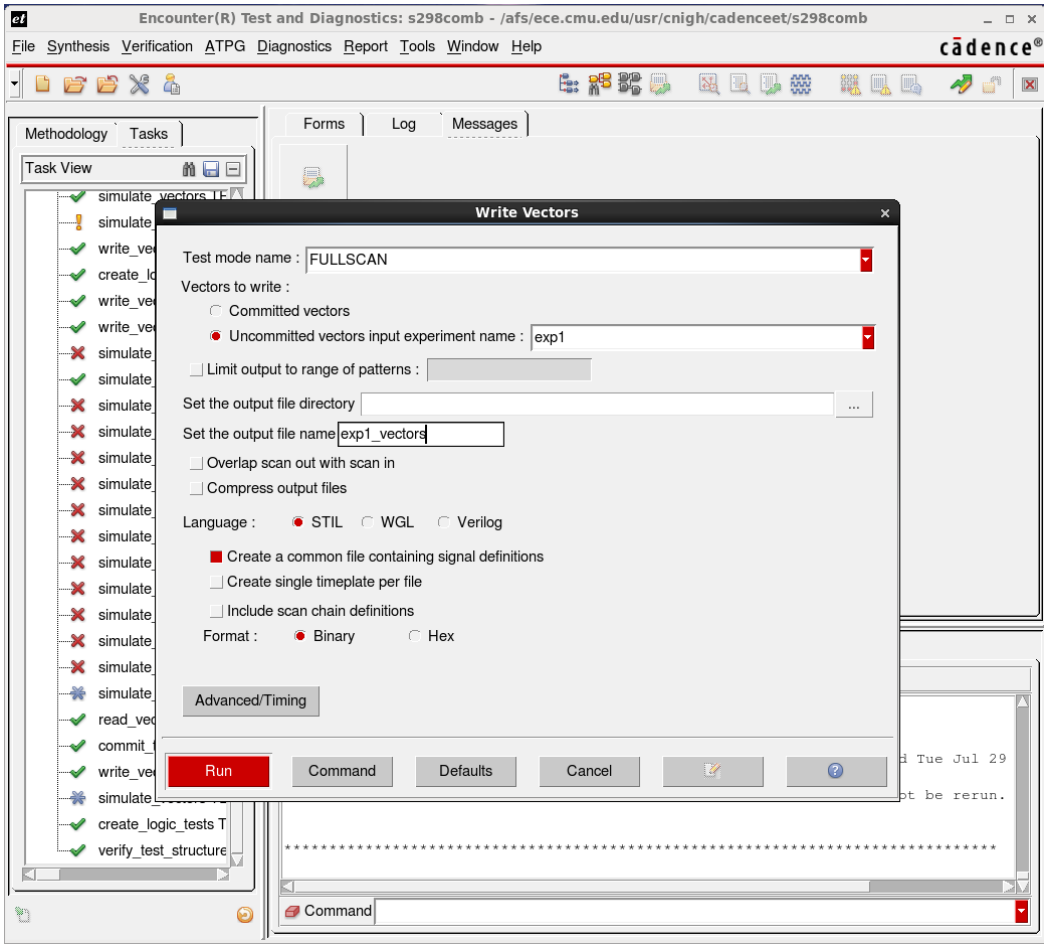
ATPG ⇒ Write Vectors …

User Inputs: Test mode name, Experiment name, Output file name, Output file language

Outputs: *STIL* Vectors file, *STIL* Signals file (with *STIL* as language)

After creating the test vectors, they must be written by Encounter Test in order to manipulate them. This manipulation will be essential to creating custom test vectors intended for a circuit with internal and boundary scan. When ATPG is performed on a combinational circuit, the bits of the resulting test vectors must be rearranged when boundary scan and internal scan is implemented.
Writing the test vectors creates a file which contains the set and sequence of test vectors that was created through ATPG. The scripts that accompany Project 5 are capable of converting and manipulating this output file for easier workability.

In access this action select *ATPG* on the toolbar, and then *Write Vectors …* . The following window should appear.



Ensure that “FULLSCAN” is entered for *Test mode name*. In the *Vectors to write* section, you must select which set of test vectors you would like to write to the file. If you chose to commit the test vectors to the circuit, select *Committed vectors*. This will write only the most recently committed sequence of test vectors. If you wish to write test vectors which have not been committed, select *Uncommitted vectors input experiment name* and select the experiment which has the test vectors you would like to write.

You can then designate the directory where the vectors will be written. If this is left blank, the created files will be placed inside the *testresults* directory of the project’s working directory. You should then define the output file name. In order to avoid confusion, I recommend that the file name somehow corresponds to the experiment that contains the test vectors.
For *Language*, select *STIL*. This is the only format from the list that can be read and created by the Project 5 scripts. Ensure that the *Format* is binary. All other options can remain the same.

There should be two *STIL* files created in the designated directory, one with the extension *signals.stil* and the other with *1.stil*. The latter is the one we are interested in, as it contains the test vectors created by ATPG.

The use of this file with the Project 5 scripts is discussed in further detail in a separate help document.

Simulating a Circuit Using Custom Vectors

Tools ⇒ Vectors ⇒ Simulate

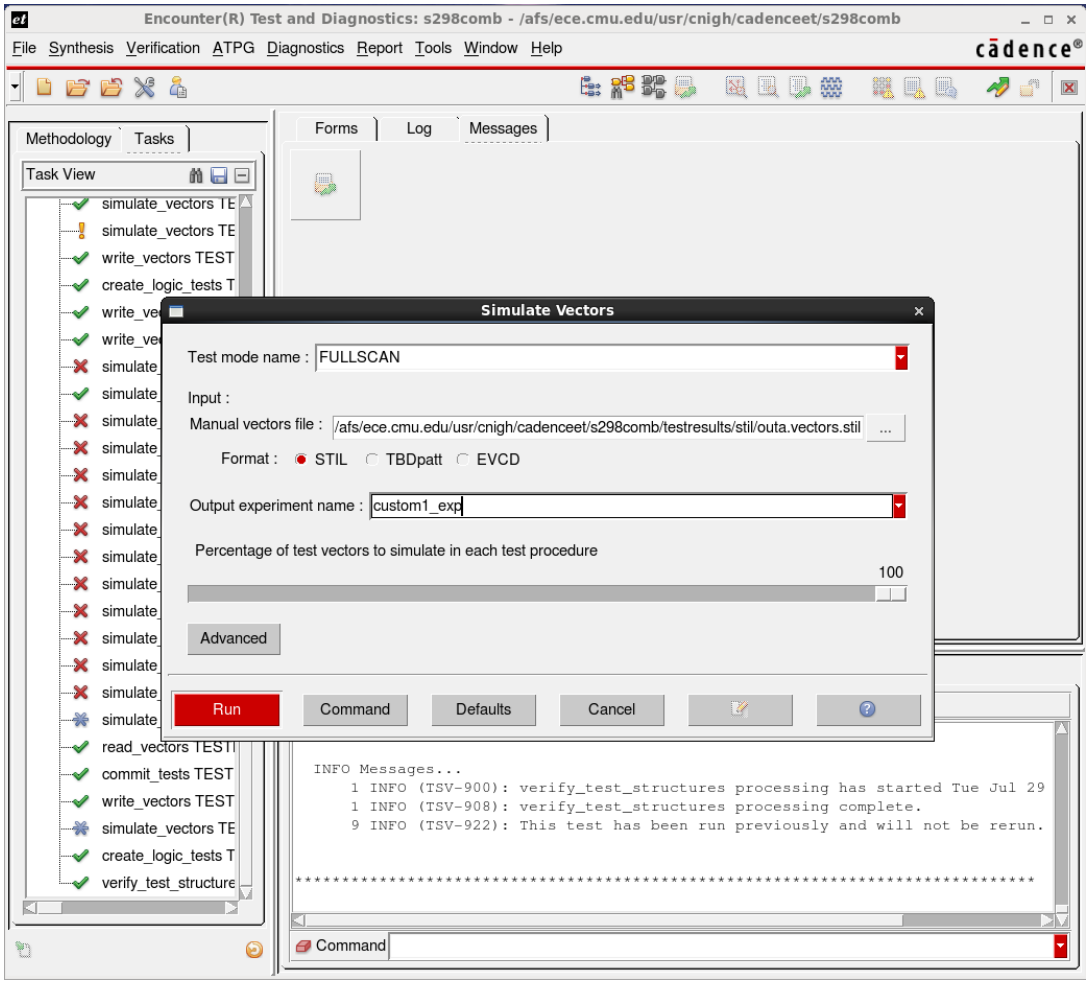
User Inputs: Test mode name, Input vectors file, Input vectors format, Experiment name

Outputs: Experiment with input test vectors

Encounter Test is capable of accepting user-created test vectors for simulation. This is how you will apply test vectors to your updated DFT, as Encounter Test's ATPG will not be able to create tests that account for the intricacies of your design.

To find the simulation action select *Tools* on the toolbar, then *Vectors* and *Simulate*.

You can also read in the test vectors to an experiment, then apply them by using the commit test vectors action described earlier. This should obtain the same result as simulation.



Enter “FULLSCAN” for *Test mode name*. For the *Manual vectors file*, select the *STIL* file that contains the vectors you wish to apply.

The Project 5 script that writes *STIL* files will create both a file of vectors with extension *.vectors.stil* and a file defining signals with extension *.signals.stil*. Both are required for proper operation, but only the vectors file should be selected in this place. It is essential that both are placed in the same directory, as the vectors file contains a `#include` statement pointing to the corresponding signals file. Again, I recommend these files be placed within a subdirectory inside the project’s working directory.

Select *STIL* for the format of the file, as this is the only one of the list which can be written by the Project 5 scripts. For *Output experiment name*, give a name for the experiment which you are creating.

In order to ensure that only static faults are tested, click *Advanced* and select the *Simulation* tab. Under the *Fault Simulation* section, select *Static* as the *Fault type*. Select *OK* and then *Run* to continue and run the simulation.

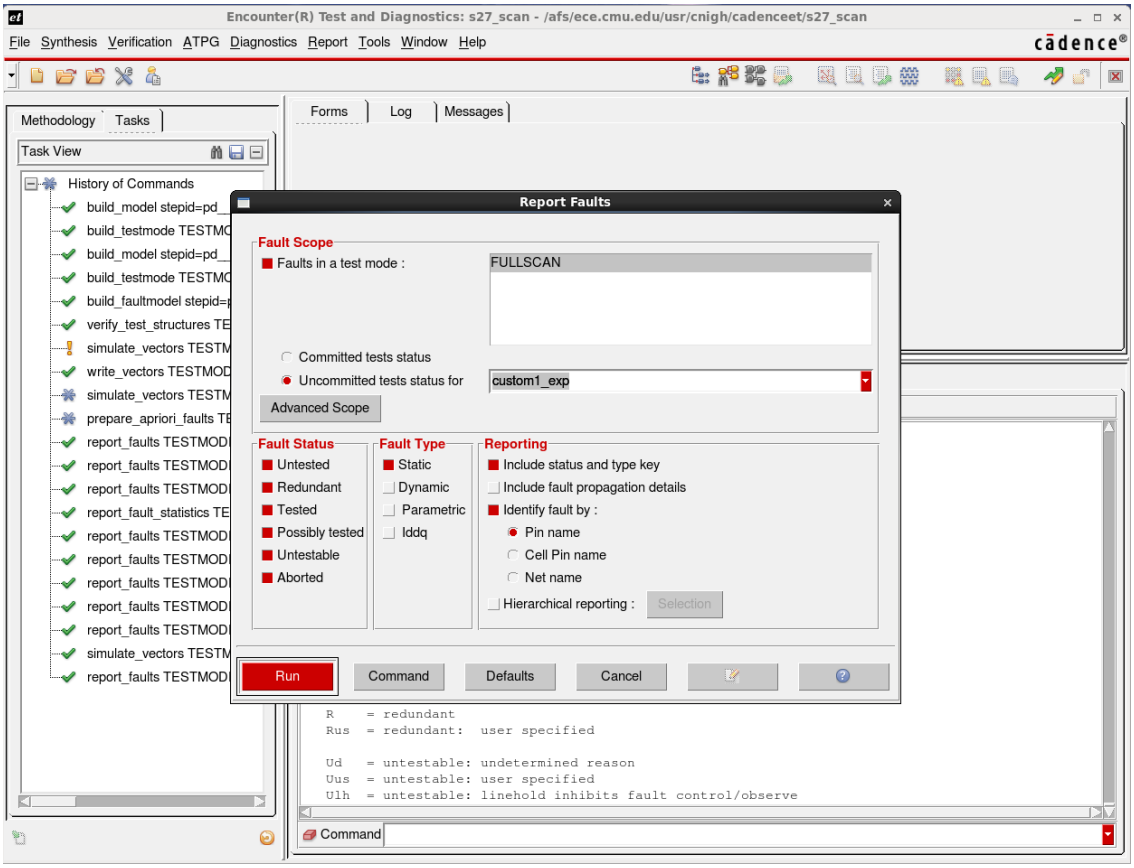
Reporting Fault Detection Information

Report ⇒ Fault ⇒ Logic Faults ...

User Inputs: Test mode name, Experiment name
Outputs: Log file containing fault information

After performing simulation, you may be interested in viewing which faults were tested and which were not. This can allow you to pinpoint faults or regions of faults which are not being considered by the current set of test vectors and implement additional tests to account for them.

To report these faults, select *Report* on the toolbar, then *Fault* and *Logic Faults …* .



Select *FULLSCAN* and the desired experiment. The additional options can be selected in order to print different information about each fault. You may wish to explore these options to find settings which best fit your goal. The set of options selected in the image above demonstrate an example which you may find useful.

Help with Encounter Test

Frequently-Seen Messages

This section contains a list of Warning, Severe Warning, and Error messages that you may see while working with Encounter Test. This is far from all of the problematic messages that you may see, but it will hopefully help you to diagnose the issue based on the message.

Build Model

You may see these messages when performing the *Build Model* step in Encounter Test.

```
ERROR (TEI-002): The 'cell [<cellname> OR noCellNameFound]' was not found
in the design source file
```

Encounter Test had an issue reading your Verilog code. Check the Verilog cell identified in the message and correct any errors in the code.

```
WARNING (TEI-143): All pins on net '<net>' of 'cell <cell>' appear to be
<sinks/sources>

WARNING (TLM-105): Multi-source net '<net>' of 'cell <cell>'

WARNING (TLM-117): TIEX is generated for sourceless net '<net>'

WARNING (TEI-110): Pin <pin> of <cell> has no external net connection for
any usage in the design.
```

There is likely an issue with the Verilog code in your circuit. Use the cell and pin names to look into this and fix any errors that you find before continuing

```
WARNING (TFW-069): [Severe] A write lock could not be obtained for

WARNING (TEI-813): [Severe] Build Model NC (IEEE 2001 Standard) Verilog
Parser:
```

The schematic viewer is either currently open or was recently open. You have not unlocked the GUI resources. Click on the lock of the upper right-hand corner of the main GUI and select *Yes* on the resulting prompt.

Build Testmode

You may see these messages when performing the *Build Testmode* step in Encounter Test.

```
ERROR (TTM-018): Unable to obtain Write lock on testMode.FULLSCAN.

WARNING (TFW-069): [Severe] A write lock could not be obtained for
```

The schematic viewer is either currently open or was recently open. You have not unlocked the GUI resources. Click on the lock of the upper right-hand corner of the main GUI and select *Yes* on the resulting prompt.

```
ERROR (TTM-051): Illegal ASSIGN PIN statement value:
```

```
WARNING (TTM-509): Missing TEST CLOCK test function polarity value in
MODEDEF on 'pin <clock>'
```

There is an issue in the pin assignment file. Check the file and make any necessary corrections before continuing.

```
WARNING (TTM-347): There is less than 96 percent active logic in this
test mode. Global fault
```

Encounter Test is able to identify which logic in the circuit is active and inactive. Ideally, there will be close to 100 percent active logic. If there is a significant amount of inactive logic there may be a problem with your design. Checking the schematic of your design in Encounter Test may be helpful, as inactive logic will be shaded gray.

Verify Test Structures

You may see these messages when performing the *Verify Test Structures* step in Encounter Test.

```
WARNING (TSV-390): There are inactive (non-scan) latches.
```

Encounter Test has found flip-flops that are not connected to active logic. This may point to a problem with your design. Checking the schematic of your design in Encounter Test may be helpful, as inactive logic will be shaded gray.

Simulate Manual Vectors

You may see these messages when performing the *Simulate Vectors* step in Encounter Test.

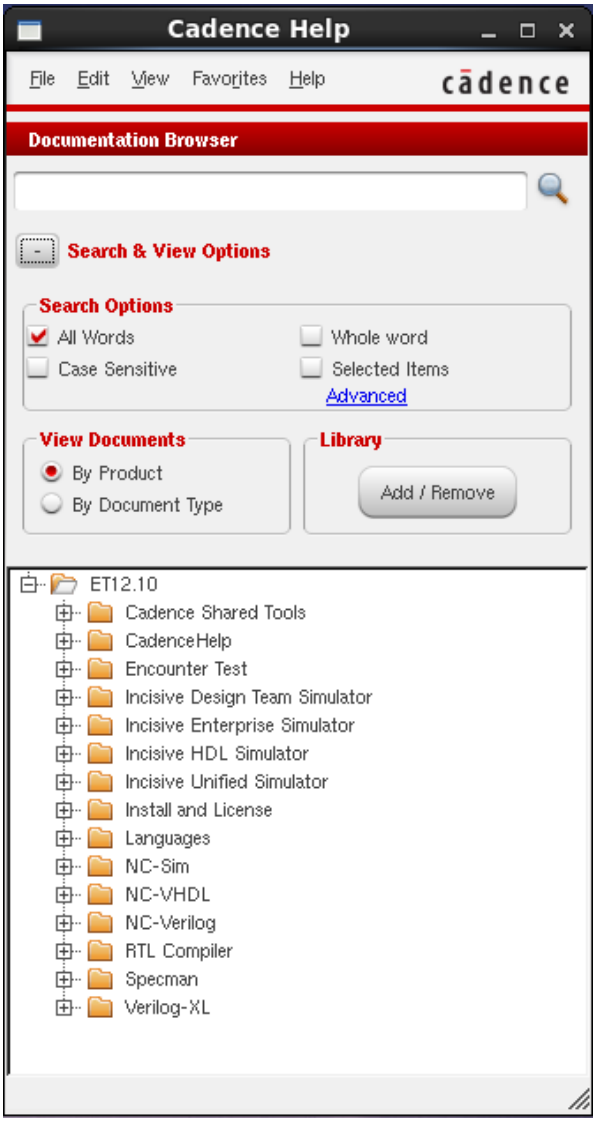
```
WARNING (TCL-710): [Severe] One or more miscompares were detected for the
```

The expected outputs in your *STIL* file do not match the simulation results. There is likely an error with your file of test vectors. Fault coverage will still be calculated, but it will be based on the simulation outputs. In order to view the simulation results and compare them to the expected values, perform a *Write Vectors* using the uncommited experiment defined in simulation.

Getting Help with Encounter Test

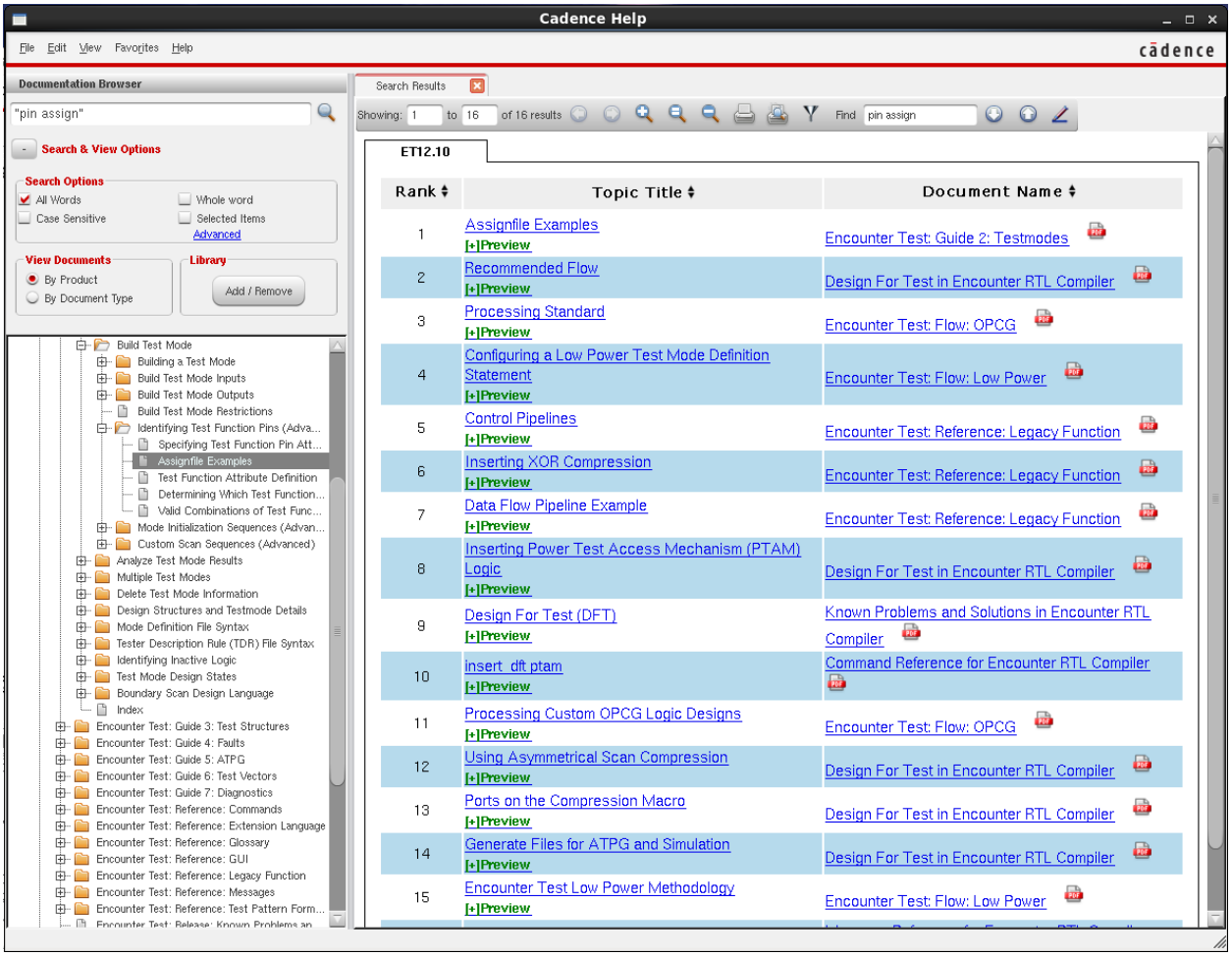
Help ⇒ List of Books …

You may find that you run into errors or problems that are not covered in this document. In this situation, you may find looking through the Cadence manuals to be helpful. In order to access these select *Help* on the toolbar and then *List of Books …* . The following window should appear.



There are a lot of different resources available, but the one that will be most useful is *Encounter Test*. You can navigate through the different folders and documents in this window, but I would recommend searching for the topic in question.

When performing a search containing words that are used in many of the documents, an error may be returned. In this case, you can place quotes around the specific phrase used for search in order to reduce the amount of returned documents.



Although these documents can be helpful in diagnosing issues with your project, the wide functionality of Encounter Test results in a large number of manual documents which do not apply to Project 5. Make sure that you do not use any aspects of Encounter Test that are beyond the scope of this project. The use of functions that were not included in the design of this project will likely result in files or outputs that are not usable for grading.