

Test Case Optimization

Heena Mankad



Test Case Optimization

- **Overview**

Chip design complexity is intensifying as per Moore's law. SOCs are getting more and more complex while integrating many modules. Processor has become the central controller in such complex SOCs. This makes it vital to verify the processor module effectively. Hence, much of the logic goes into framing an effective testcase. An effective testcase can decrease the number of testcases required, automate the simulation and abate lengthy simulations. The proposed approach is very effective for the verification of processor module in any SOC.

- **Traditional Method**

Every complex SOC has a processor module as a central controller. Verifying the processor and its interfaces to various modules is of key importance. This involves exercising the processor to execute all the instructions and the critical data/control paths. In a typical verification, the verilog testcase downloads the coded test code into the processor's memory, configures various global registers, and triggers it. The comparison of the results is optionally done either in the test code itself, or in the verilog testcase. It is required to exercise the processor to execute the test code under different possible configurations. This increases the number of testcases. Moreover, this leads to much of time consumption at the time of board level testing.

- **Contemporary Approach**

This contemporary approach is based on using a systematic way of writing the test code. The test code format shown in figure 1.0, automates the test case execution and eases the debugging later. It shows a systematic approach to verify different functionalities in a single testcase. Moreover, it is possible to have runtime switching between test code and verilog testcase. This enables runtime reconfiguration if needed.

The figure below shows the format of the test code and the verilog test case using a pseudo code.

Important terminologies are described below.

MP : Processor Module

LOAD : Load from memory location

STORE : Store to memory location

MVI : Move immediate

MOV : Move data

CMP : Compare

JNZ : Jump is not zero

JC : Jump if carry

CALL : Subroutine call

R0,R1,R2 : Local Reg

G0 : Global Reg

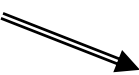
GPIO() : displays arg on GPIO

loop() : polls for a signal, memory location or int. Used for switching simulation control.

Verilog TESTCASE

Test Code

```
reset_mp(1); // reset processor module
GPIO(0xAAAA); //reset code displayed on GPIO
prog_load_mp(); //load MP prog/data memory
GPIO(0xB BBB); //load test code displayed on GPIO
init_mp(); // initializes global regs and mem locations
reset_mp(0); //unreset processor module
GPIO(0x5555); //unreset code displayed on GPIO
loop(); //loops till any event
```



```
MVI R0, 0x1
STORE R0,0x400 //initialize test counter

_test1: //Test No:1
MVI R1, 0x7
ADD R1, G0, R2 //R2=R1 + G0(G0=0x2)
CMP R2,0x9
JNZ _test_reason_1
CMP R1,0x7
JNZ _test_reason_2
JC _test_reason_3
CALL _test_inc

_test2: //Test No:2
-----
-----

_testn: //Test No:n
-----
-----
CALL _test_inc

_pass:
MVI R0, 0x80
STORE R0,0x404 //indicates "test pass"
MVI R0, 0xAA
STORE R0,0x402 //indicates pass
INT //int to switch back to verilog testcase

_test_inc: //increment test counter
LOAD 0x400,R0
INC R0
STORE R0,0x400 //increment test counter
RET

_test_reason_1: //indicates failure due to
```

```

                                wrong result in destn reg

MVI R0,0x81
STORE R0,0x404
JMP _fail

_test_reason_2: //indicates failure due to
                wrong result in source reg

-----
-----

_test_reason_3: //indicates failure due to
                wrong result in flag

-----
-----

_fail:
MVI R0, 0xBB
STORE R0,0x402 //indicates fail
INT //int to switch back to verilog testcase

```

display(); // macro to display final results (test No.,reason of failure, pass/fail) on
 GPIO
 \$finish;




Fig 1.0 Pseudo test code executed by a processor module of SOC

The test code consists of n different tests. They are executed in sequence. A test counter is maintained at memory location 0x400. There are two possibilities after the completion of each test. If the test passes, the test counter increments and the execution continues with the following test. If the test fails, the reason of its failure is stored at memory location 0x404, “fail” status is stored at 0x402 and the simulation control switches back to verilog testcase. If all tests pass, “pass” status is stored at 0x402 and 0x404 before switching back.

Verilog testcase can interact with the test code using specific status indication on any previously decided register, signal or memory location. This can be done in the implementation of macro “loop”. Moreover, each critical simulation event is displayed on GPIO to depict the path followed while simulation. The end status of the code is displayed using the macro “display”. It displays result of testcase in a pre-defined pattern on GPIO based on the contents of memory locations 0x400,0x402 and 0x404. This approach makes debugging easy as the final status on GPIO, points to failing test and the cause of error.

This approach further helps for board level testing. Many such testcases can be combined, when executed on board. Their intermediate status can be continuously displayed on GPIO.3

- **Advantages Of The Testcase Automation Approach**

Prime advantages offered are as under:

- (1) Checking more functionality in a single testcase due to specific test code format and switching simulation control.
- (2) Complete automation for indicating the status of each test at runtime.
- (3) Easy debugging of test using the status sequence followed and the final pattern on the GPIO.
- (4) Combining different testcases for board level testing.

- **Conclusion**

The approach of testcase automation for processor module in any SOC, enables verifying many functionalities in a single testcase, runtime reconfigurations through simulation switching, easy debugging using status patterns and aids executing the tests for board level testing.