esa DOCUMENT

document title/ titre du document

SST Examples of use

prepared by/*préparé par* Daniel González

reference/*réference* TEC-EDM/DGG-SST3
issue/*édition* 1
revision/*révision* 0
date of issue/*date d'édition* 15/11/04
status/*état* Draft
Document type/*type de document* Technical Note
Distribution/*distribution*

European Space Agency
Agence spatiale européenne

APPROVAL

Title *titre*	issue 1 *issue*	revision 0 *revision*

author *auteur*	date 15/11/04 *date*

approved by *approuvé by*	date *date*

CHANGE LOG

reason for change */raison du changement*	issue/*issue*	revision/*revision*	date/*date*

CHANGE RECORD

Issue: 1 Revision: 0

reason for change/*raison du changement*	page(s)/*page(s)*	paragraph(s)/*paragraph(s)*

TABLE OF CONTENTS

1 SCOPE

The objective of this document is to give some examples of the use of the SEUs Simulation Tool. The same netlist, which was obtained from a synthesis of the SpWb1-4 using Synplify for an actel rt54sx72 FPGA with TMRs and a non DDR configuration, is being used in all the cases.

2 TERMS AND ACRONYMS

DDR Double Data Rate
FPGA Field Programmable Gate Array
SEU Single Event Upset
SpWb Spacewire codec from the University of Dundee
SST SEUs Simulation Tool
TMR Triple Modular Redundancy

3 OVERVIEW

We were interested in investigating the time it would take the SST to discover the presence of a failure on a TMR register of a netlist. We also wanted to know the level of complexity of the development of such an experiment, since the Tool had been used only for generating upsets and studying the effects of them in a simulated design but not for discovering if a particular netlist is already damaged.

4 TESTS

4.1 Failure type

The same netlist is being used in both tests described hereafter. One of the flip flops (Q3) of a particular TMR register has been disconnected, so the computation of the voting is altered. The modified piece of code can be seen in figure 3.1.

```
\II_state_0_.q3\: dfc1b port map (
      d => STATE_NS(5),
      clk => SYSCLK_c,
      clr => RST_c_i_34,
      -- DGG -> This register has been modified manually
      q => open);
```

Figure 3.1 – Netlist code modification.

The cm8 voter used from the atmel library has the logical diagram depicted in figure 3.2.

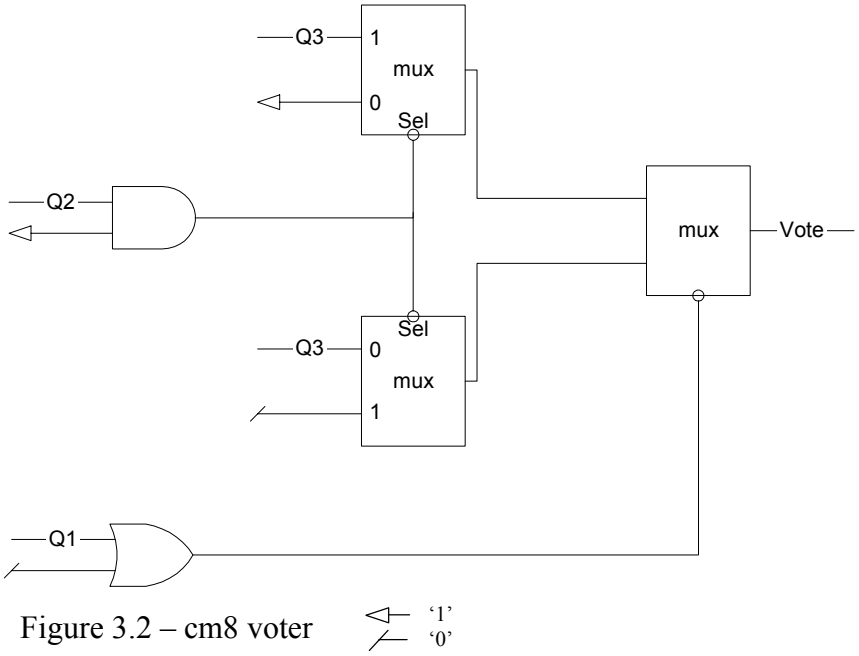

Figure 3.2 – cm8 voter

4.2 Example 1

4.2.1 TEST OVERVIEW

After inspecting the waves and concluding that the registers of a module (init_fsm_1) change their values in a particular time window (180us – 210 us), the SST is used to sequentially upset one flip flop every time the simulation is run, in order to find out which TMR register is the one not working properly. The tcl script *SST_test_loop_ff.tcl*, which can be found in Appendix A, has been written to perform this test.

4.2.2 TEST SETUP

Before running the script used to perform the search of the damaged TMR register, we need to have a initial all_instances.dat file with the selection of those flip flops (instantiated on the netlist) that are going to be inspected. To do so we use the filter instances option of the SST with the following pattern: 'init_fsm_1'. The result is that 340 instances are selected.

We could be more precise in the filtering by using the pattern: 'init_fsm_1.*q', which would select all the instances under init_fsm_1 which names contain the letter 'q' (most likely to be a register). As a result of that filtering scheme 140 instances are selected.

4.2.3 TCL SCRIPT DESCRIPTION

The test script consists on the following four steps:

- Check if all_instances.dat has at least one instance available to be selected ('Yes' in the force column). If that is the case, make a copy of this file removing all the available instances but the first one encountered, and remove this instance from the original file. The copy would be the file used by the SST hence it would only have one register available to be upset. In the next iteration of the script, the register that has already been upset won't be available at all_instances.dat.
- Run the SST using the following command:
  '*exec perl -S SST_upset_generator.pl -iread -nfilter 3 q -t 180us 30us*'
  It will generate a sst.do file which, when executed, will upset the output (q) of the selected register at a time value between 180 and 210 us.
- Execute the sst.do macro generated by SST. It will run the simulation introducing an upset.
- Check if the simulation finished without errors, in which case the script follows the above-mentioned steps again. If not, the script halts and an error message is presented on screen.

4.2.4 TIMING

Using the Modelsim *simstats* command we can determine the duration of a simulation. In our example a single run of the test bench takes 53.422 sec.

The mean duration of this test would be (assuming $N_{of\_FF} = 140$):

$$\frac{N_{of\_FF} \times T_{simul}}{2} = 3739.54 \text{ sec}$$

4.3 Example 2

4.3.1 TEST OVERVIEW

Although knowing that a particular TMR register is broken, it is sometimes difficult to locate when in our simulation a change in its value could be monitored at the output pins of the circuit. Some upsets do not produce an observable change in the behaviour of a circuit at certain times. The SST is used to upset the same register every time the simulation is run, but at different time values, in order to locate the time window in which an upset in one of the working registers of the damaged TMR can be observed. The tcl script *SST_test_loop_time.tcl*, which can be found in Appendix A, has been written to perform this test.

4.3.2 TEST SETUP

Before running the script used to perform the search of the time window in which the TMR register failure could be detected, we need:

- An initial all_instances.dat file. As we are assuming that the damaged TMR register is known, we should be using an all_instances.dat file in which the two registers of the TMR

that are not damaged (Q1 and Q2) are selected, so an upset on any of them produces a wrong vote (that could be observed or not at the output of the circuit).
We could use the filter instances option of the SST (filter pattern: '*init_fsm_1.*\\II_state_0_.q[12]\\$*') or simply editing a 'Yes' in either one or both of the lines related to the mentioned registers which are the following:

```
669 Yes  -II_state_0_.q2-                1        /tb_top/spwrlinkwrap_verif_1/spwrlink_1/ii_init_fsm_1/ii_state_h/\II_state_0_.q2\
671 Yes  -II_state_0_.q1-                1        /tb_top/spwrlinkwrap_verif_1/spwrlink_1/ii_init_fsm_1/ii_state_h/\II_state_0_.q1\
```

- Initial values for some of the variables used in the script (which will be described in the following section).

4.3.3 TCL SCRIPT DESCRIPTION

Before running the script we need to edit the following variables depending on the time window we want to test:

- window_width_value: this variable holds the window width to be used in each SST run.
- first_time_value: this variable holds the starting time for introducing upsets.
- last_time_value: No more upsets will be generated after the time value held by this variable.

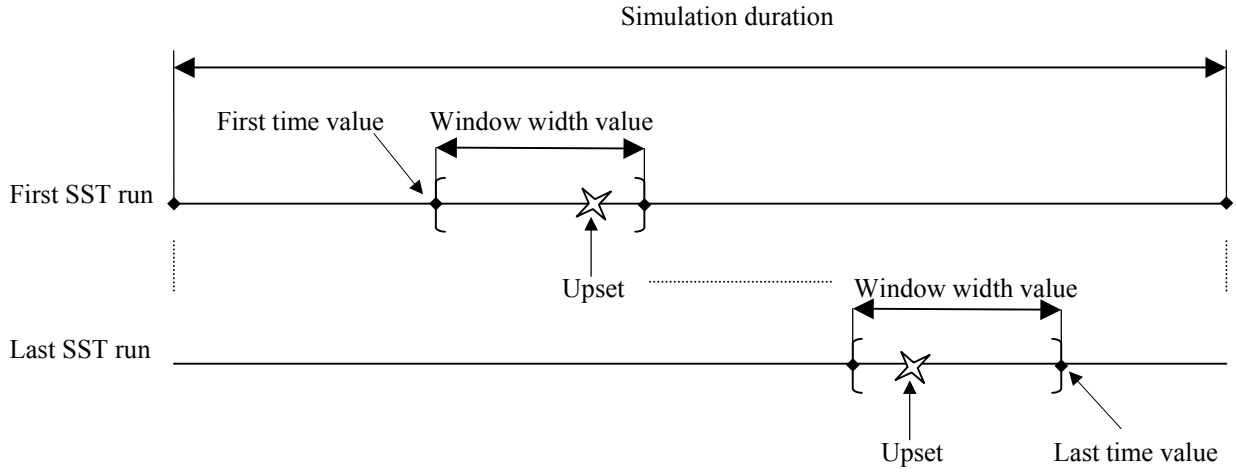

Figure 3.3 - SST_test_loop_time.tcl variables

The script consists on the following four steps:

- Check if all_instances.dat has at least one instance available to be selected ('Yes' in the force column).
- Run the SST using the following command:
  exec perl -S SST_upset_generator.pl -iread -nfilter 1 q -t $start_time_value$unit $window_width_value$unit
  It will generate a sst.do file which, when executed, will upset the output (q) of the selected register at a time value randomly chosen inside the correspondent window width interval.
- Execute the sst.do macro generated by SST. It will run the simulation introducing an upset.

- Check if the simulation finished without errors, in which case the script follows the above-mentioned steps again. If not, the script halts and an error message is presented on screen.

4.3.4 TIMING

Using the Modelsim *simstats* command we can determine the duration of a simulation. In our example a single run of the test bench takes 53.422 sec.

The mean duration of this test would be (assuming last_time_value = 220; first_time_value =20; window_width = 20; unit = us):

$$\left(\frac{last\_time\_value - first\_time\_value}{window\_width} \right) \times T_{simul} = 534.22 \text{ sec}$$

5 APPENDIX A

5.1 SST_test_loop_ff.tcl source code

```
#!/usr/local/bin/perl -w
#================================================================================================
# Design unit  : SEUs simulation Tool (SST)
#
# File name    : SST_test_loop_ff.tcl
#
# Purpose      : This script is used to upset several FF in a serial manner (one per
#                run) in order to locate a damaged TMR in a particular netlist
#                (spwrlink_tmr).
#
# Authors      : Daniel González Gutiérrez TEC-EDM
#
# Contact      : mailto:microelectronics@estec.esa.int
#                http://www.estec.esa.int/microelectronics
#
# Copyright (C): European Space Agency (ESA) 2004. No part may be reproduced
#                in any form without the prior written permission of ESA.
#
# Disclaimer   : All information is provided "as is", there is no warranty that
#                the information is correct or suitable for any purpose,
#                neither implicit nor explicit. This information does not
#                necessarily reflect the policy of the European Space Agency.
#================================================================================================
#================================================================================================
# Revision control
#
###Version 1.0 On going... --DGG
#================================================================================================
################################################################################################
# Before running this script we need to:
# * Load the design and run SST_startup.tcl
# * Have an starting all_instances.dat file that we want to consume
################################################################################################

#-------------------------------------------------------------------------
# My variables
#-------------------------------------------------------------------------
set consumed 0
```

```
set simulation_end 0

#--------------------------------------------------------------------------------
# SST directory and file names
#--------------------------------------------------------------------------------

set wire_f_dir {SST/wire_files}
set control_f_dir {SST/control_files}

set all_instances_dat [file join $control_f_dir all_instances.dat]
set all_instances_bak [file join $control_f_dir all_instances.bak]
set all_instances_tmp [file join $control_f_dir all_instances.tmp]

set sst_do [file join "../Spacewire/uod/spwb1-4/codec/src/syn/sim/" $control_f_dir sst.do]

#--------------------------------------------------------------------------------
# Script main body
#--------------------------------------------------------------------------------

while {!$simulation_end} {

  # we need two temporary files, one holding only the instance selected in this
  # particular run (AI_TMP) and the one holding all the instances but the one
  # selected (AI_CNS)

  set AI [open $all_instances_dat r]
  set AI_TMP [open $all_instances_tmp w]
  set AI_BAK [open $all_instances_bak w]

  while { [gets $AI ai_line] != -1} {
    if { ([regexp -nocase {^ *Y} $ai_line]) && ($consumed == 0)} {
      set consumed 1
      puts $AI_TMP $ai_line
      regsub -nocase {Yes} $ai_line {No} ai_bak_line
      puts $AI_BAK $ai_bak_line
    } elseif { ([regexp -nocase {^ *y} $ai_line]) && ($consumed == 1) } {
      puts $AI_BAK $ai_line
      regsub -nocase {Yes} $ai_line {No} ai_tmp_line
      puts $AI_TMP $ai_tmp_line
    } else {
      puts $AI_TMP $ai_line
      puts $AI_BAK $ai_line
    }
  }
  close $AI
  close $AI_TMP
  close $AI_BAK
  file rename -force $all_instances_tmp $all_instances_dat


  # If at least there is one instance left, execute the script and
  # run a simulation
  if {$consumed} {
    # Execute SST_upset_generator to prepare the environment to upset only
    # the FF we are interested in
    exec perl -S SST_upset_generator.pl -iread -nfilter 3 q -t 180us 30us

    # run the modelsim macro created by that script
    #vsim -restore test_loop.cp -do $sst_do
    do $sst_do

    #Check status at the end of the simulation
    view signals
    env /tb_top/uod_ctrl_1
    set tb_ok_index [.signals.tree find tb_receive_success]
    set tb_ok [.signals.tree get2 $tb_ok_index]
    set uut_ok_index [.signals.tree find uut_receive_success]
    set uut_ok [.signals.tree get2 $uut_ok_index]
    set keep_running_tb [regexp {true} $tb_ok]
```

```
    set keep_running_uut [regexp {true} $uut_ok]

    if { ($keep_running_tb) && ($keep_running_uut) } {
      destroy .source
      restart -f
    } else {
      puts stdout "---Errors found!!\n\n"
      set simulation_end 1
    }
  } else {
    set simulation_end 1
  }

  file rename -force $all_instances_bak $all_instances_dat
  set consumed 0
}

destroy .signals
```

5.2 SST_test_loop_time.tcl source code

```
#!/usr/local/bin/perl -w
#================================================================================
# Design unit  : SEUs simulation Tool (SST)
#
# File name    : SST_test_loop_time.tcl
#
# Purpose      : This script is used to upset one FF at different time windows.
#
# Authors      : Daniel González Gutiérrez TEC-EDM
#
# Contact      : mailto:microelectronics@estec.esa.int
#                http://www.estec.esa.int/microelectronics
#
# Copyright (C): European Space Agency (ESA) 2004. No part may be reproduced
#                in any form without the prior written permission of ESA.
#
# Disclaimer   : All information is provided "as is", there is no warranty that
#                the information is correct or suitable for any purpose,
#                neither implicit nor explicit. This information does not
#                necessarily reflect the policy of the European Space Agency.
#================================================================================
#================================================================================
# Revision control
#
###Version 1.0 On going... --DGG
#================================================================================
################################################################################
# Before running this script we need to:
# * Load the design and run SST_startup.tcl
# * Have an starting all_instances.dat file that we want to consume
################################################################################

#--------------------------------------------------------------------------
# My variables
#--------------------------------------------------------------------------
set ok_to_upset 0
set simulation_end 0
set window_width_value 20
# Note that we use the same unit everywhere
set unit "us"
set first_time_value 160
set last_time_value 220

#--------------------------------------------------------------------------
# SST directory and file names
#--------------------------------------------------------------------------
```

```
set wire_f_dir {SST/wire_files}
set control_f_dir {SST/control_files}

set all_instances_dat [file join $control_f_dir all_instances.dat]
set sst_do [file join "../Spacewire/uod/spwb1-4/codec/src/syn/sim/" $control_f_dir sst.do]

#--------------------------------------------------------------------------
# Script main body
#--------------------------------------------------------------------------

set start_time_value $first_time_value

while {!$simulation_end} {

  set AI [open $all_instances_dat r]
  while { [gets $AI ai_line] != -1} {
    if { ([regexp -nocase {^ *Y} $ai_line]) } {
      set ok_to_upset 1
    }
  }
  close $AI

  # If at least there is one instance to be upset, execute the script and
  # run a simulation

  if {$ok_to_upset} {
    # Execute SST_upset_generator to prepare the environment to upset only
    # the FF we are interested in

    exec perl -S SST_upset_generator.pl -iread -nfilter 1 q -t $start_time_value$unit
$window_width_value$unit

    # run the modelsim macro created by that script
    do $sst_do

    #Check status at the end of the simulation
    view signals
    env /tb_top/uod_ctrl_1
    set tb_ok_index [.signals.tree find tb_receive_success]
    set tb_ok [.signals.tree get2 $tb_ok_index]
    set uut_ok_index [.signals.tree find uut_receive_success]
    set uut_ok [.signals.tree get2 $uut_ok_index]
    set keep_running_tb [regexp {true} $tb_ok]
    set keep_running_uut [regexp {true} $uut_ok]

    if { ($keep_running_tb) && ($keep_running_uut) } {
      destroy .source
      restart -f
    } else {
      puts stdout "---Errors found!!\n\n"
      set simulation_end 1
    }
  } else {
    set simulation_end 1
  }

  # Update time values

  set start_time_value [expr ($start_time_value + $window_width_value)]
  if {$start_time_value >= $last_time_value} {
    set simulation_end 1
  }
  set ok_to_upset 0
}

destroy .signals
```