SANDIA REPORT

SAND2018-14109
Unlimited Release
Printed September 2018

Digital/Analog Cosimulation using CocoTB and Xyce

Andrew M. Smith, Jackson R. Mayo, Rob Armstrong, Richard Schiek, Pete Sholander, and Ting Mei

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology and Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2018-14109
Unlimited Release
Printed September 2018

Digital/Analog Cosimulation using CocoTB and Xyce

Andrew M. Smith and Robert C. Armstrong
Assured Digital Systems & Computation
Sandia National Laboratories
Livermore, CA 94551-0969

Jackson R. Mayo
Scalable Modeling & Analysis Systems
Sandia National Laboratories
Livermore, CA 94551-0969

Richard Schiek, Pete Sholander, and Ting Mei
Electrical Models and Simulation
Sandia National Laboratories
Albuquerque, NM 87185-1177

Abstract

In this article, we describe a prototype cosimulation framework using Xyce, GHDL and CocoTB that can be used to analyze digital hardware designs in out-of-nominal environments. We demonstrate current software methods and inspire future work via analysis of an open-source encryption core design. Note that this article is meant as a proof-of-concept to motivate integration of general cosimulation techniques with Xyce, an open-source circuit simulator.

Contents

Figures

1 Introduction

The design of digital systems often begins at a much higher level than the circuit topology and differential algebraic equations (DAEs) with which Xyce or other analog simulators are concerned. Typically, the implementation of digital hardware begins in digital programming languages, like VHDL and Verilog, and is tested using digital simulators, like Mentor Graphics's ModelSim. Properties about the digital behavior can also be proven using assertion languages such as System Verilog Assertions (SVA) and model checkers like Cadence's Jasper or OneSpin. For high level behaviors, these digital tools are often adequate assuming a *nominal environment*. However, if we wish to assert that certain design properties hold in *out-of-nominal* or adverse environments, we must also consider electrical upsets, which digital simulators are unable to model. On the other hand, full-system electrical simulations are likely computationally intractable, especially if a wide variety of upsets needs to be analyzed.

We present a prototype software framework for efficiently analyzing digital systems in the presence of out-of-nominal environments. This framework allows for the analysis of upsets in purely digital systems or coupled digital/analog systems. Our approach uses standard digital design simulators while the design under test (DUT) is in nominal conditions, and transitions to analog simulation only for the time and location of the upset of interest (UOI). Several software challenges arise in defining this framework; in this paper, we describe how we address these challenges using a simple example. Section 2 outlines the example target design used throughout this article, Section 3 describes how to set up the tools and project directories for running analog/digital cosimulations, and Section 4 describes current and future analysis methods of cosimulation results.

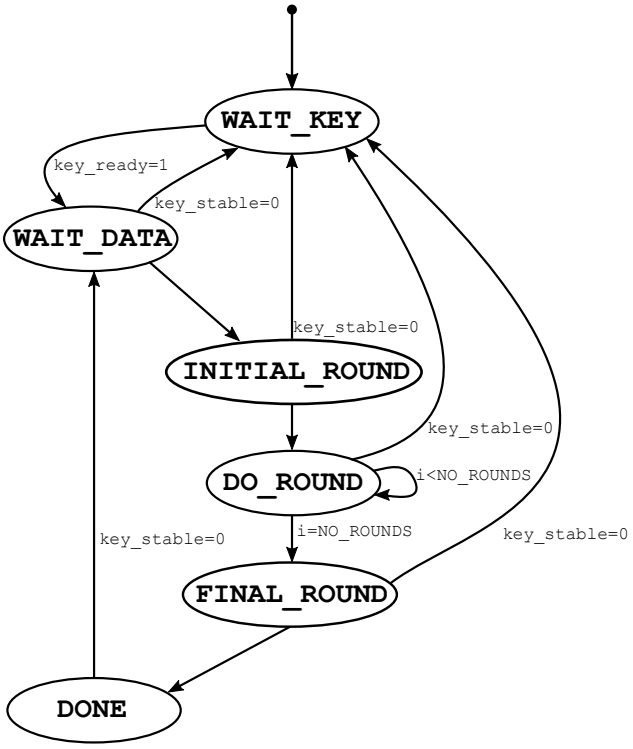

Figure 1: Flowchart diagram of the controlling logic within the tiny_aes AES encryption core design.

2 An example problem: AES encryption core finite state machine

To illustrate our software framework and analysis, we use a portion of the tiny_aes Advanced Encryption Standard (AES) core design. VHDL code for this project is available on OpenCores [3]. More specifically, we are concerned with the finite state machine in the AES design that serves as the logic controller for the encryption algorithm. Figure 1 shows the high-level functionality of this design.

2.1 Preprocessing the encryption core

The tiny_aes encryption core is designed in behavioral VHDL, a digital modeling language with high-level programming constructs, motivated by, but not strictly tied to, its physical implementation. At this level, testing can be done through simulation, universal verification methodology (UVM) testing, or more rigorous formal verification through the use of tools like Jasper or OneSpin. Unless specified explicitly in the design, such tools can describe only nominal digital behavior. We wish to understand how designs behave in adverse environments, such as under the influence of a single-event upset. Consequences of such behavior can be very loosely approximated by modeling one-off bit-flips in digital logic. What is

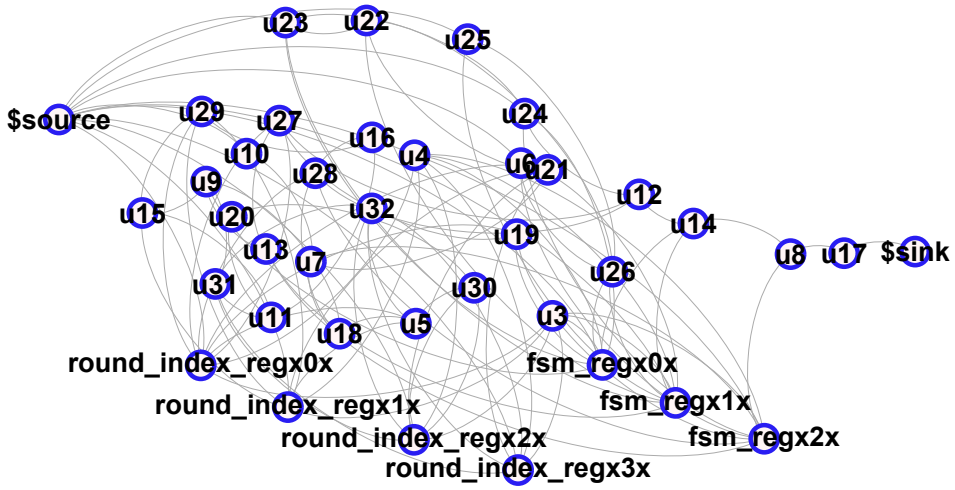

Figure 2: Synthesized netlist of the tiny_AES encryption core controller logic using Sandia's CMOS7 target technology. Each node has a digital and analog model associated with it. "$source" and "$sink" nodes represent inputs and outputs to the system respectively. "fsm_regx2x" (bottom right) represents the location of our upset in Section 4.

missing, however, is the ability to characterize the upset behavior based on the physical characteristics of the UOI. For this level of fidelity, we require an analog description of what is happening during an electrical upset. To achieve an analog representation of our digital model, we must first synthesize the behavioral VHDL into a netlist comprised of gates from a specified technology. The target technology we use for this example is Sandia's CMOS7 library, used for many ASICs at Sandia. The CMOS7 library also includes corresponding analog circuit descriptions for each digital gate (which is common for most target technologies). The digital netlist resulting from the synthesis process (shown in Figure 2) can also be used with the digital analysis tools described above, as well as with network-based analysis methods [6]. We will use both the synthesized netlist and the target technology analog circuit files to analyze the AES design in our cosimulation framework.

3 GHDL/Xyce Cosimulation Framework

We begin describing our cosimulation framework by introducing the external tools used. For digital simulation, we use the open-source GHDL simulator [2]. We use this simulator because it is free, and thus easily transferable to other analysts, but other popular simulators such as ModelSim will also work within this framework. For analog simulation, we use the Sandia developed Xyce simulator. For cosimulation, we must also have interfaces to each simulator. This is not trivial, and requires tool support since starting and stopping simulations whenever values are requested can be costly and lead to incorrect results if not careful about synchronization. For interfacing with GHDL, we use CocoTB an open-source automated cosimulation and testbench library [1]. CocoTB is typically used for generating and running VHDL or Verilog testbenches in Python, and supports a wide range of popular simulators, including GHDL. For the analog interface, we will use the recently developed Xyce/Python libraries; as of release 6.10, these libraries ship with Xyce. We will use Python scripts to manage and synchronize the cosimulation of Xyce and GHDL, since their corresponding interfaces have Python libraries.

3.1 Setting up the environment

In this section, we walk through how to correctly set up an environment for using our digital/analog co-simulation framework. This will assume an account on one of Sandia's HPC machines; if this is not available, independent installations of Xyce and GCC will be required.

Installing Python

Python version 2.7.12 is the only tested version for this cosimulator; it is expected that later versions will work as well. For a stable install, run the following shell commands:

```
# On HPC clusters, ensure you're using GCC v. 6.0.1
module load tce
module load gcc/6.1.0

# Install python locally (so far this only works for 2.7.12):
mkdir ~/python
cd ~/python
wget http://www.python.org/ftp/python/2.7.12/Python-2.7.12.tgz
tar zxfv Python-2.7.12.tgz
find ~/python -type d | xargs chmod 0755
cd Python-2.7.12
./configure --enable-shared --prefix=\$HOME/python
```

```
make
make install
```

Installing GHDL and CocoTB

To checkout and install the latest versions of GHDL and CocoTB, run the following commands:

```
# Install ghdl and cocotb
git clone https://github.com/potentialventures/cocotb
git clone https://github.com/tgingold/ghdl
cd ghdl
./configure --prefix=/usr/local
make
make install
```

Setting the Xyce/Python environment

Xyce 6.9 is currently deployed on Sandia's SRN capacity machines with the Xyce/Python interface libraries (these are also included in independent installations). To load Xyce 6.9 with radiation models (necessary for certain upsets) on the SRN capacity machines, enter:

```
module load xyce/XyceRad/serial
```

For setting up the Xyce/Python interface, Sholander, et al. [4] has more information on how to run the Mixed Signal Interface with Xyce 6.9/6.10 on Sandia High-Performance Computing (HPC) platforms, including the location of the necessary libraries and some Python-based examples.

Finalizing the cosimulation environment

Either run or place these environment variable declarations in your $HOME/.bashrc file. These will ensure that Python and associated libraries are on your path.

```
#Edit .bashrc for environment variables.
export PATH=$HOME/python/Python-2.7.12:$HOME/python/bin
export PYTHONPATH=$HOME/python/Python-2.7.12
export PYTHON_HOME=$HOME/python/Python-2.7.12
export LD_LIBRARY_PATH=$HOME/python/Python-2.7.12:$LD_LIBRARY_PATH
```

Finally, it may be required to edit one of CocoTB's makefiles to ensure that the installed Python libraries are included when simulation projects are built. Go to the 'makefiles' directory where CocoTB was installed (if the preceding steps were done in your home directory, it will be in $HOME/cocotb/makefiles). Using your favorite text editor, open 'Makefile.inc' and navigate to the line that begins with 'export INCLUDES := ...'. Add ' -I$(PYTHON_HOME)' to the end of this line. To test out the environment, change directory to 'cocotb/examples/dff/tests' and run:

```
TOPLEVEL_LANG=vhdl SIM=ghdl make
```

3.2 Setting up a cosimulation project

While the Xyce/Python interface is a Python library that can be included directly in any project, CocoTB is a Makefile-based interface that requires a special directory structure and parameters to run correctly. Figure 3 shows the directory structure used for the tiny_AES experiment. The parent directory consists of CocoTB Makefiles (Makefile, Makefile.ghdl, and Makefile.sim) with project-specific parameters set. It also contains two subdirectories. The "input" folder contains the entire digital design (aes_fsm_encrypt_no_rounds14_coupled.vhd), the digital component for the upset region (fsm_regx2x_digital.vhd), the corresponding analog circuit for the upset region (fsm_regx2x_.cir), a stimulus file that specifies which input vectors to simulate, and a Bash script (build_ghdl.sh) for compiling/synthesizing the digital design via GHDL. The "sim_build" folder is where the VHDL libraries are built and CocoTB simulation output files are stored.

We refer the reader to our Sandia GitHub repository [5] for how parameters are set for this project, and recommend the use of this project as a template for other similar projects. Once your environment is set up according to 3, simply enter 'make' from the parent directory to run the project. Also note that this project is located on a Sandia Official Use Only (OUO) repository server, so may only be accessed by users with proper permissions.

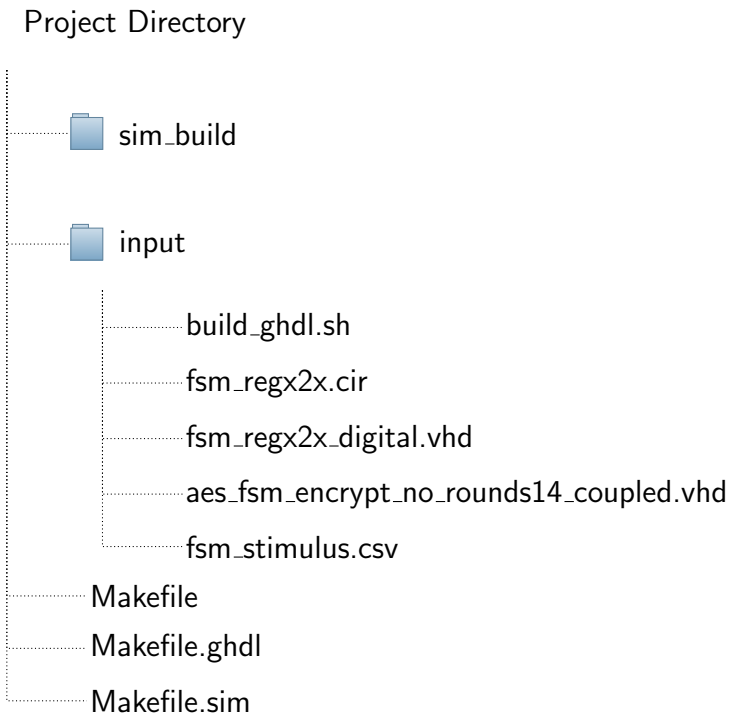

Figure 3: Directory structure for a sample CocoTB/Xyce cosimulation project.

4 Example analysis of the AES encryption core

With the cosimulation project set up, we now wish to run and analyze the state machine within the tiny_AES encryption core design (described in Chapter 2). For this example experiment, our UOI involves the effect of a photocurrent on an internal flip-flop. We wish to control the time at which this upset occurs (or, the time that we switch the simulation of the flip-flop from digital to analog), and measure the effect of the UOI on the entire design. Our project (set up in Chapter 3) can be run by setting the UOI parameters in config.py and running the Makefile[1].

4.1 Specifying an upset

Specifying the UOI requires defining spatial and temporal parameters. Spatial parameters include how many gates in the digital design are to be simulated as analog at the time of the upset. These parameters must be incorporated into the design files in the "input" directory. For our example, we choose one flip-flop in the tiny_AES FSM named "fsm_regx2x", a gate near the output of the design. Adjusting these parameters can be done in an automated way, but tools to do so are currently in development and not publicly available. Temporal parameters of the UOI define the time interval that an analog simulation is active. We call t_{start} the time that the analog simulation begins and t_{end} the time it ends. These are specified in "build.py" and are easily adjustable.

4.2 Initializing an analog circuit

Analog circuits contain more parameters than their digital counterpart. For instance, a digital model of a gate can be represented by a truth table, while its corresponding analog model is represented by a network of transistors and potentially complex, nonlinear DAEs. As a result, initializing an analog circuit from a digital state is non-trivial, since the only information provided by the digital state is input and output voltage. For CMOS gates, we performed experiments to see if the analog state can be "learned" from the digital state in various contexts. Flip-flops in particular can be difficult for this task due to the time-dependence of the outputs on internal voltages. Figure 4 shows the relationship of internal voltages in the analog model to the resulting digital output of a CMOS7 flip-flop over 100 Xyce simulations with randomized inputs. Here we see clear-cut patterns corresponding to digital 1s and 0s at the output. For this particular flip-flop, this means that if we wish to initialize the output to a digital 1, we initialize all the internal voltages to the values indicated by the blue dots (and equivalently, the red dots for a digital 0). Similar experiments were conducted on the other CMOS7 gates present in the tiny_AES design with the same results. Note that these results may be dependent on the target technology and clock speed, so such

[1] A batch file can easily be written for parameter sweeps, etc., however, we focus on running one instance for the sake of simplicity

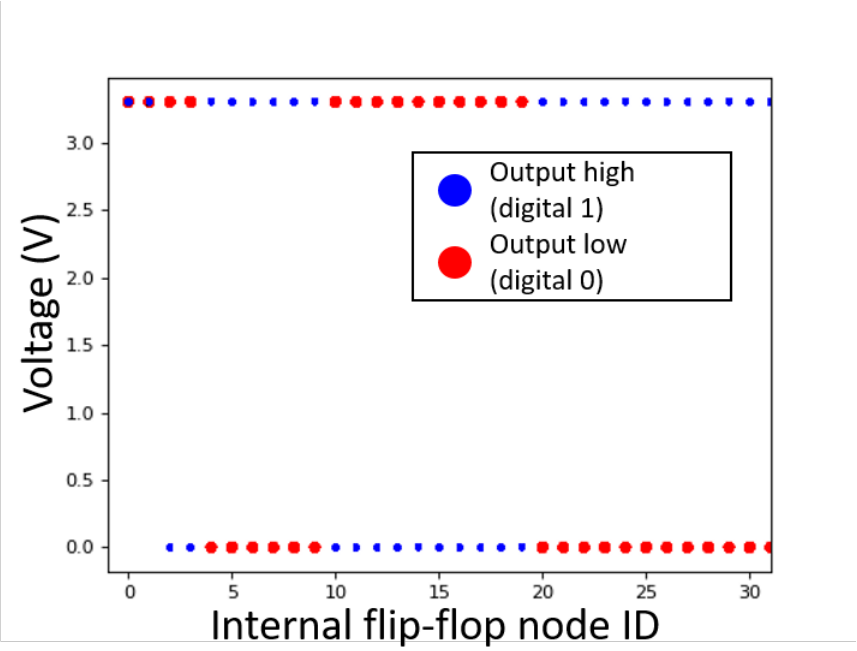

Figure 4: The internal voltages of 100 Xyce simulations of a CMOS7 flip-flop with randomized inputs. Red circles indicate those simulations that resulted in a low output (digital 0), and blue circles indicate simulations that resulted in a high output (digital 1).

a straightforward approach may not always work. Sensitivity analysis over these parameters is left for future work.

4.3 Analyzing the results of a simulation

Since a full digital system and an analog component have been simulated in GHDL and Xyce, respectively, we now have output files corresponding to both. If we examine the ".vcd" file output by GHDL from the time the UOI was imposed and onwards, we see the digital effect of the analog upset. Examining the ".prn" file output by Xyce allows us to observe the analog behavior of the upset.

Figure 5a shows the result of a purely digital simulation with no upset as a baseline for comparison. Here, the AES FSM goes through a typical cycle of encrypting a key. After all the good states have been traversed, the state machine outputs a high "finished" signal to indicate that a full round key has been generated. For validation, equivalent results are observed when running digital/analog cosimulations nominally (without any upsets) with several different, independent analog components. We then apply a photocurrent upset on one of the flip-flops, "fsm_regx2x", at 800 ns. Figure 5b shows the effect of this upset in the red box. After the first "finished" signal is asserted, the state machine starts to process another key. However, at 800 ns, the "key_ready" de-asserts, which should cause the FSM to reset. However, we see the upset causes the "finished" signal to pulse high anyways, which

could have a negative effect downstream from the encryption core. We examine the analog effect of the photocurrent in Figure 5c.

4.4 Future analysis techniques

The presented cosimulation techniques exhibit a proof of concept for including analog behavior in digital analysis. The task of building a simulation project remains mostly manual and tedious. Through future software engineering efforts, however, automating most of the simulation process seems tractable. Automation would allow for sensitivity analysis and uncertainty quantification to be performed on analog upsets in digital simulation. Capturing the digital effect of electrical upsets within a design enables more powerful analysis methods that would otherwise be intractable at the analog level. Formal methods, for instance, involving the proof of safety and security properties is a difficult task. We can leverage tools used for formal analysis of discrete systems using the digital projection of analog upset behavior to prove out-of-nominal properties as well (or at least give confidence bounds for specific upset scenarios). Demonstrations of this have been shown on small problems, but more research and development in automating cosimulation and projection methods is still required.

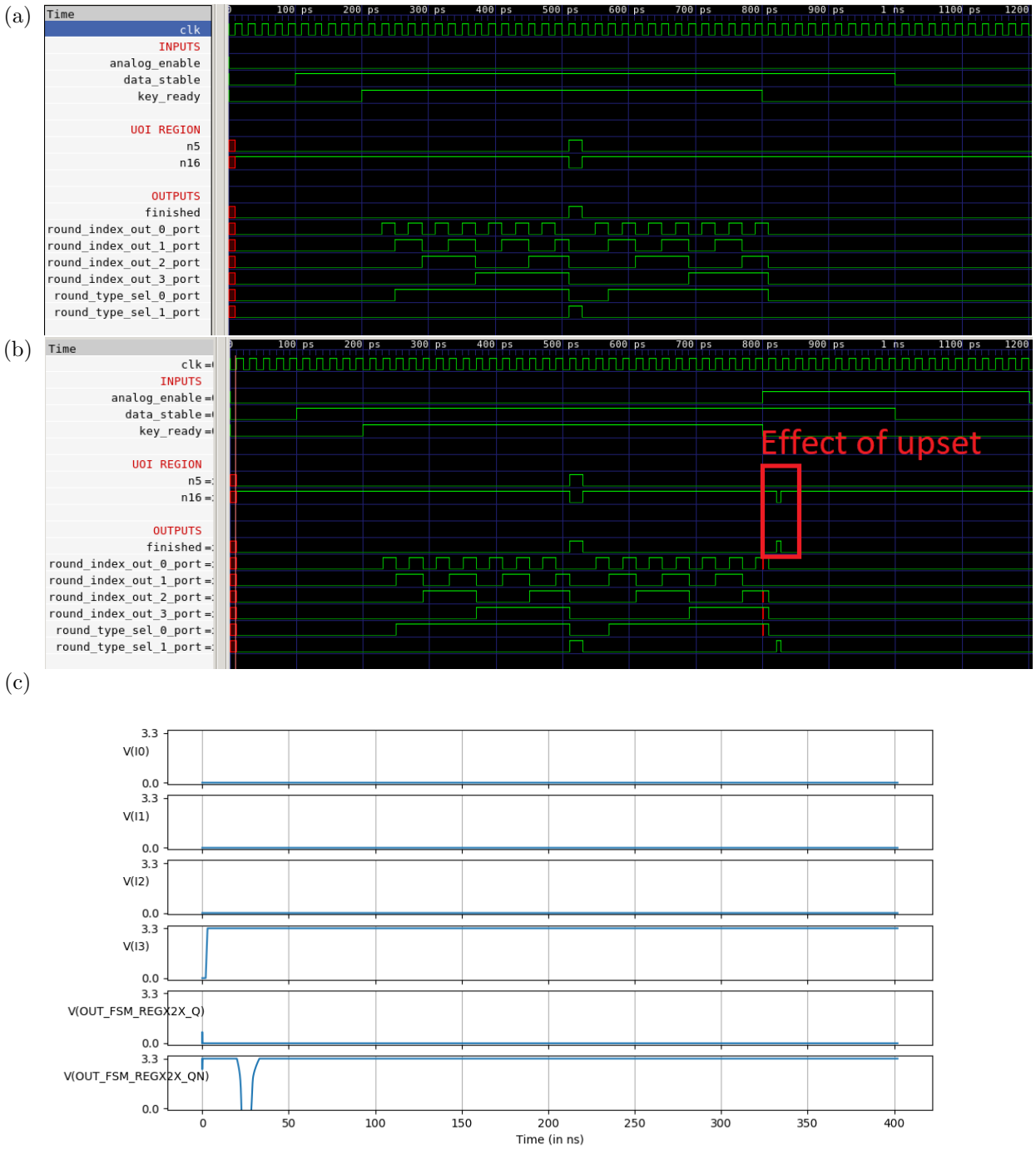

Figure 5: Results of cosimulation on the finite state machine of the tiny_AES encryption core. (a) shows the result of a nominal, purely digital simulation. (b) is the result of applying a photocurrent upset to the flip-flop "fsm_regx2x" at 800 ns; we see that, even though the "key_ready" input is not high, the upset causes the state machine to register as "finished". (c) shows the analog effect of the upset on the flip-flop's output "OUT_FSM_REGX2X_QN".

References

[1] CocoTB. http://potential.ventures/cocotb/, Last accessed on 2018-10-15.

[2] Tristan Gingold. GHDL. http://ghdl.free.fr/, Last accessed on 2018-10-15.

[3] Homer Hsing. Tiny AES. https://opencores.org/project/tiny_aes, Last accessed on 2018-10-15.

[4] Peter E. Sholander and Richard L. Schiek. Application note: Mixed signal simulation with Xyce. Technical Report SAND2018-TBD, Sandia National Laboratories, 2018.

[5] Andrew M. Smith. PyCAT FSM. https://gitlab.sandia.gov/amsmith/pycat_FSM, Last accessed on 2018-10-31.

[6] Andrew M Smith, Jackson R Mayo, Vivian Kammler, Robert C Armstrong, and Yevgeniy Vorobeychik. Using computational game theory to guide verification and security in hardware designs. In *Hardware Oriented Security and Trust (HOST), 2017 IEEE International Symposium on*, pages 110–115. IEEE, 2017.


Sandia National Laboratories