Smart Card Security

Volume II (of II)

Asier Goikoetxea Yanci

A themed portfolio submitted to

The Universities of

Glasgow

Edinburgh

Strathclyde

Heriot Watt

For the Degree of

Doctor of Engineering in System Level Integration

2011-February

Sponsored by:

List of Technical Reports and Publications:

Engineering Doctorate

Glitch Attack and Power Analysis Simulation Environment

Author: Asier Goikoetxea Yanci

Date: 2007-05-15

Sponsored by:

Table of Contents

1 Introduction

During the development process of an ASIC, the design goes through several test steps to make sure its behaviour and performance (e.g. area, power and speed) matches that intended in the initial specification. These tests happen at different stages or levels of the development (e.g. RTL, gate netlist and post-layout) and they are made considering the normal operating conditions the circuit has been designed for. However, devices such as Smart Cards can be deliberately forced to work under abnormal conditions in order to extract otherwise inaccessible information. Doing so is considered an attack.

Some attack examples include applying noise and or glitches to the power supply rails, external clocks and or any other external signal, such as communication lines. These could result on injecting faults by corrupting data and or skipping CPU instructions. Laser attacks (a.k.a. light attacks) can also be employed to inject faults into the design and/or locate functions on the die. Current consumption could be monitored for levels that indicate the use of certain features or operations (e.g. usage of crypto-block or writing to the NVM). Furthermore, monitoring the current consumption is the basis for techniques such as differential power analysis (DPA) to disclose the encryption key of a cryptographic block.

Atmel's design flow included some limited glitch attack, laser attack and DPA tests on silicon. Testing devices for these kinds of threads at this development stage can be very expensive, as any weakness identified at this stage could imply the need to develop new masks. Hence the need to develop a test environment that could test the designs' robustness against these attacks.

In order to fill this gap, a simulation environment was designed aimed at testing designs against glitch attacks and encryption engines against power analysis. This report covers the work carried out on this subject and the results obtained from it. The next sections are divided as follows: section two provides a background of the two different attacks targeted by the simulation environment. Section three covers how the simulation environment has been structured and implemented. Section four covers the simulation environment validation process and the results obtained. Section five presents the conclusions drawn by this work. And, finally, section six, propose a few possible future development lines for this simulation environment.

2 Attacks Background

This section is divided in two sections which will cover the two different attacks targeted by the simulation environment. Firstly glitch attacks will be introduced and, afterwards, differential power analysis (DPA) will be covered.

2.1 Glitch attacks

The voltage that digital systems should be powered to is limited by the technology it has been developed with, which sets both the upper and lower limits. If a device is powered to a voltage level below the lower limit, transistors might not go into the saturation state, thus, its correct behaviour in this circumstance is by no means warranted. On the other hand, powering the device to a voltage level above the upper limit might damage the device permanently. As long as the power supply is relatively constant and within the above mentioned limits, the device should behave correctly.

A noisy supply voltage (e.g. with a ripple and/or spikes) can inject faults into a device (e.g. errors, data corruption and/or alter the program execution flow), making it misbehave. Glitches can produce similar or worse effects. They can be applied to different places of a design, on its power or input pins. Applying the same pulse on different places and at different times may result on injecting different faults.

Smart Card devices are designed with a built-in voltage regulator, which can filter some of the noise present on the external power supply and regulate the voltage level to the level required by the Smart Card's internal circuitry. For slow voltage level changes on the Vcc pin, the voltage regulator would correct itself so that the output voltage is kept constant. However, for fast or sudden voltage level changes on the Vcc pin (glitch), the voltage regulator could reflect that change on its output. Figure 1 shows the effects on internal power voltage when a glitch is applied to external power line. Obviously, internal glitch's waveform may vary depending on the circuit load at the time the glitch is applied and on the voltage regulator.

When a glitch is applied to an input pin, however, power voltage could be raised via electro static discharge (ESD) protection diodes in the input pads. This is shown on figure 2, where for $Vpin > Vdd$, $Vdd = Vpin – Vdiode$; and for $Vpin < GND$, $GND = Vpin + Vdiode$.

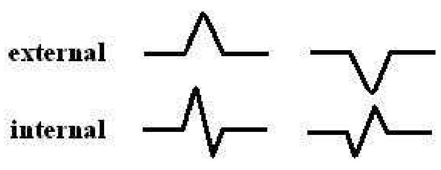


Figure 1 Example of resulting internal power waveform to an external pulse

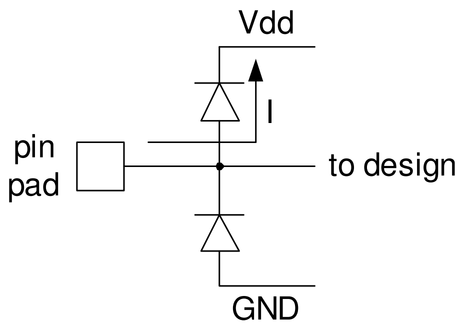


Figure 2 Input pin to Vdd connection under glitch attack

Temporal internal supply level changes do not propagate equally across the whole device due to parasitic resistors and capacitors on the power rails. In addition to over and under saturating transistors, the uneven propagation of the voltage level results on adjacent circuit blocks being powered at different voltage levels, which can also inject faults into the design. As covered in the report 'Simulation results of pulses applied to a counter's power signal', glitches are more effective when applied close to the clock's positive edge, as it helps latching or registering any error injected in the combinational logic.

2.2 Power analysis

The dynamic power consumed by a CMOS circuit is related to the data processed by this circuitry. This phenomenon is commonly known as side-channel leakage, and enables guessing the processed data by analysing the power consumption waveform. One kind of power analysis is simple power analysis (SPA), which consists of visually monitoring the power consumption to identify data. SPA could be use to identify the operand value of an instruction executed by a CPU. Also, due to the power consumption patterns of an EEPROM write/erase cycles, SPA could also be used to identify these.

Another kind of analysis possible on the consumed power is differential power analysis (DPA). DPA is an analysis technique originally proposed by Krocher [1] to guess the encryption key used by a cryptographic algorithm. This is achieved by exploiting the side-channel information leaked by the cryptographic algorithm implementations when processing a plaintext.

Figure 3 shows graphically the factors involved in a power analysis. Here, the aim is guessing the encryption key, which is secret. The attacker must know the cryptographic algorithm used during the encryption/decryption process, although the actual implementation might be unknown. The attacker must also have control over the data sent for encryption or know which data is being processed on encryption operations.

Two typical encryption algorithms used by Smart Cards are data encryption standard (DES) and advanced encryption standard (AES). These algorithms have a few common features:

- they are symmetric, i.e. they use the same key for encrypting and decrypting;
- the substitution box (S-box) is their basic building block;
- the input of these S-boxes is a combination of the encryption key and input data;
- the encrypted/decrypted data is generated after processing the input data several times (or rounds) through these S-boxes.

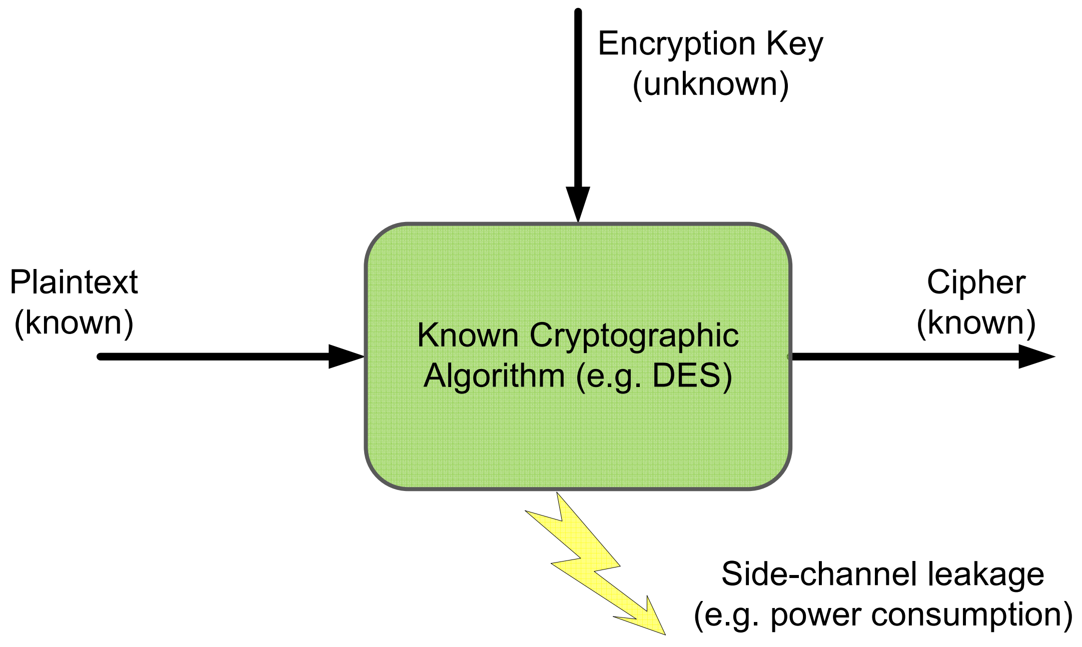


Figure 3 Factors involved in a power analysis

By controlling the input data of the cryptographic algorithm (e.g. plaintext when performing an encryption operation), the attacker can control the input of individual S-boxes in the first encryption/decryption round. Every encryption/decryption operation results in a power consumption waveform, here referred to as a power trace. The attacker would repeat this operation until all the S-box input combinations are covered several times, producing enough power traces to perform the analysis and to guess the encryption key bits associated to the target S-box.

Different analysis methods have been proposed since differential power analysis was introduced. In its original version, DPA used a statistical difference-of-means technique to guess the encryption key. The other common technique for guessing the encryption key is correlating the sampled power traces with an estimation of power consumption. Another development of the DPA is differential electromagnetic analysis (DEMA).

3 Literature review

For the literature review, please refer to the simulation environment literature review section in the first Volume of this portfolio, 3.1.1 Literature Review for Fault Injection and Side-channel.

4 Simulation environment approach and implementation

Designs can be simulated at different levels, behavioural level, RTL level, gate level and transistor level. High level simulations are used to validate the design’s functional model, whereas low level simulations are used to validate the signals’ timing and value.

The first feature to be implemented in this simulation environment was the glitch attack capability. Because glitch attack simulations require a tool capable of simulating how changes in the supply voltage affect the design, it was decided to develop the simulation environment around a transistor level simulation tool, such HSPICE or Nanosim (a SPICE-like simulation tool). The simulation environment was later updated to also perform DPA on DES cryptographic blocks.

In synthesis, the simulation environment is a set of Perl scripts and a configuration file. It creates a set of input files to be used by the simulation tool. These files depend on the targeted attack. The following sub-sections describe the simulation environment in more detail and the configuration specific to each attack.

4.1 General description

This simulation environment is a set of Perl scripts developed around the Nanosim simulation tool. The Perl scripts, totalling around 2,300 lines of code, are used to automate and ease the glitch attacks and power analysis simulations by:

- generating the required stimulus for the design under test (DUT)
- generating the configuration files for the simulation tool
- launching the simulation tool
- doing any required post-simulation analysis

The simulation configuration is defined in the file `simulation.cfg` (see Appendix A in the Volume I of this portfolio), and currently only the Nanosim simulation tool is supported. Figure 4 shows a high level diagram of the simulation environment.

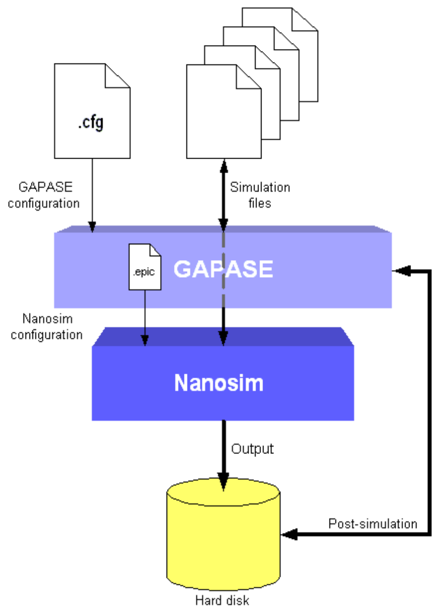


Figure 4 Diagram of the GAPASE simulation environment

Three kinds of simulations are possible: normal; glitch attack; and power analysis. Normal simulations are those where the designs are simulated under normal circumstances and no attack or analysis is aimed at them. This kind of simulation can be used to ensure the DUT's correct behaviour before being subjected to attacks or analysis. Glitch attack simulations are those where the DUT is targeted by a glitch in the power supply and/or noise is applied to other input signals. Power analysis simulations are those where a cryptographic block's implementation is tested against side-channel leakage.

All three simulation types need the same basic input files to run a simulation: the simulation configuration file, `simulation.cfg`, whose contents are covered in the next sub-sections; the HSPICE netlist(s) of the DUT (parasitic information can optionally be included on a different file and format); the DUT's input stimulus, which are specific to the simulation type and DUT; and the technology's SPICE model. In addition to these input files, GAPASE also creates the Nanosim configuration file to meet each simulation's requirements, `epic_pow.cfg`. Figure 5 shows the GAPASE's simulation diagram flow.

Finally, GAPASE has been written in a modular approach using Object-oriented Perl (OOP). This design decision enables adding features in the future, such as targeting additional cryptographic algorithms, e.g. AES, or including the power trace analysis currently done in Matlab. Figure 6 shows a modular diagram of the current state of GAPASE.

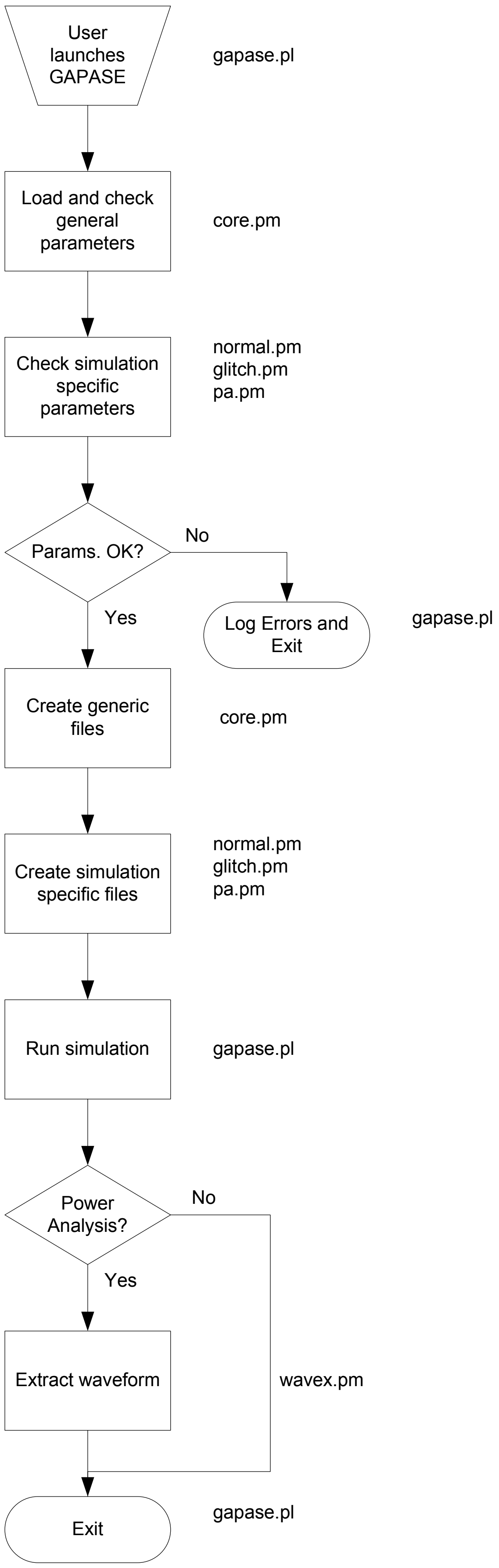


Figure 5 Diagram of the simulation flow

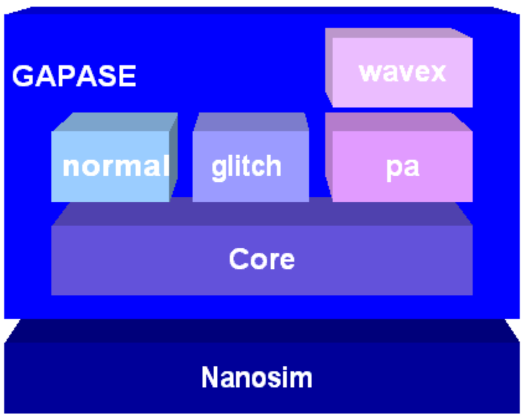


Figure 6 Modular diagram of GAPASE

4.2 Normal simulation

A normal simulation exercises the DUT with the stimulus and supply voltage defined in the `simulation.cfg` file. Here are the mandatory parameters for a normal simulation:

```
SIMULATION_NAME=
DESIGN_NETLIST=
VOLTAGE=
VOLTAGE_NODE=
STIMULUS=
SIMULATION_TIME=
TEMPERATURE=
MODEL_LIB=
MODEL_LIB_CALLS=
SIMULATION_TYPE=
DETACH=
```

These are optional simulation parameters:

```
LOG_LEVEL=
DESIGN_RC_NETLIST=
DESIGN_RC_TYPE=
```

The `SIMULATION_NAME` parameter defines the name assigned to the simulation and to the simulation's output file. It also defines the name of the directory where the simulation output and the configuration file will be stored. If a directory with the same name exists, the simulation environment would allow overwriting the old directory (effectively deleting it and creating a new one) or aborting the simulation. This last option would generate an error log message.

The parameter `DESIGN_NETLIST` points to the test circuit's HSPICE netlist file. If the file referred to by this parameter does not exist, the simulation is aborted and an error message generated.

The parameter `VOLTAGE` defines the voltage level to which the test circuit will be powered for the simulation. This voltage level will be applied to the node defined by `VOLTAGE_NODE`.

The parameter `VOLTAGE_NODE` defines the name of the circuit's power supply node, e.g. 'mixvdd!' or 'vdd!'.

The parameter `STIMULUS` points to the PWL file with the circuit's input stimulus information. If the file referred to by this parameter does not exist, the simulation is aborted and an error message generated.
The input stimulus waveforms defined in `STIMULUS` must be normalised to the voltage level defined in `VOLTAGE`. An amplitude value of 0 in a waveform meaning that 0V are applied to that particular input port; an amplitude value of 1 in a waveform meaning that `VOLTAGE` is applied to that particular input port.

The parameter `SIMULATION_TIME` defines the simulation time. Time unit must be specified, where 'n' means nanosecond, 'u' means microsecond, 'm' means millisecond and 's' means second.

The parameter `SIMULATION_TYPE` defines which kind of simulation is going to be performed. Possible values are: `NORMAL`, `GLITCH` or `PA`. For this case it should be set to `NORMAL`. When set to `NORMAL`, any glitch and DPA related parameter is ignored.

The parameter `TEMPERATURE` defines the simulation temperature in degrees Celsius.

The parameter `MODEL_LIB` points at the SPICE model library to be used in the simulation. If the file referred to by this parameter does not exist, the simulation is aborted and an error message generated.

The parameter `MODEL_LIB_CALLS` lists the model calls to fine tune the simulation.

The parameter `LOG_LEVEL` defines the log level to be used in the current simulation. It can be set to `HIGH` or `LOW`. If not defined, it is defaulted to `LOW`.

The parameter `DETACH` determines whether the simulation is detached from the console terminal or not. Two values can be defined for this parameter: `YES` or `NO`. Detaching the simulation allows closing the console terminal the simulation was launched from even when the simulation is still running without interrupting it. This allows logging out the session after launching the simulation. Not defining this parameter or giving it a wrong value would have the same effect as setting it to `NO`.

The parameter `DESIGN_RC_NETLIST` points to the circuit's parasitic information file. If this parameter is empty, the parameter `DESIGN_RC_TYPE` is also ignored. If the file referred to by this parameter does not exist, the simulation is aborted and an error message generated.

The parameter `DESIGN_RC_TYPE` defines the circuit´s parasitic file´s format. This parameter is linked to `DESIGN_RC_NETLIST`,and its value only checked if the parent parameter is defined. Two are the options, `SPEF` or `HSPICE`. If the netlist is in the SPEF format, it is parsed to modify the instance and bus naming convention to suit that of HSPICE.

All the parameters associated to normal simulations also apply to glitch and power analysis simulations. Not defining a mandatory parameter would result on aborting the simulation and generating error messages.

4.3 Glitch attack simulation

A glitch attack simulation enables applying glitches and/or noisy power sources to the DUT's `VOLTAGE_NODE` and/or the GND. It also allows applying noise to the DUT's input stimulus signals. Noisy power sources and input stimulus signals can be applied with the use of SPICE PWL waveforms. Glitches can be defined with a simulation environment built-in glitch generation function or with SPICE PWL waveforms.

In addition to the parameters associated to a normal simulation, a glitch attack simulation also has the following parameters associated:

```
POWER=
POWER_START=
DELAY_FOR_PULSE=
PULSE_START_TIME=
PULSE_START_VALUE=
PULSE_P1_TIME=
PULSE_P1_VALUE=
PULSE_P2_TIME=
PULSE_P2_VALUE=
PULSE_END_TIME=
PULSE_END_VALUE=
GND=
GND_START=
GND_DELAY_FOR_PULSE=
GND_PULSE_START_TIME=
GND_PULSE_START_VALUE=
GND_PULSE_P1_TIME=
GND_PULSE_P1_VALUE=
GND_PULSE_P2_TIME=
GND_PULSE_P2_VALUE=
GND_PULSE_END_TIME=
GND_PULSE_END_VALUE=
STIMULUS_NOISE=
STIMULUS_NOISE_START=
```

Parameters POWER and POWER_START are used to define a noisy power source or glitch on the DUT's VOLTAGE_NODE. The parameter POWER points at a SPICE PWL file containing the power source's behaviour. This is an optional parameter, but once defined, it has to point at a valid file. Otherwise the simulation environment will exit with an error message. The parameter POWER_START states the time at which the waveform described in POWER should be applied to the VOLTAGE_NODE. This parameter is optional and checked only if POWER is defined. If this parameter is not defined, it is defaulted to 0n.

The POWER waveform must be defined as a normalised offset to the continuous supply voltage level defined in VOLTAGE. An amplitude value of 0 in the waveform meaning there is no change on the supply voltage level applied to the VOLTAGE_NODE; an amplitude value of 1 in the waveform meaning that the supply voltage level is increased by VOLTAGE; and an amplitude value of -1 in the waveform meaning that the supply voltage level is decreased by VOLTAGE. It is mandatory to name this waveform with vpower_waveform.

Parameters GND and GND_START are used to define a noisy power source or glitch on the DUT's ground power rail or GND. The parameter GND points at a SPICE PWL file containing the GND's behaviour. This is an optional parameter, but once defined, it has to point at a valid file. Otherwise the simulation environment will exit with an error message. The parameter GND_START states the time at which the waveform described in GND should be applied to the DUT's GND. This parameter is optional and checked only if GND is defined. If this parameter is not defined, it is defaulted to 0n. The GND waveform must also be defined as a normalised offset to the continuous voltage level applied to GND, i.e. 0V. This waveform must be named with vgnd_waveform.

Noise can also be applied to a DUT's input signal with parameters STIMULUS_NOISE and STIMULUS_NOISE_START. Again, the parameter STIMULUS_NOISE points at a SPICE PWL file containing the noisy behaviour of the input stimulus signals. This is an optional parameter, but once defined, it has to point at a valid file. Otherwise the simulation environment will exit with an error message. The parameter STIMULUS_NOISE_START states the time at which the waveform described in STIMULUS_NOISE should be applied to the DUT's input signals defined in STIMULUS. This parameter is optional and checked only if STIMULUS_NOISE is defined. If this parameter is not defined, it is defaulted to 0n. The STIMULUS_NOISE waveform must also be defined as a normalised offset to the stimulus signals defined in the file pointed at by the parameter STIMULUS. It is recommended to prefix the name of the noise signals. Prefixes 'org_' and 'nom_' are already used by the simulation environment.

In addition to the waveforms, the simulation environment can also apply glitches to both power rails with its built-in glitch generator function. This function allows applying any glitch definable by up to 4 points, see Figure 7. Figure 8 shows the glitches that can be defined with this function.

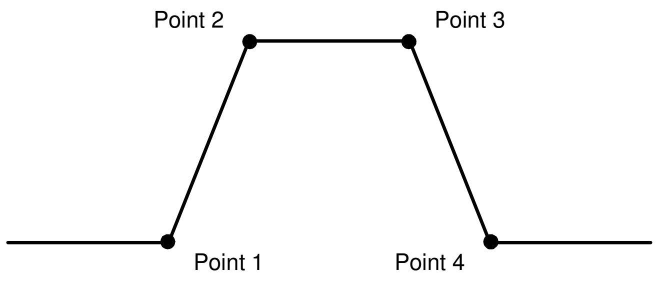


Figure 7 Four point definable glitch

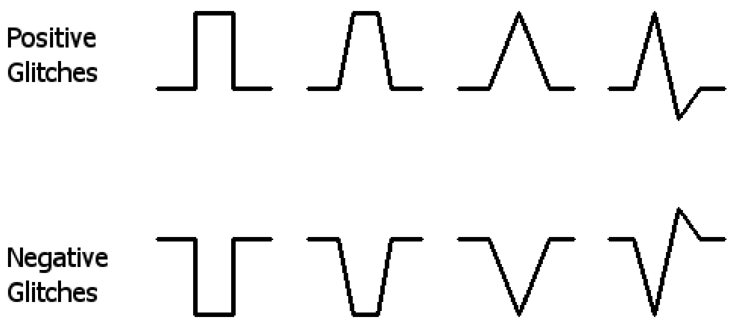


Figure 8 Example of applicable glitches on the simulation environment

The built-in glitch generation function can be used to apply glitch to both power supply nodes, `VOLTAGE_NODE` and GND. Nine parameters are used for each case, two parameters to define the time and normalised amplitude offset of each point in Figure 7 and one parameter to define a reference time point before the glitch is applied.

For glitches to be applied in `VOLTAGE_NODE`, the time reference parameter is named as `DELAY_FOR_PULSE`. The parameters associated to each point are named as follows:

- point 1: `PULSE_START_TIME`, and `PULSE_START_VALUE`,
- point 2: `PULSE_P1_TIME` and `PULSE_P1_VALUE`,
- point 3: `PULSE_P2_TIME` and `PULSE_P2_VALUE`,
- point 4: `PULSE_END_TIME` and `PULSE_END_VALUE`.

All four time parameters must be defined when applying a glitch. The time values defined in these parameters must be an incremental offset from the `DELAY_FOR_PULSE` parameter so that the following condition is met:

0<= PULSE_START_TIME < PULSE_P1_TIME < PULSE_P2_TIME < PULSE_END_TIME

All four value parameters take normalised offset values. Parameters PULSE_P1_VALUE and PULSE_P2_VALUE are mandatory. Parameters PULSE_START_VALUE and PULSE_END_VALUE are optional. When these optional parameters are not defined, they are defaulted to 0, which is the recommended value. Not defining any mandatory parameter would result in logging an error message an exiting the simulation environment.

The same applies to the parameters for GND glitches. On this case, the time reference parameter is named as GND_DELAY_FOR_PULSE. The parameters associated to each point are named as follows:

- point 1: GND_PULSE_START_TIME, and GND_PULSE_START_VALUE,
- point 2: GND_PULSE_P1_TIME and GND_PULSE_P1_VALUE,
- point 3: GND_PULSE_P2_TIME and GND_PULSE_P2_VALUE,
- point 4: GND_PULSE_END_TIME and GND_PULSE_END_VALUE.

As explained in the report 'Counter Simulation Results', the closer a glitch happens to a clock edge, the more effective it is. The following set of formulas can be used to calculate the value of parameters DELAY_FOR_PULSE or GND_DELAY_FOR_PULSE in relation to clock edges.

For referring to the beginning of a negative-edge:

$$n\_clocks \times period \quad (1)$$

For referring to midway of a negative-edge:

$$(n\_clocks \times period) + \frac{t_{fall}}{2} \quad (2)$$

For referring to the beginning of a positive-edge:

$$(n\_clocks \times period) + \frac{period}{2} \quad (3)$$

and for referring to midway of a positive-edge:

$$\left(n\_clocks \times period\right) + \frac{period}{2} + \frac{t_{rise}}{2} \quad (4)$$

where *period* represents clock period time, i.e. *1/frequency*, *n_clocks* is the clock cycle number to act as a reference for the glitch, t_{fall} represents clock signal's fall time and t_{rise} represents clock signal's rise time. The values of t_{fall} and t_{rise} are defined by the variables `tfall` and `trise` in the file `nanosim/nanosim_deck/vtran_setup`.

The parameter `DELAY_FOR_PULSE` is the parent parameter for glitches on `VOLTAGE_NODE`. If this parameter is defined, all remaining 8 parameters are taken into account. Otherwise they are ignored. Equally, the parameter `GND_DELAY_FOR_PULSE` is the parent parameter for glitches on GND. All remaining 8 parameters are considered only if this is defined.

The simulation environment allows applying in a simulation any of the noise sources already mentioned or a combination of them. SPICE vector operations are used to apply any of the noise sources defined above. Figure 9 and Figure 10 show the DUT's supply waveforms are generated and the name of each vector as used in the simulation environment.

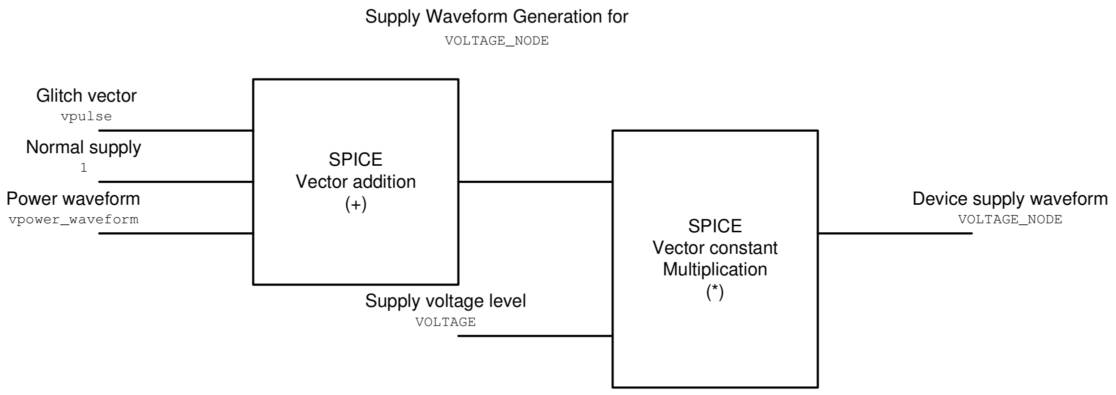


Figure 9 The DUT's supply waveform generation

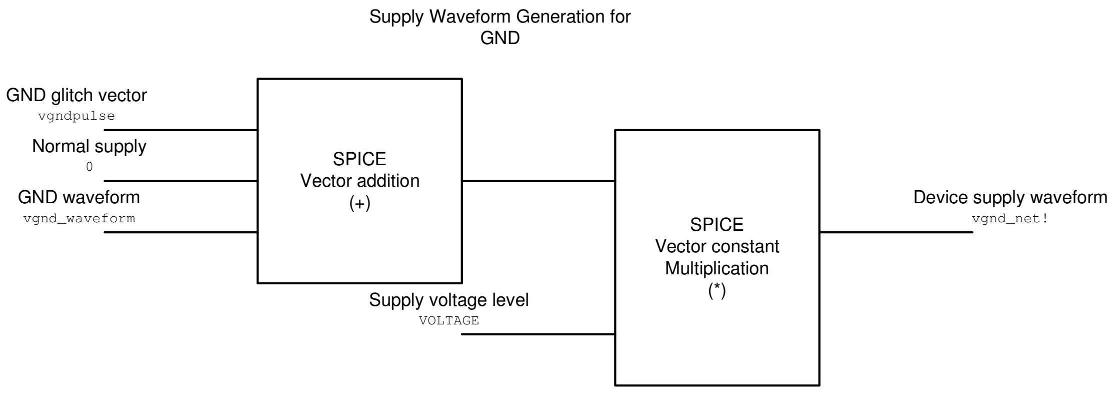


Figure 10 The DUT's GND supply generation

HSPICE design netlist files define GND as ‘0’ instead of giving it a node name. In order to apply glitches and noise on the GND, its default definition needs to be changed with a net name. The simulation environment parses the HSPICE netlist file to replace all the ‘0’ related to GND with the node name `vgnd_net!`. This is done only the first time a design netlist file is used within the simulation environment.

4.4 Power analysis simulation

The power analysis feature of the simulation environment is divided into two steps. The first one is power trace generation, which is achieved by simulating the data encryption process in Nanosim. The second step is power trace analysis, which is achieved by running a Matlab analysis script. Figure 11 shows a diagram of the files and steps involved on performing power analysis with the current environment. The following sub-sections discuss these steps in more detail.

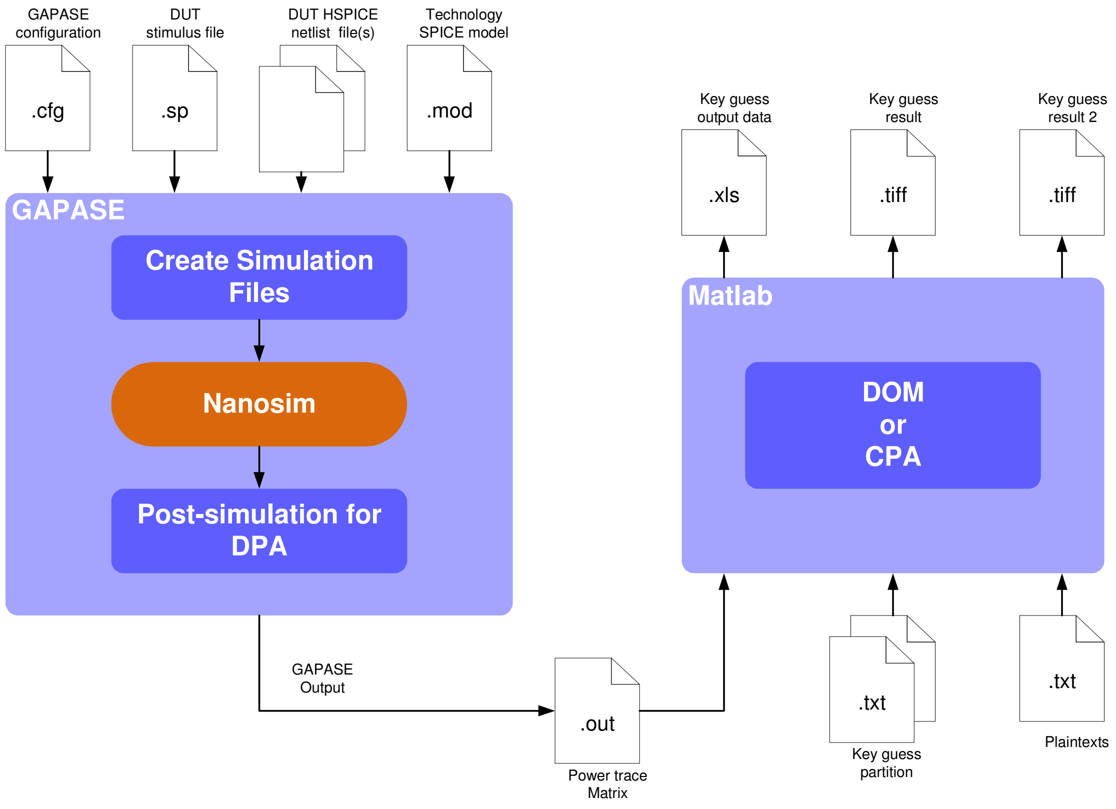


Figure 11 Diagram of the key guessing process

4.4.1 Power trace generation with GAPASE

The following parameters are used to configure a power analysis simulation:

```
CRYPT=
CLOCK=
ROUNDS=
SAMPLES=
CPU_NUMBER=
CLOCK_IN_NAME=
CRYPT_RESET=
FULL_RESET=
FORCE_NETS=
LOAD_RUN_CYCLES=
START_CRYPT=
KEY=
TARGET_KEY_BITS=
KEYREG=
DATAREG=
KEY_NET=
DATA_NET=
```

The parameter `CRYPT` defines the target cryptographic algorithm. Currently, the simulation can only target DES modules, but this parameter allows defining future target algorithms such as AES.

The parameter `CLOCK` defines the clock frequency in kHz that the target cryptographic module will be exercised at.

The parameter `ROUNDS` defines the number of rounds to be simulated. Theoretically, the power trace generated on the first (or last) round is enough to carry a DPA on a DES module. More complex power analysis methods (e.g. second order DPA) need the power trace generated on the first two rounds. This parameter lets you configure the amount of rounds to be simulated and extracted.

The parameter `SAMPLES` defines the number of times the 64 basic plaintexts should be encrypted. A standard implementation will need fewer samples than a secured one.

The parameter `CPU_NUMBER` indicates how many CPUs of the server or PC will be used to run the simulation.

The parameter `CLOCK_IN_NAME` defines the cryptographic module's clock input pin name where the simulation environment will apply the frequency defined by the parameter `CLOCK`.

The parameter `CRYPT_RESET` defines the cryptographic module's reset input pin. The simulation environment will create automatically the required stimulus signal for this input.

The parameter `FULL_RESET` defines whether the cryptographic module should be fully or partially reset after computing a plaintext and before computing the next one. Full reset implies forcing all registers and nets to a reset status. Partial reset implies forcing only registers to a reset status. KEY and DATA registers are excluded from this later case. If this parameter is left blank or a wrong value is entered, the parameter would be defaulted to `YES` and a warning message will be logged.

The parameter `FORCE_NETS` defines whether the input or output of KEY, DATA and `START_CRYPT` registers should be forced at the beginning of each plaintext encryption. If this parameter is left blank or a wrong value is entered, the parameter would be defaulted to `INPUT` and a warning message will be logged.

The parameter `LOAD_RUN_CYCLES` determines whether KEY and DATA load operation should happen in the same clock cycle the encryption is launched or not. If the same cycle is used, KEY, DATA and `START_CRYPT` registers are forced to their right value within the same clock cycle. If different cycles are used, KEY and DATA registers are forced to their right value within one same clock cycle, and the register `START_CRYPT` is forced on the next clock cycle. If this parameter is left blank or a wrong value is entered, the parameter would be defaulted to `DIFFERENT` and a warning message will be logged.

The parameter `START_CRYPT` defines the name of the register that triggers the encryption process. This register is forced to logic 1 by the simulation environment at the beginning of each encryption.

The parameter `KEY` defines the encryption key value to be used in the current simulations. The key is introduced in hexadecimal, 8 bytes long for DES cryptographic modules.

The parameter `TARGET_KEY_BITS` identifies the target SBOX. The simulation environment uses this parameter to identify the position of the DATA and KEY bits associated to each SBOX. Currently only SBOX_1 can be targeted.

The parameter `KEYREG` indicates the name given to the register set used to store the key. The simulation environment will parse the DUT's SPICE netlist to look for these registers to identify their input/output net names. Alternatively, the parameter `KEY_NET` can be filled with each key registers' input and output nets' name.

The parameter DATAREG indicates the name given to the register set used to store the data (plaintext or cipher). Again, the simulation environment will parse the DUT's SPICE netlist to look for these registers to identify their input/output net names. Alternatively, the parameter DATA_NET can be filled with each key registers' input and output nets' name.

Unless otherwise specified, all parameters are mandatory. Missing any mandatory parameter will result on logging an error and aborting the simulation.

4.4.1.1 Issues and solutions to a DES module's simulation

A common approach on some power analysis setups is to simulate only the required minimum block to perform power analysis, see Figure 12. The GAPASE's power analysis feature, however, has been designed to simulate the whole cryptographic module. The main benefit of simulating the required minimum block is the simulation performance, as power traces will be generated faster than simulating the whole module. Simulating the whole module, however, brings other benefits, such as testing the cryptographic module as it is, without taking away any of its components. It also enables easily targeting any SBOX or testing countermeasures other than those applied to the SBOXes. For example dummy cycles.

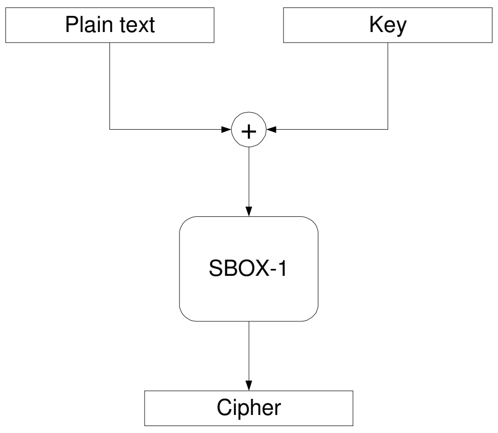


Figure 12 The minimum required circuitry to test a DES cryptographic block's sensitivity to power analysis

The main drawback of simulating the whole cryptographic module is performance. The performance suffers not only due to the need of simulating the whole cryptographic module's behaviour, but also by the need to initialise the DES module for each encryption with the right set of key and plaintext values.

Since only the first or first two rounds are needed to perform a power analysis, GAPASE only simulates the first few rounds (defined in ROUNDS). After simulating the defined amount of rounds, the cryptographic module is reset before moving onto the next plaintext encryption. As a result of halting the encryption process, the value left on the key register is no longer valid and needs to be updated.

Since the DES modules to be used in GAPASE have an 8-bit I/O interface, setting the key and plaintext for each encryption requires 16 clock cycles, 8 for setting the key and 8 for setting the plaintext. Launching the encryption would require an additional clock cycle. In other words, generating a 2-round (i.e. clock cycle) power trace requires the simulation of 19 clock cycles, of which only the last two produce valid data. This represents 10.52% of the whole simulation. This is even worse for the case of generating only a 1-round power trace, as only 1 out of 18 clock cycles produces valid data, 5.55% of the whole simulation.

GAPASE cuts short the simulation time required to initialise the DES module for each encryption by forcing the input or output nets of key and plaintext registers as well as the register that launches the encryption process to the desired value. For this purpose, Nanosim `force` and `release` commands are used. As described above, the parameter FORCE_NETS defines whether the input or output nets will be forced. Key and plaintext register input or output nets are forced at the same time and released at the same time too. The inputs or output of the register that launches the encryption, START_CRYPT, can be forced and released together with key and plaintext registers or on the next clock cycle to these registers. This is specified by the parameter LOAD_RUN_CYCLES. Figure 13 shows, in relation to the clock signal, the case where the input nets of plaintext, key and START_CRYPT registers are forced and released on the same cycle on a 2-round simulation. Figure 14 shows the case where the output nets are forced.

With this approach, the DES module could be initialised and an encryption launched within the same clock cycle. As a result, 100% of the simulation time would produce a valid power trace. Alternatively, the key and plaintext registers could be initialised one clock cycle before the encryption process is launched. In this case, between 66% and 50% of the simulation time would produce a valid power trace depending on the amount of simulated rounds, two or one.

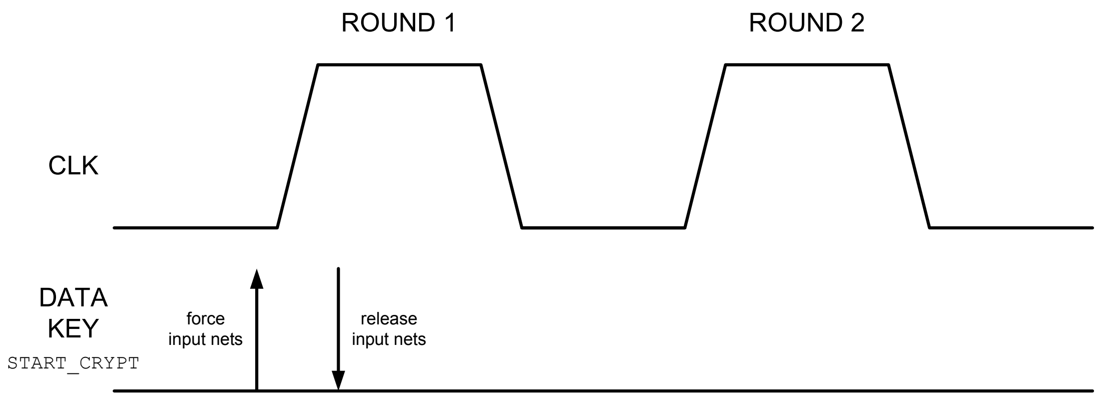


Figure 13 Plaintext and key input nets force release times on a 2-round simulation

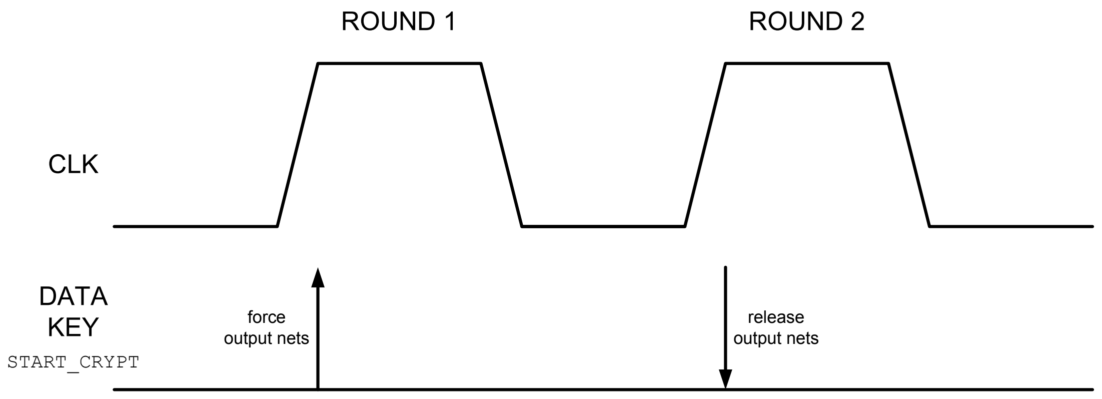


Figure 14 Plaintext and key output nets force and release times on a 2-round simulation

4.4.1.2 Power analysis simulation process

Once the simulation is configured by filling in the `simulation.cfg` file and launched, the simulation environment follows the next process. As shown in Figure 6, the simulation environment checks first that all mandatory parameters are defined. If any of these is missing or wrong, GAPASE would abort the simulation. If all parameters are OK, GAPASE would follow into creating the required files for the simulation.

All three simulation types create two stimulus and a Nanosim simulation configuration files:

- **`stimulis_nom.spi`:** this is the parent Nanosim configuration file. It defines the model library and the library calls to be used in the current simulation. It also points at the DUT netlist file and its stimulus file, `var_stim.sp`, see Appendix A for an example. The contents of this file are independent from the target simulation.

- **`var_stim.sp:`** this is a simulation type dependant stimulus file. This file holds the DUT's input stimulus and power information. On a glitch attack, glitch parameters and power supply waveforms are written into this file. On a power analysis simulation, this file holds the DUT's power supply definition and the clock signal generated based on the parameters `CLOCK_IN_NAME`, `CLOCK` and `SIMULATION_TIME`. A vector named '`not_connected_vector_only_used_for_sampling_at_tres_time_units`' is also defined in this file. More on this vector below. See Appendix A for a sample file.
- **`epic_pow.cfg:`** this is also a simulation type dependant stimulus file. This file configures Nanosim for the target simulation. It defines the simulation time and print resolutions and the sub-circuit current amplitude resolution. It also states SPICE model accuracy to be used during the simulation. Higher accuracy models take longer to simulate. These configuration parameters are coded within the Perl scripts and are a common requirement to all simulation types. The power analysis simulation also fills this file with a set of force-release statements to initialise the target DES modules and reset them before each new encryption. More on the force-release statements below. Again, see Appendix A for a sample Nanosim configuration file.

The DES module is reset after simulating the amount of rounds stated in the `ROUNDS` parameter. Resetting the DES module using the 'reset' input would cost precious simulation time as all nets and registers are set to their right voltage levels. In normal conditions, this process would take a whole clock cycle. Repeating this process over and over throughout the whole simulation process would have a severe impact on the simulation environment's performance.

Again, GAPASE cuts this time short by running first a short reset simulation of the DES module and reporting each node's value once it has been reset. This information is later used by GAPASE to quickly reset the whole DES module, or part of it (depending on the parameter `FULL_RESET`), with Nanosim force-release statements. These force-release statements are also included in the `epic_pow.cfg` file.

There is one last aspect that needs to be considered before launching the Nanosim simulation. Nanosim is an event driven simulation tool, and hence it only prints to the output file the value of a net when it changes instead of its value at every sampling time. This output approach might well be more time efficient than writing the net value for every sampling time and will result on smaller output files.

The power analysis process in Matlab, on the other hand, needs the current consumption value at every sample time, plus all power traces should be of the same length, i.e. same amount of samples. There are two possible ways of achieving this objective. One alternative is letting Nanosim run as normal and converting at the post-simulation stage the event driven output into a time driven one. This alternative might produce a better simulation performance at the expense of making the post-simulation step more complex.

The other alternative, and the one used by GAPASE is creating an unconnected SPICE vector that changes its value on every sampling time. This alternative makes the post-simulation step easier by providing a time stamp at which a current value should be provided. If at any given time stamp or sample time the current consumption value is not provided, the previous value is copied. The unconnected net name used by GAPASE is `not_connected_vector_only_used_for_sampling_at_tres_time_units` and it is located in the `var_stim.sp` file.

Once all the required files are created, the Nanosim simulation is launched. After finishing the simulation, the post-simulation step is started. Here the simulation output file(s) is parsed to extract only the power trace related to each encryption. All the information is then formatted as a matrix and stored into a file, where each row represents a power trace and each column represents a current sample.

4.4.2 Power trace analysis in Matlab

The power trace analysis step takes the output file generated on the simulation step, the key guess partitioning data and the plaintexts used in the simulation step. Two analyses can be carried on the generated power traces, difference-of-means (DOM) and correlation power analysis (CPA). Each analysis technique needs its own key guess partition data. The plaintexts must be in the same order as they were encrypted in the simulation step. Currently, GAPASE only targets the SBOX_1; hence, the power traces can only leak the 6 key bits associated to this SBOX.

DOM guesses a single key-bit at a time. Guessing all 6 key bits implies generating the key guess partition data and analysing the power traces for each target key bit. The DOM analysis script used in this environment is based on the Matlab script written by Andrew Burnside for his research.

CPA guesses multiple key bits at a time, typically all the input bits of the target SBOX. The CPA analysis script used in this environment has been written by me from scratch.

On either analysis, the analysis results are stored into an Excel file and two image files. One image shows the key with highest probability of a correct guess. The other image shows where in time it happens with the highest probability.

5 Simulation environment validation and results

This simulation environment needed to be validated before being used as a part of the design flow. The validation was divided into two parts, glitch and DPA. Glitch attack validation was achieved by testing the same design with the simulation environment and on silicon. DPA validation was achieved by testing two different DES modules with GAPASE; one a standard implementation with no countermeasures and the other one using countermeasures.

5.1 Glitch attack validation

The same counter circuit tested in SimEnvTech2 was used to validate the glitch attack feature of GAPASE. This counter was instantiated in the Tartalo (01OKA) test chip, covered in LaserTech1, and tested for a range of glitches of different amplitude and widths. The circuit's response to some of these glitches was also simulated on an updated version of GAPASE. Figure 15 shows a diagram of the test counter.

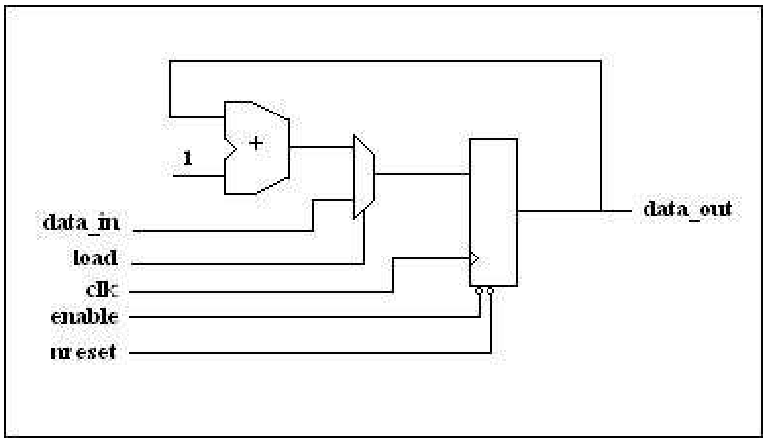


Figure 15 Counter

For test simplicity, in both cases (simulation and silicon test), the counter circuit was powered through a voltage regulator subjected to a load, as shown in Figure 16. The voltage regulator used in the validation process is the low security one used in the glitch detector characterisation process, GlitchTech1. The voltage regulator was subjected to a fixed 80 ohms resistive load and a 2.2nF capacitive load.

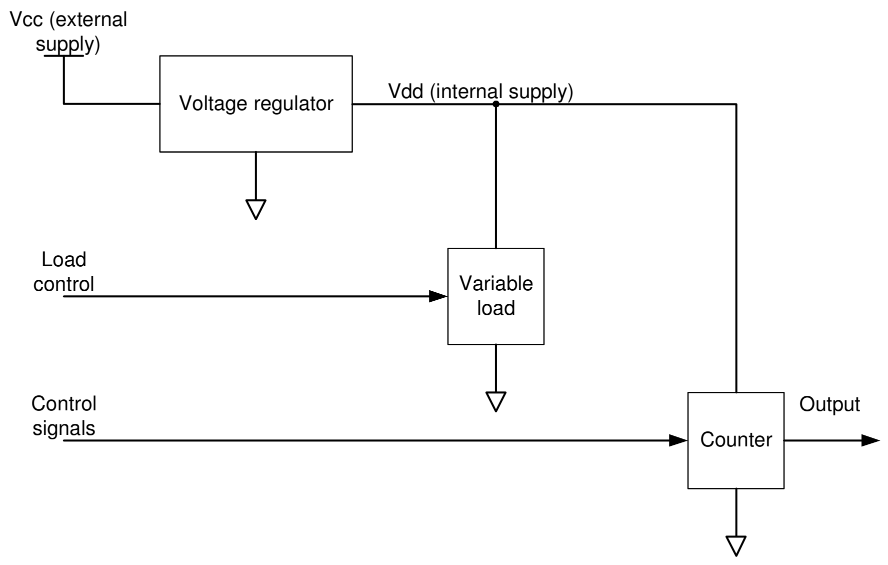


Figure 16 Setup for Glitch Attack validation

5.1.1 Silicon test

The glitch detector characterisation setup covered in the report GlitchTech1 was also used in the glitch attack validation process. Figure 17 shows a diagram of the setup used for the silicon test. The test environment was also configured and controlled by the same application, *VGlitch nenagi 4.14*. Figure 18 shows this application set to test the counter circuit (Tartalo test type option), which allows defining different load and glitch parameters.

The counter used in the validation was instantiated in the test-chip 01OKA (a.k.a. Tartalo) and the voltage regulator and load were instantiated in the test-chip 01VGA (a.k.a. Arrano). The voltage regulator in 01VGA powered both, the load and the counter as shown in Figure 17. The pulse generator HP81110A was connected to the voltage regulator's Vcc input and used to power and apply glitches to both test-chips.

The front-end application configured the pulse generator to apply glitches of different amplitudes and widths. Once the pulse generator was setup, the test control was transferred to the 'Test control IP' instantiated in the Smart Card Emulation Board. This IP set the load and controlled the counter's stimulus. After triggering a glitch attack, the 'Test control IP' monitored the counter's output and recorded the errors, which were later logged by the front-end application. Refer to the GlitchTech1 report for more details on the front-end application, the HP 81110A pulse generator, the MP300 device, the Voyager System (Smart Card Emulation Board) and the test board.

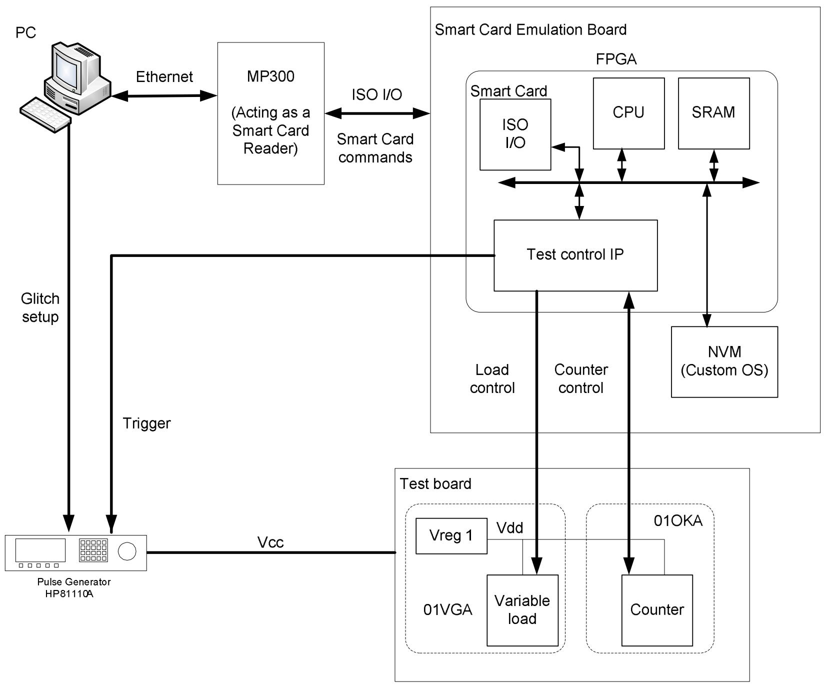
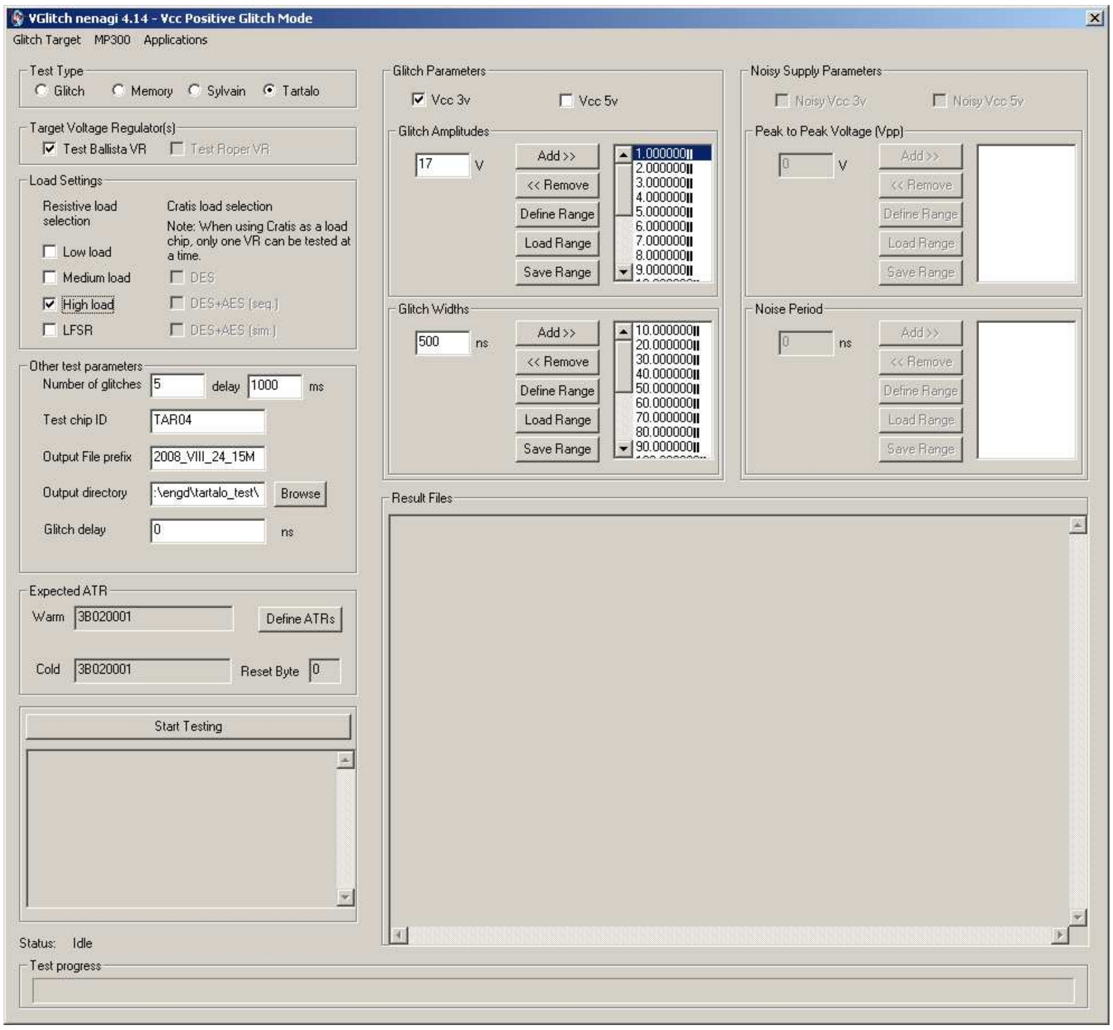


Figure 17 Counter circuit's silicon test setup



Test Type: ( ) Glitch ( ) Memory ( ) Sylvain (•) Tartalo

Target Voltage Regulator(s): [✓] Test Ballista VR [ ] Test Roper VR

Load Settings

Resistive load selection: [ ] Low load [ ] Medium load [✓] High load [ ] LFSR

Cratis load selection
Note: When using Cratis as a load chip, only one VR can be tested at a time.
[ ] DES [ ] DES+AES (seq.) [ ] DES+AES (sim.)

Other test parameters
Number of glitches: 5 delay: 1000 ms
Test chip ID: TAR04
Output File prefix: 2008_VIII_24_15M
Output directory: :\engd\tartalo_test\
Glitch delay: 0 ns

Expected ATR
Warm: 3B020001
Cold: 3B020001
Reset Byte: 0

Status: Idle
Test progress

Glitch Parameters: [✓] Vcc 3v [ ] Vcc 5v

Glitch Amplitudes: 17 V
1.000000
2.000000
3.000000
4.000000
5.000000
6.000000
7.000000
8.000000
9.000000

Glitch Widths: 500 ns
10.000000
20.000000
30.000000
40.000000
50.000000
60.000000
70.000000
80.000000
90.000000

Noisy Supply Parameters: [ ] Noisy Vcc 3v [ ] Noisy Vcc 5v

Peak to Peak Voltage (Vpp): 0 V

Noise Period: 0 ns

Result Files

Figure 18 Silicon test front-end

Sixteen different glitch configurations were applied to the test board over a base supply power of 3V, all positive square glitches. Each glitch configuration was repeated 20 times whilst the counter was holding data (i.e. not counting) and whilst counting. The voltage regulator was subjected to a constant high load. This test was repeated with 4 different 01OKA test-chips. Test results whilst counting are shown in Table 1. Test results whilst holding data are shown in Table 2.

Table 1 Number of error injections whilst counting

Test-chip	**Glitch amplitude (V)**	**Glitch width (ns)**			
		20	**100**	**200**	**300**
oka01	17	0	2	2	2
	15	0	4	2	2
	12	0	0	0	1
	9	0	0	0	0
oka04	17	20	20	20	20
	15	20	20	20	20
	12	20	20	20	20
	9	20	20	20	20
oka07	17	0	2	0	2
	15	0	0	0	0
	12	0	0	0	0
	9	0	0	0	0
oka08	17	0	4	3	6
	15	0	2	3	3
	12	0	7	2	0
	9	0	0	0	0

The most common kind of error injection whilst counting was temporarily stopping the increment process. The oka04 showed an unusual amount of errors that could be explained by a faulty device or test setup. No glitch attack injected errors whilst holding data.

Table 2 Number of error injections whilst holding

Test-chip	Glitch amplitude (V)	Glitch width (ns)			
		20	100	200	300
oka01	17	0	0	0	0
	15	0	0	0	0
	12	0	0	0	0
	9	0	0	0	0
oka04	17	0	0	0	0
	15	0	0	0	0
	12	0	0	0	0
	9	0	0	0	0
oka07	17	0	0	0	0
	15	0	0	0	0
	12	0	0	0	0
	9	0	0	0	0
oka08	17	0	0	0	0
	15	0	0	0	0
	12	0	0	0	0
	9	0	0	0	0

5.1.2 Glitch attack simulations

For the simulation case, the latest simulation environment version was used, which enables higher accuracy simulations and solves the leakage simulation issue highlighted in the report SimEnvTech1.

Five different glitches were simulated:

- -1V amplitude and 100ns wide (not used with silicon test);
- -1V amplitude and 200ns wide (not used with silicon test);
- 12V amplitude and 20ns wide;
- 15V amplitude and 100ns wide; and
- 15V amplitude and 200ns wide

For each of these glitch configurations, the following simulations were run:

- Holding data
- Counting data:
  - When counter value was 0x03
  - When counter value was 0x10
  - When counter value was 0x3E

These simulations were run by applying a glitch to the Vcc pin of the validation setup shown in Figure 16 and by applying a glitch waveform to the Vdd pin of the validation setup. In the latter case, the applied Vdd waveform was a previously simulated response of the voltage regulator to the same or similar glitch.

For most cases, all simulations run fine. Only few simulations failed when a glitch was applied to Vcc due to a simulated malfunction of the voltage regulator's band-gap. The simulation results, Table 3, show that simulated glitch attacks have a similar response to silicon attack whilst holding a data. For glitch attacks whilst counting, the simulated attacks also produce a similar behaviour, indicating the need of a big pulse to inject any error to the test circuit in question. Other circuits might need less severe glitches to be affected by a glitch.

Table 3 Results of the glitch attack validation simulations

Glitch	Simulation type	Injected error
-1V, 100ns	holding	No error injection
	counting – 0x03	No error injection
	counting – 0x10	No error injection
	counting – 0x3E	No error injection
-1V, 200ns	holding	No error injection
	counting – 0x03	No error injection
	counting – 0x10	No error injection
	counting – 0x3E	No error injection
12V, 20ns	holding	No error injection
	counting – 0x03	No error injection
	counting – 0x10	Counter reset
	counting – 0x3E	Counter reset
15V, 100ns	holding	No error injection
	counting – 0x03	Count delayed by 1 clock cycle
	counting – 0x10	Count delayed by 1 clock cycle
	counting – 0x3E	Counter reset
15V, 200ns	holding	No error injection
	counting – 0x03	Count delayed by 2 clock cycles
	counting – 0x10	Count delayed by 2 clock cycles
	counting – 0x3E	Count delayed by 2 clock cycles

5.2 Power analysis validation

For validation purposes, two different DES modules were tested with the simulation environment:

- DES-A, a standard implementation of a DES module without countermeasures; and
- DES-B, a DES implementation with S-BOXes protected against power analysis.

Several power analysis simulations were run on both DES implementations. Common simulation parameters were:

- Full DES implementation
- HSPICE CMOS netlist
- No parasitic netlist
- 0.18um technology node SPICE model
- Target SBOX-1
- Two rounds simulated per power trace
- DES module fully reset between plaintext encryptions
- Force the registers' input nets
- Different load and run cycles

Simulation variables were:

- encryption key, randomly chosen
- simulation accuracy
- plaintexts, set manually and randomly

All power traces were analysed with DOM and CPA analysis methods, see Appendix C for the Matlab power analysis scripts. Table 4 shows the results for each simulation case and Figure 19 to Figure 32 show the graphical results of few of these power analyses.

Table 4 DES validation simulation parameters and results

Simulation ID	DES module	Key	Full key	Plaintexts	DES clock (MHz)	Simulation accuracy settings	Samples per trace	Simulation time	DOM	CPA
desa_64_1	DES-A	2E3A44BB4248D774	YES	64[1]	50	set_print_tres **0.001ns** set_sim_tres **0.001ns** set_ckt_subi **0.0001nA** set_sim_ires **0.0001nA** **set_sim_eou sim=2 model=2 net=2**	30000	13h12m37s	✓	✓
desa_64_2					10	set_print_tres **2ns** set_sim_tres **2ns** set_ckt_subi **0.1nA** set_sim_ires **0.1nA** **set_sim_eou sim=2 model=2 net=2**	75	26m43s	✗	✗
desa_64_3					50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA** **set_sim_eou sim=2 model=2 net=2**	3000	6h12m44s	✓	✓
desa_64_4					50	set_print_tres **0.4ns** set_sim_tres **0.4ns** set_ckt_subi **0.1nA** set_sim_ires **0.1nA** **set_sim_eou sim=2 model=2 net=2**	75	16m09s	✗	✗
desa_64_5		940AEBA0604CF479	YES	64[1]	50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA** **set_sim_eou sim=2 model=2 net=2**	3000	4h58m31s	✓	✓
desa_64_6		83373F5D2CF55BC8	YES	64[1]	50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA** **set_sim_eou sim=2 model=2 net=2**	3000	5h29m12s	✗	✗
desb_64_1	DES-B	2E3A44BB4248D774	YES	64[1]	50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA** **set_sim_eou sim=2 model=2 net=2**	3000	14h10m57s	✗	✗

Simulation ID	DES module	Key	Full key	Plaintexts	DES clock (MHz)	Simulation accuracy settings	Samples per trace	Simulation time	DOM	CPA
desb_256_1	DES-B	2E3A44BB4248D774	YES	256[2]	50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA**	3000	2d1h15m11s	✗[3]	✗[3]
desb_256_2				256[2]	50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA**	3000	2d1h11m49s		
desb_256_3				256[2]	50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA**	3000	2d2h53m44s		
desb_256_4				256[2]	50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA**	3000	2d1h28m57s		
desb_250_1		0000000040008010	NO	250[4]	50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA**	3000	2d2h32m17s	✗[5]	✗[5]
desb_256_2				250[4]	50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA**	3000	2d0h25m49s		
desb_256_3				250[4]	50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA**	3000	2d0h39m48s		
desb_256_4				250[4]	50	set_print_tres **0.01ns** set_sim_tres **0.01ns** set_ckt_subi **0.001nA** set_sim_ires **0.001nA**	3000	2d0h12m26s		

Note 1: Manually selected plaintexts to exercise only the target SBOX
Note 2: Randomly generated 1024 plaintexts split in four simulations
Note 3: Power analysis run over the 1024 power traces
Note 4: Randomly generated 1000 plaintexts split in four simulations
Note 5: Power trace analysis run over the 1000 power traces

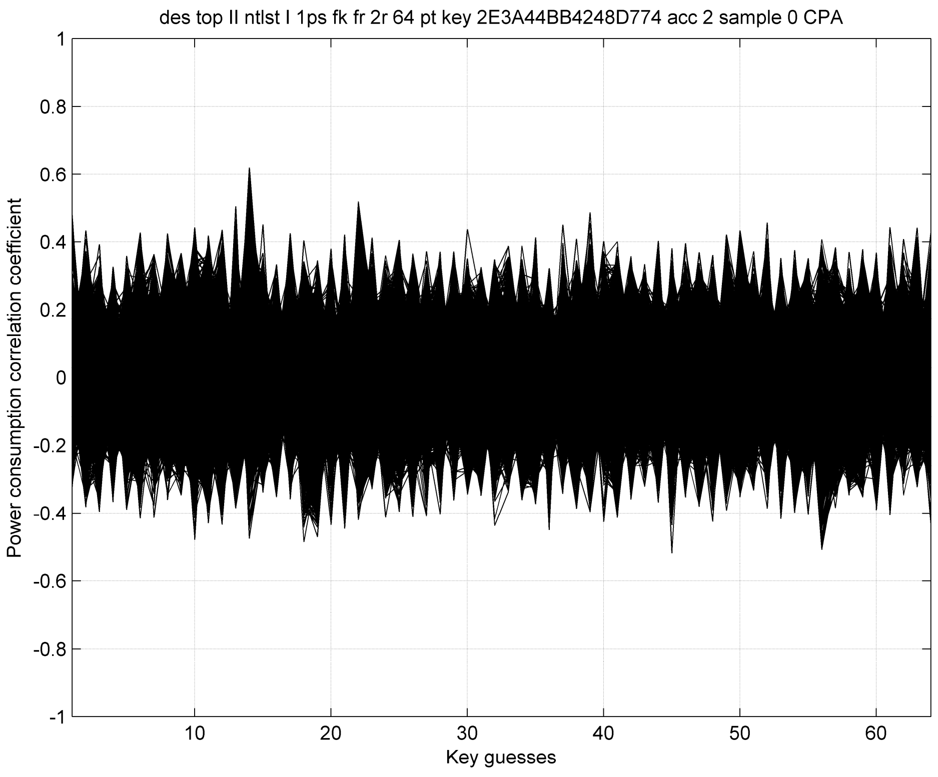


Figure 19 CPA results for simulation desa_64_1. Right key 14, correlation value 0.6204

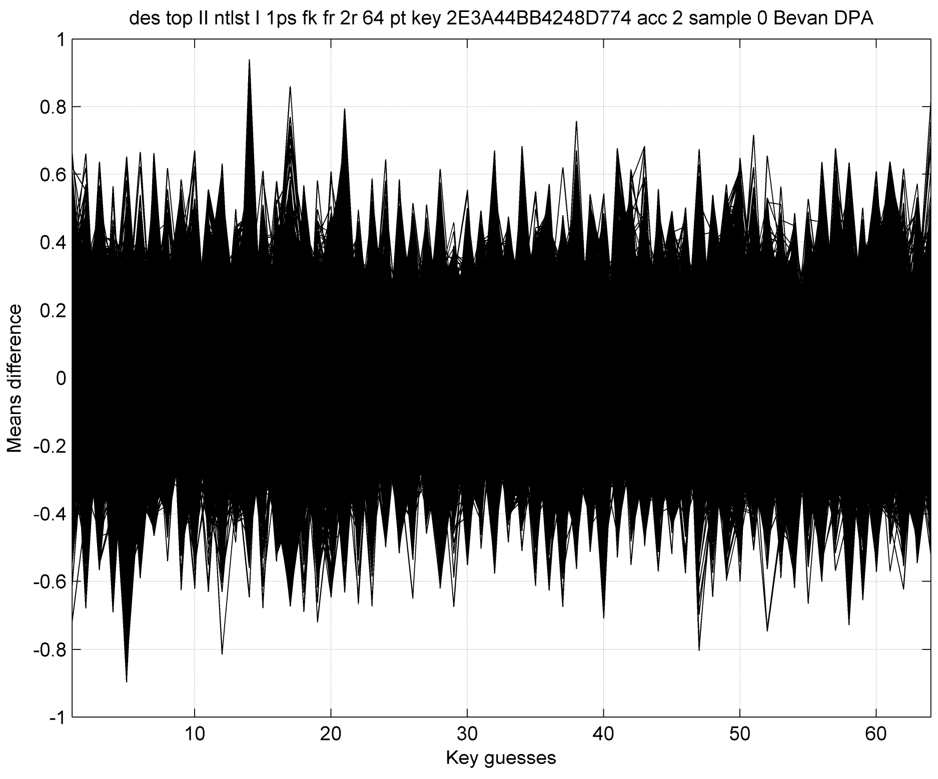


Figure 20 DPA results for simulation desa_64_1. Right key 14, correlation value 0.9404

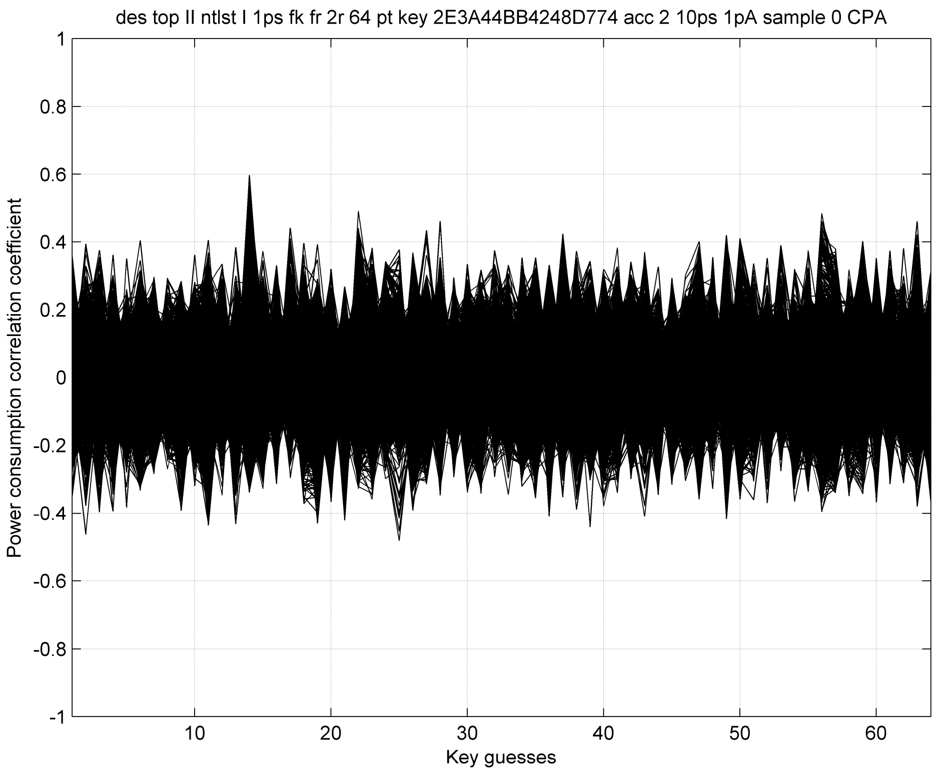


Figure 21 CPA results for simulation desa_64_3. Right key 14, correlation value 0.5972

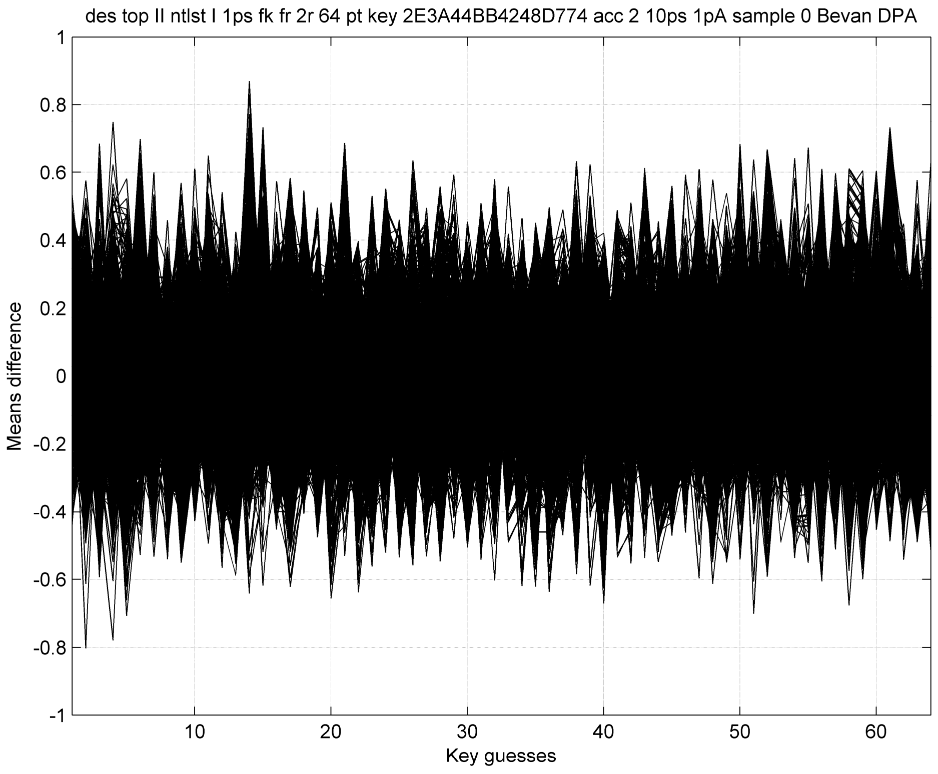


Figure 22 DPA results for simulation desa_64_3. Right key 14, correlation value 0.8695

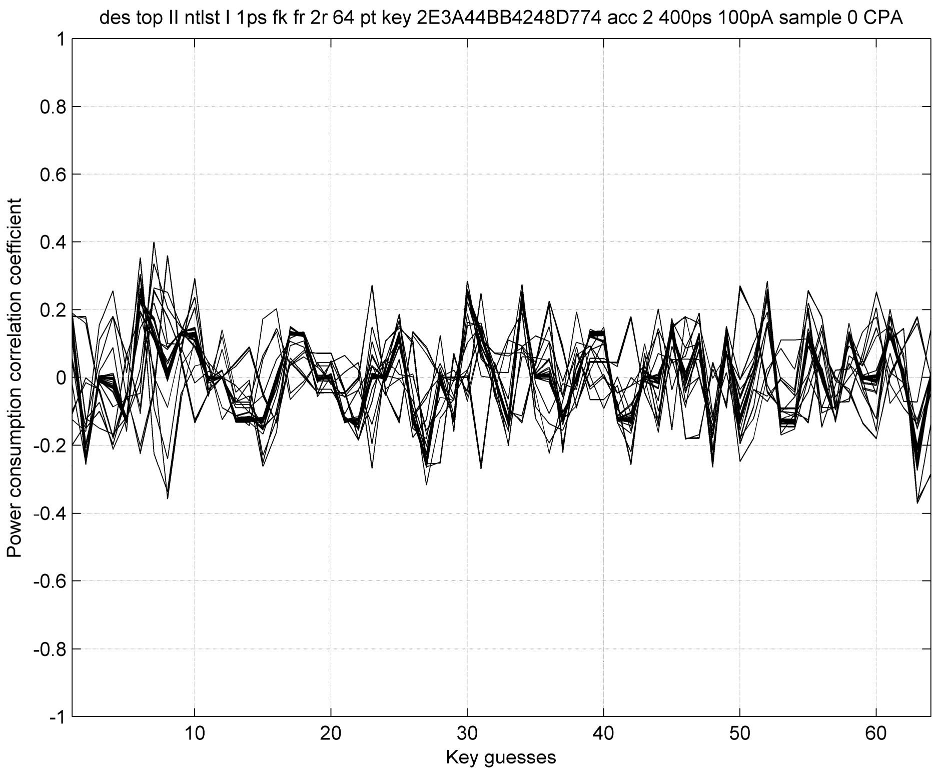


Figure 23 CPA results for simulation desa_64_4. Right key 14, correlation value 0.0901

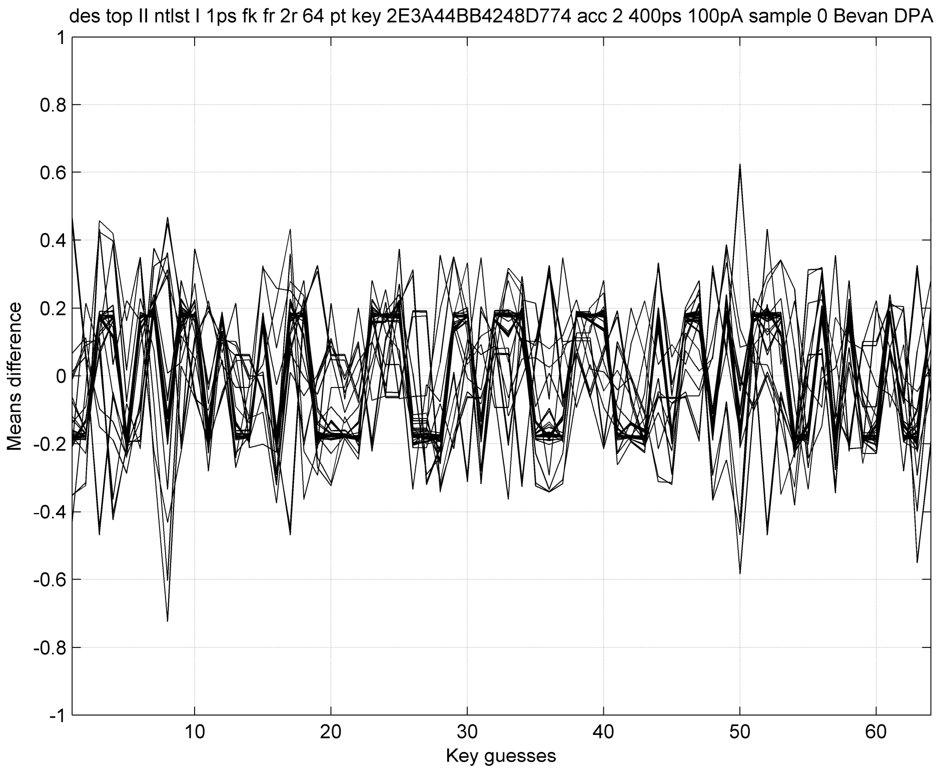


Figure 24 DPA results for simulation desa_64_4. Right key 14, correlation value -0.2112

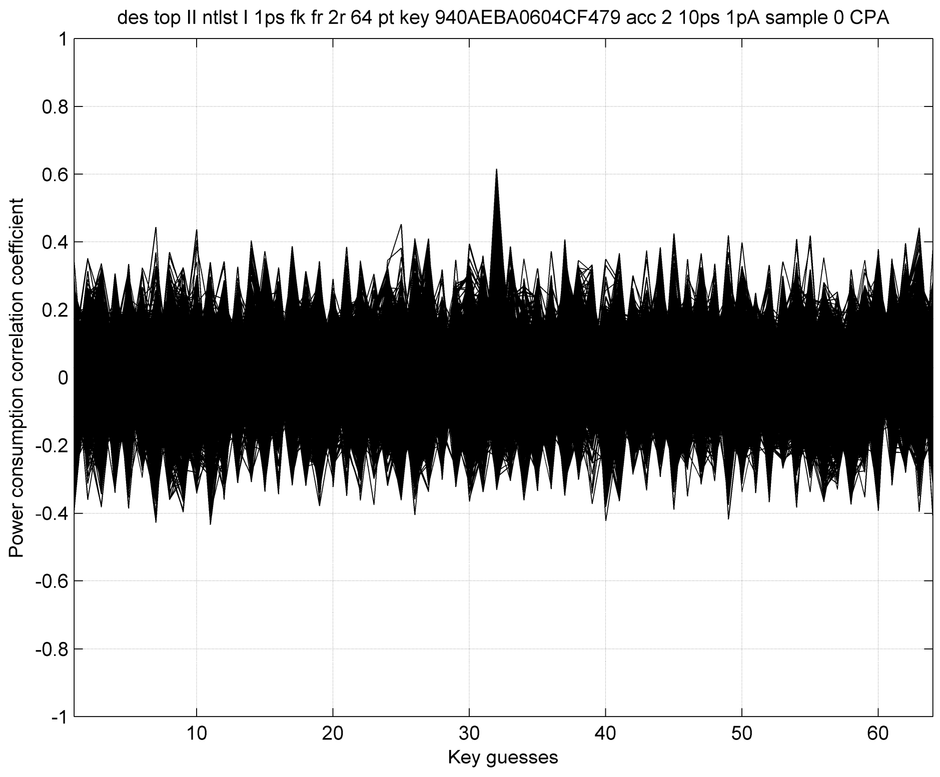


Figure 25 CPA results for simulation desa_64_5. Right key 32, correlation value 0.6168

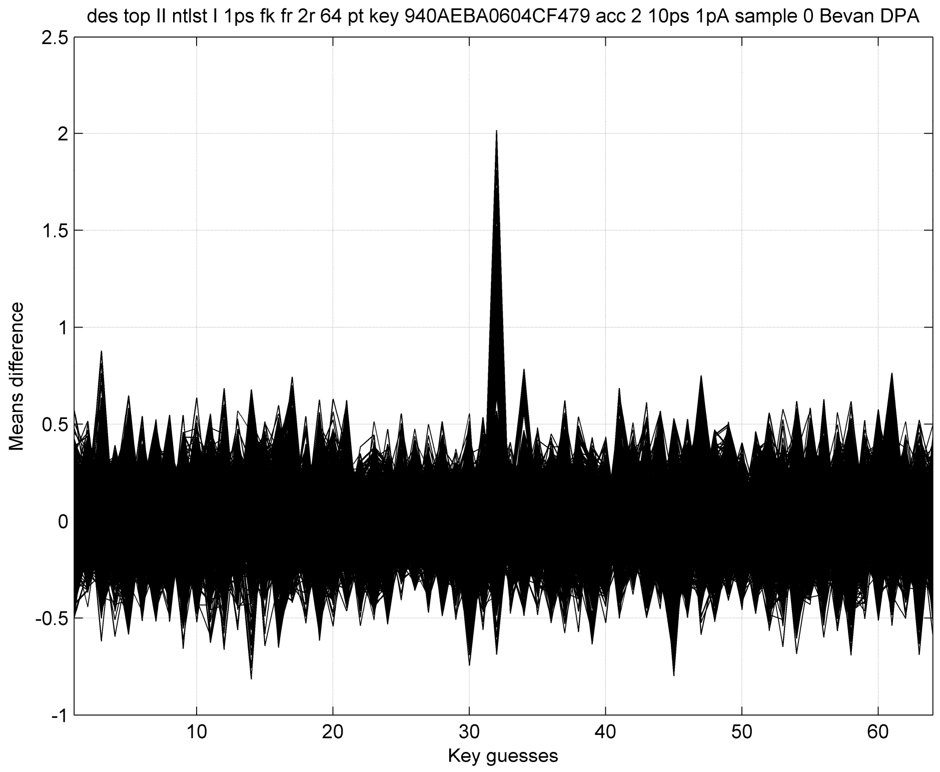


Figure 26 DPA results for simulation desa_64_5. Right key 32, correlation value 2.016

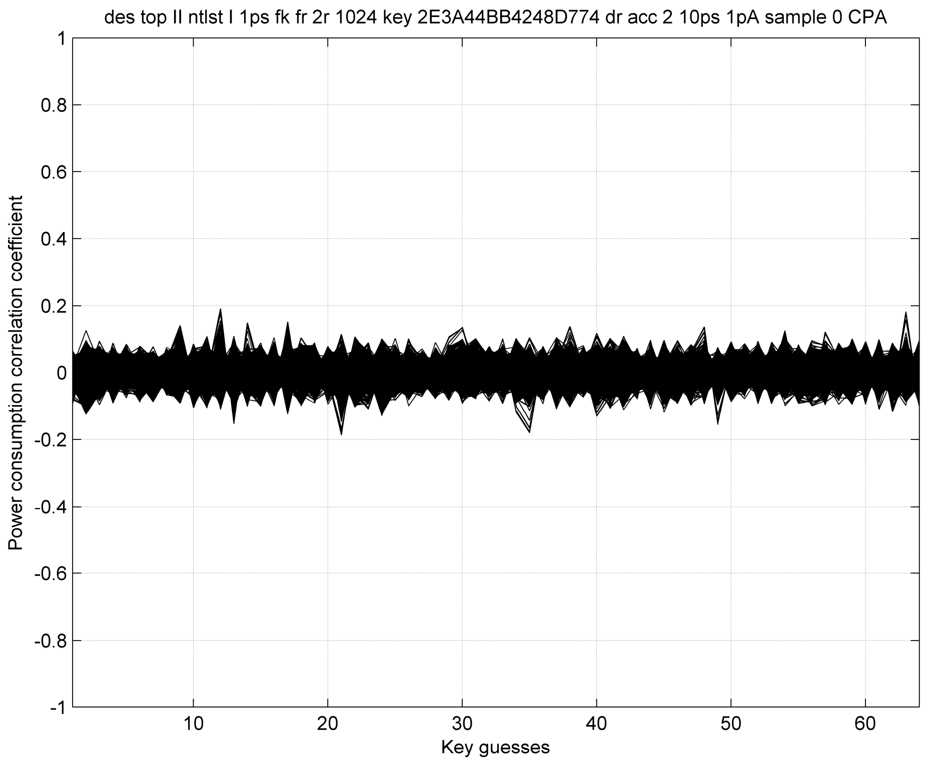


Figure 27 CPA results for simulations desb_256_1 to desb_256_4. Right key 14, correlation value 0.1484

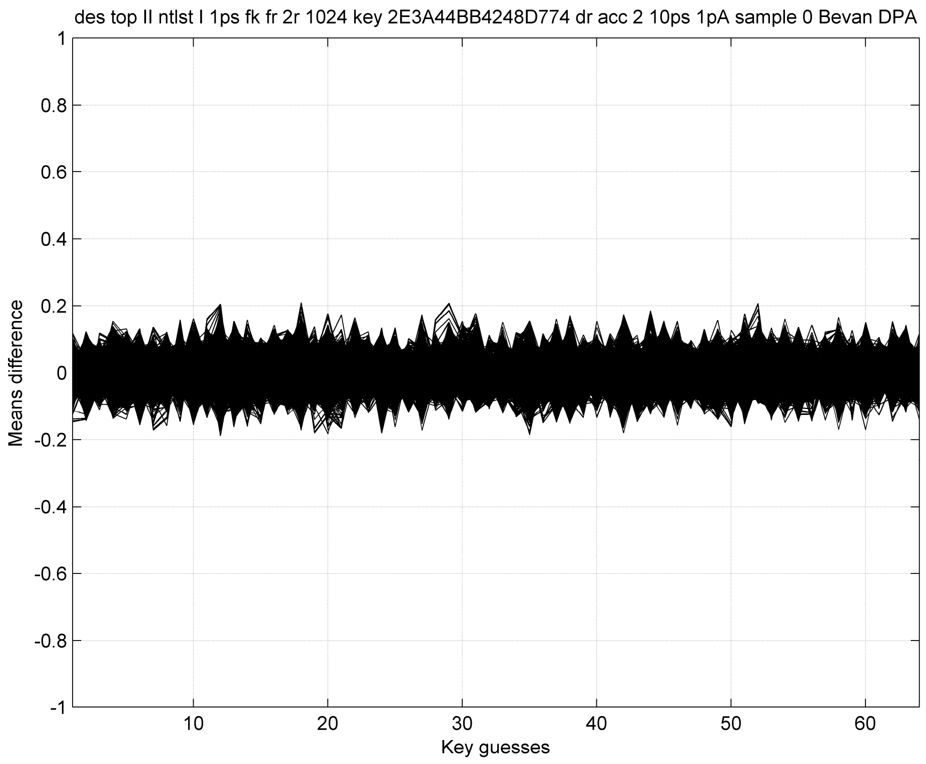


Figure 28 DPA results for simulations desb_256_1 to desb_256_4. Right key 14, correlation value 0.1567

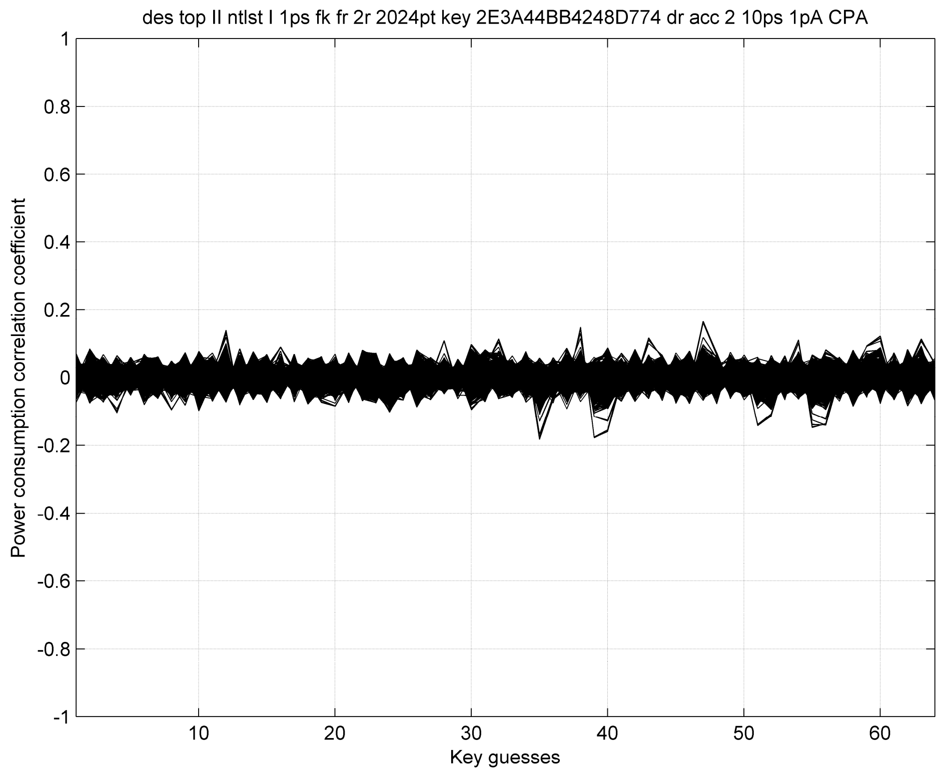


Figure 29 CPA results for simulations desb_256_1 to desb_250_4, 2048 plaintexts

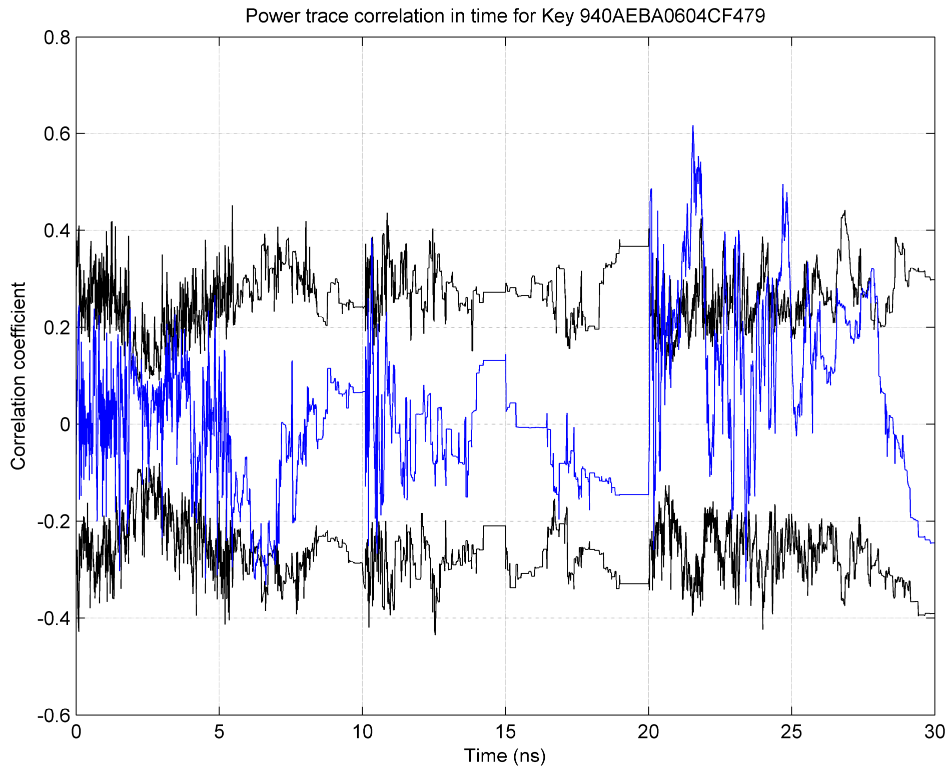


Figure 30 The right key's power trace correlation in time (blue) with the wrong keys' power trace correlation boundaries (black)

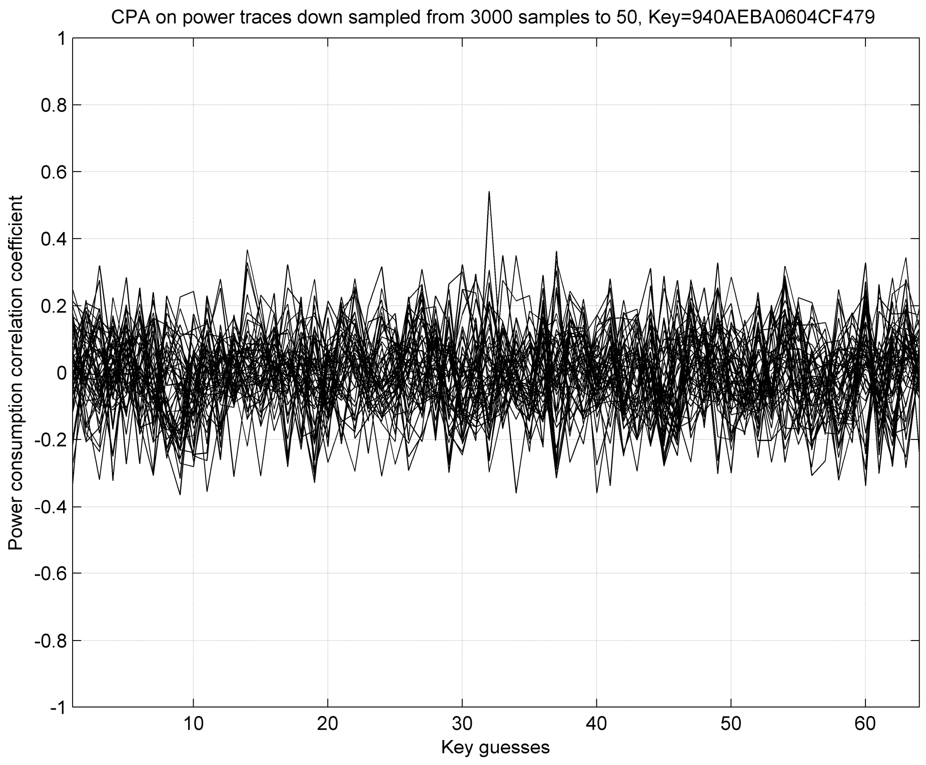


Figure 31 CPA results for simulation desa_64_5. Power traces down sampled from 3000 samples to 50 samples. Right key 32, correlation value 0.5418

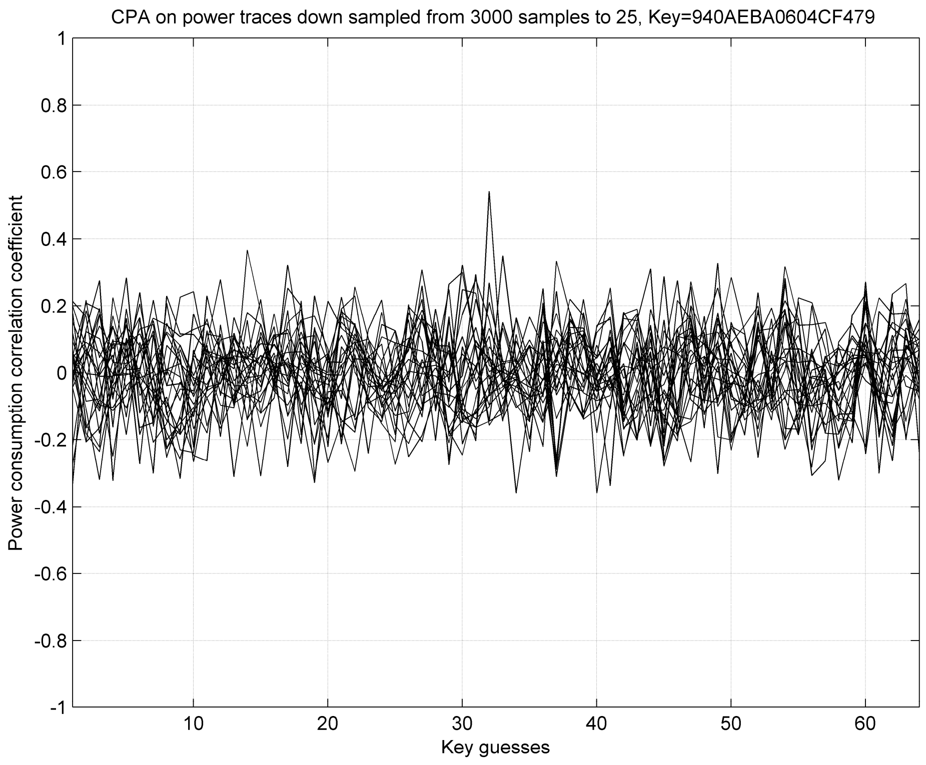


Figure 32 CPA results for simulation desa_64_5. Power traces down sampled from 3000 samples to 25 samples. Right key 32, correlation value 0.5418

5.3 Discussion on Validation Results

They key difference between the former, SimEnvTech2, and current glitch attack simulations is the simulation accuracy defined by GAPASE. Prior glitch attack simulations were run with lower simulation accuracy and on an older GAPASE version than the ones run during the validation process. Low simulation accuracy resulted on a poor model of the transistor's leakage behaviour and the need for glitches far more aggressive than real ones, to actually inject a fault.

With the current GAPASE, simulation results are more in tune with real glitch attacks. In fact, comparing them with silicon tests, simulated glitch attacks are shown to be even more sensitive, which could be in part due to the higher controllability on simulations. 15V glitches were capable of injecting similar errors with simulation and two of the silicon devices. One of the other two devices, oka07, withstood the 15V glitch attacks, whereas the other one, oka04, failed all tests, which could be due to test setup errors.

Silicon and simulation tests also proved that a glitch affects combinational blocks directly but not non-combinational or storage blocks. Furthermore, they also show that a clock's positive/negative-edges are the better point to apply these glitches.

GAPASE is also capable of simulating negative glitches and glitches on GND. These glitches were not tested with silicon devices due to setup limitations. However, by extrapolating the positive glitch results, a -1V glitch attacks on the Vcc is likely not inject any error.

Focusing on the simulations in more detail, simulating the voltage regulator's behaviour is as important as setting the right simulation accuracy. However, simulating the voltage regulator's response to a glitch attack is not only time consuming, but under certain circumstances simulations can also fail to converge. GAPASE works around these two issues by defining waveforms `vpower_waveform` and `vgnd_waveform` and applying them to Vdd! (instead of Vcc) and GND respectively. These waveforms might be a previously simulated Vdd! and/or GND response to a glitch. This feature allows reducing the simulation time from around 6 and a half minutes down to just over 1 minute (about 16.5% of the original simulation time). Bigger circuitry simulations could yield similar time savings, although this extreme has not been tested.

Regarding the DPA feature, GAPASE is a valid tool for guessing the encryption key of DES implementations with no countermeasures, a.k.a. vanilla implementations. The theoretical minimum 64 plaintexts were enough to guess the key bits associated to the SBOX1 on two cases (encryption keys 2E3A44BB4248D774 and 940AEBA0604CF479), but failed on a third case (encryption keys 83373F5D2CF55BC8). The failed case needs further research to understand why. One significant aspect of the later key is that the key bits associated to the SBOX1 in the first round are logic zero. This could be a particular case where the analysis fails, or just a coincidence.

Other reasons that could explain the failed guess might be related to:

- the need for more plaintext encryptions for these special cases. On such case, the recommended minimum amount of plaintexts would be higher than the theoretical 64 ones;
- the need for a higher simulation time resolution and/or higher SPICE model accuracy; or
- improvements needed on the power analysis algorithm.

Nanosim parameters '`set_sim_tres`' and '`set_print_tres`' define a simulation's time and print resolutions respectively, and the parameter '`set_sim_eou`' defines the Nanosim simulation engine's and SPICE model's accuracy. GAPASE sets the first two parameters to the same value; the later one is set to the default value. Setting it to a higher value (e.g. '`set_sim_eou sim=3 model=3 net=2`') would yield more accurate simulations but it would also take longer to simulate.

With the current accuracy setting, Nanosim is capable of running the encryption process with a simulation time resolution of 2ns (desa_64_2); however, this resolution is not enough to carry a successful power analysis with GAPASE. The same applies when defining a simulation time resolution of 0.4ns (desa_64_4).

Analysing the power traces of desa_64_3, computing a round takes around 5ns. With a simulation and print resolution of 1ps, 'desa_64_1' defined these critical 5ns with 5000 sample points. With a simulation and print resolution of 10ps, 'desa_64_3' defined 500 sample points. With a simulation and print resolution of 400ps, 'desa_64_4' defined 12 sample points. Finally, with a simulation and print resolution 2ns, 'desa_64_2' defined only 2 sample points. Only the former two simulations guessed the encryption key correctly.

Figure 33 shows the power traces of the simulations 'desa_64_2', 'desa_64_3' and 'desa_64_4'. While 'desa_64_3' yields power traces with enough differences to guess the encryption key, simulation 'desa_64_4' yields mostly identical power traces, except for few ones, and simulation 'desa_64_2' yields 64 identical power traces, this is the reason why the power analysis failed.

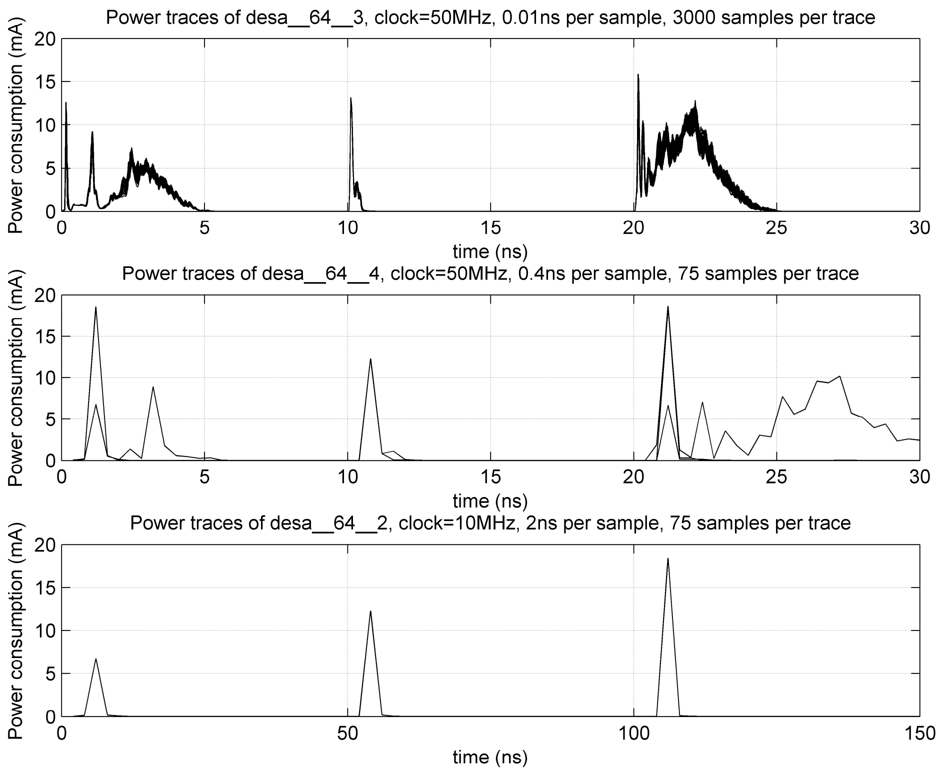


Figure 33 Power traces of desa_64_2, desa_64_3 and des1_64_4

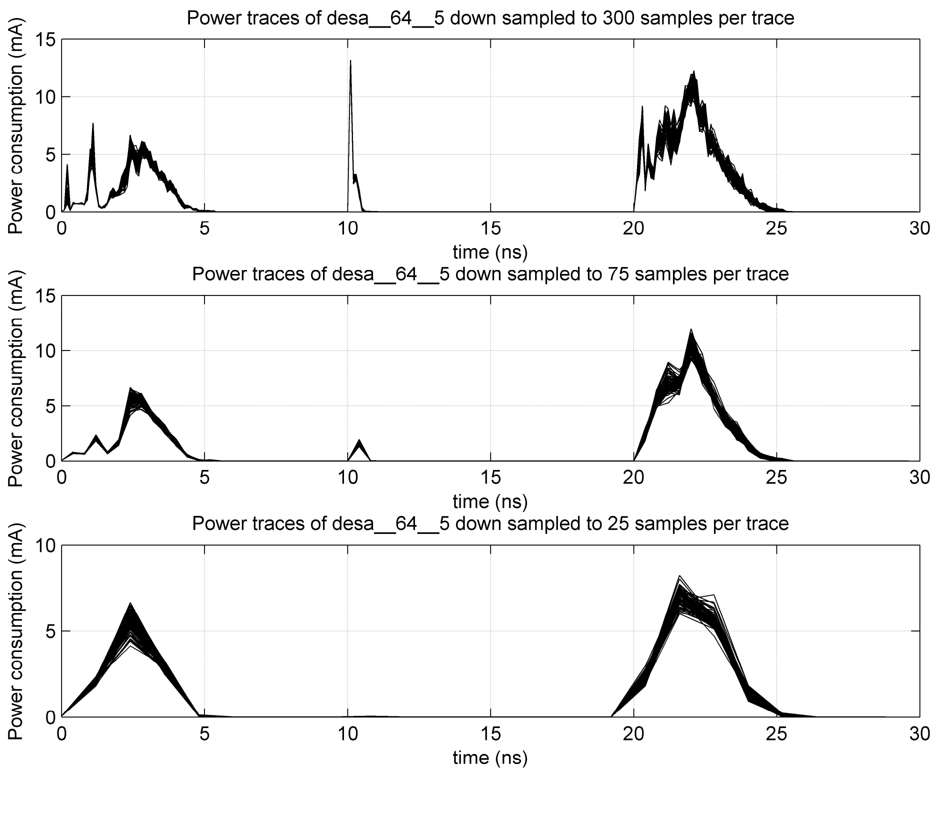


Figure 34 Power traces desa_64_5 down sampled to 300, 75 and 25 samples per trace

These power traces and power analysis results indicate that there is a minimum simulation time resolution to succeed with the power analysis, any resolution lower than that would minimise the chances of success. Although not tested, a simulation time resolution of 0.1ns or 0.05ns could be enough to achieve power analysis, yielding 50 and 100 samples respectively for the meaningful 5ns. Reducing the simulation time resolution from 10ps to 50ps or 100ps will improve on the simulation performance.

On the other hand, the power traces can be down sampled and yet guess the key correctly. This is the case shown in Figure 31 and Figure 32, where the power traces of the simulation 'desa_64_5' have been down sampled from 3000 samples to 50 and 25 samples respectively and yet the right key was guessed. Down sampling the power traces reduces the correlation coefficient of all key guesses, so down sampling should be used with care. Figure 34 shows the actual power traces when 'desa_64_5' is down sampled to 300, 50 and 25 samples. On each case, an encryption round is identified with 50, 8 and 4 samples, not far from the amount of samples defined by 'desa_64_2' and 'desa_64_4' yet with significantly different results.

This means that the simulation time resolution parameter, '`set_sim_tres`', has tighter restrictions than the print time resolution parameter, '`set_print_tres`', so that '`set_sim_tres`' > '`set_print_tres`'. Furthermore a lower '`set_print_tres`' implies fewer write accesses to the hard drive, and hence, less time to simulate.

As for the DES-B module, the power analysis was run over 64, 1000, 1024 and 2024 plaintexts without any success. Some reasons that could explain the failure could be:

- the need to simulate parasitic information with certain countermeasures, such as this one;
- the need for a higher amount of plaintexts; and or
- DES-B being secure against DPA/CPA analysis and requiring a different analysis algorithm or method, e.g. electromagnetic analysis (EMA).

Simulating the parasitic information decreases the simulation performance severely, plus it would need setting the parameter '`set_sim_eou`' to '`set_sim_eou sim=3 model=3 net=2`', which would further decrease the simulation's performance. Considering that simulating 1024 plaintexts took 2 days using 4 CPU cores, simulating the parasitic information might not be feasible. Simulating more plaintexts might also not be feasible, as simulating, say, 10000 plaintexts using 4 CPU cores could take around 20 days, a long wait in the design flow.

The practicality of a power analysis simulation tool is measured by the combination of simulation accuracy and the time required for performing the power analysis. So much so that it can be a key factor when determining its feasibility as a production power analysis tool.

Encrypting 64 plaintexts with a silicon implementation of a DES module running at 50MHz takes under a second, and running enough encryptions to guess the SBOX1 key bits of a vanilla DES would take few a minutes. Running the same operation on a DES module within a Smart Card takes longer due to the communication overhead and the noise generated by the additional logic switching.

Simulating the DES module in GAPASE does not have the Smart Card communication overhead. Still, simulating the first two rounds of 64 plaintext encryption took around 5 hours on a vanilla DES implementation (DES-A) and around 14 hours on a DES with a countermeasure (DES-B). This amount of time is excessive, as a successful power analysis on a DES with countermeasures needs the simulation of a higher number of plaintexts, which increases the required simulation time. Furthermore, the circuit complexity of some countermeasures could result in increasing the simulation time per round.

Here, the limitation factor of the simulation speed is given by the simulation tool, Nanosim, and/or its configuration. As previously discussed, one approach to improve the simulation's performance is reducing the simulation time resolution (`set_sim_tres`), setting the simulation print time (`set_print_tres`) to a lower number can improve it further. However, these changes are unlikely to be enough.

A recently published work that also uses Nanosim to perform power analysis on AES, [2], also uses high resolution settings for the simulation and current values. This work does not make any reference to the simulation time, although it can be deduced that simulation times are manageable. The key difference between the resolutions used by GAPASE and [2] is the resolution of the current consumption, as both define the same simulation time resolution, 10ps.

Where GAPASE defines a current resolution of 1pA, the simulation environment in [2] defines a 1uA, a 10^6 times less accurate unit. This clearly indicates that the current settings of the parameters '`set_ckt_subi`' and '`set_sim_ires`' are overkill for the performance of Nanosim. Reducing the current consumption resolution will boost the simulation performance and, with peak consumptions between 12 and 20mA and meaningful power consumption areas being around 5mA, a current consumption setting of 1uA would yield current consumption values of up to 4 digits, in principle enough to carry out successful power analysis. However, the gain in performance by GAPASE needs to be quantified to determine whether this measure is enough or further improvements are required.

GAPASE simulates the whole DES module, which is the easiest way to test several countermeasures such as dummy cycles. The downside of this approach is the time spent simulating circuit blocks that are not targeted by the power analysis. There are three other approaches to speed up the simulation process within Nanosim:

- Nanosim allows setting different simulation configuration parameters to different sub-circuits. Currently the whole design shares the same configuration parameters, setting lower Nanosim simulation engine accuracy and, if possible, simulation time resolution for the not targeted circuit blocks will improve the simulation performance.
- Nanosim is an RTL to transistor level simulation tool, allowing the co-simulation of RTL and SPICE netlist. Simulating part of the DES module as a SPICE netlist (e.g. SBOX1) and the rest of the design in RTL could dramatically shorten the simulation time, however, this feature requires an additional license. This license was not available when developing GAPASE; hence, it does not take advantage of this feature.
- Simulating only the targeted SBOX. This approach might be the faster one, but it is also the most limited one in terms of testable countermeasures and can set higher requirements on the test environment itself.

Another possible line of action could be using alternative simulation tools such as PrimePower, which is capable of generating dynamic and static power consumption waveforms of simulations at gate and RTL levels. PrimePower would result in faster simulation than Nanosim, although losing some accuracy, which could be balanced out with additional plaintext simulations.

Furthermore, simulating with PrimePower would avoid the current step of generating the SPICE netlist needed by GAPASE, making the flow simpler. Also, it would allow using simple RTL testbenches or even the ones used during the development process as a source of stimulus to generate the power traces for the power analysis.

6 Conclusion

GAPASE provides Atmel designers with means of testing their designs against two common attack techniques, glitch attacks and power analysis. The glitch attack feature is functional for small and medium size circuits, and simulation results can provide clues as to how a circuit responds to glitch attacks. Simulating big circuits could be time consuming.

Power analysis is possible with GAPASE for vanilla DES implementations but guessing the encryption key of DES implementations with countermeasures still might not be feasible. The power analysis feature needs improvements to boost the simulation performance to make it more usable. The main lines of action should focus on the simulation accuracy and co-simulating or using alternative simulation tools. These improvements and others are covered in the next section.

7 Future work

The simulation environment needs improvements to its power analysis feature in order to be usable within Atmel's design flow. The changes should focus on three main fronts:

- **Improve simulation performance:** As previously discussed, one option of improving the simulation speed could be setting lower current consumption simulation accuracy for the whole design. Further performance boost can also be achieved by lowering the simulation accuracy of non targeted circuit blocks. A higher performance improvement could be achieved through co-simulation or using other simulation tools such as PrimePower.
- **Add new features:** Two main feature implementations can be suggested. On the one hand extending the target SBOX from the current one, SBOX1, to any one. On the other hand enabling testing AES cryptographic algorithm implementations. This would make the whole simulation environment more useful to Atmel for their current and future needs.
- **Improve the post-simulation analysis step:** The analysis stage is currently run in Matlab, which is limited to a single computer and license. Integrating the analysis stage into the GAPASE script set would ease the designer's or tester's job. Alternatively, the post-simulation step could format the power traces to be readable and analysed by readily available third party power analysis tools such as RisCure^{TM}. This would allow running more complex analyses than currently available in GAPASE and avoid instantiating analyses already available on software tools licensed to Atmel.

No further work is planed for the glitch attack feature, although it could still benefit from the proposed work for the power analysis feature. For example, co-simulating RTL and SPICE netlists could enable studying how an error injected in the SPICE netlist can propagate to other parts of the design, as well as simulating bigger designs.

Appendix A

stimulis_nom.spi

```
* Define libraries
.lib './hl49at58.8kr400.mod' nonmatching
.lib './hl49at58.8kr400.mod' model_58K8
.lib './hl49at58.8kr400.mod' process_tolerances
.lib './hl49at58.8kr400.mod' techno
.lib './hl49at58.8kr400.mod' nonmc
.lib './hl49at58.8kr400.mod' mos_nom

* Include design (and design_rc if defined) netlist
.include './input/netlist/extracted/hspiceFinal'

* Include design stimulus file
.include './input/stimulus/var_stim.sp'

.end
```

var_stim.sp

```
* Define parameters
.param sup=1.6
.param temp=25

* Setup power supply
EP1 vdd! 0 poly(2) vpower_voltage 0 0 0 0 1 1
vgnd! gnd! 0 0

vvpower_voltage vpower_voltage 0 sup

ESV1 clkCpu 0 poly(2) vdd! 0 org_clkCpu 0 0 0 0 0 1

* Define input stimulus
V1 org_clkCpu 0 PWL (
+ 0.1n 0.00
+ 10n 0.00
+ 10.1n 1.00
+ 20n 1.00
+ 20.1n 0.00
+ 30n 0.00
+ 30.1n 1.00
+ 40n 1.00
+ 40.1n 0.00
+ 50n 0.00
+ 50.1n 1.00
...
+ 2470.1n 1.00
+ 2480n 1.00
+ 2480.1n 0.00
+ 2490n 0.00
+ 2490.1n 1.00
+ 2500n 1.00
+ 2500.1n 0.00
+ 2510n 0.00
+ 2510.1n 1.00
+ 2520n 1.00
+ 2520.1n 0.00
+ 2530n 0.00
+ 2530.1n 1.00
+ 2540n 1.00
+ 2540.1n 0.00
+ 2550n 0.00
+ 2550.1n 1.00
+ 2560n 1.00
+ )
V2 not_connected_vector_only_used_for_sampling_at_tres_time_units 0 PWL (
+ 0n 0.00
+ 0.01n 1.00
+ 0.02n 0.00
+ 0.03n 1.00
+ 0.04n 0.00
+ 0.05n 1.00
+ 0.06n 0.00
+ 0.07n 1.00
+ 0.08n 0.00
+ 0.09n 1.00
+ 0.1n 0.00
+ 0.11n 1.00
+ 0.12n 0.00
+ 0.13n 1.00
+ 0.14n 0.00
+ 0.15n 1.00
+ 0.16n 0.00
+ 0.17n 1.00
+ 0.18n 0.00
+ 0.19n 1.00
+ 0.2n 0.00
...
+ 2559.8n 0.00
+ 2559.81n 1.00
+ 2559.82n 0.00
+ 2559.83n 1.00
+ 2559.84n 0.00
+ 2559.85n 1.00
+ 2559.86n 0.00
+ 2559.87n 1.00
+ 2559.88n 0.00
+ 2559.89n 1.00
+ 2559.9n 0.00
+ 2559.91n 1.00
+ 2559.92n 0.00
+ 2559.93n 1.00
+ 2559.94n 0.00
+ 2559.95n 1.00
+ 2559.96n 0.00
+ 2559.97n 1.00
+ 2559.98n 0.00
+ 2559.99n 1.00
+ )
```

epic_pow.cfg

```
set_print_tres 0.001ns
set_sim_tres 0.001ns
set_ckt_subi 0.0001nA
set_sim_ires 0.0001nA
set_sim_eou sim=2 model=2 net=2

print_node_i vddv16!
print_node_v clkCpu
print_node_v xkeyschedule_call.desrun
print_node_v not_connected_vector_only_used_for_sampling_at_tres_time_units
```

```
; Force-release statements for the reset input signal
force_node_v v=0 ti=0.1n no=reset
rel_node_v no=reset ti=3840n
; Force-release statements for the data number 0
; -- Force-release statements for initialising the crypto block
force_node_v v=0.0000V ti=5n no=xcipher_call.xu87.net24
rel_node_v no=xcipher_call.xu87.net24 ti=5.1n
force_node_v v=1.6000V ti=5n no=xcipher_call.xu87.net27
rel_node_v no=xcipher_call.xu87.net27 ti=5.1n
force_node_v v=0.0000V ti=5n no=xcipher_call.xu87.net18
rel_node_v no=xcipher_call.xu87.net18 ti=5.1n
force_node_v v=1.6000V ti=5n no=xcipher_call.xu87.net26
rel_node_v no=xcipher_call.xu87.net26 ti=5.1n
force_node_v v=0.0000V ti=5n no=xcipher_call.xu85.net21
rel_node_v no=xcipher_call.xu85.net21 ti=5.1n
...
; -- 'xdesrun' force-release statements
force_node_v v=1.6 ti=25.2n no=xkeyschedule_call.n2470
force_node_v v=0 ti=25.2n no=xkeyschedule_call.n13
rel_node_v no=xkeyschedule_call.n2470 no=xkeyschedule_call.n13 ti=30.2n
; -- 'xdesreg' force-release statements
force_node_v v=0 ti=5.2n no=xcipher_call.n860
force_node_v v=0 ti=5.2n no=xcipher_call.n10
rel_node_v no=xcipher_call.n860 no=xcipher_call.n10 ti=10.1n
force_node_v v=0 ti=5.2n no=xcipher_call.n870
force_node_v v=0 ti=5.2n no=xcipher_call.n10
rel_node_v no=xcipher_call.n870 no=xcipher_call.n10 ti=10.1n
force_node_v v=0 ti=5.2n no=xcipher_call.n880
force_node_v v=0 ti=5.2n no=xcipher_call.n10
rel_node_v no=xcipher_call.n880 no=xcipher_call.n10 ti=10.1n
force_node_v v=0 ti=5.2n no=xcipher_call.n890
force_node_v v=0 ti=5.2n no=xcipher_call.n10
...
; -- 'xkeyreg' force-release statements. Target bits are 'SBOX_1'
force_node_v v=0 ti=5.2n no=xkeyschedule_call.n2070
force_node_v v=0 ti=5.2n no=xkeyschedule_call.n9
rel_node_v no=xkeyschedule_call.n2070 no=xkeyschedule_call.n9 ti=10.1n
force_node_v v=0 ti=5.2n no=xkeyschedule_call.n2080
force_node_v v=0 ti=5.2n no=xkeyschedule_call.n70
rel_node_v no=xkeyschedule_call.n2080 no=xkeyschedule_call.n70 ti=10.1n
force_node_v v=1.6 ti=5.2n no=xkeyschedule_call.n2090
force_node_v v=0 ti=5.2n no=xkeyschedule_call.n70
rel_node_v no=xkeyschedule_call.n2090 no=xkeyschedule_call.n70 ti=10.1n
...
; Force-release statements for the reset input signal
force_node_v v=0 ti=0.1n no=reset
rel_node_v no=reset ti=3840n
; Force-release statements for the data number 1
; -- Force-release statements for initialising the crypto block
force_node_v v=0.0000V ti=65n no=xcipher_call.xu87.net24
rel_node_v no=xcipher_call.xu87.net24 ti=65.1n
force_node_v v=1.6000V ti=65n no=xcipher_call.xu87.net27
rel_node_v no=xcipher_call.xu87.net27 ti=65.1n
force_node_v v=0.0000V ti=65n no=xcipher_call.xu87.net18
rel_node_v no=xcipher_call.xu87.net18 ti=65.1n
...
```

Appendix B

cpa_asier.m

```
function cpa_asier(steam)

filename1=[steam '.out'];

message=['Running CPA on ' filename1];
disp(message)
clear message;

disp('Reading trace file')
data=load(filename1);
data_mat=size(data);

num_plaintexts = data_mat(1);
num_samples    = data_mat(2);
data=data';

% Load D-function files
disp('Loading partition files')
dfnc=zeros(64,data_mat(1));
for i=1:64,
    dfncf=sprintf('Dfnc_cpa_sbox_1_k_%d.txt',i-1);
    dfnc(i,:)=dlmread(dfncf);
end

% For each key guess, correlate
correlation=zeros(64,data_mat(1));
disp('Calculating trace correlation')
for i=1:64,
    for j=1:num_samples,
        correlation_matrix= corrcoef(dfnc(i,:), data(j,:));
        key_guess(i,j) = correlation_matrix(1,2);
    end
end

% Plot and save key guess results
figure(1)
plot(key_guess)
grid on
xlabel('Key guesses')
ylabel('Power consumption correlation coefficient')
figure_title=regexprep(steam,'_',' ');
```

```
title([figure_title ' CPA'])
print('-f1','-dtiff','-r600',[steam '_correlation_guess.tiff'])

figure(2)
plot(key_guess')
grid on
xlabel('Time samples')
ylabel('Correlation coefficient')
title([figure_title ' CPA 2'])
print('-f2','-dtiff','-r600',[steam '_correlation_guess_2.tiff'])

save([steam '_key_guess.mat'], 'key_guess', '-mat');
disp('Storing results an Excel file')
xlswrite([steam '_correlation.xls'],key_guess')

disp('finished');
```

dpa_asier.m

```
function dpa_asier(steam)

filename1=[steam '.out'];

message=['Running DPA on ' filename1];
disp(message)
clear message;

disp('Reading trace file...')
tic;
data=load(filename1);
data_mat=size(data);

num_plaintexts = data_mat(1);
num_samples    = data_mat(2);

t=toc;
total_time=t;
message=sprintf('... trace loaded in %d seconds',t);
disp(message)
clear message;

% For each key
for i=1:64,
    message=sprintf('Generating key guess %d',i);
    disp(message)
    clear message;
    tic;

    numZero=zeros(1,64);
    numOnes=zeros(1,64);

    % Load the D function of the current key
    dfnc=zeros(1,num_plaintexts);
    dfncf=sprintf('Dfnc_dom_bit_1_sbox_1_k_%d.txt',i-1);
    dfnc(1,:)=dlmread(dfncf);

    % Group each input data's trace into zeros or ones
    for j=1:num_plaintexts,
        if(dfnc(1,j)==0)
            numZero(i)=numZero(i)+1;
            tempzeros(numZero(i),:)=data(j,:);
        else
            numOnes(i)=numOnes(i)+1;
            tempones(numOnes(i),:)=data(j,:);
        end
    end

    % Prepare the data for the DPA
    mean_ones = mean(tempones);
    mean_zeros = mean(tempzeros);
    % var (X,1) already normalises by N so don't do that again
here/numZero(i);
    varzero_norm=var(tempzeros,1);
    varones_norm=var(tempones,1);% see above comment /numOnes(i);
    normDPA=zeros(1,num_samples);
    BevanDPA=zeros(1,num_samples);
    GoubinDPA=zeros(1,num_samples);

    % Free some memory
    clear tempones;
    clear tempzeros;
    clear numOnes;
    clear numZero;

    % Do normal DPA
    normDPA = mean_ones - mean_zeros;

    % Do Bevan DPA
    temp_numerator=mean_ones-mean_zeros;
    temp_denominator=(varones_norm+varzero_norm).^0.5;
    BevanDPA=temp_numerator./temp_denominator;

    % Free some memory
    clear varzero_norm;
    clear varones_norm;
    clear temp_numerator;
    clear temp_denominator;
    clear mean_ones;
    clear mean_zeros;

    % Save current key guess results
    file_name=sprintf('%s_normDPA_%d.mat',steam,i);
    save(file_name, 'normDPA', '-mat');
    file_name=sprintf('%s_BevanDPA_%d.mat',steam,i);
    save(file_name, 'BevanDPA', '-mat');

    % Free some memory
    clear normDPA;
    clear BevanDPA;
    clear j;

    t=toc;
    total_time=total_time + t;
    message=sprintf('... key guess %d calculated in %d seconds',i,t);
    disp(message)
    clear message;

end



% Perform a statistical difference between the recently calculated waveforms
% and the general mean waveform. Store the results.
% Save and plot the results.

%normDPA
disp('Saving normal DPA data')
tic;

normDPA=zeros(64,data_mat(2));
for i=1:64,
    file_name=sprintf('%s_normDPA_%d.mat',steam,i)
    load(file_name);
    normDPA_final(i,:)=normDPA;
    clear normDPA;
```

```
end

figure(1)
plot(normDPA_final)
grid on
xlabel('Key guesses')
ylabel('Means difference')
figure_title=regexprep(steam,'_',' ');
title([figure_title ' Normal DPA'])
print('-f1','-dtiff','-r600',[steam '_DOM_normal_guess.tiff'])

figure(2)
plot(normDPA_final')
grid on
xlabel('Time samples')
ylabel('Normalised power consumption')
figure_title=regexprep(steam,'_',' ');
title([figure_title ' Normal DPA 2'])
print('-f2','-dtiff','-r600',[steam '_DOM_normal_guess_2.tiff'])

file_name=sprintf('%s_DOM_normal_guess.mat',steam)
save(file_name, 'normDPA_final', '-mat');
xlswrite([filename1 '_normDPA.xls'],normDPA_final')
clear normDPA_final;



%modifiedDPA
disp('Saving Bevan DPA data')

BevanDPA=zeros(64,data_mat(2));
for i=1:64,
    file_name=sprintf('%s_BevanDPA_%d.mat',steam,i)
    load(file_name);
    BevanDPA_final(i,:)=BevanDPA;
    clear BevanDPA;
end

figure(3)
plot(BevanDPA_final)
grid on
xlabel('Key guesses')
ylabel('Means difference')
figure_title=regexprep(steam,'_',' ');
title([figure_title ' Bevan DPA'])
print('-f3','-dtiff','-r600',[steam '_DOM_Bevan_guess.tiff'])
```

```
figure(3)
plot(-BevanDPA_final,'black')
grid on
xlabel('Key guesses')
ylabel('Means difference')
%axis([1 64 -1 1])
figure_title=regexprep(steam,'_',' ');
title([figure_title ' Bevan DPA'])
print('-f3','-dtiff','-r600',[steam '_DOM_Bevan_guess_bw.tiff'])

figure(4)
plot(BevanDPA_final')
grid on
xlabel('Time samples')
ylabel('Normalised power consumption')
figure_title=regexprep(steam,'_',' ');
title([figure_title ' Bevan DPA 2'])
print('-f4','-dtiff','-r600',[steam '_DOM_Bevan_guess_2.tiff'])

file_name=sprintf('%s_DOM_Bevan_guess.mat',steam)
save(file_name, 'BevanDPA_final', '-mat');
xlswrite([filename1 '_BevanDPA.xls'],BevanDPA_final')
clear BevanDPA_final;

t=toc;
total_time=total_time + t;
message=sprintf('Running DPA on %s took %d seconds',filename1,total_time);
disp(message)
clear message;

disp('finished');
```

References

1. Kocher, P., J. Jaffe, and B. Jun. *Differential Power Analysis*. in *CRYPTO*. 1999: LNCS 1666.
2. F. Regazzoni et al., “Evaluating Resistance of MCML Technology to Power Analysis Attacks” in *Transaction on Computational Science IV*, LNCS 5430, Springer-Verlag, pp. 230-243, 2009

Engineering Doctorate

Simulation results of pulses applied to a counter’s power signal

Author: Asier Goikoetxea Yanci

Date: January 2005

Sponsored by:

Engineering and Physical Sciences Research Council

Table of Contents

Chapter 1 – Introduction

An ASIC design flow covers several tests that need to be performed during a chip's design to make sure the final product's behaviour matches the RTL model's behaviour and that it meets timing, power and area constraints. This will provide developers with a high degree of confidence that the product's behaviour is correct under normal circumstances, or those circumstances taken into account when developing the product. These tests, however, usually do not provide any assurance of the design's behaviour under abnormal circumstances, whether they are intentional or unintentional (e.g. an attack or an external device's failure).

A product can be attacked in several ways. It can be attacked by altering voltage levels on the power source, it can be attacked by altering the clock frequency, it can be attacked by applying a laser to a cell/device to make it more leaky, taking the device to extreme temperatures, etc. Some can be more intrusive than others and some may require physical access to the silicon. Although some designs include additional circuitry to make sure that the design is working in a "safe" environment, if access to the silicon is gained these security measures may be circumvented, leaving the product more vulnerable to attack.

In order to test what could happen to a design when subjected to such attacks, a simulation environment has been constructed, which is capable of simulating power source voltage change and clock frequency change attacks.

This section of the report covers a simulation of a product's response to a power source voltage change attack (i.e. an attack that applies a pulse or glitch on the power line). The chosen design for this simulation is a simple counter shown in Figure 1, the Verilog code of which can be found in Appendix A. This design is simulated with Nanosim, which is provided with the design's netlist and a stimulus file to provide both a normal stimulus to design's inputs and to attack the design.

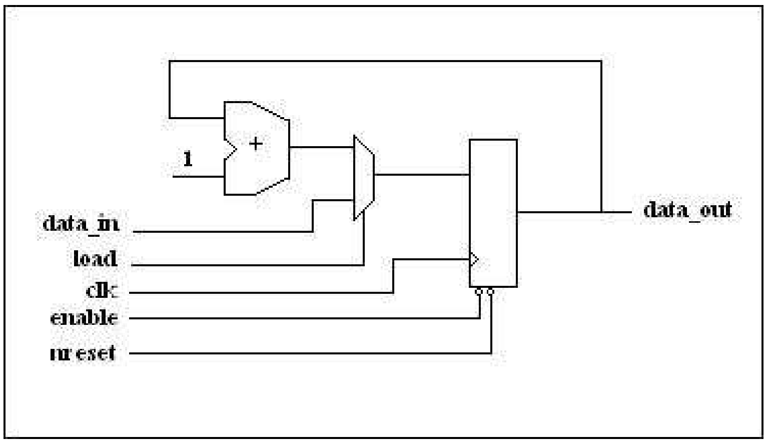


Figure 1 Counter

This document has been divided into four sections. This first section serves as an introduction to the rest of the document. Section 2 covers a design's expected response to pulses on its power line, what the considerations are when simulating a design's response and when pulses should be applied (i.e. when is a design more vulnerable to these attacks). Section 3 covers eight simulations of different pulses and the design's response to each of them, analysing what happens on the internal nodes of the design. Finally, Section 4 draws some conclusions from these simulations and proposes additional ones and future work.

Chapter 2 – Effects of a pulse

A design can be attacked using different pulses in different places of the design. They can be applied on a product's power pins or on the silicon's power lines. Applying the same pulse at different places in the circuit may produce a different response, as some designs have an embedded power regulator that would filter any power fluctuation. Figure 2 shows an example of pulses that can be applied with this simulation environment and Figure 3 shows an example of the differences between the pulse that is applied to the designs power pin and the power waveform that could be generated internally by an embedded regulator.

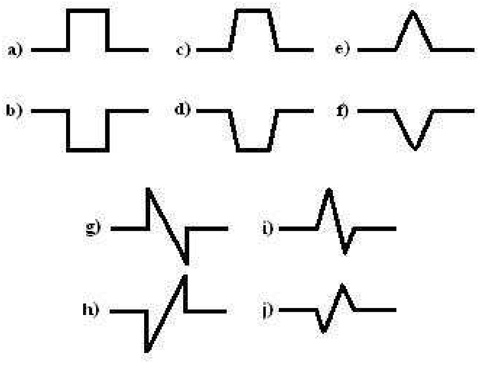


Figure 2 Example of applied pulses in the simulation environment

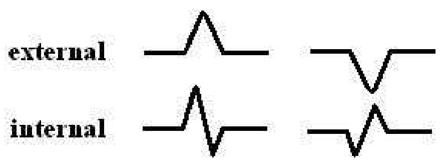


Figure 3 Example of resulting internal power waveforms to an external pulse

What is the expected behaviour/effect of applying a pulse on the power supply?

When a design is powered with a power supply that changes its voltage value, internal transistor and cell parameters may be affected by this. Some of these parameters are v_{oh}, v_{ol}, v_{ih} and v_{il}. Applying a pulse has exactly the same effect.

In an ideal design/silicon, (e.g. Figure 4), when there is a fluctuation on the power line, it is reflected identically all over the design immediately and with no delay, thereby changing all cells' v_{oh}, v_{ol}, v_{ih}

and v_{il} parameters at the same time. Hence, a pulse on an ideal circuit would have a minimum or no effect on a design's overall behaviour.

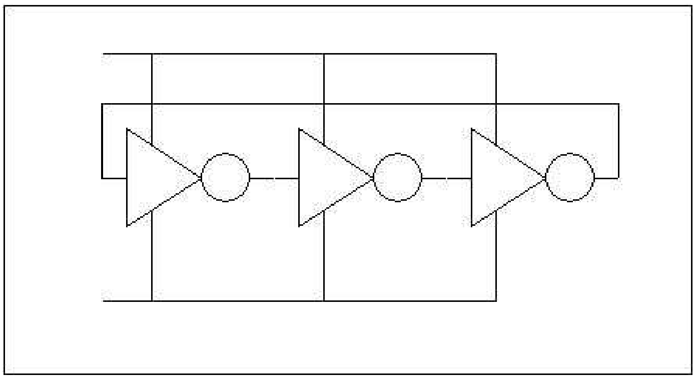


Figure 4 Example of ideal silicon

However, real design/silicon, Figure 5, has parasitic resistors and capacitors, not only on the signal lines, but also on the power lines. Because of this, a fluctuation on the power source, is not reflected identically on the whole design. In this case, the closer a cell is sitting to the power source, the more closely aligned the cell's power fluctuation will be to fluctuations on the power line.

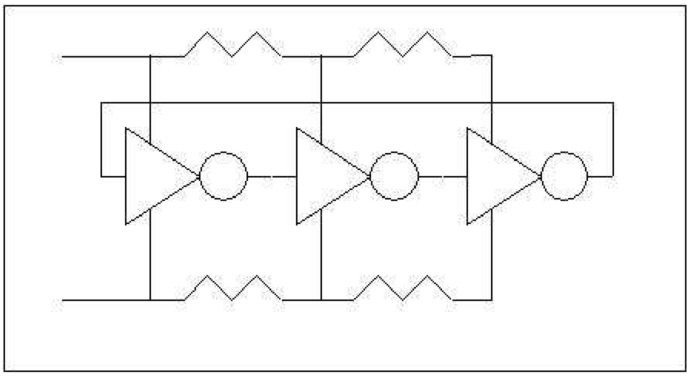


Figure 5 Example of ideal silicon

When the pulse has the optimum width and height, it can happen that, for an instant, two contiguous cells are being powered with different values. This would cause both devices to have different v_{oh}, v_{ol}, v_{ih} and v_{il} parameters. If the difference is large enough, it could result in a cell reading the wrong logic value from another cell's output (i.e., a logic 0 output could be interpreted as a logic 1 input and vice versa). This is the main target when applying an attack of this nature.

When should this attack be applied? When is it most effective?

Different designs can be more or less sensitive to certain attacks. Designs can be divided into at least three different groups: combinatorial design, non-combinatorial synchronous design, and non-combinatorial asynchronous design. A combinatorial circuit's output depends only on the circuit's inputs (e.g. an OR gate); it cannot remember what happened previously, it has no memory. A non-combinatorial circuit's output depends both on input values and the design's internal register values (e.g. a FSM); thus it remembers what happened previously. Registers represent the memory feature. Register values in a non-combinatorial design can be updated synchronously (with a synchronous clock) or asynchronously (without a clock).

A pulse applied to a combinatorial circuit's power supply could affect the circuit's output during the time the pulse is applied. When the pulse is finished, assuming the inputs are kept at the same logic values as before the pulse, as there is no memory effect, after a transitory time the circuit's output should revert to its original value. The transitory time would vary depending on the circuit's complexity and how far from the output the value switch happened. In this type of design, the time the pulse is applied is not important; the effect will be the same.

A non-combinational/hierarchical design is usually composed of several combinational blocks and registers. As explained above, the effects of a pulse applied to a combinational block are transitory. However, the effects of a pulse applied to a register may be different: the memory effect of the register may result in the output changing due to the glitch being latched, resulting in a permanent change rather than a temporary one. In this case a change/corruption of a register's value could affect the circuit's outputs and force it to behave differently to the way it would under normal circumstances.

As on a synchronous design, registers are updated at a clock's edge (positive and/or negative), a design can be more vulnerable to this attack when it happens just before or while the register is being updated. Additionally, on these designs, the clock tends to be periodic and deterministic, making it easier to find out when a pulse may be applied to have the best chance of disrupting the circuit's behaviour.

At the time of writing this document, asynchronous designs have not been analysed. As they are used much less frequently than synchronous designs their study may have less immediate relevance for us. There are different forms of asynchronous design style, each of which could be analysed, and which may have very different properties. It is possible that certain types of asynchronous design may be well suited to detecting glitch attacks that corrupt their circuit's performance as these circuits may lock up. However, detailed analysis of the lock up conditions would need to be undertaken to assess the protection these circuits may provide. This is for a later study.

Chapter 3 – Simulation results

In order to simulate a counter's behaviour to a pulse attack, it was necessary to create a circuit netlist. This netlist has been edited to add a resistor in parallel to each transistor, so transistor leakage can be simulated. However, power line parasitics have not been added, as the addition of these parasitic values would result in a very slow simulation (although more precise). The next two sections describe the pulse used in these simulations and an analysis of the simulations' outputs.

Pulses used for the simulation

Eight different simulations have been performed, where all applied pulses had waveform i) shown in Figure 2. The simulations start at different times, have different lengths and reach different high and/or low voltage values.

This waveform has been chosen over other waveforms because it matches a voltage regulator's possible output waveform when it suffers a pulse on its input. Additionally, both initial voltage rise and subsequent voltage fall can cause the desired register's value to switch. All pulses have been referenced to simulation time 140ns. Pulses start after an offset time from this.

Before starting to analyse the circuit's response to different pulses, it should be noted that, due to design model inaccuracies (not modelling power lines' parasitic resistor and capacitors), pulses used in this simulation are more severe than those applied on real silicon. If a silicon device suffers any of the pulses used in this simulation, it may be permanently damaged.

Simulation outputs

This section shows the circuit's response to each pulse and analyses it. For those cases where the circuit is affected, a deeper understanding is achieved by going down a design level. First, the circuit's normal behaviour is shown, followed by the circuit's response to each pulse.

Normal behaviour

In order to find out when the circuit misbehaves, its response to different pulses should be compared against the design's normal behaviour, which is shown in Figure 6.

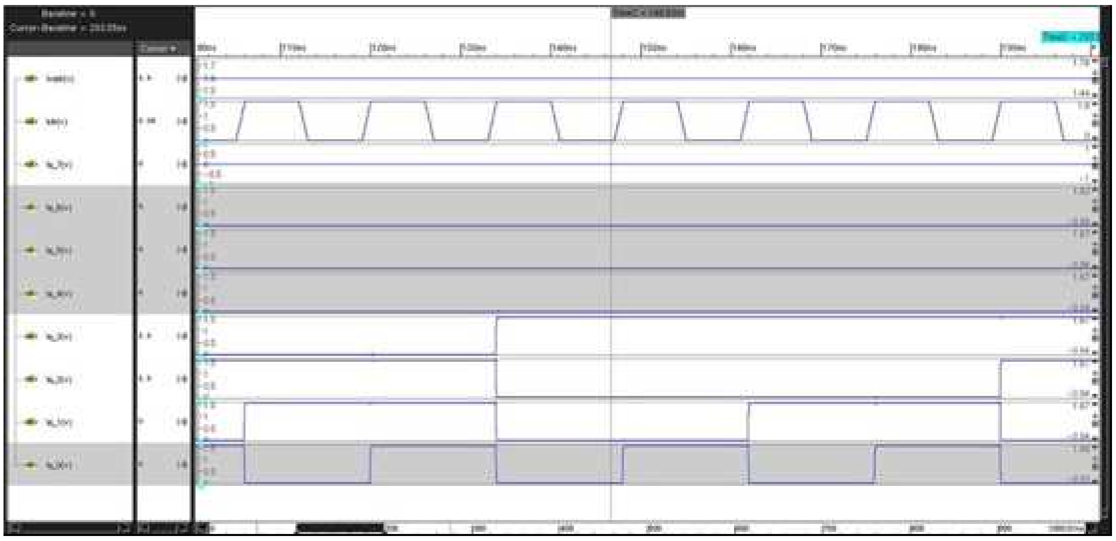


Figure 6 The design's normal behaviour

At 140ns, the circuit's output is 0x08. At 150ns, the circuit's output becomes 0x09 and by 170ns it is 0x0A. These are the output values that should be present if the circuit is operating correctly.

Pulse 1

At 140ns, this pulse starts to rise from 1.6v to 6.4v, which is reached at 142ns. It falls down to -3.2v at 149ns, rising to 1.6V at 154ns. Figure 7 depicts the counter's response to this pulse. These are the parameters to create this pulse:

```
VOLTAGE=1.6
DELAY_FOR_PULSE=140.0n
PULSE_START_TIME=0n
PULSE_START_VALUE=0
PULSE_P1_TIME=2n
PULSE_P1_VALUE=3
PULSE_P2_TIME=9n
PULSE_P2_VALUE=-3
PULSE_END_TIME=14n
PULSE_END_VALUE=0
```

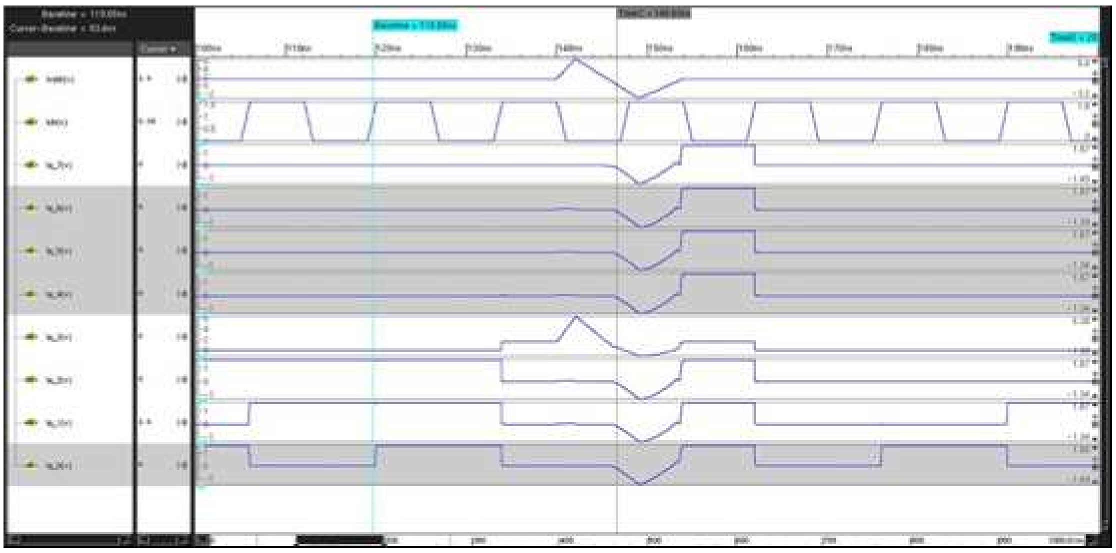


Figure 7 Design response to Pulse 1

Comparing Figure 6 and Figure 7, it is obvious that this pulse affects the circuit's behaviour; in fact, this pulse sets all outputs to high. At 135ns, the circuit's output is 0x08; at 150ns, when VDD is -2.22V, is should be 0x09 but instead, the real voltage at the outputs is between -1.02V and -1.24V. At 152ns, just after the pulse, the circuit's output is set to 0xFF, instead of 0x09. Finally, at 165ns, when the design's output should be 0x0A, the real output is 0x00.

If this circuit works as an iteration counter, indicating the end of iterations when it reaches 0xFF with an OR gate, at about 152ns it would indicate the end of iterations. This corruption could have consequences for other circuits depending on this value or for the overall system of which this circuit is a part.

Analysis

All outputs have been set to high, no matter what their previous state. Let's understand why. Q7, which was low before the pulse, is set to high after the pulse. Figure 8 shows the register Q7, and another cell whose output is the Q7 register's input. This structure applies to registers on this design.

Net N30 is the data that is stored into register Q7 on a clock positive edge if the register is enabled. N30 is the result of an OR operation of, the AND between data_7 (a1) and n13 (a2), and the AND between N42 (b1) and n12 (b2). As the design is counting and not loading a number, n13 is 0, so data_7 AND n13 results in 0. Therefore, N30 will be the result of N42 AND n12, as shown in Figure 9.

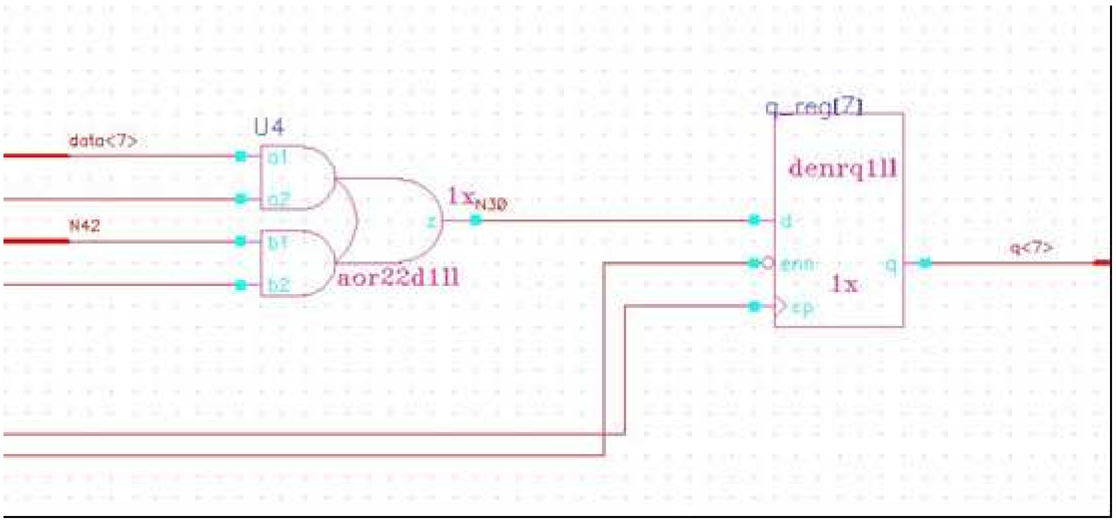


Figure 8 Structure of Q7 and input cell

n12 is the result of a combinational block, which is shown in Figure 19, page 80. As shown in Figure 9, as soon as the pulse has finished, n12 recovers its original value. N42, which is also the result of a combinational block, however, does not recover its original value until nearly 1ns after the pulse is finished. This is because N42 is the MSB of a combinational block, whose inputs are other registers' outputs. As registers' outputs have changed, it makes sense that N42 is also different. In this particular case, when VDD is recovered, all registers are set to logic 1. As the combinational block to which N42 is MSB output is an incremental counter, 1ns is the time that is needed to calculate 0xFF + 1. If the registers' output was set to 0xFE, N42 would not become logic 0 until the next clock cycle. Let's see what happens inside the register then.

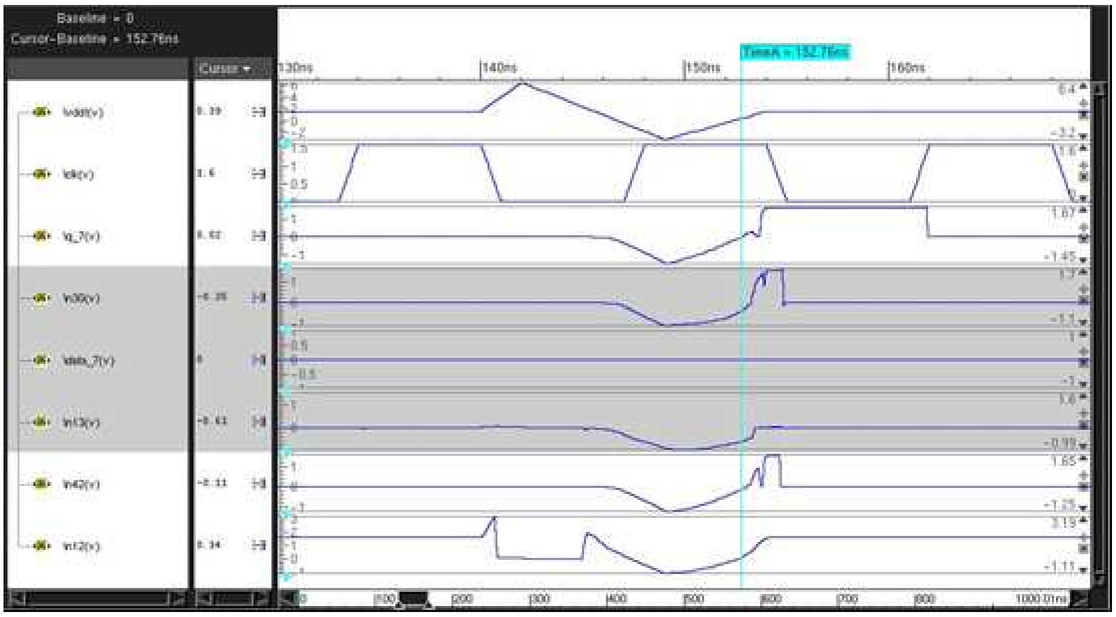


Figure 9 Signals of Q7's input combinational cell

The register used in this design has three inputs (data, enable and clock) and one output (q). Figure 10 shows the register's internal circuitry and Figure 11 shows the register's behaviour when the pulse is applied.

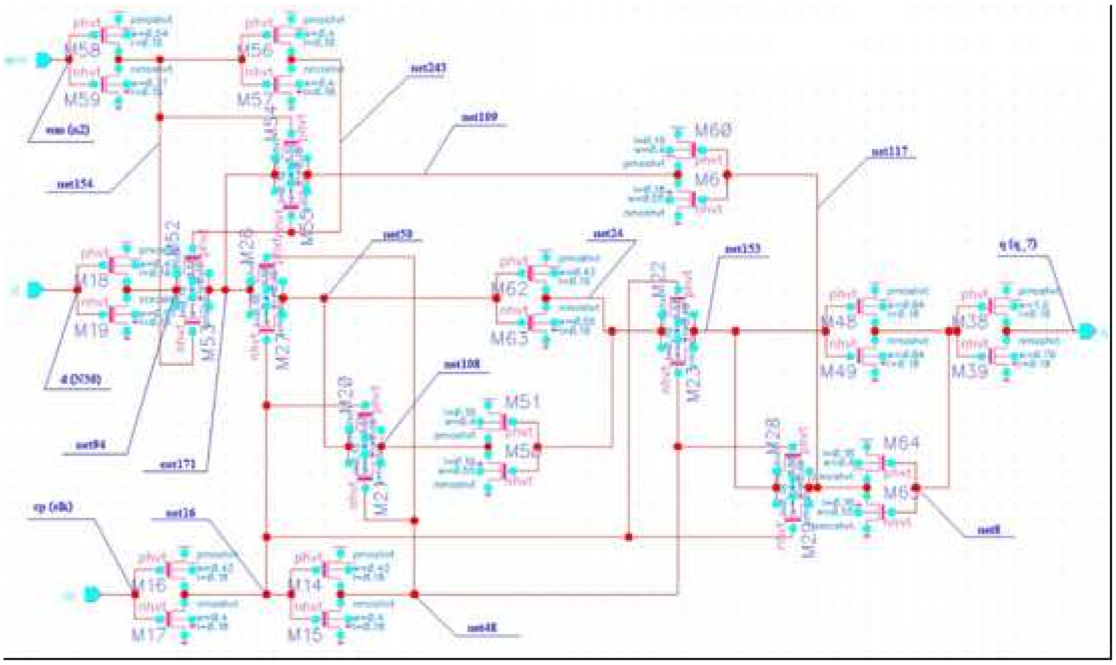


Figure 10 Register internal schematic

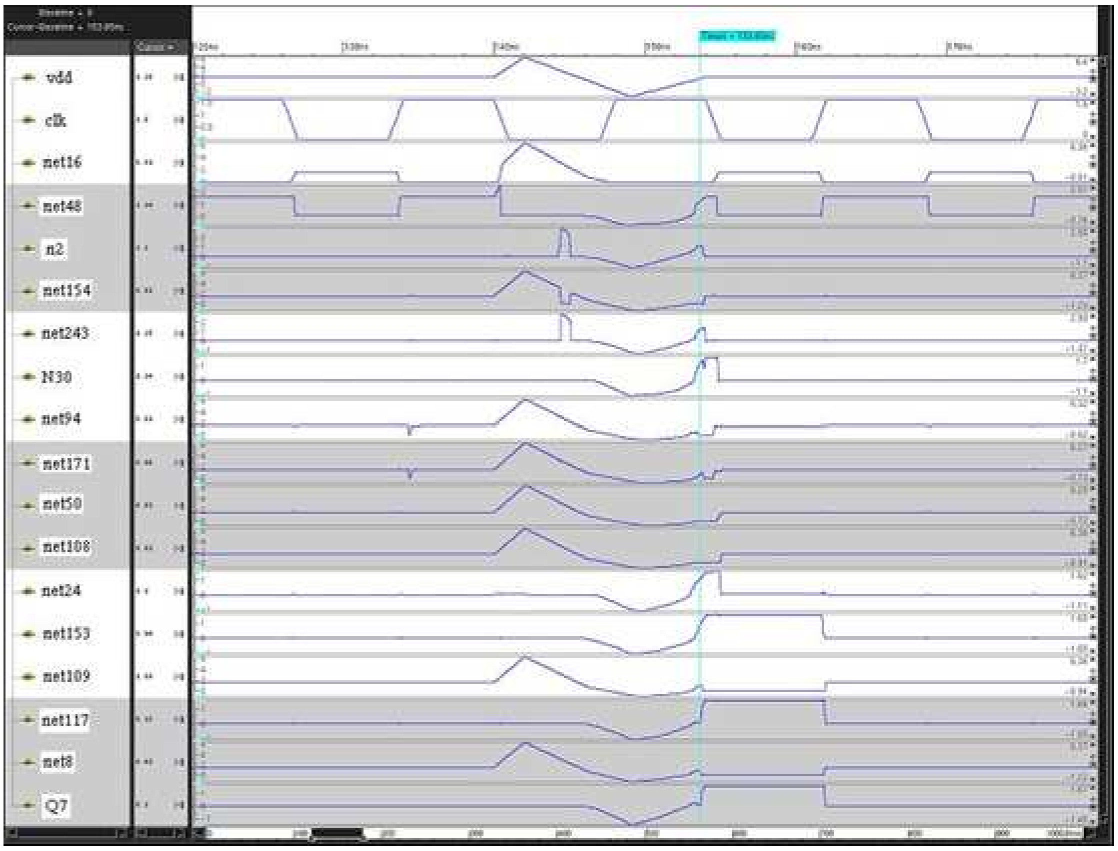


Figure 11 Register's internal signals

The first impression looking at Figure 11, is that when a pulse is applied, the voltage changes on all nets at logic 1, according to voltage variations on VDD; and that the voltage on all logic 0 nets remain unchanged until VDD goes below 0.8V. At this point logic 0 nets also follow VDD's change trend. When VDD reaches -0.8V, all nets stop following VDD and start to change in a way which results in setting the register's output to high. In addition to this, net n2 shows singularities, which affect nets net154 and net243 and may also affect other internal nets. Further analysis shows a late clock positive-edge detection (net48 going high while net16 stays low when VDD goes over the threshold voltage).

Checking on the register's behaviour when the n2 input goes high, it shows that it connects net171 with net109, disconnecting net94 from net171 (i.e. holding the register's value instead of updating it with the value present at its input). In this case, both data, N30, and output, Q7, values are logic 0, so this singularity has no effect on the register's behaviour. This can be probed by checking nets net94, net109 and net171. They all show a very similar waveform when this singularity happens and always following changes on VDD. If net94 and net109 have different logic values, net171 will change its logic value for as long as the first singularity on net n2 is present. Therefore, other internal nets will also be affected, as shown in Figure 12.

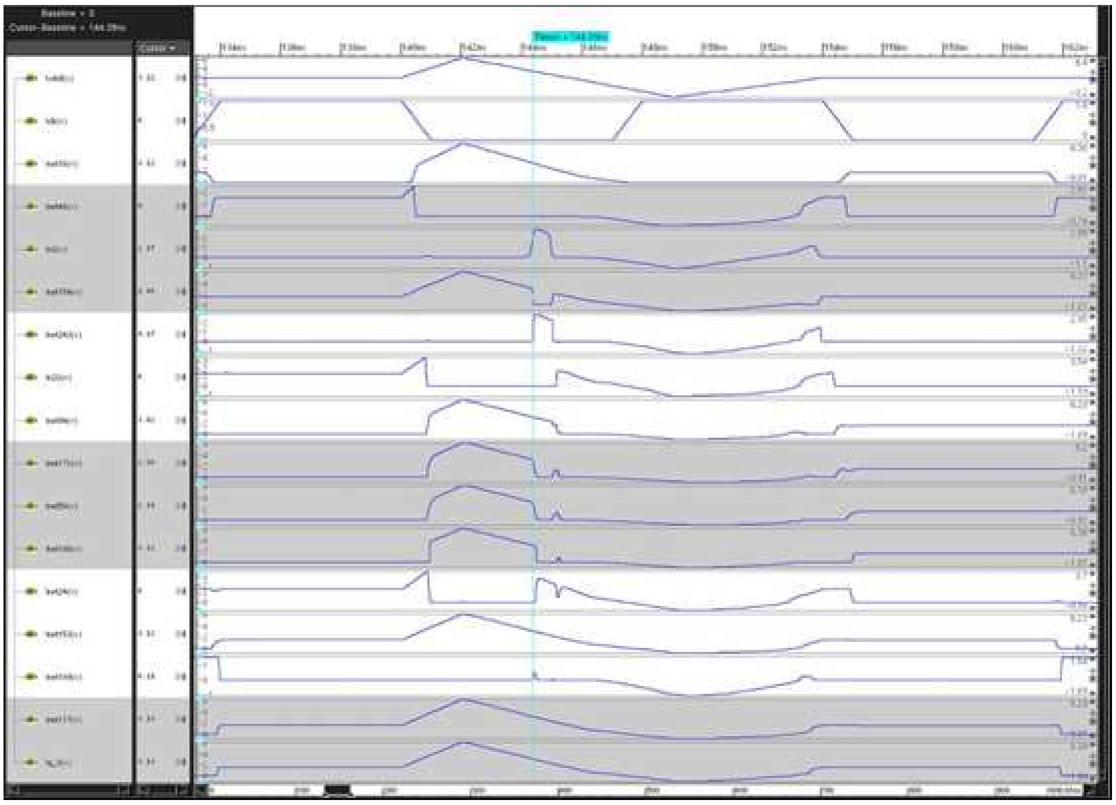


Figure 12 Register Q3 internal nets' voltage values

Continuing with Figure 11 and as stated before, all nets follow VDD's trend between the points where VDD is 0.8V (@146.14ns) and about -0.75V (@151.62ns). During these points, all nets' voltages decrease, initially, due to the leakage of the transistors. When the voltage difference between VDD and different nets' voltages are over a threshold voltage, PMOS transistors start to be polarised and start to conduct, setting the nets' voltage values to negative.

When VDD rises over -0.75v until it reaches 0.39V, all nets' voltages values gradually go to 0v, still following VDD's trend. It is at this point when n2's second singularity starts to happen and the Q7 register's internal nets start to evolve in a different manner to the trend they have shown until now. Nets net94, net24 and net8 are the fastest ones to rise.

When VDD rises, nets net94, net24, net8 and Q3 show a tendency to rise faster than other nets. At about VDD=0.72V, the register's internal circuitry starts to look like Figure 13 (Figure 14 is an equivalent circuit for Block A and B). When VDD rises to 0.88V, a late clock positive edge is starting to be detected and a second singularity on n2 continues to raise its voltage, although internally it happens more slowly. At this time N30 is starting to rise and to induce a change on net94, whose tendency was to rise. Net24 has just become 0.8V, which forces net108 to stay at logic 0, and net50 remains logic 0, which does not affect net24's logical value. Net8 is also rising and starting to influence Q7's voltage value.

By the time VDD reaches 0.94V, the register's internal circuitry looks like Figure 15. At this point net50 and net108 are still logic 0, and net24 still is logic 1. net153 is approximately 0.21V and net8 has just become logic 1.

As it can be seen on Figure 15, net24 depends solely on net50 and net108, which are both logic 0. This enables a stable logic value at net24. Because of this, net24 induces a logic value change on net8, from logic 1 to logic 0, and Q7 starts to be set to a logic 1 again.

This analysis can be applied qually to all registers in this simulation. Therefore, there is no need to analyse all individual cases.

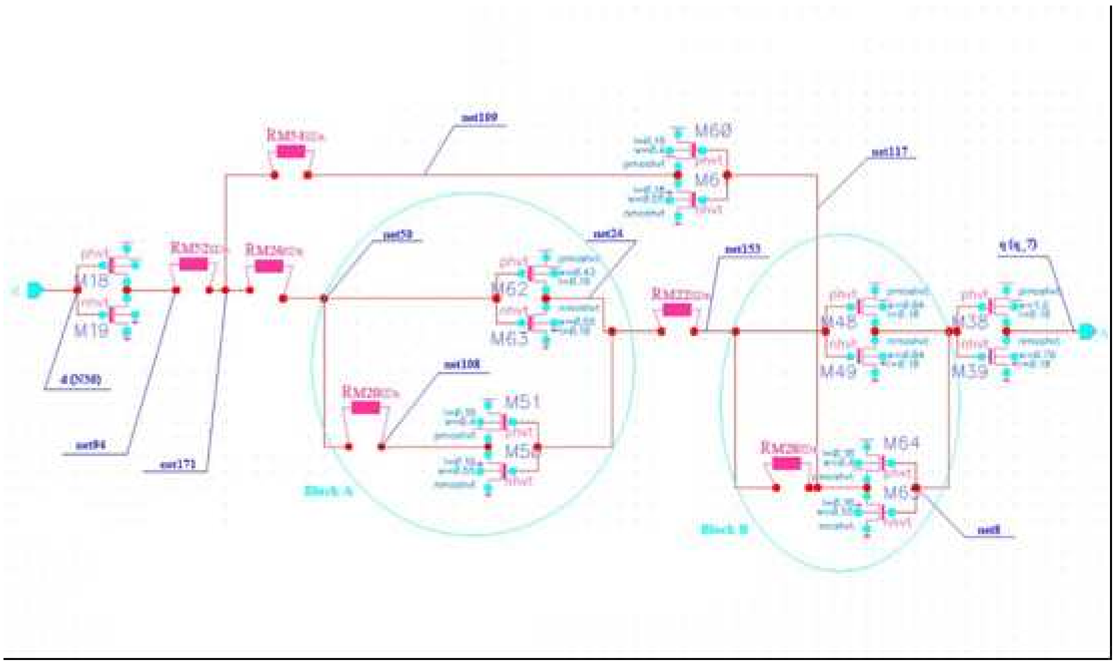


Figure 13 Register's internal circuit @ VDD = 0.72v

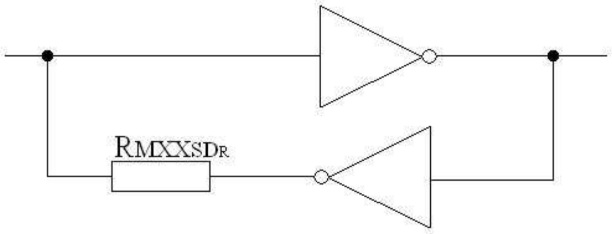


Figure 14 Block equivalent circuit

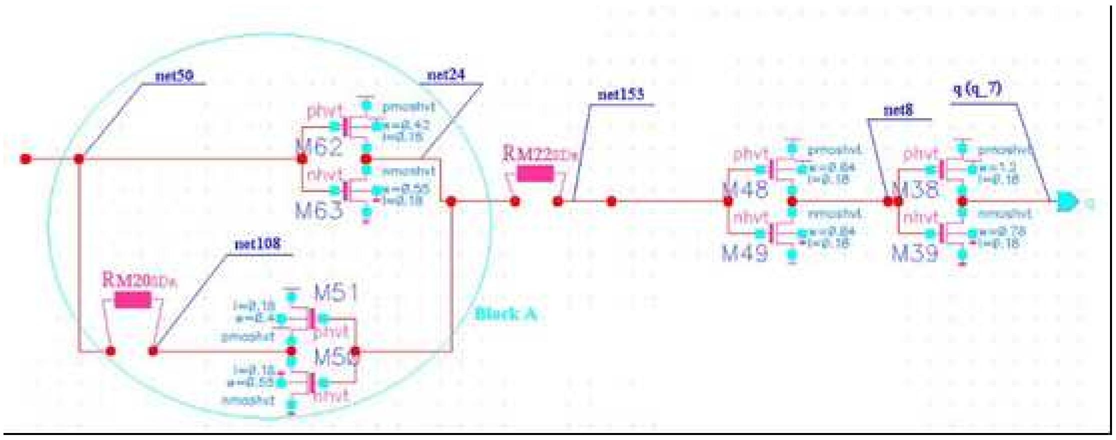


Figure 15 Register’s internal circuit @ VDD = 0.94v

Pulse 2

This pulse, at 140ns starts to rise from 1.6v to 6.4v, which is reached at 142ns. At this point it falls down to 0V at 149ns, rising to 1.6v at 154ns. It is, therefore, very similar to Pulse 1. These are the parameters to create this pulse:

```
VOLTAGE=1.6
DELAY_FOR_PULSE=140.0n
PULSE_START_TIME=0n
PULSE_START_VALUE=0
PULSE_P1_TIME=2n
PULSE_P1_VALUE=3
PULSE_P2_TIME=9n
PULSE_P2_VALUE=-1
PULSE_END_TIME=14n
PULSE_END_VALUE=0
```

Although Pulse 1 and Pulse 2 are very similar, Figure 16 shows different behaviour from the one seen in Figure 7. Although all the registers' outputs show a tendency to be set to logic 1, this tendency is interrupted in most registers, and the design shows normal behaviour after the pulse and keeps counting as if nothing had happened.

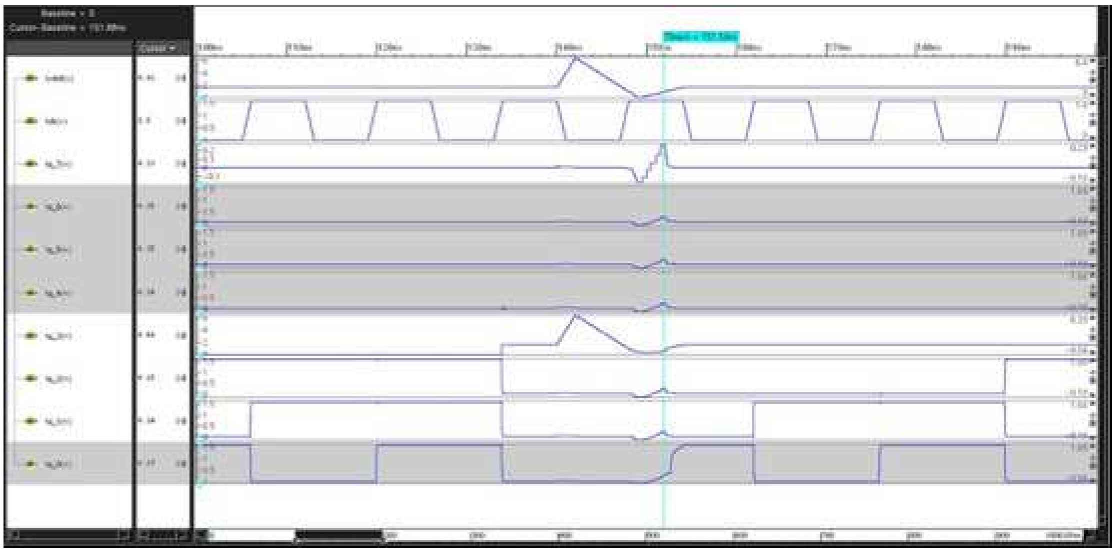


Figure 16 Design's response to Pulse 2

When the tendency to set all registers to logic 1 is interrupted, all registers that were logic 0 before the pulse was applied reach a maximum voltage on their output of 0.24V, except for Q7, which is 0.29V, and for Q0, which is 0.26V. The only register that was logic 1 before the pulse, Q3, has a voltage of 0.53V when the rise tendency is interrupted in other registers.

There are three registers to be analysed in this simulation, Q7, which keeps its logic 0 value after the pulse, Q3, which keeps its logic 1 value after the pulse and Q0, which changes its logic value from 0 to 1 after the pulse. The following section explains what happens internally in each case.

Analysis

Figure 17 shows the Q7 register's internal behaviour for this pulse. As shown with the previous pulse, register behaviour is primarily affected when VDD goes under 0.8V, so that is going to be the starting point for this pulse's analysis.

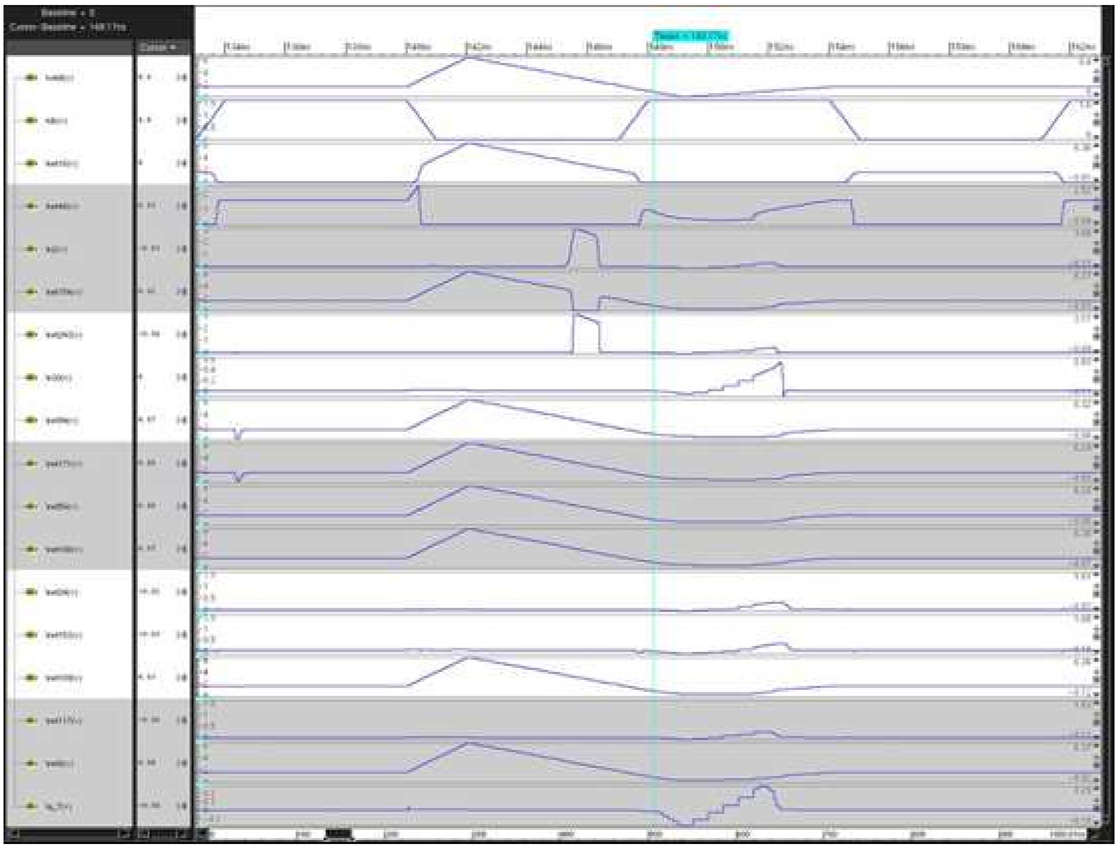


Figure 17 Q7 register's internal signals for Pulse 2

When VDD goes below 0.8v, all signals follow VDD's trend. When VDD becomes 0V, all nets that were supposed to be logic 1 keep a voltage of about a transistor threshold voltage (around 0.5V), and all nets that were supposed to be logic 0 have a negative voltage. As soon as VDD increases, those nets that where supposed to be logic 0, start to follow VDD and increase their nets' voltage values, and those nets that were suppose to be logic 1, remain around the threshold voltage until VDD rises above 0.8V.

When VDD is 0.64V, nets net94, net171, net109, net50, net108 and net8 are around 0.4V, and nets net24, net153 and Q7 are around 0.1V. This seems to indicate a tendency on Q7 to become a very low voltage, i.e. logic 0.

As VDD rises to 0.82V all nets increase their voltage by approximately the same amount, 0.11V. The voltage increase of low voltage nets can be explained by parasitic capacitors. This increase of voltage does not induce a behaviour change on a register's internal nets, which would by now look like Figure 15. In fact, from this point on, it shows a clear tendency to set the register's output to logic 0.

Figure 18 shows register Q3's internal behaviour, where the output remains logic 1 and unchanged after the pulse. During the pulse, this register shows behaviour not seen so far. The register's input net drops from logic 1 to logic 0 when VDD is over a voltage of approximately 3V. A voltage drop on the input net produces a cascade effect that is stopped with net24. This cascade effect does not affect the register's output because it only happens when the clock is logic 0. In the event that a clock positive edge happens while the input shows this voltage drop, the register's output would be affected and therefore so would the design's behaviour.

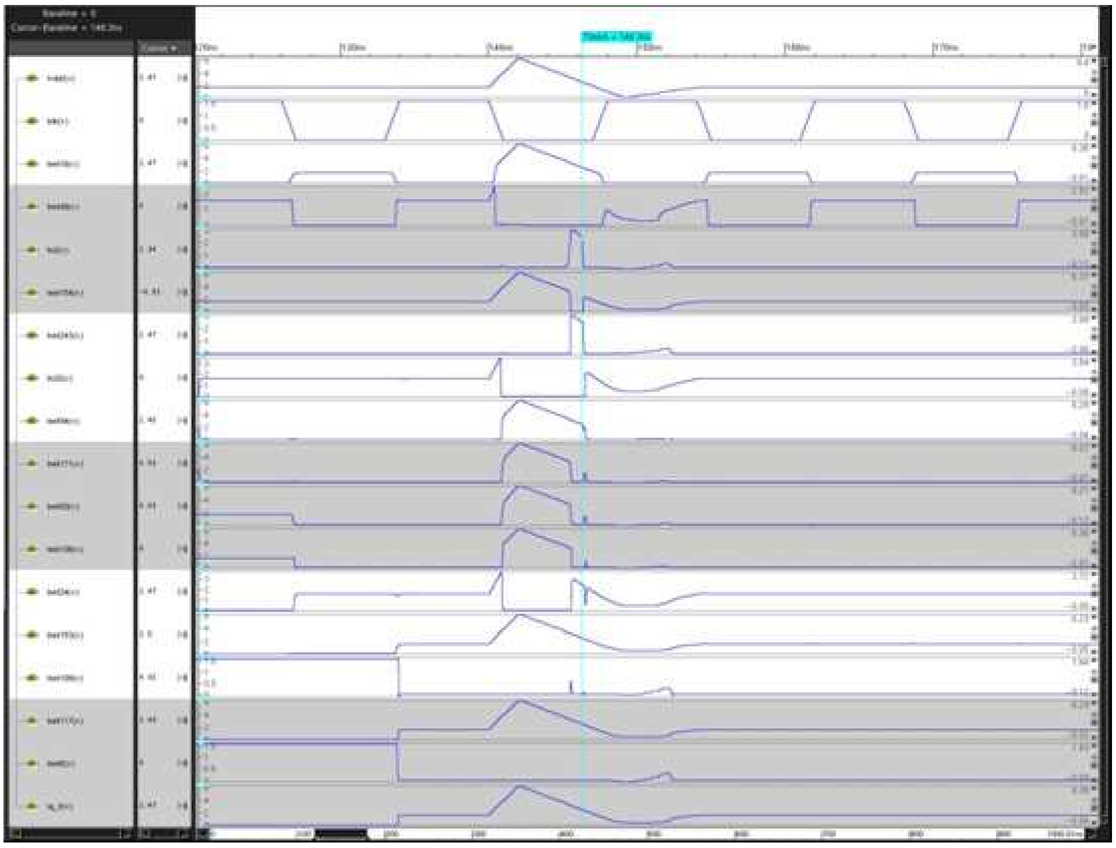


Figure 18 Q3 register's internal signals for Pulse 2

Figure 19 shows register Q3's data input circuit. As the input to U8's a2 input is always logic 0, all circuits prior to this point have been omitted in Figure 19. Additionally, clock and enable circuits have been omitted. N38 is the counter's output, which indicates the next logic value to be stored in register Q3. Net n12, U8's b2 input, indicates that counting is enabled.

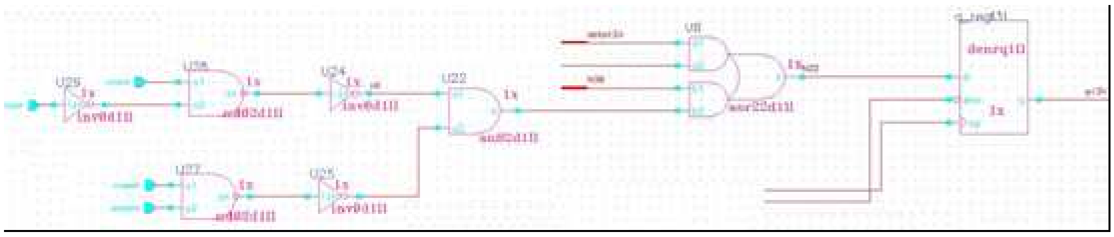


Figure 19 Register Q3 with combinational block

Figure 20 shows how n12, and all related input nets, evolve when the pulse is applied. Inputs count, enable and nreset do not change their voltage level during the pulse, keeping it at 1.6V, whereas load keeps its voltage level at 0V. Under normal circumstances, i.e. no pulse, the outputs of gates U27 and U28 (n8 and n10 in Figure 20) would be logic 0. However, as VDD rises, they are set to logic 1. This can be explained with the help of Figure 21, where the NAND gate's internals are shown. Analysing the device U27 on Figure 19, under normal circumstances, transistors U2 and U4 in Figure 21 would be closed, while transistors U1 and U3 would be open. But when VDD rises over 1.6V + $V_{threshold}$, transistors U1 and U3 would also be correctly polarised, closing them, which causes the gate's output to rise to a logic 1. This also applies to device U28. It is enough to induce a change on one of these devices' output to change register Q3's input logic value.

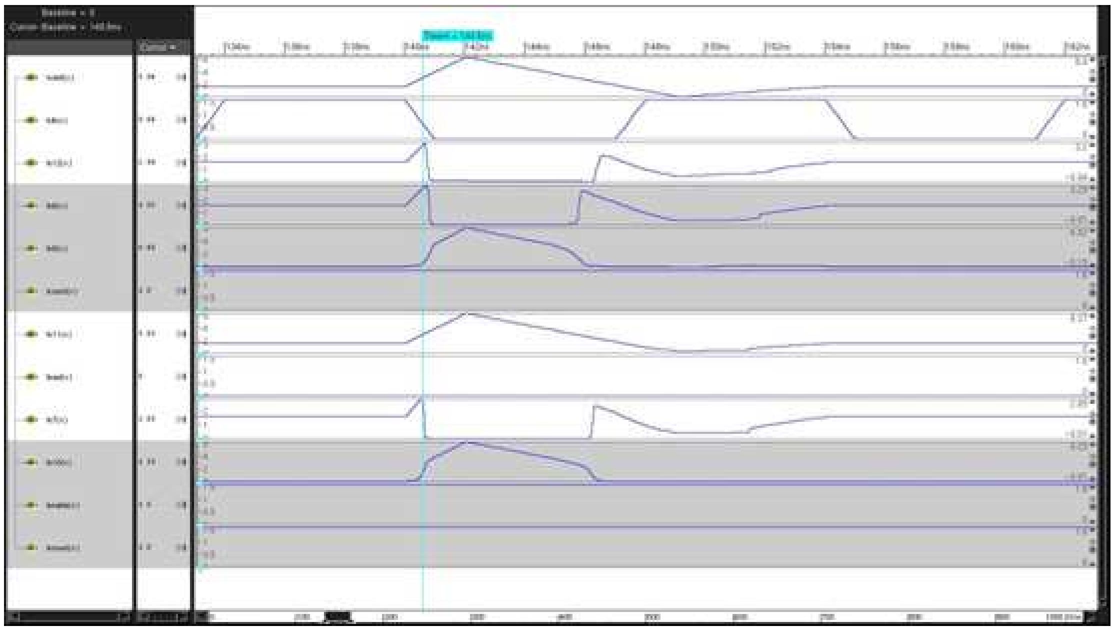


Figure 20 Combinational block's signals

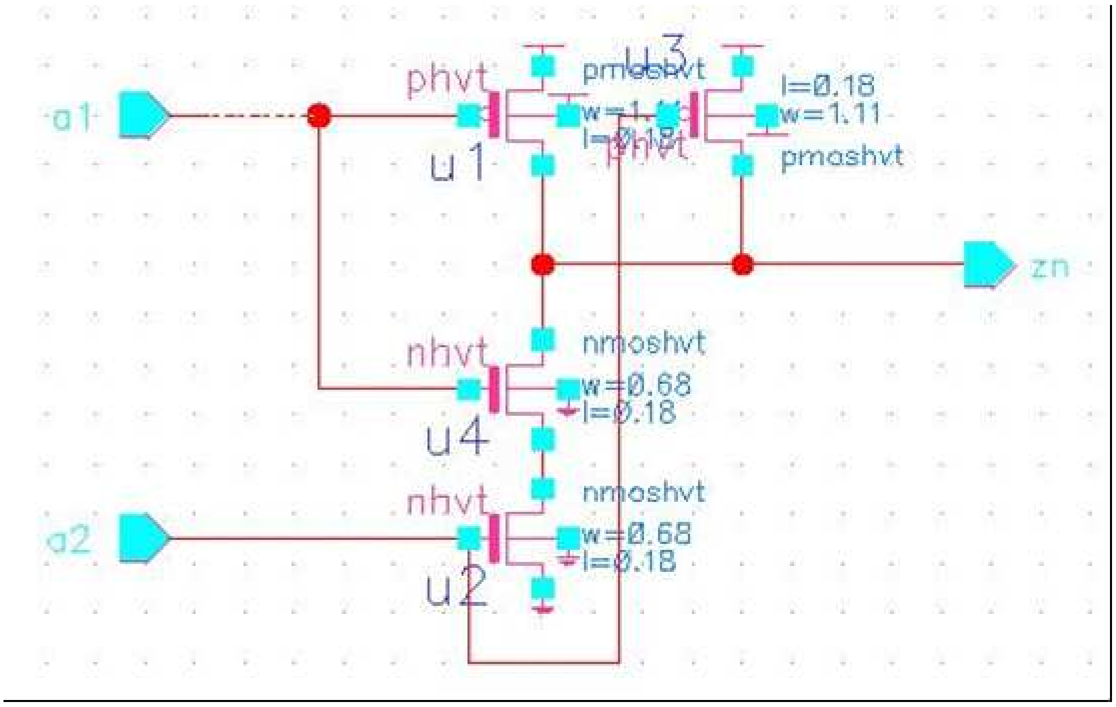


Figure 21 NAND gate's internals

Back on Figure 18, after the pulse has reached its lowest point and starts to recover, most signals' voltage value starts to rise too. As parasitic capacitors did not have time to discharge through transistor leakage, when VDD starts to rise, nets that were high before the pulse keep a higher voltage that nets where the logical value was 0. When VDD reaches 0.82V it would look like the circuit in Figure 15, enabling the register's output recovery. Therefore, when VDD recovers for this pulse, Q3 continues with its previous value unchanged.

Register Q0 is going to be analysed next. This register's nets' voltage values, while the pulse is applied, are shown in Figure 22. The first thing to mention for these waveforms is the pulse that appears on net24 on the clock's negative-edge. At this clock's negative-edge, net24 should be set to logic 1, which initially happens, but, because of the same phenomenon that happened on Q3's input, Q0's input goes from logic 1 to logic 0, switching net24's logic value back to 0. Hence the pulse on net24. Before the next clock's positive edge net24 is set to logic 1 while VDD is still higher than the nominal voltage. When VDD is 0.91V, before reaching 0V, the register's internals represent the circuit in Figure 14. net50 and net108 are logic 0 and net24 is logic 1, which induces net153 to become logic 1, net8 to become logic 0 and net117 to become logic 1. As VDD continues to decrease, all net voltages decreas too, net8 being the slowest one, keeping 0.62V when VDD is 0v. When VDD starts to recover, net8 and net109 are again the slowest ones. When VDD goes over 0.64V, net50 has 0.21V, net108v, net24 0.47v, net153 has 0.37v and net8 has 0.6V. As VDD rises over 0.8V, net24 reaches the threshold voltage faster than net50, determining the logic value to which the register is going to be set, which is logic 1.

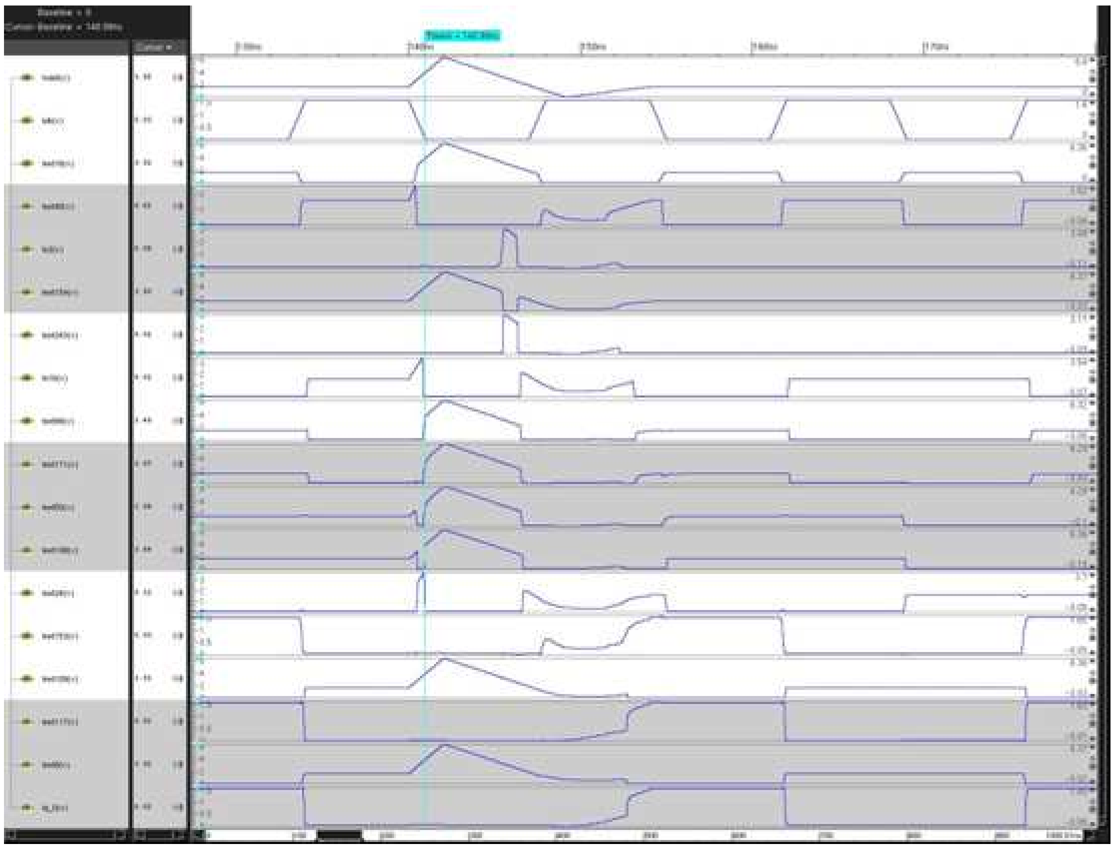


Figure 22 Register Q0's response to Pulse 2

Pulse 3

This pulse starts to rise at 140ns from 1.6V to 6.4V, which is reached at 142ns. At this point it falls down to 0V at 146.7ns, rising to 1.6V at 148.5ns, it is, therefore, similar to Pulse 2, but faster. Figure 23 depicts the counter's response to this pulse. These are the parameters to create this pulse:

```
VOLTAGE=1.6
DELAY_FOR_PULSE=140.0n
PULSE_START_TIME=0n
PULSE_START_VALUE=0
PULSE_P1_TIME=2n
PULSE_P1_VALUE=3
PULSE_P2_TIME=6.7n
PULSE_P2_VALUE=-1
PULSE_END_TIME=8.5n
PULSE_END_VALUE=0
```

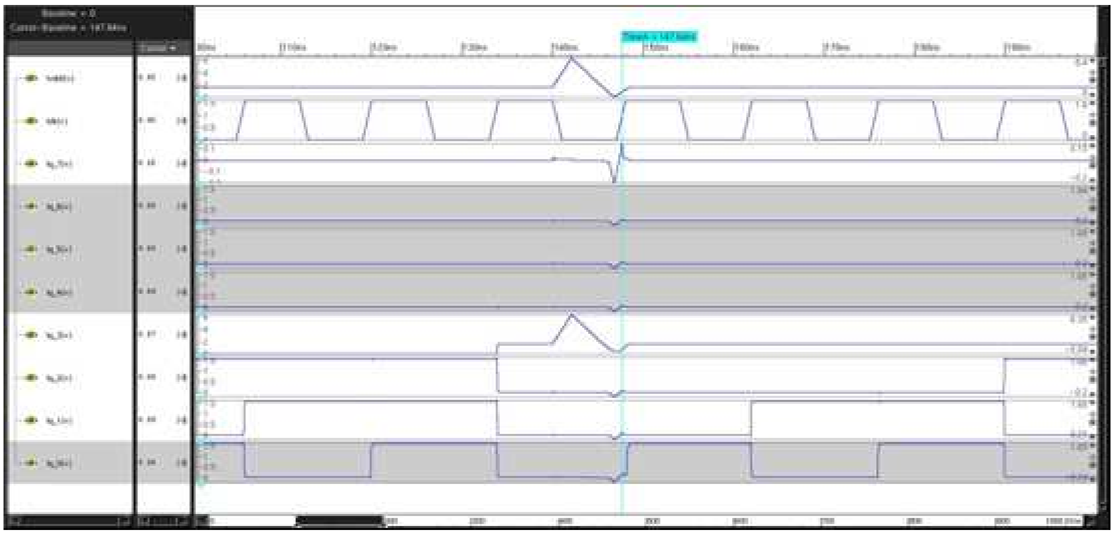


Figure 23 Design's response to Pulse 3

As can be seen from Figure 23, of the nets that are logic 0, Q7 reaches the highest peak when VDD is rising back to 1.6V. This pulse also delays Q0's output a little, which ends up being set to logic 1, as it should be. For this pulse, which looks very similar to the response of previous pulses, Q0 and Q7 could be analysed.

Analysis

Figure 24 shows register Q7's behaviour with this pulse, which is very similar to its reaction to the previous pulse. Figure 25, Q0's response to this pulse, also shows a similar behaviour to the previous pulse. As everything explained for the previous pulse applies to this pulse as well, no further analysis has been made.

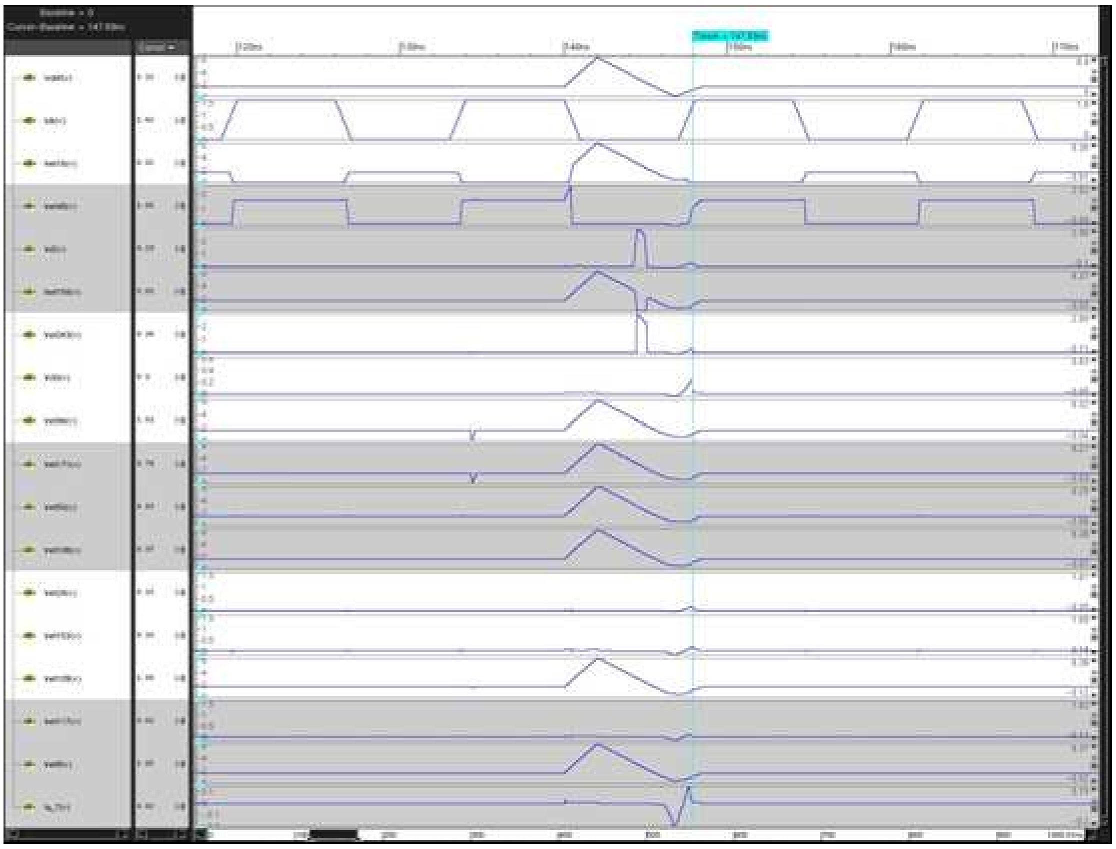


Figure 24 Register Q7's response to Pulse 3

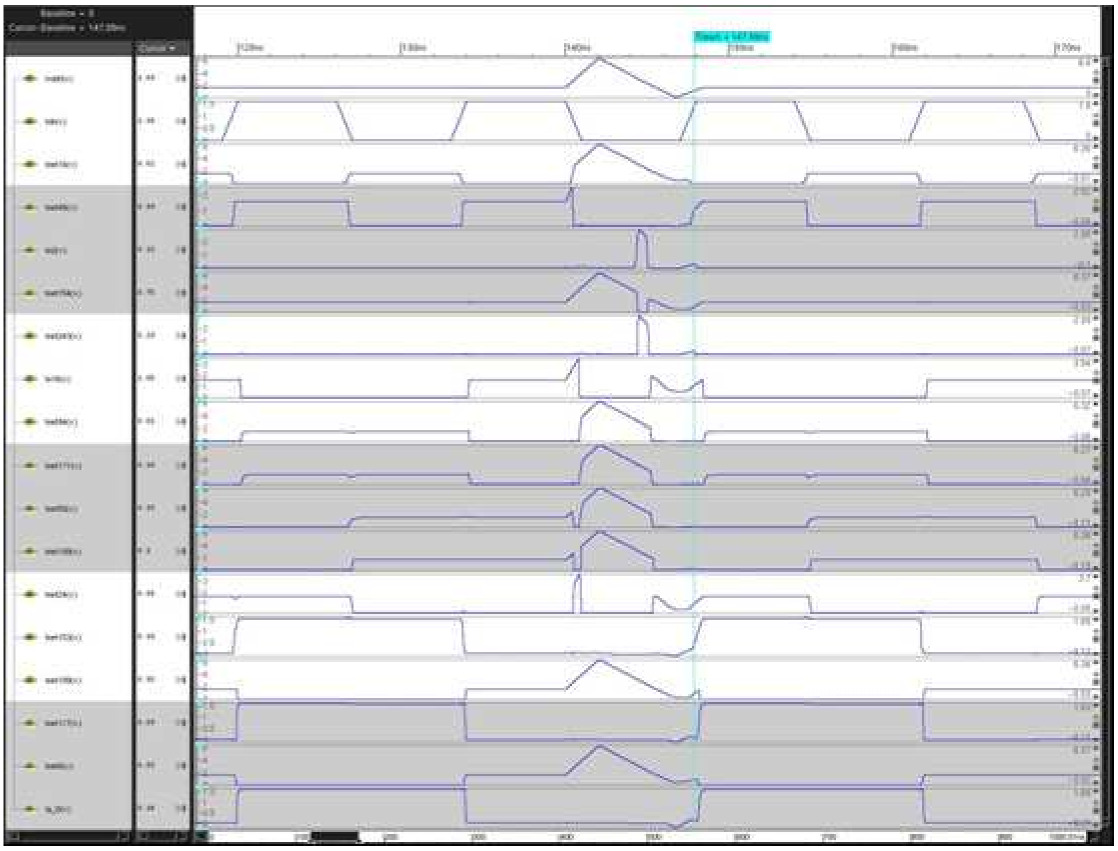


Figure 25 Register Q0's response to Pulse 3

Pulse 4

This pulse starts to rise at 145.7ns from 1.6v to 6.4v, which is reached at 147.6ns. At this point it falls down to 0V at 152.3ns, rising to 1.6V at 154ns. This is a slightly faster pulse (0.2ns) than Pulse 3 and is applied 5.7ns later than Pulse 3. Figure 26 depicts the counter’s response to this pulse. These are the parameters to create this pulse:

```
VOLTAGE=1.6
DELAY_FOR_PULSE=140.0n
PULSE_START_TIME=5.7n
PULSE_START_VALUE=0
PULSE_P1_TIME=7.6n
PULSE_P1_VALUE=3
PULSE_P2_TIME=12.3n
PULSE_P2_VALUE=-1
PULSE_END_TIME=14n
PULSE_END_VALUE=0
```

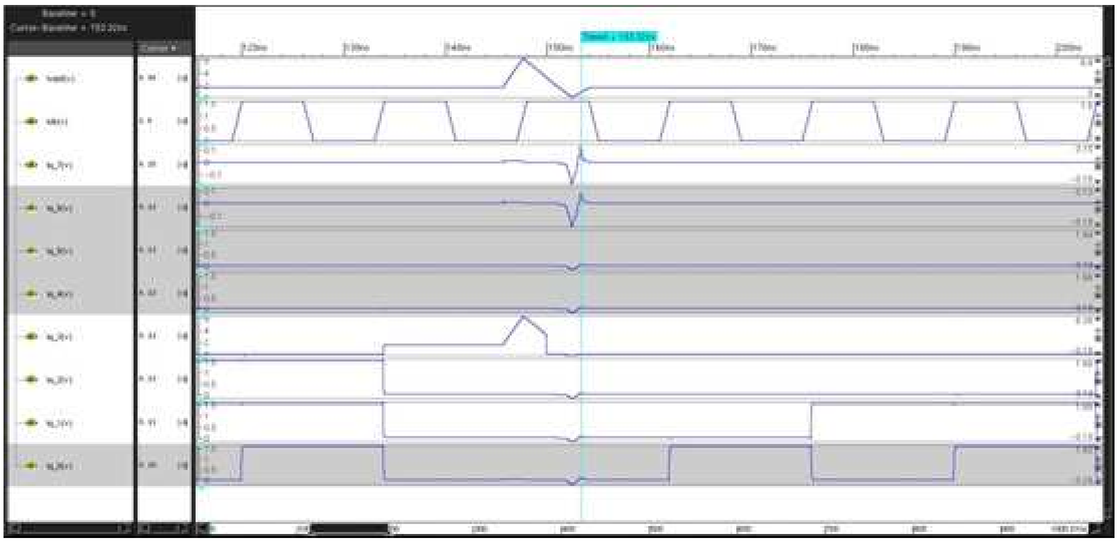


Figure 26 Design's response to Pulse 4

So far, the design’s response to applied pulses has been:

- setting all registers’ outputs to logic 1 and
- no change in the design’s behaviour or counting sequence.

This pulse shows a different behaviour to those seen until now; the design’s output is set to logic 0. In Figure 26 it is interesting to note that register Q3’s output is set to logic 0 when VDD is 3.23V and while decreasing, and that all register outputs show a tendency to rise when VDD is rising from 0V to 1.6V. Again, the main registers to be analysed in order to understand the effects of this pulse are, Q7, as it is the one which rises higher than any other register; Q3, as its output value is suddenly cut from logic 1 to logic 0; and Q0 which should be logic 1 when the pulse is rising.

Analysis

Register Q7’s behaviour in Figure 27 shows a well-known behaviour, while only clock related internal signals show new behaviour. Nets net16 and net48 do not change their logic values when the clock positive edge happens, but do change later. This is explained because, before the clock starts to rise, M16 and M15 are conducting. When the clock signal starts to rise, VDD is higher than the nominal value and still rising, which means that M16 keeps conducting. When the clock signal reaches its maximum value, VDD is starting to decrease but is still well over the nominal value, whichforces M16 to conduct. At this point, M17 is also conducting but net16 keeps its logic 1 value, forcing net48 to be logic 0. When VDD decreases to below 1.6V + $V_{threshold}$, M16 stops conducting. net16 is set to logic 0, and net48 is set to logic 1. It is at this point when a positive edge is detected. As data on N30 has not changed and net109 has the same logical value as net94, this distortion on clock positive edge detection has no effect on this register’s behaviour.

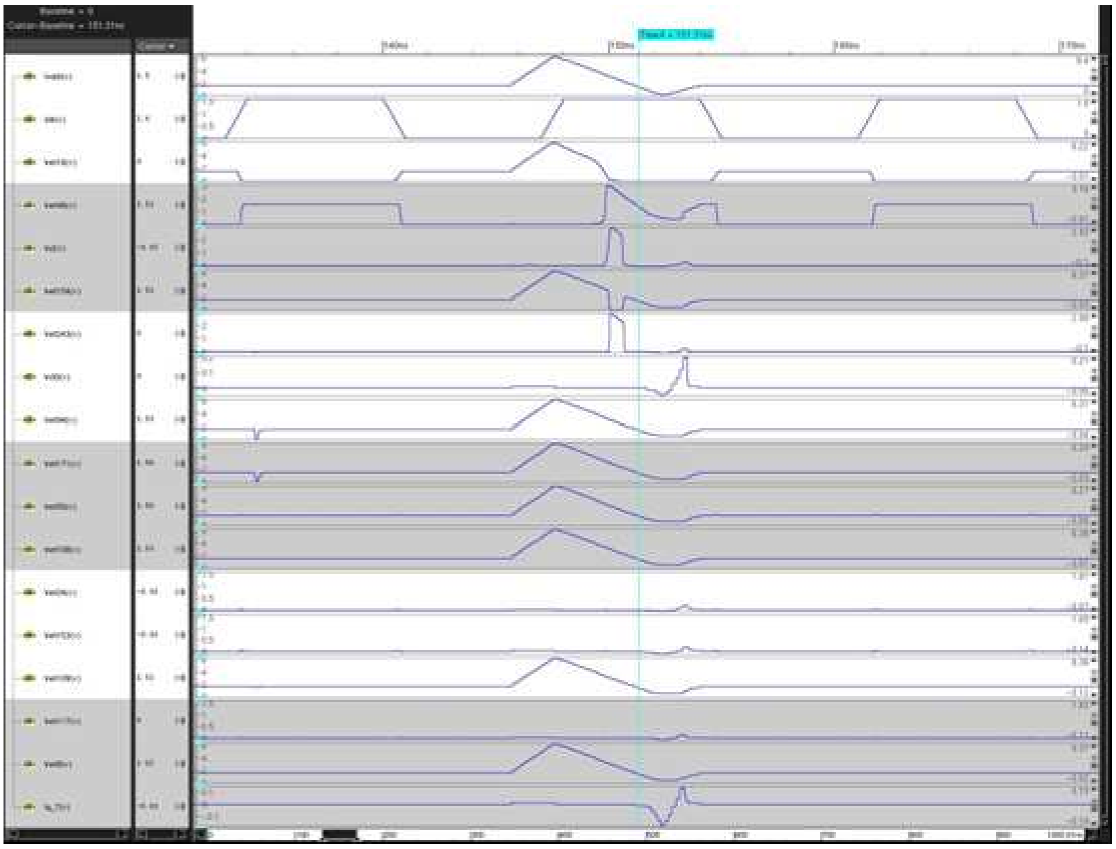


Figure 27 Register Q7's response to Pulse 4

Q7 shows a tendency to set its output to logic 1 when VDD is recovering, but this tendency is stopped. As seen with previous pulses, nets which have negative voltages, start to raise their voltage level when VDD starts to rise over 0v, whereas nets that have positive voltages will start to show this tendency when VDD becomes higher than them.

At the time that Q7's output is rising, the clock is logic 1 and the register's output value depends on nets net50, net108 and net24, which follow a tendency to recover their correct logic values. Q7's output rises only as a side effect of parasitic capacitors of transistors M38 and M39 as Q7's output is forced to logic 0 as soon as net8 goes over 0.8V.

Register Q3, however, shows a different behaviour. In Figure 28, while the clock signal is logic 0, net24 has the same value as the register input, which is logic 1, and net153, also logic 1, is not affected by net24. When VDD starts to rise, the input signal voltage drops until it becomes logic 0, as per the phenomenon explained in Figure 19. This causes a logic value change on net24, which now is logic 0. Additionally, before the input signal can be recovered to its original value, a clock positive edge is "detected", which forwards net24's logic 0 value further into the register until its output switches from logic 1 to logic 0. From this point on, VDD continues to decrease until it reaches 0v, when it rises again to 1.6V. As nets net50, net108 and net24 do not change their logic values, the register's output remains at logic 0 instead of its original value.

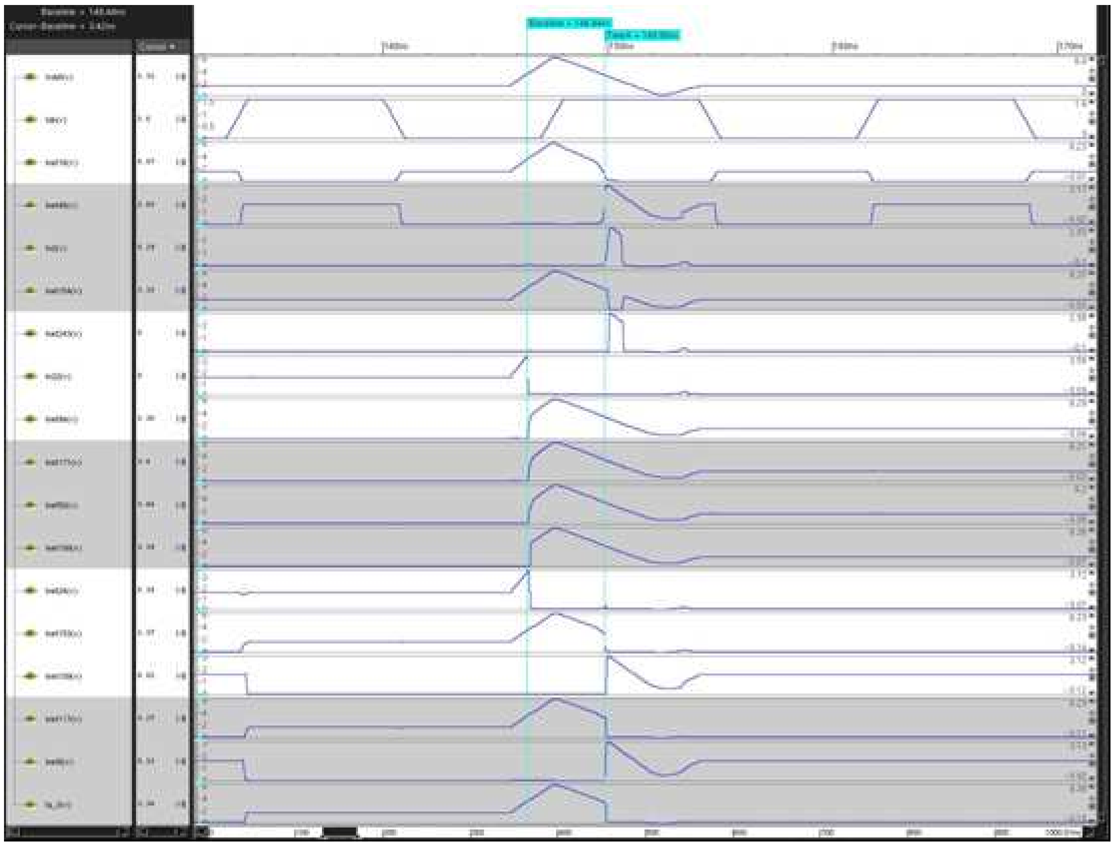


Figure 28 Register Q3's response to Pulse 4

The next register to be analysed is Q0. When the pulse is happening, it should be set to logic 1, instead it remains at logic 0. Figure 29 shows this register's internal signals. For the Q0 case, as well as for Q3, the input signal switches from high to low as a result of a voltage rise on VDD, hence, when the positive edge on clock's signal is "detected", the register's input is logic 0, which is forwarded throughout the registers' internal nets, setting it to logic 0. As can be seen, once the clocks positive edge is detected, Q3 and Q0 have the same evolution.

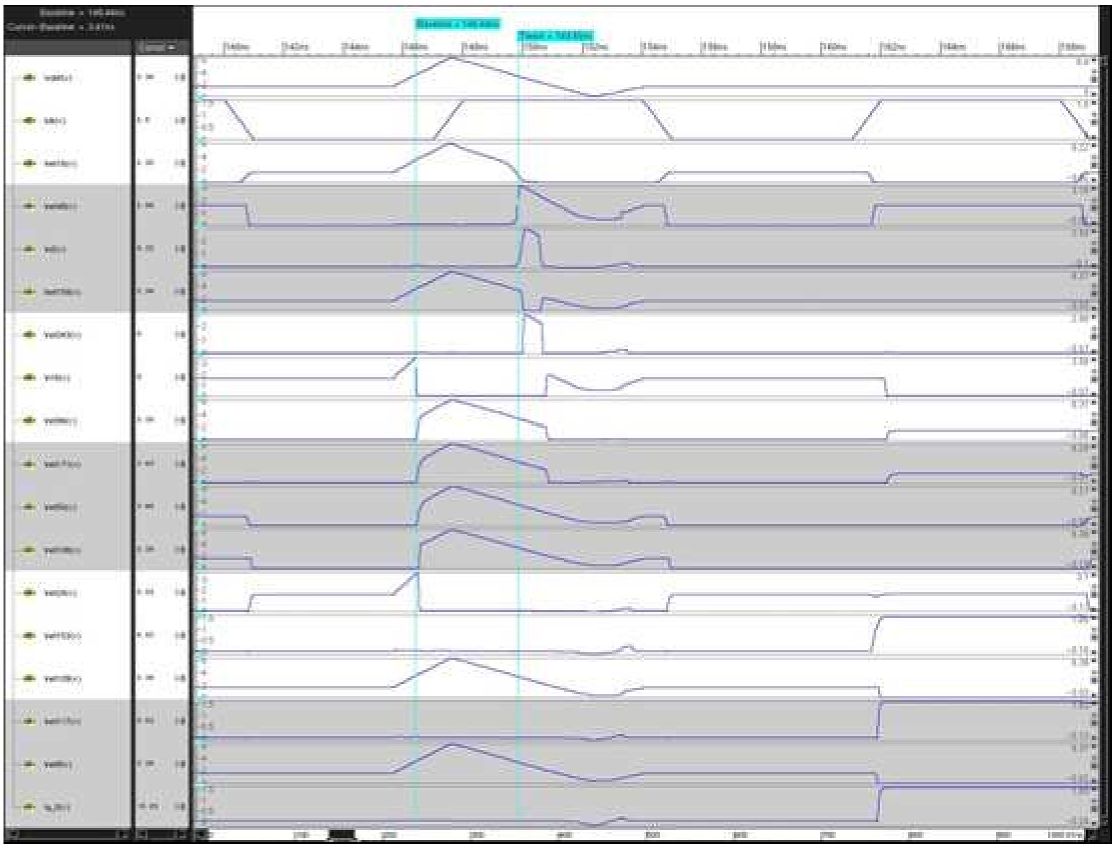


Figure 29 Register Q0's response to Pulse 4

Pulse 5

This pulse starts to rise at 142.3ns from 1.6V to 6.4V, which is reached at 144.2ns. At this point it falls down to 0V at 148.9ns, rising to 1.6V at 150.6ns. This is, therefore, the same pulse as Pulse 4 but applied 3.4ns earlier than Pulse 4. Figure 30 depicts the counter's response to this pulse. These are the parameters to create this pulse:

```
VOLTAGE=1.6
DELAY_FOR_PULSE=140.0n
PULSE_START_TIME=2.3n
PULSE_START_VALUE=0
PULSE_P1_TIME=4.2n
PULSE_P1_VALUE=3
PULSE_P2_TIME=8.9n
PULSE_P2_VALUE=-1
PULSE_END_TIME=10.6n
PULSE_END_VALUE=0
```

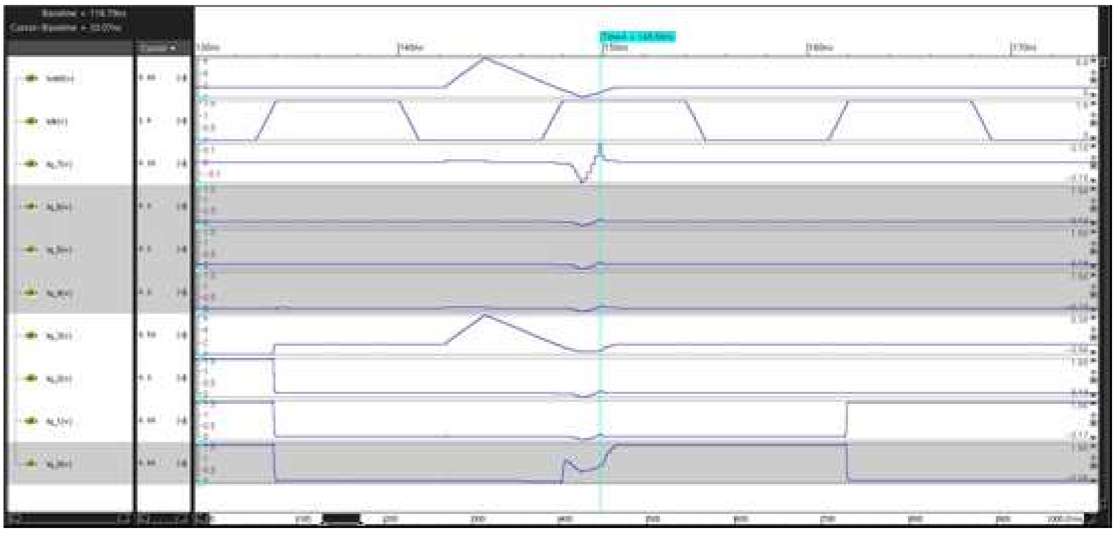


Figure 30 Design's response to Pulse 5

Comparing Figure 30 and Figure 26, we see that a small shift on the pulse induces different behaviour on the circuit. In this case, the same pulse used on Figure 26 has been applied slightly earlie r resulting in no effect on the circuit's behaviour. Referring to the registers analysed in the previous case, only affected registers will be analysed, i.e. Q3 and Q0.

Analysis

Figure 31 shows how the register's input value drops to 0v when VDD goes over 3.54V and does not recover until VDD decreases back to 2V. The clock positive edge is detected just 0.3ns after the input value has recovered its original value. At this time net24 is logic 1, as before the pulse. This causes the register to keep its previous value and to continue as if nothing had happened. This has been possible because of the timinh of nets n2 and n22. If the first singularity on n2 ended earlier than n22 restores its value, there would be a chance to change the registers' output value if, and only if, the clock positive edge happened in between n2's fall and n22's rise.

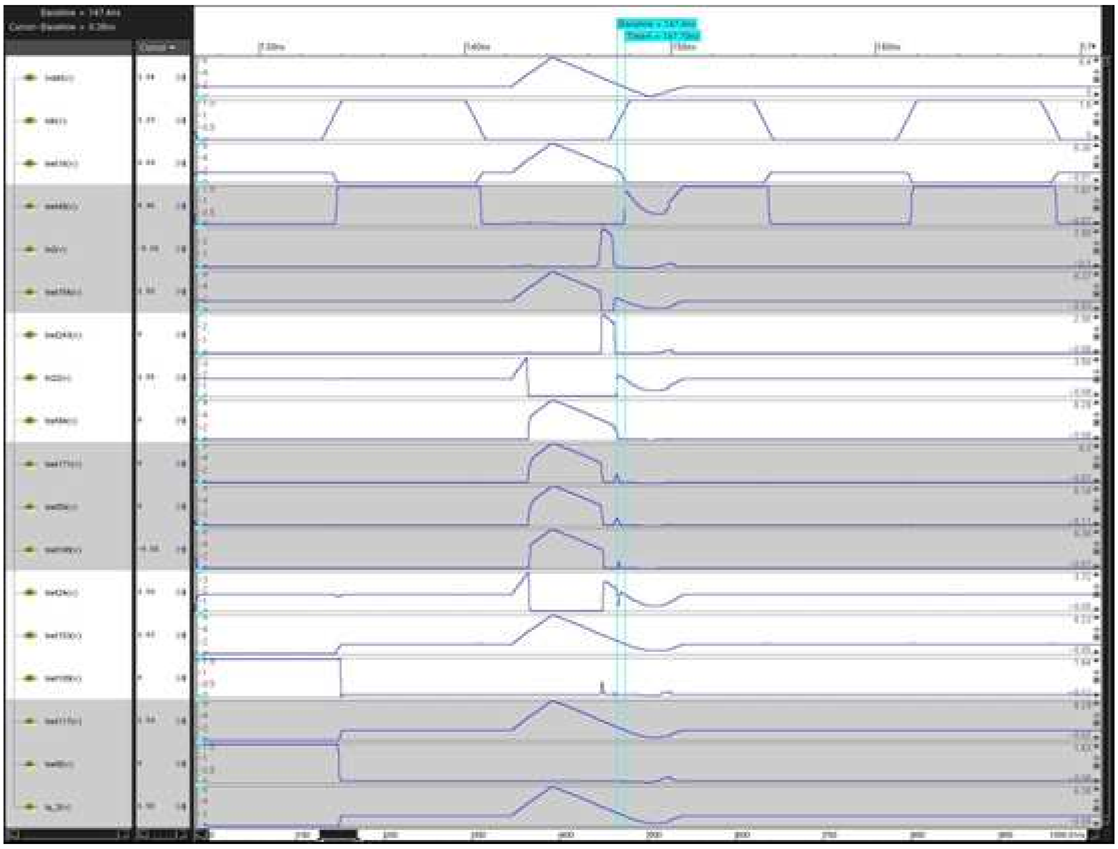


Figure 31 Register Q3's response to Pulse 5

Reading Figure 32 it can be seen that in this case too, the register's input is affected by VDD's rise, which is also recovered just before the clock's positive edge is detected. This results in setting the register's output to logic 1, as it should be. It can also be seen that if the clock's positive edge had been detected about 0.3ns earlier, this register's output would not have changed from logic 0 to logic 1, whereas in the case of register Q3, 0.3ns early detection of clock positive edge would not affect its output, with an overall result of delaying/dephasing the count by one compared to the original behaviour.

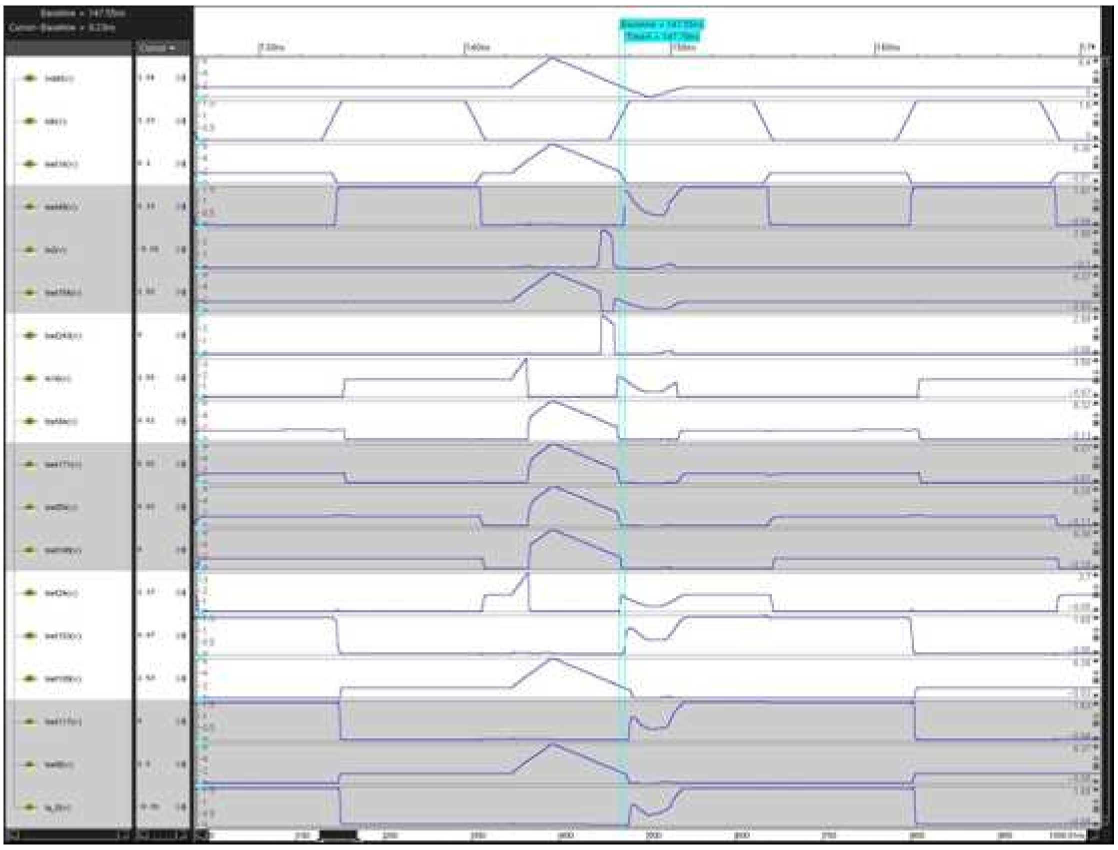


Figure 32 Register Q0's response to Pulse 5

Pulse 6

This pulse starts to rise at 142.33ns from 1.6V to 6.4V, which is reached at 144.2ns. At this point it falls down to 0v at 144.6ns, rising to 1.6V at 150.6s. This therefore starts at the same time and has the same length as Pulse 5 but VDD's fall is much faster and its recovery is much slower than Pulse 5. These are the parameters to create this pulse:

```
VOLTAGE=1.6
DELAY_FOR_PULSE=140.0n
PULSE_START_TIME=2.3n
PULSE_START_VALUE=0
PULSE_P1_TIME=4.2n
PULSE_P1_VALUE=3
PULSE_P2_TIME=4.6n
PULSE_P2_VALUE=-1
PULSE_END_TIME=10.6n
PULSE_END_VALUE=0
```

The design's response to Pulse 6 can be seen in Figure 33. It is clear that this pulse resets the registers' outputs to logic 0. Also, Q3, which was high before and during the pulse, needs longer than other registers to set its output to logic 0. Again there are three registers to be analysed, Q7, Q3 and Q0.

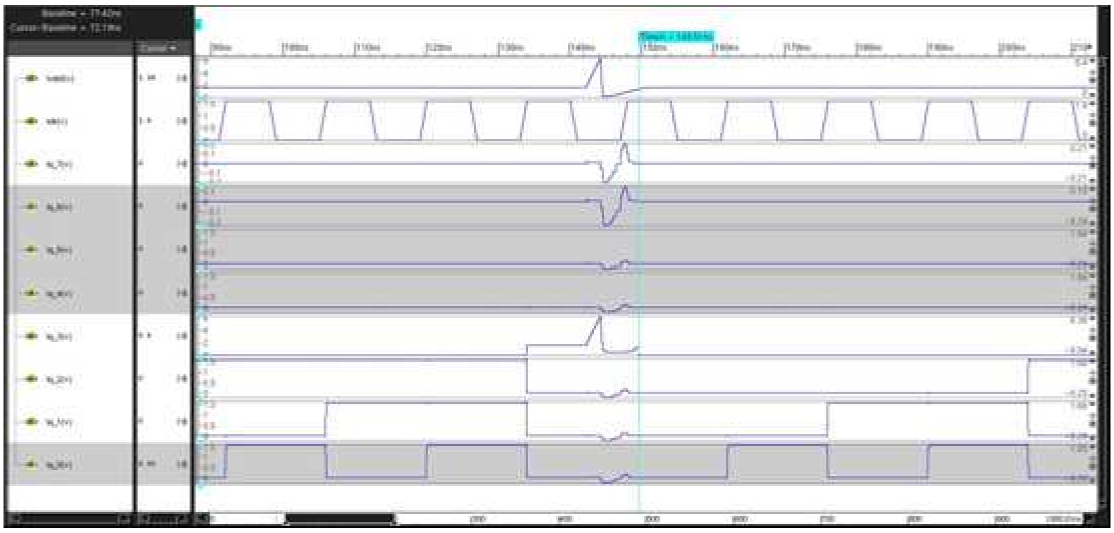


Figure 33 Design's response to Pulse 6

Analysis

Figure 34 shows the impact of this glitch in the Q7 register's internal nets. While VDD rises, Q7's logic 1 nets increase their voltage too, while logic 0 nets remain almost unchanged. VDD's fast fall induces a fast reaction on the registers' nets, and in general, the design's nets. Nets that were logic 0 would reduce their voltage, reaching as low as -0.28V on some register's internal nets and as low as -0.99V on the register's enable input. From this point, VDD starts to rise very slowly. At VDD=0.03V, we have the following voltage on different data nets: N30 has -0.16V; net94 has 0.83V; net171 has 0.91V; net50 has 1.03V; net108 has 0.69V; net24 has -0.01V; net153 has -0.25V; net109 has 0.72V; net117 has -0.24; net8 has 0.5V; and Q7 has -0.2V. Control nets' voltage values are: net16 has 0.69V; net48 has -0.1V; n2 has -0.89V; net154 has 0.01V; and net243 has -0.52V. At this point it could be said that, so far, the register has a tendency to keep its value.

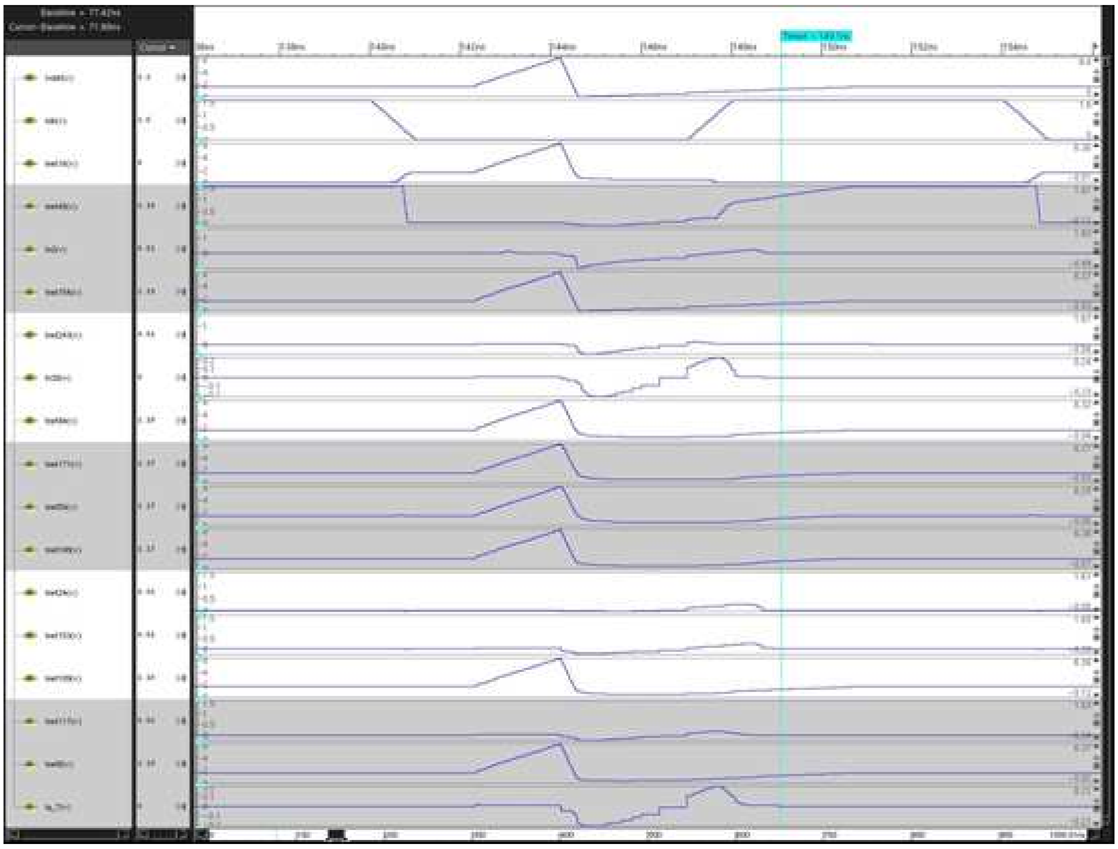


Figure 34 Register Q7's response to Pulse 6

While VDD continues increasing, all data related nets decrease their voltage a little more before starting to rise again at VDD=0.26V. As previous logic 0 nets' voltage values are still negative, whereas previous logic 1 nets' voltage values are still over VDD, previous logic 0 nets' voltages rise faster than previous logic 1 nets (due to the parasitic capacitance on transistors). Despite this, logic 0 nets do not rise fast enough to switch or induce a switch on the register's logic value and, when VDD rises over 0.67V, logic 1 nets start to increase their voltage values too.

Q7 to N30 feedback slows down the recovery of logic 1 nets, but this feedback is stopped when a positive edge is detected on the clock, after which all nets tend to stabilise on the same logic value as before the pulse.

After analysing Figure 35, it can be said that the rise on VDD induces a logic value switch on the data input net, which is forwarded to net24, which is passed to net153 on the clock's positive edge, resulting in setting the register's output value to logic 0. A more careful analysis shows that, after the clock's positive edge, net153 requires more time than usual to be set to net24's value. But let's see what is happening with this pulse.

As VDD rises over 3.54V, it induces a logic value switch on input n22 from 1 to 0. This also induces some changes inside some register nets' logic values. Affected nets are net94, net171, net50 and net108, which change from logic 0 to logic 1, and net24 which changes from logic 1 to logic 0 and not affecting any other net. As VDD falls, net22 should recover its original value, i.e. logic 1, however, it remains at logic 0 and it reaches -0.06V with a tendency to continue decreasing when VDD is 0V. Analysing register Q3's input combinational block's signals should show the reason why n22 did not recover its original value. Figure 36 show these signals' waveform.

As VDD rises over 2.61V, nets n8 and n10 start to switch their values from logic 0 to logic 1. When VDD starts to fall, both nets start a process to return back to logic 0. However, VDD falls with the same slope as both nets, which only became logic 0 when VDD is nearly 0V. This causes n6 and n8 to remain at logic 0 when they should change to logic 1 and, as a side effect, n6, n7 and n12 get negative voltage values until VDD gets over 0.54V for n8, over 0.69v for n7 and over 0.75V for n12, which does not became logic 1 again until VDD reaches 1V.

Returning to Figure 35, as VDD starts to rise again, all nets decrease for a limited period of time and then start to follow VDD's tendency. Those nets whose value during the pulse's rise was logic 0 rise faster than those whose logic value was 1, for same reasons as before. As VDD rises, but while it is less than threshold voltage, M58 is the only transistor that is conducting (Figure 9). While VDD is rising, all internal nets, whose voltage is less than VDD, follow VDD's tendency to rise. When VDD reaches 0.8V, the register's internal structure is the circuit shown in Figure 12, with the exception of net109's feedback to net171. n22 has 0.27V, net94 has 0.46V, net171 has 0.46V, net50 has 0.47V, net108 has 0.42V, net24 has 0.39V, net153 has 0.49V, net109 has 0.4V, net117 has 0.5V, net8 has 0.22V and Q3 has 0.47V. All of them have very similar values and, at first, it seems that the register's final logic value could be either 1 or 0. As soon as VDD goes over 0.9V and net48 takes a logic 1 value, the register's internal structure switches to Figure 14, where Block A's output is inverted and is used to feed its own input and Block A's output is used to feed transistor pair M48-M49, their output feeds transistor pair M38-M39, the register's output. At this time, net50 has 0.55V, net108 has 0.49V and net24 has 0.47V; net153 has 0.56V, net8 has 0.3V and Q3 has 0.55V. Block A's nets' voltages are very close each other and if net24 rises faster than net50, it may balance the register's output towards logic 1, otherwise it could be balanced towards logic 0.

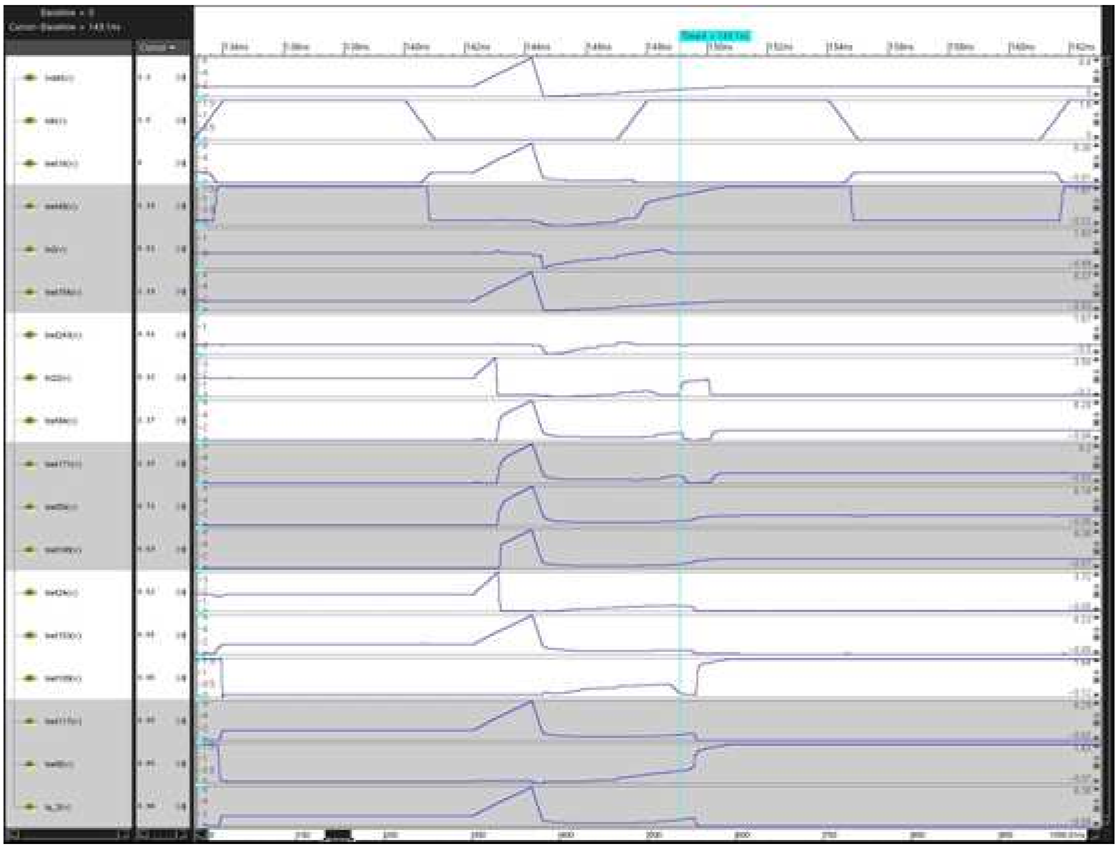


Figure 35 Register Q3's response to Pulse 6

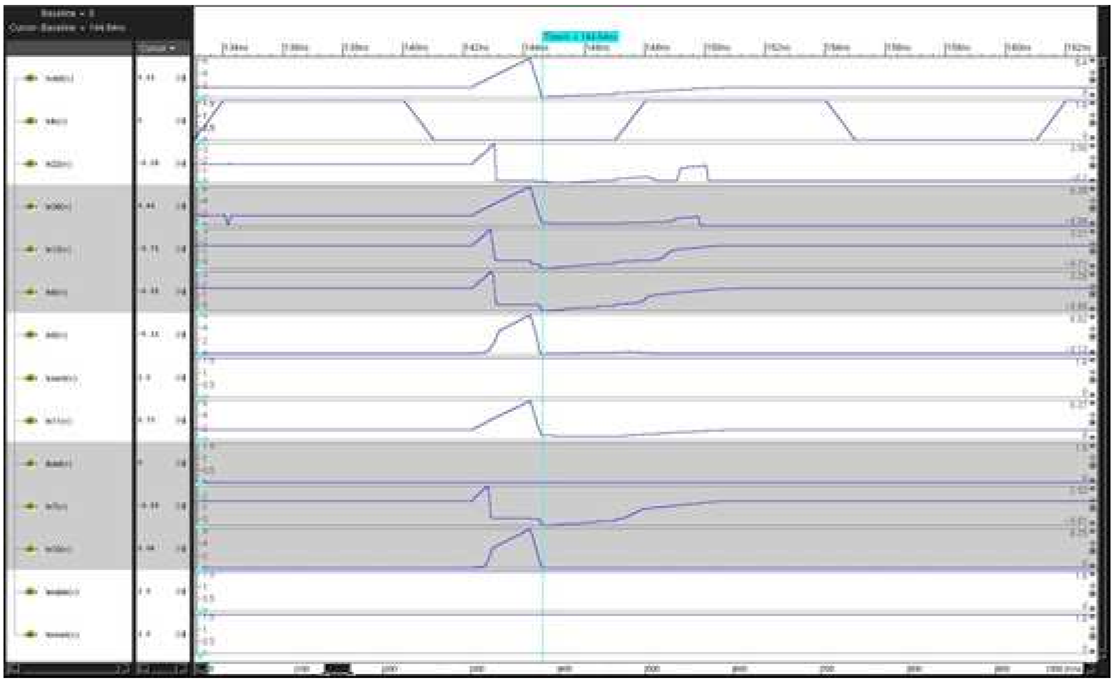


Figure 36 Register Q3's combinational input's signals

As VDD continues rising (at VDD=1.13V), net50 is 0.65V and starts to induce a change on net24's voltage, which has risen to 0.54V,. net108 is 0.58V. From this point on, net50 and net108 start to increase their voltages faster than before. While this is happening, net153 has 0.63V and net8 has 0.42V and the register's output has reached 0.8V.

As net50's voltage increases, net24 starts to decrease its voltage and starts to induce the same tendency on net153, and this to net8. However, as net8 is still lower than the threshold voltage, i.e. 0.55V, net8 has little or no effect over Q3's output, which continues increasing its voltage value as VDD rises. This, in turn increases the register's input voltage so it is considered a logic 1 again. But, as the register has already detected a positive edge on the clock, its input's logic value cannot be forwarded to net50 and net24. This way, net24 follows the new tendency towards logic 0, as net50 is becoming logic 1. As M63's v_{gs} rises, net24 and net153 become logic 0 faster. Only when net153 decreases enough to polarise transistor M48 does net8 boost its voltage to become logic 1 and start to influence Q3's output to become logic 0, which happens as fast as net8 rises.

Finally, the register's output is stabilised with a logic 0 value when VDD is 1.36V. The register's input still shows a logic 1 value but due to feedback, it becomes logic 0 before the clock's negative edge.

Register Q0's behaviour is also affected by Pulse 6 (Figure 37). In this case the register's output should be set to logic 1, instead, it remains logic 0. Again, with a quick look, the reason why this happens can easily be discovered. As with the previous case, Q0's input switches from logic 1 to logic 0 because of the rise of VDD. Due to the fast fall on VDD, the register's input has no time to recover, so by the time VDD falls below the threshold voltage, the input's new value has been propagated to net24. From this point on, the explaination for Q3 applies to Q0 with the difference that net50, net108 and net24 are more defined/separated in this case than on the former one, thus, stabilising Q0's output faster than Q3's output.

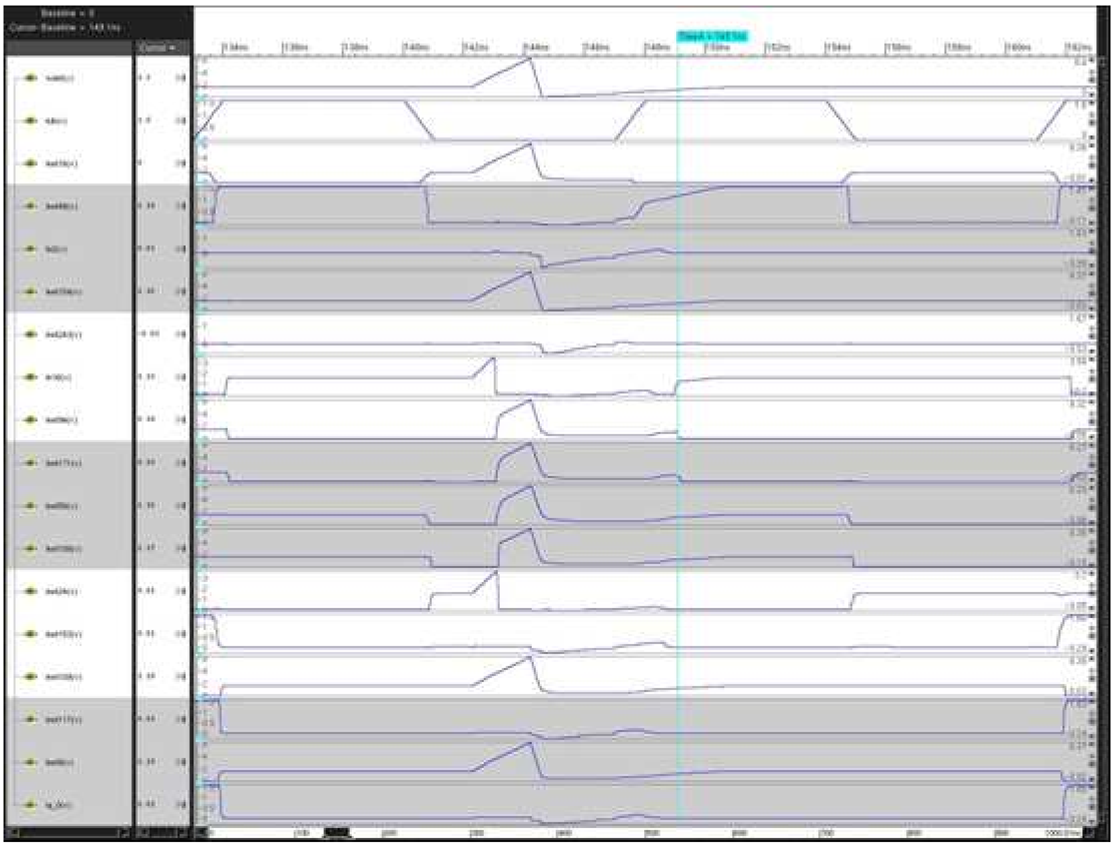


Figure 37 Register Q0's response to Pulse 6

Pulse 7

This pulse starts to rise at 142.3ns from 1.6V to 6.4V, which is reached at 144.2ns. At this point it falls down to 0v at 146.6ns, rising to 1.6V at 150.6ns. This therefore starts at the same time and has the same length as Pulses 5 and 6. With this pulse, VDD falls faster than Pulse 5 but more slowly than Pulse 6. VDD's recovery is, on the other hand, slower than Pulse 5 but faster than Pulse 6. These are the parameters to create this pulse:

```
VOLTAGE=1.6
DELAY_FOR_PULSE=140.0n
PULSE_START_TIME=2.3n
PULSE_START_VALUE=0
PULSE_P1_TIME=4.2n
PULSE_P1_VALUE=3
PULSE_P2_TIME=6.6n
PULSE_P2_VALUE=-1
PULSE_END_TIME=10.6n
PULSE_END_VALUE=0
```

This pulse affects the circuit's behaviour in a new way compared to previous pulses. It dephases, by one clock cycle, the circuit's output when comparing it against the original output (Figure 5). The circuit's response to Pulse 7 is represented in Figure 38.

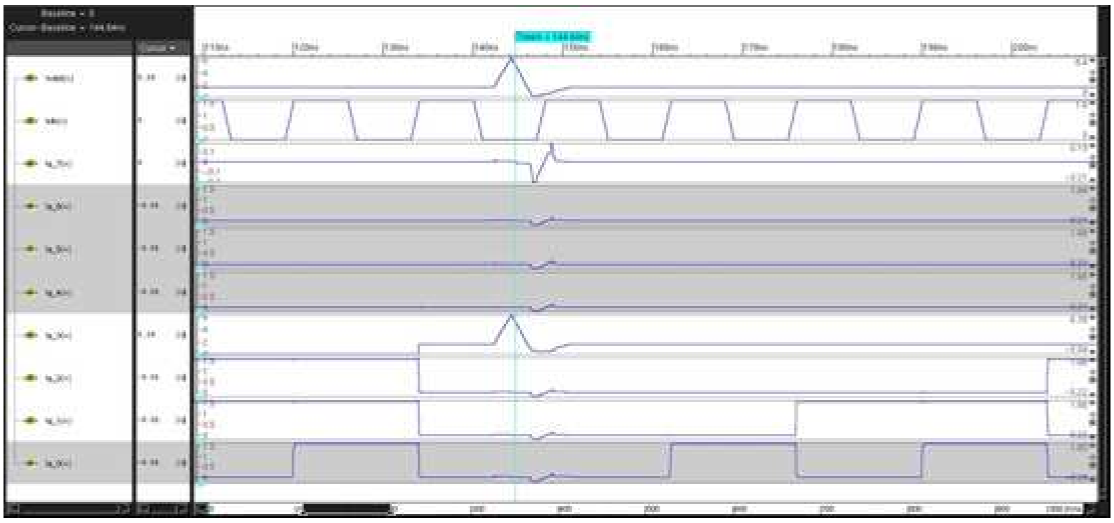


Figure 38 Design's response to Pulse 7

Just before the pulse, the circuit's output is 0x08. After the pulse it should be 0x09, however, it remains 0x08. This could be interpreted as Q0 being the only affected output, however checking the circuit's previous responses to other pulses, it seems more likely that all registers keep the same value after the pulse as they had before it.

Analysis

Looking at Figure 39 it can be seen that register Q3's initial state is logic 1 and that the register's input is also logic 1 before and after the pulse. While the pulse is happening, however, the register's input becomes logic 0. This phenomenon has been previously explained. When the register's input becomes logic 0, this change is forwarded up to net24 and net108, which take logic 0 and logic 1 values respectively. When VDD is falling, n2 has its first singularity, which disables the register's input N22 and takes the register's output as its input. This results in net24 and net108 changing their logic values again, to logic 1 and logic 0.

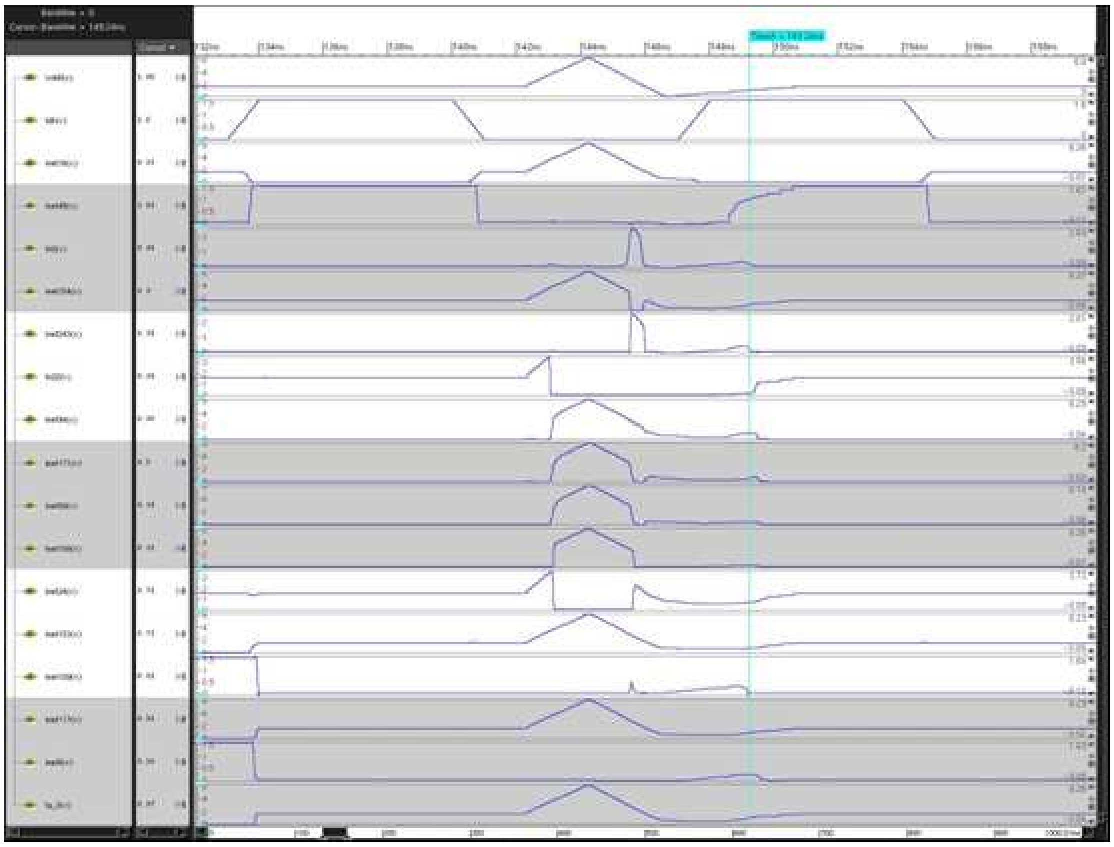


Figure 39 Register Q3's response to Pulse 7

When n2's first singularity disappears, N22 again becomes the register's input and as such it would be expected that the internal nets would change according to this new input. However, net171 and net50 only change slightly and this change is not affected on net24 or net108. Checking the speed with which net94, net171 and net50 change their logic values from 0 to high (when input N22 falls from logic 1 to logic 0), it can be seen that net94 is the fastest and net50 the slowest. This happens because net94 is driving net171 and net50 through parasitic capacitors on transistor pairs M52-M53 and M27-M26. These same capacitors slow down the voltage rise on net171 and net50. Despite n2's first singularity finishing when VDD is still 1.62V and net94 has 1.22V, both are falling and cannot increase the voltage on net171 and net50 fast enough. Finally, by the time VDD goes under 0.96V, net171 has reached only 0.83V and net50 0.49V, which is not enough to trigger a change on net24's value, which remains at logic 1 when it should have changed. As VDD decreases, net171 and net 50 also decrease their voltages.

By the time VDD becomes 0V, net94 has 0.55V, net171 0.53V, net50 0.39V, net24 0.87V and net108 0V. net153 has 0.72V, net8 -0.05V, net117 0.67V and Q3 0.54V. As explained previously, as VDD starts to rise, those nets whose voltage is lower than VDD start to raise their voltage, while those nets whose voltage is higher than VDD follow a trend to reduce their voltage. net24 has 0.84V, which polarises M50 and keeps net108 to 0V for as long as net24 does not go below this voltage and delays VDD's increase effecting these two nets.

When VDD reaches 0.64V, no transistor is conducting and all nets increase their voltage only as an effect of VDD rising. It is not until VDD has 0.9V that switch transistor pairs start to conduct and transistors in general start to be polarised. At this point the register's internal circuit starts to be the circuit shown in Figure 14. The register's input no longer has an effect on net24, which from this point on evolves depending only on the values of net50, net108 and net24, which are 0.36V, 0.31v and 0.64V.

As VDD rises over 1V, net117 rises over 0.85V, polarising transistor M61 and forcing net109 to 0V, and both net50 and net24, raise their voltage values, net24 being the one that reaches the threshold voltage first. As soon as net24's voltage goes over 0.75V, net108 starts to decrease its voltage and this starts to pull net50 down; in return, net24 is pulled up faster. As net24 rises, net153 also rises. Again, when net153 goes over 0.75V, net8 starts to decrease, ensuring that Q3 and net117 are kept at logic 1. From this point on, all signals follow the trend to set the register's output to logic 1.

Q0's response to Pulse 7, Figure 40, as well as for Q3, is to keep the same logic value before and after the pulse. In this case, the register's input just before the pulse is logic 1 (i.e. the logic value to which the register would have been set on the next clocks positive edge if no pulse had been applied) also falls to logic 0 as VDD rises over 3.59V. When the input falls to logic 0, it affects nets net94, net171, net50, net24 and net108 and they also change their logic values. At this point the register's input and output have the same logical value, so when n2's first singularity happens, it has no effect on previous nets and so they all keep the same logic values.

While VDD is decreasing, all logic 1 nets follow this trend more or less as fast as VDD, until VDD becomes less than 0.8V, where the nets' voltages decrease until the nets' voltages go under 0.8V too, where their rate of voltage decrease becomes even slower.

Again, when VDD starts to rise, all nets whose voltages are lower than VDD also start to rise. Those nets whose voltage is higher than VDD continue to decrease. And when VDD is higher than all nets' voltages, all nets increase their voltage. When net8 goes over 0.73V (at VDD 0.83V), Q0 and net117 change their tendency to rise with VDD and start to decrease their voltages. As VDD continues to rise and the clock's positive edge is detected, the register's internal circuit looks like Figure 14, net50 and net108 continue to rise, going over the threshold voltage and stopping net24's and net153's voltage rise, starting to decrease their voltage values and ensuring the register's output is set to logic 0.

This way, this register too, keeps the same logic value as it had before the pulse.

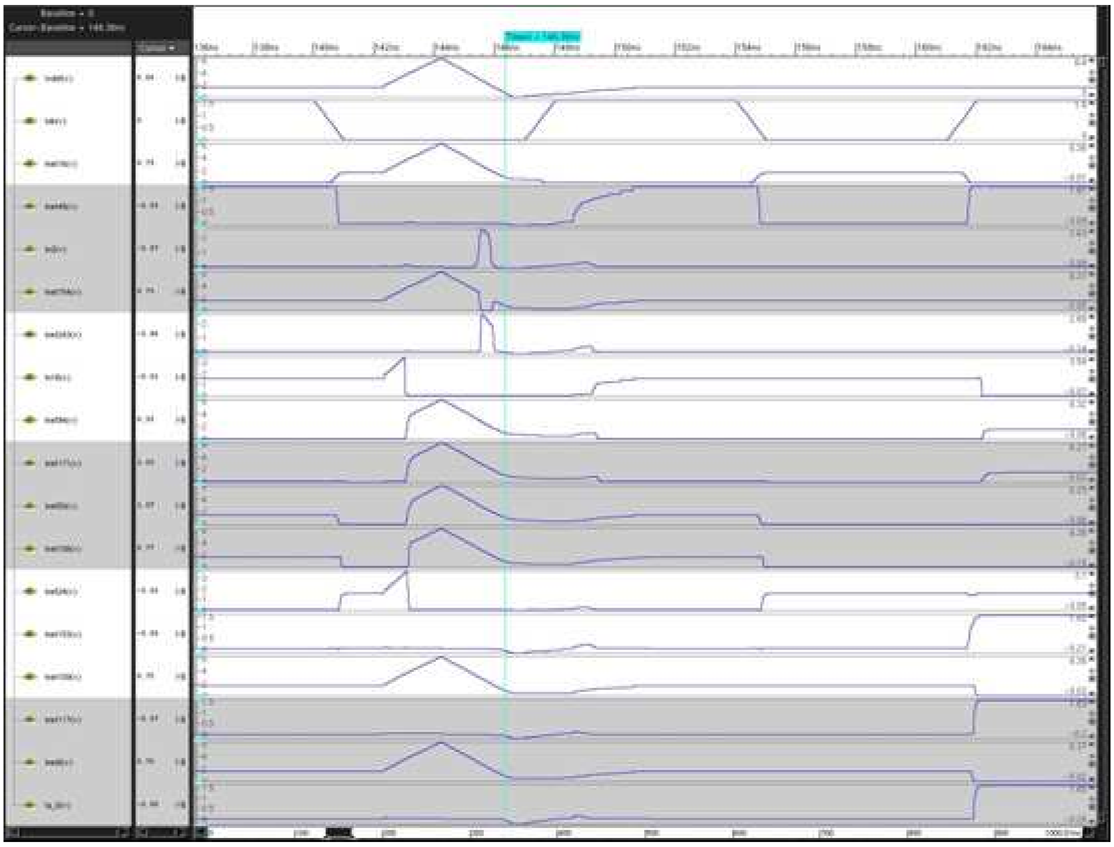


Figure 40 Register Q0's response to Pulse 7

Pulse 8

This is the last pulse that has been applied to this design. It starts, at 143.35ns, to rise from 1.6V to 3.2V, which is reached at 145.2ns. At this point it falls down to 0.8V at 147.6ns, rising to 1.6V at 151.6ns. This pulse is the same as Pulse 7 only it starts 1 nanosecond later and VDD reaches a maximum of 3.2V and a minimum of 0.8V, instead of 6.4V and 0V. These are the parameters to create this pulse:

```
VOLTAGE=1.6
DELAY_FOR_PULSE=140.0n
PULSE_START_TIME=3.35n
PULSE_START_VALUE=0
PULSE_P1_TIME=5.2n
PULSE_P1_VALUE=1
PULSE_P2_TIME=7.6n
PULSE_P2_VALUE=-0.5
PULSE_END_TIME=11.6n
PULSE_END_VALUE=0
```

As can be seen in Figure 41, this pulse has no effect on the circuit's overall behaviour. Although register Q0 needs about a nanosecond to start to set its output. Registers Q0 and Q3 could provide interesting reading, therefore these are the only ones to be analysed with regard to this pulse.

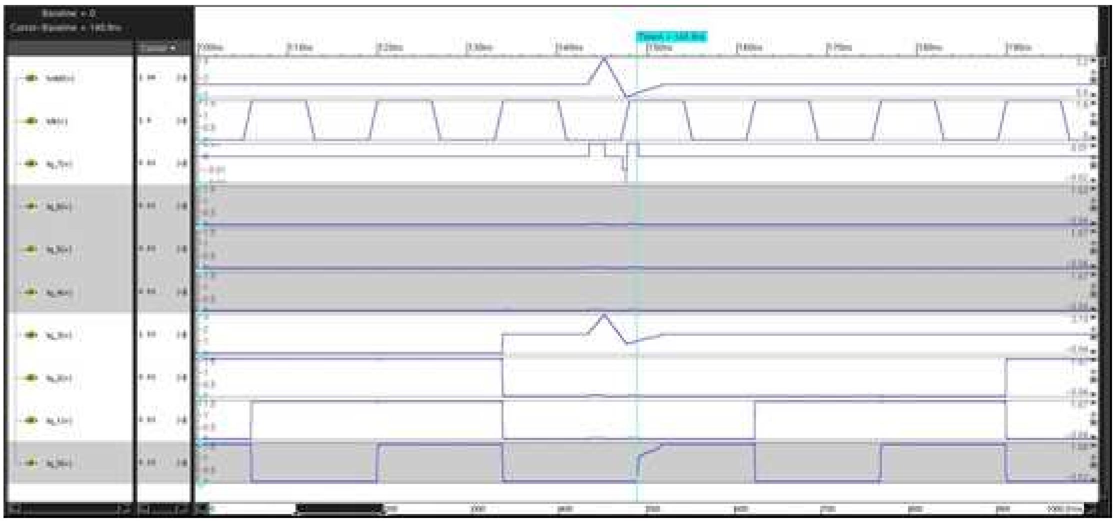


Figure 41 Design's response to Pulse 8

Analysing Figure 42, it can be seen that register Q3's input falls to logic 0 when VDD rises over a certain value. This time, however, n2's singularity happens at the same time. So, when n22 falls to logic 0, net94 rises to logic 1, starting to induce a voltage rise on net171 and net50. But as n2's singularity happens, the register's input is quickly feedback with its own output, i.e. net109 gets connected to net171, instead of n22. As net109 is logic 0, net171 and net50 stop increasing their voltages and are set as logic 0 again. Just when n2's singularity disappears, input n22 starts to recover its original value, i.e. logic 1, and the register works normally again. These changes are minimally reflected on net24, and net108 is not affected at all. The result of all this are the pulses found on net171, net50, net24 and net109.

When VDD reaches its minimum, i.e. 0.8V, all logic 1 nets keep a voltage of around 0.8V, whereas all logic 0 nets have a voltage of around 0v. As VDD starts to recover, logic 1 nets rise with VDD and logic 0 nets remain at logic 0. Additionally, during VDD's rise, the clock's positive edge is detected, reasserting the register's output to a logic 1 value.

This pulse, then, has no effect on this register's behaviour. The register's output would be affected only if net24's logic value is inverted at the same time that a clock positive edge is detected. The minimum voltage on net24 during n2's singularity is 1.1V, which is still high enough to be considered logic 1, making it difficult to affect the register's behaviour.

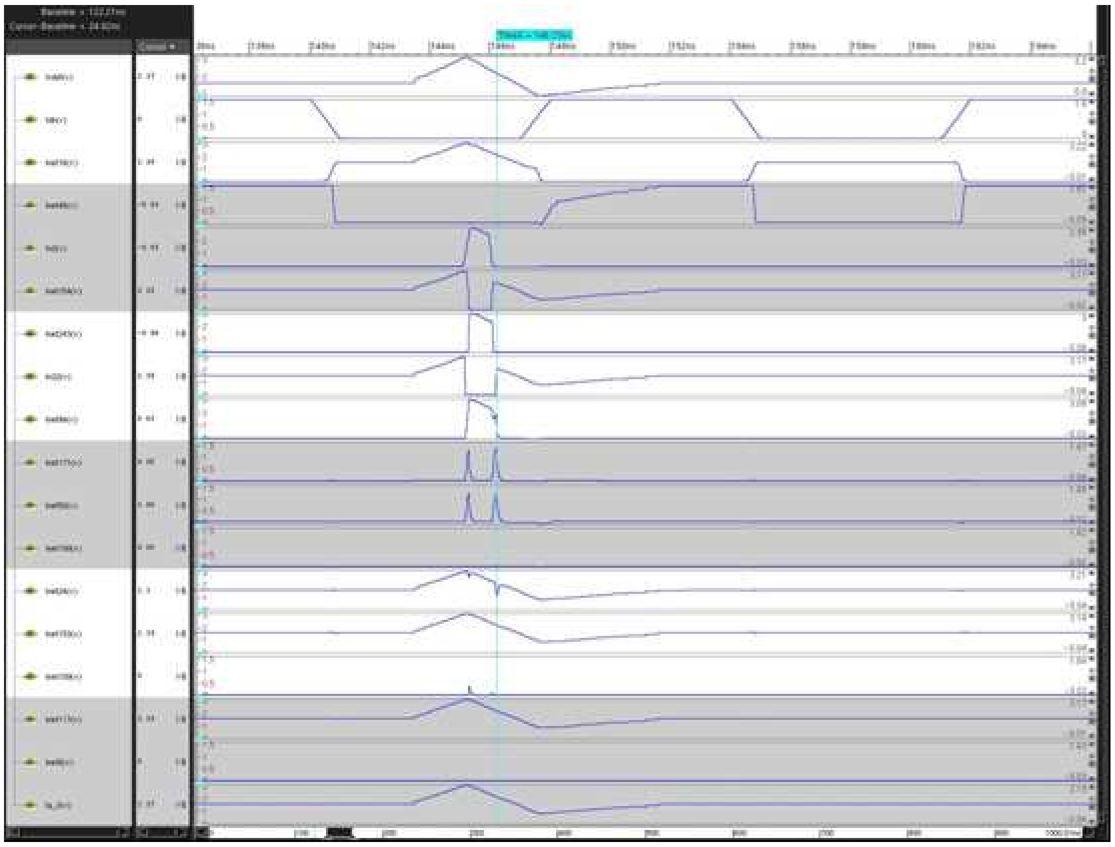


Figure 42 Register Q3's response to Pulse 8

Figure 43 shows a very similar waveform for the Q0 register's input, n16, and the n2 input. However in this case, when the n2 singularity happens, the register's input is feed with its own output and since both net109 and net94 are logic 1. This does not reverse the n16 falls' effect on nets net171, net50, net24 and net108. When n2's singularity disappears and the register's input recovers to its original value, VDD is still higher than the nominal value, giving enough time for internal signals to be set to the new input conditions.

When VDD rises, a positive edge is detected on the clock, connecting net24 (logic 1) with net153 (logic 0) and inducing a change to net153 to become logic 1. When it reaches over 0.8V, it is translated by switching net8, net117, net109 and Q0's logic values; thus setting the register to the new value, logic 1.

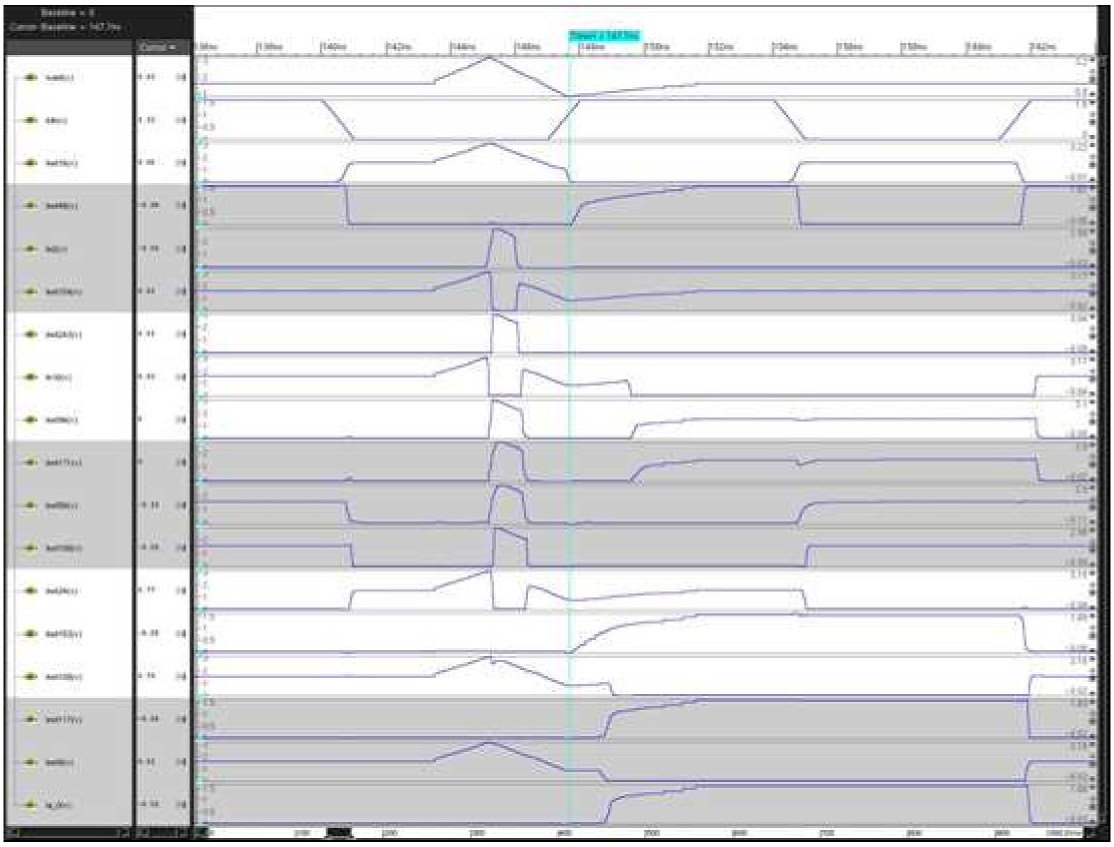


Figure 43 Register Q0's response to Pulse 8

Chapter 4 – Conclusions

Before starting to draw any conclusion, a few parameters or settings of these simulations should be remembered. At the time of making these simulations, Nanosim's feature to model the transistor's leakage was unknown. Leakage was modelled by putting resistors in parallel to each transistor (i.e. between drain and source pins). This leakage feature depends on several factors, among them on W/L parameters. However, as calculating an individual leakage equivalent resistor for each transistor would require a huge effort, all resistors are of the same value.

Something else to consider when setting up the simulation environment, is that real silicon designs do not distribute their power supply equally throughout the whole design, because power lines are not ideal, they have resistivity. This is a feature not simulated here because mathematically the design would become far more complex and it would take far longer to simulate.

Because of this, pulses applied to these simulations are far more aggressive or extreme than those applied on real attacks. In fact, the pulses applied to the simulations could physically damage the device.

On real devices, pulses can be applied to either only the power pins or the power pins and the I/O signals. Both attack approaches have potentially different effects on the I/O pins' internal nets and the logic connected to them. Either way, both would impact the on-chip I/O level shifters that convert off-chip voltage levels to on-chip voltage levels. In the worst case scenario, when the I/O signals are not altered, the level shifter might interpret a logic high as a logic low.

Here, the simulation of level shifters has been avoided and, hence, the pulse has been applied only to VDD, leaving input signals unaffected by the pulse. Despite the input signals not being affected by the pulse, these simulations help us gain a better understanding of what happens inside when a pulse is applied.

These simulations show that, as mentioned in Chapter 2, a pulse affects combinational blocks only temporarily, whereas non-combinational blocks could be affected more permanently. They also show that a clock's positive/negative edges are the best points to apply these pulses.

When analysing register internals, it can be seen that registers work on both clock edges, one to load the input data and the other to update the register's output. These simulations have shown that a register's output can be more easily corrupted if the input data is modified/corrupted just before the register updates its output. Registers used in these simulations load the input data on clock negative edges and update their outputs on clock positive-edges, making them, therefore, more vulnerable if the pulse happens close to the positive edge. Obviously, when a register's output is corrupted, devices that make use of that output are going to be affected.

Before starting the simulation it was expected that with the same pulse, different behaviours on different registers would be observed. However, in each simulation, all registers responded equally. In one case all registers were set to logic 1, in some cases all registers were set to logic 0, in another case, after the pulse, all registers kept the same logic value as before the pulse, and in some cases, pulses did not affect the circuit's behaviour. The fact that all registers follow the same pattern can be explained with three factors: one the factors that would help to get this outcome is not modelling the power line's parasitics; another factor would be the fact that all registers are symmetrical, where their inputs depend on the same combinational cell, although it is true that an input of these cells is different from one to another; and the last factor would be fact that design's input voltages do not change with VDD.

If power parasitics had been modelled and simulated, different cells (e.g. registers) would be powered at different voltages and therefore not all registers would be affected in the same way. It also would help to prove the statement made in Chapter 2, which says that VDD changes could affect different cells' v_{oh}, v_{ol}, v_{ih} and v_{il} parameters in different ways.

In these simulations, transistors have been switching their working regions from saturation to linear and cut-off regions. When VDD goes below 0.8V (inverter threshold voltage), the transistors' parasitic parameters can play an important role by keeping the nets' voltages stable for longer and minimising the pulse's effect. Nets net50, net24, net153 and net8 are the nets that have a major role in the register's final value. Parasitic parameters should help keeping these nets and feedback nets with the correct value at all times.

Finally, it has been proved that little differences between two pulses can have two totally different outcomes on the design's behaviour. An example of this can be pulses 5, 6 and 7. Dropping the glitch point P1's voltage value (parameter `PULSE_P1_VALUE`) by 8V at a rate of 1.7 V/ns (pulse 5) or 20 V/ns (pulse 6) is the difference between not affecting the design's behaviour and resetting the design's output to 0x00. Dropping P1's voltage value by 8V at a rate of 3.3 V/ns, on the other hand, results in skipping one count operation.

Future work

This simulation environment is work still in progress, where several things can be improved. The first thing to improve is using Nanosim's leakage model and avoiding using resistors in parallel with transistors. In order to do this, simulation accuracy should be increased, which could mean slowing down the simulation. This change should not affect the design's behaviour when applying the same pulses.

It could be interesting to analyse the design's behaviour when input voltages change with VDD. This change may affect the design's behaviour when applying some of the pulses, particularly those where the design's behaviour changes as result of a singularity on one of the signals. Applying the same to the clock signal should also be studied. Although these two settings have not been used in these simulations, the simulation environment is enabled to vary all input/stimulus signals (including clock) as VDD changes.

All pulses in these simulations have been applied to the VDD signal. Designs can also be attacked on the GND signal, particularly those with an embedded regulator where this kind of attack makes more sense. A design without a regulator is powered directly by the voltage applied to the VDD pin. A design with an embedded regulator is powered by the regulator's output voltage, and the regulator is powered by the voltage applied to the VDD pin. In the first case, a pulse applied on the VDD pin is going to be transmitted to the whole design (parasitic parameters may affect the waveform). In the second case, however, the regulator works as a filter between the VDD pin and the design itself. The regulator's response to a pulse applied on VDD would be the waveform that the design is powered with. An example of a regulator's response to a pulse is shown in Figure 3.

When a pulse is applied on GND, however, it would be transmitted throughout the whole circuit. For the case where there is not an embedded regulator, this pulse may affect the design's response in a similar way to when it is applied on VDD. But for the case where there is an embedded regulator, the same pulse on VDD and GND would have different effects on the design's behaviour. For the latter case, as the regulator has a capacitor on its output, unless this capacitor is disconnected, VDD! will tend to follow changes on GND. In the case where the capacitor is disconnected, VDD! will remain unchanged during the pulse. Hence, applying a pulse on GND to a design with an embedded regulator may have one or more effect depending on whether the regulator's output capacitor is connected or not. In order to allow simulating all these cases, applying a pulse on GND should be enabled.

Simulating a design where power line parasitic resistors and capacitors are taken into account could be interesting to compare against simulations where they have not been considered. As simulating a design with these parameters can take much longer than simulating without them, modelling only part of the power line's parasitic parameters could also be considered to reduce the simulation time.

Equally, creating a netlist from an extracted view instead of from a schematic view (the one used on these simulations) should also be tested, as the former one has additional parasitic information, such as parasitic capacitors between different nets, that are not taken into account by the latter case. This may add more accuracy to the design's response to a pulse, although probably at the cost of adding extra simulation time.

The design used in these simulations has a feedback feature, as the register's inputs are dependent on their outputs. It would be desirable to simulate a design without such feature so it is clear that any behavioural change on the design is not a result of the feedback. Also, the register's internal organisation of parameters could be changed to see the effect of a pulse on a register with these changes.

A study of asynchronous circuits would also be merited to see if they could provide protection against glitch attacks through a lock-up mechanism, which is a characteristic of the design style, and an estimate of the area penalty resulting from this approach.

Appendix A

Design's RTL code.

```
module counter (input [7:0] data,
                input clk, enable, count, load, nreset,
                output reg [7:0] q);



  always @ (posedge clk)
  begin
    if (~nreset)
      q <= 0;
    else if (enable)
         begin
           if (load)
             q <= data;
           else if (count)
                  q <= q+1;
         end
  end

endmodule
```

Engineering Doctorate

Tartalo test-chip documentation

Author: Asier Goikoetxea Yanci

Date: February 2006

Sponsored by:

Table of contents

Glossary of terms

DICE	Dual interlocked storage cell
ELT	Enclosed layout transistor
EMA	Electromagnetic analysis
ESD	Electro static discharge
RHBD	Radiation hardening by design
SEE	Single event effect
SERT	Single event resistive topology
SET	Single event transient
SEU	Single event upset
SPA/DPA	Single/differential power analysis

Chapter 1 – Introduction

Over the years, a wide range of attack techniques have been developed to crack secure systems or ICs. These attacks can be grouped as intrusive, non-intrusive, destructive, non-destructive, etc. The list below names some of the primary attack methods:

- Glitch attack (applied to power, ground or data signal)
- Simple/Differential Power Analysis (SPA/DPA)
- Electromagnetic Analysis (EMA)
- Laser/Light attacks
- Back-side emissions

Designs on this test chip will be tested against either glitch or laser attacks, which are both non-intrusive and, if correctly applied, non-destructive. The main purpose of performing glitch attacks on this test chip is to validate a previously developed glitch attack simulation environment, which simulates how a glitch in power, ground or other signals affects the device under test. The purpose of performing laser attacks is to have a better understanding of how attacking different parts of a design affects the designs' overall behaviour, and to study the effectiveness of different radiation hardening techniques or countermeasures when being targeted with a laser beam. It will also test how error detection techniques and layout rules can minimise a device's sensitivity to light attacks.

Chapter 2 –Laser and glitch test chip (tartalo/01OKA)

The following sections discuss the test-chip design, the package I/Os and test methodology.

2.2.- Designs implemented on tartalo/01OKA test-chip

The test chip is divided into two circuits: one will be used for glitch attacks, the other for laser attacks. To help with the simulation environment validation, the same design as the one previously used in the simulation environment is implemented on this test chip. This will allow comparing the behaviour in the simulation environment with the test-chip's behaviour.

As the target of laser tests is not only to understand how it affects the design but also to test different countermeasures used against radiation induced effects, several circuits with different countermeasures are implemented. These circuits will be referred to as laser test circuits (LTC).

2.2.1.- Glitch attack simulation environment design validation

The counter used in the simulation environment and implemented in the test-chip is a simple 8-bit incremental counter. Inputs to this counter are *data*, *clock*, *enable*, *reset*, *load* and *count*; and the only outputs of this counter are the values stored in the registers, *q*. This is the counter's Verilog RTL code:

```
module counter (input [7:0] data,
                input clk, enable, count, load, nreset,
                output reg [7:0] q);

  always @ (posedge clk)
  begin
    if (~nreset)
      q <= 0;
    else if (enable)
         begin
           if (load)
             q <= data;
           else if (count)
                  q <= q+1;
         end
  end

endmodule
```

Figure 1 shows the counter's design implemented in Atmel standard cells, as it has been used during simulations and implemented on this test chip. Figure 2 shows counter's symbol as used at the top level of the test chip.

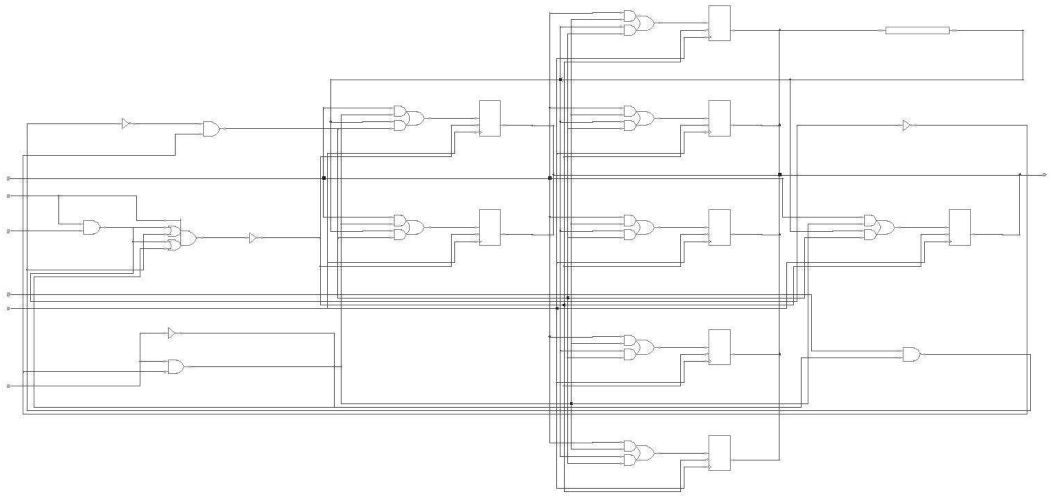


Figure 1 Schematic of the counter to be used on glitch tests

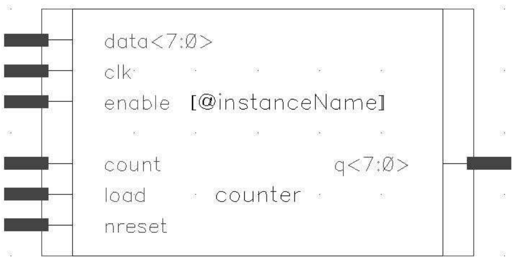


Figure 2 Counter's symbol

To better understand the circuit's behaviour when a glitch is applied and, improve the simulation environment, having access to some internal nets is desirable. However, none of the internal nets have been routed to a physical pin on the test-chip due to the limited amount of pins available in the test-chip package. Instead, if monitoring an internal net is desired, that net should be probed with the appropriate micro-probing equipment.

2.2.2.- Laser target circuits

The report "*Laser and silicon*" (now the laser attack literature review in Volume I) covers effects that radiation and lasers can induce on a device, and some countermeasures against radiation are described. However, no evidence has been found of any of these countermeasures being used to protect devices against laser attacks. Some of these radiation countermeasures are implemented on the instantiated LTCs so that their effectiveness against laser attacks can be tested.

A laser can only induce single event transient (SET) and single event upset (SEU) events on silicon. Since SEU events can have worse effects on a circuit's behaviour than SET, implemented countermeasures focus on tackling SEU events. More precisely, the implemented countermeasures are:

- Radiation hardened by design (RHBD)
- combinational SEU mitigation
- parity error detection
- covering the design with a metal layer

In addition to the above countermeasures, different register cell layouts were instantiated to observe how the layout affects the circuit's response.

All these countermeasures are implemented on different instantiations of the same simple test circuit, which includes combinational blocks and registers. This basic test circuit, shown in figure 3, is a registered two 4-bit input adder.

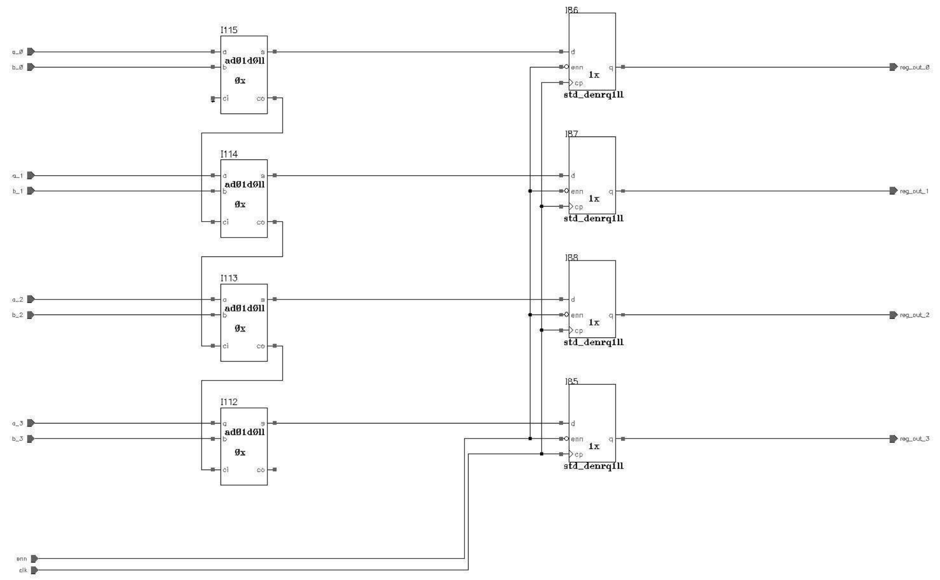


Figure 3 Basic light attack test circuit

2.2.2.1.- Different techniques to be tested on this test chip

2.2.2.1.1.- RHBD

Radiation hardening by design is being used to mitigate radiation induced upsets and is based on specific design techniques to achieve it[1]. Some of these techniques involve: using voting logic; dual interlocked storage cell (DICE)[2] latch; enclosed layout transistors[1]; or single event resistant topology (SERT) SEU immune memory cells[3].

The voting logic circuitry monitors the output of several identical blocks and chooses the more common output among them. Using this approach to make a design immune to radiation or laser implies instantiating the same design at least three times (increasing the overall area and power consumption) and designing a voting circuit to decide the final output. This is illustrated in figure 4. If this technique is used to make a block or function immune to such attack, the voting logic circuitry should still be designed to be immune against laser attacks, otherwise an attacker could target it and by-pass all benefits provided by this technique. Due to the area and power penalties resulting from this technique, it has been discarded from the scope of the research.

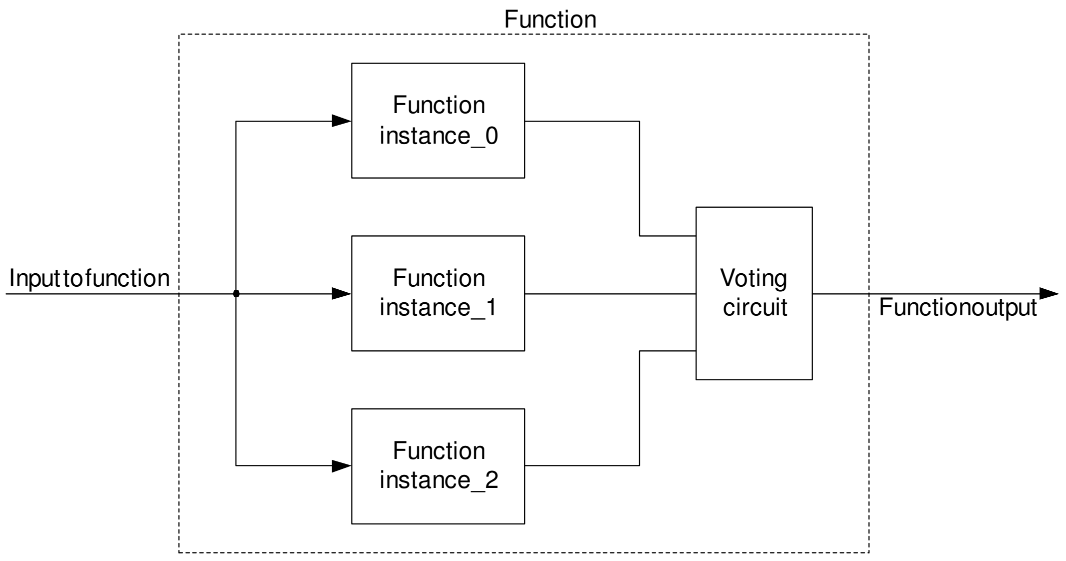


Figure 4 Voting circuit example

In normal transistor operation, the current should flow from the source to the drain, however, a transistor under radiation effects leaks current through the edges[1]. Enclosed layout transistors (ELTs), figure 5, are edgeless transistors that, together with a guard ring, minimise or eliminate the leakage current introduced by radiation. This layout style means, these transistors are bigger and more power hungry than normal ones. This technique is being tested on this test chip to check how they perform under a light attack, whether it is in sequential or combinational logic.

DICE and SERT SEU radiation hardened registers have been designed with SEU immunity in mind. Since SERT SEU, figure 6, has been designed for low power, as no nodes are going to be driven to high and low at the same time, this is the implementation chosen for this test chip.

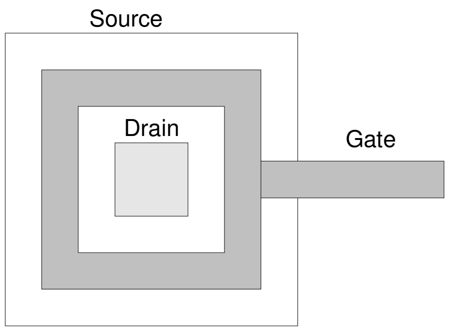


Figure 5 Example of an enclosed layout transistor (ELT)

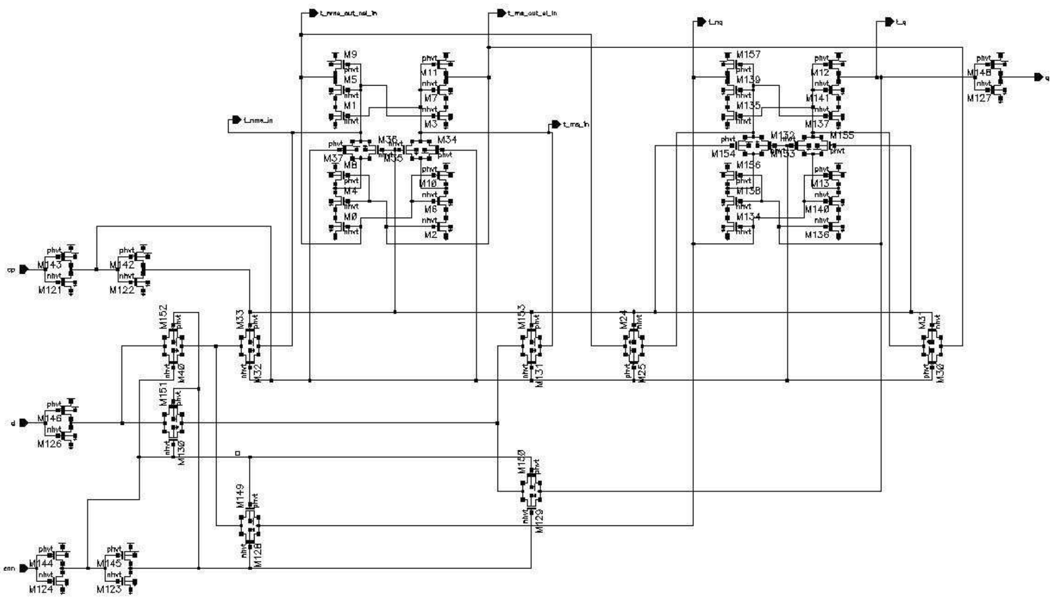


Figure 6 Radiation hardened register

2.2.2.1.2.- Combinational SEU mitigation

All instances will include a technique proposed in [4] to mitigate SEU on combinational blocks. This technique, shown in figure 7, compares a combinational block's output value with a delayed version of itself. As long as both are the same logical value, the comparison circuit's output takes the correct value. When an SEU is induced on the combinational circuit, inputs on the SEU mitigation circuit's comparator will initially differ. In this case, the comparison circuit's output is not driven, holding its output value due to the parasitic capacitance at its output. If the SEU lasts longer than the delay time, both the targeted output and its delayed version will be corrupted and the comparison circuit will propagate this error.. As with the voting technique circuit, the output circuitry in this approach is also sensitive, and an attack to this part of the design might propagate an error straight away. Furthermore, a simultaneous attack on the combinational block and delay circuit could reduce the mitigation time.

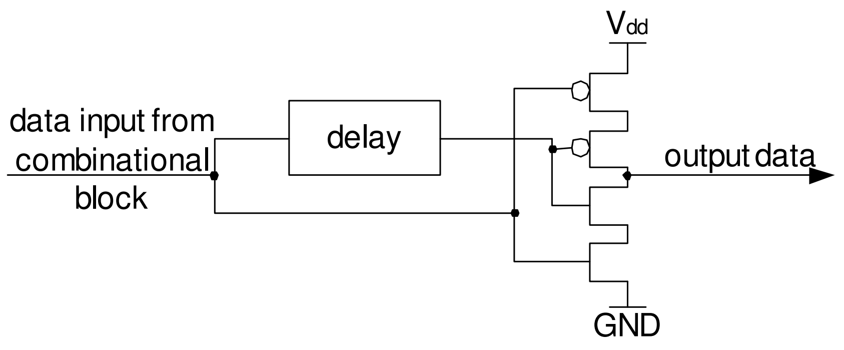


Figure 7 Combinational block SEU mitigation approach

An SEU mitigation combinational block is located between the adder circuit's output and registers' inputs, as shown in figure 8.

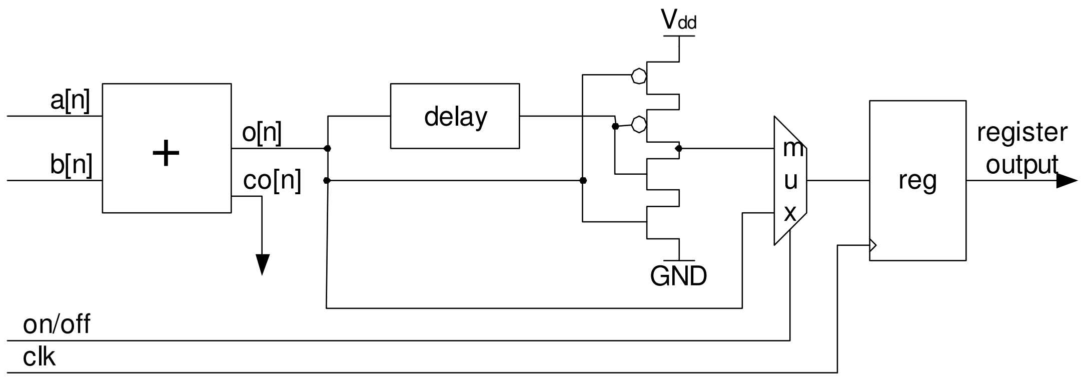


Figure 8 Combinational block SEU mitigation implementation

2.2.2.1.3.- Parity

Several error detection techniques (parity, CRC, etc.) can be used to detect an error or corruption of data being transmitted between two nodes, such as two computers or a CPU and a peripheral. Each technique has a different degree of accuracy and is therefore able to detect more or less errors.

When a node or peripheral unit is used to store information, storing not only the information but also the error detection code might be desired, so errors on stored information can be detected. By taking this same approach at a lower level, data stored into a set of registers could be accompanied by error detecting code with the aim of detecting any corruption of the stored information, be it a human induced error or not. An error detection circuit at the registers' output could analyse their contents and trigger an alarm when an error is detected.

For this case of study a simple error detection technique has been chosen, a parity scheme, which is shown in figure 9. Despite its limitations for error detection, its low storage overhead and the simplicity of the implementation makes it easier to test. The XNOR gate generates the parity value of the 4-bit adder and it is stored into a register together with the adder's output. An additional XOR gate checks the values stored in the registers. Under normal circumstances an odd number of ones will be stored in the registers and the XOR gate's output will be high. However, if for any reason the registers hold an even number of ones, the XOR gate's output will change to low, indicating an error.

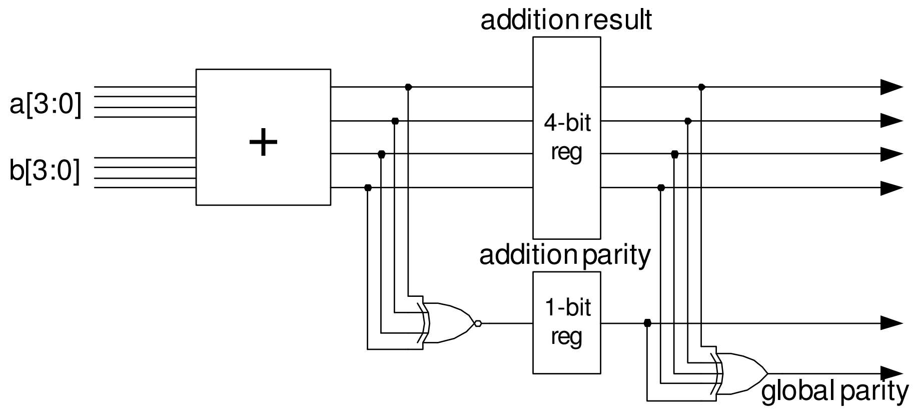


Figure 9 Error detection with parity scheme

This technique is included in all instances of the circuit on the test chip.

2.2.2.1.4.- Cover with a metal layer

Since a laser beam that is set to induce an SEU on a transistor without permanently damaging it cannot pass through metal layers, one of the circuit instances will be covered with a metal layer to measure the effectiveness of the metal layer at blocking the laser.

2.2.2.1.5.- Different layouts

A laser attack's success depends not only on the timing, i.e. when a specific transistor is exposed to light, but also on which transistors can be targeted individually or as a group. This is highly affected by the physical layout. Registers in different instantiations will be arranged in different ways to analyse how the laser affects different arrangements. Figure 10 shows three different layouts used, where the dot is used to indicate a register's orientation

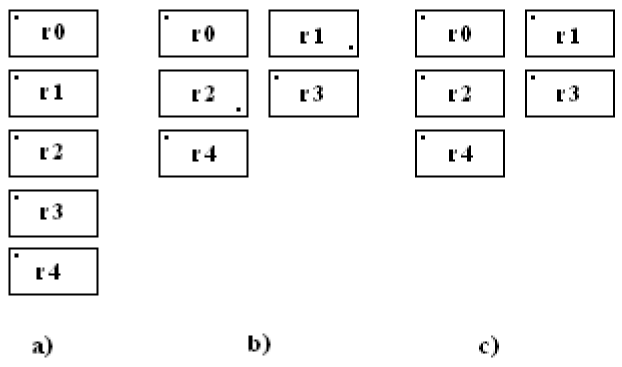


Figure 10 Different layouts used on the test-chip

2.2.3.- Test chip construction

In order to help analyse what happens within a register when it is exposed to a laser, some internal nets will be monitored, by taking them outside of the chip. Figure 11 shows a schematic where all the previously mentioned measures are in place. Since there are not enough available pins on the chip to route out each register's internal nets, they are multiplexed so only information related to one register can be accessed at a time.

Taking the design in figure 3 as a basis for each instantiation, 6 different flavours of this design have been implemented. Three different instances are implemented with different register layouts. The three modules designed using standard cells are named *ltc_std_cell_lyt_a_0*, *ltc_std_cell_lyt_b*, and *ltc_std_cell_lyt_c*, where letters *a*, *b* and *c* represent the register layout implemented from figure 10. In addition to these, a copy of module *ltc_std_cell_lyt_a_0*, named *ltc_std_cell_lyt_a_1*, is implemented with a metal layer on top of it. One module is instantiated with ELT transistors instead of standard ones. This module is named *ltc_elt_cell_lyt_a*. The last laser test circuit, *ltc_rh_cell_lyt_a*, uses standard cells except for the registers, which are SERT SEU immune.

The standard and ELT cells laser test circuits' symbol is shown in figure 12. The laser module with SERT SEU immune registers needs more visibility of internal nets compared to standard cell circuits. Its symbol is shown in figure 13.

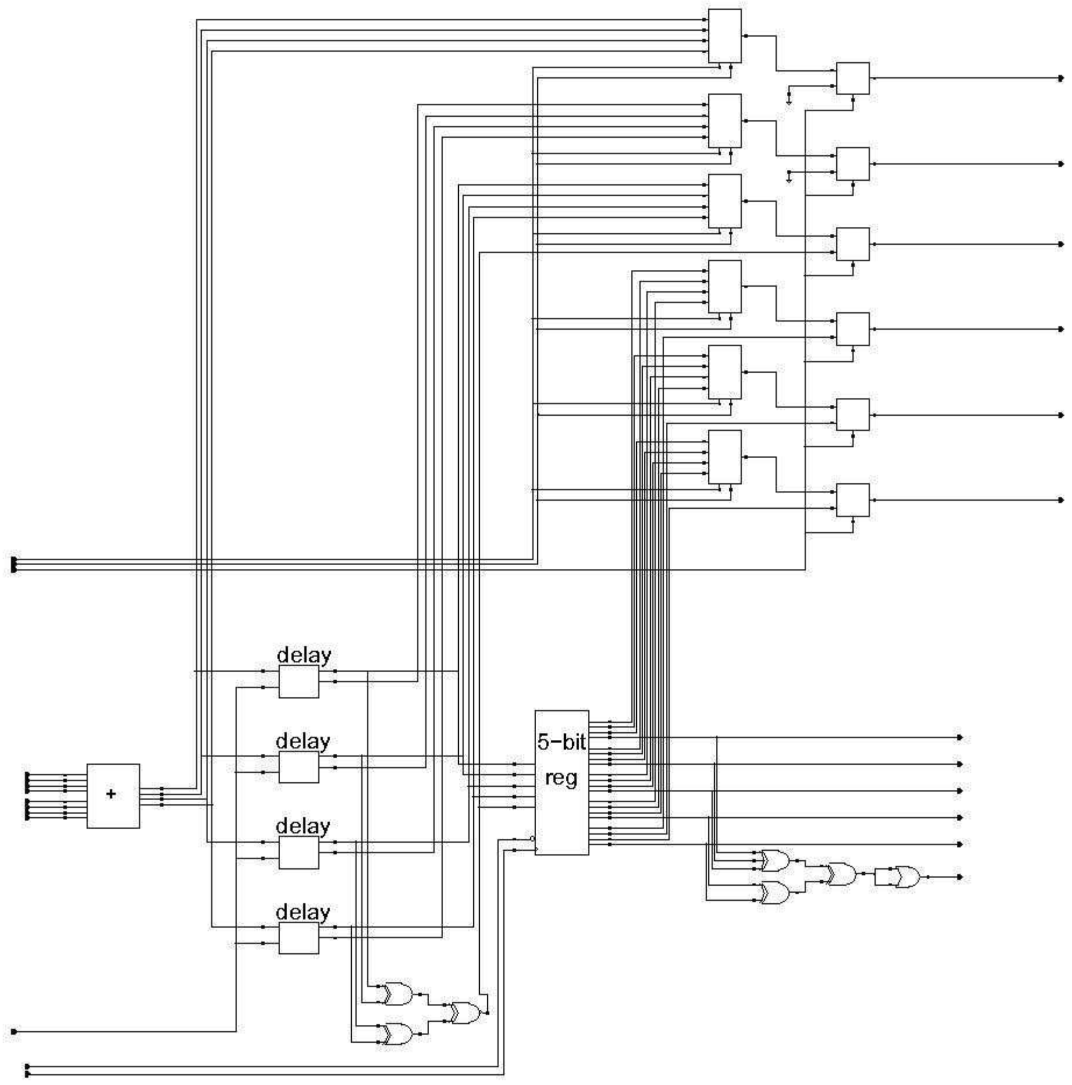


Figure 11 Schematic of a laser test circuit

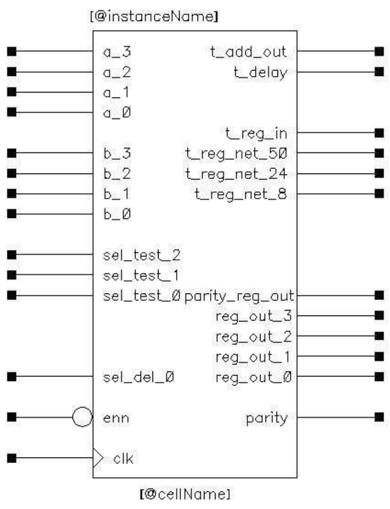


Figure 12 Standard and ELT laser test circuit module symbol

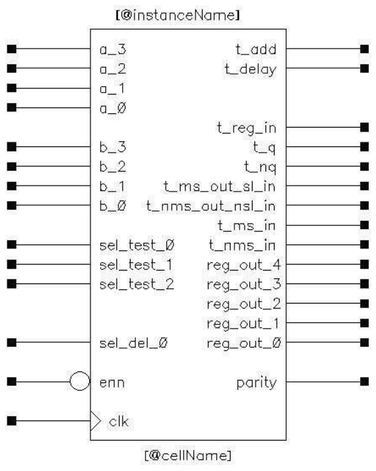


Figure 13 SERT SEU laser test circuit module symbol

Again, as all these modules require more pins than allowed by the test-chip package, all modules' outputs are multiplexed. Hence, only one laser test circuit can be tested at a time. For the same reason, the outputs from the XOR gates have not been forwarded to a pin on the chip.

The 8 bit counter inputs/outputs are also limited by pin restrictions. *Data input* bus and the control input *load* have been grounded, control inputs *enable* and *count* have been set to Vdd.

Different Vdd pins for the laser circuits and the simulation validation circuit have been put in place, so when testing the counter against glitch attacks, the laser circuit will not affect the results.
Finally, the test-chip's top level schematic is shown in figure 14.

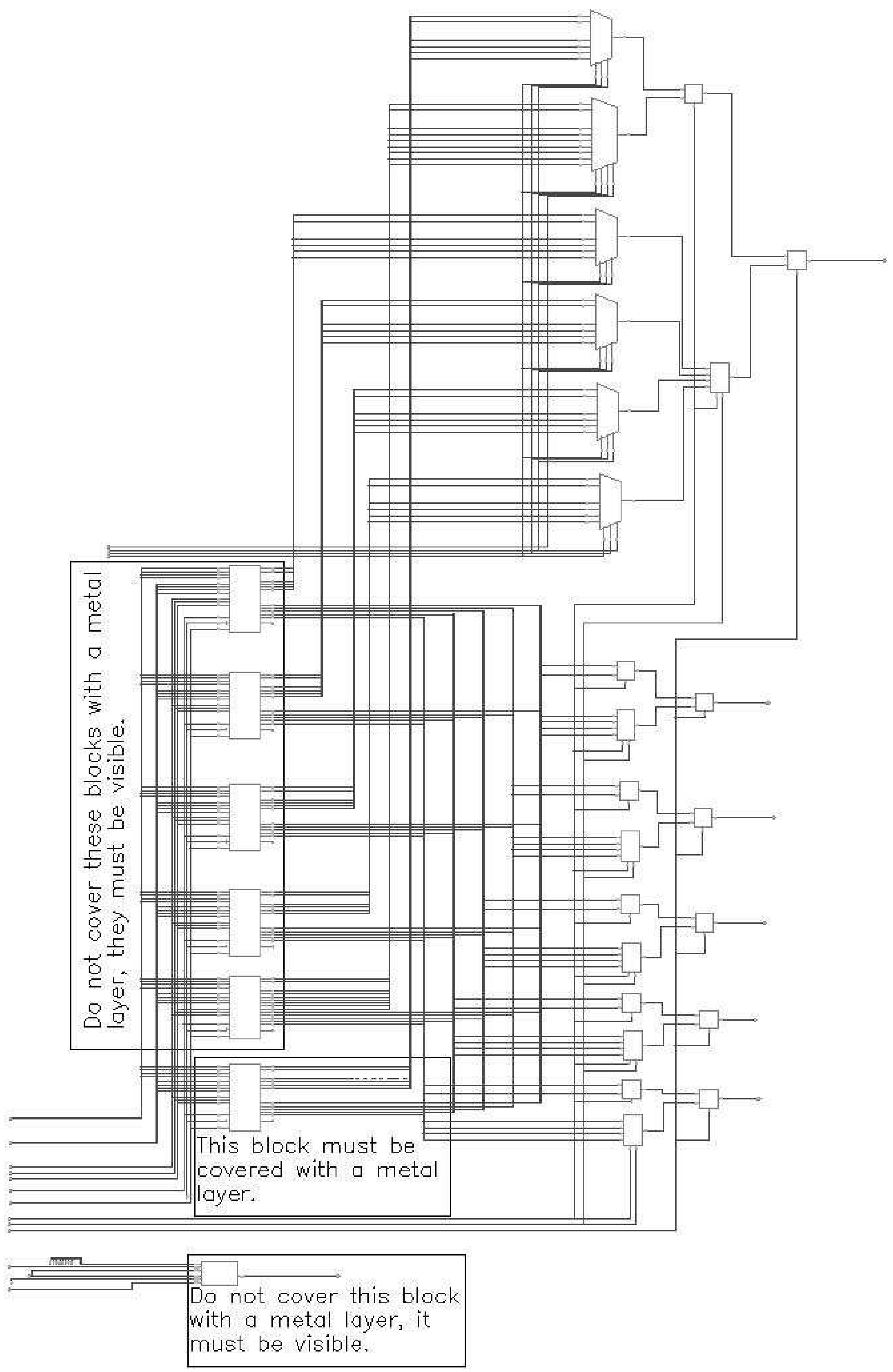

Do not cover these blocks with a metal layer, they must be visible.
This block must be covered with a metal layer.
Do not cover this block with a metal layer, it must be visible.

Figure 14 Testchip top level schematic

2.3.- Test-chip pin names

The test-chip 01OKA targeted two projects, the glitch and laser circuits covered in this report and another project not related to this research. This test-chip was assembled on a PLCC84 package. Out of the 84 available pins, 38 were allocated to this project. The remaining 46 pins were allocated to the other project. The pin out for this project is listed in table 1.

Table 1 Test-chip pin out

Pin number	Pin name	Type	Design	Description
33	clk_ctp	input	counter	Clock signal
34	nreset_ctp	input	counter	Reset signal
44	q_ct_7p	output	counter	Counter’s MSB output
41	q_ct_6p	output	counter	Counter’s output
40	q_ct_5p	output	counter	Counter’s output
39	q_ct_4p	output	counter	Counter’s output
38	q_ct_3p	output	counter	Counter’s output
37	q_ct_2p	output	counter	Counter’s output
36	q_ct_1p	output	counter	Counter’s output
35	q_ct_0p	output	counter	Counter’s LSB output
42	vdd_ct	power	counter	Counter design’s Vdd pin
48	a_ls_3p	input	laser test	MSB input to the adder
47	a_ls_2p	input	laser test	input to the adder
46	a_ls_1p	input	laser test	input to the adder
45	a_ls_0p	input	laser test	LSB input to the adder
52	b_ls_3p	input	laser test	MSB input to the adder
51	b_ls_2p	input	laser test	input to the adder
50	b_ls_1p	input	laser test	input to the adder
49	b_ls_0p	input	laser test	LSB input to the adder
56	sel_register_2p	input	laser test	MSB to select a register within each module
55	sel_register_1p	input	laser test	select a register within each module
53	sel_register_0p	input	laser test	LSB to select a register within each module
57	sel_delayp	input	laser test	Adder to register delay selector
58	clk_lsp	input	laser test	Laser modules’ clock signal
61	sel_module_2p	input	laser test	MSB to select laser module
60	sel_module_1p	input	laser test	select laser module
59	sel_module_0p	input	laser test	LSB to select laser module

27	sel_net_3p	input	laser test	MSB to select an internal net to monitor
26	sel_net_2p	input	laser test	select an internal net to monitor
25	sel_net_1p	input	laser test	select an internal net to monitor
24	sel_net_0p	input	laser test	LSB to select an internal net to monitor
62	Internal_netp	output	laser test	Internal nets' output
63	parity_bitp	output	laser test	reg_out[3:0]'s odd parity (xnor)
31	reg_3p	output	laser test	MSB register's output
30	reg_2p	output	laser test	register's output
29	reg_1p	output	laser test	register's output
28	reg_0p	output	laser test	LSB register's output
43	vdd_ls	power	laser test	Laser test module's Vdd pin

2.4.- Test methodology

This test-chip has a total of seven circuits, one to be tested against glitch attacks and the remaining six to be tested against light attacks. Only one circuit will be tested at a time and only circuit(s) targeted for attack will be powered, i.e. when applying glitch attacks, only the counter circuit will be powered, keeping all laser test circuits not powered.

A test board has been developed so circuits in the test-chip can be tested against the attacks mentioned in this document. This board allows stimulus signals to be provided and monitoring outputs to analyse the effects of a particular attack on them.

2.4.1.- Test board

A test board has been designed to test the test-chip. It is connected to a megaAVR-starter kit, which can control laser modules as well as the counter circuit for glitch attack. As the AVR and test-chip will be working at different voltage levels, the test board includes some simple level shifters. Several jumpers allow the AVR to be connected to laser circuits or the counter. It is also possible to monitor the test-chip's outputs with an oscilloscope while still reading them with the AVR. Figure 15 shows a diagram of the test board connected to the AVR board.

Two pins are not connected to the AVR: input *sel_delayp*, which can only be set with a jumper on the test board, and output *internal_netp*, which can be monitored with an oscilloscope. All remaining pins are connected to the AVR as shown in Table 2.

Every circuit in the Tartalo test-chip was powered with the voltage regulator embedded in the arrano test-chip (01VGA).

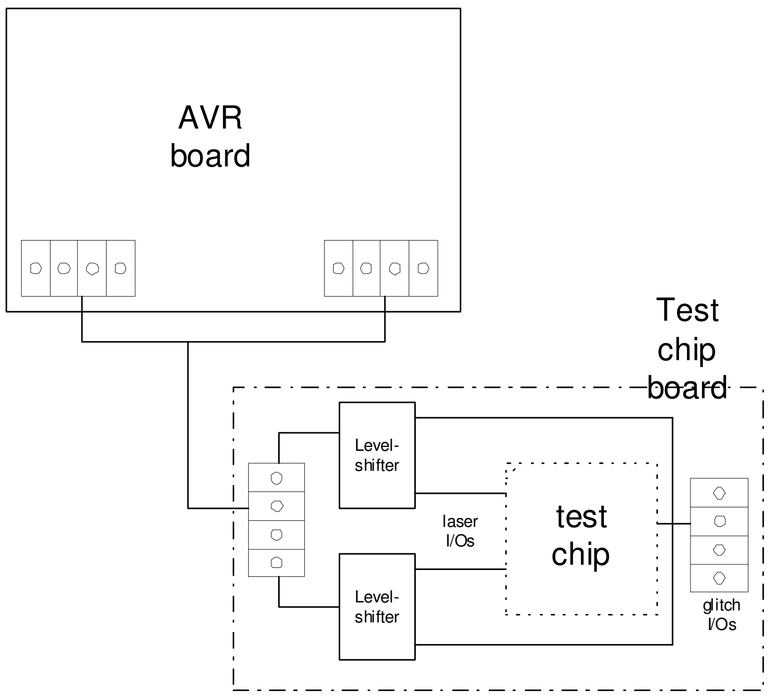


Figure 15 Test board diagram

Table 2 AVR ports functions

AVR ports	Function laser	Function counter
A:7	sel_module_2p	q_ct_7p
A:6	sel_module_1p	q_ct_6p
A:5	sel_module_0p	q_ct_5p
A:4	parity_bitp	q_ct_4p
A:3	reg_3p	q_ct_3p
A:2	reg_2p	q_ct_2p
A:1	reg_1p	q_ct_1p
A:0	reg_0p	q_ct_0p
B:7	LED	LED
B:6	LED	LED
B:5	LED	LED
B:4	LED	LED
B:3	LED	LED
B:2	LED	LED
B:1	LED	LED
B:0	LED	LED
C:7	a_ls_3p	-
C:6	a_ls_2p	-
C:5	a_ls_1p	-
C:4	a_ls_0p	-

C:3	b_ls_3p	-
C:2	b_ls_2p	-
C:1	b_ls_1p	-
C:0	b_ls_0p	-
D:7	switch	switch
D:6	switch	switch
D:5	switch	switch
D:4	switch	switch
D:3	switch	switch
D:2	switch	switch
D:1	switch	switch
D:0	switch	switch
E:7	sel_net_3p	-
E:6	sel_net_2p	-
E:5	sel_net_1p	-
E:4	sel_net_0p	-
E:3	sel_register_2p	-
E:2	sel_register_1p	-
E:1	sel_register_0p	nreset_ctp
E:0	clk_lsp	clk_ctp

2.4.2.- Glitch attacks

The current report focuses on laser countermeasures. Please refer to the "Glitch Attack Simulation Environment" for details on the glitch attack silicon test methodology.

2.4.3.- Laser attacks

The laser test circuits will be tested to check how the implemented countermeasures respond to a light attack. All circuits or modules will be simultaneously stimulated by the microcontroller, however only one module will be targeted at a time and its outputs monitored. The module to be tested will be selected with input pins *sel_module_0p*, *sel_module_1p* and *sel_module_2p*. Table 3 shows the module selection values.

The initial aim was to target individual transistors and groups of transistors in the combinational area and registers. The fibre glass coating the test-chip was finished with covers all transistors and does not allow carrying out the tests as intended, see Figure 16. Instead, the whole combinational logic will be targeted at once. Registers will be tested in two ways:

- all registers belonging to the same LTC will be targeted together, and
- all registers belonging to all LTCs will be targeted together .

Table 3 Module selection

sel_module_2p	sel_module_1p	sel_module_0p	selected laser module
0	0	0	*ltc_std_cell_lyt_a_1*
0	0	1	*ltc_rh_cell_lyt_a*
0	1	0	*ltc_std_cell_lyt_a_1*
0	1	1	*ltc_rh_cell_lyt_a*
1	0	0	*ltc_std_cell_lyt_a_0*
1	0	1	*ltc_std_cell_lyt_b*
1	1	0	*ltc_std_cell_lyt_c*
1	1	1	*ltc_elt_cell_lyt_a*

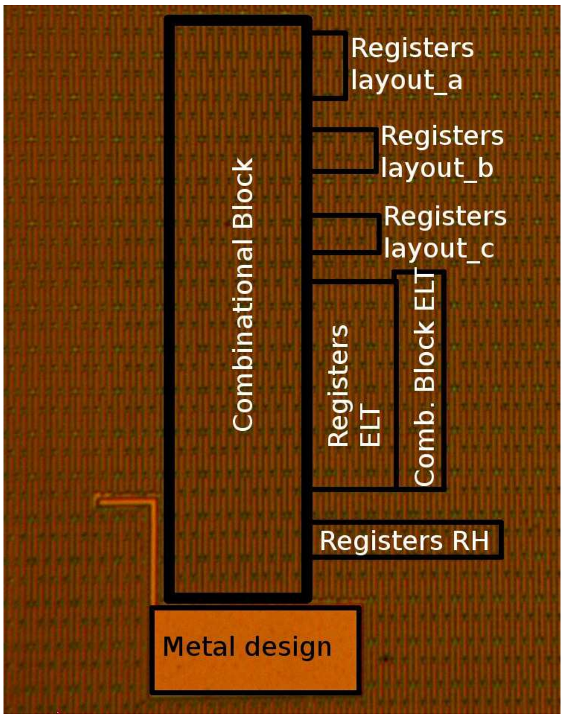


Figure 16 Top view of the laser test circuits' area

The test procedure sets the designs to continuously perform addition operations and monitoring of the results. Initially, and in order to confirm that all designs were operational, they will be tested on their own. After this initial verification, the countermeasures' effectiveness will be tested by targeting the designs with a laser. The laser will be directed at the combinational logic and registers of each design and the spot size set so that the laser will hit the whole area covered by the target block, i.e. the whole combinational block, and all the registers of the target LTC.

Combinational block tests will be carried out by applying laser pulses to them while performing addition operations, and tests on the registers carried out by applying laser pulses to them when they are holding data.

For as long as no errors are detected, the target LTC will continue to be exercised. When an erroneous output is detected, the AVR will freeze the LTC stimulus and show an error code with LEDs. Laser attacks will be triggered manually while the target LTC is being stimulated.

Only ELT transistors and the LTC with metal layer are expected to pass the tests. The SERT SEU registers are expected to fail when the laser targets more than one transistor. Any attack on the combinational block or its countermeasures is expected to produce an SEU after a given amount of time.

2.5.- Test results

The laser test results are presented here. For glitch attack results, please check the report SimEnvTech1 (Glitch Attack and Power Analysis Simulation Environment). Two laser spot sizes were applied: spot 1 (or spt1) targeting all LTCs' registers, and spot 2 (or spt2) targeting all registers of the target LTC. Table 4 shows the actual laser spot sizes. Table 5 to Table 10 show the test results per LTC. Two test-chips were tested and two energy levels applied, 75 and 150. AB represents the LTC's input parameter (e.g. 0x20 is a=2, b=0, where 'a' and 'b' are the adder inputs). Up to 200 laser attacks were applied per test. The tables below indicate how many laser attacks were needed to inject up to 6 errors (e.g. 6 in 17 means that 17 attacks were required to inject 6 errors, 0 in 200 means that no errors were detected after 200 attacks). In addition to this information, the tables below also show the energy threshold level to inject a fault. Appendix A shows the injected error values for each case. Attacks on the combinational logic always resulted in error injections.

Table 4 Laser spot sizes

Spot	X (um)	Y (um)	Description
1	307	97	targets all registers at once
2	105	105	targets the interested registers only

Table 5 Test results of the Metal module

Module	Chip	AB	Energy				Energy threshold for error injection	
			75		150			
			spt 1	spt 2	spt 1	spt 2	spt 1	spt 2
Metal	6	0x00	6 in 7	0 in 200	6 in 6	0 in 200	45	500+
		0x10	6 in 21	0 in 200	6 in 7	0 in 200		
		0x20	6 in 17	0 in 200	6 in 6	0 in 200		
		0x40	6 in 6	0 in 200	6 in 6	0 in 200		
		0x80	6 in 8	0 in 200	6 in 6	0 in 200		
	7	0x00	6 in 6	0 in 200	6 in 6	0 in 200	48	500+
		0x10	6 in 6	0 in 200	6 in 6	0 in 200		
		0x20	6 in 6	0 in 200	6 in 6	0 in 200		
		0x40	6 in 6	0 in 200	6 in 6	0 in 200		
		0x80	6 in 6	0 in 200	6 in 6	0 in 200		

Table 6 Test results of the RH medule

Module	Chip	AB	Energy				Energy threshold for error injection	
			75		150			
			spt 1	spt 2	spt 1	spt 2	spt 1	spt 2
RH	6	0x00	6 in 8	0 in 200	6 in 6	0 in 200	43	500+
		0x10	6 in 6	0 in 200	6 in 6	0 in 200		
		0x20	6 in 6	0 in 200	6 in 6	0 in 200		
		0x40	6 in 6	0 in 200	6 in 6	0 in 200		
		0x80	6 in 6	0 in 200	6 in 6	0 in 200		
	7	0x00	6 in 6	0 in 200	6 in 6	0 in 200	51	500+
		0x10	6 in 6	0 in 200	6 in 6	0 in 200		
		0x20	6 in 6	0 in 200	6 in 6	0 in 200		
		0x40	6 in 6	0 in 200	6 in 6	0 in 200		
		0x80	6 in 6	0 in 200	6 in 6	0 in 200		

Table 7 std_a test results

Module	Chip	AB	Energy				Energy threshold for error injection	
			75		150			
			spt 1	spt 2	spt 1	spt 2	spt 1	spt 2
std_a	6	0x00	0 in 100	0 in 200	6 in 6	6 in 6	54	100
		0x10	6 in 19	0 in 200	6 in 6	6 in 6		
		0x20	6 in 10	0 in 200	6 in 6	6 in 6		
		0x40	6 in 14	0 in 200	6 in 6	6 in 6		
		0x80	6 in 24	0 in 200	6 in 6	6 in 6		
	7	0x00	6 in 6	0 in 200	6 in 6	6 in 6	51	105
		0x10	6 in 6	0 in 200	6 in 6	6 in 6		
		0x20	6 in 6	0 in 200	6 in 6	6 in 6		
		0x40	6 in 6	0 in 200	6 in 6	6 in 6		
		0x80	6 in 6	0 in 200	6 in 6	6 in 6		

Table 8 std_b test results

Module	Chip	AB	Energy				Energy threshold for error injection	
			75		150			
			spt 1	spt 2	spt 1	spt 2	spt 1	spt 2
std_b	6	0x00	6 in 6	6 in 139	6 in 6	6 in 6	49	94
		0x10	6 in 6	0 in 200	6 in 6	6 in 6		
		0x20	6 in 6	0 in 200	6 in 6	6 in 6		
		0x40	6 in 6	1 in 201	6 in 6	6 in 6		
		0x80	6 in 6	6 in 189	6 in 7	6 in 6		
	7	0x00	6 in 6	0 in 200	6 in 6	6 in 6	54	94
		0x10	6 in 6	0 in 200	6 in 6	6 in 6		
		0x20	6 in 6	0 in 200	6 in 6	6 in 6		
		0x40	6 in 6	0 in 200	6 in 6	6 in 6		
		0x80	6 in 6	0 in 200	6 in 6	6 in 6		

Table 9 std_c test results

Module	Chip	AB	Energy				Energy threshold for error injection	
			75		150			
			spt 1	spt 2	spt 1	spt 2	spt 1	spt 2
std_c	6	0x00	6 in 7	3 in 200	6 in 10	6 in 6	58	94
		0x10	6 in 7	0 in 200	6 in 6	6 in 6		
		0x20	6 in 6	1 in 200	6 in 6	6 in 6		
		0x40	6 in 6	1 in 200	6 in 12	6 in 6		
		0x80	6 in 6	6 in 10	6 in 7	6 in 6		
	7	0x00	6 in 6	0 in 200	6 in 6	6 in 6	54	87
		0x10	6 in 6	0 in 200	6 in 6	6 in 6		
		0x20	6 in 6	0 in 200	6 in 6	6 in 6		
		0x40	6 in 6	0 in 200	6 in 12	6 in 6		
		0x80	6 in 6	0 in 200	6 in 6	6 in 6		

Table 10 ELT test results

Module	Chip	AB	Energy				Energy threshold for error injection	
			75		150			
			spt 1	spt 2	spt 1	spt 2	spt 1	spt 2
ELT	6	0x00	6 in 35	0 in 200	6 in 6	0 in 200	65	82
		0x10	6 in 6	3 in 200	6 in 6	6 in 6		
		0x20	6 in 6	0 in 200	6 in 6	6 in 6		
		0x40	6 in 6	6 in 35	6 in 6	6 in 6		
		0x80	6 in 6	0 in 200	6 in 6	6 in 6		
	7	0x00	6 in 24	0 in 200	6 in 6	0 in 200	48	79
		0x10	6 in 6	6 in 139	6 in 6	6 in 6		
		0x20	6 in 6	0 in 200	6 in 6	6 in 6		
		0x40	6 in 6	6 in 210	6 in 6	6 in 6		
		0x80	6 in 6	6 in 15	6 in 6	6 in 6		

Chapter 3 – Discussion

Metal layer and SERT SEU registers were the only techniques that were successful in protecting the whole circuit and the registers, respectively, against direct laser attacks limited to the target LTC. Parity error detection could detect only errors injected in the registers limited to one bit flips. All other countermeasures failed to protect the design.

The ELT devices' failure to protect the circuit was unexpected as, based on their effectiveness against radiation, they were expected to be robust against laser attacks too. However, these transistors failed, both at combinational level, and at register level. No explanation was found for this failure, although several factors could have contributed to this result. One factor could be that the laser affects the silicon in a different way to radiation, yet with similar outcomes. In this case, it would explain how even with the absence of edges, these transistors failed when the laser targeted them. It could also be due to design and or process faults, which resulted in a weaker device. Equally, the absence of guard-rings around each transistor might be the source of leakage.

The D type flip-flop based on this ELT transistor was about 5 times bigger than a D-type FF based on standard ones. This area penalty is far too high, even for the benefits it could bring.

The combinational SEU mitigation block's failure was expected, as it only protects the combinational block if radiation, or the laser in our case, targets the combinational block for less than t_{delay}. The t_{delay} parameter is design dependant and it can be incremented or reduced as desired. Long t_{delay} might be desired to protect combinational blocks against lasers, as these can be present longer than a particle hit. However, the longer the delay, the higher will be its impact on the circuit's performance, as any change at this countermeasure's input, even a genuine one, would require t_{delay} to be propagated to its output. The laser setup available at Atmel allows two operation modes; fixed-width pulses of 5ns or a continuous laser beam. A continuous laser beam could be used to bypass this countermeasure and inject faults into the circuit by targeting the combinational block. For this particular design, however, fixed-width pulses were enough to inject these faults. Another weakness of this countermeasure is that if the output four transistors are hit by the laser, the countermeasure will instantly fail to avoid a fault injection into the next part of the circuit. Hence the use of this countermeasure does not enhance the design's robustness against laser attack and instead can result in a performance penalty.

Layout techniques in combination with parity error detection did not provide any significant robustness improvement either, as a laser spot big enough to target all the registers would inject faults to all layout approaches. All of the detected attacks relied solely on the parity error detector.

Regarding the effectiveness of the parity error detection technique, this approach could only protect the registers' content in a limited way, as any fault injected at the combinational logic would go unnoticed. Unitary bit flips could easily be detected and it could be used to detect register level attacks. But when it comes to targeting all the registers, this approach failed to detect some of these attacks. Initially it was expected that when targeting all registers, they would always be set to the same value, let's say logic 0. This would mean that the result of the addition, as well as the parity, would have been forced to logic 0. By encoding the addition's output with odd parity, such laser attack would have been detected by the global parity as an invalid number. However, the registers' were forced into different states or values each time the laser was applied to them. Only the error injections on ELT based registers were the more consistent; especially with high energy laser attacks. The injected error value would almost always be 0x1F when all registers were targeted at once, and 0x10 when only ELT registers were targeted.

This lack of fault injection consistency made it difficult to detect some of the attacks. However, by understanding how the output of an XOR cell is affected when it is targeted by a laser, a global parity scheme that, under normal circumstances provides the opposite value, could be set. This way the global parity signal would indicate that either the registers or the XOR itself have been targeted by a laser. This result could not be tested as, due to the lack of I/O pins in the test-chip, the global parity signal was not taken out of the chip, but calculated by the AVR board.

The metal layer over the design avoided the injection of any fault into the circuit when attacked from the top-side when it was the sole target. This is exclusively due to the fact that the laser cannot penetrate metal layers without causing permanent damage. ICs with a high number of layers and routing could be enough to avoid fault injections with lasers to some of the silicon parts. But using a metal plate also has its drawbacks. The metal plate cannot be left floating, as it could be charged with static electricity and eventually damage the device. It needs to be set to a certain voltage. A metal plate also adds to the parasitic capacitances with neighbouring tracks, potentially impacting the tracks' performance, i.e. device's performance, and increasing the dynamic power consumption.

Another important drawback of covering the circuit with a metal layer is that, either an additional metal layer is required for this plane or that a routing layer is partially used for protecting the device underneath. Taking into account the cost implications of adding an additional layer, this approach might not be an option. Furthermore, the back-side laser attack technique would defeat this countermeasure, making it useless.

SERT SEU registers were the only other technique to successfully protect the registers against laser attacks when they were the only target. Hardening all registers within a Smart Card with SERT SEU registers is not practical, however, it could be considered as a valid option for certain critical registers, such as I/O registers, e.g. DES related registers. To do so, a license should be applied for, as the SERT SEU register used in this test-chip is patented. Optionally, an alternative custom SERT SEU register design could be developed. Developing a custom SERT SEU register could also be beneficial for hardening SRAM memories.

Finally, an interesting behaviour was noticed when targeting all LTCs' registers. In such case, all LTC designs would fail, including SERT SEU registers and registers underneath the metal plate. Later tests achieved error injection on SERT SEU registers even without targeting them but targeting a large enough number of transistors, including ELT ones.

Laser, as well as radiation, makes transistors leaky. When the laser hits the PMOS and the NMOS transistors on an inverter, it might result in a small, temporal, shortcut between Vdd and Ground. As a result, when targeting a large enough area (or high number of transistors) a localised or global shortcut could be generated. This is what might have happened when targeting all LTCs' registers.

The mere size of ELT transistors seems to have played a key role. This is perhaps due to the resulting higher contact area than normal transistors when they become leaky, hence, being leakier than normal transistors. In fact, despite the higher number of transistors, when targeting all the combinational logic made of normal transistors, it did not inject any error on the metal layer LTC.

Two conclusions can be drawn from these results. A circuit robust against direct laser attacks could be affected by its neighbouring logic. Hence, care should also be taken with the neighbouring logic's robustness. The second conclusion is that ELT transistors could be located around a laser sensor, such as a flip-flop to increase its sensitivity to laser attack.

Chapter 4 – Conclusion

All countermeasures except for the metal plate and the ELT transistors were expected to fail in protecting the circuits against laser attacks. However, ELT transistors failed to protect the design whilst SERT SEU registers performed much better on this task. The metal plate would not protect the designs against new attack techniques, such as back-side attacks. SERT SEU registers, on the other hand, should still be useful for back-side attacks. Using these or similar techniques should be considered for critical parts of the Smart Card.

Despite ELT transistors failing in their task, they can still be useful as light detectors. From the remaining countermeasures, only the parity detection approach could still be used if the effects of a laser attack on an XOR gate can be determined.

Chapter 5 – Future work

This research line was stopped with the tests mentioned here. An additional test-chip was developed instantiating several standard and custom cells used on Atmel devices. This device was not tested due to a change of interest by Atmel. Details about the instantiated cells can be found in the report "Lamia Test-chip documentation". Since the Lamia test-chip report does not contribute to the laser attack/countermeasure knowledge, this report is not included in the submitted portfolio.

There is no plan to further develop this topic. However, in the case of doing so, using SERT SEU registers or alternative designs for sensitive registers should be considered. ELT transistors could also be used as part of the light detector scheme embedded in the Smart Card device.

Appendix A Extended Test Results

Test-chip 6:

Module	AB	Energy								Energy threshold for error injection	
		75				150					
		spt 1		spt 2		spt 1		spt 2			
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2
Metal	0x00	0x14	1	-	N/A	0x1B	1	-	N/A	45	500+
		0x14	2	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
	0x10	0x05	1	-	N/A	0x1F	1	-	N/A		
		0x05	2	-	N/A	0x15	1	-	N/A		
		0x05	6	-	N/A	0x1F	1	-	N/A		
		0x05	4	-	N/A	0x1F	1	-	N/A		
		0x05	5	-	N/A	0x1F	2	-	N/A		
		0x05	3	-	N/A	0x1F	1	-	N/A		
	0x20	0x06	4	-	N/A	0x1F	1	-	N/A		
		0x06	1	-	N/A	0x0F	1	-	N/A		
		0x06	1	-	N/A	0x1F	1	-	N/A		
		0x06	1	-	N/A	0x1F	1	-	N/A		
		0x06	6	-	N/A	0x1F	1	-	N/A		

		0x06	4	-	N/A	0x1F	1	-	N/A		
	0x40	0x1D	1	-	N/A	0x0F	1	-	N/A		
		0x1D	1	-	N/A	0x1F	1	-	N/A		
		0x1D	1	-	N/A	0x0F	1	-	N/A		
		0x1F	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x1F	1	-	N/A	0x16	1	-	N/A		
	0x80	0x1B	1	-	N/A	0x1F	1	-	N/A		
		0x1B	1	-	N/A	0x1F	1	-	N/A		
		0x0E	1	-	N/A	0x1F	1	-	N/A		
		0x1C	1	-	N/A	0x1F	1	-	N/A		
		0x1C	2	-	N/A	0x1F	1	-	N/A		
		0x1C	2	-	N/A	0x1F	1	-	N/A		

Module	AB	Energy								Energy threshold for error injection	
		75				150					
		spt 1		spt 2		spt 1		spt 2			
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2
RH	0x00	0x11	2	-	N/A	0x1F	1	-	N/A	43	500+
		0x11	1	-	N/A	0x1F	1	-	N/A		
		0x11	1	-	N/A	0x1F	1	-	N/A		
		0x11	1	-	N/A	0x10	1	-	N/A		
		0x11	1	-	N/A	0x1F	1	-	N/A		
		0x11	2	-	N/A	0x1F	1	-	N/A		
	0x10	0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		

	0x20	0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x11	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
	0x40	0x10	1	-	N/A	0x1F	1	-	N/A		
		0x15	1	-	N/A	0x1F	1	-	N/A		
		0x15	1	-	N/A	0x1F	1	-	N/A		
		0x15	1	-	N/A	0x1F	1	-	N/A		
		0x11	1	-	N/A	0x1F	1	-	N/A		
		0x11	1	-	N/A	0x1F	1	-	N/A		
	0x80	0x10	1	-	N/A	0x1F	1	-	N/A		
		0x01	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		

Module	AB	Energy								Energy threshold for error injection		
		75				150						
		spt 1		spt 2		spt 1		spt 2				
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2	
std_a	0x00	-	N/A	-	N/A	0x1F	1	0x17	1	54	100	0x00/E=75/spt1 -> 100 shots in total
		-	N/A	-	N/A	0x1F	1	0x17	1			
		-	N/A	-	N/A	0x1F	1	0x16	1			
		-	N/A	-	N/A	0x1B	1	0x17	1			
		-	N/A	-	N/A	0x1F	1	0x16	1			

		-	N/A	-	N/A	0x1F	1	0x16	1		
	0x10	0x07	9	-	N/A	0x1F	1	0x07	1		
		0x13	1	-	N/A	0x1F	1	0x07	1		
		0x17	3	-	N/A	0x1F	1	0x07	1		
		0x07	1	-	N/A	0x1F	1	0x07	1		
		0x07	3	-	N/A	0x1F	1	0x07	1		
		0x07	2	-	N/A	0x1F	1	0x07	1		
	0x20	0x07	1	-	N/A	0x0F	1	0x06	1		
		0x07	1	-	N/A	0x0F	1	0x06	1		
		0x06	3	-	N/A	0x1F	1	0x06	1		
		0x07	1	-	N/A	0x1F	1	0x06	1		
		0x12	1	-	N/A	0x1F	1	0x06	1		
		0x16	3	-	N/A	0x1F	1	0x06	1		
	0x40	0x10	1	-	N/A	0x0F	1	0x07	1		
		0x10	3	-	N/A	0x0F	1	0x07	1		
		0x05	3	-	N/A	0x1F	1	0x06	1		
		0x10	1	-	N/A	0x1F	1	0x06	1		
		0x07	5	-	N/A	0x1F	1	0x07	1		
		0x10	1	-	N/A	0x1F	1	0x06	1		
	0x80	0x0E	4	-	N/A	0x1F	1	0x06	1		
		0x1F	6	-	N/A	0x0F	1	0x06	1		
		0x09	2	-	N/A	0x1F	1	0x06	1		
		0x0B	5	-	N/A	0x1F	1	0x0E	1		
		0x0E	3	-	N/A	0x1F	1	0x06	1		
		0x0B	4	-	N/A	0x1F	1	0x0E	1		

Module	AB	Energy								Energy threshold for error injection	
		75				150					
		spt 1		spt 2		spt 1		spt 2			
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2
std_b	0x00	0x1B	1	0x19	24	0x17	1	0x07	1	49	94
		0x1B	1	0x14	85	0x17	1	0x07	1		
		0x1F	1	0x14	16	0x13	1	0x07	1		
		0x1F	1	0x18	2	0x17	1	0x07	1		
		0x1F	1	0x18	3	0x17	1	0x17	1		
		0x1F	1	0x19	9	0x13	1	0x07	1		
	0x10	0x1B	1	-	N/A	0x17	1	0x07	1		
		0x15	1	-	N/A	0x17	1	0x07	1		
		0x1B	1	-	N/A	0x1F	1	0x17	1		
		0x13	1	-	N/A	0x17	1	0x07	1		
		0x05	1	-	N/A	0x13	1	0x07	1		
		0x05	1	-	N/A	0x17	1	0x07	1		
	0x20	0x05	1	-	N/A	0x12	1	0x17	1		
		0x05	1	-	N/A	0x12	1	0x17	1		
		0x03	1	-	N/A	0x1B	1	0x07	1		
		0x05	1	-	N/A	0x1B	1	0x07	1		
		0x03	1	-	N/A	0x1B	1	0x17	1		
		0x03	1	-	N/A	0x1B	1	0x17	1		
	0x40	0x05	1	0x05	1	0x16	1	0x17	1		
		0x07	1	-	N/A	0x16	1	0x17	1		
		0x07	1	-	N/A	0x1E	1	0x07	1		
		0x07	1	-	N/A	0x17	1	0x07	1		
		0x05	1	-	N/A	0x1F	1	0x07	1		
		0x07	1	-	N/A	0x1F	1	0x07	1		
	0x80	0x09	1	0x05	7	0x1B	1	0x0F	1		

0x40/E=75/spt2 -> 201 shots in total

		0x09	1	0x09	10	0x1B	1	0x0F	1		
		0x09	1	0x15	113	0x1A	1	0x0F	1		
		0x09	1	0x0D	5	0x1B	2	0x0F	1		
		0x09	1	0x01	11	0x1B	1	0x0F	1		
		0x09	1	0x0C	43	0x1F	1	0x0F	1		

Module	AB	Energy								Energy threshold for error injection	
		75				150					
		spt 1		spt 2		spt 1		spt 2			
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2
std_c	0x00	0x02	1	0x05	1	0x0C	1	0x1E	1	58	94
		0x1F	1	0x05	1	0x1E	1	0x1E	1		
		0x06	1	0x05	2	0x1F	1	0x1E	1		
		0x02	1	0x05	6	0x0D	2	0x1E	1		
		0x04	2	-	N/A	0x16	1	0x1E	1		
		0x16	1	-	N/A	0x16	4	0x1E	1		
	0x10	0x07	1	-	N/A	0x1F	1	0x1E	1		
		0x07	1	-	N/A	0x0F	1	0x1F	1		
		0x1B	2	-	N/A	0x1F	1	0x1E	1		
		0x15	1	-	N/A	0x0F	1	0x0E	1		
		0x07	1	-	N/A	0x15	1	0x1E	1		
		0x10	1	-	N/A	0x15	1	0x1E	1		
	0x20	0x16	1	0x03	7	0x16	1	0x1E	1		
		0x16	1	-	N/A	0x1D	1	0x1E	1		
		0x1F	1	-	N/A	0x1F	1	0x0E	1		
		0x1F	1	-	N/A	0x16	1	0x0E	1		
		0x1F	1	-	N/A	0x16	1	0x1E	1		
		0x1F	1	-	N/A	0x16	1	0x1E	1		
	0x40	0x16	1	0x11	18	0x1F	1	0x1E	1		
		0x1F	1	-	N/A	0x1F	1	0x1E	1		

		0x1F	1	-	N/A	0x16	2	0x1E	1		
		0x1F	1	-	N/A	0x1F	2	0x1E	1		
		0x16	1	-	N/A	0x1F	2	0x1E	1		
		0x1F	1	-	N/A	0x16	4	0x1E	1		
	0x80	0x0E	1	0x09	2	0x1C	1	0x1E	1		
		0x1F	1	0x09	2	0x0B	1	0x1E	1		
		0x1F	1	0x09	3	0x0B	1	0x1E	1		
		0x1D	1	0x09	2	0x1C	1	0x1E	1		
		0x1D	1	0x09	1	0x0E	2	0x0E	1		
		0x1F	1	0x09	1	0x1C	1	0x1E	1		

Module	AB	Energy								Energy threshold for error injection	
		75				150					
		spt 1		spt 2		spt 1		spt 2			
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2
ELT	0x00	0x18	8	-	N/A	0x1F	1	-	N/A	65	82
		0x18	11	-	N/A	0x1F	1	-	N/A		
		0x18	6	-	N/A	0x1F	1	-	N/A		
		0x18	2	-	N/A	0x1F	1	-	N/A		
		0x18	6	-	N/A	0x1F	1	-	N/A		
		0x18	2	-	N/A	0x1F	1	-	N/A		
	0x10	0x19	1	0x11	5	0x1F	1	0x10	1		
		0x19	1	0x10	23	0x1F	1	0x10	1		
		0x19	1	0x11	17	0x1F	1	0x10	1		
		0x10	1	-	N/A	0x1F	1	0x10	1		
		0x18	1	-	N/A	0x1F	1	0x10	1		
		0x10	1	-	N/A	0x1F	1	0x10	1		
	0x20	0x10	1	-	N/A	0x1F	1	0x10	1		

0x10/E=75/spt2 -> 245 shots in total

		0x10	1	-	N/A	0x1F	1	0x10	1		
		0x10	1	-	N/A	0x1F	1	0x10	1		
		0x10	1	-	N/A	0x1F	1	0x10	1		
		0x10	1	-	N/A	0x1F	1	0x10	1		
		0x10	1	-	N/A	0x1F	1	0x10	1		
	0x40	0x10	1	0x14	1	0x1F	1	0x10	1		
		0x10	1	0x14	2	0x1F	1	0x10	1		
		0x10	1	0x14	1	0x1F	1	0x10	1		
		0x10	1	0x14	2	0x1F	1	0x10	1		
		0x1C	1	0x14	22	0x1F	1	0x10	1		
		0x10	1	0x14	7	0x1F	1	0x10	1		
	0x80	0x10	1	-	N/A	0x1F	1	0x10	1		
		0x10	1	-	N/A	0x1F	1	0x10	1		
		0x10	1	-	N/A	0x1F	1	0x10	1		
		0x10	1	-	N/A	0x1F	1	0x10	1		
		0x10	1	-	N/A	0x1F	1	0x10	1		
		0x10	1	-	N/A	0x1F	1	0x10	1		

Test-chip 7:

Module	AB	Energy								Energy threshold for error injection	
		75				150					
		spt 1		spt 2		spt 1		spt 2			
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2
Metal	0x00	0x04	1	-	N/A	0x1C	1	-	N/A	48	500+
		0x04	1	-	N/A	0x1F	1	-	N/A		
		0x1F	1	-	N/A	0x1D	1	-	N/A		
		0x04	1	-	N/A	0x0E	1	-	N/A		
		0x1F	1	-	N/A	0x1C	1	-	N/A		
		0x1F	1	-	N/A	0x1F	1	-	N/A		
	0x10	0x1F	1	-	N/A	0x1F	1	-	N/A		
		0x0F	1	-	N/A	0x1F	1	-	N/A		
		0x1F	1	-	N/A	0x0F	1	-	N/A		
		0x1F	1	-	N/A	0x15	1	-	N/A		
		0x1F	1	-	N/A	0x1D	1	-	N/A		
		0x0D	1	-	N/A	0x1D	1	-	N/A		
	0x20	0x1D	1	-	N/A	0x16	1	-	N/A		
		0x1D	1	-	N/A	0x16	1	-	N/A		
		0x1F	1	-	N/A	0x16	1	-	N/A		
		0x1F	1	-	N/A	0x1F	1	-	N/A		
		0x1F	1	-	N/A	0x1F	1	-	N/A		
		0x1F	1	-	N/A	0x16	1	-	N/A		
	0x40	0x1F	1	-	N/A	0x1F	1	-	N/A		
		0x1F	1	-	N/A	0x1F	1	-	N/A		
		0x1D	1	-	N/A	0x0F	1	-	N/A		
		0x1F	1	-	N/A	0x1F	1	-	N/A		
		0x1F	1	-	N/A	0x1F	1	-	N/A		

		0x1F	1	-	N/A	0x1F	1	-	N/A		
	0x80	0x0F	1	-	N/A	0x1F	1	-	N/A		
		0x1F	1	-	N/A	0x1F	1	-	N/A		
		0x1F	1	-	N/A	0x0E	1	-	N/A		
		0x1F	1	-	N/A	0x0F	1	-	N/A		
		0x1F	1	-	N/A	0x1F	1	-	N/A		
		0x1F	1	-	N/A	0x1F	1	-	N/A		

Module	AB	Energy								Energy threshold for error injection	
		75				150					
		spt 1		spt 2		spt 1		spt 2			
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2
RH	0x00	0x04	1	-	N/A	0x1F	1	-	N/A	51	500+
		0x04	1	-	N/A	0x1F	1	-	N/A		
		0x04	1	-	N/A	0x1F	1	-	N/A		
		0x04	1	-	N/A	0x1F	1	-	N/A		
		0x04	1	-	N/A	0x1F	1	-	N/A		
		0x04	1	-	N/A	0x1F	1	-	N/A		
	0x10	0x04	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
	0x20	0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		

	0x40	0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
	0x80	0x10	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x14	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1F	1	-	N/A		
		0x10	1	-	N/A	0x1D	1	-	N/A		

Module	AB	Energy								Energy threshold for error injection	
		75				150					
		spt 1		spt 2		spt 1		spt 2			
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2
std_a	0x00	0x1E	1	-	N/A	0x1F	1	0x1F	1	51	105
		0x1E	1	-	N/A	0x1F	1	0x1F	1		
		0x1F	1	-	N/A	0x0C	1	0x1F	1		
		0x1F	1	-	N/A	0x1D	1	0x1B	1		
		0x0F	1	-	N/A	0x1D	1	0x1F	1		
		0x1C	1	-	N/A	0x0C	1	0x1A	1		
	0x10	0x0F	1	-	N/A	0x1F	1	0x0F	1		
		0x0F	1	-	N/A	0x1F	1	0x0F	1		
		0x0F	1	-	N/A	0x1F	1	0x0F	1		
		0x0F	1	-	N/A	0x0F	1	0x0C	1		
		0x0F	1	-	N/A	0x1D	1	0x0F	1		
		0x1F	1	-	N/A	0x1D	1	0x1F	1		
	0x20	0x0F	1	-	N/A	0x0F	1	0x03	1		

		0x0F	1	-	N/A	0x0F	1	0x0B	1		
		0x0F	1	-	N/A	0x1F	1	0x0B	1		
		0x1F	1	-	N/A	0x1E	1	0x0F	1		
		0x1F	1	-	N/A	0x0F	1	0x0F	1		
		0x0E	1	-	N/A	0x0F	1	0x0B	1		
	0x40	0x1E	1	-	N/A	0x1C	1	0x0F	1		
		0x0F	1	-	N/A	0x1C	1	0x0B	1		
		0x0F	1	-	N/A	0x0D	1	0x0B	1		
		0x1F	1	-	N/A	0x1E	1	0x0B	1		
		0x1F	1	-	N/A	0x0E	1	0x0B	1		
		0x1F	1	-	N/A	0x0F	1	0x0B	1		
	0x80	0x1F	1	-	N/A	0x0C	1	0x0B	1		
		0x1F	1	-	N/A	0x0C	1	0x0B	1		
		0x1F	1	-	N/A	0x1F	1	0x0B	1		
		0x1F	1	-	N/A	0x1E	1	0x0B	1		
		0x0E	1	-	N/A	0x1E	1	0x0F	1		
		0x0F	1	-	N/A	0x0E	1	0x0F	1		

Module	AB	Energy								Energy threshold for error injection	
		75				150					
		spt 1		spt 2		spt 1		spt 2			
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2
std_b	0x00	0x1F	1	-	N/A	0x0A	1	0x1B	1	54	94
		0x1F	1	-	N/A	0x02	1	0x1B	1		
		0x1F	1	-	N/A	0x02	1	0x1B	1		
		0x1F	1	-	N/A	0x1E	1	0x1B	1		
		0x1E	1	-	N/A	0x1A	1	0x1B	1		
		0x1B	1	-	N/A	0x0E	1	0x1B	1		
	0x10	0x1F	1	-	N/A	0x1F	1	0x0B	1		
		0x1F	1	-	N/A	0x1F	1	0x0B	1		

		0x1F	1	-	N/A	0x1F	1	0x0B	1		
		0x1F	1	-	N/A	0x0F	1	0x0B	1		
		0x1F	1	-	N/A	0x17	1	0x0B	1		
		0x1F	1	-	N/A	0x0F	1	0x0B	1		
	0x20	0x1F	1	-	N/A	0x17	1	0x0F	1		
		0x1F	1	-	N/A	0x0E	1	0x0B	1		
		0x1F	1	-	N/A	0x0E	1	0x0B	1		
		0x1F	1	-	N/A	0x0A	1	0x0B	1		
		0x1F	1	-	N/A	0x0A	1	0x0B	1		
		0x1F	1	-	N/A	0x0A	1	0x0B	1		
	0x40	0x1F	1	-	N/A	0x06	1	0x0F	1		
		0x1F	1	-	N/A	0x06	1	0x0F	1		
		0x1F	1	-	N/A	0x16	1	0x0F	1		
		0x1D	1	-	N/A	0x0E	1	0x0F	1		
		0x1D	1	-	N/A	0x0E	1	0x0F	1		
		0x1E	1	-	N/A	0x0E	1	0x0F	1		
	0x80	0x1F	1	-	N/A	0x0A	1	0x0B	1		
		0x1F	1	-	N/A	0x0A	1	0x0B	1		
		0x1F	1	-	N/A	0x0E	1	0x0B	1		
		0x1F	1	-	N/A	0x0E	1	0x0B	1		
		0x1E	1	-	N/A	0x06	1	0x0B	1		
		0x1F	1	-	N/A	0x0E	1	0x0B	1		

Module	AB	Energy								Energy threshold for error injection	
		75				150					
		spt 1		spt 2		spt 1		spt 2			
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2
std_c	0x00	0x1F	1	-	N/A	0x41	1	0x0B	1	54	94
		0x1F	1	-	N/A	0x41	1	0x0B	1		
		0x1F	1	-	N/A	0x1F	1	0x0B	1		

		0x1F	1	-	N/A	0x41	1	0x0B	1		
		0x1F	1	-	N/A	0x1F	1	0x0B	1		
		0x1F	1	-	N/A	0x0B	1	0x0B	1		
	0x10	0x1F	1	-	N/A	0x07	1	0x11	1		
		0x1F	1	-	N/A	0x0F	1	0x1B	1		
		0x1D	1	-	N/A	0x0F	1	0x1B	1		
		0x1F	1	-	N/A	0x0F	1	0x1B	1		
		0x1D	1	-	N/A	0x15	1	0x1B	1		
		0x1D	1	-	N/A	0x15	1	0x1B	1		
	0x20	0x1F	1	-	N/A	0x16	1	0x0B	1		
		0x1F	1	-	N/A	0x1F	1	0x0B	1		
		0x1F	1	-	N/A	0x1F	1	0x0B	1		
		0x1D	1	-	N/A	0x0E	1	0x1B	1		
		0x1D	1	-	N/A	0x0F	1	0x0B	1		
		0x1E	1	-	N/A	0x0F	1	0x0B	1		
	0x40	0x1D	1	-	N/A	0x0F	2	0x1F	1		
		0x1D	1	-	N/A	0x0C	4	0x0D	1		
		0x1D	1	-	N/A	0x1F	1	0x0F	1		
		0x1F	1	-	N/A	0x0F	3	0x0F	1		
		0x1D	1	-	N/A	0x1C	1	0x0F	1		
		0x1F	1	-	N/A	0x0F	1	0x0F	1		
	0x80	0x1F	1	-	N/A	0x1C	1	0x1F	1		
		0x1D	1	-	N/A	0x1C	1	0x1F	1		
		0x1D	1	-	N/A	0x0F	1	0x1B	1		
		0x0E	1	-	N/A	0x0F	1	0x0B	1		
		0x1D	1	-	N/A	0x0F	1	0x09	1		
		0x0F	1	-	N/A	0x0F	1	0x1B	1		

Module	AB	Energy								Energy threshold for error injection	
		75				150					
		spt 1		spt 2		spt 1		spt 2			
		injected value	shots to error	injected value	shots to error	injected value	shots to error	injected value	shots to error	spt 1	spt 2
ELT	0x00	0x18	4	-	N/A	0x1F	1	-	N/A	65	82
		0x18	4	-	N/A	0x1F	1	-	N/A		
		0x18	12	-	N/A	0x1F	1	-	N/A		
		0x18	11	-	N/A	0x1F	1	-	N/A		
		0x18	1	-	N/A	0x1F	1	-	N/A		
		0x18	1	-	N/A	0x1F	1	-	N/A		
	0x10	0x18	1	0x11	120	0x1F	1	0x10	1		
		0x18	1	0x11	3	0x1F	1	0x10	1		
		0x1C	1	0x11	12	0x1F	1	0x10	1		
		0x18	1	0x11	2	0x1F	1	0x10	1		
		0x18	1	0x11	1	0x1F	1	0x10	1		
		0x18	1	0x11	1	0x1F	1	0x10	1		
	0x20	0x1C	1	-	N/A	0x1F	1	0x10	1		
		0x18	1	-	N/A	0x1F	1	0x10	1		
		0x1C	1	-	N/A	0x1F	1	0x10	1		
		0x1C	1	-	N/A	0x1F	1	0x10	1		
		0x18	1	-	N/A	0x1F	1	0x10	1		
		0x1C	1	-	N/A	0x1F	1	0x10	1		
	0x40	0x1C	1	0x14	163	0x1F	1	0x10	1		
		0x14	1	0x14	24	0x1F	1	0x10	1		
		0x14	1	0x14	7	0x1F	1	0x10	1		
		0x1C	1	0x14	1	0x19	1	0x10	1		
		0x14	1	0x14	11	0x1D	1	0x10	1		
		0x14	1	0x14	4	0x1F	1	0x10	1		
	0x80	0x10	1	0x10	1	0x1D	1	0x10	1		
		0x10	1	0x10	3	0x1F	1	0x10	1		

		0x10	1	0x10	1	0x1F	1	0x10	1		
		0x10	1	0x10	3	0x1D	1	0x10	1		
		0x10	1	0x10	6	0x1F	1	0x10	1		
		0x14	1	0x10	1	0x19	1	0x10	1		

References

1. Donald C. Mayer, R.C.L. *Designing Integrated Circuits to Withstand Sapce Radiation*. 2003 [cited; Available from: www.aero.org/publications/crosslink/summer2003/06.html.
2. Calin, T., M. Nicolaidis, and R. Velazco, *Upset Hardened Memory Design for Submicron CMOS Technology.* IEEE Transactions on Nuclear Science, 1996. **43**(6): p. 2874-8.
3. J. Gambles, K.H.a.S.W., *Radiation Hardness of Ultra Low Power CMOS VLSI.*
4. P. Mongkolkachit, B.B., *Design Technique for Mitigation of Alpha-Particle-Induced Single-Event Transients in Combinational Logic.* IEEE Transactions on Device and Materials Reliability, 2003. **3**(3): p. 89-92.

Engineering Doctorate

The glitch detector design parameters and the SPICE simulation environment and its detection characterisation in silicon with the 01VGA test-chip (codename Arrano)

Author: Asier Goikoetxea Yanci

Date: November 2010

Sponsored by:

Acknowledgements

This work was possible only thanks to the input, help and/or support of several Atmel groups and employees. Thanks to David Dougan and Louis Frew, both from the Analog Group for helping to fine tune the operational amplifier used in the new glitch detector and providing me with the voltage regulators used on Smart Cards to test the glitch detector.

I want to thank the Layout Group for the work they did with the test-chip 01VGA (Arrano), as the power source requirements of this device where substantially different to those they encounter on a normal device. Thanks also to the System Development Group for letting me use one of their Orcad licenses to develop the test-board, their support with the Voyager System (used as a motherboard to test several test-chips) and for their help on sourcing different components needed on the test-board.

I also would like to thank Jalib Ahmed for providing me with the Cratis RTL and program code to instantiate it in the Voyager System, as well as for his help with setting the FPGA constraints. The use of Cratis shortened the test setup development time and provided me with a standard communication channel to setup and launch tests on the test-board.

Last but not least, I also thank John Connor and Nigel Davenport, both from the security group. Nigel for giving me to access the Security Evaluation Lab when I needed it and for providing me with the glitch attack setup (only after a health and safety chat). John, for providing me with the source code of the front-end glitch attack control application, which fast-tracked the test setup development and provided me with an automated means to carry out glitch attacks.

Table of Contents

1 Introduction

Smart Cards can be subjected to a series of threats and attacks to exploit design weaknesses and to gain access or knowledge of otherwise secured information or data. Applying glitches on the Smart Card's power rails (Vcc and/or GND) is a commonly used attack technique which can inject faults into the Smart Card making it misbehave.

Smart Cards include a built-in voltage regulator which can filter out some noise and glitches present on the Smart Card's power rails, but other noise and glitches might still pass through it and can affect the circuit's behaviour. Glitch detectors are used to monitor for abnormal variations and voltage values on the power supply at the voltage regulator's input (Vcc) and output (Vdd). As with any other detector, glitch detectors have an operational detection range, and any glitch outside that range might not be detected. Some fast glitches (short width) are out of the detection range of commonly used detectors and are capable of injecting faults into the Smart Card without being detected.

A line of work of this research resulted in a new glitch detector capable of detecting certain fast glitches current detectors miss. This detector, whose patent is available in GlitchPub3, has a slower response time than current ones, so it was concluded that it could only be use in conjunction with current detectors to improve the overall detection range. Papers GlitchPub1 and GlitchPub2 cover the detector and its simulation and silicon performance. Volume I of this portfolio covers this same information in more detail.

This report covers the simulation setup, test-chip design and the test environment used in the process of this research line. Chapter 2 provides a quick review of the glitch detector design. Chapter 3 covers the simulation environment, parameters and the different designs used in the simulation process. Chapter 4 concludes this report with a description of the test-chip and test environment, including test equipment and a custom built board.

2 The new glitch detector design

The proposed glitch detector, shown in Figure 1, is a glitch sensitive mono-stable circuit, prone to fault injection through glitches. The inverter and operational amplifier make up a mono-stable circuit which, under normal operation circumstances, produces a logic low output.

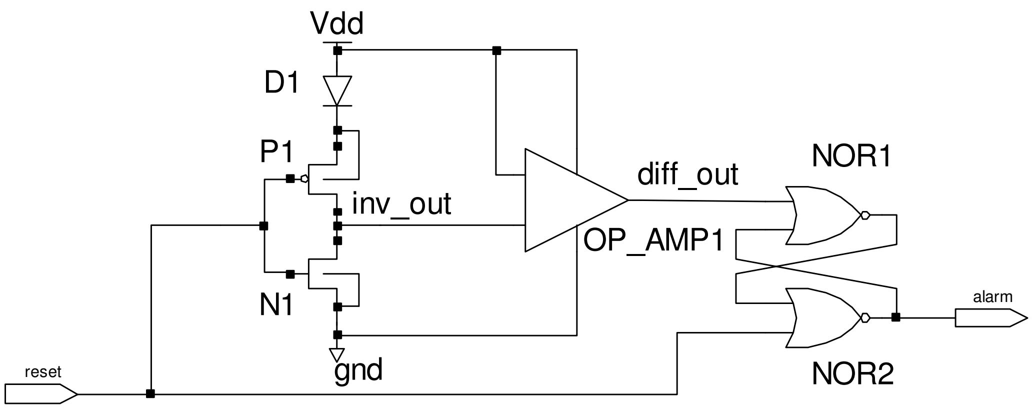


Figure 1 Preferred embodiment

The presence of a positive glitch on Vdd can force the operational amplifier's output to logic high, thus indicating the detection of the glitch event in the power supply. As covered in the paper GlitchPub1 and the section 3.2.2 Design in the Volume I of this portfolio, the effect of a glitch on the operational amplifier's output is temporal, and hence, its output will eventually revert to a logic low. Since the glitch attack and its detection are asynchronous events, the RS latch registers the glitch detection so that it can be processed by the CPU when it is ready.

The reset input can be used to reset the RS latch and to force the mono-stable circuit's normal state. Correct dimensioning of devices D1, P1 and N1 is the key for correct operation of this glitch detector. Chapters 3 and 4 of this report indicate the dimensions used for the simulations and silicon tests respectively.

3 The glitch detector simulation setup

Two different versions of the new glitch detector were simulated to check their behaviour under glitch attacks and to test their validity as glitch detectors. This chapter explains the simulation environment used in this process and the design parameters of the tested glitch detectors.

The proposed glitch detector was designed to detect glitches on the Smart Card's internal power supply, i.e. Vdd. In other words, it was designed to detect abnormal voltage fluctuations at the voltage regulator's output. A voltage regulator's response to a glitch depends on: the glitch itself, the voltage regulator design and the load the voltage regulator is subjected to when the glitch is applied. Hence, the glitch detector validation simulation environment had to include simulation of the voltage regulator that powered the glitch detector as well as modelling the load to which a Smart Card voltage regulator may be subjected. Figure 2 shows a diagram of the simulated environment.

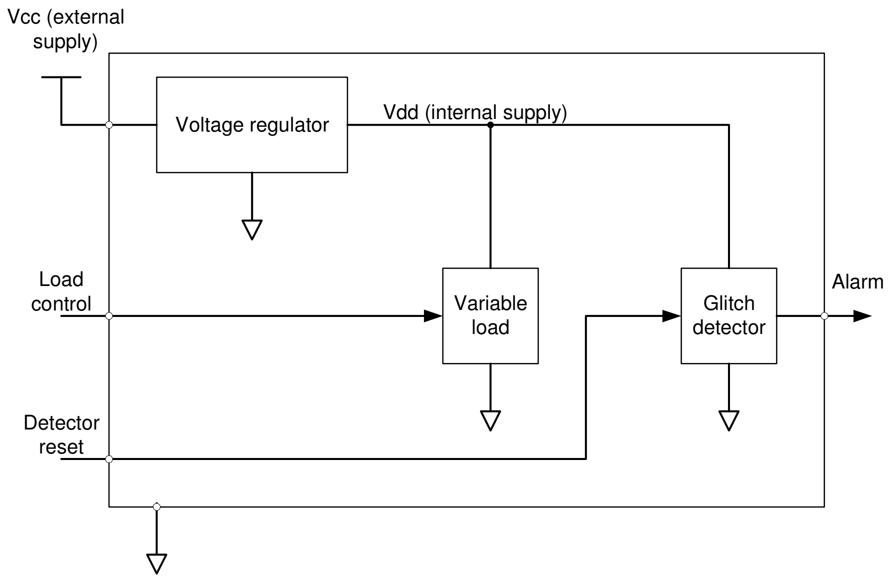


Figure 2 Glitch detector test diagram

The voltage regulator chosen to test the proposed glitch detector is a voltage regulator used with certain Smart Cards for GSM applications, as these tend to have a lower security requirement and, hence, tend to be less robust than voltage regulators used in Smart Cards targeting other applications. Built-in voltage regulators power both capacitive and resistive loads. For simplicity reasons, fixed capacitive and resistive loads were simulated. The capacitive load was constant on all simulations and set to 2.2nF. The capacitive load represented the voltage regulator's output capacitance and the digital device's equivalent capacitive load. Three resistive loads were considered for three different load scenarios: high load (80 Ohms), medium load (1K6 Ohms) and low load (3K2 Ohms). Resistive loads were fixed to one of these values on each simulation. All load values were chosen by Louis Frew based on prior experience.

Regarding the simulation tool, in principle, GAPASE could have been used to simulate the circuitry in Figure 2. However, HSPICE was preferred for the following reason: GAPASE is a simulation environment based on Nanosim, which is more focused toward simulating digital circuits and although Nanosim's analogue circuit simulation accuracy can be increased, the performance versus accuracy penalty of HSPICE was considered affordable for this kind of simulation.

The circuitry in Figure 2 was simulated for several standard power supply scenarios and SPICE model cases, all of which are covered in section 3.2.3.1 Simulation Test and Results of Volume I of this portfolio. Power supply scenarios include clean and noisy external Vcc supplies of 3V and 5V and two glitches not detected by the detectors included in the voltage regulator used in the simulations. These glitches were: a) raising Vcc from 2.7V to 7V then back to 2.7V within 100ns; and b) raising Vcc from 3V to 15V then back to 3V within 10ns. These glitches were defined by three point waveforms as shown in the sample main SPICE simulation file below.

Due to the amount of possible different power and load scenarios to be simulated for each design (18 in total), it was decided to simulate only extreme SPICE model cases, i.e. worst-case and best-case. Worst-case model include the following parameters: `mos_wcs, rlow, clow, temp=125`. Best-case model include: `mos_bcs, rhigh, chigh, temp=-40`.

```
*******************************
.lib '../../hl49at58.85kr140.mod' process_tolerances
.lib '../../hl49at58.85kr140.mod' mos_wcs
.lib '../../hl49at58.85kr140.mod' techno
.lib '../../hl49at58.85kr140.mod' nonmc
.lib '../../hl49at58.85kr140.mod' nonmatching
.lib '../../hl49at58.85kr140.mod' model_58k85
.lib '../../hl49at58.85kr140.mod' rlow
.lib '../../hl49at58.85kr140.mod' clow

***** read in the netlist. *****

** power supply design
*** attach high load
 .include './glue_high'
*** attach medium load
*.include './glue_medium'
*** attach low load
*.include './glue_low'
*** voltage regulator
.include '../../ascpf41s/hspices/cmos.sch_simu/netlist/hspicefinal'
```

```
** glitch detector design
.include './hspicefinal'

** circuits initial status
.include './hvt_inv_hvt_opamp_high_load_deck_wcs_trans.ic'

************************
*    options
************************
*******************************************************************
* speed options ***************************************************
*******************************************************************
*.options nomod ingold=2 method=trap fast numdgt=6 post probe
*******************************************************************
*******************************************************************
* accuracy options ************************************************
*******************************************************************
.options  nomod ingold=2 method=gear numdgt=6 post probe
*******************************************************************
************************
* include op points
************************
************************
* parameter statements
************************
.param vdd=1.6v
.param vcc=3v
.param vee=0v
.param toler_bias=1.0
.param toler_c=1.0
.param delay=7us
.param trise=0.1us
.param torise=0.05us
.param tprise=100ns
.param tfall=0.1us
.param pwidth=9.9us
.param period=20us
.temp 125
.op 5us
************************
* stimulus
************************
*** when using voltage regulator uncomment these ***
vpower_noise power_noise 0 pwl (0 1 1000u 1 1000.002u 5 1000.098u 5 1000.1u 1)
ep2 mixvdd! 0 poly(2) power_noise 0 vcc_voltage 0 0 0 0 0 1
vvcc_voltage vcc_voltage 0 vcc

*** signals
vpower_on_reset por 0 pwl(0 0 1000n vdd 1500n vdd 1500.1n 0)
************************
* tran analysis
************************
.tran 1n 3m uic
************************
* probe nodes
************************
.probe tran v(vdd!)
.probe tran i(vvdd!)
.probe tran v(ground_net)
.probe tran i(vground_net)
.probe tran v(mixvdd!)
.probe tran i(ep")
.probe tran v(alarm)
.probe tran v(alarm_set)
.probe tran v(loop_reset)
.probe tran v(loop_out)
.probe tran v(net61)
.probe tran v(por)
.probe tran v(xi28.net18)
.probe tran v(xi28.net9)
```

```
.probe tran v(xi27.a)
.probe tran v(xi27.b)
.probe tran v(xi27.c)
.probe tran v(xi27.d)
.probe tran v(xi27.e)
.probe tran v(xi27.f)
.probe tran v(xi27.g)
.probe tran v(xi27.h)
.probe tran v(xi27.net027)
.probe tran v(xi27.su)
************************
.end
```

3.1 Simulated Glitch Detector Designs

Two designs were simulated in this environment and referred to as Design A and Design B in paper GlitchPub1 and Volume I of this Portfolio. Design A had its diode and RS latch made of low leakage transistors and the inverter and operational amplifier made of high voltage transistors. Design B was made entirely of low leakage transistors. Low leakage transistors are designed with a higher voltage threshold (V_{th}) than their normal counterparts. Increasing the V_{th} helps reducing the leakage current, an increasing issue with deep-submicron technologies; however, it also reduces the transistor performance when comparing to transistors with lower V_{th}

In both cases, the modified inverter was made of transistors with identical width and length (W/L) dimensions. D1: 400/400 nm (W/L); P1: 73/1.2 um (W/L) and; N1: 7.3/1.5 um (W/L). The operational amplifiers in both designs were also of similar dimensions, whereas the RS latches were made with Atmel standard cells of 0.18um. These designs are confidential and only available to Atmel in the following location:

```
/home/asierg/derivable/research_data/01_fault_tolerant/01_03_giltch_det
ector/02_chip_design/ultimate_detector
```

4 The glitch detector silicon characterisation setup

In order to validate the proposed detector, its behaviour in silicon needed to be characterised. A test-chip was designed for this purpose with four versions of the new glitch detector, and a few test-chip dice were characterised in a purpose built test environment. The following two sections of this chapter explain different aspects related to the glitch detector characterisation.

The first section covers the test-chip in more detail than has been covered before in Volume I of this portfolio. The second section covers the different tools and equipment used in the test environment and how are they connected.

4.1 The Arrano/01VGA test-chip design

A test-chip was designed to characterise the glitch detection range of different glitch detector versions. Four versions of the proposed glitch detector were instantiated and powered by two different voltage regulators in a similar setup to that used in the simulation environment, Figure 2. Figure 3 shows a diagram of the Arrano test-chip, where GD_0, GD_1, GD_2 and GD_3 represent the different glitch detector versions and VR1 and VR2 represent the two different voltage regulators. The test-chip also included four SRAM cells with different structures to test each structure's robustness against glitch attacks. The tests on the SRAMs were not carried out due to a lack of time; hence, these SRAMs are not covered in this report.

The instantiated glitch detector versions differ as follows:

Glitch detectors GD_1, GD_2 and GD_3 instantiate identical modified inverters with the following parameters: D1 low leakage transistor with dimensions 4/4 um (W/L), P1 high voltage transistor with dimensions 1.8/11 um (W/L) and N1 high voltage transistor with dimensions 1.8/5.5 um (W/L). They also instantiated identical low leakage RS latches. The OpAmp instantiated with the GD_1 is identical to the one used in the simulation process, although made of low leakage transistors. The OpAmps instantiated with GD_2 and GD_3 are both made of high voltage transistors, GD_3's being slightly faster than GD_2.

The glitch detector GD_0 instantiated a modified inverter with the following parameters: D1 low leakage transistor with dimensions 4/4 um (W/L), P1 low leakage transistor with dimensions

1.8/11 um (W/L) and N1 low leakage transistor with dimensions 1.8/5.5 um (W/L). This glitch detector also instantiates an RS latch identical to that of GD_1, GD_2 and GD_3. The OpAmp is, again, identical to that of GD_1.

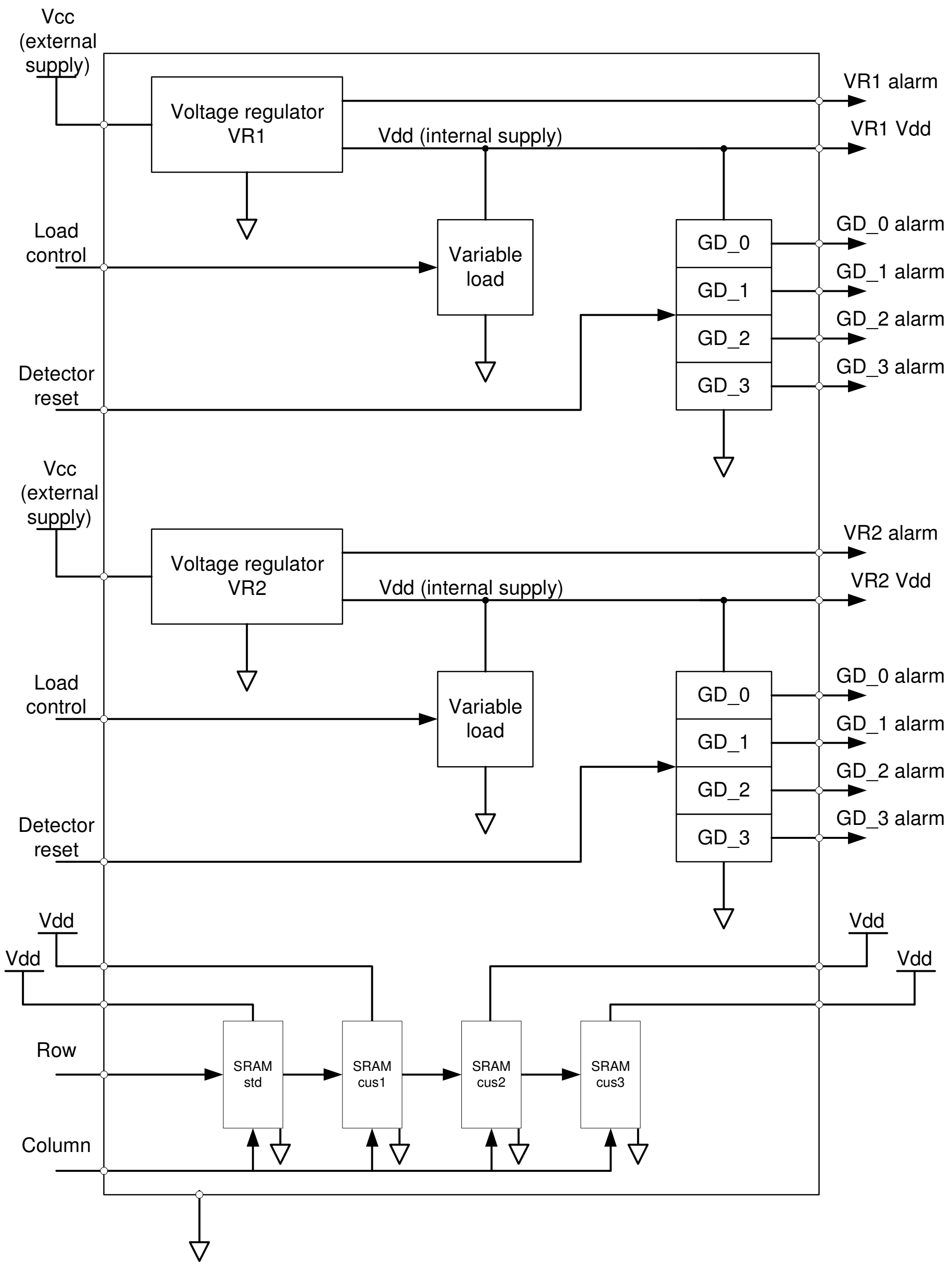


Figure 3 Diagram of the Arrano/01VGA test-chip

Regarding the voltage regulators, VR1 is the same voltage regulator as the one used in the simulation environment and designed for products with low security requirements. VR2 is a voltage regulator designed for products with higher security requirements than the former one. These voltage regulators have different responses to a glitch attack, as shown in Figures 3-15 and 3-16 of the Volume I of this Portfolio.

For each glitch attack case, the output of all glitch detectors connected to the targeted voltage regulator are monitored for detection and detection time, e.g. GD_0, GD_1, GD_2, GD_3 and VR1_alarm. All glitch detector outputs are buffered and connected to output pads in the silicon die. Buffering the glitch output might delay the glitch detection signal reaching the monitoring device. However, this delay should be equal, or similar, for all detectors. Including those built into the voltage regulators.

The test-chip was assembled into an 84-pin PLCC package. The pin list of the Arrano/01VGA test-chip is shown in Table 1. Pins not shown in the table are not connected to the Arrano die.

Table 1 Pinout of the Arrano test-chip

Pin	Name	Description	Direction	Level shifter	ESD	Pad type
12	Vcc_b	Power supply for VR1 (5.5v to 2.7v)	Power	No	Yes	ascvcc
3	Vdd!_b	VR1's internal vdd! (1.6v)	Power	No	No	metal
9	high_b	Input to force high load to VR1	I	Yes[1]	Yes	ascio
7	medium_b	Input to force medium load to VR1	I	Yes[1]	Yes	ascio
5	low_b	Input to force low load to VR1s	I	Yes[1]	Yes	ascio
11	rst_glitch_b	Reset signal for all VR1 glitch detectors	I	Yes[1]	Yes	ascio
23	bgo_1	VR1 glitch detector 1's output	O	No	Yes	esd_pad
22	bgo_2	VR1 glitch detector 2's output	O	No	Yes	esd_pad
20	bgo_3	VR1 glitch detector 3's output	O	No	Yes	esd_pad
18	bgo_4	VR1 glitch detector 4's output	O	No	Yes	esd_pad
17	balarm	VR1's embedded alarm signal	O	No	Yes	esd_pad
1	GND	Ground connector for all circuits	Power	No	Yes	ascgnd
74	Vcc_r	Power supply for VR2 (5.5v to 2.7v)	Power	No	Yes	ascvcc
83	Vdd!_r	VR2's internal vdd! (1.6v)	Power	No	No	metal
77	high_r	Input to force high load to VR2	I	Yes[2]	Yes	ascio
79	medium_r	Input to force medium load to VR2	I	Yes[2]	Yes	ascio
81	low_r	Input to force low load to VR2	I	Yes[2]	Yes	ascio
75	rst_glitgh_r	Reset signal for all VR2 glitch detectors	I	Yes[2]	Yes	ascio
63	rgo_1	VR2 glitch detector 1's output	O	No	Yes	esd_pad
64	rgo_2	VR2 glitch detector 2's output	O	No	Yes	esd_pad
66	rgo_3	VR2 glitch detector 3's output	O	No	Yes	esd_pad
68	rgo_4	VR2 glitch detector 4's output	O	No	Yes	esd_pad
69	ralarm	VR2's embedded alarm signal	O	No	Yes	esd_pad
16	Vram_std	Power supply for standard RAM	Power	No	Yes	vdd

		(1.6v)				
14	Vram_cus_1	Power supply for customised RAM 1 (1.6v)	Power	No	Yes	vdd
72	Vram_cus_2	Power supply for customised RAM 2 (1.6v)	Power	No	Yes	vdd
70	Vram_cus_3	Power supply for customised RAM 2 (1.6v)	Power	No	Yes	vdd
39	data_0	RAM bit line 0 input output	I/O	No	Yes	esd_pad
38	data_0b	RAM bit line 0# input output	I/O	No	Yes	esd_pad
43	data_1	RAM bit line 1 input output	I/O	No	Yes	esd_pad
41	data_1b	RAM bit line 1# input output	I/O	No	Yes	esd_pad
47	data_2	RAM bit line 2 input output	I/O	No	Yes	esd_pad
49	data_2b	RAM bit line 2# input output	I/O	No	Yes	esd_pad
44	data_3	RAM bit line 3 input output	I/O	No	Yes	esd_pad
45	data_3b	RAM bit line 3# input output	I/O	No	Yes	esd_pad
51	data_4	RAM bit line 4 input output	I/O	No	Yes	esd_pad
50	data_4b	RAM bit line 4# input output	I/O	No	Yes	esd_pad
56	data_5	RAM bit line 5 input output	I/O	No	Yes	esd_pad
54	data_5b	RAM bit line 5# input output	I/O	No	Yes	esd_pad
60	data_6	RAM bit line 6 input output	I/O	No	Yes	esd_pad
62	data_6b	RAM bit line 6# input output	I/O	No	Yes	esd_pad
57	data_7	RAM bit line 7 input output	I/O	No	Yes	esd_pad
58	data_7b	RAM bit line 7# input output	I/O	No	Yes	esd_pad
37	row_0	Row 0 select	I	No	Yes	esd_pad
35	row_1	Row 1 select	I	No	Yes	esd_pad
32	row_2	Row 2 select	I	No	Yes	esd_pad
30	row_3	Row 3 select	I	No	Yes	esd_pad
29	row_4	Row 4 select	I	No	Yes	esd_pad
28	row_5	Row 5 select	I	No	Yes	esd_pad
26	row_6	Row 6 select	I	No	Yes	esd_pad
24	row_7	Row 7 select	I	No	Yes	esd_pad
33	Vcc_LS	Power supply for high voltage devices on esd_pad (must be externally regulated)	Power	No	Yes	ascvcc
53	Vdd_LS	Power supply for low voltage devices on esd_pad (must be externally regulated)	Power	No	No	metal

Note 1: Pad's logic powered with VR1's Vcc and Vdd.
Note 2: Pad's logic powered with VR2's Vcc and Vdd.

4.2 Tools and equipment used in the test environment

The different glitch detector versions were tested and characterised in a test environment similar to the one used by the Atmel Smart Card ICs Security Group, which is shown in the Figure 4. Here, the front-end software application, VGlitch, runs and controls the test process. VGlitch configures the Smart Card for each test via the Micropross MP300 TC2 (MP300 for short reference), which acts as an advanced Smart Card Reader. VGlitch also sets up the pulse generator HP81110A to power the Smart Card and to apply different glitches when signalled by a trigger signal. When ready, VGlitch commands the MP300 to generate a trigger signal for the pulse generator, which then applies a burst of glitches to the Smart Card. VGlitch then queries the Smart Card status and records it for later inspection by the test engineer.

VGlitch is a glitch attack characterisation application developed in-house by John Connor, an employee of Atmel. This application connects over the Ethernet network to the MP300 device with the targeted Smart Card attached to it. VGlitch can communicate with the target Smart Card by sending and receiving APDU commands through the MP300. VGlitch also connects to the pulse generator over a proprietary GPIO port to configure the supply and glitch parameters.

The **Micropross MP300 TC2** is a versatile Smart Card testing device. Amongst other features, it enables injection of perturbations into the Smart Card pins, such as glitches in the power supply and forcing individual Smart Card pins to a given voltage value. It can also generate trigger signals. Despite its glitch generation capabilities, the HP81110A pulse generator is preferred for its wider pulse amplitude range.

The **HP81110A** is a Hewlett-Packard/Agilent pulse pattern generator that can generate a pulse in steps of +/-1V over the base output voltage, which is user definable. The signal amplitude limits ranges from 100mV up to 20V. It can generate variable pulse widths ranging from 3.03ns to 999.5s. In the Security Group's setup, pulses are applied in a burst mode after the trigger signal is set high, although a single pulse can also be generated.

The test environment used in this work to test and characterise the new glitch detector is based on this same setup. There are, however, two main differences between both test environments:

- **The target device**. The test-chip has no embedded Smart Card functionality and the targeted glitch detectors can only be accessed directly through the test-chip's I/O ports. The solution was to use the Voyager System, a Smart Card Emulator, to interface between the MP300 and the test-chip. The Voyager System communicates with the MP300 via the standard Smart Card I/O port. The Voyager System's configurable I/O ports interface to the Arrano test board where several test-chips are assembled, including the Arrano/01VGA test-chip. The Voyager System and the Arrano test board are covered in sections 4.2.1 to 4.2.3.
- **The front-end test application**. The Smart Card emulated to test the glitch detectors communicated using different APDU commands than in the Security Group's original

setup. Also, the new test setup was designed to test several test-chips against glitch attacks, hence requiring VGlitch to configure more test parameters than the Security Group's original setup. VGlitch was modified to fit the new requirements. More details on the VGlitch application are provided in section 4.2.4.

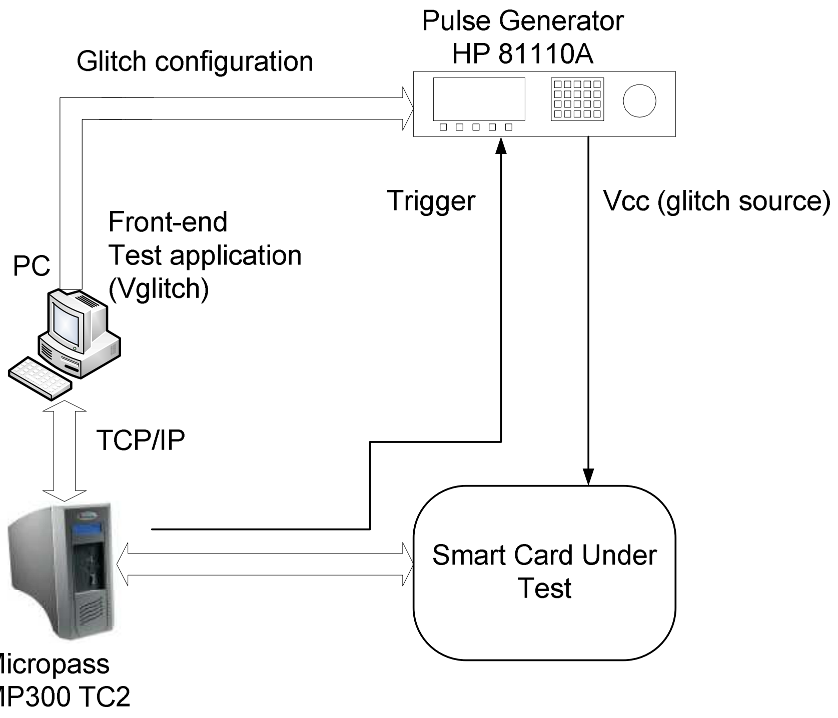


Figure 4 The Security Group's glitch test setup

Figure 5 shows a high level diagram of the test environment used in this work. Figure 6 shows a picture of the actual setup and equipment.

In addition to the front-end application and the target device, with the new test setup, the trigger signal is generated by the target device instead of the MP300. The next few sections cover the target device and the front-end application in more detail. Section 4.2.1 provides a broad overview of the Voyager System and why it has been chosen as part of the test environment. Section 4.2.2 covers the design instantiated into the Voyager System's FPGA that carries the intended tests. The specific APDU commands used by the Voyager System are also covered in this section. Section 4.2.3 provides a high level overview of the Arrano test board, including the different test-chips it targets. The last section, 4.2.4, provides an insight to the changes made to the original VGlitch application.

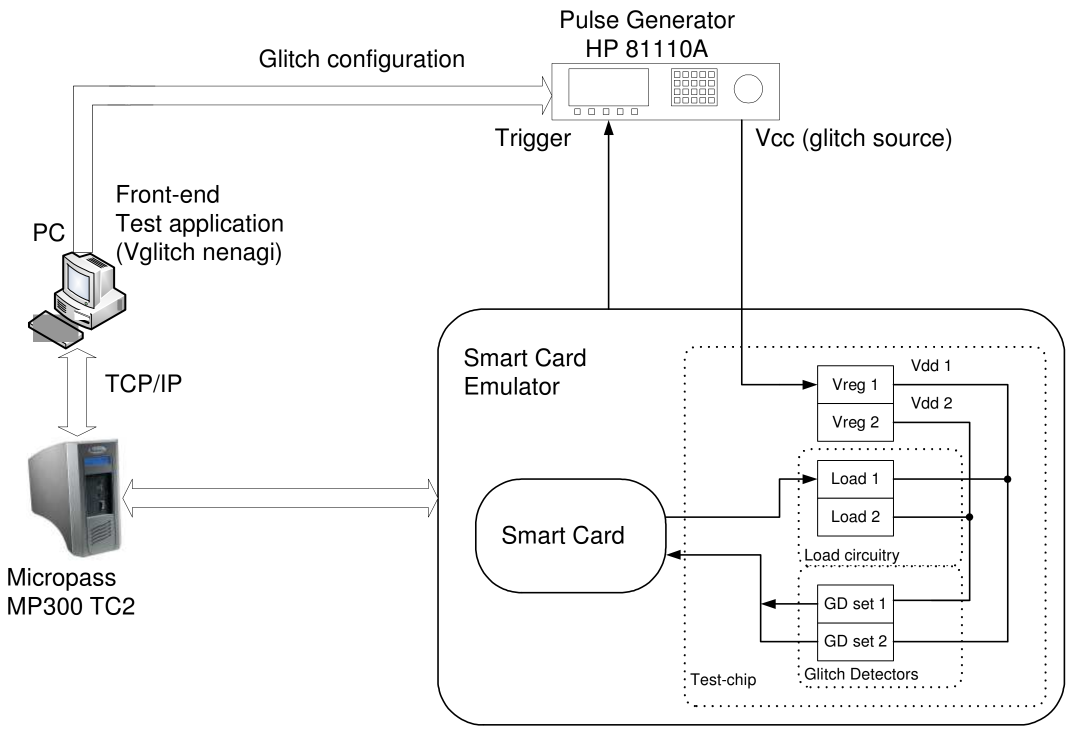


Figure 5 The Security Group's test environment adapted to test Arrano test-chip

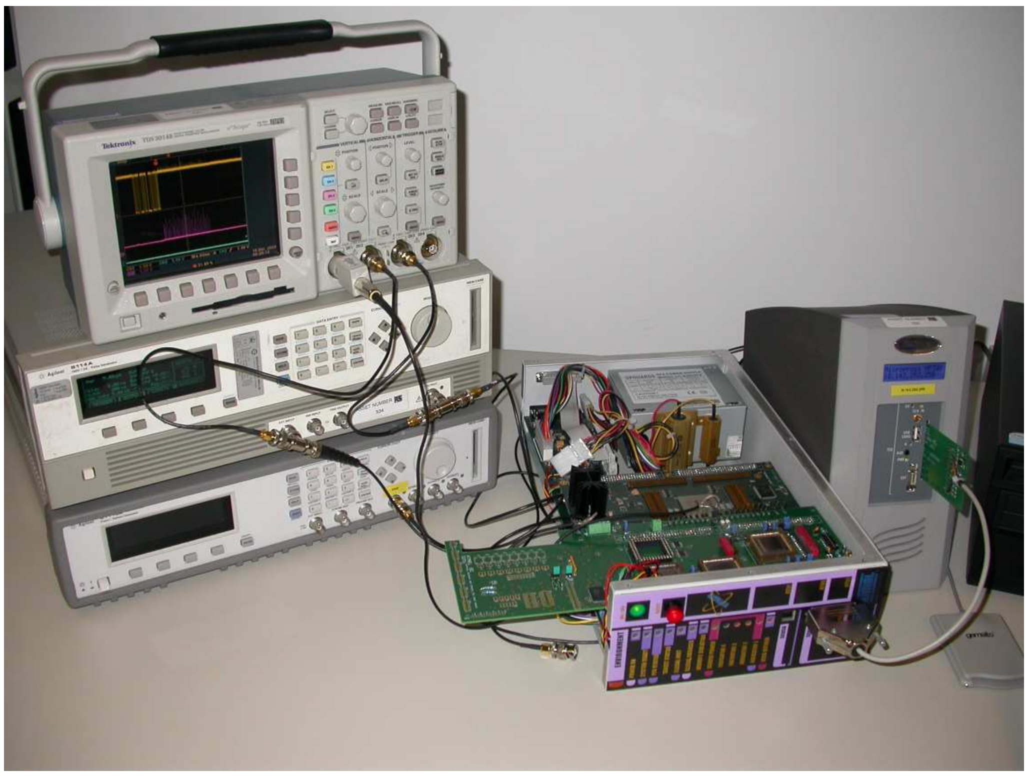


Figure 6 The test setup with used the equipment

4.2.1 Voyager System

The Voyager System is a Smart Card Emulation Platform manufactured by Atmel. It is capable of emulating Atmel's ARM, AVR and HC05 based Smart Cards and provides customers with means to prototype, develop and debug their software code before committing it to Smart Card devices.

The main reasons why the Voyager System is suitable for this test environment are the XC2V3000 Xilinx FPGA which can communicate with the MP300 through a standard Smart Card I/O port and a large set of I/O ports connected to the FPGA, with which test-chips can be configured, controlled and monitored. Furthermore, the availability of instantiating into the FPGA an already designed Smart Card device helps shorten the test environment design time.

4.2.2 The FPGA design

For this test environment, the FPGA in the Voyager System instantiates a version of the Cratis device as used by Jalib to perform DPA on an FPGA emulating a Smart Card that runs encryption operations. The original Cratis design and the one instantiated in this test environment share the following IPs: the AVR3 CPU core, an SRAM memory, a timer and an I/O module. The Cratis version used in this test environment also includes the *periphery_top* module, see detailed diagram of the test environment in Figure 7, which interfaces with the different test-chips in the Arrano test board; see next the section for more information on the Arrano test board.

The module *periphery_top* was purposely designed for this test environment and can carry tests on different test-chips assembled in the Arrano test board. It is placed at the Cratis logical memory address 0x1100 and it can be configured by accessing its registers. Check the *periphery_top* module's code for more details. This code is confidential and only available by Atmel in the following location:
`/home/asierg/derivable/research_data/04_appendix_test_chips/04_03_arrano/code/periphery_top.svn.dump.zip`

Both Cratis implementations run a basic OS developed by the Applications Group. This code enables Cratis to communicate with APDU commands through the standard Smart Card I/O port. Two APDU operations are possible with this OS, write and read data to and from the Data Memory. The APDU commands have the following fields:

Command	Address	Size	Data
2 Bytes	2 Bytes	2 Bytes	*n* Bytes

The ‘command’ field indicates whether this is a read (command = 0x0010) or a write (command = 0x0012) APDU command. The ‘address’ field indicates the base logical address to write to or read from. The ‘size’ field indicates the number of bytes to write or read. Finally, the ‘data’ field is only used with write commands. This field holds the data to be written to the memory address pointed at by the ‘address’ field. In the case of a read command, the data field forms part of the reply sent by Cratis to the MP300. Two APDU command examples are:

- 0012 1102 01 02 → Set a resistive load of 1K6 ohms
- 0010 111F 01 → Read the glitch detection register

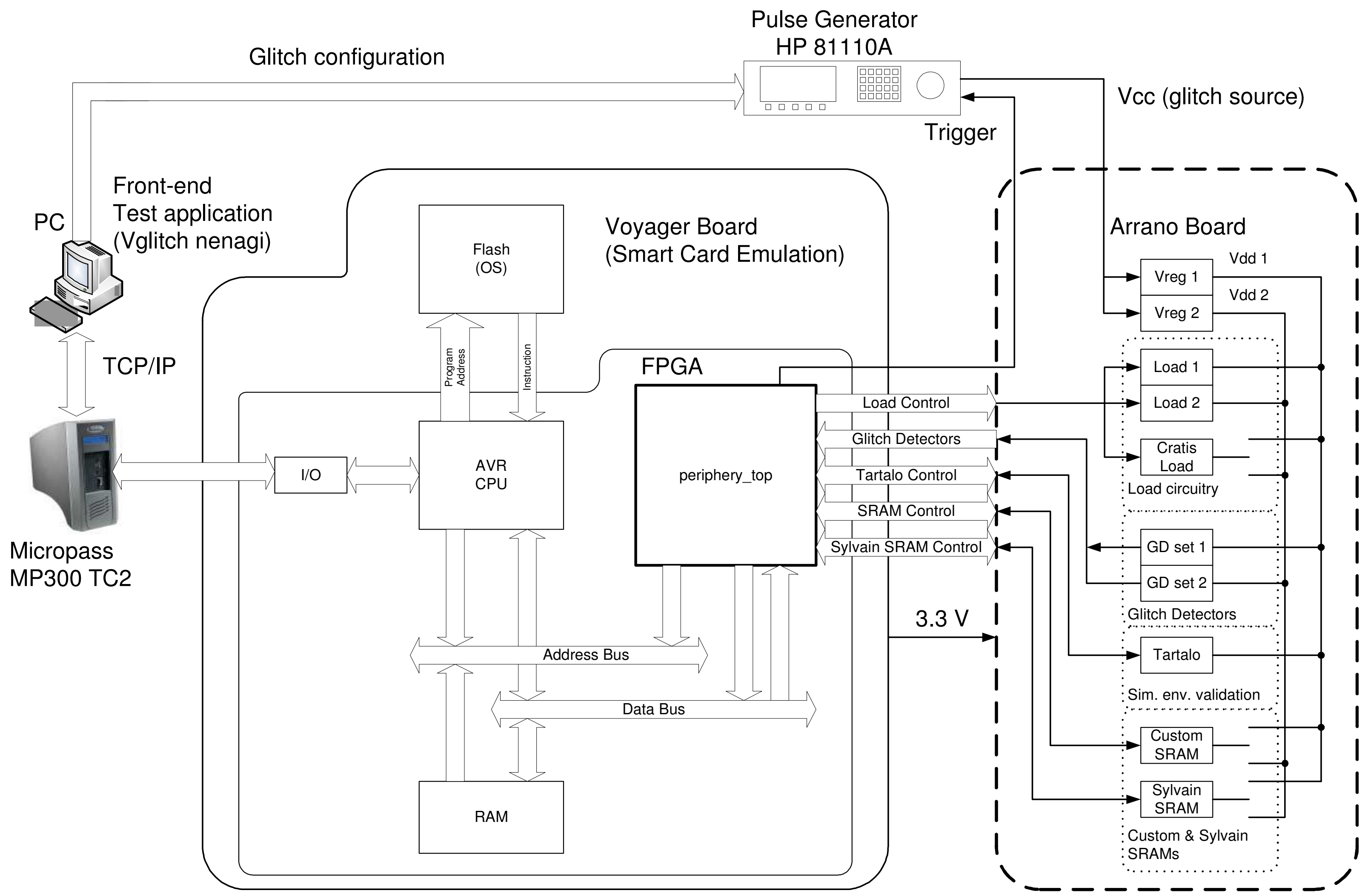


Figure 7 Detailed test environment diagram

4.2.3 The arrano test board

A PCB board with 4 layers (2 signal and 2 power) was designed to test Arrano (01VGA), Tartalo (01OKA) and the 01HLB memory test-chips when subjected to glitch attacks, see Figure 8. The Arrano test-chip is covered in the section 4.1. The Tartalo test-chip is covered in the report LaserTech1. It was assembled into the Arrano test board to characterise the behaviour of its built-in counter, which was powered by the VReg1 instantiated in the Arrano test-chip. For more information about the tests carried on the Tartalo test-chip's counter and the result, check the report SimEnvTech1, section 5.

The 01HLB memory test-chip instantiates six 64KByte memory modules designed with different memory leakage techniques. This test-chip was assembled into this board to test the memory instances' robustness against glitch attacks. However, unsolved issues with the memory access prevented carrying out the desired tests.

In addition to these test-chips, the Arrano test board also allows assembling a silicon instantiation of Cratis, to operate as a load to the voltage regulators instantiated into the Arrano test-chip. The Cratis chip can be configured to run cyclic AES, DES or AES and DES operations.

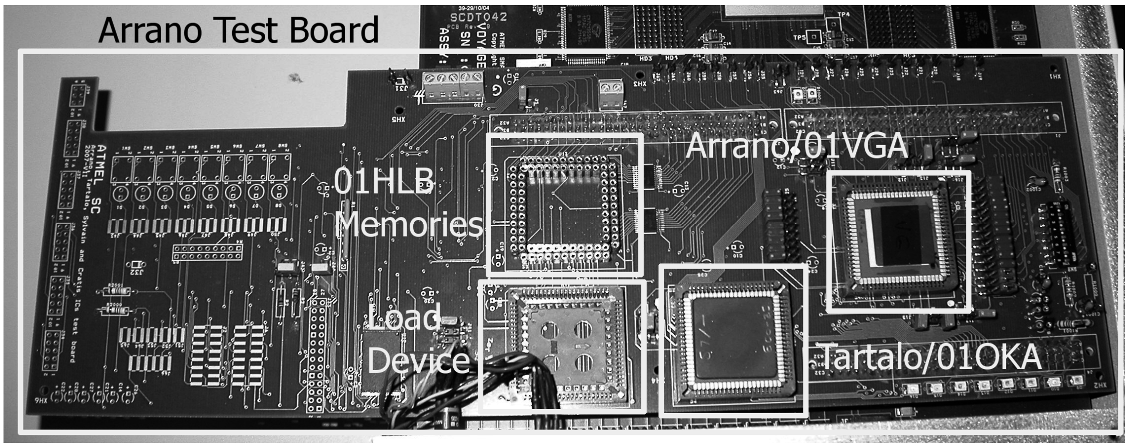


Figure 8 Top view of the Arrano test board

The schematics of this board are confidential and only available to Atmel in the following location: `/home/asierg/derivable/research_data/04_appendix_test_chips/04_03_arrano/schematics/orcad.zip`

4.2.4 The front-end application (VGlitch nenagi 4.14)

The front-end application used to launch and control the tests is '*VGlitch nenagi*', which is based on '*VGlitch 4.12*' developed by John Connor. As previously explained, this application communicates with the Pulse Generator HP 81110A to set glitch parameters such as amplitude, width, amount of glitches in burst mode and delay between glitches. The pulse generator powers the Smart Card under test and injects glitches on receipt of a trigger signal. The application also communicates with the Micropross MP300 TC2 to send and receive APDU commands to and from the Smart Card under test.

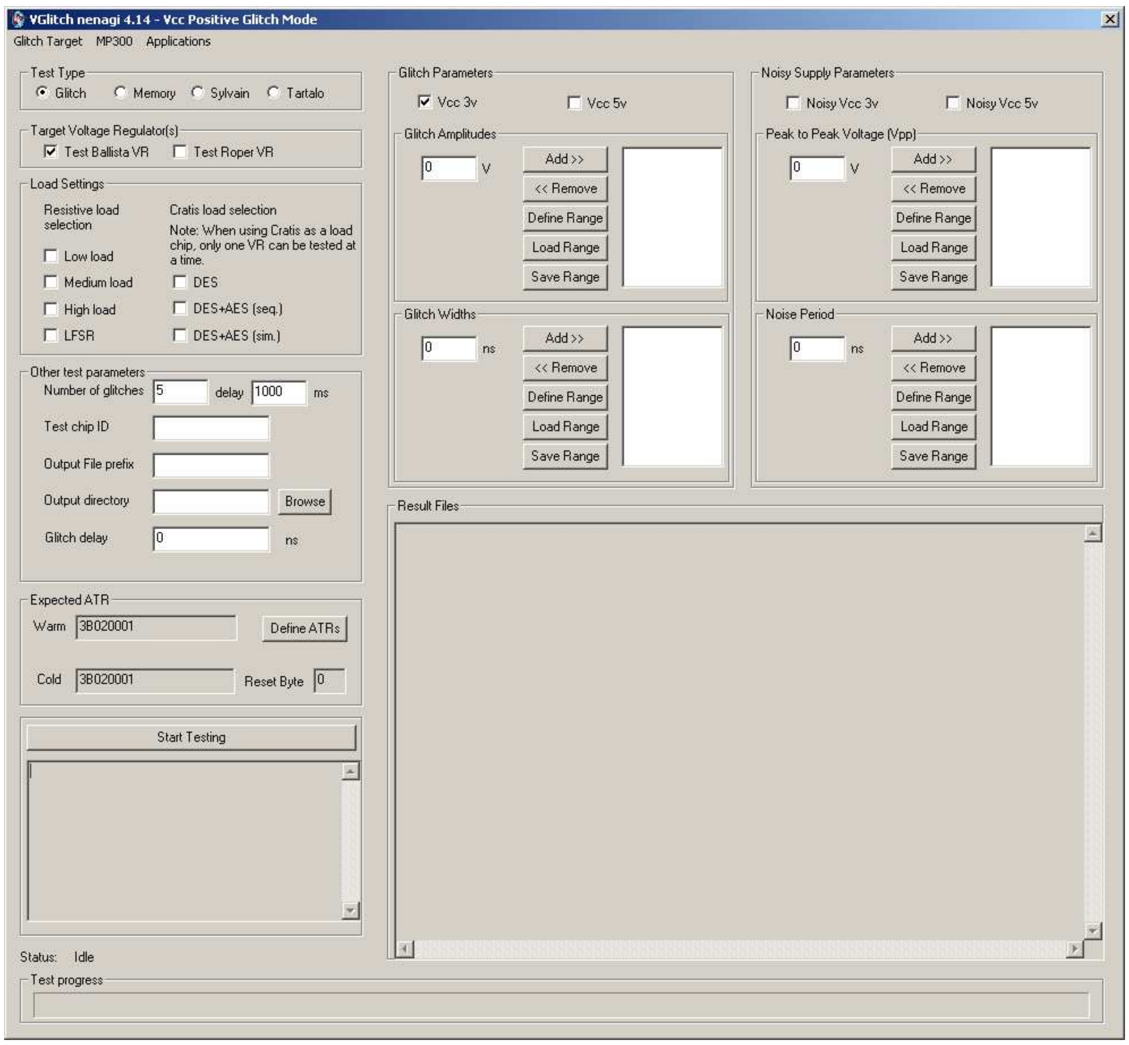
VGlitch nennagi 4.14, the front-end application used on this research line, re-used many of the features of the original application, including the used equipment as well as a similar test flow. However, the change and addition of around 1900 lines of code of the original application was needed to adapt it to the test requirements of this research.

The user interface (UI) of *VGlitch nenagi 4.14* is shown in Figure 9. The application allows selecting between 4 different target tests:

- Glitch, to characterise the glitch detectors' detection range;
- Memory, to test different memory cells instantiated in the Arrano test-chip;
- Sylvain, to test the robustness of the different memory modules instantiated into the 01HLB test-chip; and
- Tartalo, to test the behaviour of the counter in the Tartalo test-chip to validate the simulation environment.

On each test case, the pulse generator can apply a set of glitches and noise over a base supply of 3V or 5V that powers one of the voltage regulators embed in the Arrano test-chip. The load these voltage regulators are subjected to is defined in the *Load Settings* field, see Figure 9. Low, Medium, High and LFSR loads are achieved with the resistive loads build into the Arrano test-chip. DES and both DES+AES loads are achieved by running encryption/decryption operations on the Cratis load device powered by one of the voltage regulators embed into the Arrano test-chip.

The normal operation of *VGlitch nennagi 4.14* is as follows: lock the access to the MP300 and the pulse generator; for each base supply case, for each load scenario and for each glitch, apply a number of glitches and check back the target device's status. For each supply case and for each load scenario, create a CSV file with the device status after each glitch attack.



Test Type: (● Glitch) ○ Memory ○ Sylvain ○ Tartalo

Target Voltage Regulator(s): [✓ Test Ballista VR] ☐ Test Roper VR

Load Settings

Resistive load selection: ☐ Low load ☐ Medium load ☐ High load ☐ LFSR

Cratis load selection: Note: When using Cratis as a load chip, only one VR can be tested at a time. ☐ DES ☐ DES+AES (seq.) ☐ DES+AES (sim.)

Other test parameters

Number of glitches: 5 delay: 1000 ms

Test chip ID:

Output File prefix:

Output directory:

Glitch delay: 0 ns

Expected ATR

Warm: 3B020001

Cold: 3B020001 Reset Byte: 0

Status: Idle

Test progress

Glitch Parameters: [✓ Vcc 3v] ☐ Vcc 5v

Glitch Amplitudes: 0 V

Glitch Widths: 0 ns

Noisy Supply Parameters: ☐ Noisy Vcc 3v ☐ Noisy Vcc 5v

Peak to Peak Voltage (Vpp): 0 V

Noise Period: 0 ns

Figure 9 VGlitch nenagi 4.14 user interface

The source code of VGlitch nenagi 4.14 is also confidential and only available to Atmel in:
`/home/asierg/derivable/research_data/04_appendix_test_chips/04_03_arrano/code/vglitch_nenagi.svn.dump.zip`

Engineering Doctorate

SRAM Memory Partitioning for Leakage Reduction

Author: Asier Goikoetxea Yanci

Date: 2007-09-27

Sponsored by:

Table of contents

Abbreviations

APDU	Application Protocol Data Unit
API	Application Protocol Interface
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DES	Data Encryption Standard
DRAM	Dynamic RAM
DRV	Data Retention Voltage
EEPROM	Electronic Erasable Programmable Read Only Memory
IP	Intellectual Property
ISO	International Standards Organisation
MMU	Memory Management Unit
NVM	Non-Volatile Memory
OS	Operating System
RAM	Random Access Memory
RNG	Random Number Generator
SC	Smart Card
SCD-A	Smart Card Device A
SCOS	Smart Card Operating System
SoC	System on Chip
SPI	Serial Peripheral Interface
SPM	Scratch-pad Memory
SRAM	Static RAM
TDES	Triple-DES

1 Introduction

Newer technology processes result on smaller devices, higher integration capacity and an increase in performance. However, these benefits are also acompained by other undesirable features, such as process variations across the wafer and between different wafers, and an increase of leakage current – source of the static power, specially undesirable in portable and contactless applications, as it wastes the battery's life. As an illustration, when the cryptographic module is performing an encryption or decryption operation, the rest of the device goes to idle mode in order to save on dynamic and static power consumption, specially from SRAM. Hence, adopting newer technology should be made by minimising its impact on the leakage current.

The two main sources of leakage are: a) sub-threshold leakage; and b) gate tunnelling. Reducing transistor geometries forces a reduction on the supply voltage (Vdd) for two reasons; the first one in order to avoid damaging the device, the second one in order to reduce the dynamic power. Reducing Vdd, on the other hand, has a negative impact on the gate's performance, as it gets poorer with it. The performance of a gate can be boosted by reducing its threshold voltage (Vt). However, doing so would result on not fully turning the device off, i.e. going into weak inversion, and making the device leaky. This creates, then, the paradigma of sacrificing power for performance or vice versa.

Gate tunnelling, in the other hand, has a more direct link with the geometry than the sub-threshold leakage. As devices get smaller and Vdd reduced, the devices' gate oxide also gets thinner. Thinner gate oxides are needed to induce the electric field or channel for the current to flow through from the source to the drain. The side effect of thinner gate oxides, however, is an increase of the gate leakage due to the proximity of the gate and bulk.

Plenty of effort has been made toward minimising the leakage, ranging from foundry level to the software or application level. SOI wafers are said to be less leaky than CMOS ones [1]. Transistor design improvements such as fin FET transistors [2] or high dielectric (high-k) gate oxides [3] have also been suggested. Design wise, high-Vt and low-Vt transistors can be mixed so that, low-Vt ones (more leaky) are used for critical paths, and high-Vt ones (less leaky) are used for the rest of the logic [4].

Higher level leakage reduction techniques involve system level techniques that can be transparent to or exploited by the software. Dynamic voltage scaling (DVS) and dynamic frequency scaling (DFS) are popular techniques that allow reducing the dynamic and static power consumption by dynamically setting the device's maximum performance [5]. Reducing the frequency reduces the performance and dynamic power consumption. Reducing the voltage reduces the performance, dynamic and static power consumptions.

SRAMs are compact intellectual properties (IP) with a high transistor density and, in small system-on-chip (SoC) devices, they often account for more transistors than the rest of the logic design. This makes them the highest source of leakage current on these SoCs, which is set to worsen in the future with the moving to newer technology nodes and an increasing amount of SRAM within the SoC.

Several techniques, other than those already mentioned, have been developed to reduce the leakage contribution of on-chip memories such as changing the cell bit structure [6, 7]. Another popular technique is combining memory partitioning (replacing a monolithic memory with smaller sized ones) with power management techniques [8, 9], which allows saving power via hardware techniques and clever memory usage by the software applications. This technique has extensively been used in cache and scratch-pad memories (SPMs) for reducing both dynamic and static power consumptions and usually impacts the compilers, as they need to be aware of the memory partitions.

The dynamic power consumed by a circuit is defined by its effective capacitive load, the supply voltage and the frequency the circuit is exercised at, just as defined in (1). Smaller memory arrays have a smaller capacitive load for read/write operations than bigger ones, hence, partitioning or dividing a monolithic memory into smaller size memories results in reducing the dynamic power consumption [8]. The static power consumed by a circuitry, on the other hand, depends on the supply voltage and its leakage current. This technique typically reduces the static power by lowering the supply voltage level of the partitions not being accessed [9, 10]. The reduced supply voltage needs to be above the data retention voltage (DRV) if the contect of these partitions is not to be lost [10, 11].

$$P_{dynamic} = C_{eff} \cdot Vdd^2 \cdot F_{clock} \qquad (1)$$

1.1 This works' scope

In the particular case of Smart Cards, SRAMs are also the single highest source of leakage. The high leakage restriction imposed on Smart Cards used in mobile and contactless applications, the tendency to increase the amount of SRAM in newer Smart Card products and the increase of leakage current with newer technologies, drove the need to reducing the power consumed by SRAMs.

The Atmel Memory Group proposed partitioning the system memory and powering the partitions according to their use. The power policy proposed by the Memory Group include the following:

- **In-use:** A partiton is in-use when any data held within this partition is being accessed either for a read or write operation. On this circumstance, the partition should be fully powered to avoid any performance penalty.
- **Data retention:** When a partition holds valid data but is not being accessed, it is said to be in data retention mode. While on this mode, supply voltage should be reduced so that leakage is minimised, but always above the DRV, to avoid loosing any data.
- **Not used:** A partition is not used when it does not hold any valid data. In this case, the partition's supply can be turned off to avoid any contribution towards the system's leakage current.

This work looks at the applicability of above techniques to the SRAMs built into Smart Cards. The remaining of this report is divided as follows; Section 2 provides with an overview of previous works. Section 3 covers the different aspects that can determine the usage of the data memory on a small embedded system, such as Smart Cards. Section 4 focuses on the case study of the Smart Card Device A (SCD-A) running the Smart Card OS (SCOS). Section 5 includes a discussion on what benefits could be obtained on both, SCOS and other OS and application cases, by using a partitioned memory. Section 6 draws the conclusion obtained from this research line and, Section 7, proposes some future line of action in this topic.

The outcome of this work is the report 'Proposed Memory Partitioning Approach for SRAMs in Smart Card Devices', which focuses on determining the partitioning approach, partition size, powering policy, impact on the scrambler and other aspects that need to be considered when applying memory partitioning.

2 Related work

Please refer to sections 4.1 Literature Review and 4.2.1 Literature Review in the Volume I of this Portfolio for details on related works.

3 Determining factors of the memory's usage

The data memory usage of an embedded system depends on several factors, one of the main factors is the embedded system's target application. This includes whether the embedded system is a single- or multi-application and, in the later case, the number of applications that can be run concurrently at any one time. This factor is, in great measure, tightly coupled to the OS running on it.

Other factors that also influence the memory usage are shown in Figure 3-1. This section descrives how some of these factors can determine the memory usage.

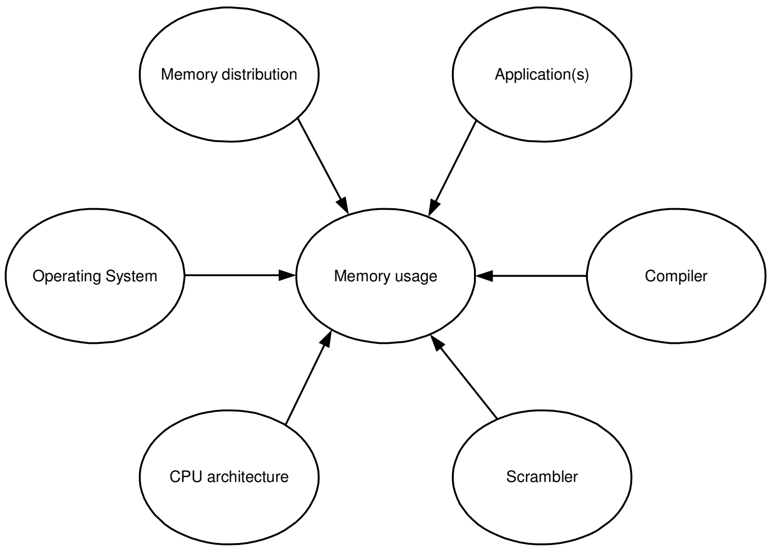


Figure 3-1 Factors that influence on the memory usage

3.1 *Operating Systems and Applications*

3.1.1 General overview

An OS is software that interfaces between the application programs and the hardware, abstracting the application from the underlying hardware and providing a common interface and description for different hardware. A typical computer system structure is depicted in Figure 3-2, where the OS is divided into three different layers: device drivers; hardware abstraction layer (HAL); and the core of the operating system.

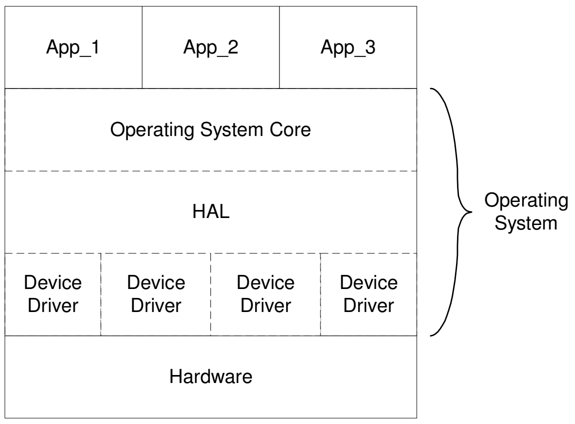


Figure 3-2 Typical structure of a computer system

The device drivers provides some call routines or application program interfaces (APIs) so that the layers above can mask the particular implementation of the hardware feature that is accessed via the device driver. Device drivers could be used to control and/or access I/O devices, memory and other hardware functions via these routines.

The HAL further abstracts the hardware from the operating system, providing a common platform on which the OS is run. This layer's function is to mask, from the operating system core, as many hardware specific features as possible, hence, making the OS as hardware independent as possible. In turn, this facilitates porting the core OS to a different hardware by minimising or avoiding the impact in the core OS. If a particular OS does not implement this layer, the OS core would be affected on each port to a higher or lower degree, depending on how the hardware architectures differ.

The operating system core implements all the different rules and policies under which the system is controlled or managed. These rules and policies can include application scheduling, hardware management, I/O management, memory management and many more. The particular implementation and features can vary from one OS to another. The next sub-section covers the particularities of different OSs used on Smart Card applications.

The OS core also provides the applications with a standardised API, so that they can use the general functions that are provided by the OS. For example, allow access to files and directories, or hardware specific functions, such as using a co-processor or a communication port.

When the previous layers are well defined, the OS can run applications developed by different parties, as is the case with modern computers. These applications do not need to know the hardware's ins and outs, but instead, use the OS to exploit them when necessary. How the applications use the hardware will greatly depend on the application being run. Take for instance a computer system with a multimedia co-processor. A video player application could be using this co-processor to boost the playback performance. Such an application could, perhaps playback media from different sources, e.g. locally stored, broadcasted or streamed. For each media source, different hardware will be required. A plain text editor, on the other hand, will have much lower hardware requirements.

The memory usage of different applications can also vary. For example, the memory required by a plain text editor application is obviously much lower than that required by a video player application.

Generally, these concepts also apply to Smart Cards. The following sub-section shows how these concepts are applied to Smart Cards, and how these and a given set of particulars of the Smart Card applications can effect on the usage of the memory.

3.1.2 Smart Card Operating Systems and Applications

Operating systems running on Smart Cards are a scaled down version of the OSs described above, adapted to the limited resources and target applications of Smart Cards. These OSs can be divided in two groups: native and non-native OSs. Native OSs are usually developed by Smart Card vendors and are tightly coupled to the hardware they run on. These tend to be proprietary operating systems, where running a third party application is extremely costly and difficult.

Non-native OSs, such as STARCOS and MULTOS, on the other hand, are not fixed to any particular hardware. These are commercial off-the-shelf (COTS) OSs that can be used on a wide range of Smart Cards, partially independently from the Smart Card manufacturer and vendors. Partially, because not all non-native OSs have been ported to all Smart Cards and also because the use of a given OS is decided by the vendor depending on the Smart Card’s target application. Unlike native OSs, non-native OSs potentially allow third party applications to be run on the Smart Card. They also can facilitate the installation of new applications once the Smart Card has been deployed.

There are different reasons why a vendor will be more inclined to use a native or a non-native operating system. The main reasons are the target application and the costs of scale. Smart Card devices can be classified as single-application or multi-application. Single application Smart Cards are usually programmed to target a single, particular application through its life-time, whereas multi-applications tend to target several applications. Multi-application ready OSs can also enable executing the applications either one at a time or several applications concurrently.

In general terms, native OSs are primarily single-application whereas non-native OSs are primarily multi-applications, and just as with PCs, different applications will result in different memory usage and even different footprints. However, unlike native OSs, vendors need to pay royalties for using non-native OSs, and this might increase the product cost.

Again, similar to PC OSs, Smart Card OSs should and do differentiate between OS and application memory. For security reasons, multi-application OSs should define application level memory data access boundaries. Hence, not allowing a given application to access another application's data.

The OS of small devices can be identified by yet another parameter – the data linkage – which can be variable or static. A variable data OS allows size-variable data or allocating data or memory dynamically, e.g. a variable array or buffer. These devices need a memory management unit (MMU), which could be instantiated in hardware and/or software. A static data OS only allows fixed variable data, which is fixed to a defined memory location at compile or execution time. Both OS types will also yield different memory usage patterns.

3.2 Compilers

The programming language, and hence the compiler, impacts where the variables are located in the memory. When programing for a MMU-less CPU/OS, i.e. a data static OS, the data is allocated to certain memory areas defined by pragma directives in the source code. This allows, for example, the targeting of specific memory sections to run certain functions. The variables defined within the pragma directives could be considered as C static variables. C compilers also use the C-Stack to store all other variables that are not defined within pragma directives. This could be considered as C volatile variables. For the case of data variable OSs, C compilers use the heap to store variable-size data when invoking the `malloc` function or similar.

Other programing languages can use different polices to store variables. For instance, Java compilers place all variables in the heap.

3.3 Scrambler

Among other security features, Smart Cards include a memory scrambling unit. This unit's function is to scatter the memory's logical addresses in the physical domain, as well as the data bit order. The result is that continuous addresses in the logical domain may no longer be continuous in the physical domain. A visual example of this scattering is shown in Figure 3-3.

The memory scrambling differs every time the Smart Card is powered up or reset. This means, the scattering is unpredictable. Hence, even if the same application is run several times, each time accessing the same logical addresses, the physical addresses accessed may differ for each different run.

It must be noted, however, that the scrambler only alters the spatial domain of the memory. The temporal domain is not affected by the scrambler, therefore, the amount and duration of IDLE states remain unchanged every time a given piece of code is run.

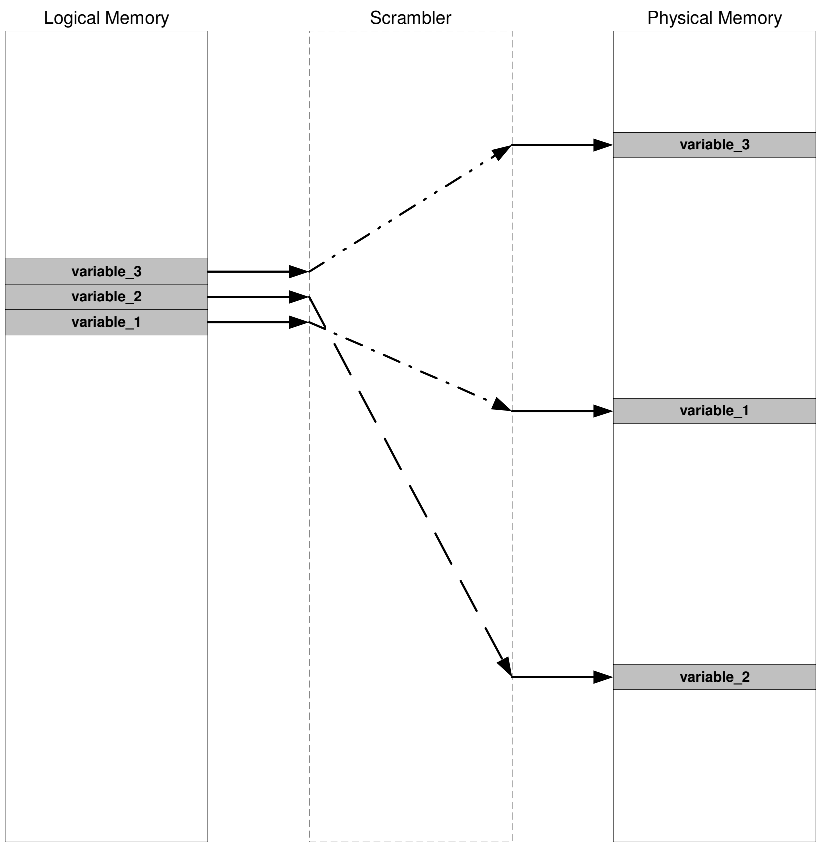


Figure 3-3 Example of how a scrambler could scatter logical addresses in the physical domain

Memory scrambling is another issue to be solved in order to efficiently use a partitioned system memory, as using the current scrambling approach does not take partitions into account. This means, memory addresses belonging to a given logical partition could be scattered across different physical partitions, decreasing the effectiveness of this leakage reduction technique.

3.4 CPU architecture

Different CPU architectures might also have an impact on the memory usage. Current CPUs used by Atmel Smart Cards mainly focus on 8-bit architectures, although a few 32-bit CPUs have also been introduced. In the 8-bit domain, the CPU accesses the SRAM data directly, usually within a clock cycle, and without any intermediary. This means that if a particular variable (other than those stored in the internal general purpose registers) is to be read or written to, the system SRAM will always be accessed.

In the 32-bit domain, on the other hand, two options are possible regarding the system memory and its usage; a CPU with cache or a CPU without cache (or SPM). 32-bit CPUs without cache memory (or SPM) will behave in the same fashion as the 8-bit CPUs, where the system RAM is directly access when a variable is required.

CPUs with cache memories will have a different usage of the system memory, it will only be accessed in the event that the variable the CPU is looking for is not in the cache memory. Furthermore, SoC using such a CPU could embed a slower system RAM, be it SRAM or DRAM. This implies the need for wait states when accessing the system memory.

Although at the time of writing this report there is no knowledge of any Smart Card using a CPU with cache memory, this sort of CPU might be used in the future with the introduction of Smart Cards with a higher memory capacity and with higher performance (e.g. USIM).

Because the Smart Cards developed at Atmel are based on 8-bit CPUs and 32-bit CPUs without cache memory, the memory use of these types of CPUs is only considered in this report.

3.5 Summary

This section has shown how the memory usage is affected by the different decisions taken at different levels. Due to the wide range of applications where a given Smart Card can be used, and due to the lack of in-depth knowledge (and the possibility of gaining this knowledge) of the different OS and applications that run on Smart Cards, their memory usage must be determined based on information available in house. This is covered in the next section.

4 SRAM usage case study: SCD-A running SCOS

A case study was setup to gather knowledge of the actual memory usage pattern of Smart Cards. Since Smart Card manufacturers do not have access to their customers' native OSs and even less so to their applications, nor in this case was there access to a non-native OS, this case study was run using an Smart Card OS (SCOS) developed by Atmel's Applications division in Rousset, France. This SCOS was developed for Smart Cards to be used as cryptographic co-processors in embedded systems. This OS was developed for the Smart Card Device A (SCD-A), hence, this was the one used in this case study.

Due to confidentiality issues, the OS used in this work will be referred to as SCOS, and the Smart Card will be referred to as SCD-A. The following subsections contain a description of the SCD-A and SCOS. This is followed by a description of the data gathering environment and the obtained results.

4.1 About SCD-A

The SCD-A is a Smart Card developed to target several applications, including GSM, banking and government applications. These are its relevant features:

- High-performance, Low-power, secure AVR RISC architecture microcontroller
- Compliant with GSM, 3CGG and EMV2000 Specifications
- 144 Kbytes Flash
- 144 Kbytes EEPROM
- 8 Kbytes SRAM
- Two ISO7816-3 compliant I/O ports
- SPI interface
- RNG (Random Number Generator)
- Hardware DES/TDES
- 32-bit Cryptographic Accelerator for Public Key Operations
- CRC

4.1.1 Data memory map

Figure 4-1 shows a diagram of the SCD-A's data memory. The data memory is divided in two main sections: volatile and non-volatile memory. The EEPROM represents the non-volatile memory section of the memory map, whereas the SRAMs and registers represent the volatile memory. Focusing on the SRAMs, Standard and Shared memories make up the 8-Kbyte mentioned in the specification.

These SRAMs have a very distinctive function. The Standard Memory (6-Kbyte) is the system memory, which will be used exclusively by the CPU. The Standard Memory is in fact a combination of two 3-Kbyte memories. Built in 0.18µm technology and powered at 1.8V, each memory has a leakage current of 2.1µA and a dynamic power consumption of 17µW/MHz. On the other hand, the Shared Memory (2-Kbyte) is shared between the CPU and the advanced multiplier (AdvX).

The Shared Memory is a single-port SRAM, so only one processing unit can access it at a time, either the CPU or the AdvX. When the AdvX starts its execution, the CPU cannot access this memory until the AdvX computation has finished. Despite this, nothing stops a software engineer from writing code that uses this SRAM as a scratchpad, i.e. storing temporal system data into it. For example, executing the application protocol data unit (APDU) command does not involve running the AdvX. However, this is not considered to be the normal procedure by Atmel customers, as doing so would introduce further complexity into the OS and the memory management.

As a result, the use of the Shared Memory is not considered in this case study. Despite omitting this SRAM in this analysis, the same leakage reduction techniques could be applied and are recommended for this memory too.

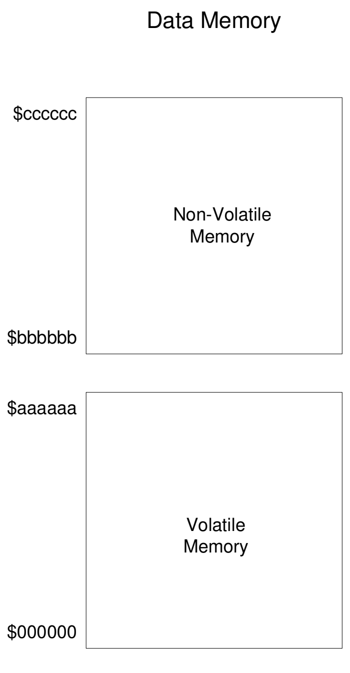


Figure 4-1 SCD-A's Data Memory Map

4.2 About SCOS

The SCOS is an operating system developed by Atmel to run on its Smart Card products. This OS has been ported to a series of devices and it was used for this case study as the only available OS for a Smart Card. The characteristics of this OS are described below.

Data static OS: Every data is of a fixed size. There is not dynamic data allocation nor variable size arrays. The size and location of each data is designated either at programming or compile time. Once the software has been compiled, the data size and logical address will not be altered.

Application data is stored in a size limited C-Stack. The amount of stack to be used by the application is determined by the application itself. Again, this data will be defined at programming or compile time. Dynamic allocation of memory is not allowed, however, memory addresses of local variables belonging to finished or expired functions might be used as local variables by other functions.

Application static OS: Once the Smart Card has been personalised, the OS does not allow more applications to be loaded or deleted.

The **communication** with the external world is carried out via an SPI interface, rather than the standard ISO I/O interface of the ISO 7813 Smart Cards.

Figure 4-2 shows how the SCOS divides the SCD-A system memory. Seven main sections can be identified: an unnamed section; DATAVOL_LIB; DATAVOL_OS; DATAVOL_APP_AUTH; a not used section; RSTACK and CSTACK. The last section, CSTACK, is where the application stores its variables. The RSTACK is used by the software's flow. All other sections are used by the OS layer and/or any other software layer below the OS. As it can be seen from this figure, all in all, the SCOS only defines sections that take in total slightly less than half the whole system memory. By powering off the non-used segments of the SRAM, the leakage consumption could be almost halved.

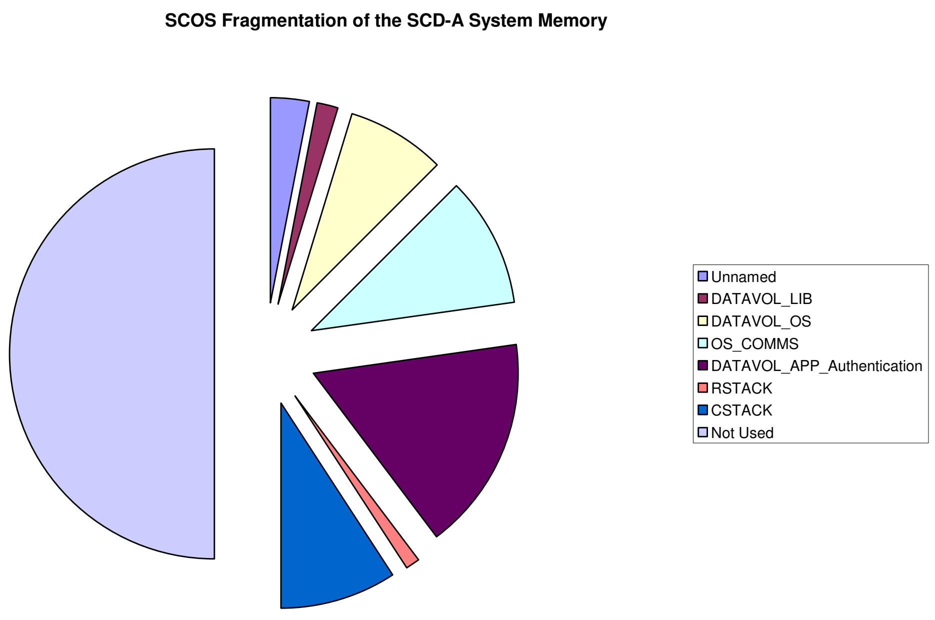


Figure 4-2 SCOS System Memory Map

4.3 Data gathering environment setup

In order to decide the best memory partitioning approach, the SRAM utilisation needs to be studied in the following terms: the accessed SRAM addresses; the access times; and the IDLE times. The best way to gather the required information for this research is to execute applications on a Smart Card and monitor the SRAM usage. In order to have full access to the SRAM address bus, this data gathering process needs to be based on simulations. Hence, a Smart Card model; an OS with applications; and a test setup or environment capable of simulating real transactions and monitoring the SRAMs were needed.

Atmel already had a simulation environment for testing the Smart Cards at system level. This environment, Figure 4-3, instantiates an RTL/HDL Smart Card and can operate in either tester or reader mode. Atmel carries out all their tests by running the test environment in tester mode. The tester module communicates with the Smart Card through the ISO I/O port to perform behavioural tests on the Smart Card.

The test environment for the data gathering process should simulate the Smart Card in a real environment. The Smart Card should carry or execute transactions sent by an external application through a Smart Card Reader. The reader test mode in Atmel's test environment was not operational and only the top level instantiation of this operation mode was implemented. Development of the reader test module was required for this experiment.

Furthermore, Atmel's test environment does not collect any information on the use of any of the IPs in the Smart Card, hence, implementation of the SRAM usage monitoring feature was also required.

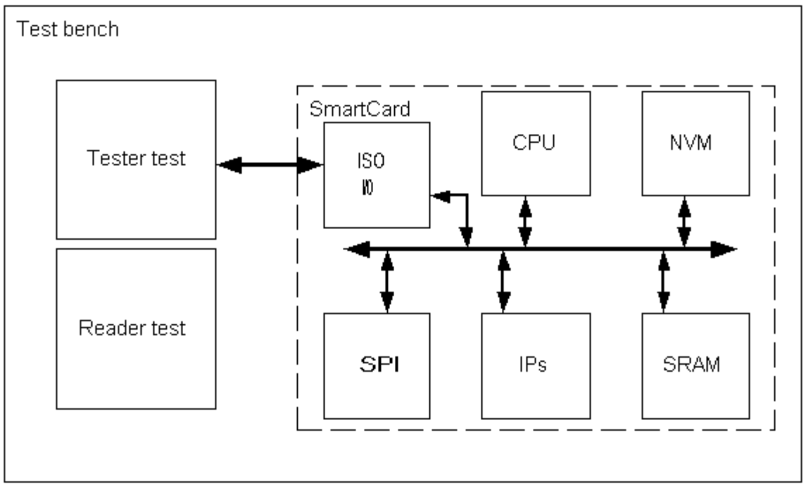


Figure 4-3 Diagram of Atmel's standard test environment for Smart Cards

In addition to defining the Smart Card device to be used, the SCOS also restricts which communication channel between the Smart Card and the Smart Card Reader can be used. Instead of the standard ISO I/O port used by Smart Cards, the SCOS communicates with the external world via the SPI port. This has no impact on the Smart Card itself, but it does effect the test environment. Another implication of using this OS is the communication protocol. It uses the Block Protocol Stack, which is a proprietary wrapper protocol for the standard APDU commands and was developed by Atmel

In summary, Atmel's test environment needed to be updated so that it could work as a reader, running applications and communicating with the Smart Card through the SPI port as well as using the Block Protocol Stack. An update to monitoring the I/Os of the SRAM IP was also required. Finally a Perl script was developed to generate the statistical information of the memory's usage. Figure 4-4 shows the block diagram of the resulting reader mode. Figure 4-5 shows the final test setup used for these tests.

Appendix A provides a more in depth knowledge of the test environment. Appendix B lists the features of the SRAM analysis Perl script.

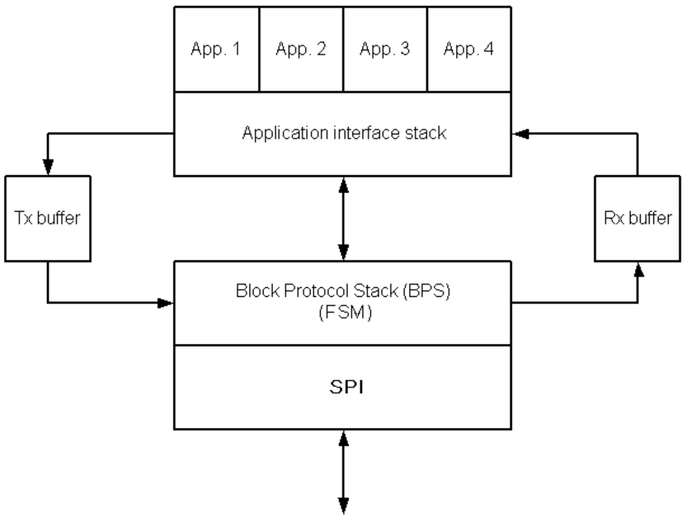


Figure 4-4 Reader's block diagram

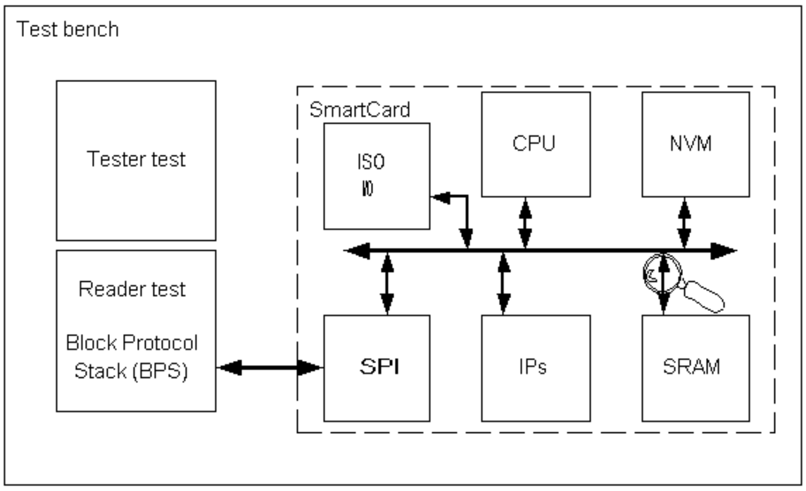


Figure 4-5 Block diagram of the used test setup

4.4 Test results

Tests were performed by running three different applications on the Smart Card:

- Smart Card personalisation
- random number generation
- some basic APDU commands

The Smart Card personalisation test consisted of the reader sending data (i.e. applications) to be stored in the non-volatile memory and to change the Smart Card IC's life cycle status. Smart Cards are personalised with the applications and data to be used during their life cycle. The random number generation test generated a series of random numbers, from 1-byte to

256-bytes long. The APDU command test consisted of selecting a test application and sending four test APDU commands. The statistical logical SRAM usage for each of these tests is shown in Table 4-1. For each application the data generated by the SRAM analysis script was too large to be shown in this report, therefore, Appendix C shows only the SRAM usage data that was generated when running the RNG application.

Table 4-1 Logical SRAM usage simulation results

		APDU	**RNG**	**Personalisation**
Simulation time		12,545,825ns (88,618 cycles)	3,205,569,757ns (19,774,170 cycles)	750,729,977ns (5,592,126 cycles)
Total access time	read	10.97% (9,718 cycles)	4.52% (894,201 cycles)	8.67% (484,817 cycles)
	write	10.51% (9,316 cycles)	3.56% (703,342 cycles)	6.58% (367,998 cycles)
Read access times	20400 ps	28.49%	1.16%	63.32%
	142800 ps	71.42%	98.84%	36.67%
	202560 ps	0.09%	0.00%	0.00%
Write access times	20400 ps	24.95%	1.30%	63.84%
	142800 ps	74.81%	98.70%	36.15%
	202560 ps	0.25%	0.00%	0.01%
% of accessed addresses	read	1.95%	8.08%	6.23%
	write	7.72%	9.89%	8.11%
Block sizes	1	7,319	990,299	388,442
	2	-	1	-
	3	3,905	202,414	154,791
IDLE states	Total amount	11,224	1,192,714	543,234
	Mean	7	16	9
	Deviation	29	61	47

This table shows the total simulation time for each of the test cases, which depends on the test itself. The total amount of time spent on read and write operations are also presented, this time is shown as a percentage of the total simulation time.

Another piece of information shown in this table is the different read and write access times and the percentage of occurrences for each of these access times. These are a percentage of the total read and write times. Note that few of the read/write access times show a 0.00% value. This is due to the small amount of memory accesses at these access times.

The total amount of the addresses accessed for read and write are also presented. These are presented as a percentage of the total SRAM addresses accessed. Finally, block size occurrences are shown. Block size represents the amount of consecutive memory address accesses.

The logical and physical memory use distributions are shown in the figures below. The logical addresses in Figure 4-6 and the physical addresses in Figure 4-7. The colours indicate the activity of each address for each application, where blue indicates a low activity, at least one access, and red indicates a high activity. APP. 1 shows the memory usage for the simple APDU command. APP. 2 shows the memory usage for the Personalisation commands. APP. 3 shows the memory usage for the RNG commands.

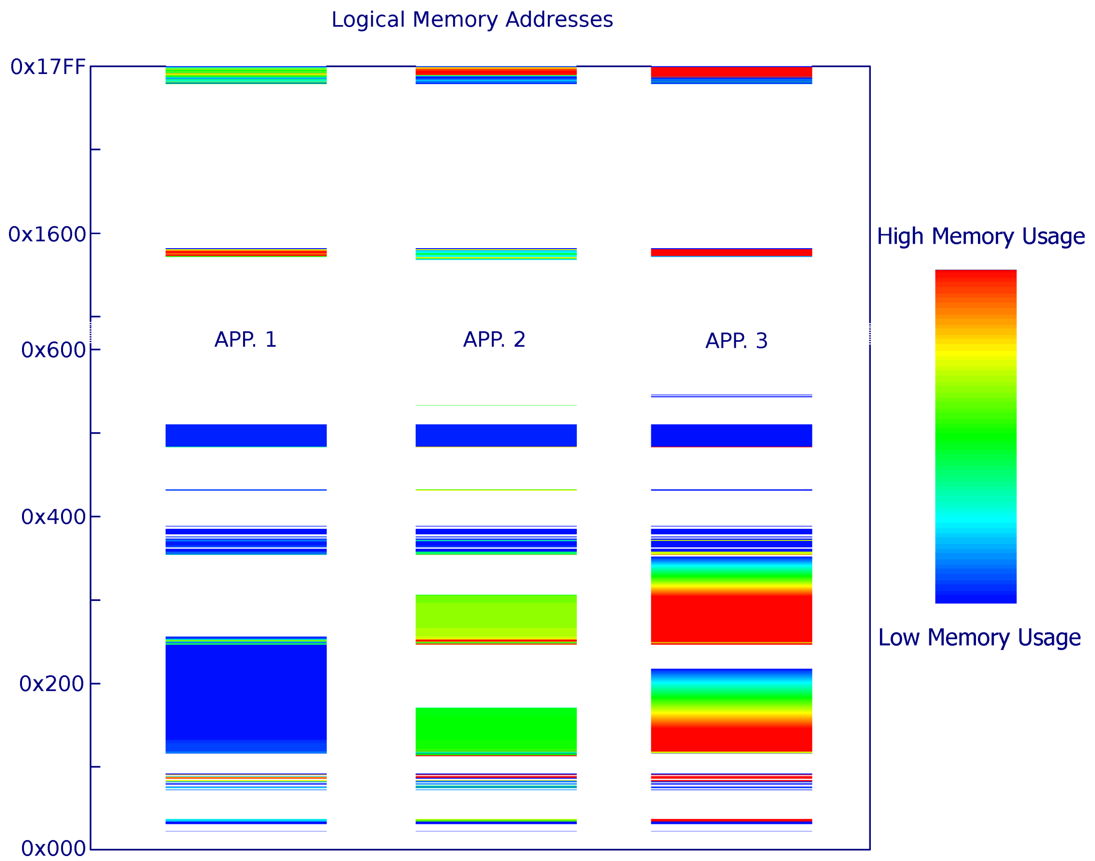


Figure 4-6 Accessed logical addresses for each application

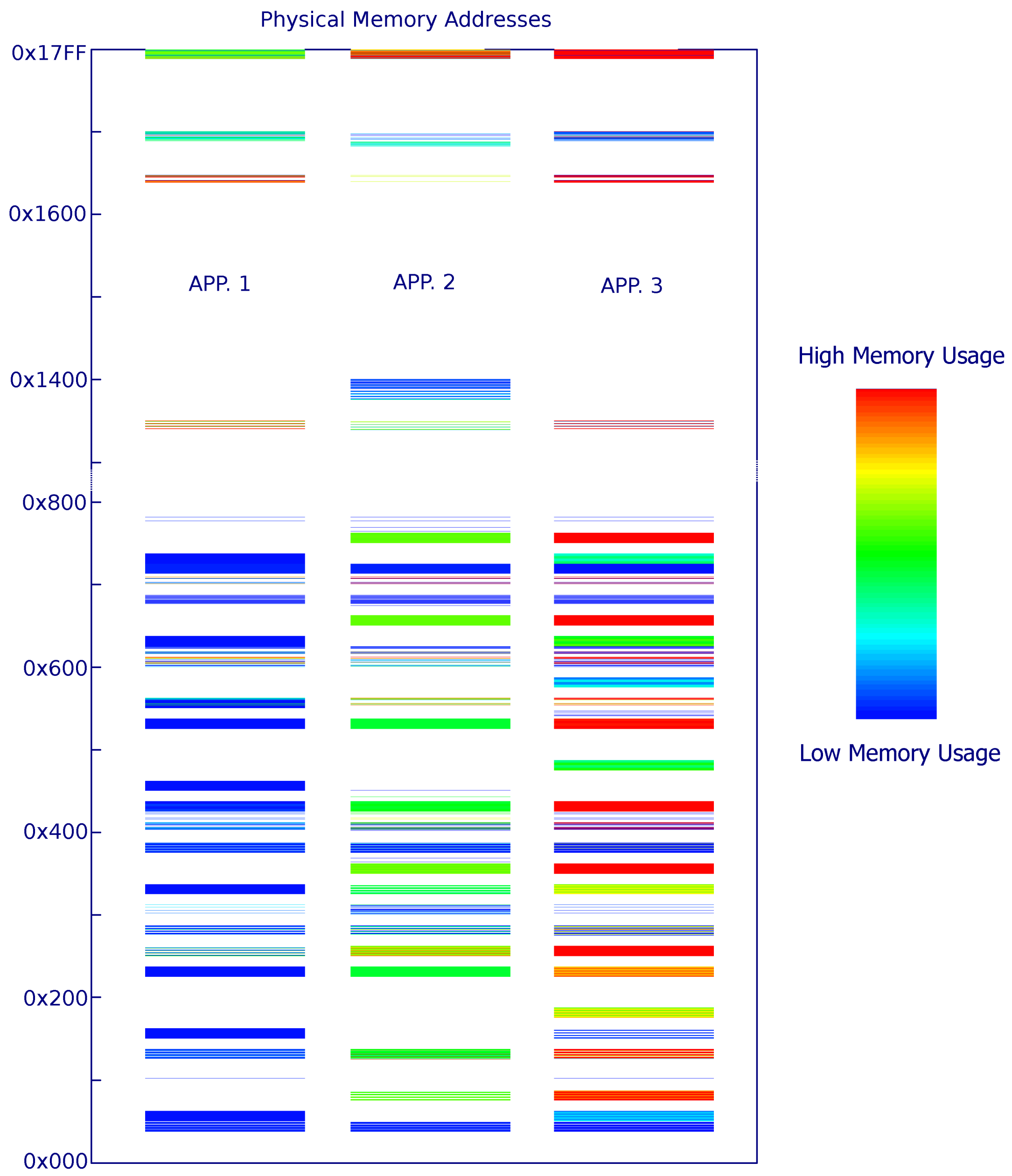


Figure 4-7 Accessed physical addresses for each application

5 Discussion

As expected, different applications have different memory requirements in terms of the amount of memory needed and the spatial and temporal domain accesses. The scrambler also plays an important role in the memory's spatial usage. In fact, the data is more scattered in the physical memory than in the logical one, which is relatively compact, Figure 4-6 vs. Figure 4-7.

Different customers will use different OSs and applications, which will result in a considerably broader memory usage pattern. These usage patterns are made worse by the scrambler. This means that it is not possible to have a partitioning approach that is optimal for all cases or customers. Certainly not with the data available in the current research. Nevertheless, some general conclusions can be drawn that could help to reduce the power consumption for all or most of the cases or customers.

Out of the total available system memory in the SCD-A, the Standard SRAM in Figure 4-1, the SCOS uses or allocates slightly less than half of it, see Figure 4-2. All memory areas defined in Figure 4-2 could be combined easily into one memory unit, allowing the other memory unit to power off in order to save on dynamic and static power. Static for obvious reasons. Dynamic as currently, by design, both memory units are accessed concurrently for read operations, producing a higher power consumption that what is actually needed.

As long as all applications are limited to define their variables within the C-Stack area, the second memory unit could remain powered off. This shows that small hardware changes and software decisions could help in reducing the power consumption. According to [12], the leakage current of the gated memory could be cut by up to 86% – from 2.1 µA to 0.29 µA – with no impact on the area or performance. Also, it would no longer contribute toward the dynamic power consumption.

Despite the SCOS defining an overall memory size of nearly 3-Kbyte, none of the applications accessed more than 8.08% and 9.89% of the 6-Kbyte memory for read and write operations respectively. Furthermore, the memory is only accessed for between 15% and 21% of the run time, remaining IDLE for 79% to 85% of the time. This clearly indicates that there is room for further power savings should the 3-Kbyte memories be partitioned and a powering policy applied depending on the partition access and content.

The simulations show three specific memory usage patterns. The first is that all applications perform more SRAM write operations than read operations. This can be explained by the initialisation of variables, a common behaviour when entering functions, and the use of software loops and/or counter variables, where it is likely that a given variable will be updated several times before that same variable is read.

The second pattern is that single addresses are accessed approximately 10 times more often than three consecutive addresses. No information was generated regarding which addresses were accessed individually or consecutively. However, accessing three logical addresses in a row could be explained by the use of the memory stack to store or recover the program counter (which in the CPU of this Smart Card is three bytes long) when entering and leaving software routines. In fact, the R-Stack usage suggests this behaviour. Figure 5-1 shows the number of accesses to several consecutive logical addresses in the R-Stack section, the actual addresses are not shown for confidentiality reasons.

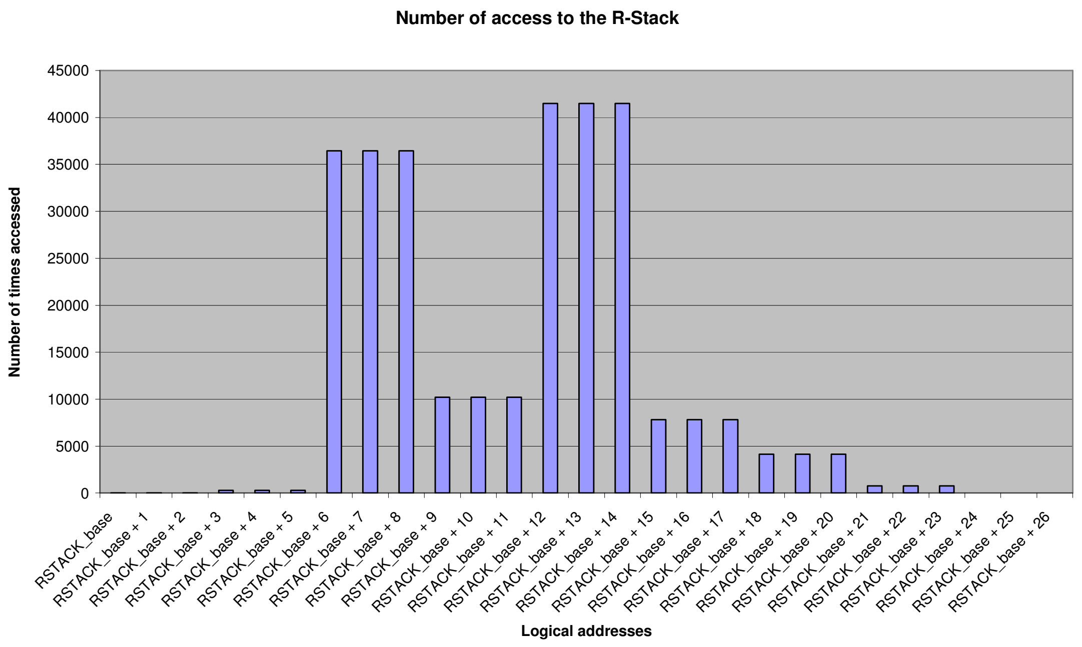


Figure 5-1 Consecutive accesses to the R-Stack

Two consecutive address accesses, on the other hand, show a very particular behaviour, as this addressing mode is used only once in two of the simulations and twice in the third simulation. Other than storing the program counter value, the stack is also typically used to store the value of general purpose registers before or after calling a routine, but there is no evidence of this kind of access in the R-Stack.

Something else could explain accessing two concurrent addresses, using two consecutive data indirect addressing mode instructions with pre-increment or post-increment of a pointer. This is, consecutive read-write, write-read, read-read or write-write memory accesses.

Despite the intended logical block size, the actual physical block size is 1 for all cases. This means that, regardless of how the logical memory is accessed, no consecutive physical memory addresses were accessed. To make matters worse, the address scrambling can change each time the Smart Card is powered up.

A solution to the scrambler issue could be to apply individual scramblers to each partition, just as though each partition was a small individual SRAM with its own built-in scrambler. This way all the addresses in a logical partition would be scrambled into the addresses for an equivalent physical partition. Furthermore, if all partitions are the same size, then an additional scrambling tier could be added so that the different partitions could also be scrambled. The benefit of a two tier scrambling approach is that any partitions that are powered off could be re-scrambled the next time they are powered on.

The third pattern that can be highlighted in the table is the memory access time and its distribution. Three different access times are used in all three simulations. These access times represent the period of the system clock cycle when the memory was being accessed and not the amount of time that it took for the data to be written to the SRAM array, nor the amount of time that it took for the data to be present at the SRAM's output data bus. These variations in the access time are the result of the system clock speed changes. It must be noted that the system clock may be partially controlled by the OS depending on the performance required by the operation or the command to be carried out.

The possibility of powering different partitions individually immediately suggests the need for an embedded memory power management unit and/or powering mechanisms within the memory. A power management unit also implies the need for some sort of intelligence, so that segments can be enabled in time for the next access. Furthermore, there is a time delay associated with the supply voltage changes from DRV to fully on and vice versa. These changes are not immediate. This penalty will worsen when powering partitions on and off, as in this case, the delay is greater.

When looking at how each of the above usage patterns impact the powering strategy initially proposed in this research, the fastest memory access times should be considered as the worst case scenario as they force the segments' powering mechanism to rise and lower the supply voltage level at faster speeds. This requirement can be particularly demanding when several individual accesses are separated by only one idle (no access) cycle each.

The stress imposed on the powering mechanism can be relaxed if a switching policy is implemented in the memory power management unit. This policy could enable partitions to be powered on and off by software, even if it is through a lite-MMU, as this is the easiest way to determine the validity of the data held within a particular partition. This policy could also use hardware to control each partition’s power supply value according to the partitions’ access rate. Partitions could be fully powered as soon as an access to them is detected and then powered to a DRV level a given amount of idle cycles after the partition was accessed for the last time. This technique will not avoid the cases where the powering mechanisms are asked to lower and raise the voltage in consecutive clock cycles, but it could minimise this event.

The memory access delay generated from changing the partitions’ supply voltage level could, in some cases, be dismissed due to the Smart Card application requirements, but this is not warrantied to be always the case. Hence, this powering policy should be used with care.

Other than that, the data generated in this research focuses purely on memory access parameters, which is only part of a bigger picture and not enough on its own to suggest the best memory partitioning approach. Valuable information is lost by ignoring the meaning of each access and/or their association to different parts of the code. For example, differentiating static and volatile data and their validity period could help determine which partitions can be powered off.

Software can define variables at different points in the code. These variables can be classified as global or static (static) or, temporal or volatile (volatile). From a memory usage point of view, static variables should be present in the memory for as long as that particular software is running. Volatile variables, on the other hand, are only valid within the code-block where they were defined. These variables are discarded once the code-block that defined them has been executed. For example, the variables defined within a function will disappear from the memory when this function has finished its task.

When analysing variables’ behaviour at a system level, the software should be divided into two main layers, the OS and the application. The OS is the piece of code that will be running continuously. For the case of the OS, every static variable is valid and should be kept for the period that the Smart Card is powered on. Volatile variables, on the other hand, will be created and destroyed as they are only needed by the OS in order to perform certain operations.

When it comes to the applications, these are pieces of code executed in an ad-hoc mode, that is, they are launched depending on the job to be performed. Applications can have variables that need to be present for as long as it is running and variables that can be generated or destroyed on the fly. However, from the system's perspective, when the application is finished most or all of the variables related to the application are no longer valid. From this perspective, most or all application variables are volatile variables.

Taking this perspective into account, one or more partitions could be assigned to store all application variables, power these partitions on when a given application is selected (i.e. running) and power them off when no application is selected (i.e. only the OS is running).

Another option to fill the knowledge gap and propose a partitioning approach is to analyse the SRAM usage at system level, i.e. understand how different software layers use the SRAM. The results shown in this work represent the memory usage made by the communication channel, the application and the OS. From this data, it is not possible to determine when the SRAM is accessed in order to perform OS specific operations or when it is accessed to execute the requested command. This also applies to the accessed memory addresses and the memory requirements for the OS and the applications. A finer analysis could be obtained if memory accesses are classified as follows:

- on the Smart Card power up, any memory usage is related to the OS;
- memory usage while commands are being transmitted could be related to communication;
- memory usage between a command sent by the reader and a response sent by the Smart Card can be related to the application; and
- any other memory access is related, again, to the OS

This memory usage classification can result in a memory partitioning approach easily applicable to other applications and OSs. One result of this analysis could be defining an ISO I/O buffer sized partition. This partition could be powered off while the Smart Card is awaiting a command. Upon the ISO I/O IP detecting an incoming APDU command, it could automatically power up the partition associated with the communication buffer. Once the command has been executed and the result or reply forwarded, the OS could power off the communication buffer.

This partitioning approach might not result in the highest power savings. However, it has the potential to minimise the impact on the compiler. This is of special interest to Atmel, as Atmel's customers use C compilers designed by third party companies, such as IAR.

Finally, it must be assumed that not all customers would be interested in a partitioned SRAM, also, porting their OS might be too costly for them to exploit this feature. Hence, the partitioned SRAM should be as transparent as possible in the CPU's, the system's, the operating system's and the programmers' eyes. If backwards compatibility between partitioned and non-partitioned architectures is not provided, it might result in difficulties with marketing the Smart Card.

6 Conclusions

It has been shown that memory usage is highly application specific. The effectiveness of memory partitioning as a leakage reduction technique depends on the thorough study of the memory usage by these and other applications, as well as understanding how variables are used by the OS and applications. Techniques to improve the data gathering have been presented to help with this matter.

The pros and cons of this technique have been argued, and possible solutions to the different issues here discussed have also been presented. Partitioning a Smart Card memory will have an impact on the whole system and process, starting with the SRAM module itself, to the Smart Card system and even the programming approach. The impact on the compiler must be minimised. The success of this approach is tied to the success at each of these levels.

With time, OSs developed for Smart Cards and small ICs should take this or any other power reduction techniques into account.

7 Future work

The next step for this research line is to decide its applicability or determine if there is any interest in this topic from the memory group in France, who are responsible for taking this topic forward. Additional applications could be simulated for them if necessary, in which case the test setup should be updated to differentiate between OS and application SRAM accesses. If this topic is to be followed up, specific design alternatives or improvements to the scrambler should be investigated and tested.

Reference

1. M. R. Casu and P. Flatresse, “Converting an Embedded Low-Power SRAM from Bulk to PD-SOI”, in *Proceedings of the International Workshop on MTDT*. 2002
2. X. Wu, F. Wang, Y. Xie, “Analysis of Subthreshold Finfet Circuits for Ultra-Low Power Design” in *IEEE Int. SOC Conference*, September 2006, pp. 91-92
3. S. Datta, G. Dewey, M. Doczy, B.S. Doyle, B. Jin, J. Kavalieros, R. Kotlyar, “High Mobility Si/SiGe Strained Channel MOS Transistors with HfO/sub 2/TiN Gate Stack” in *IEEE Int. Electron Devices Meeting*, Dec. 2003, pp. 28.1.1-28.1.3
4. M. Ashouei, A. Chatterjee, A. D. Singh and V. De, “A Dual-Vt Layout Approach for Statistical Leakage Variability Minimization in Nanometer CMOS”, in *Proceedings of the ICCD*. 2005
5. T. D. Burd, T. A. Pering, A. J. Stratakos and R. W. Brodersen, “A Dynamic Voltage Scaled Microprocessor System”, in *IEEE Journal of Solid-State Circuits*, November 2000, Vol. 35, n 11, p 1571-1580
6. K. Roy, S. Mukhopadhyay and H. Mahmoodi-Meimand, “Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits”, in *Proceedings of the IEEE,* February 2003, Vol. 91, n. 2, p 305-327
7. L. Benini, A. Macii and M. Poncino, “Energy-Aware Design of Embedded Memories: A Survey of Technologies, Architectures, and Optimization Techniques”, *ACM Transactions on ECS*, p5-32, February 2003, Vol. 2, No. 1
8. O. Oztruk and M. Kandemir, “Nonuniform Banking for Reducing Memory Energy Consumption” , in *Proceedings of the conference on DATE*. 2005
9. O. Golubeva, M. Loghi, M. Poncino and E. Macii, “Architectural Leakage-Aware Management of Partitioned Scratchpad Memories”, in *Proceedings of the conference on DATE*. 2007
10. H.Qin, Y. Cao, D. Markovic, A. Vladimirescu and J. Rabaey, “SRAM Leakage Suppression by Minimizing Standby Supply Voltage”, *5th ISQED*. 2004, IEEE Computer Society.
11. K.Flautner, N.S. Kim, S. Martin, D. Blaauw and T. Mudge, “Drowsy Caches: Simple Techniques for Reducing Leakage Power”, in *Proceedings of ISAC*, 2002
12. M. Powell, S.H. Yang, B. Falsafi, K. Roy and T.N. Vijaykumar, “Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories” in *Proceedings of ISLPED*, 2000.

Appendix A Test-bench Update

Atmel's original Smart Card test-bench focused on carrying out some behavioural tests on the Smart Card's RTL model. This is achieved by loading a test program or firmware into the Smart Card and running a series of pre-defined tests. This firmware would respond to commands received via the I/O port, execute a given tests such as a DES encryption or decryption, and reply with whether the test passed or failed. A high level diagram of this setup is shown in Figure 7-1

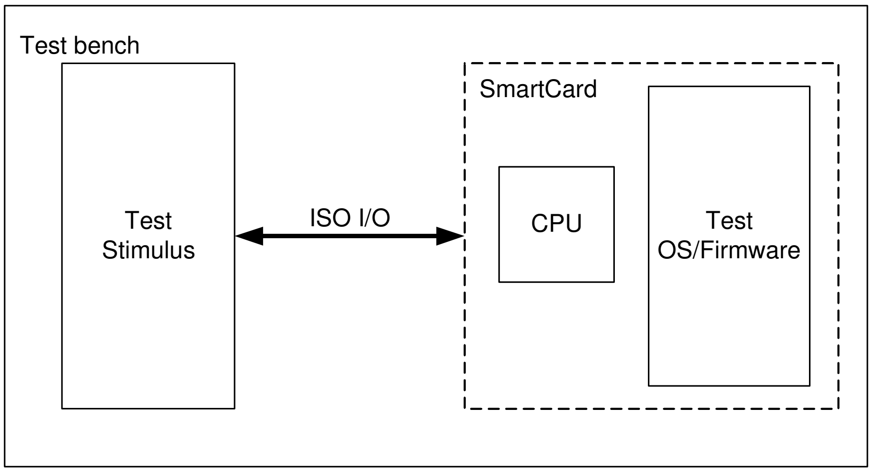


Figure 7-1 Atmel's typical HDL Smart Card test environment

This case study needed to exercise the Smart Card like a real application. Figure 7-2 shows a typical example of a real application. Here, the Smart Card communicates with the Smart Card Reader through the I/O port, while the Smart Card Reader works as an interface between the application run on the computer and the Smart Card.

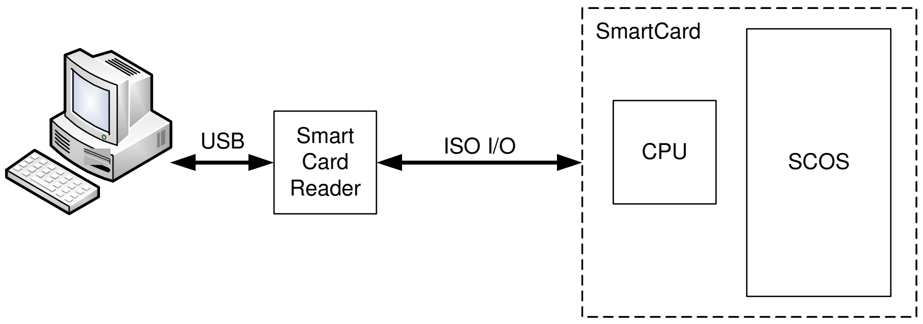


Figure 7-2 Smart Card connection example

For this case study, the Smart Card Reader function of Atmel's original Smart Card test-bench was not functional, this was developed as part of this research. Furthermore, since

SCOS uses the SPI port to communicate instead of the I/O port, this also needed to changed. Follow the next steps to patch Atmel's Smart Card test-bench.

Create a directory where the analysis environment will be placed:

> *Assumption: In this example assume that the directory name is '/projects/test'. This, of course, can be changed to something else.*

```
$ mkdir /projects/test
```

Move into this directory and checkout the '*Longbow*' test environment. This will result in creating a directory named '*longbow*'; this directory needs to be renamed to '*sc*':

```
$ cd /projects/test
$ cvs checkout longbow
$ mv longbow sc
```

Get the latest usage environment patch file from '*/projects/ram_leakage/release/usage_setup_v1_0.patch*' and apply it.

```
$ cp /projects/ram_leakage/release/usage_setup_v1_0.patch.tgz .
$ tar -xzf usage_setup_v1_0.patch.tgz
$ patch -p1 < usage_setup_v1_0.patch
$ rm usage_setup_v1_0.patch*
```

Run the '*run_me_first*' script so it finishes setup of the environment. This script will give execution permission to those scripts that need it and it will update the local `~/.cshrc` and `~/.aliases` files with `IPUSE_LOCAL` and '`ipusim`' definitions respectively. This script also creates the simulation output directory. Since simulation results might take a lot of space, this script can generate a symbolic link from the '*sc/sim/rtl/out*' directory to any desired directory. This is done by passing the full path to the target directory as a parameter when running the script '*run_me_first*'. If no parameter is passed an '*sc/sim/rtl/out*' directory will be created in the user's environment tree. When defining a target directory, this directory must exist before calling the script. Finally, this script will erase itself after being run.

> *Assumption: In this example the simulation outputs will be stored in '/net_temp13/test/out'.*

```
$ mkdir /net_temp13/test
$ mkdir /net_temp13/test/out
$ chmod +x run_me_first
$ run_me_first /net_temp13/test/out
```

Open a new console for the environment variables to take effect. Test if the setup is working by running the following command, which can be found in the file '*sc/run_ipusim*':

Note: this simulation takes about 12 minutes to run.

```
$ ipusim ROM144144_2_base_release_at98_asier.s19 -mode user -
sawed -xaf1 0x0600 -fill 0xff -tsp 0x374d45575451534b -romkey
0x3bba -kwf 0x3f3456
```

Now the environment should be updated. Figure 7-3 should a diagram of the resulting test-bench.

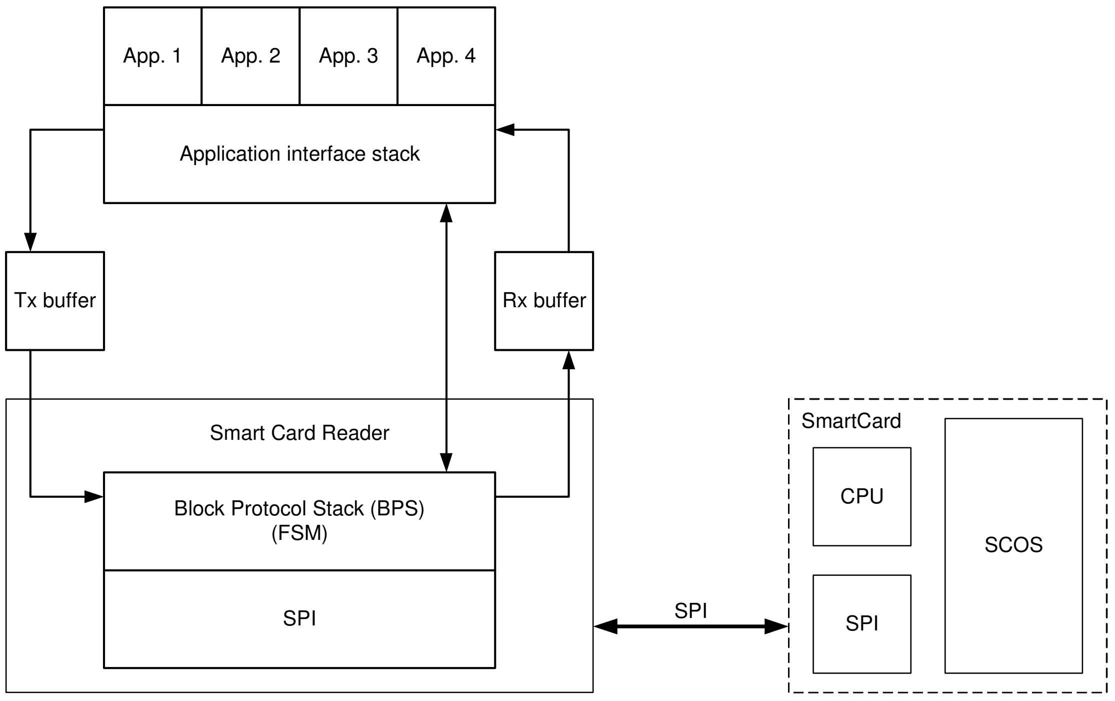


Figure 7-3 Atmel's HDL Smart Card test environment updated for this research

Appendix B Analysis Script

The data generated by the simulations was analysed using the Perl script analysis.pl. This script has 1955 lines of code. Here is a description of the script:

Which version is covered here?

- 0.5.2

What does it do?

- Generate an analysis on IP usage
- Currently it focuses on SRAM usage

How does it do it?

- Opens VCD file generated in a from a previous simulation
- Gets the signals' indexes
- For each time-stamp, gets the value for the signals of interest
    - Generate partial usage information of when the target IP is being accessed
- When the whole simulation has been processed, generate a statistical analysis based on the generated information
- Stores the statistical information and the action carried at each clock cycle

How is data managed?

- Each targeted signal is stored in a hash variable where the following aspects can be defined:
    - Function
    - I/O name
    - VCD index value
    - Width
    - Current value
- Each VCD index value is a hash to the following parameters:
    - Index name
    - Index position
    - Index value
- Targeted signals can be referenced by:
    - Function name
    - Signal name
    - Index
- Other data is stored as follows:
    - Read and write access
    - Size of contiguous accesses

- Selected and unselected cycles
- Cycle-wise action

Appendix C Usage Analysis Results

Memory usage for random number generation. Due to confidentiality reasons, the accessed logical and physical addresses are not shown in the analysis results below.

```
Statistical analysis of the SCD-A's ram module's usage.

Statistical analysis version: 0.5.2

Atmel Confidential DO NOT copy. For internal use only

Date: Mon Oct  1 16:37:53 2007


ram usage is as follows:
==================================


Simulation run information
==========================

Simulation time = 3205569757000ps
Simulated clock cycles = 19774170
Average clock frequency = 6.16869121528838 MHz
Maximum clock frequency = 24.5098039215686 MHz
Minimum clock frequency = 6.0909055471095 KHz


Access in percentages (clock cycles)
====================================

 access mode |    (%)     | number of cycles |
-------------+----------+------------------+
  por        |   0.00   |            134  |
  clear      |   0.00   |            513  |
  read       |   4.52   |         894201  |
  write      |   3.56   |         703342  |
 total usage |   8.08   |        1598190  |


Access times are divided as follows:
====================================

Read:
-----
  time(ps)  |     (%)    | ocurrences |
------------+-----------+------------+
     20400  |    1.16   |     10353  |
    142800  |   98.84   |    883838  |
    202560  |    0.00   |         9  |

Write:
-----
  time(ps)  |     (%)    | ocurrences |
------------+-----------+------------+
     20400  |    1.30   |      9153  |
    142800  |   98.70   |    694166  |
    202560  |    0.00   |        23  |


Accesses addresses are as follows:
==================================

Read: 8.08% of the addresses where accessed in this mode

---------------------------+----------+
          Address          |  times   |
  Logical    |  Physical   | accessed |
-------------+-------------+----------+
             |             |          |
             |             |          |
                                      |
 Data removed due to confidentiality  |
               reasons                |
             |             |          |
             |             |          |

Write: 9.89% of the addresses where accessed in this mode

---------------------------+----------+
          Address          |  times   |
  Logical    |  Physical   | accessed |
-------------+-------------+----------+
             |             |          |
             |             |          |
                                      |
 Data removed due to confidentiality  |
               reasons                |
             |             |          |
             |             |          |


Sequential blocks accesses are as follows:
==========================================

   Logical   |               |
  block size |  occurrences  |
-------------+---------------+
          1  |       990299  |
          2  |            1  |
          3  |       202414  |

   Physical  |               |
  block size |  occurrences  |
-------------+---------------+
          1  |      1597543  |

Active distribution is as follows:
==================================

Note: If the minimum required amount of unselected clock cycles to consider the
      IP block IDLE is 1, this section will provide the same information as the
      Logical block sizes shown at "Sequential blocks accesses are as follows:".

 Active cycles |  occurrences  |
---------------+---------------+
            1  |       990299  |
            2  |            1  |
            3  |       202414  |


IDLE distribution is as follows:
================================

 IDLE cycles |  occurrences  |
-------------+---------------+
          1  |       497609  |
          2  |       171856  |
          3  |       101852  |
          4  |        74883  |
          5  |        39721  |
          6  |        47316  |
          7  |        30642  |
          8  |        45586  |
          9  |        24791  |
         10  |        11587  |
         11  |        55713  |
         12  |        12659  |
         13  |          834  |
         14  |         1029  |
         15  |         1865  |
         16  |          209  |
         17  |          127  |
         18  |          104  |
         19  |         110  |
         20  |         191  |
         21  |         218  |
         22  |        1065  |
         23  |         844  |
         24  |         380  |
         25  |         612  |
         26  |          21  |
         27  |           6  |
         28  |           1  |
         29  |           3  |
         30  |           4  |
         31  |           3  |
         32  |           2  |
         33  |           4  |
         34  |           4  |
         35  |           4  |
         36  |           1  |
         37  |           6  |
         38  |           2  |
         39  |           1  |
         40  |           3  |
         41  |           1  |
         42  |          86  |
         43  |       19782  |
         44  |       12671  |
         45  |         286  |
         46  |           3  |
         47  |         376  |
         48  |         157  |
         49  |           6  |
         50  |           3  |
         51  |           4  |
         52  |           2  |
         53  |           1  |
         54  |           1  |
         55  |           1  |
         56  |           7  |
         57  |           1  |
         58  |           3  |
         59  |           1  |
         60  |           2  |
         61  |           1  |
         63  |           8  |
         64  |           2  |
         65  |           2  |
         66  |           1  |
         67  |           3  |
         68  |           1  |
         69  |           4  |
         70  |           4  |
         71  |           1  |
         72  |           2  |
         74  |           2  |
         75  |           1  |
         76  |           1  |
         77  |           6  |
         79  |           3  |
         80  |           2  |
         81  |           2  |
         82  |           4  |
         83  |           4  |
         84  |          10  |
         86  |           1  |
         87  |           2  |
         88  |           3  |
         89  |           1  |
         90  |           4  |
         91  |           3  |
         94  |           2  |
         95  |           3  |
         96  |           2  |
         97  |           2  |
         98  |           6  |
        100  |           4  |
        101  |           2  |
        102  |           1  |
        103  |           1  |
        104  |           3  |
        105  |           3  |
        106  |           1  |
        107  |           4  |
        108  |           2  |
        109  |           4  |
        110  |           2  |
        111  |           1  |
        112  |           5  |
        113  |           4  |
        114  |           2  |
        115  |           1  |
        116  |           4  |
        117  |           1  |
        118  |           3  |
        119  |          13  |
        120  |           1  |
        122  |           2  |
        123  |           4  |
        124  |           3  |
        126  |           6  |
        127  |           1  |
        128  |           3  |
        129  |           3  |
        130  |           4  |
        131  |           1  |
        132  |           2  |
        133  |           5  |
        134  |           4  |
        135  |           3  |
        137  |           2  |
        138  |           3  |
        140  |           3  |
        141  |           2  |
        142  |           2  |
        143  |           1  |
        144  |           4  |
        145  |           2  |
        147  |           6  |
        148  |           1  |
        149  |           2  |
        150  |           4  |
        151  |           3  |
        152  |           2  |
        153  |           2  |
        154  |           7  |
        157  |           5  |
        158  |           1  |
        159  |           4  |
        160  |           5  |
        161  |           3  |
        162  |           1  |
        164  |           1  |
        165  |           3  |
        166  |           1  |
        167  |           2  |
        168  |          10  |
        169  |           3  |
        170  |           1  |
        171  |           2  |
        173  |           3  |
        174  |           2  |
        175  |           6  |
        176  |           2  |
        177  |           2  |
        178  |           1  |
        179  |           1  |
        180  |           5  |
        182  |           3  |
        183  |           2  |
        184  |           4  |
        186  |           1  |
        187  |           2  |
        188  |           2  |
        189  |           6  |
        190  |           2  |
        191  |           4  |
        192  |           1  |
        193  |           2  |
        195  |           1  |
        196  |          10  |
        197  |           5  |
        198  |           5  |
        199  |           1  |
        200  |           2  |
        201  |           2  |
        202  |           1  |
        203  |          11  |
        204  |           2  |
        205  |           4  |
        206  |           3  |
        207  |           3  |
        208  |           1  |
        209  |           3  |
        210  |           7  |
        211  |           8  |
        212  |           2  |
        213  |           2  |
        214  |           4  |
        215  |           1  |
        216  |           1  |
        217  |           9  |
        218  |           3  |
        219  |           3  |
        220  |           1  |
        221  |           2  |
        222  |           4  |
        223  |           2  |
        224  |          11  |
        225  |           2  |
        227  |           2  |
        228  |           1  |
        229  |           1  |
        230  |           4  |
        231  |           8  |
        232  |           3  |
        235  |           1  |
        236  |           2  |
        237  |           1  |
        238  |           4  |
        239  |           3  |
        240  |           3  |
        241  |           3  |
        242  |           3  |
        244  |           3  |
        245  |           3  |
        246  |           4  |
        247  |           1  |
        248  |           2  |
        249  |           3  |
        250  |           3  |
        251  |           1  |
        252  |           8  |
        253  |           2  |
        254  |           6  |
        255  |           4  |
        256  |           3  |
        258  |           4  |
        259  |           5  |
        260  |           1  |
        261  |           2  |
        262  |           2  |
        263  |           2  |
        264  |           6  |
        265  |         112  |
        266  |           8  |
        267  |           2  |
        269  |           2  |
        270  |           3  |
        272  |         146  |
        273  |            6  |
        274  |            3  |
        276  |            4  |
        277  |            1  |
        279  |            3  |
        280  |            5  |
        281  |            3  |
        282  |            3  |
        287  |            5  |
        290  |           91  |
        294  |            6  |
        297  |          167  |
        301  |            4  |
        308  |            5  |
        315  |            2  |
        321  |            1  |
        322  |            8  |
        329  |            5  |
        336  |            3  |
        342  |           13  |
        343  |            1  |
        346  |        18612  |
        349  |          238  |
        350  |            5  |
        353  |        15273  |
        356  |         1308  |
        357  |            6  |
        363  |          528  |
        364  |            5  |
        371  |            5  |
        378  |            2  |
        385  |            4  |
        391  |           77  |
        392  |            5  |
        396  |          174  |
        399  |            5  |
        401  |            7  |
        406  |            7  |
        413  |            5  |
        420  |            7  |
        427  |            3  |
        517  |            1  |
       1074  |            1  |
       1075  |            2  |
       1076  |            1  |
       1077  |            1  |
       1078  |            1  |
       1080  |            1  |
       1083  |            1  |


IDLE statistical data:
======================

Number of IDLE states: 1192714
Mean IDLE cycles:      15.2391772042585
Standard deviation:    60.2692807586774
```

Detecting Voltage Glitch Attacks on Secure Devices

Asier GOIKOETXEA YANCI
Institute for System Level Integration
asier.goikoetxea@sli-institute.ac.uk

Stephen PICKLES
Atmel
stephen.pickles@ekb.atmel.com

Tughrul ARSLAN
University of Edinburgh
t.arslan@ed.ac.uk

Abstract

Secure devices are often subject to attacks and behavioural analysis in order to inject faults on them and/or extract otherwise secret information. Glitch attacks, sudden changes on the power supply rails, are a common technique used to inject faults on electronic devices. Detectors are designed to catch these attacks. As the detectors become more efficient, new glitches that are harder to detect arise. Common glitch detection approaches, such as directly monitoring the power rails, can potentially find it hard to detect fast glitches, as these become harder to differentiate from noise. This paper proposes a design which, instead of monitoring the power rails, monitors the effect of a glitch on a sensitive circuit, hence reducing the risk of detecting noise as glitches.

1. Introduction

Electronic devices are designed to operate according to their specification, which defines the operation limits for temperature, supply voltage, clock frequency, etc. Forcing the device to operate out of these specifications can result in injecting a fault, either temporal or permanent, the device's malfunction, or even permanently damaging it. A device can be intentionally subject to out of specification operation in order to increase its performance, for example over clocking a central processing unit (CPU), or injecting a fault. An example would be applying voltage glitches on the power rails of a Smart Card [1].

In an aim to protect the devices against malfunction and damage, detectors can be used to detect out of specification operation conditions. However, detectors have a detection range, where conditions outside their limits can go undetected. In the case of secure devices this detection limitation might be exploited by attackers to subject the device to even more extreme conditions in an aim to circumvent the detectors.

2. Background

Digital devices are designed to operate with a stable power source. In this circumstance, transistors, the main components of digital circuits or devices, tend to operate in the cut-off and saturation regions (OFF/ON) to indicate logic values of 1 and 0. Modifying the supply voltage can have two effects. On one side, transistors can enter the linear region. On the other side, parasitic resistors and capacitors are present in the power rails across the design. In the example in Figure 1, sudden changes in the supply voltage might not be equally spread across the whole design. As a result, different sub-circuits might be powered at different voltages, hence, inducing false readings of logic one or zero. This is the process of injecting a fault.

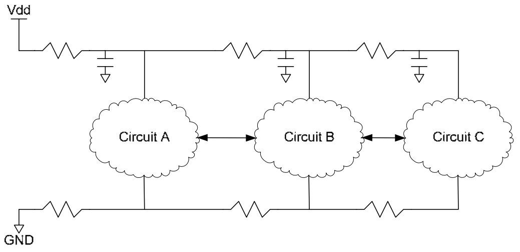


Figure 1. Parasitic resistors in power rails

Attackers often use glitches on the power supply in order to inject faults on secure devices. There are four main kinds of glitches that can be applied to the power rails of a device: a positive glitch in Vcc, Figure 2a; a negative glitch in Vcc, Figure 2b; a negative glitch in ground, Figure 2c; and a positive glitch in ground, Figure 2d.

Glitch detectors are used to detect abnormal fluctuations and values of both rails of the power supply, Vcc and ground. In devices with a built-in voltage regulator, the regulator works as filter, where the voltage level in Vdd is a filtered version of that in Vcc. Some glitches in Vcc might be filtered out by the voltage regulator and not appear on Vdd; however, some others might have an impact on Vdd. Hence, glitch detectors are used at both sides of the voltage regulator, as shown in Figure 3.

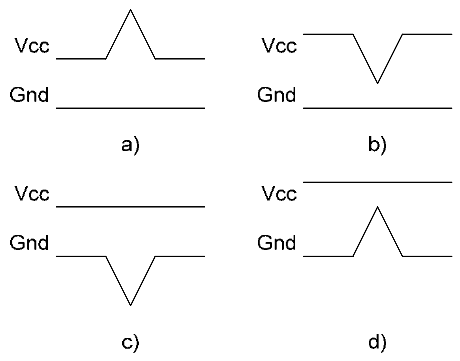


Figure 2. Common types of glitches

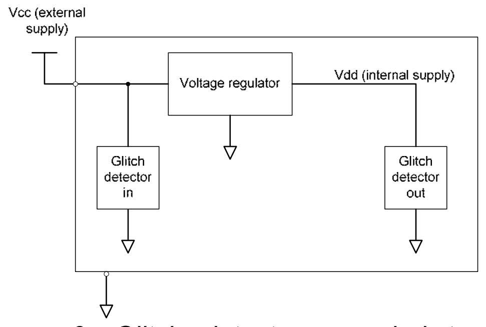


Figure 3. Glitch detectors coupled to a voltage regulator

What kind of glitches are capable of impacting the Vdd depends on two main aspects: firstly on the voltage regulator's bandwidth; and secondly on the voltage regulator's load. The first aspect is specific to the voltage regulator's design. The second one depends on the device's operation at the time of applying the glitch. The study/design of voltage regulators is out of the scope of this paper, but the load's effect on the internal Vdd will be demonstrated later on this paper.

The common approach used by glitch detectors to detect abnormalities in the supply voltage, is to directly monitor the supply rails. Furthermore, they aim at detecting the glitch while it is happening. Often, this poses a challenge when designing detectors capable of detecting fast glitches; as legitimate noise could be falsely identified as a glitch.

Due to the lack of publicly available information on glitch detectors and their dependency on the voltage regulators and load, the work carried in this paper focuses on two positive glitches identified as difficult to be detected by the glitch detectors used with a voltage regulator here named as VR1. These glitches, applied to the VR1's Vcc, are defined here as G1, Figure 4a, and G2, Figure 4b. G1 is a positive glitch on Vcc which rises from 2.7v to 7v and falls back to 2.7v within 100ns. G2 is a positive glitch on Vcc which rises from 3v to 15v and falls back to 3v within 10ns.

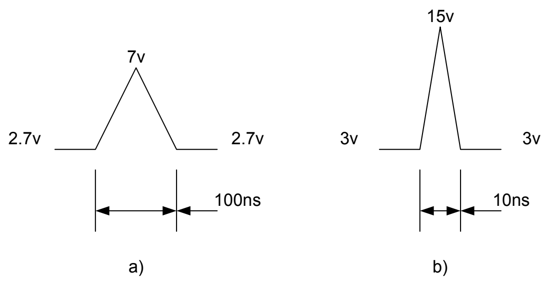


Figure 4. Target glitches: a) G1, 2.7v-7v-2.7v @ 100ns; and b) G2, 3v-15v-3v @ 10ns

2.1. Related work

In order to detect fast glitches that might be at the limit or out of the detection range of common detectors, a glitch detector that detects glitches by monitoring the glitches' effects on a sensitive circuit is proposed. This approach is already used by detectors [2] and [3]. The detector in [2] bases on monitoring fault injections into single-bit registers. It also protects static random access memory (SRAM) read operations against glitch attacks by monitoring the memory array and the sense amplifier. This simple design is capable of detecting glitches on both Vcc and ground. The primary drawback of this design is the need for a clock signal in order to latch the effect or presence of a glitch, which must be equal to or a multiple of the system's clock frequency. Since it cannot be known when a glitch is going to be applied, a clock frequency twice as fast as the system frequency should be used.

Detector [3] is more suited for the asynchronous nature of glitches, as it uses operational amplifiers (OpAmp) to continuously compare the response of three different RC filters to glitches on the supply power. This design is also capable of detecting glitches in Vcc and ground. Two drawbacks can be identified with this design. Since the RC circuits are basically voltage dividers, they will be a continuous current drain. This drainage can be reduced by increasing the resistors' sizes, which would impact negatively on the area. The other drawback with this design is its dependency on the correct instantiation of resistors and capacitors, as process variations can affect their real value.

Detector [2] is more suited to detect glitches on Vdd rather than Vcc, whereas [3] could potentially be used at both sides of a voltage regulator. The glitch detector covered here will be used to detect positive glitches on Vdd, although it could also be used to detect them on Vcc.

3. Circuit description

The glitch detector proposed in this paper, and shown in Figure 5, is made of three parts: a modified inverter; a comparator; and an RS latch. In normal supply conditions and when the circuit is disabled (*reset* input is logic one), Vout is a logic zero. It results on the OpAmp's output being set to a logic zero and resetting the RS latch to zero. When the circuit is enabled (*reset* input is a logic zero), the modified inverter sub-circuit provides a constant voltage at its output, which is lower than Vdd due to the voltage drop in the diode D1. As Vout is lower than Vdd, the OpAmp's output is a logic zero. As a result, the value stored in the RS latch is unchanged.

When the detector is enabled, the inverter circuit can be substituted by its equivalent circuit in Figure 6. Here the inverter's transistor P1, which is conducting, is substituted by its equivalent R_{onP} (in the order of ohms). The inverter's transistor N1, which is not conducting, is substituted by its equivalent R_{offN} (in the order of mega ohms) in parallel with C_N, the parasitic capacitance between its source and drain. In addition to the inverter sub-circuit, Figure 6 also shows the equivalent load seen by the inverter's output, the OpAmp's input, represented by the $Z_{in.a.o}$ impedance and which is in the order of mega ohms.

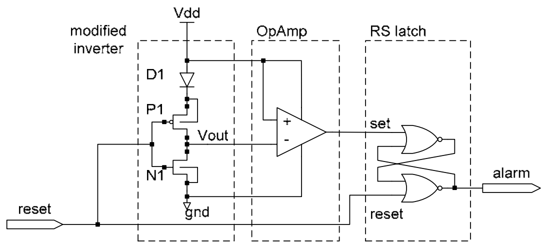


Figure 5. Proposed glitch detector

The proposed glitch detector's response to a glitch attack when the detector is enabled is shown in Figure 7 and can be described as follows. While Vdd is stable, Vout is stable too and set to Vdd minus the voltage drop in the diode D1, which is around the diode's V_{th}. If, as a result of a glitch, Vdd rises, the parasitic capacitor C_N is charged further, raising the voltage level at Vout. When Vdd falls, C_N starts to leak its charge through R_{offN} and $Z_{in.a.o}$, hence, lowering Vout. If Vdd falls at a higher rate than C_N is discharged, the diode D1 will, eventually, become inversely polarized. As a result, Vout will become higher than Vdd even after the glitch's effects on Vdd have disappeared. This phenomenon can be detected by the comparator, which sets its output to a logic one, setting the RS latch to a logic one, triggering the glitch detection alarm. Upon glitch detection, the CPU can run the appropriated routine or procedure related to glitches on the supply power. A typical response to a glitch would be resetting the whole device.

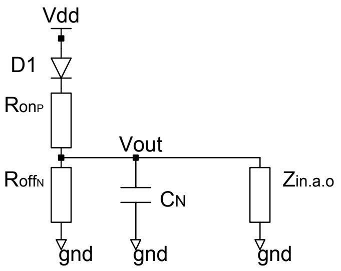


Figure 6. The inverter's equivalent circuit and load

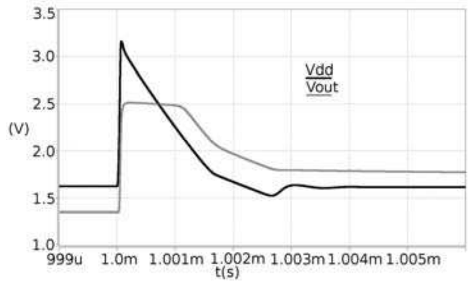


Figure 7. The modified inverter's response to a glitch

After detecting a glitch, the glitch detector would have to be reset in order to detect further glitches. This is achieved by setting the *reset* input to a logic one and back to logic zero. This process will discharge the capacitor C_N and reset the RS latch. If no action is taken after detecting a glitch, C_N will gradually discharge due to the leakage current, gradually reducing Vout to its previous level, i.e. below Vdd. However, the RS latch would continuously indicate the detection of a glitch, and no further glitch would be detected.

4. Test

Two versions of the glitch detector in Figure 5 were tested in SPICE for several supply voltage and environmental circumstances in a test environment that emulated their utilization on a secure/real product. These glitch detectors were powered via a SPICE model of the voltage regulator VR1, which also powered a fixed resistive load that emulated the load presented by the secure/real product. The setup diagram can be seen in Figure 8.

The first glitch detector, Design A, had its diode and RS latch made of low leakage transistors and the inverter and OpAmp made of high voltage transistors. The second glitch detector, Design B, was made entirely of low leakage transistors. Low leakage transistors are designed with a higher V_{th} than their normal counterparts. Increasing the V_{th} helps reducing the leakage current, an increasing issue with deep-submicron technologies; however, it also reduces the transistor performance when comparing to transistors with lower V_{th}.

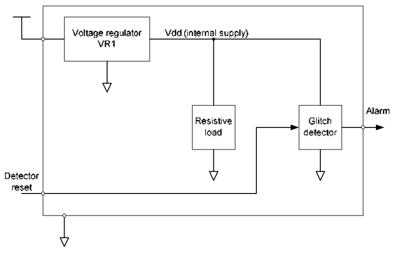


Figure 8. Test setup diagram

Four supply voltage scenarios where covered: two noisy power supplies; and two fast glitches. In all cases, these supply voltages were applied to the Vcc in Figure 8. Noisy power sources consisted of 3v and 5v sources with a ripple of +/-10% at 100KHz. These are valid supply sources for VR1, and no glitches should be detected. The applied glitches were two fast glitches that common detectors in VR1 might struggle to detect. Applied glitches consisted on the glitches G1 and G2 covered before.

For each glitch, three different resistive load scenarios were tested: high load (80 ohms); mid load (1,600 ohms); and low load (3,200 ohms). These values were chosen based on what prior experience showed on maximum, medium and minimum loads on 8-bit based secure devices. In addition to the resistive loads, all tests included a 2.2nF capacitive load was connected to the VR1. This capacitive load represented the VR1's output capacitance and the digital device's equivalent capacitive load. Finally, these tests were repeated for worst-case and best-case SPICE models, where worst-case included SPICE model conditions *rlow*, *clow* and a temperature of 125 degrees Celsius; and best-case included conditions *rhigh*, *chigh* and a temperature of -40 degrees Celsius.

Table 1 shows each glitch detector design's minimum, typical and maximum detection time for all four supply cases. Table 2 shows the detection time broken down in design version, load and SPICE model case for each glitch.

In addition, Figures 9 to 11 show the response of the design A's best-case model to the glitch G2 for high, medium and low loads respectively. Figure 12 shows the design A's inverter's output when powered with a noisy supply voltage 5v and a high resistive load using the best-case model.

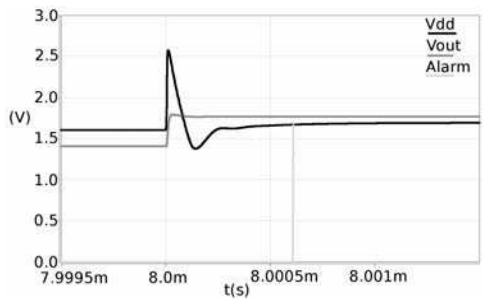


Figure 9. Glitch G2 detected by design A for high load and bcs

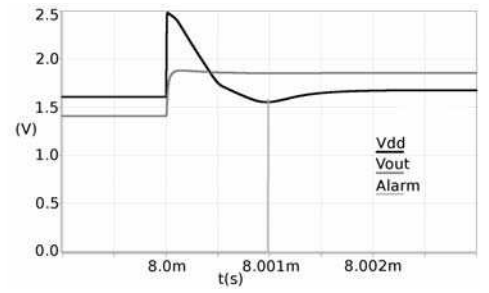


Figure 10. Glitch G2 detected by design A for medium load and bcs

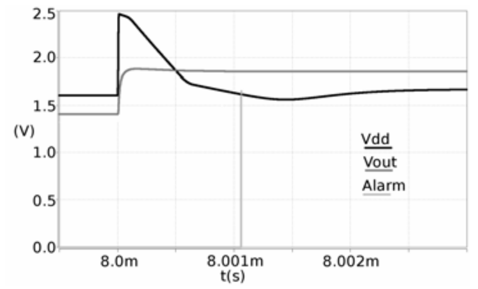


Figure 11. Glitch G2 detected by design A for low load and bcs

Table 1 Overall detection times in nanoseconds

Design	Supply	Detection time		
		min	typ	max
A	3v noise	no glitch detection		
	5v noise	no glitch detection		
	G1	571	1,144	1,508
	G2	611	1,013	1,394
B	3v noise	no glitch detection		
	5v noise	no glitch detection		
	G1	629	1,561	2,163
	G2	1,977	12,533	25,522

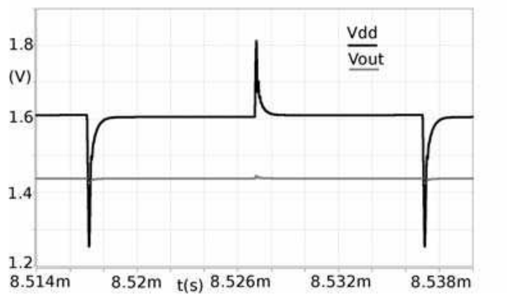


Figure 12. Vdd and Vout when powered with a noisy 5v supply with high load

Table 2 Detailed detection times

Design version	Load	SPICE model	Applied glitch	
			G1	G2
A	high	worst-case	not detected	741ns
		best-case	571ns	611ns
	mid	worst-case	1,375ns	1,281ns
		best-case	1,015ns	990ns
	low	worst-case	1,508ns	1,394ns
		best-case	1,252ns	1,062ns
B	high	worst-case	629ns	8,273ns
		best-case	not detected	not detected
	mid	worst-case	2,093ns	1,947ns
		best-case	1,359ns	25,522ns
	low	worst-case	2,163ns	2,170ns
		best-case	not detected	24,755ns

5. Discussion

As Table 1 a shows, in general terms, the glitch detector presented in this paper is capable of detecting fast glitches such as G1 and G2. The design A outperforms the design B in this task at all times, making it the best option. In a more detailed way, Table 2 indicates that design A failed to detect a glitch in one of the 12 tests. This is due to the capacitor C_N failing to charge enough so that the diode D1 would stop conducting.

Taking into account the design A's and B's transistor properties, A being faster than B was expected, since low leakage transistors are slower. What comes as a surprise is their discrepant sensitivity to process and temperature variations. Design A suffers an increase of detection time between 21.28% and 35.47% when comparing worst-case scenarios against best-case ones. Design B, on the other hand, suffers a more severe impact, needing up to 12 times more time to trigger the glitch detection signal. This impact is explained by the fact that, for the design B's best-case scenario, glitch G2 is already on the limits of its detection range, as Vout just manages to be over Vdd after the glitch. This small difference between Vout and Vdd results on a longer detection time.

Focusing on design A, despite being capable of detecting glitches that common detectors built-in VR1 might struggle to detect, on average, the detection takes in the order of 0.5 to 1.5 microseconds. This time might seem excessive in computation terms; certainly, it is not immediate. Taking a Smart Card running an 8-bit CPU at 20MHz as an example, this CPU could execute around 20 instructions per microsecond. However, from a system level point of view, Smart Cards work in the basis of data transactions. The 0.5 to 1.5 microseconds detection time are within the time required to compute these transactions. Hence, despite not being fast detectors, glitches can still be detected within a transaction.

Analyzing Figures 9 to 11, the detection time can be divided into two sections; one that goes from the beginning of the glitch until Vout becomes higher than Vdd, and another section that goes from the crossing point of Vout and Vdd until the alarm is triggered. The time width of the first section varies for the different loads, whereas the second section has similar widths for all loads, around 600ns. This proves the previous statement made regarding the load's impact on both, the voltage regulator's response to a glitch and the time required to detect such glitch. Two factors contribute to this behaviour; on one side, higher loads subject the voltage regulator to a higher stress, reducing its response time. On the other side, the higher load discharges faster the VR1's built-in capacitor. The result is a faster drop of Vdd when the glitch is in the falling edge.

The first section's time width, then, is primarily dependant on the voltage regulator and its load. The second section's time width, however, is primarily determined by the OpAmp's response time. Detection time can be reduced by instantiating a faster OpAmp. However, care must be taken if noise is not to be mistaken with a glitch, as Figure 12 shows that a noisy power supply can cause Vdd becoming lower than

Vout for a short period of time, around 152ns in this case. Nevertheless, design A's OpAmp has room for improvement, response can be halved without risking the detection of noise.

The diode D1 plays a key role on the OpAmp's fine tuning, as by holding Vout higher than Vdd even after the glitch has expired, it provides the comparator with more time to compare the signals. This means that with the use of the diode, a comparator relatively slower than the glitch can be used, so that noise effects on Vdd do not trigger the glitch detector.

6. Conclusion

This paper has covered the factors around one of the typical attack techniques on secure devices, glitch attacks. A glitch detector capable of detecting fast glitches that common detectors built-in VR1 might struggle to detect has been presented. However, since all detectors are limited to a given detection range, this detector is not a replacement of current ones, rather, it is a supplement.

This design, like current ones, has a degree of sensitivity to process variations, which affects its already slow detection time. Nevertheless, the present design allows reducing this time by fine-tuning the OpAmp and minimizing the risk of detecting noise as a glitch.

7. Future work

The next step is to test this glitch detector on silicon, so that it can be tested in a real environment. Several versions of the detector will be instantiated and this will be tested with a random load source. Instantiated detectors will be characterized for both glitch detection and detection time.

8. Acknowledgement

The authors would like to acknowledge Louis Frew and David Dougan for their help with sourcing the voltage regulator and fine-tuning the glitch detector. The authors would also like to acknowledge EPSRC and Atmel for supporting this research.

References

[1] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks", *Proceedings of the IEEE*, Vol. 92 Issue 2, Feb.2006, pp 370-382.

[2] E.S. Kim and J.H. Kim, "Voltage glitch detection circuits and methods thereof", *US Patent Office*, US 2007/0058452 A1.

[3] C.Y. Kim, S.J. Jun, and E.S. Kim, "Voltage-glitch detection device and method for securing integrated circuit device from voltage glitch attack", *US Patent Office*, US 7,085,979 B2.

Characterization of a Voltage Glitch Attack Detector for Secure Devices

Asier GOIKOETXEA YANCI
Institute for System Level Integration
asier.goikoetxea@sli-institute.ac.uk

Stephen PICKLES
Atmel
stephen.pickles@ekb.atmel.com

Tughrul ARSLAN
University of Edinburgh
t.arslan@ed.ac.uk

Abstract

Glitch attacks are often used to inject temporal faults into secure devices and devices manipulating or holding sensitive data. The main countermeasure against these kinds of attacks involves the correct design of a built-in voltage regulator. Another important countermeasure is a glitch detector circuit. Common glitch detection approaches, such as directly monitoring the power rails, can potentially find it hard to detect fast glitches, as these become harder to differentiate from noise. This paper presents the silicon test results of a voltage glitch detector circuit which, instead of monitoring the power rails, monitors the effect of a glitch on a sensitive circuit, hence reducing the risk of detecting noise as glitches.

1. Introduction

Secure devices, such as Smart Cards, have a built-in voltage regulator. Its main function is converting the external power source level, defined by the standard [1], into a power source level the internal technology operates at. A side effect of having a voltage regulator is that it can filter some of the noise present at its input (Vcc), as well as some of the glitches a Smart Card can be subjected to [2], and still provide a clean and steady power source at its output (Vdd). What kind of noise can it actually filter out and what is the impact of the noise at the regulator's output depends on two factors, the regulator's design and the load it is subjected to, the design being the main factor.

Voltage regulators cannot filter out all noises and glitches present at their input, so certain external noises and glitches can also appear at the voltage regulators' output. This voltage alterations at the output of the voltage regulators are responsible for fault injection, hence the need for glitch detectors.

Glitch detectors are usually built-into the voltage regulators, and monitor the voltage fluctuations at the input and at the output of the voltage regulator, so that an alarm signal can be triggered in the event of a glitch.

Detecting glitches by monitoring the power rails posed the challenge of differentiating fast glitches from legitimate noise. An alternative approach of detecting glitches is monitoring the glitches' effects on a sensitive circuit. This alternative approach has been used in [3-5].

The work carried out in this paper presents the detection range characterization of four versions of the detector proposed in [5] and are compared against the detection range characterization of power rails monitoring detectors.

2. Background

Digital devices are designed to operate with a stable power source. Sudden changes in the supply voltage might not be equally spread across the whole device due to the parasitics in the power rails. As a result, different sub-circuits might be powered at different voltages, hence, enabling fault injections. Glitch attacks are sudden changes in the power supply generated by attackers aiming at injecting faults on devices.

Glitch detector circuits are used to detect abnormal fluctuations and values on the power rails that are resulting from glitch attacks. There are four main kind of glitches that can be applied to the power rails of a device: a positive glitch on Vcc, Figure 1a; a negative glitch on Vcc, Figure 1b; a negative glitch on ground, Figure 1c; and a positive glitch on ground, Figure 1d.

In devices with a built-in voltage regulator, the regulator works as a filter, where the voltage level in Vdd is a filtered version of that in Vcc. Some glitches in Vcc might be filtered out by the voltage regulator and not appear on Vdd; however, some others might have an impact on Vdd. Hence, glitch detectors are used at both sides of the voltage regulator, as shown in Figure 2.

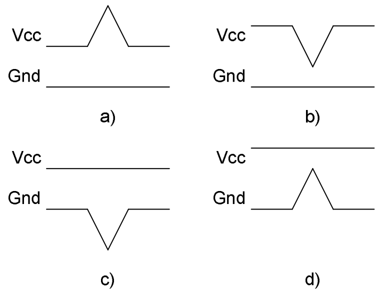


Figure 1. Common types of glitches

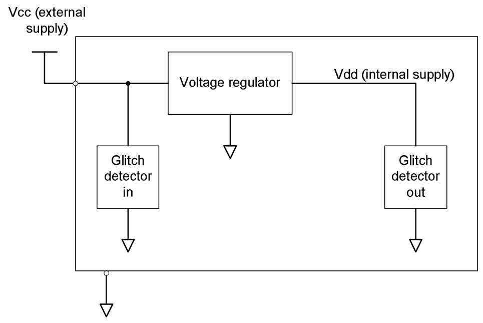


Figure 2. Glitch detectors coupled to a voltage regulator

The common approach used to detect glitches is to directly monitor the supply rails and aim at detecting it while it is happening [5]. The main challenge with this approach is for the detector not to identify permissible noise as a glitch.

Glitch detectors [3-5] aim at detecting glitches by monitoring the impact of a glitch to a sensitive circuit. The glitch detector [3] identifies glitch attacks by monitoring fault injections into single-bit registers and the sense amplifier of an SRAM. The main drawback of this design is that it is a synchronous design, whereas glitches are asynchronous.

The glitch detector [4] is more suited for the asynchronous nature of glitches, as it identifies them by monitoring RC circuits with and operational amplifier. The drawbacks of this design are: the continuous current drainage of the RC circuits; and the correct instantiation of resistors and capacitors, as process variations can affect their real value.

The glitch detector presented in [5], and shown in Figure 3, aimed at detecting glitches by monitoring the impact of a glitch on the output of a sensitive circuit, the modified inverter. This detector was designed to be used to detect positive glitches on Vdd. Although it could also be used to detect them on Vcc, this case was not tested.

Simulation results showed that this design was capable of detecting two fast glitches that current detectors built-in a given voltage regulator could not. The results also showed that this design was far slower than current ones when detecting glitches. This, however, was not an issue, as the 0.5 to 1.5 microseconds detection time are within the time required to compute command or data transactions on devices such as Smart Cards.

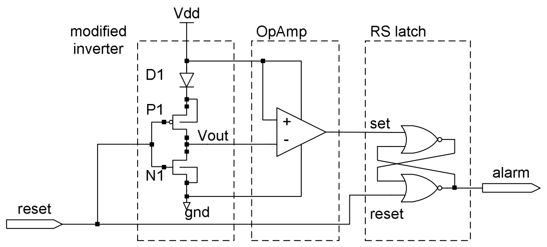


Figure 3. Proposed glitch detector

The rest of this paper presents the silicon characterization results of the glitch detector proposed in [5].

3. Test-chip

In order to test and characterize the proposed design for different scenarios, four versions of the proposed glitch detector were instantiated and powered by two different voltage regulators, as shown in Figure 4, where GD_0, GD_1, GD_2 and GD_3 represent the different glitch detector versions.

The instantiated versions differ in the following: GD_0 was designed using low leakage transistors for the diode D1 and the RS latch and high voltage transistors for the inverter and the OpAmp; GD_1 was designed using low leakage transistors throughout; GD_2 was designed using high voltage transistors throughout; and GD_3 was designed using high voltage transistors throughout too but with a different OpAmp design. Low leakage transistors are designed with a higher V_{th} than their normal counterparts. Increasing the V_{th} helps reducing the leakage current, an increasing issue with deep-submicron technologies; however, it also reduces the transistor performance when comparing to transistors with lower V_{th}.

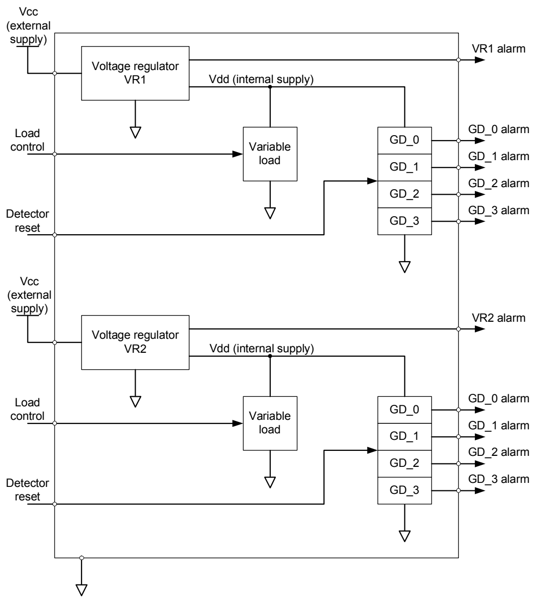


Figure 4. Block diagram of the test-chip

Two sets of the four mentioned versions were instantiated. One set of detectors monitored the output of the VR1 voltage regulator, designed for products with low security requirements. The other set monitored the output of the VR2 voltage regulator, designed for products with higher security requirements than the former one. These voltage regulators have different responses to a glitch attack, as shown in Figures 5 and 6. All the detectors within a group shared the reset input signal and had their outputs buffered to an output pin each.

In addition to the proposed glitch detectors, each voltage regulator also powered a fixed capacitive load of 2.2 nF and an independently controllable variable resistive load, which represented the typical loads the regulators could be subjected to. Each variable resistive load instance could be set to any of the following values: 80 Ohms (high load); 1K6 Ohms (medium load); or 3K2 Ohms (low load).

Finally, each voltage regulator also had built-in glitch detectors which input and output detector's combined detection range would be characterized and compared against that of the proposed design. Only the combined signal was monitored as the aim of these tests was to determine the detection range improvement provided by the proposed detector over current detection range. A side effect of this comparison approach is not being able to establish a direct comparison between the proposed designs against those built-in ones monitoring Vdd. However, the work in [5] concluded that the proposed detector could only be used in conjunction with existing ones, not as a replacement. Hence, the alarm signals of the built-in detectors of each voltage regulator were combined into a single signal and buffered to an output pin.

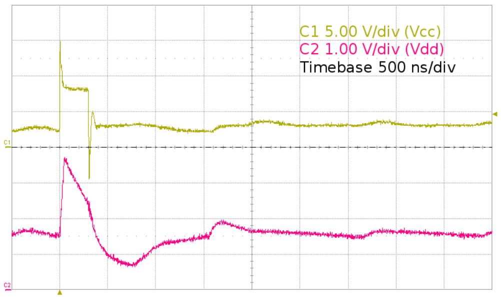


Figure 5. VR1's response to a 17V and 300ns glitch under high load, Vcc = 3V

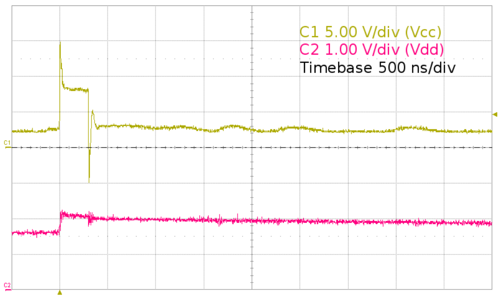


Figure 6. VR2's response to a 17V and 300ns glitch under high load, Vcc = 3V

4. Test Methodology

Each glitch detector version was characterized for detection range and detection delay using both voltage regulators, four resistive load combinations (high, medium, low and variable) and two base voltage levels, 3V and 5V. Furthermore, these tests were repeated in four different test-chips.

Glitches were applied with the HP81110A pulse generator. This device would power one voltage regulator at a time and apply a glitch to it when triggered by the control board. After applying a glitch, the output of all detectors connected to this voltage regulator were monitored and timed. A block diagram of the test setup is shown in Figure 7.

The applied glitch width range was between 10ns and 500ns, using 26 different widths in total. The glitch amplitude range was between 2V and 17V when using a base voltage of 3V, and between 2V and 15V when using a base voltage of 5V. In the former case 16 different amplitudes were used, whereas in the later one just 14. This amplitude difference was due to the fact that the pulse generator HP81110A could not generate a pulse higher than 20V (3V + 17V; 5V + 15V). In total, 416 and 364 different glitches where applied to characterize each glitch detector, for 3V and 5V base voltages respectively.

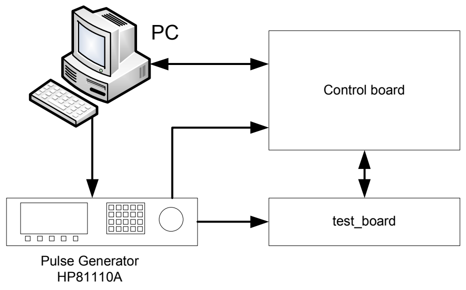


Figure 7. Diagram of the test environment

5. Results

Due to the limited space and the amount of data generated, only few representative results are shown. Figures 8 to 11 show the detection limit of the four different versions of the proposed detector when powered with the VR1 for high and low loads. Figure 12 shows the combined detection limit of the VR1 voltage regulator's built-in detectors, again for high and low loads. Figure 13 compares the detection range limit of the GD_3 detector and the VR1's combined detection range. All these figures focus on the widths between 10ns and 150ns as this is where the most useful information lies.

When looking at the detection range of a glitch detector, the boundary line determines the limit between the detected glitches (above it) and the not detected ones (below it). It must be noted that the patterns shown by this Figures are repeated when VR1 is powered with the base voltage of 5V. In that case, however, the proposed designs fail to detect few of the shorter glitches.

The detectors monitoring the VR2's output hardly detected any glitch at all. Hence no valid characterization or comparison could be done for the VR2 case.

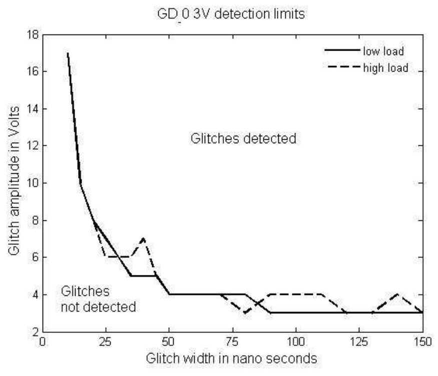


Figure 8. GD_0 3V

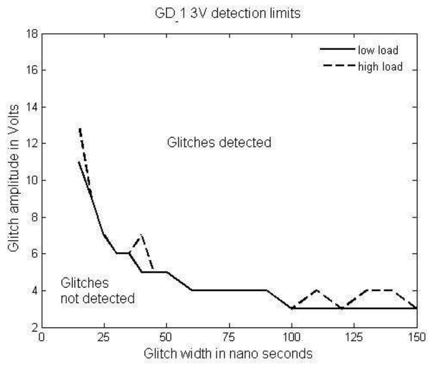


Figure 9. GD_1 3V

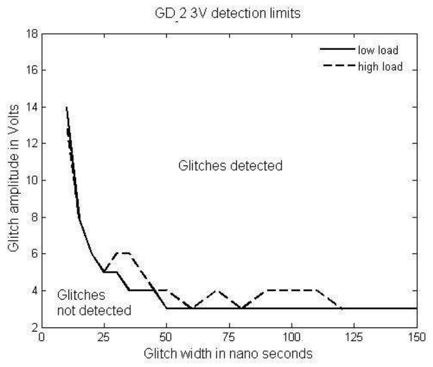


Figure 10. GD_2 3V

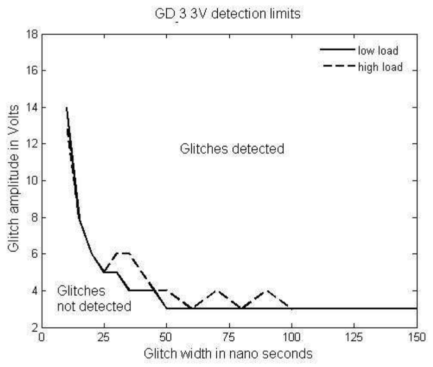


Figure 11. GD_3 3V

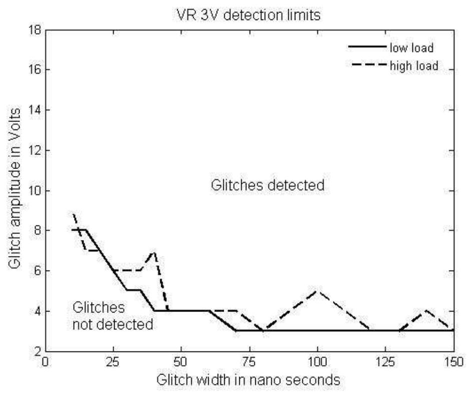


Figure 12. VR1 3V

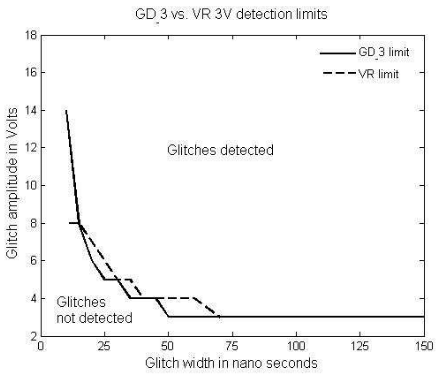


Figure 13. GD_3 vs VR1 3V

Finally, Table 1 presents the maximum, minimum and average detection time for each detector when VR1 is subject to high and low loads. The long detection times are seen close to the detection limit, whereas short detection times are seen far from the detection limit.

Table 1 Overall detection times in nanoseconds

Detector	Load	Detection time		
		min	mean	max
GD_0	High	870	1200	1640
	Low	910	1310	2460
GD_1	High	1100	1400	2005
	Low	1250	1600	2020
GD_2	High	310	420	1250
	Low	320	470	1380
GD_3	High	310	440	1300
	Low	310	460	1370
VR1	High	40	55	400
	Low	50	60	440

6. Discussion

When comparing glitch detectors, the detection range extension and detection speed are the main parameters to look at. Taking this as a basis, detectors GD_0 and GD_1 performed worse than GD_2 and GD_3 when it comes to detection range. Their timing performance was the poorest too. This is directly related to the detectors' design decisions. GD_0 and GD_1 where mainly made with low leakage transistors, which are characterized by their slow speed.

Detectors GD_2 and GD_3 showed a similar behaviour both in terms of detection range and time, where GD_3 is slightly better at detecting whereas GD_2 is slightly faster at times. In this work, expanding the detection range was a priority, hence, detector GD_3 is considered marginally better than detector GD_2.

Looking at the effect the load has in the detection range and time, it could be noticed that when VR1 is subject to lower loads, all detection ranges are expanded at the cost of speed. This phenomenon happens even with the VR1's built-in detectors.

The link between speed and load for a given glitch was described in [5]. There, the detection time was divided in two parts as shown in Figure 14: a) *t1*, the time needed for Vout to become higher than Vdd; and b) *t2*, the time between Vout becoming higher than Vdd and the raise of the alarm signal. The work in [5] showed that different loads resulted on different Vdd responses and correction speeds, producing shorter *t1* times for higher loads. Time *t2* suffered no significant variations for the different loads, as it mainly depended on the voltage difference between Vout and Vdd and the OpAmp's response time.

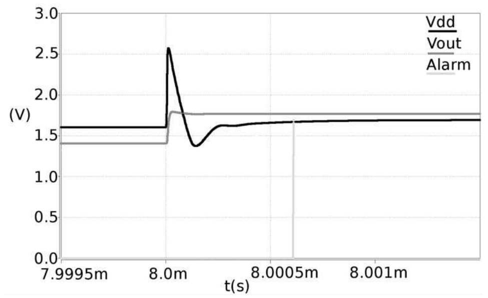


Figure 14. Simulation of glitch attack detection under a high load [5]

The detection range of the proposed detector is limited to those glitches capable of raising Vout over Vdd with a voltage difference high enough for the OpAmp to detect it. The closer a glitch gets to the detector's limit, the voltage difference at the OpAmp's input gets smaller and, therefore, it takes longer to be detected them. This phenomenon was also discussed in [5]. The results obtained in these tests corroborate the analysis made in [5].

The link between the detection range limit and the load can be explained by a combination of the two previous phenomenons where, a low load can produce a Vdd evolution such that the voltage different between Vout and Vdd becomes high enough for the glitch to be detected; whereas a high load fails to raise Vout over Vdd due to Vdd's fast recovery.

Figure 13 demonstrates the improvement in the VR1's detection range when adding the detector GD_3. This case is repeated across most of the tests carried in this work. The detection times of GD_3 (and GD_2) are, however, between 8 and 9 times slower than those of VR1.

These detection time differences are considerably big. However, they need to be put in the Smart Card context. The worst possible propagations of a successful undetected fault injection are the modification or corruption of data in a non-volatile memory (NVM), and replaying with false data to a transaction initiated by a Smart Card Reader. In the first case, the writing time required to modify an NVM is defined by its technology, which can be in the order of 2 milliseconds for the case of EEPROMs. In the seconds case, despite nowadays the CPU in a Smart Card can be running at up to 50MHz (i.e. executing about 50 instructions within a microsecond.), data and commands transactions are given to a lower speed, defined in [1], and usually taking longer than these detection times. Hence, upon the detection of a glitch, although not immediately, the Smart Card could reset itself to protect any data or communication corruption.

Regarding the VR2 voltage regulator, none of the proposed glitch detector versions showed any improvement on the detection range. This is explained by the very distinctive responses of VR1 and VR2 to the same glitch, just as shown in Figures 5 and 6. While VR1 shows an extreme fluctuation as soon as the glitch is applied, VR2 only raises its output slightly before gradually reducing it to its original value. This slightly rise is not high enough to raise Vout over Vdd, which in turn enables glitch detection. Furthermore, the proposed detector was designed with the VR1 behaviour in mind, where the regulator's output raises considerably and fast before falling also considerably and fast. This last behaviour is especially important for the detector to work.

The combined detection characterization in VR1 and VR2 hides the performance of the individual detectors at their input and output. Whilst the individual comparison was not the objective of this work, having that information could have helped with discerning the detection source provided by the built-in detectors and any overlapping between them. Also, it could have helped on determining the performance improvement when comparing to glitch detectors at the output of VR1 and VR2.

7. Conclusion

Different versions of the proposed glitch detector were characterized for detection range and times. Two of these versions showed a detection range capable of complementing the current detection range of the low security voltage regulator, while failing to provide any additional benefit when using with a more secure one.

The voltage regulator they were powered with was the single factor with the highest impact on their detection capabilities. Other two factors affecting the detection capabilities included: a) the detector design; and b) the voltage regulator load, although this last factor had a lower impact than the previous two.

The combined monitoring of the built-in detectors' behaviour did not allow us making any further analysis on how the instantiated versions compare against the individual built-in detectors, especially the internal ones of the more secure voltage regulator.

The results of four detector versions indicate that there is still room for improvement. By making the comparator even faster and more sensitive, the detection range could be expanded and the detection time reduced, whilst avoiding detecting noise as a glitch.

Finally, it would be interesting adapting this design to monitor the input of the voltage regulator and compare against the detection range of the built-in detectors, especially on the case of the secure voltage regulator, VR2.

8. Acknowledgement

The authors would like to acknowledge Nigel Davenport and John Connor, both from Atmel, for their help and support during the test lab and with the test equipment. The authors would also like to acknowledge EPSRC and Atmel for supporting this research.

References

[1] ISO/IEC 7816 Smart Card Standard.

[2] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks", *Proceedings of the IEEE*, Vol. 92 Issue 2, Feb.2006, pp 370-382.

[3] E.S. Kim and J.H. Kim, "Voltage glitch detection circuits and methods thereof", *US Patent Office*, US 2007/0058452 A1.

[4] C.Y. Kim, S.J. Jun, and E.S. Kim, "Voltage-glitch detection device and method for securing integrated circuit device from voltage glitch attack", *US Patent Office*, US 7,085,979 B2.

[5] Goikoetxea Yanci, A.; Pickles, S.; Arslan, T., "Detecting Voltage Glitch Attacks on Secure Devices," *ECSIS Symposium on Bio-inspired Learning and Intelligent Systems for Security (BLISS '08)*, pp.75-80, 4-6 Aug. 2008.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 March 2008 (20.03.2008)

PCT

(10) International Publication Number
WO 2008/033712 A2

(51) International Patent Classification:
H03K 19/003 (2006.01) *G06K 7/00* (2006.01)
H03K 5/1252 (2006.01) *H04B 1/38* (2006.01)
G01R 13/32 (2006.01)

(21) International Application Number: PCT/US2007/077715

(22) International Filing Date: 6 September 2007 (06.09.2007)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/519,382 11 September 2006 (11.09.2006) US

(71) Applicant *(for all designated States except US)*: **ATMEL CORPORATION** [US/US]; 2325 Orchard Parkway, San Jose, California 95131 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: **YANCI, Asier, Goikoetxea** [ES/GB]; Flat 3/3, 3 Buccleuch Street, Scotland, Glasgow G3 6SJ (GB).

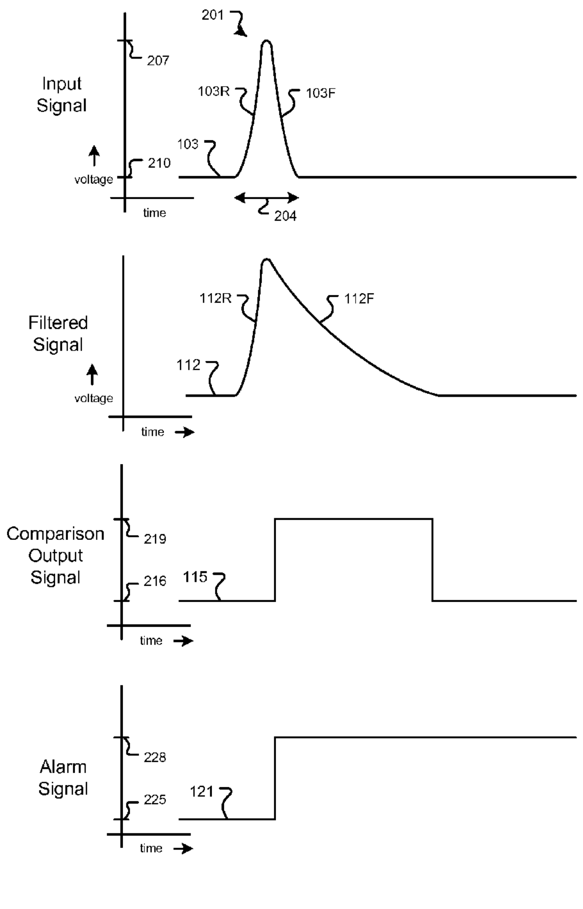
(74) Agents: **SMYTH, Matthew, J.** et al.; Fish & Richardson P.C., P.O. Box 1022, Minneapolis, Minnesota 55440-1022 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, PL,

[Continued on next page]

(54) Title: DETECTING VOLTAGE GLITCHES



(57) Abstract: In some implementations, an apparatus includes a filter circuit that receives an input signal and generates in response a filtered signal; and a comparison circuit that receives the input signal and the filtered signal, and outputs in response a comparison output signal having a first level when a magnitude of the filtered signal is less than or substantially equal to a magnitude of the input signal and a second level when the magnitude of the filtered signal is greater than the magnitude of the input signal. The filter circuit can be configured to generate the filtered signal including applying a first transfer function to the input signal when the magnitude of the input signal is substantially constant or increasing, and applying a second transfer function to the input signal when the magnitude of the input signal is decreasing.

WO 2008/033712 A2

PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *without international search report and to be republished upon receipt of that report*

Detecting Voltage Glitches

TECHNICAL FIELD

The disclosed implementations relate to electrical circuits.

BACKGROUND

Secure integrated circuit cards, commonly referred to as smart cards, may be of the form of an embedded integrated circuit hardware device that is small enough to fit into a user's pocket. Secure integrated circuit cards can be used in many situations where critical information must be stored and shared. For example, set-top boxes that facilitate pay-per-view or video-on-demand features can use a secure integrated circuit card to supply user account information to a provider along with a request for access to such features, and to subsequently decrypt encrypted digital video streams that may be provided in response to the request. As another example, a Subscriber Identity Module (SIM) card in a Global Systems for Mobile Communications (GSM) phone can be used to store a user's personal information, such as his or her phone book, device preferences, preferred network(s), saved text or voice messages and service provider information. A SIM card can allow a user, for example, to change handsets while retaining all of his or her information on the SIM card. Smart cards can be used in a variety of applications (e.g., electronic payment systems, including specialized auto-debit devices such as public transportation cards and personal identification documents, such as passports, drivers licenses, and medical identification cards).

Because of the potential value of data stored in a smart card, a conventional smart card employs various techniques to secure the protected data, including physically protecting circuits that store the protected data and encrypting the data and data processing algorithms in various ways. Hackers may employ various techniques to access or corrupt the protected data. For example, a hacker may grind off a portion of the smart card packaging in order to access internal signals and bypass security measures that may be in place. A hacker may slow or speed up a clock signal or subject a power supply to voltage glitches—actions that may have the effect of placing the hardware in a vulnerable state. In particular, a hacker may inject a voltage glitch on a smart card voltage supply or voltage reference rail, for example, to temporarily shift the threshold voltages of transistors or logic gates. In some implementations, such a voltage glitch can cause the hardware to skip certain procedures, allowing the hacker to commandeer portions of the logic, hijack data before it is encrypted, obtain information regarding device architecture or the protected data itself, etc.

SUMMARY

In some implementations, an apparatus includes a filter circuit that receives an input signal and generates in response a filtered signal; and a comparison circuit that receives the input signal and the filtered signal, and outputs in response a comparison output signal having a first level when a magnitude of the filtered signal is less than or substantially equal to a magnitude of the input signal and a second level when the magnitude of the filtered signal is greater than the magnitude of the input signal. The filter circuit can be configured to generate the filtered signal including applying a first transfer function to the input signal when the magnitude of the input signal is substantially constant or increasing, and applying a second transfer function to the input signal when the magnitude of the input signal is decreasing.

The input signal can be a voltage signal corresponding to a voltage supply rail of a system. The filter circuit and the comparison circuit can be configured to detect a voltage glitch on the voltage supply rail. The filter circuit and the comparison circuit can be configured to detect a voltage glitch on the voltage supply rail that causes, within a predetermined time period, a level of the voltage supply rail to begin at substantially a nominal voltage level, rise to a maximum voltage level, and settle back to substantially the nominal voltage level. In some implementations, the predetermined time period is substantially 10 nanoseconds or less. In some implementations, the predetermined time period is substantially 100 nanoseconds or less.

In some implementations, the filter circuit includes a complementary metal-oxide semiconductor (CMOS) inverter having a voltage supply input, a voltage reference input, a logic input and a logic output; and a diode having an anode terminal and a cathode terminal. In some implementations, the voltage supply input of the CMOS inverter is coupled to the cathode terminal of the diode, the anode terminal of the diode is coupled to the input signal, the voltage reference input of the CMOS inverter is coupled to a ground reference of the system, the logic input of the CMOS inverter is coupled to a voltage level substantially corresponding to the ground reference of the system, and the logic output of the CMOS inverter provides the filtered signal. The first transfer function can correspond to a charging function of parasitic capacitance in the CMOS inverter and current through a portion of the CMOS inverter and the diode when the diode is in a forward-biased state. The second transfer function can correspond to a discharging function of parasitic capacitance in the CMOS inverter and current through a portion of the CMOS inverter and the diode when the diode is in a reverse-biased state.

The comparison circuit can include a comparator having a positive input, a negative input and a comparator output; wherein, the positive terminal is coupled to the input signal, the negative terminal is coupled to the logic output of the CMOS inverter and the comparator output provides the comparison output signal.

The apparatus can further include an alarm circuit that receives the comparison output signal and outputs in response an alarm output signal having a first mode or a second mode, wherein the alarm circuit is configured to initially output the alarm output signal in the first mode and output the alarm signal in the second mode when the comparison output signal transitions to the second level. The alarm circuit can include at least one of a latch; a set-reset flip-flop; or a circuit configured to store a value, receive an input, and provide an output based on the input relative to the stored value. The alarm circuit can be configured to persistently output the alarm signal in the second mode once the comparison output signal transitions to the second level.

The apparatus can further include a reset circuit that is configured to cause the alarm circuit to output the alarm signal in the first mode following a reset condition. The apparatus can further include a protective circuit that is activated in response to the alarm circuit outputting the alarm signal in the second mode. The protective circuit can include a reset circuit that resets at least a portion of another circuit that is coupled to the input signal. The protective circuit can include a power control circuit that powers down at least a portion of another circuit that is coupled to the input signal.

In some implementations, a method includes receiving an input signal; determining if a magnitude of the input signal is increasing or remaining substantially constant, or decreasing; generating a filtered signal including applying a first transfer function to the input signal if the magnitude is increasing or remaining substantially constant, and applying a second, different transfer function to the input signal if the magnitude is decreasing; and comparing the filtered signal and the received input signal and providing an output signal having a first level if a magnitude of the filtered signal is less than or equal to the magnitude of the input signal and a second level if the magnitude of the filtered signal is greater than the magnitude of the input signal.

The method can further include latching the value of the output signal when the output signal transitions from the first level to the second level. Providing an output signal having the second level can include detecting a voltage glitch on the input signal. Detecting a voltage glitch on the input signal can include detecting a voltage glitch that causes, within a predetermined time period, a level of the input signal to begin at substantially a nominal level, rise to a maximum voltage level, and settle back to substantially the nominal voltage level. In some implementations, the predetermined time period is substantially 10 nanoseconds or less. In some implementations, the predetermined time period is substantially 100 nanoseconds or less.

In some implementations, a method includes generating an input signal corresponding to a level of a supply voltage of a device; determining if the level is increasing or remaining substantially constant, or decreasing; generating a filtered signal including applying a first transfer function to the input signal if the level is increasing or remaining substantially constant, and applying a second, different transfer function to the input signal if the level is decreasing; and asserting an alarm signal if a magnitude of the filtered signal exceeds a magnitude of the input signal.

The method can further include activating a protective circuit if the magnitude of the filtered signal exceeds the magnitude of the input signal. Activating the protective circuit can include resetting at least a portion of the device. Activating the protective circuit can include powering down at least a portion of the device. The method can further include comparing the filtered signal and the input signal to determine if the magnitude of the filtered signal exceeds the magnitude of the input signal.

The details of one or more implementations are set forth in the accompanying drawings and the description below. Other features, objects, and advantages will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF DRAWINGS

FIG. 1 is a block diagram of a system that can detect voltage glitches.

FIG. 2 provides a series of waveforms that depict various signals with reference to FIG. 1.

FIG. 3 is a schematic diagram of an exemplary circuit that can implement the system shown in FIG. 1.

FIGS. 4A-F illustrate various alternative configurations for the circuit that is shown in FIG. 3.

FIG. 5 is a block diagram of an exemplary application in which a voltage glitch detector can be employed.

Like reference symbols in the various drawings indicate like elements.

DETAILED DESCRIPTION

FIG. 1 is a block diagram of a system 100 that can detect voltage glitches on an input signal 103. As shown, in one implementation, the system 100 includes a filter circuit 106 and a comparison circuit 109. The filter circuit 106 receives as input the input signal 103 (e.g., x) and generates in response a filtered signal 112 (e.g., y). In the implementation shown, the filter circuit 106 generates the filtered signal 112 by applying one of two transfer functions to the input signal 103. In particular, if the magnitude of the input signal 103 is substantially constant or increasing (dx/dt is greater than or equal to zero), the filter circuit 106 applies a first transfer function, $H_1(x)$, to the input signal 103. If the magnitude of the input signal 103 is decreasing (dx/dt is less than zero), the filter circuit 106 applies a second transfer function, $H_2(x)$, to the input signal 103.

In some implementations, the first transfer function $H_1(x)$ substantially passes the input signal 103 through with little change, and the second transfer function $H_2(x)$ applies a decay function to the input signal 103 that has a small time constant. Thus, the two transfer functions $H_1(x)$ and $H_2(x)$ can have the effect of substantially passing a rising edge of a glitch through while delaying the falling edge of the glitch, as is depicted in FIG. 2 and described in greater detail below.

The time constant associated with the second transfer function $H_2(x)$ can influence the effect the filter circuit 106 has on a glitch on the input signal 103. For example, in some implementations, a very small time constant has very little effect on a long-duration glitch, or a glitch with a very slow falling edge; on the other hand, the same small time constant can have a much greater relative effect on a shorter-duration glitch, or a glitch with a very fast falling edge. Accordingly, the sensitivity of the filter circuit 106 can be tuned by adjusting the time constant corresponding to the second transfer function $H_2(x)$.

The comparison circuit 109 receives as input the input signal 103 and the filtered signal 112 and compares their relative magnitudes. Based on the comparison, the comparison circuit 109 outputs a comparison output signal 115 that can have a first level when the magnitude of the filtered signal 112 is less than or substantially equal to the magnitude of the input signal 103, and a second level when the magnitude of the filtered signal 112 is less than the magnitude of the input signal 103.

In operation, the comparison circuit 109 and the filter circuit 106 can detect glitches in the following manner. At steady state (e.g., a state in which the magnitude of the input signal 103 is substantially constant), the filter circuit 106 applies the first transfer function $H_1(x)$ to the input signal 103 to generate a filtered signal 112 having a substantially similar profile as the input signal 103 (e.g., substantially similar shape with a possible small decrease in magnitude). At steady state, because the filtered signal 112 has a substantially similar profile as the input signal 103, the comparison circuit 109 will provide the comparison output signal 115 at the first level.

When the input signal 103 is increasing, as on the rising edge of a positive glitch, the filter circuit 106 will continue applying the first transfer function $H_1(x)$, which, as described above, can pass through the input signal 103 substantially unchanged. Accordingly, when the input signal 103 is increasing, the comparison circuit 109 will again provide the comparison output signal 115 as the first level, since the input signal 103 and the filtered signal 112 have substantially similarly magnitudes.

When the input signal 103 is decreasing, as on a falling edge of a positive glitch, the filter circuit 106 will apply the second transfer function $H_2(x)$ to the input signal 103. As described above, the second transfer function $H_2(x)$ is, in some implementations, a delay function having a time constant. In these implementations, the filter circuit 106 causes the filtered signal 112 to have a delayed and differently shaped response relative to the input signal 103. Accordingly, for glitches in the input signal 103 having sharp falling edges (e.g., fast glitches), the magnitude of the input signal 103 itself will decrease faster than the magnitude of the corresponding filtered signal 112, resulting in a period of time during which the filtered signal 112 will have a greater magnitude than the magnitude of the input signal 103. The comparison circuit 109 detects this condition and provides the comparison output signal 115 during this period at the second level. After some period of time, (e.g., a period of time related to the time constant), the magnitude of the filtered signal 112 will settle back to substantially the same magnitude as the input signal 103, and the comparison circuit 109 will again provide the comparison output signal 115 at the first level.

In some implementations, the comparison circuit 109 requires a minimum amount of time (e.g., a hold time) to detect a difference in magnitudes of the input signal 103 and the filtered signal 112. To meet such a minimum hold time, the time constant can be appropriately tuned (e.g., made sufficiently large) to ensure that a characteristic glitch that the system 100 is designed to detect will cause the filtered signal 112 to have a magnitude that is greater than the magnitude of the input signal 103 for a period of time that is at least as long as the minimum hold time.

To record the occurrence of a glitch even after the magnitude of the filtered signal 112 has settled back to substantially the same magnitude as the input signal 103, and the comparison circuit 109 has again begun providing the comparison output signal 115 at the first level, the system 100 can include an alarm circuit 118. In some implementations, the alarm circuit 118 latches the value of the comparison output signal 115 when the comparison output signal 115 transitions from the first level to the second level. In this manner, the comparison circuit 109 can provide a signal (e.g., the comparison output signal 115) that indicates that a glitch *is occurring*, whereas the alarm circuit 118 can provide a signal (e.g., an alarm output signal 121) that indicates that a glitch *has occurred*. Other portions of the system 100 (not shown in FIG. 1) can employ the alarm output signal 121 to initiate any action that may be desirable following a glitch, as is described in greater detail with reference to FIG. 5.

FIG. 2 provides a series of waveforms that depict various signals described above with reference to FIG. 1. In particular, FIG. 2 shows a glitch 201 on the input signal 103 having duration 204 and a peak value 207. As shown, the glitch 201 has a rising edge 103R, during which the magnitude of the input signal 103 rises from a nominal value 210 toward the peak value 207, and a falling edge 103F, during which the magnitude of the input signal 103 falls back to the nominal value 210. As shown, the glitch 201 falls smoothly back to the nominal value 210, but in other implementations, the glitch 201 includes a region in which the magnitude of the input signal 103 undershoots the nominal value 210, then increases towards the nominal value 210 (or slightly overshoots the nominal value 210, again undershoots the nominal value 210) and gradually settles out. In these implementations, the duration 204 can include the period starting with the rising edge 103R and ending with the input signal 103 substantially settled out at the nominal value 210.

As shown in one implementation, the filtered signal 112 increases in magnitude in substantially the same manner that the input signal 103 increases. That is, the filtered signal 112 has a rising edge 112R that has substantially the same profile (e.g., slope, timing and relative increase in magnitude) as the input signal 103, when the input signal 103 is increasing. As described above, the rising edge 112R of the filtered signal 112 can result from the filter circuit 106 (shown in FIG. 1) applying the first transfer function $H_1(x)$ to the input signal 103.

As shown in one implementation, the filtered signal 112 deceases in magnitude in a delayed manner relative to a corresponding decrease in magnitude of the input signal 103. That is, the falling edge 112F of the filtered signal 112 is less steep than the corresponding falling edge 103F of the input signal 103. As described above, the falling edge 112F of the filtered signal 112 can result from the filter circuit 106 applying the second transfer function $H_2(x)$ to the input signal 103. In one particular implementation, the profile (e.g., slope of the falling edge 112F at its "midpoint") can be related to the time constant of the second transfer function $H_2(x)$. That is, the slope of the falling edge 112F at its midpoint can be greater in absolute magnitude when the time constant is smaller, relative to the slope of the falling edge 112F at its midpoint when the time constant is larger.

FIG. 2 also shows the comparison output signal 115. In the implementation shown, the comparison output signal 115 is provided at a second level 219 whenever the magnitude of the input signal 103 is less than the magnitude of the filtered signal 112. Based on the level of the comparison output signal 115, the alarm circuit 118 can initially provide an alarm output signal 121 in a first mode (depicted by the level 225), then provide the alarm output signal 121 in a second mode (depicted by the level 228).

FIG. 3 is a schematic diagram of an exemplary circuit 300 that can implement the system 100 that is shown in FIG. 1. As shown, the input signal 103 can be a voltage signal corresponding to a voltage supply rail 302 of a system. The filter circuit 106, in one implementation, includes a complementary metal-oxide semiconductor (CMOS) inverter 305 having a pMOS (p-channel metal oxide semiconductor (MOS)) transistor 305A, an nMOS (n-channel MOS) transistor 305B, a voltage supply input 305C, a voltage reference input 305D, a logic input 305E and a logic output 305F. The filter circuit 106, as shown, also includes a diode 308 having an anode terminal 308A and a cathode terminal 308B. The anode terminal 308A of the diode 308 is coupled to the input signal 103 (e.g., the voltage supply rail 302 of a corresponding system), the cathode terminal 308B of the diode 308 is coupled to the voltage supply input 305C of the CMOS inverter 305. The voltage reference input 305D of CMOS inverter 305 is coupled to a ground reference 311 of the corresponding system, as is the logic input 305E of CMOS inverter 305. Finally, the logic output 305F of the CMOS inverter 305 provides that filtered signal 112, which is coupled to the comparison circuit 109.

The comparison circuit 109, in one implementation as shown, includes a comparator 314 having a positive input 314A, a negative input 314B, and a comparator output 314C. As shown, the positive input 314A of the comparator 314 is coupled to the input signal 103 (e.g., the voltage supply rail 302), and the negative input 314B of the comparator 314 is coupled to the filtered signal 112, which, as described above, is provided by the logic output 305F of the CMOS inverter 305. The comparator output 314C is coupled to the alarm circuit 118. In one implementation, as shown, the alarm circuit 118 includes a set-reset flip-flop 317. Operation of the exemplary circuit 300 is now described.

The output voltage of the filter circuit 106 (voltage at the logic output 305F and on the filtered signal 112) is described relative to three modes of the input signal 103: the input signal 103 at steady state (e.g., substantially constant); the input signal 103 increasing in magnitude; and the input signal 103 decreasing in magnitude. When the input signal 103 is at steady state, the voltage of the filtered signal 112 is also at steady state and at a magnitude that is slightly lower than the magnitude of the input signal 103. In particular, as shown, the CMOS inverter 305 is configured to provide at its logic output 305F the filtered signal 112 at a value that corresponds to a logic '1' (which is generally characterized by a level that is close to the supply voltage), since its logic input 305E is tied to the ground reference 311 of the system. Because of the diode 308, the maximum steady-state voltage of the logic output 305F of the CMOS inverter 305 will be approximately the voltage of the input signal 103, less a standard voltage drop across the diode 308 when the diode is forward-biased, less any (small) voltage drop across the channel of the pMOS transistor 305A.

When the voltage of input signal 103 is increasing, as on the rising edge of a voltage glitch, the voltage of the logic output 305F also increases. At a fixed current, the rate of increase of the voltage of the logic output 305F is limited by parasitic capacitance of the CMOS inverter 305 (depicted as capacitor 320). However, an increased current is available to charge the parasitic capacitance 320, and counteracts this limit to the increase in voltage. In particular, when the input signal 103 increases, the forward bias across the diode 308 also increases slightly, allowing more current to flow through the diode 308. Moreover, as the voltage of the input signal 103 increases, the gate-source voltage of the pMOS transistor 305A also increases (in absolute magnitude), allowing more current to also flow through the pMOS transistor 305A and charge the parasitic capacitance 320. Because the charge rate of capacitance is related to the current that flows through the capacitance, the increased current, in some implementations, charges the parasitic capacitance 320 very quickly, resulting in a voltage at the logic output 305F (and, correspondingly, on the filtered signal 112) having a profile that is substantially similarly to the voltage profile on the input signal 103 (e.g., with a slightly smaller magnitude, due to the voltage drop(s) described above). Put another way, the CMOS inverter 305 and the diode 308 can pass through increases in voltage on the input signal 103 with little change. Put yet another way, the behavior of the filtered signal 112 (e.g., y) can be described as a first transfer function (e.g., $H_1(x)$) applied to the input signal 103 (e.g., x) when the magnitude of the input signal 103 is substantially constant or increasing, and the first transfer function in can be described as approximately a unity, or near-unity pass-through function.

When the voltage of the input signal 103 is decreasing, as on the falling edge of a voltage glitch, the voltage of the logic output 305F also decreases. In particular, a decreasing voltage on the input signal 103 reduces the forward bias across the diode 308 and reduces the gate-source voltage of the pMOS transistor 305A, both of which reduce the current available to maintain the voltage on the parasitic capacitance 320. As less current is available to maintain the voltage on the parasitic capacitance 320, the voltage will decrease as charge on the parasitic capacitance leaks out or is dissipated by other current paths that are not shown in FIG. 3; however, the rate of decrease in voltage will be limited by the parasitic capacitance 320 (e.g., the voltage will decrease based on a time constant associated with the parasitic capacitance 320). Accordingly, the parasitic capacitance 320 will discharge slower than it is charged. In implementations in which the input signal 103 decreases in magnitude very quickly (e.g., a short-duration glitch, or glitch with a steep falling edge), the voltage of the input signal 103 can dip below the voltage of the filtered signal 112, as the parasitic capacitance is discharging. The duration of time during which such a condition can exist will depend on the time constant associated with the parasitic capacitance 320 and to the slope of the falling edge of the glitch (e.g., the "sharpness" of the glitch). That is, for fast glitches, the CMOS inverter 305 and the diode 308 can pass through decreases in voltage on the input signal 103 in a delayed manner. Put another way, the behavior of the filtered signal 112 (e.g., y) can be described as a second transfer function (e.g., $H_2(x)$) applied to the input signal 103 (e.g., x) when the magnitude of the input signal 103 is rapidly decreasing relative to a time constant corresponding to the second transfer function, and the second transfer function can be described as a delay function.

As described above, when the input signal 103 is at a steady state or increasing, in magnitude, the filtered signal 112 will have a profile that is similar to the input signal 103, but the magnitude of the filtered signal 112 will be slightly less than the magnitude of the input signal 103. Accordingly, when the magnitude of the input signal 103 is at a steady state or increasing, the comparison circuit 109 will provide the comparison output 115 at a first level. In particular, in one implementation as shown, the comparator 314 will provide the comparison output signal 115 at a first level (e.g., a logic '0'), indicating that the level of the positive input 314A is greater than the level of the negative terminal 314B. However, as described above in some implementations, when the input signal 103 is decreasing in magnitude, a condition can exist in which the voltage of the input signal 103 can dip below the voltage of the filtered signal 112, at which point, the comparator 314 will provide the output comparison signal 115 at a second level (e.g., a logic '1'), indicating that the level of the positive input 314A is less than the level of the negative input 314B.

As shown in one implementation, the alarm circuit 118 (e.g., a set-reset (SR) flip-flop 317) can capture the transition of the comparison output signal 115 from the first state to the second state and can accordingly switch from a first mode (e.g., a logic '0' on the alarm signal 121, indicating a non-alarm condition) to a second mode (e.g., a logic '1' on the alarm signal 121, indicating an alarm condition). The alarm signal can be provided to other circuits to initiate an appropriate action, examples of which are provided with reference to FIG. 5.

FIG. 3 illustrates one example circuit 300 that can detect positive glitches on an input signal, such as the positive voltage rail of a power supply. Various other circuits are contemplated. For example, FIG. 4A illustrates a circuit 401 that includes a reset line 405 by which the alarm portion of the circuit can be reset. As other examples, FIGS. 4B-4F illustrate various alternative configurations for the filter circuit 106 portion of the circuit 300. In particular, FIG. 4B illustrates a circuit in which the input signal 103 can be applied to the logic input of the CMOS inverter; FIG. 4C illustrates a variation of the circuit shown in FIG. 4B, in which the filtered signal 112 is provided at the cathode terminal/CMOS inverter voltage supply junction, rather than at the logic output of the CMOS inverter. FIGS. 4D-F illustrate variations in which a second diode 410 is added between the CMOS voltage reference input and the voltage reference itself, and the filtered signal 112 is provided by various different terminals of the CMOS inverter.

As the reader will appreciate, the variations shown in FIGS. 4A-4F can be combined with variations in the configuration of the comparison circuit 109 (shown in FIG. 3) in order to detect either positive or negative glitches on either a voltage supply rail or a voltage reference rail. Glitches can also be detected on negative voltage supply rails. In addition, multiple different circuits can be combined to detect multiple kinds of glitches (e.g., positive glitches on a voltage supply rail, negative glitches on a voltage supply rail, positive glitches on a voltage reference rail, negative glitches on a voltage reference rail, etc.). Moreover, multiple circuits configured to detect the same kind of glitch can be provided, each with different time constants, to detect glitches of varying durations or glitches that have rising or falling edges with varying slopes. For example, applicants simulated a first circuit design having a nominal supply voltage of 1.7 V and a glitch detector capable of detecting a 100 nanosecond (ns) glitch having a peak voltage of 7 V. Applicants simulated a second circuit design having a nominal supply voltage of 1.7 V and a glitch detector capable of detecting a 10 ns glitch having a peak voltage of 15 V.

FIG. 5 is a block diagram of an exemplary smart card 500—a device in which voltage detection systems and methods described herein can be employed. The smart card 500 includes a processor 501 that can execute programming instructions stored in memory 504 in order to process data that is also stored in the memory 504 or received through an interface 507. The memory 504 can represent multiple different kinds of memory, such as, for example, ROM or RAM, flash, DRAM, SRAM, etc. For example, in some implementations, program instructions are stored on ROM, and the processor 501 uses some form of RAM to store intermediate data as the programming instructions are executed. The interface 507 can work in conjunction with a wireless communication channel (not shown) that includes, for example, RF (radio frequency) signals that are adapted for a particular communication protocol (e.g., a protocol characterized by ISO/IEC 14443 or ISO/IEC 15693 (ISO refers to the International Organization for Standardization; IEC refers to the International Electrotechnical Commission)). In some implementations, the interface 507 works in conjunction with a wired communication channel (not shown) that is adapted for a particular communication protocol (e.g., a protocol characterized by ISO/IEC 7816 or ISO/IEC 7810).

Together, the processor 501, memory 504 and interface 507, along with other components (not shown), make up a power load 510 that is supplied with power by a power system that is now described. In one implementation, as shown, the smart card 500 can receive power from one of three sources: an internal power storage device 513 (e.g., a battery or power storage capacitor), a direct external power source (e.g., through external power contacts 517A and 517B), or an indirect external power source (e.g., through energy transmitted by radio frequency or other electromagnetic radiation external to the smart card and received by the smart card through a power coil 520). In some implementations, a smart card 500 only employs one of above-described power sources; in other implementations, a smart card employs multiple methods of receiving power (e.g., a combination card).

Once power is received or otherwise provided to the smart card 500, it can be processed by a power circuit 523. In some implementations, the power circuit 523 stores a portion of the received power in a local power storage device (e.g., the power storage device 513, if present). The power circuit 523 can also switch between multiple sources of power. For example, in a "combination card," the power circuit 523 can switch between a direct external power source (e.g., the power contacts 517A and 517B) and an indirect external power source (e.g., the power coil 520), depending, for example, on whether the smart card 500 is situated in a contact-based card reader that provides power to the power contacts 517A-B or whether the smart card 500 is situated in an electromagnetic field that induces current in the power coil 520.

In one implementation, as shown, the power circuit 523 includes a voltage regulator 526 that provides a voltage reference 529 and a voltage supply 532 to the power load 510. In some implementations, the voltage regulator 526 regulates the voltage supply 532 to a level that is suitable for the power load 510. As shown, the power circuit 523 provides a single voltage supply 532 and voltage reference 529, but in other implementations, additional voltage supplies and/or references can also be provided.

A glitch detector 535 is also included in the smart card 500. In some implementations, the glitch detector 535 includes circuits or performs methods that are described herein. For example, the glitch detector 535 can detect voltage glitches on the voltage supply 532 or voltage reference 529 rails of the smart card 500. The smart card 500 can also include protective circuitry 538 that initiates various actions if the glitch detector 535 detects a voltage glitch. For example, in some implementations, the protective circuitry 538 resets a portion of the smart card 500 (e.g., the processor 501 and/or memory 504) upon detection of a voltage glitch. In some implementations, the protective circuitry 538 powers down at least a portion of the smart card 500 upon detection of a voltage glitch.

In some implementations, a smart card, such as the smart card 500 described herein, is more secure than a smart card that does not include a glitch detector. In particular, the smart card 500 can detect a voltage glitch attack and initiate appropriate action, such as resetting or powering down the smart card 500, to prevent a hacker from obtaining protected information from the smart card 500 through the voltage glitch attack.

A number of implementations have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the disclosed implementations. For example, smart cards are one application for the circuits and methods described herein, but these circuits and methods can be applied to other devices. Modifications can be made to detect various kinds of glitches, including positive and negative glitches on positive, reference and negative voltage rails. Time constants can be adjusted to detect glitches of varying durations or glitches having rising or falling edges with various slopes. Accordingly, other implementations are within the scope of the following claims.

WHAT IS CLAIMED IS:

1. An apparatus comprising:

a filter circuit that receives an input signal and generates in response a filtered signal; and

a comparison circuit that receives the input signal and the filtered signal, and outputs in response a comparison output signal having a first level when a magnitude of the filtered signal is less than or substantially equal to a magnitude of the input signal and a second level when the magnitude of the filtered signal is greater than the magnitude of the input signal;

wherein the filter circuit is configured to generate the filtered signal including applying a first transfer function to the input signal when the magnitude of the input signal is substantially constant or increasing, and applying a second transfer function to the input signal when the magnitude of the input signal is decreasing.

2. The apparatus of claim 1, wherein the input signal is a voltage signal corresponding to a voltage supply rail of a system.

3. The apparatus of claim 2, wherein the filter circuit and the comparison circuit are configured to detect a voltage glitch on the voltage supply rail.

4. The apparatus of claim 3, wherein the filter circuit and the comparison circuit are configured to detect a voltage glitch on the voltage supply rail that causes, within a predetermined time period, a level of the voltage supply rail to begin at substantially a nominal voltage level, rise to a maximum voltage level, and settle back to substantially the nominal voltage level.

5. The apparatus of claim 4, wherein the predetermined time period is substantially 10 nanoseconds or less.

6. The apparatus of claim 4, wherein the predetermined time period is substantially 100 nanoseconds or less.

7. The apparatus of claim 1, wherein the filter circuit comprises:

a complementary metal-oxide semiconductor (CMOS) inverter having a voltage supply input, a voltage reference input, a logic input and a logic output; and

a diode having an anode terminal and a cathode terminal;

wherein the voltage supply input of the CMOS inverter is coupled to the cathode terminal of the diode, the anode terminal of the diode is coupled to the input signal, the voltage reference input of the CMOS inverter is coupled to a ground reference of the system, the logic input of the CMOS inverter is coupled to a voltage level substantially corresponding to the ground reference of the system, and the logic output of the CMOS inverter provides the filtered signal.

8. The apparatus of claim 7, wherein the first transfer function corresponds to a charging function of parasitic capacitance in the CMOS inverter and current through a portion of the CMOS inverter and the diode when the diode is in a forward-biased state.

9. The apparatus of claim 7, wherein the second transfer function corresponds to a discharging function of parasitic capacitance in the CMOS inverter and current through a portion of the CMOS inverter and the diode when the diode is in a reverse-biased state.

10. The apparatus of claim 7, wherein the comparison circuit comprises a comparator having a positive input, a negative input and a comparator output; wherein, the positive terminal is coupled to the input signal, the negative terminal is coupled to the logic output of the CMOS inverter and the comparator output provides the comparison output signal.

11. The apparatus of claim 1, further comprising an alarm circuit that receives the comparison output signal and outputs in response an alarm output signal having a first mode or a second mode, wherein the alarm circuit is configured to initially output the alarm output signal in the first mode and output the alarm signal in the second mode when the comparison output signal transitions to the second level.

12. The apparatus of claim 11, wherein the alarm circuit comprises at least one of a latch; a set-reset flip-flop; or a circuit configured to store a value, receive an input, and provide an output based on the input relative to the stored value.

13. The apparatus of claim 11, wherein the alarm circuit is configured to persistently output the alarm signal in the second mode once the comparison output signal transitions to the second level.

14. The apparatus of claim 11, further comprising a reset circuit that is configured to cause the alarm circuit to output the alarm signal in the first mode following a reset condition.

15. The apparatus of claim 1, further comprising a protective circuit that is activated in response to the alarm circuit outputting the alarm signal in the second mode.

16. The apparatus of claim 15, wherein the protective circuit comprises a reset circuit that resets at least a portion of another circuit that is coupled to the input signal.

17. The apparatus of claim 15, wherein the protective circuit comprises a power control circuit that powers down at least a portion of another circuit that is coupled to the input signal.

18. A method comprising:

receiving an input signal;

determining if a magnitude of the input signal is increasing or remaining substantially constant, or decreasing;

generating a filtered signal including applying a first transfer function to the input signal if the magnitude is increasing or remaining substantially constant, and applying a second, different transfer function to the input signal if the magnitude is decreasing; and

comparing the filtered signal and the received input signal and providing an output signal having a first level if a magnitude of the filtered signal is less than or equal to the magnitude of the input signal and a second level if the magnitude of the filtered signal is greater than the magnitude of the input signal.

19. The method of claim 18, further comprising latching the value of the output signal when the output signal transitions from the first level to the second level.

20. The method of claim 18, wherein providing an output signal having the second level comprises detecting a voltage glitch on the input signal.

21. The method of claim 20, wherein detecting a voltage glitch on the input signal comprises detecting a voltage glitch that causes, within a predetermined time period, a level of the input signal to begin at substantially a nominal level, rise to a maximum voltage level, and settle back to substantially the nominal voltage level.

22. The method of claim 21, wherein the predetermined time period is substantially 10 nanoseconds or less.

23. The method of claim 21, wherein the predetermined time period is substantially 100 nanoseconds or less.

24. A method comprising:

generating an input signal corresponding to a level of a supply voltage of a device;

determining if the level is increasing or remaining substantially constant, or decreasing;

generating a filtered signal including applying a first transfer function to the input signal if the level is increasing or remaining substantially constant, and applying a second, different transfer function to the input signal if the level is decreasing; and

asserting an alarm signal if a magnitude of the filtered signal exceeds a magnitude of the input signal.

25. The method of claim 24, further comprising activating a protective circuit if the magnitude of the filtered signal exceeds the magnitude of the input signal.

26. The method of claim 25, wherein activating the protective circuit comprises resetting at least a portion of the device.

27. The method of claim 25, wherein activating the protective circuit comprises powering down at least a portion of the device.

28. The method of claim 24, further comprising comparing the filtered signal and the input signal to determine if the magnitude of the filtered signal exceeds the magnitude of the input signal.

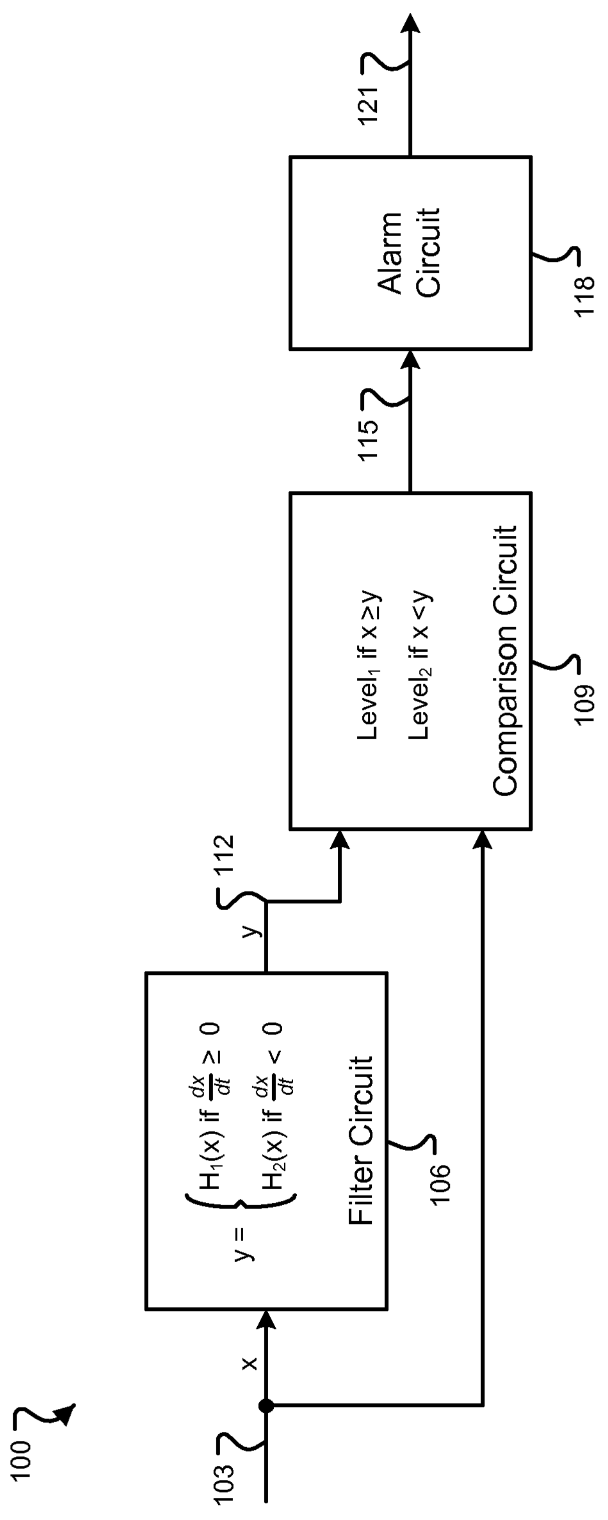

100
103
x
$y = \begin{cases} H_1(x) \text{ if } \frac{dx}{dt} \geq 0 \\ H_2(x) \text{ if } \frac{dx}{dt} < 0 \end{cases}$
Filter Circuit
106
y
112
$\text{Level}_1 \text{ if } x \geq y$
$\text{Level}_2 \text{ if } x < y$
Comparison Circuit
109
115
Alarm Circuit
118
121

FIG. 1

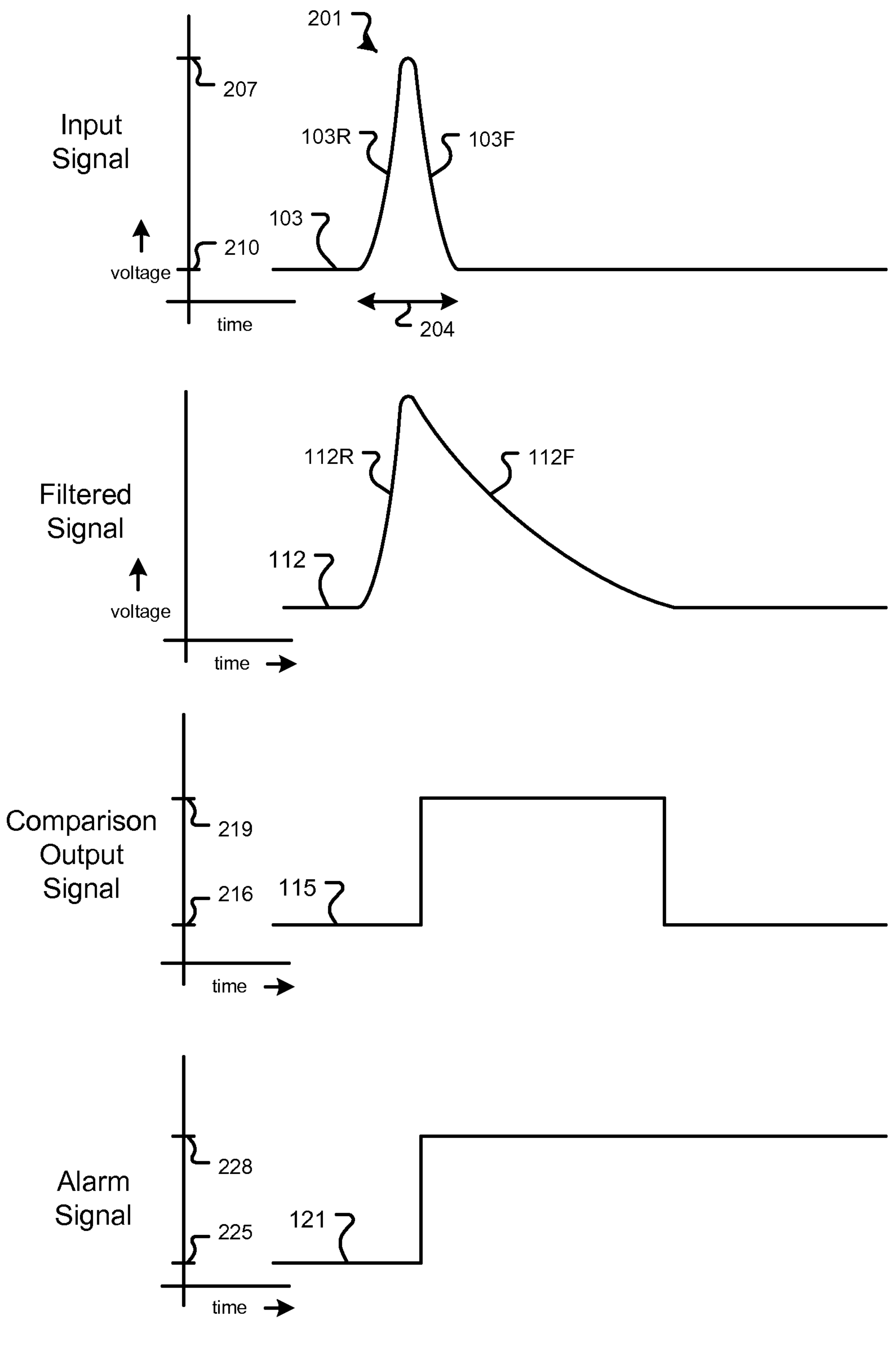

Input Signal
voltage
time
201
207
210
103R
103F
103
204
Filtered Signal
voltage
time
112R
112F
112
Comparison Output Signal
time
219
216
115
Alarm Signal
time
228
225
121

FIG. 2

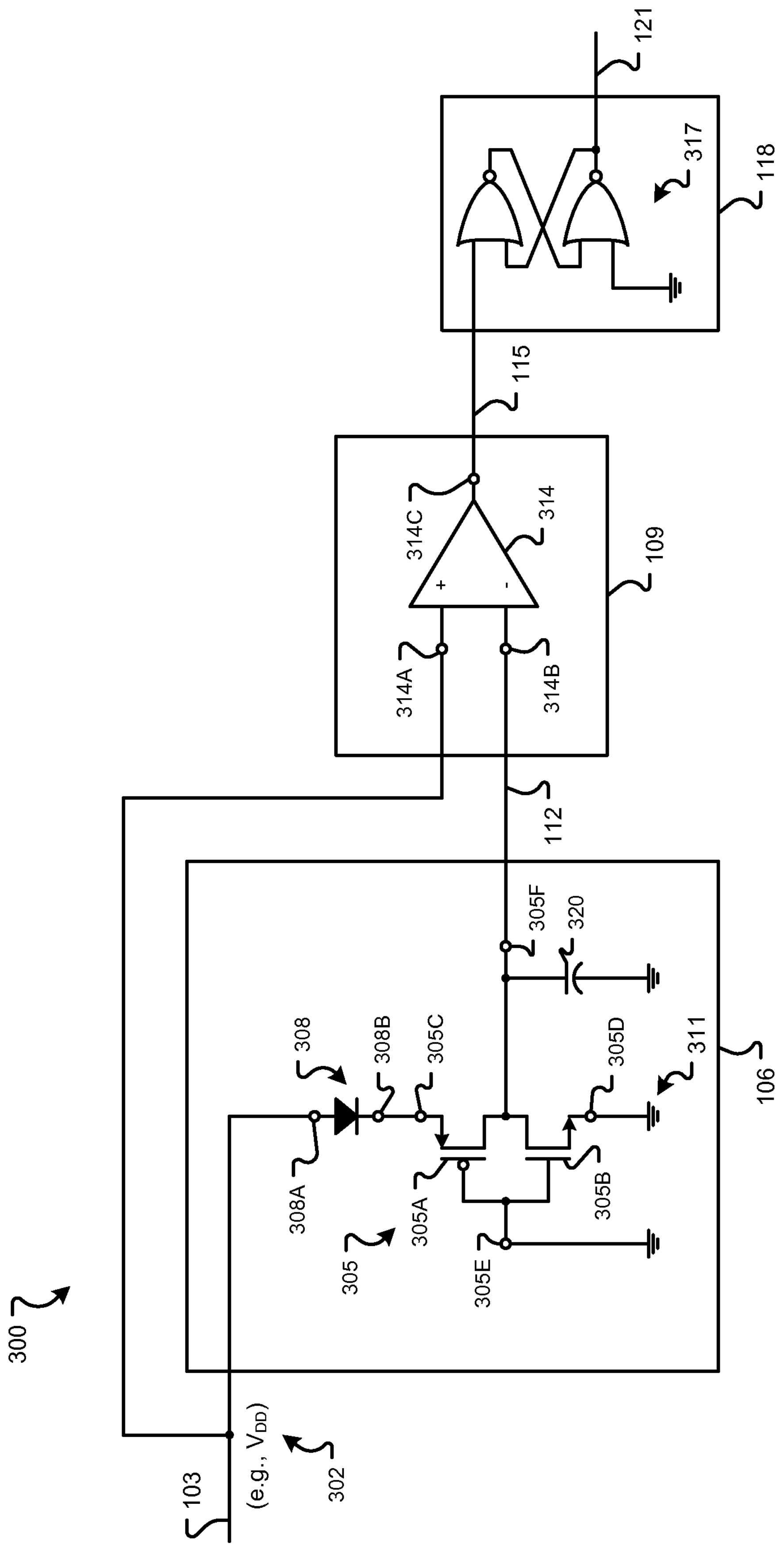

300
103
(e.g., V_{DD})
302
106
305
305A
305B
305C
305D
305E
305F
308
308A
308B
311
320
112
109
314
314A
314B
314C
+
-
115
118
317
121

FIG. 3

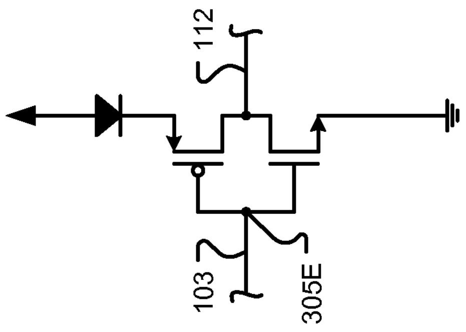


FIG. 4B

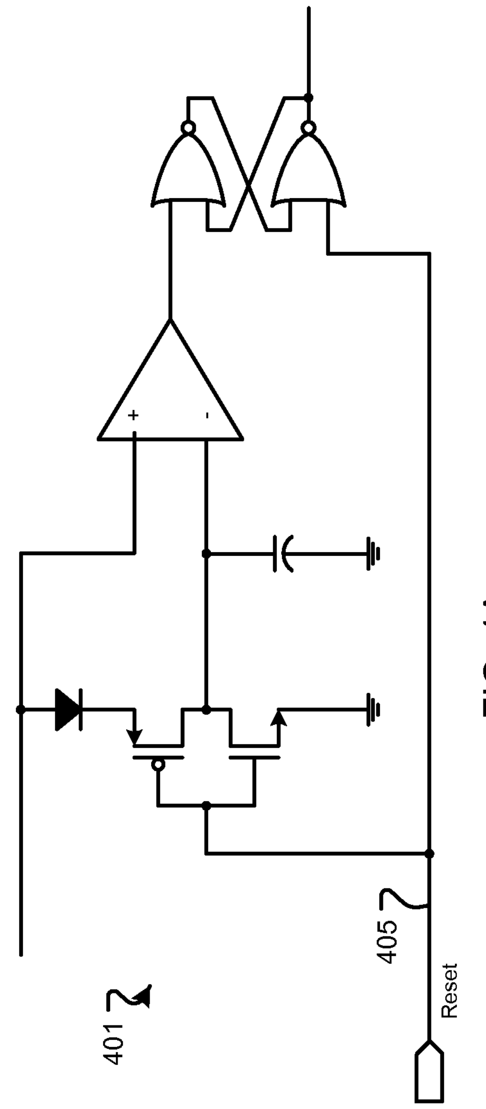


FIG. 4A

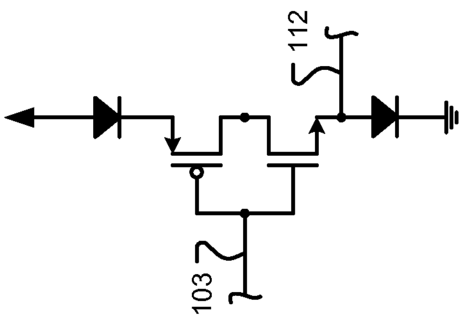


FIG. 4F

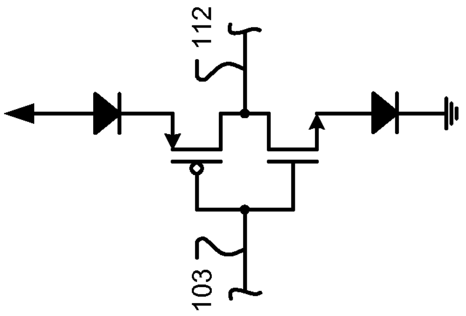


FIG. 4E

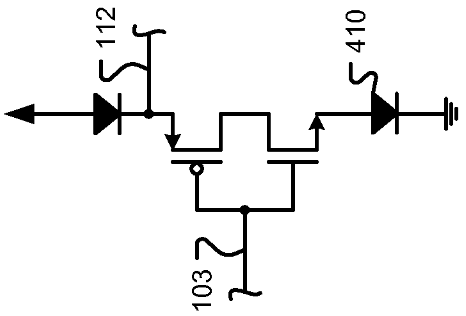


FIG. 4D

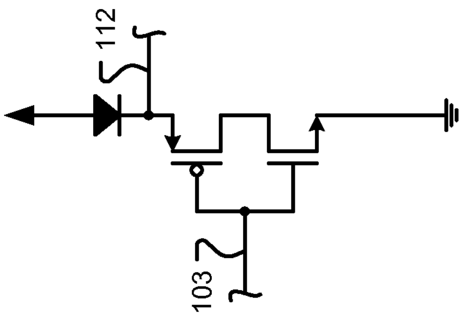


FIG. 4C

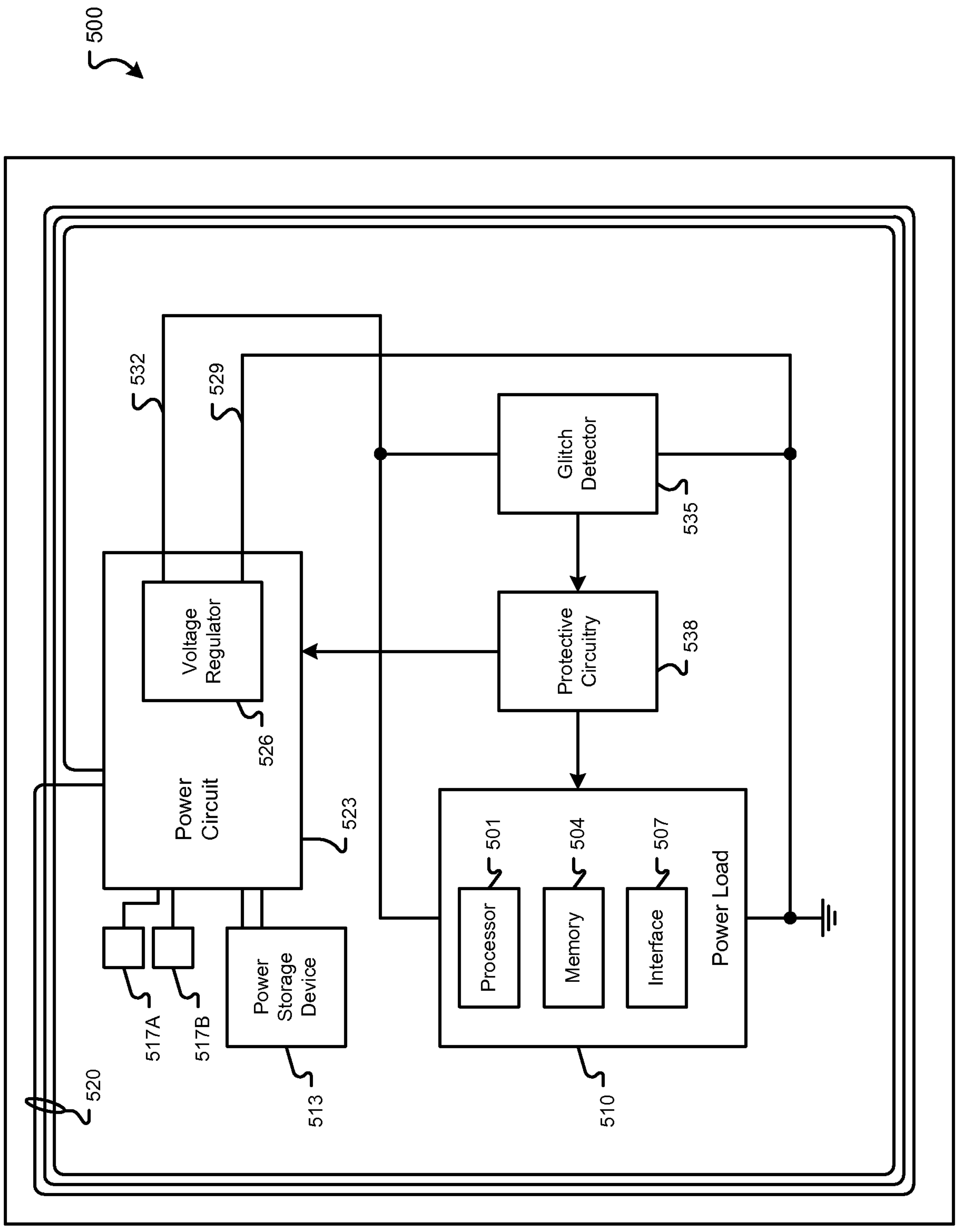

500
Glitch Detector
535
Protective Circuitry
538
Voltage Regulator
526
532
529
Power Circuit
523
Processor
501
Memory
504
Interface
507
Power Load
510
Power Storage Device
513
517A
517B
520

FIG. 5