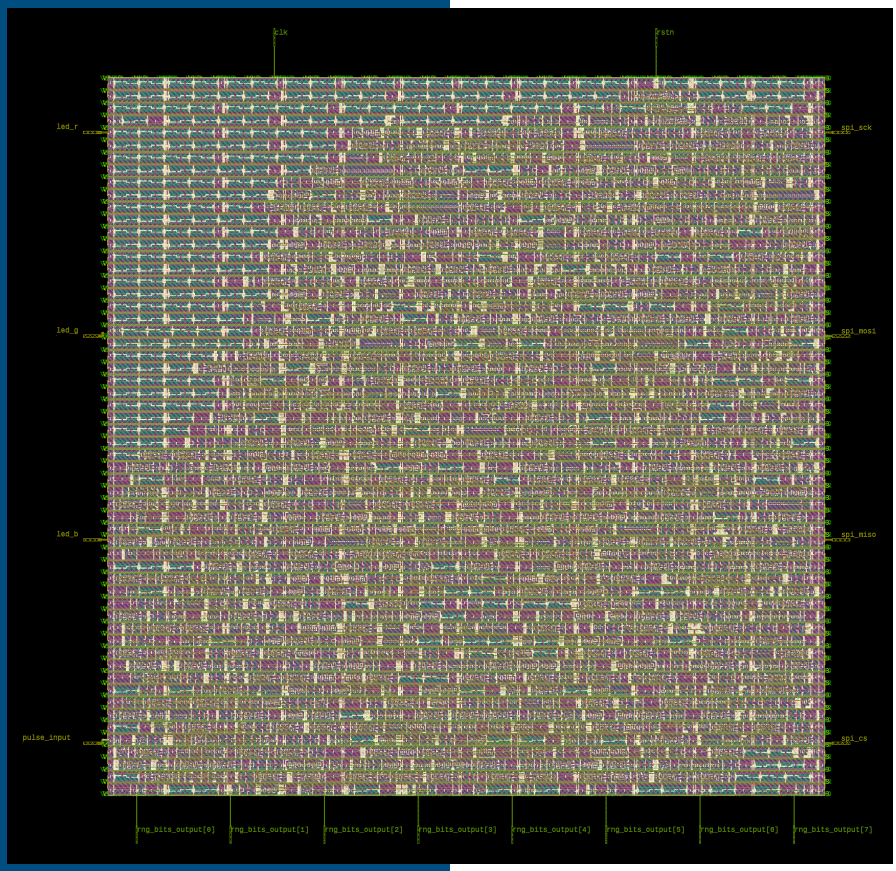
Radioactive Decay Random Number Generator

2025 Microelectronics Project for 4MEO



Gr.3

Frédéric Druppel 18053
Jonathan Borlée 195242

01/06/2025

ECAM
BRUSSELS ENGINEERING SCHOOL

Summary

Introduction

For the microelectronics course of the first year of the master's program in electronics engineering at the ECAM Brussels Engineering school, we were assigned to develop a project that could be transformed into an integrated circuit chip.

We opted to create a random number generator that utilizes radioactive decay as its entropy source.

Context

Random number generators are typically classified into two categories : random and pseudorandom.

- Pseudorandom number generators (PRNGs) are widely used in computers. They generate numbers that look to be random, but are truly based on a seed value (a starting point) and an algorithm. It means that if you know the algorithm and the seed, you can figure out all the numbers that will be generated. It's fast and adequate for most uses, but not secure or truly random.
- True random number generators (TRNGs) leverage physical effects that are indeterminate. There are many examples like noise in electronic circuits, but in the case of this project, we are using radioactive decay.

Radioactive decay is a natural phenomenon that occurs randomly at the atomic level. Despite each atom type having an average rate of decay, it is impossible to tell exactly when any particular atom will decay. The delay between decays is random, though it has a statistical distribution (an exponential distribution). This makes it a good choice for producing true random numbers.

In this project, we will measure the time between two decay events and convert that into digital values. Since the time between two decays is random, the numbers we get should also be random.

TRNGs are commonly used in domains where the unpredictability and entropy of random numbers are crucial for system integrity and security. This is because PRNGs, with their deterministic nature, fall short in such situations.
In cryptography, TRNGs are used for generating secure encryption keys, initialization vectors, and nonces. These components prevent attackers from predicting or replicating them. TRNGs are also essential in hardware security modules (HSMs) and smartcards, where embedded TRNGs safeguard sensitive data. In regulated industries like online gambling and lotteries, TRNGs ensure unbiased and non-reproducible outcomes, which are essential for fairness and legal compliance.
Furthermore, TRNGs are employed in secure authentication mechanisms, such as generating one-time passwords or cryptographic tokens. They are also utilized in high-stakes scientific simulations or audits where true randomness is required to avoid bias or interference. Overall, TRNGs are preferred in environments where even a small risk of predictability or repeatability could lead to critical failures or security breaches.[1]

[1] A great example of a true random number generator used for securing internet traffic is Cloudflare's « Wall of Entropy », https://en.wikipedia.org/wiki/Lavarand, https://www.fastcompany.com/90137157/the-hardest-working-office-design-in-america-encrypts-your-data-with-lava-lamps

State of the Art

In recent years, there has been a significant push towards open-source hardware, particularly in the realm of chip design. This movement aims to democratize the process of designing integrated circuits, making it more accessible to students, researchers, and small companies. To facilitate this transformation, various tools and platforms have been developed to support this initiative.

Open-Source Design Tools

Traditionally, chip design tools, sometimes referred to as EDA tools, have been expensive and primarily used by large corporations. However, there are now several free and open-source alternatives available, including :

- OpenLane2, a full digital design flow from RTL to layout (GDS).
- Yosys, used for Verilog synthesis.
- Magic and KLayout, for viewing and editing chip layouts.

These tools make it possible to go from simple Verilog or VHDL sources to a ready-to-fabricate chip design without needing proprietary software.

Open PDKs : `sky130` and `gf180mcu`

To manufacture a chip, you require a PDK (Process Design Kit), which provides all the essential data, including design rules, component models, and simulation parameters.
Two major open-source PDKs are currently available :

• `sky130`
Developed by SkyWater in collaboration with Google, this is a 130nm CMOS process. It supports digital, analog, and mixed-signal designs. `sky130` is currently the most widely used open-source PDK.[2]

• `gf180mcu`
Provided by GlobalFoundries, also in collaboration with Google, this is a 180nm process enabling the creation of designs manufacturable at GlobalFoundries's facility on their 0.18µm 3.3V/6V MCU process technology.[3]

Other open-source PDKs exist (like IHP Open Source PDK on SG13G2 process nodes), but OpenLane2 only supports `sky130` and `gf180mcu` natively.[4]

TinyTapeout

TinyTapeout is an educational initiative that allows anyone to submit a small chip design and have it fabricated.
Users submit simple modules written using a hardware description language (HDL), and multiple projects are combined on the same chip through a Multi-Project Wafer (MPW). This approach provides an excellent opportunity to gain hands-on experience with ASIC design without incurring high costs.

---

[2] https://github.com/google/skywater-pdk

[3] https://github.com/google/gf180mcu-pdk

[4] https://openlane2.readthedocs.io/en/latest/faq.html,
https://openlane2.readthedocs.io/en/latest/usage/about_pdks.html#using-non-volare-pdks

chipflow.io and efabless

Chipflow.io[5] is a cloud-based platform that simplifies chip design. It provides tools for synthesis, layout, and project management without having to install anything locally.

Efabless[6] is a key player in the open silicon space. They support project submissions to MPWs (often funded by Google) and offer resources to validate and prepare designs for tapeout.

Caravel

Caravel is an open-source SoC platform provided by Efabless. It allows users to integrate their own design on a « user project area » situated next to a « harness »[7].
This harness includes standard interfaces (GPIO, SPI), a Wishbone bus for communication, and a small RISC-V processor for control.

With Caravel, a user block (like a random number generator) can be tested on real silicon, while taking advantage of the built-in infrastructure (clock, reset, power, I/O, processor, etc.) to test, troubleshoot, and/or validate the design.

TinyTapeout 01 through 09 used Efabless' Caravel platform to bring the user's designs to life.

[5] https://www.chipflow.io/

[6] https://efabless.com/

[7] https://ieee-cas.org/files/ieeecass/slides/unic_cass_2024_slides.pdf

Design

The random number generator chip is designed to communicate with a « master » processor, which controls the configuration of several parameters stored inside registers.
The design can be broken down into its five core components : decay pulse input, number generation, seed generation, data interface, and debug interface.
A main system clock is required to coordinate the operation of these components; in this design, a 12 MHz clock is used. This frequency is a common standard in digital systems, and low-cost 12 MHz oscillators are widely available, making the clock source straightforward to implement on a typical PCB.

Decay Pulse Input

As the random number generator relies on radioactive decay events, a dedicated pulse input is required to interface with a particle detection system. In this design, the input is intended to receive pulses corresponding to individual decay events.

Since the project focuses on the design of an integrated circuit, the supporting analog and high-voltage circuitry required to operate a Geiger-Müller (GM) tube is considered external to the IC. This external circuitry is responsible for :

- Biasing and operating the GM tube, which typically requires high voltage.
- Conditioning the output signal by shaping and filtering the raw GM tube output to produce clean, standardized digital pulses.

The radioisotope, GM tube, and circuitry were kindly provided by Manoel da Silva, who made a similar project[8] a few years ago.

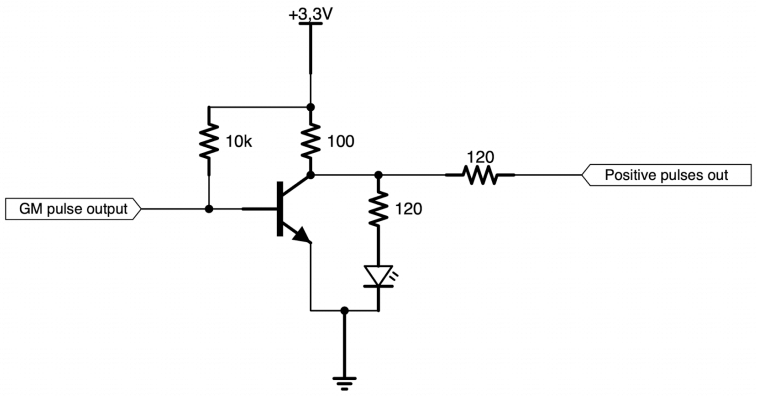


Fig. 1 : Schematic of the GM pulse signal inverter circuit

A circuit shown in figure 1 consisting of a few resistors and a transistor was added to the output to invert the GM tube's signal. The result is a train of positive logic-level pulses, each representing a detected radioactive decay particle. These pulses are then fed into the IC's digital logic, where they are used for either entropy seeding or decay-time interval sampling, depending on the selected mode of operation.

[8] https://hackaday.io/project/4628-nuclear-random-number-generator

Number Generation

The system provides two main modes of random number generation: LFSR-based generation and decay-time sampling.

LFSR-Based Generation

A 16-bit Linear Feedback Shift Register (LFSR) is implemented to generate pseudo-random numbers. The design supports four selectable LFSR feedback polynomials, allowing for flexibility in sequence characteristics and period length. The user can manually select the desired polynomial through configuration inputs.

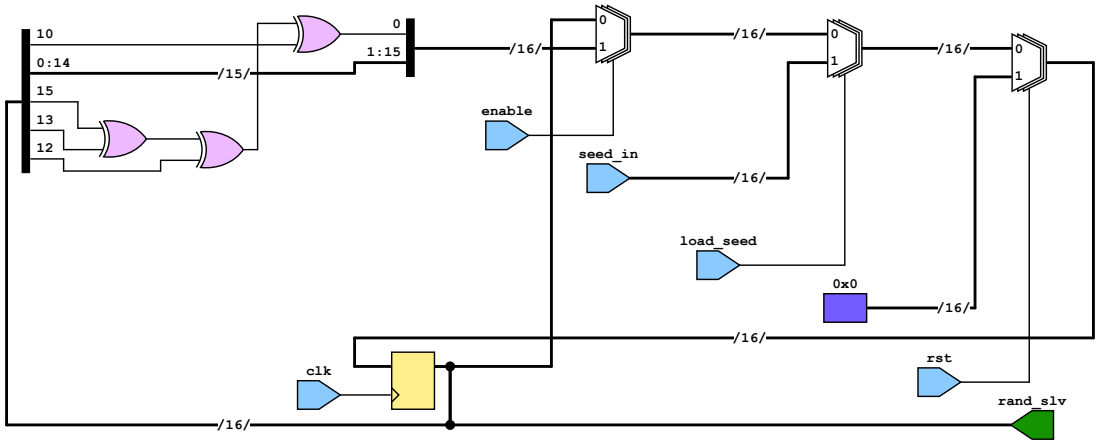


Fig. 2 : LFSR netlist diagram, with taps on bit 10, 12, 13 and 15 (polynomial n° 0 in the design)

The seed value for the LFSR can be:

- Manually configured, providing repeatable sequences for testing purposes.
- Automatically sampled from radioactive decay pulses (see Seed Generation section), introducing true entropy into the pseudo-random sequence and enabling initialization with unpredictable values.

This mode allows high-speed random number generation suitable for scenarios where throughput is critical and statistical randomness is sufficient.

Decay-Time Sampling

In this mode, randomness is derived directly from the stochastic nature of radioactive decay events. The system samples the timestamps of two successive decay pulses and computes the time interval between them. This interval is then converted to a 16-bit value, providing a true random number based on quantum processes.

This method yields lower throughput compared to the LFSR mode as it is limited by the time interval between two decay events, but provides high-entropy outputs, making it suitable for cryptographic seeding or applications where unpredictability is required.

Seed Generation

When a seed derived from radioactive decay is asked, a 16-bit seed is generated from the time between two radioactive decay pulses, much like when the RNG is in decay sampling mode. This seed is then used by the LFSR to generate its sequence.

Data Interface

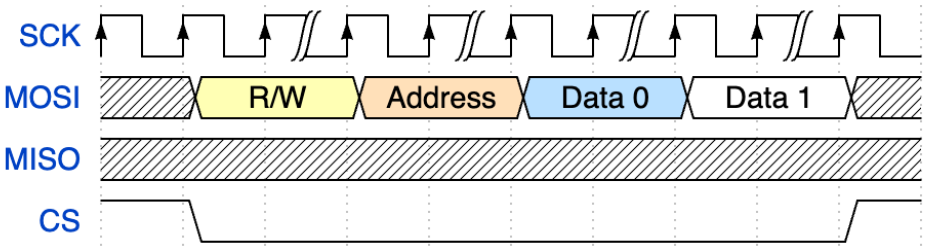
The device has a Serial Peripheral Interface (SPI) bus, which a master can use to write data to internal configuration registers, read those configuration registers back, and of course read a random number from.

Each transaction should be 32 bits long, divided into 4 bytes :
- First byte : MSB indicates whether the transaction is a write (`0`) or read (1) operation
- Second byte : register address
- Third byte : register value MSB
- Fourth byte : register value LSB

The registers are laid out as follows :

Write registers

In order to start a write transaction, a `0b0XXXXXXX` header must be transmitted. Then, the following addresses can be chosen to interact with the registers.

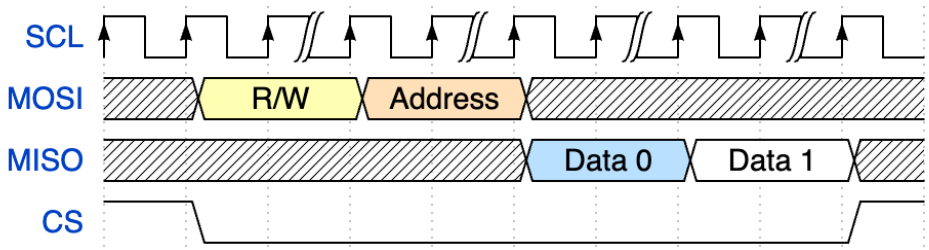


Address	Description	Data 0	Data 1	Example
0x00	Set random number generator mode	DNC	Last bit for Decay sampling/LFSR	0x00000001
0x01	Set manual seed (2 bytes unsigned) for LFSR	First part of unsigned	Second part of unsigned	0x0001BEEF
0x02	Generate seed from radioactive decay pulses for LFSR	DNC	DNC	0x00020000
0x03	Choose LFSR Polynomial (2 last bits)	DNC	Last 2 bits to choose polynomial	0x00030002

Note : MISO does not put out data during a write operation.

Read registers

In order to start a read transaction, a `0b1XXXXXXX` header must be transmitted. Then, the following addresses can be chosen to interact with the registers.



Address	Description	Data 0	Data 1	Example
0x00	Read generated number (2 bytes unsigned)	First part of unsigned	Second part of unsigned	MOSI :0x8000XXXX MISO :0xXXXXADAF
0x01	Set manual seed (2 bytes unsigned)	First part of unsigned	Second part of unsigned	MOSI :0x8001XXXX MISO :0xXXXXBEEF
0x02	Read generated seed from radioactive decay (2 bytes unsigned)	DNC	DNC	MOSI :0x8002XXXX MISO :0xXXXXCAFE
0x03	Read LFSR polynomial (2 last bits)	DNC	Last 2 bits to choose polynomial	MOSI :0x80030000 MISO :0xXXXXXX02

Note : MISO and MOSI example data signals shown synchronized.

Debug Interface

In order to test the registers, special write addresses are available to enable an RGB LED during the FPGA test phase.
Those addresses are :

- `0x10` : Red LED
  - Write : last bit of `Data 1` controls its state
  - Read : last bit of `Data 1` shows its state
- `0x12` : Green LED
  - Write : last bit of `Data 1` controls its state
  - Read : last bit of `Data 1` shows its state
- `0x14` : Blue LED
  - Write : last bit of `Data 1` controls its state
  - Read : last bit of `Data 1` shows its state

In order to test the RNG output, the 8 Least Significant Bits (LSBs) of the output number are brought out as digital signals on the FPGA development board.

Simulation

Before the design is transformed into a functional integrated circuit, rigorous tests are conducted to ensure its intended functionality. These tests include compilation, unit testing, SPICE simulation, and real-life simulation. The simulation process is often iterative, as the design is frequently modified when tests fail.
The SPICE simulation was excluded from this project since the design is entirely digital.

Compilation

This step is useful for checking for mismatched data types, syntax errors, and other issues in the source code. By using interactive code linters on an IDE, the number of errors encountered when compiling the sources can be reduced. Afterward, library documentation and AI tools such as GitHub Copilot can be used to rectify compilation errors and warnings.

Unit Testing[9]

This step checks the actual behavior of the different modules of the design, or the design as a whole.
By writing test benches that simulate SPI transactions, confirming register manipulation, clock signal cutter, edge cases, etc., it is possible to verify if the signals behave as expected inside of the design.

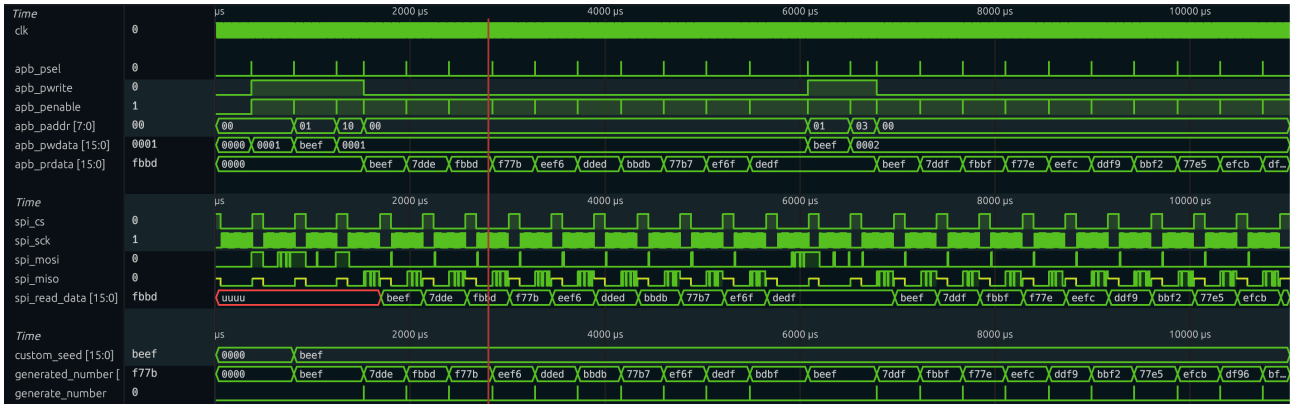


Fig. 3 : Signal data of internal registers of the design. VUnit test bench, Surfer wave visualizer

The diagram on figure 3 illustrates how writing data on the SPI bus according to the read/ write register addresses triggers an internal Advanced Peripheral Bus (APB)[10]. The APB then responds by writing or reading the internal registers that correspond to the SPI Master commands.[11]
For completeness, here are the commands depicted in the diagram (pay attention to the `apb_paddr`, `apb_pwrite`, `apb_pwdata`, and `apb_prdata` vectors, and the SPI bus) :

- `0x00000001` : Set mode to LFSR
- `0x0001BEEF` : Set custom seed to `0xBEEF`
- `0x00100001` : Turn on red LED
- `0x8000` : Read generated number (command repeated 10 times)
- `0x0001BEEF` : Set custom seed to `0xBEEF`, again
- `0x00030002` : Use LFSR polynomial n° 2 (taps on bits 1, 2, 3, and 15)
- `0x8000` : Read generated number (command repeated 10 times)

[9] https://en.wikipedia.org/wiki/Unit_testing

[10] https://en.wikipedia.org/wiki/Advanced_Microcontroller_Bus_Architecture#Advanced_Peripheral_Bus_(APB)

[11] While capturing screenshots for this report, a design issue that had gone unnoticed was identified. This highlights the importance of thorough unit testing.

FPGA

FPGA implementation is used to assess the design's synthesis feasibility, behavior during unit tests, compatibility with the geiger counter, and functionality as a SPI device. This is because the physical implementation of the design may have imperfections not accounted for in the tests.
Once the design has been tested on an FPGA, the tests can be adjusted to better reflect the real implementation. If the design works as designed and meets the requirements, it can be marked as validated for production.

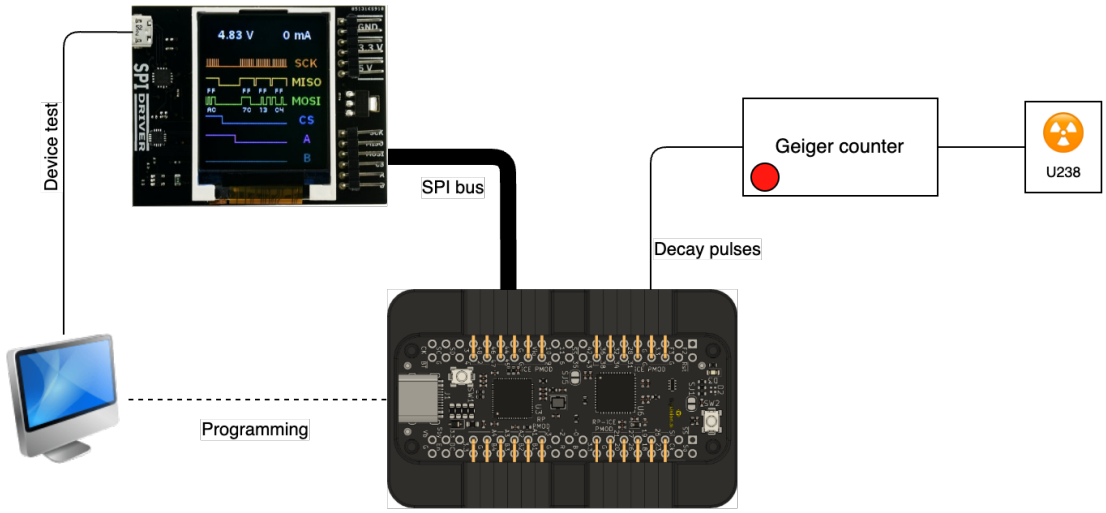


Fig. 4 : Block diagram of the test setup for the FPGA implementation

The FPGA chip used for the tests is the ICE40UP5K[12], which boasts 5.3k LUTs, 1Mb SPRAM, 120Kb DPRAM, 8 Multipliers, and 24 I/Os accessible through PMOD connectors on a pico-ice development board[13].
As shown in figure 4, the geiger counter is connected to a PMOD pin, and an SPI interface is connected to the FPGA via a PMOD connector. The SPI master interface used is an SPIDriver[14], which has a standalone app as well as a Python library to enable easy scripting. It is hooked up to a computer running a test script which sends the same commands as the main unit test bench.

```
$ python3 design_test.py
Read 10 numbers from LFSR with 0xBEEF as set starting seed and polynomial 0 :
['0xbeef', '0x7dde', '0xfbbd', '0xf77b', '0xeef6', '0xdded', '0xbbdb', '0x77b7', '0xef6f', '0xdedf']

Read 10 numbers from LFSR with 0xBEEF as set starting seed and polynomial 2 :
['0xbeef', '0x7ddf', '0xfbbf', '0xf77e', '0xeefc', '0xddf9', '0xbbf2', '0x77e5', '0xefcb', '0xdf96']

Read 10 numbers from LFSR with 0xb293 as random starting seed and polynomial 0 :
['0xb293', '0x6527', '0xca4e', '0x949d', '0x293b', '0x5277', '0xa4ef', '0x49df', '0x93be', '0x277c']

Read 10 numbers from radioactive decay sampling :
['0xefb1', '0xefb1', '0xefb1', '0x5ab1', '0x1e3f', '0x7b99', '0x5730', '0x5730', '0xb4f8', '0xb4f8']
```

Fig. 5 : Output of the test script controlling the SPI bus

We can observe in figure 5 that the output of the FPGA is the same as the unit test (figure 3) when the same starting seed and polynomial are used in LFSR mode. This confirms that the design functions as expected in the real world on an FPGA.
Ten other numbers are generated with the LFSR, but with a seed calculated from decay pulses, and ten other numbers from direct decay sampling mode.

---

[12] https://www.latticesemi.com/en/products/fpgaandcpld/ice40ultraplus

[13] https://pico-ice.tinyvision.ai/

[14] https://spidriver.com/

Hardening workflow

Once the design has been validated in simulation and by running it on an FPGA, the hardening workflow can start. This step of the project uses the `VHDLClassic`[15] workflow from OpenLane2, with the default `Sky130A` PDK.

There are a lot of default included steps in the OpenLane2 VHDLClassic flow, but put simply, the flow automates the full digital ASIC implementation process—from VHDL synthesis using Yosys, through placement, routing, and timing analysis with OpenROAD, to final checks and layout generation with Magic and KLayout.
It systematically ensures functional correctness, design rule compliance, and manufacturability, while offering checkpoints and repair steps for timing, power, and physical violations throughout the flow.

First Run

The first satisfactory (and successful) output from an OpenLane2 run looks like this :

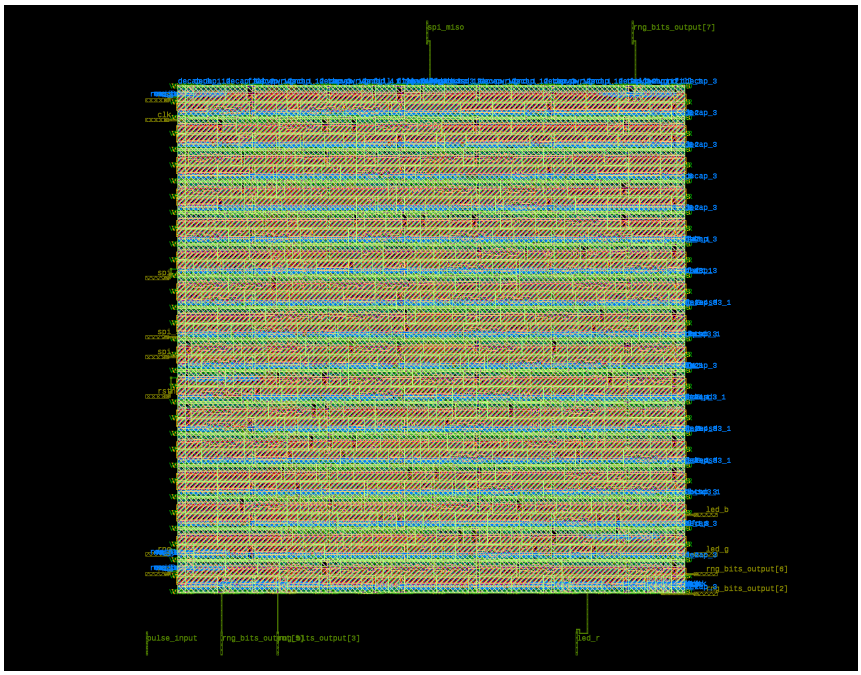


Fig. 6: First OpenLane2 run GDS output, KLayout preview

We can observe in figure 6 that the workflow successfully converted our sources into a fully functional ASIC. However, there are a few improvements we can make :

- Finalize the design : At this stage, the design lacked some of its features (for example, the `pulse_input` port is not connected to any internal circuitry).
- Specify port placement : The KLayout preview shows that the workflow placed the ports seemingly randomly.

[15] https://openlane2.readthedocs.io/en/latest/reference/flows.html#vhdlclassic

Port Placement

The port placement can be specified in a `.cfg` file, which is then integrated into the OpenLane2 `config.json` file. This allows the design's ports to be positioned within one of the four sections: North, South, East, and West. Additionally, a minimum distance between ports can be set for certain ports. Other settings can be specified, but they were not utilized in this project.

The pin configuration specified for the design is as follows :
- North :
  - `clk` and `rtsn`, with a minimum distance of 0.1 micron
- South :
  - All of rng_bits_output
- East :
  - SPI bus
- West :
  - `pulse_input` and all of the led outputs

Design Iterations

While the OpenLane2 toolchain and workflows can run locally, a continuous integration (CI) workflow has been added to the project's GitHub repo. This CI workflow allows anyone to work on the project sources, and when the test passes, run the workflow via a runner on a remote server instead of having to install all the toolchains and their dependencies on the local machine.
As of the time this report was written, the runner is self-hosted on a homelab.[16], but a modified version of the workflow could be run on GitHub's free runners[17].

When a modification is pushed to the `main` branch, the CI workflow is picked up by the runner, and a new release is created with the output files of OpenLane2's workflow.

Final Run[18]

Once the design is thoroughly validated in simulation and on FPGA, and no more design iterations are required, a final run of the workflow can be conducted.
A summary of the key metrics obtained from the final OpenLane2 digital implementation flow is provided in the appendices of this report. These metrics confirm the robustness and manufacturability of the design, with no reported timing, DRC, or LVS violations. The design demonstrates excellent power efficiency, minimal IR drop, and good clock performance across various PVT corners. This validates the functional correctness and physical soundness of the random number generator ASIC, ensuring it meets the expected criteria for silicon fabrication.

---

[16] https://fredcorp.cc/tools#homelab

[17] Initially this was the case when CI was implemented, but the runners couldn't install or compile the toolchains, leading to their refusal to run. Consequently, a self-hosted runner was adopted as the final solution.

[18] The latest design workflow output files are available on the GitHub repo : https://github.com/fred-corp/RDRNG/releases/latest

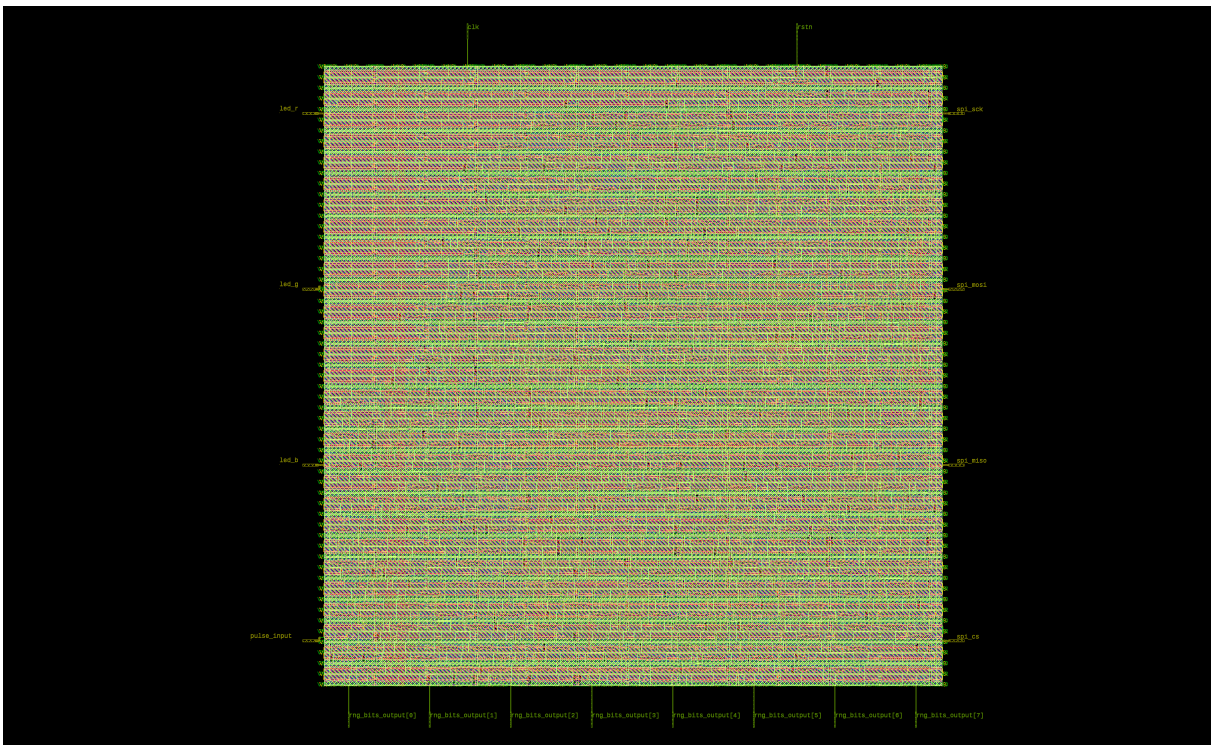


Fig. 7 : Final design render in KLayout

Figure 7 shows the silicon layout GDS file in KLayout, and figures 8 and 9 show a 3D render of the same GDS file with gds3xtrude and OpenSCAD. We can observe that the port placement was respected, as the ports were put in the correct specified section.

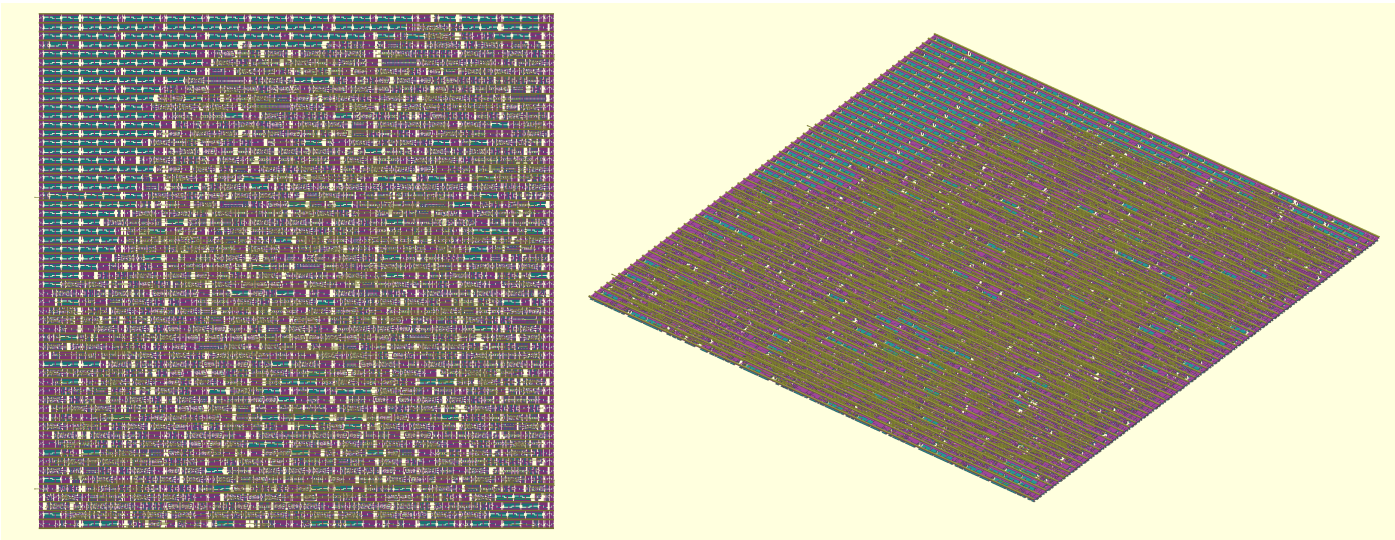


Fig. 8 : Top view of 3D render of final design run

Fig. 9 : Isometric view of 3D render of final design run

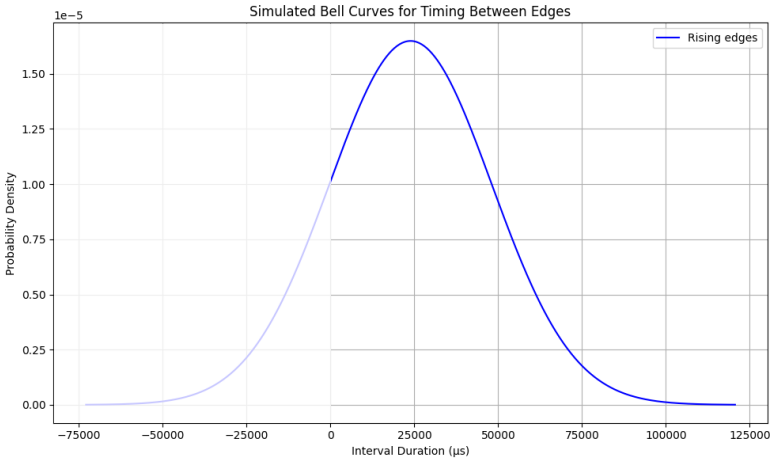
Statistical Analysis of the Radioactive Decay Source

The claim of « true randomness » generated by radioactive decay as an entropy source can be verified by recording the Geiger counter pulses for 50 seconds and analyzing the statistical values obtained from this signal.

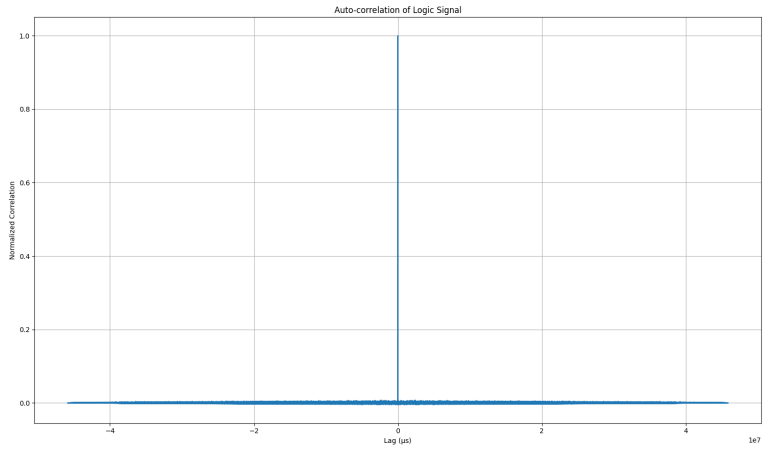
The raw values for the timing statistics between rising edges are as follows :

- Count: 1920
- Mean: 23876.95 µs
- Standard Deviation: 24196.18 µs
- Minimum: 7.0 µs
- Maximum: 183601.0 µs
- Median: 16380.0 µs

We can then generate a bell curve graph to show the spread of the pulses (figure 10).



Fix. 10 : Bell curve diagram of a 50 second recording of the geiger counter output pulses



Another analysis that can be performed is an autocorrelation analysis of the signal (figure 11).

A value of autocorrelation of 1 is observed around the 0 µs point (which is expected), and it is very close to 0 everywhere else. This strongly supports the use of radioactive decay as a reliable source of randomness.

Fig. 11 : Autocorrelation analysis of a 50 second recording of the geiger counter output pulses

The timing statistics from the 50-second recording of radioactive decay pulses show high variability and a distribution pattern consistent with exponential decay, suggesting a random source. While supportive, this alone does not confirm true randomness. Formal statistical tests are needed to rigorously validate that the pulse intervals conform to the theoretical behavior of a Poisson process.[19]

[19] This is beyond the scope of this project, but scientific papers which go more in depth about the randomness of radioactive decay are available in the sources.

Design Improvements

While the design performs effectively both in simulation and on FPGA, it can be further enhanced by incorporating additional features.
For instance, adding standard communication interfaces such as USB, I2C, and UART would enable direct and more flexible interaction with external systems, including computers and embedded controllers with other interfaces. Expanding the range of random numbers beyond the current 16-bit limit would significantly improve its capabilities. Moreover, implementing the diode noise principle[20] through a PN junction within the chip itself would eliminate the need for an external Geiger counter and radioactive isotopes, thereby reducing the project's overall size and complexity.
Additional improvements could include the development of a standardized code library for seamless interfacing, and the inclusion of a standard design reference such as a schematic to facilitate integration and reproducibility.

Conclusion

This project gave us the opportunity to explore the intersection between microelectronics and quantum randomness. By using radioactive decay as the source of entropy, we were able to build a true random number generator that goes beyond traditional pseudo-random approaches. The design combines two modes —LFSR and decay-time sampling— so it's versatile enough to suit different needs, from high-speed generation to high-entropy outputs for secure applications.
Throughout the project, we moved step by step from design to simulation, FPGA testing, and finally preparing the layout for an actual chip using open-source tools and workflows. It was especially rewarding to see our design behave consistently across all these platforms. The use of tools like OpenLane2 not only allowed us to go deep into chip design, but also showed that open-source silicon is more accessible than ever.

Overall, this project was a great learning experience, giving us hands-on insight into the full process of designing, testing, and preparing a custom design for IC fabrication.

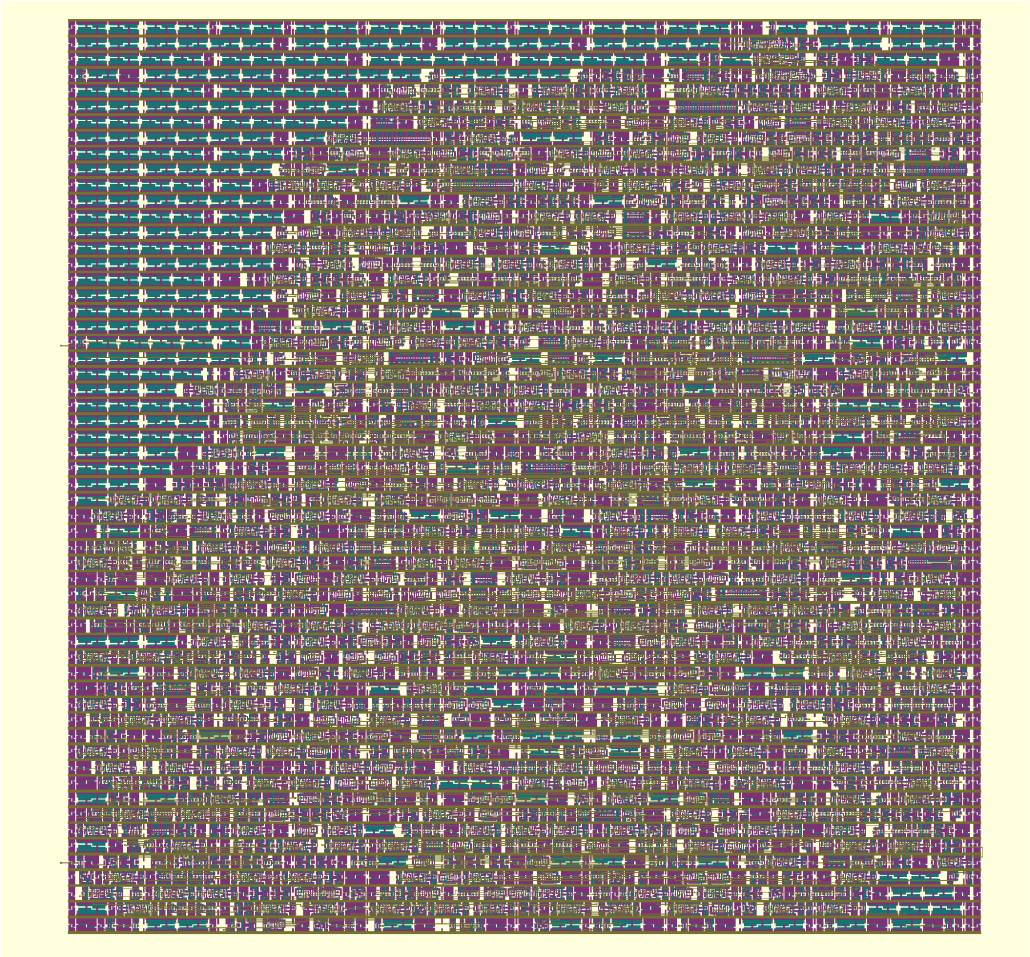
[20] https://doi.org/10.1002/ecjc.1046

Sources

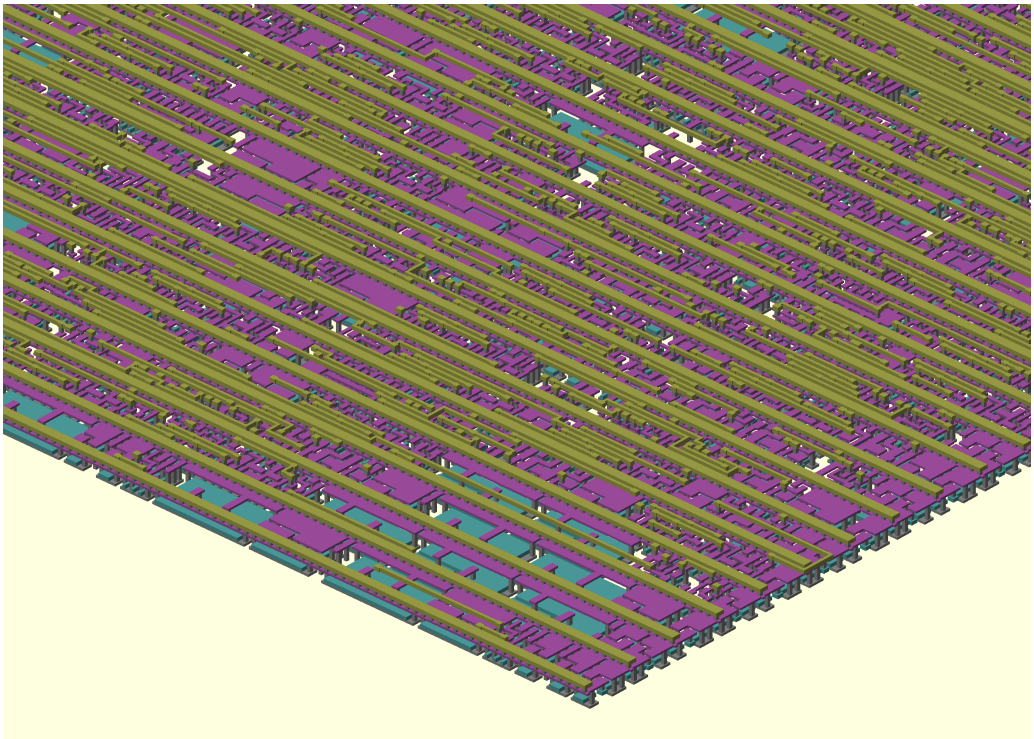
- Random number generation, Wikipedia, https://en.wikipedia.org/wiki/Random_number_generation
- Hardware random number generator, Wikipedia, https://en.wikipedia.org/wiki/Hardware_random_number_generator
- Cover page illustration created with OpenLane2, `gds3xtrude`, and OpenScad
- Apple Intelligence for syntax and grammar verification
- GitHub Copilot for code debugging
- Open Logic FPGA Standard library, https://github.com/open-logic/open-logic
- WaveDrom to create signal bus diagrams, https://wavedrom.com/
- TerosHDL, Yosys, GHDL to generate netlist diagrams from VHDL sources
- OSS CAD Suite for RTL Synthesis, https://github.com/YosysHQ/oss-cad-suite-build
- TinyVision pico-ice, https://pico-ice.tinyvision.ai/
- R. Zafar et al., « Randomness from Radiation: Evaluation and Analysis of Radiation-Based Random Number Generators », 30 September 2024, arXiv: arXiv:2409.20492. doi : 10.48550/arXiv.2409.20492.
- J. L. Anderson et G. W. Spangler, « Serial statistics. Is radioactive decay random », J. Phys. Chem., vol. 77, no. 26, p. 3114–3121, Dec. 1973, doi : 10.1021/j100644a019.
- A. Alkassar, T. Nicolay, et M. Rohe, « Obtaining True-Random Binary Numbers from a Weak Radioactive Source », in Computational Science and Its Applications – ICCSA 2005, Springer, Berlin, Heidelberg, 2005, p. 634–646. doi : 10.1007/11424826_67.

Category	Metric	Value
Design Stats	Total standard cells	1.747
	Standard cell area	15,523.6 μm²
	Die area	30,364 μm²
	Core area	24,891.4 μm²
	Utilization (standard cells)	62,4%
Power	Total power	0.0005396 mW
	– Internal	0.0003664 mW
	– Switching	0.0001732 mW
	– Leakage	0.0000194 mW
Timing	Setup worst slack (range)	63.91 – 65.53 ps
	Hold worst slack (range)	0.10 – 0.87 ps
	Setup/Hold violations	0
Clock Skew	Worst skew (setup)	0.31 ns (maximum)
	Worst skew (hold)	–0.31 ns (minimum)
DRC and LVS	Magic DRC errors	0
	KLayout DRC errors	0
	LVS errors	0
Violations	Slew violations (maximum)	3
	Fanout violations	15
	Capacitance violations	0
IR Drop	Worst drop on VPWR	0.000353 V
	Worst drop on VGND	0.000301 V
Routing	Total wirelength	30,574 μm
	Routed nets	1.366
	Vias (single-cut)	8.994
	Routing DRC errors	0
	Antenna violations	1 (handled with diode cell)

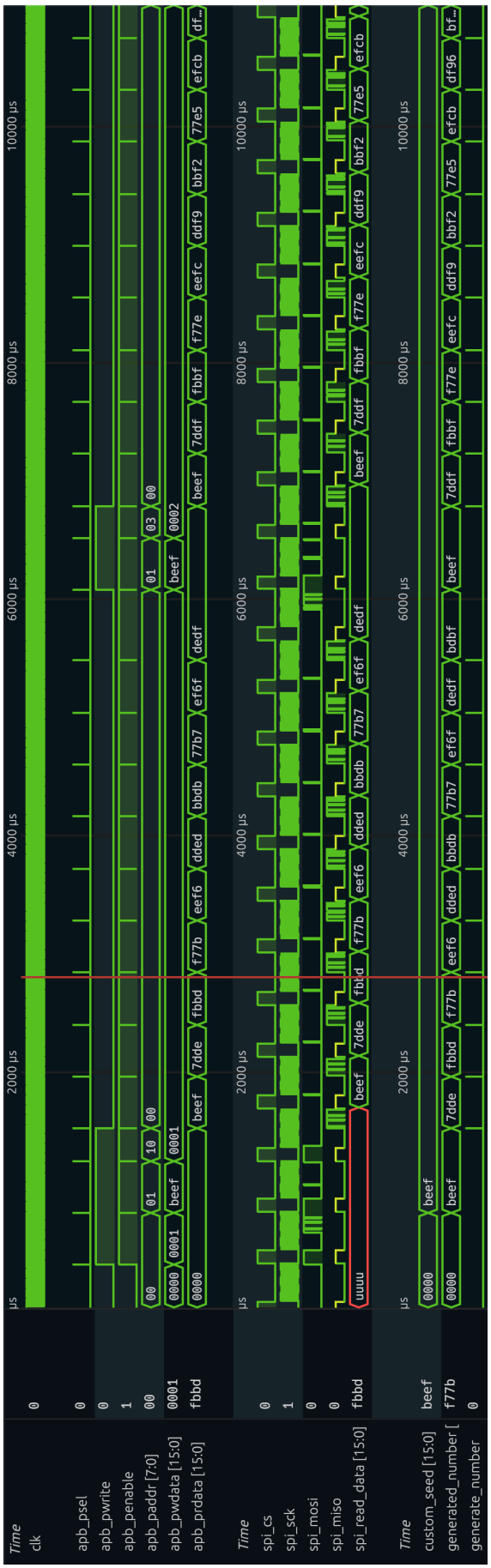
OpenLane2 Flow metrics for the final flow



Top view of 3D render of final design run



Close-up of the bottom right corner of the final run



Signal data of internal registers of the design.
VUnit test bench, Surfer wave visualizer