Emulation of Non-volatile Digital Logic for Batteryless Intermittent Computing

Simone Ruffini*, Kasim Sinan Yildirim* and Davide Brunelli†
*Department of Information Engineering and Computer Science, University of Trento, Italy
†Department of Industrial Engineering, University of Trento, Italy
simone.ruffini@studenti.unitn.it, {kasimsinan.yildirim, davide.brunelli}@unitn.it

***Abstract*—Recent engineering efforts gave rise to the emergence of devices that operate only by harvesting power from ambient energy sources, such as radiofrequency and solar energy. Due to the sporadic ambient energy sources, frequent power failures are inevitable for these devices that rely only on energy harvesting. These devices lose the values maintained in volatile hardware state elements upon a power failure. This situation leads to intermittent execution, which prevents the forward progress of computing operations. To countermeasure power failures, these devices require non-volatile memory elements, e.g., FRAM, to store the computational state. However, hardware designers can only represent volatile state elements using FPGAs in the market and current hardware description languages. As of now, there is no existing solution to fast-prototype non-volatile digital logic. This paper enables FPGA-based emulation of any custom non-volatile digital logic for intermittent computing. Therefore, our proposal can be a standard part of the current FPGA libraries provided by the vendors to design and validate future non-volatile logic designs targeting intermittent computing.**

***Index Terms*—intermittent computing, non-volatile digital logic, fpga, emulation**

I. Introduction

The recent engineering and research efforts in electronics and energy harvesting led to the rise of batteryless devices that operate only by harvesting energy from ambient energy sources. A batteryless device stores the harvested energy in an energy buffer, e.g., a small capacitor. When the capacitor's voltage level falls below a certain threshold, the device turns off due to a power failure. The device can only harvest energy and charge its capacitor until the voltage level exceeds an operating threshold, which turns on the device.

Batteryless devices lose the values maintained in their volatile hardware state elements upon power failures. Consequently, power failures lead to *intermittent* execution and computing, which prevents the forward progress of computational operations. Batteryless devices maintain non-volatile memory elements, e.g., FRAM, to save the volatile hardware across power failures. Using the latest successfully saved computational state, they can recover computation when they turn on after a power failure. Therefore, they can progress computation correctly during intermittent execution [1], [2].

Designing hardware targeting intermittent computing requires handling power failures and providing the necessary state recovery features. For instance, existing hardware accelerators (e.g., JPEG encoding/decoding) are designed to run on continuous power. If their energy requirements exceed the size of the energy storage capacitor, they will never complete their operation and provide output to the rest of the system. The microarchitecture of these accelerators should be re-designed/modified to operate them intermittently in bursts whose energy requirements are smaller than the energy storage size. Hardware designers should incorporate non-volatile memory elements into their architecture to log the hardware state at the end of each burst.

The de facto procedure to validate the functional correctness of continuously-powered hardware is prototyping via hardware description languages (HDLs) and field-programmable gate arrays (FPGAs). However, using FPGAs and HDLs, hardware designers can only represent volatile state elements. As of now, there is no existing solution to prototype non-volatile digital logic using FPGAs and HDLs. There are software-based solutions, e.g., NVPSim simulator [3], that can simulate the architectural components forming a non-volatile processor. The NVPSim users can select different configurations for the main high-level components in a processor, e.g., cache size and organization. Based on these user-selected configurations, NVPSim enables an offline exploration of the impact of these configurations on the performance of only a specific non-volatile processor.

This paper enables fast prototyping and validating any custom transiently-powered hardware composed of mixed volatile and non-volatile digital logic. Our main contribution is to introduce an emulation technique that imitates the "non-volatile" behavior of any custom non-volatile digital logic on the volatile FPGAs in the market. Our proposal can be a standard part of the current FPGA libraries provided by the vendors to design and validate non-volatile logic designs. We also present a two-dimensional discrete cosine transform design (a core component for JPEG encoders) that can work intermittently, and demonstrate its emulation to evaluate our approach. We believe that our work proposes the first attempt to introduce the fundamental building block for fast prototyping future digital logic designs targeting intermittent computing.

The article is organized as follows. Related work is reviewed in Section II. Section III presents the architecture of the Non-volatile logic emulation on FPGAs, providing details for all the blocks of the framework. Validation and analysis are discussed in a case study presented in Section IV. Section V contains the experimental evaluation on the case study, whereas Section VI concludes the paper.

978-3-9819263-6-1/DATE22/©2022 EDAA

II. Related Work

The recent literature presented many attempts to design and implement self-powered intermittent systems. As an example, non-volatile processors (NVPs) have emerged as a hardware solution in [4], [5] enabling an IoT device to resume instantly at the minimum energy intake after a power failure. However, design tools or methodologies for designing NVPs are still missing.

HomeRun [6] proposed an HW/SW co-design approach that ensures an uninterruptible program section during unpredictable energy intake. The proposed design approach is validated using a real prototype. However, the proposed design approach cannot be used to emulate any intermittent computing logic.

A comprehensive design space exploration is provided in [7], which helps to understand the implications and complex interactions of designing a complex intermittent processor system. However, their models are not implemented in any configurable simulation or emulation tool.

A design methodology to enable transient systems to reactive schemes is proposed in [8]. It is specific for task-based architectures and implemented on a well-known commercial MCU exploiting its NV-memory on board. Contrary to our FPGA-based architecture, it is not suitable to provide a generic emulation environment.

Simulation frameworks for intermittent computing are also available. Fused [9] is a full-system simulator for energy-driven computers. It is designed for energy and performance simulation and can explore new hardware and software designs. It could also simulate intermittent powered systems only at a high level of abstraction. NVPSim [3], already presented in the previous section, supports only a fully custom architecture not frequently used in IoT and embedded systems. While [10] describes a system-level simulator supporting flexible energy behavior configuration for both the processor and peripherals. With such a simulator, one may estimate performance for IoT sensor node configurations. The emulation technique presented in our paper permits the assessment of non-volatile features in any hardware (not only IoT nodes), prototype, or any digital design, even if not realized with NV-memories.

Moreover, our paper presents an emulation technique for non-volatile logic by using off-the-shelf FPGAs. Our solution permits any preliminary design component to work into pre-existing intermittent computing systems. This feature allows testing the system as a whole.

III. Non-volatile Logic Emulation on FPGAs

For debugging and functional verification purposes, we propose to imitate the behavior of a non-volatile digital logic system (e.g., a system under design) by using FPGAs. To this end, we require two core functionalities. First, we need to emulate the persistence of the non-volatile memory and the delays of the read/write operations. Second, we need to emulate the irregular power supply typical to energy-harvesting systems. Therefore, the basic entities of our FPGA emulation scheme are the non-volatile memory emulation entity (`NV_MEM`) and the intermittency emulation entity (`INT_EMU`).

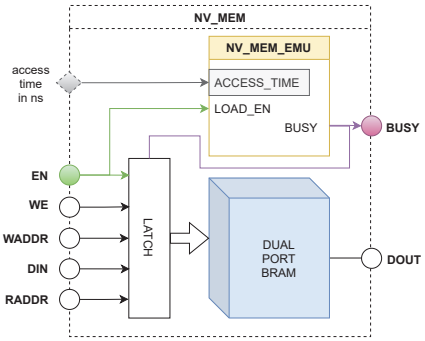


Fig. 1: `NV_MEM` entity composition: non-volatile emulation block (`NV_MEM_EMU`), simple dual-port ram and a latch on the input ports

A. Non-volatile Memory Emulation

Prototyping non-volatile digital logic circuits require the emulation of non-volatile memory using the volatile state elements of FPGAs as core functionality. To this end, we designed a memory module, named `NV_MEM`, with interchangeable RAM primitives that can emulate the read/write constraints of fast non-volatile memories (i.e., FeRAM, ReRAM). Our approach emulates the read/write latency of new non-volatile memories by leveraging the volatile memory already available on FPGA. The means used to achieve this emulation are through a latching mechanism on the input ports of the ram primitive and a signal that indicates the working status of the emulated non-volatile memory.

The main core of `NV_MEM` is the non-volatile memory emulation block (`NV_MEM_EMU`, see Fig. 1). This entity is configured with an access delay, i.e., the simulated time the memory will be unavailable to new read or write requests. When the enable (`EN`) signal is asserted `NV_MEM` synchronously accepts (at the first available rising edge of the clock) the read and write operation presented at the input ports. In Fig. 1 the selected memory primitive is a simple dual-port block RAM hence each request can be both a read and write operation. Once the request is accepted, the latch acquires the inputs and keeps them unmodified until the `NV_MEM` access time is complete. This period is observable from the outside via the `BUSY` signal. The assertion of this signal notifies the user that a request is pending, and the operation will be completed once the signal is de-asserted.

The memory primitive of `NV_MEM` can be easily changed with any memory core implementable in VHDL or synthesizable on FPGA. The changes necessary to correctly generate a new memory core are to map the inputs to the latch stage of `NV_MEM`.

B. Intermittency Emulator

This block triggers a reset signal (`RESET_EMU`) to emulate a power failure. It can generate random triggers as well as follow an energy trace of a realistic energy harvesting scenario, as presented in [11]. In the latter case, `INT_EMU` fetches voltage levels from a ROM that holds a pre-collected voltage trace. It is also possible to choose the frequency at which new voltage

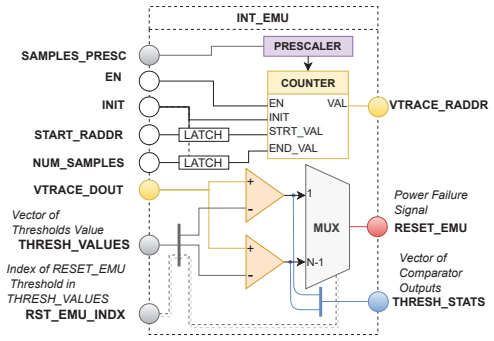


Fig. 2: The internal components and input/output signals of the `INT_EMU` block.

values are read from this memory (thanks to the pre-scaler). Each voltage level gets compared against several pre-configured threshold values using comparators. If the voltage in the trace is smaller than one of the threshold values, then the corresponding comparator asserts a signal.

One of these thresholds is used to generate the system power failure. With a multiplexer placed in front of all the comparators, the output of the desired threshold comparison operation becomes the `RESET_EMU` signal. The power failures are imitated by connecting the reset signal of the volatile entities to the `RESET_EMU` signal. Upon a power failure emulation, the expected behavior is that all sequential logic is halted and restored to its default state.

The additional comparators, which are not used for reset emulation, are instead used for energy-level monitoring. For example, if the intermittent system is intended to work between 3.3V and 2.5V, then a threshold at 2.8V can be used to inform the system that the energy level is low. The hardware then can use that information to execute specific blocks (e.g., state saving functions) or even stop if necessary.

IV. Case Study: Intermittent Two-Dimensional Discrete Cosine Transform Emulation

In this section, we present an example emulation of non-volatile logic for intermittent computing. We consider the Two-Dimensional Discrete Cosine Transform (2D DCT) since it is the key component of a JPEG encoder. 2D DCT converts pixel matrices into coefficients. JPEG encoder can later compress it to generate a JPEG image. We took an open-source 2D DCT core [12] as a baseline and provided the necessary architectural changes to construct an intermittent 2D DCT (`i-2DDCT`). The result of these modifications is a peripheral that can compute the DCT of an 8x8 pixel block, and restart the process from where it has left upon a power failure without recomputing the full block from scratch. `i-2DDCT` is one of the core building blocks necessary to construct a full JPEG encoder for intermittent systems. `i-2DDCT` computes the two-dimensional discrete cosine transform of an input matrix of size 8×8 and outputs the same size matrix of DCT coefficients.

A. Two-Dimensional DCT Definition

The one dimensional DCT transforms the input sequence $x = [x_0, \ldots, x_{N-1}]^T$ of N real numbers into another sequence

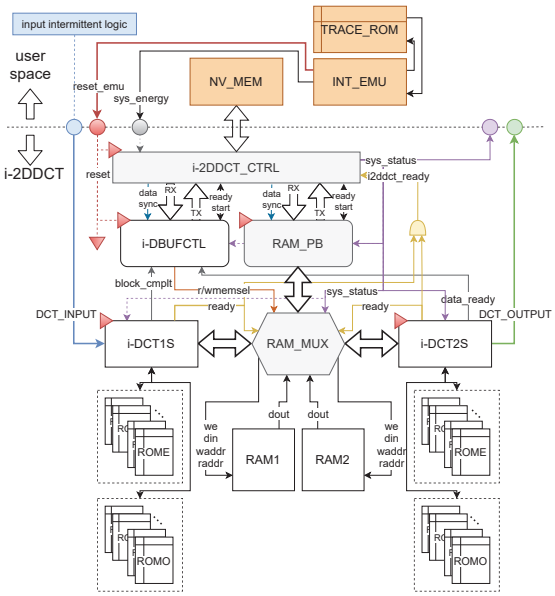


Fig. 3: Block view of `i-2DDCT` core and the components used for emulation.

of N real numbers $X = [X_0, \ldots X_{N-1}]^T$, where X_k is the coefficient of the k^{th} cosine functions with frequency $\pi k/N$. Multidimensional variants of the DCT are straightforward compositions of one-dimensional DCT's along each dimension. A two-dimensional DCT of an N-by-N input matrix X is defined as $Y = CXC^T$ where

$$C_{i,j\in[0,N-1]} = \alpha(i)\cos\left[\frac{\pi}{N}\left(j+\frac{1}{2}\right)i\right] \quad \alpha(i) = \begin{cases} \frac{1}{\sqrt{N}} & i = 0 \\ \sqrt{\frac{2}{N}} & i > 0 \end{cases}$$

The overall `i-2DDCT` architecture is depicted in Figure 3. This architectural implementation follows the row/column approach where the first 1D DCT of each row of the 8x8 input data is computed. Then, these intermediate values are transposed. Following, for each row of the transposed matrix, another 1D DCT is applied. These operations result in the 2D DCT of the original input. Multiply operations are done via Distributed Arithmetic logic, i.e. precomputed products are stored in ROMs and pushed into the MAC (Multiply And Accumulate) pipeline to calculate DCT coefficients.

B. Main Components Forming 2D DCT Core

Figure 3 presents the open source 2D DCT core components [12] in white color. `i-DCT1S` and `i-DCT2S` blocks implement the first and second stage of DCT computation. We refer to them as `i-DCTxS` since their implementations are very similar. The 2D DCT computation is processed through the help of pre-computed constant cosine values, which are stored in multiple ROM memories (`ROMO` and `ROME`).

`i-2DDCT` uses double buffering for DCT computation. Two simple dual-port RAMs (visible in Fig. 3) are switched once every time a block of DCT coefficients is completely pushed/computed by `i-DCT1S`. This event is dispatched by the

signal block_cmplt to i-DBUFCTL, which is responsible for the control of the double buffering technique. As an example, firstly i-DCT1S starts using the write ports of RAM1. After it finishes its computation, it starts to use the write ports of RAM2. Therefore, at the same time, i-DCT2S can read the 1D-DCT coefficients generated by i-DCT1S from RAM1 and generate the final 2D-DCT coefficients. This ping-pong scheme reduces downtime and enables a continuous flow of data.

C. Blocks Added to Handle Power Failures

The blocks introduced to manage power failures are depicted in gray color in Figure 3. The intermittent computation is achieved via checkpoints/backups of state-holding registers and memory. The contents of these are pushed to and pulled from the non-volatile memory (NV_MEM).

1) Intermittency Controller: The i-2DDCT_CTRL is responsible for the correct functional operation of i-2DDCT during intermittent execution. i-2DDCT_CTRL implements the main finite state machine for i-2DDCT control, the backup strategy of the system, and the complementary logic necessary for backup transfers from the volatile architecture (i-2DDCT) and non-volatile memory (NV_MEM). i-2DDCT checkpoints are composed of registers from i-DCTxS, RAM1, RAM2, and i-DBUFCTL.

2) RAM Controller: RAM_PB is responsible for the transfer of checkpoint data in/to RAM blocks. The transfer proceeds over two lines TX, for volatile to non-volatile memory, and RX, for non-volatile to volatile memory. i-2DDCT_CTRL is also involved during these and identifies the location where data checkpoint will be stored in NV_MEM.

3) Checkpoints Handler: RAM1 and RAM2 receive the DCT computation registers from i-DCTxS. Upon recovery from power failures, these blocks should be also loaded with the checkpoint contents from NV_MEM. RAM_MUX entity groups together multiplexers for RAM ports control so that RAM access is performed correctly and not simultaneously. The control signals for these multiplexers are the system status (sys_status), i-DCTxS ready signals, and r/wmemsel signals from i-DBUFCTL responsible for the double-buffer switching.

D. Finite State Machine for Intermittent Execution

i-2DDCT_CTRL finite state machine controls the execution of the whole system. It signals its status to each i-2DDCT entity and the user space with the sys_status signal. The states of i-2DDCT are as follows:

1) *Running*: the system is in normal execution mode. The 2D-DCT coefficients will be available on the output port after some delay unless there is a power failure.
2) *Halted*: i-2DDCT halts and no synchronous logic is active. Backup logic activates the transition to and exit from this state. Upon exit from this state, the core can resume computation.
3) *Pre-Checkpoint*: i-2DDCT prepares a checkpoint. i-DCTxS complete their active computation and push their state registers to the RAM. Once that is done, the system will directly proceed to push the checkpoint to non-volatile memory.
4) *Push-Checkpoint*: RAM_PB and i-DBUFCTL work together with i-2DDCT_CTRL to push a system checkpoint to NV_MEM. The role of i-2DDCT_CTRL is to correctly position data inside NV_MEM (fixed locations) and to relay the synchronization signal that abstracts the memory delays of NV_MEM.
5) *Pull-Checkpoint*: This state is simply the reversed version *Push-Checkpoint*. The direction of the transfer is reversed but the same blocks are used. The main difference is that i-2DDCT can execute this state only once and only at system startup. This is the way by which the core can restore computation after a system shutdown.
6) *Init-Checkpoint*: In this case i-2DDCT initializes itself using the checkpoint data by loading its state registers. This state affects primarily i-DCTxS since they need to recover their state from RAM. After this stage, the system goes into halt mode and waits to restart computation from the state held in the checkpoint.

E. Backup Policy

The Backup Policy of a checkpoint-driven intermittent architecture is the process by which the system decides when to store a snapshot of the system's status. The decision logic highly affects the intermittent capabilities of the system, for example, the resilience to energy fluctuations and their speed. A system that can back up its state fast but infrequently will potentially lose valuable work but has the certainty of noncorrupted backups. Hence this policy is tightly integrated with the energy model in which the system works. i-2DDCT_CTRL uses a conservative and dynamic approach to energy response. When the energy drops below a minimum operation threshold level, the system backups the state and waits until the energy level returns to a safe condition. Hence i-2DDCT_CTRL dynamically reacts to energy-level transitions and conservatively halts in hazardous energy reserves.

F. Emulation Blocks

Figure 3 presents the emulation blocks in orange color. The main blocks introduced in Section III, i.e., non-volatile memory emulation (NV_MEM) and intermittency emulator (INT_EMU) are connected to i-2DDCT. These blocks generate the reset signal as well as emulate the behavior of the non-volatile memory via the voltage traces stored in the TRACE_ROM.

V. Evaluation

In this section, we present FPGA simulations to demonstrate the intermittent execution capabilities of i-2DDCT with the help of our non-volatile emulation technique.[1] The components we used in our implementations were written in VHDL-2008 and simulated with XSIM Vivado simulator in VIVADO 2020.2 [13]. We set the core system clock to 100MHz and the NV_MEM access delay to 50ns as indicated in [14].

[1]The source code of our i-2DDCT implementation can be accessed at https://github.com/simoneruffini/i-2DDCT and the source code regarding non-volatile emulation can be accessed at https://github.com/simoneruffini/NORM .

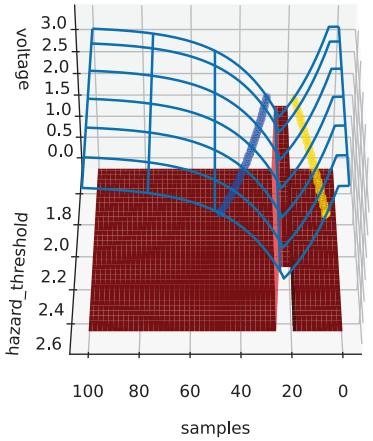


Fig. 4: This graph shows the voltage trace (wireframe), the RST_EMU assert time (red), the stop/start time of the architecture (yellow/blue). By reducing hazard_threshold value (hence the position of each yellow and blue dot), the available time to checkpoint decreases too (the distance between the yellow dot and the rising edge of the red curve).

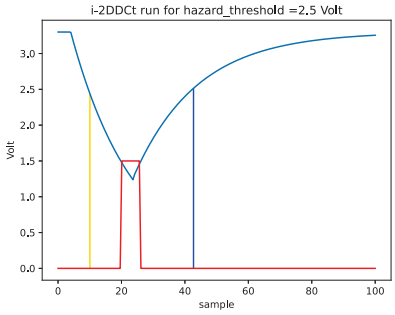


Fig. 5: The curve (blue) represents the voltage trace used for each run in the test bench where hazard_threshold is 2.5 V. The red line is the period when the system shuts down due to RST_EMU on. The yellow line represents the instant where i-2DDCT stops execution and the blue one when it restarts.

A. Continuous Execution Behavior

Under normal execution (no intermittency in the simulation), the latency between first latched input data and first DCT transformed output is 85 clock cycles. Since the design is pipelined, 64 point input data is transformed in 64 clock cycles into 2D-DCT coefficients when the pipeline is fully utilized.

B. Intermittent Execution Behavior

To simulate the transient energy-domain i-2DDCT was connected to the emulated reset signal of INT_EMU. When the voltage trace is below rst_emu_threshold, reset_emu is asserted, i-2DDCT halts and resets all sequential logic to a default state to simulate a system shutdown.

INT_EMU is configured with two thresholds: rst_emu_threshold and hazard_threshold. The first one is set to 1.5 Volts, and the second one is variable in the range of 2.7 to 1.9 Volts with 10mV steps. These levels

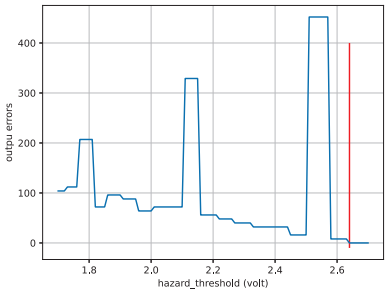


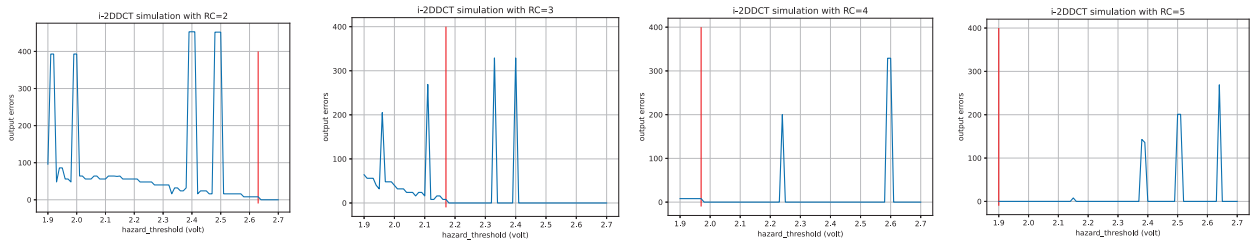
Fig. 6: The number of errors on the output of each run with different hazard_threshold. This graph presents some spikes, which are corner cases in the backup strategy where checkpoint transfers present some bugs. The red line indicates the hazard_threshold value from which the core starts to compute DCT coefficients correctly in each run (zero errors).

are chosen to simulate i-2DDCT as a system that can reliably operate above 1.9V (the lower limit of hazard_threshold) and shut down below 1.5V (rst_emu_threshold). i-2DDCT uses the output of hazard_threshold comparator to infer the system's energy level. The comparator output is input for the sys_energy line. This value is used by the i-2DDCT_CTRL backup policy. The voltage trace used in this simulation emulates a system powered by an energy harvesting device(e.g., solar panel) and a buffer capacitor.

The testbench comprises an input process that feeds a sequence of predefined values to the core, and an output process that captures the 2D DCT coefficients and verifies their correctness. Figure 4 presents multiple runs on the same voltage trace performed by changing hazard_threshold. As visible in Fig. 5 the voltage trace is divided into three zones: at first constant voltage, then a voltage drop (the capacitor discharge s), lastly the voltage returns slowly to the nominal value (capacitor recharges). When exposed to this curve i-2DDCT experiences the following computational stages:

1) A period of normal execution where the input data is converted to 1D-DCT and from 1D-DCT to 2D-DCT coefficients.
2) A system halt followed by checkpoint preparation and transfer to the non-volatile memory when the voltage trace steps below the hazard_threshold.
3) System shutdown when the voltage trace is below the rst_emu_threshold.
4) System power on and checkpoint restore once the trace is over rst_emu_threshold.
5) Normal execution from the restored checkpoint state when the voltage level is over the hazard_threshold.

Reducing the hazard_threshold value, from 2.7 to 1.9 Volts, i-2DDCT will have more time to execute in normal mode, but, as this threshold decreases, the checkpoint store time the system will have also decreases. Therefore, for some hazard_threshold values on this specific voltage trace, the core will not be able to correctly save its state upon an emulated reset. This issue is shown in Fig. 6, which presents the cu-



(a) (b) (c) (d)

Fig. 7: `i-2DDCT` simulations with different RC values for the voltage trace. The simulation with $RC = 1$ is in Fig. 6. The spikes are due to the bugs in the logic of `i-2DDCT`.

mulative number of errors for a specific `hazard_threshold` (a simulation run). An error is when the 2D DCT coefficient captured by the output process, differs concerning the correct one calculated in advance, hence the cumulative error of a run is the sum of each time the two values differ. For `hazard_threshold` below 2.64 Volts, the system is unable to compute correctly in a transient domain.

C. Voltage Trace tuning

The concept of the previous simulation was to find a `hazard_threshold` value that enables `i-2DDCT` to work reliably in a specific transient energy domain (the voltage trace). Since the voltage trace was modeled as a capacitor discharge and charge graph, the user can modify the RC value of the capacitor to change its discharge slope. The objective is to find RC values that enable `i-2DDCT` to correctly work with each `hazard_threshold` value in the range of 1.9 to 2.7 Volts. The graphs in Fig. 7 show exactly this process. Each graph is a result of the same simulation model as in Fig. 4, i.e. multiple runs with different `hazard_threshold` and a fixed voltage trace. However, we ran the testbench four times with different RC values. Fig. 7a shows that the input gets correctly computed in DCT coefficients for `hazard_threshold` between 2.7 and 2.64 Volts, this is the same as with $RC = 1$ in Fig. 6. Instead, for $RC = 3, 4$ the limit decreases and the range extends, respectively: $[2.18, 2.7]$V and $[1.97, 2.7]$V. At last, an always correct execution is achieved for $RC = 5$ since no errors are generated during each run.

The spikes visible in these graphs are due to an edge case in the intermittent behavior of `i-2DDCT`. A bug occurs when the system resets without processing the input pixel-block completely. This situation leads to an incorrect restart of the processing when the core restarts execution after an emulated reset. Hence the output process captures incorrect coefficients. This situation is visible in figures since the spikes are in the same order as the input data (512 pixels). With the help of our emulation strategy, we were able to observe these bugs without the realization of the whole system, which can be costly and time-consuming. Our emulation method allowed us to verify if the modifications to the 2D DCT core (for power failure handling and intermittency handling) work correctly. Thanks to our proposal, such cases can be observed, functionally verified, and further bugs can be removed.

VI. Conclusion

We introduced an FPGA-based emulation technique to design and validate non-volatile digital logic that is powered transiently. Using this technique, we presented the emulation of a two-dimensional discrete cosine transform design that can work intermittently. We believe that our technique will be the fundamental building block for fast prototyping non-volatile logic designs.

Acknowledgement

This work was supported by the program "Dipartimenti di Eccellenza (2018-2022)" of the Italian Ministry for University and Research (MUR).

References

[1] A. Rodriguez Arreola *et al.*, "Approaches to transient computing for energy harvesting systems: A quantitative evaluation," ser. ENSsys '15, New York, NY, USA, 2015, p. 3–8.

[2] D. Balsamo *et al.*, "Graceful performance modulation for power-neutral transient computing systems," *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 738–749, 2016.

[3] Y. Gu *et al.*, "Nvpsim: A simulator for architecture explorations of nonvolatile processors," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 147–152.

[4] Y. Liu *et al.*, "Ambient energy harvesting nonvolatile processors: From circuit to system," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[5] K. Ma *et al.*, "Nonvolatile processor architectures: Efficient, reliable progress with unstable power," *IEEE Micro*, vol. 36 (3), pp. 72–83, 2016.

[6] C.-K. Kang *et al.*, "Homerun: Hw/sw co-design for program atomicity on self-powered intermittent systems," in *Proceedings of the International Symposium on Low Power Electronics and Design*, ser. ISLPED '18, New York, NY, USA, 2018.

[7] J. San Miguel *et al.*, "The eh model: Early design space exploration of intermittent processor architectures," in *2018 51st Annual IEEE/ACM Int. Symposium on Microarchitecture (MICRO)*, 2018, pp. 600–612.

[8] D. Balsamo *et al.*, "A control flow for transiently powered energy harvesting sensor systems," *IEEE Sensors Journal*, vol. 20, no. 18, pp. 10 687–10 695, 2020.

[9] S. T. Sliper *et al.*, "Fused: Closed-loop performance and energy simulation of embedded systems," in *2020 IEEE Int. Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 263–272.

[10] T. Wu *et al.*, "An extensible system simulator for intermittently-powered multiple-peripheral iot devices," in *Proceedings of the 6th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, ser. ENSsys '18, New York, NY, USA, 2018, p. 1–6.

[11] J. Hester *et al.*, "Ekho: realistic and repeatable experimentation for tiny energy-harvesting sensors," in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, 2014, pp. 330–331.

[12] K. M. Bergmann Andreas, Yuce Emrah, "Discrete cosine transform core," https://opencores.org/projects/mdct, last update Apr 10, 2017.

[13] Xilinx, "Vivado Design Suite," https://www.xilinx.com/products/design-tools/vivado.html, Jan. 2021, last accessed: Jan. 31, 2021.

[14] Cypress Semiconductor Corp., "F-ram™ technology brief," https://www.cypress.com/file/46186/download, June 2016.