FPGA Fingerprint Tarot Fortune Reader using Hénon Map PRNG

Miranda Chen   
Singapore University of Technology and Design  
Singapore  
m48chen@uwaterloo.ca

Anthony Sun Yi  
Singapore University of Technology and Design  
Singapore  
1004944@mymail.sutd.edu.sgKeith Lee Zhengxian   
Singapore University of Technology and Design  
Singapore  
1005994@mymail.sutd.edu.sg

Leow Yew Rei   
Singapore University of Technology and Design  
Singapore  
yewrei\_leow@mymail.sutd.edu.sg

Tian Wyndham  
Singapore University of Technology and Design

Singapore  
1006320@mymail.sutd.edu.sg

*Verilog Code at:*

[*https://github.com/m-chenie/chaotic-tarot-reader*](https://github.com/m-chenie/chaotic-tarot-reader)

*Explanation and Demonstration video at:*

[*https://www.youtube.com/watch?v=RqVWFEaKl3I*](https://www.youtube.com/watch?v=RqVWFEaKl3I)

Keywords—pseudo random number generator, FPGA, Verilog, Hénon map, tarot card

# **Introduction & Summary**

In this project, we present a novel fusion of biometric security, hardware-based randomness, and symbolic visualization through tarot. The system integrates an R307 fingerprint sensor, an FPGA running a PRNG or chaos theory logic, and a Raspberry Pi 4 Model B powering an LCD display that reveals tarot cards based on user-specific entropy.

The core of this system lies in biometric authentication via fingerprint input. Traditional biometric implementations are often vulnerable due to their software-based architecture. Our approach enhances security by leveraging an FPGA to process fingerprint data in hardware. Sensitive data never leaves the FPGA unencrypted—instead, we apply SHA-256 hashing to fingerprint templates, storing only cryptographic hashes in BRAM. This ensures tamper resistance, eliminates software attack surfaces, and mitigates threats such as replay attacks or fingerprint reconstruction.

Once authenticated, the FPGA utilizes the entropy derived from the fingerprint—either through hashed templates, noise, or timing variations—as a seed for a pseudo-random number generator (PRNG) or a chaos-based system. The generated output is a randomized index corresponding to a tarot card from a standard 78-card deck. This index is transmitted to the Raspberry Pi, which maps it to a symbolic tarot representation.

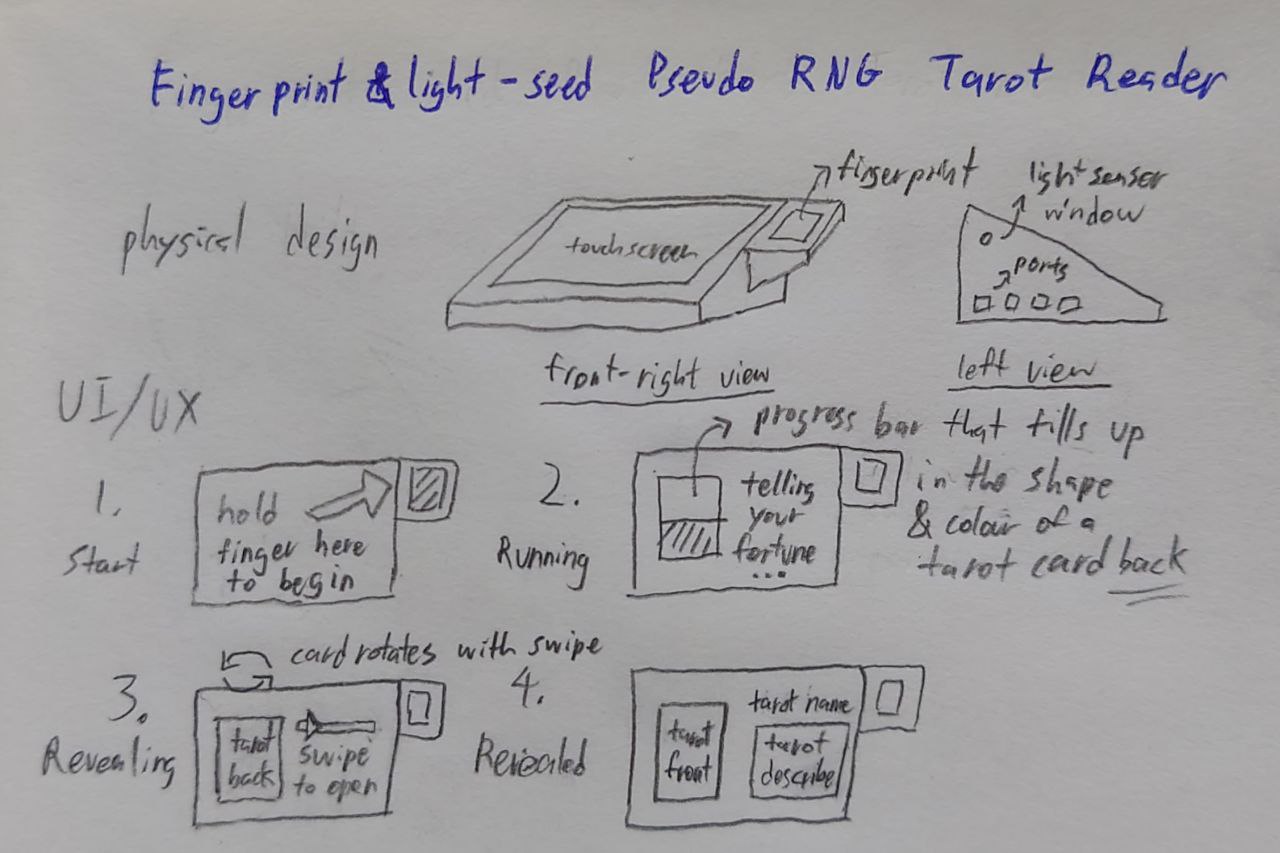
On the Raspberry Pi, a graphical interface displays the selected tarot card and a dynamically generated narrative. This visual output not only adds a layer of user engagement but also exemplifies the fusion of hard security with expressive symbolism. The experience becomes personal, unpredictable, and secure—each interaction unique to the user’s fingerprint.

By combining secure biometrics, hardware-accelerated cryptography, and mystical storytelling, this project explores the boundary between digital security and human meaning, making it suitable for applications in personalized authentication, interactive installations, or secure entertainment interfaces.

# **Custom Enclosure CAD Design for FPGA-Integrated Biometric System**

For the Overarching Design Rationale, we made the concept sketch (Fig 1) with user-friendliness in mind. To do this, we had a discussion and decided that the most time- and effort-effective method was to make our design completely touch-based. By using a touchscreen and fingerprint scanner, we can avoid the complexity and reliability issues with designing and incorporating moving parts. While the touchscreen allows us to easily design and update artistic flair and instructions to optimize user enjoyment and ease-of-use, without needing to manually paint or write every update. And instead focus on spending our time in getting the PRNG code to work flawlessly, which is the main purpose of this project.

As we continued to refine our design, we designed it in CAD to be made with 3D printing, as that would allow us the most customizability for our aesthetic design, and would allow us to make custom crevices to hold all the components inside (Fig 1.5, 2) while providing easy wiring and accessibility for troubleshooting with our actual prototype (Fig 2.1). The BOM is in Table 1.

Fig 1: Design Concept Sketch

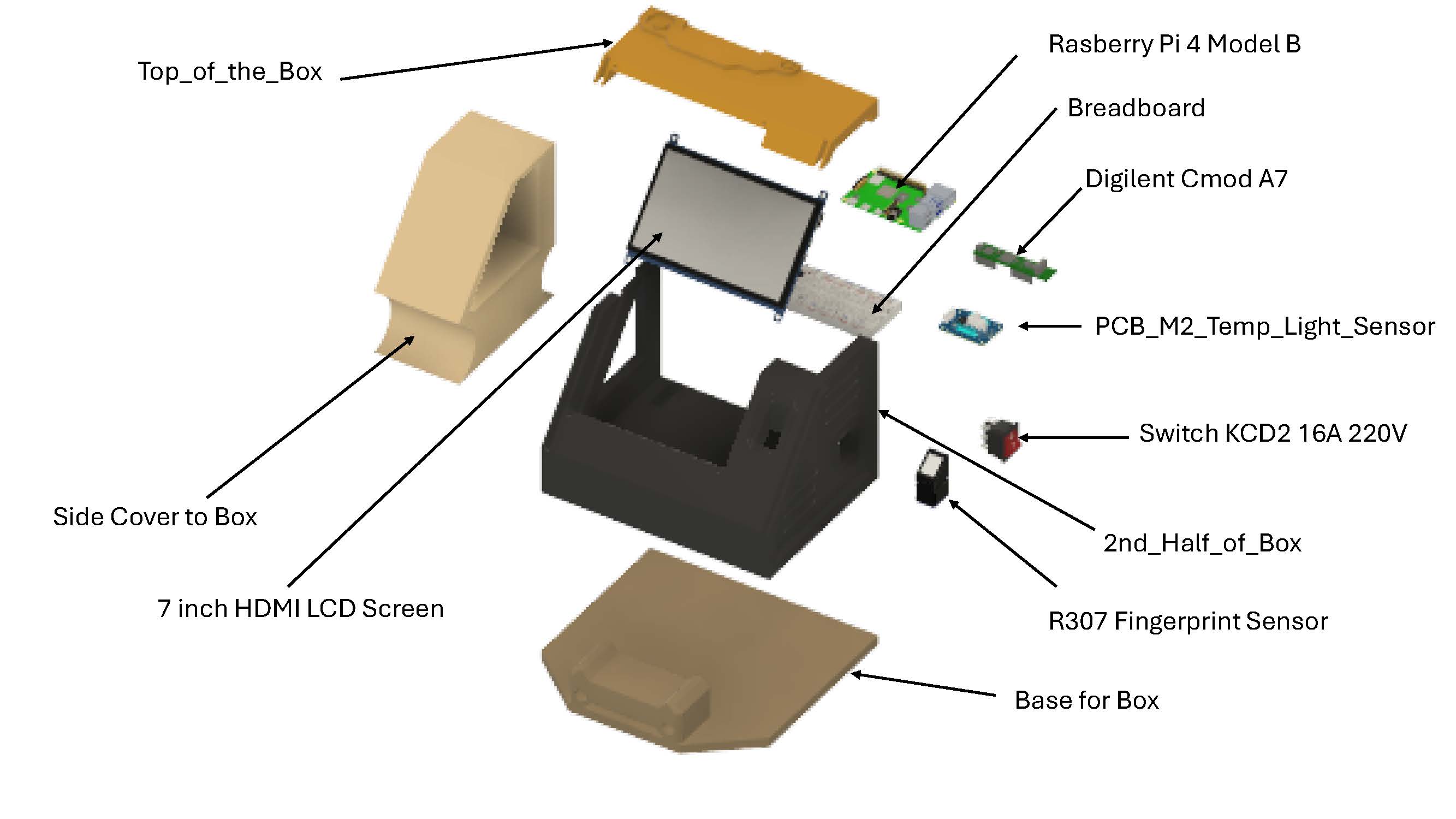


Fig 1.5: Exploded View of Fingerprint Tarot Card Reader box



Fig 2: Assembled View of Fingerprint Tarot Card Reader box



Fig 2.1: Photo of Actual Prototype of Tarot Box

|  |  |
| --- | --- |
| **Component** | **Details** |
| Raspberry Pi 4 Model B | Main processing unit |
| Breadboard | For prototyping connections |
| Digilent Cmod A7 | FPGA development board |
| PCB\_M2\_Temp\_Light\_Sensor | Custom PCB for sensors |
| Switch KCD2 16A 220V | Power switch |
| R307 Fingerprint Sensor | Biometric input sensor |
| 7 inch HDMI LCD Screen | Display interface |
| Top\_of\_the\_Box | 3D printed enclosure part |
| Side Cover to Box | 3D printed side panel |
| 2nd\_Half\_of\_Box | Lower main chassis structure |
| Base for Box | Bottom support structure |

Table 1: Bill of Materials

# **Pseudo Random Number Generator Algorithm**

Chaotic maps can be used to build pseudo random number generators and can be used in cryptographic algorithms [1]. The pseudo random number generator in this project was built around the two-dimensional Hénon map, a 2D dissipative quadratic map governed by:

,

(1)

Where the constants α = 1.4 and β = 0.3 allows the Hénon map to be its classical chaotic regime and produce a “boomerang” attractor that never settles to a short orbit [2].

With only one quadratic term and single linear feedback, the Hénon map requires just two multiplications and one addition per iteration. Thus, implementation of hardware would be relatively inexpensive. In addition, since x and y feed into each other at every iteration, the entropy injected into either variable, which in this case is the fingerprint-image mean for y0 and the light-sensor seed for x0, is quickly blurred across the full 64-bit state. One dimensional map such as the logistic or tent functions cannot provide the same cross-coupling and are more prone to short numerical cycles when quantized [3]. Since many implementations of Hénon maps have been used in encryption schemes, it holds promise in being used as PRNG in this project [1] [2].

The validity of the PRNG was assessed off-chip with a Python test-bench. Using a Python implementation of the 2D Hénon map, the Hénon map was validated as a PRNG. Since only the least significant 24 bits of xn was used to produce the 3 tarot cards, a histogram was generated using the least significant 24 bits. As seen below in Figure 1, the least significant 24 bits produce a uniform distribution and show no significant pattern.

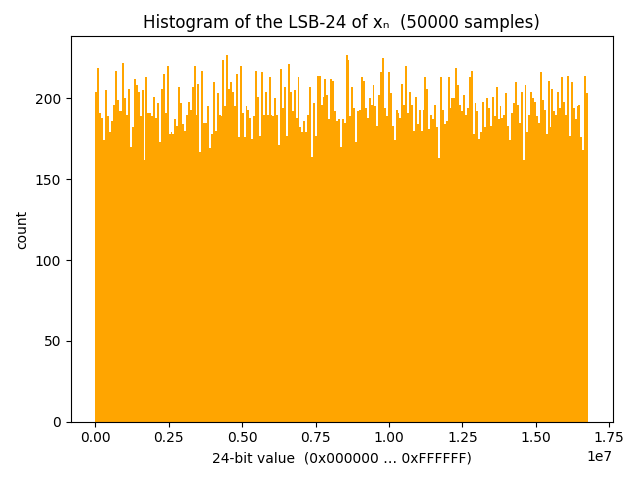


Fig 3: Histogram of LSB-24 of xn

In addition, the temporal behaviour of the Hénon map shows no visible periodicity or drift, as seen in Figure 2.

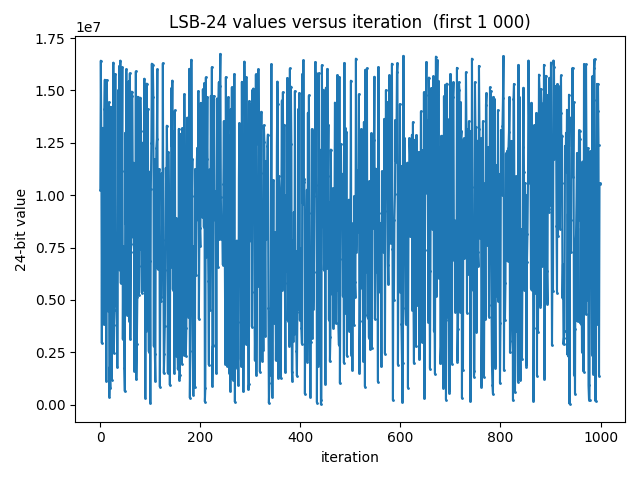


Fig 4: LSB-24 values strip-chart

# **PRNG implementation in FPGA**

To implement the PRNG, a CMOD A7 FPGA was used along with an Arduino Uno, R307 fingerprint module and a MCP3202 ADC reading data from a temperature/light sensor. Using the MCP3202 ADC and the png from the R307 as seeds, we can feed it into the Hénon map code and run it 8 times to receive our random number.

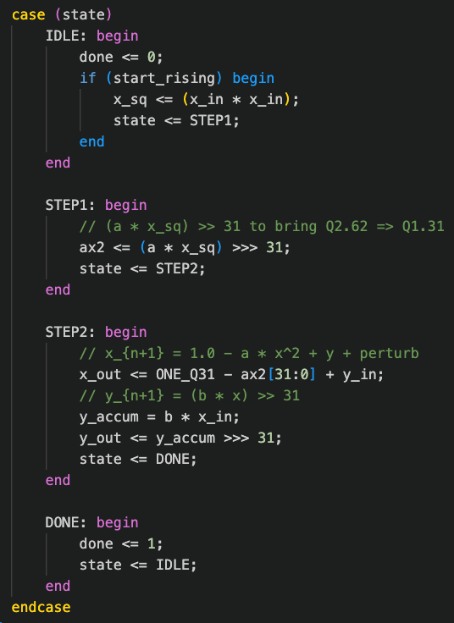
To embed the Hénon map in our design the continuous equations of (1) were quantized to Q1.31 fixed-point—one sign bit and 31 fractional bits. A single DSP slice on the Artix-7 computes the two required multiplications; an adder/subtracter and two 32-bit registers hold the current state (xn, yn). The Hénon map was iterated 16 times. The design is split into a one-step iterator (henon\_map\_q31) and a lightweight controller/driver (henon\_prng\_top). 

Fig 5: One-step Hénon computation from henon\_map\_q31.v

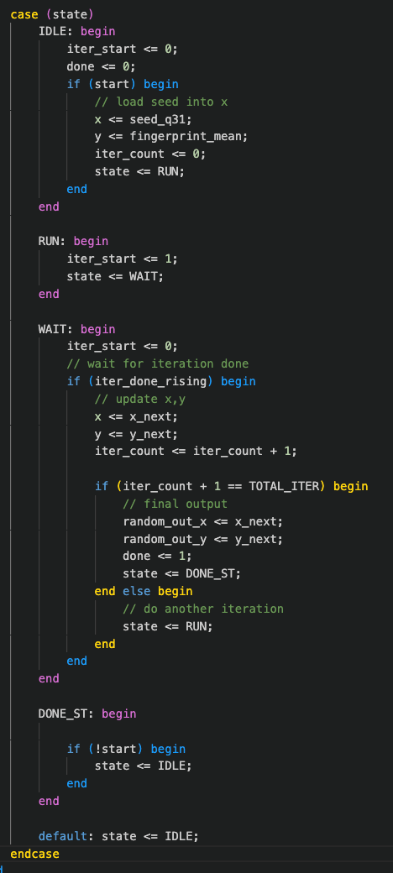
Inside the one-step iterator, a finite-state machine (FSM) is implemented, as seen in Figure 3. The 4 states correspond to the mathematical pipeline required for a single Hénon iteration. Once, the FSM reaches DONE, the handshake flag (done <= 1) is raised for one cycle and the FSM returns to IDLE, ready to accept the next start pulse.

Fig 6: henon\_prng\_top driver for henon\_map\_q31

Where Figure 3 zoomed in on the three-clock micro-pipeline that computes a single (xn+1, yn+1) pair, Figure 4 (henon\_prng\_top) moves up one abstraction level and shows how that pipeline is orchestrated over several iterations to produce a final random word for the system. The controller begins in IDLE, where a rising edge on the system-level start line loads the two sensor seeds and resets an iteration counter.

It then alternates between RUN (asserts a one-clock iter\_start pulse to trigger the datapath of Fig. 3) and WAIT (sits idle until the inner core asserts iter\_done).

After the requested number of repetitions—eight in our prototype—the machine captures the last (x, y) pair, raises the top-level done flag, and parks in DONE\_ST until the host releases start.

# **Operation of FPGA**

Using the block diagram below, we can begin to understand the basic Power and Signal system.

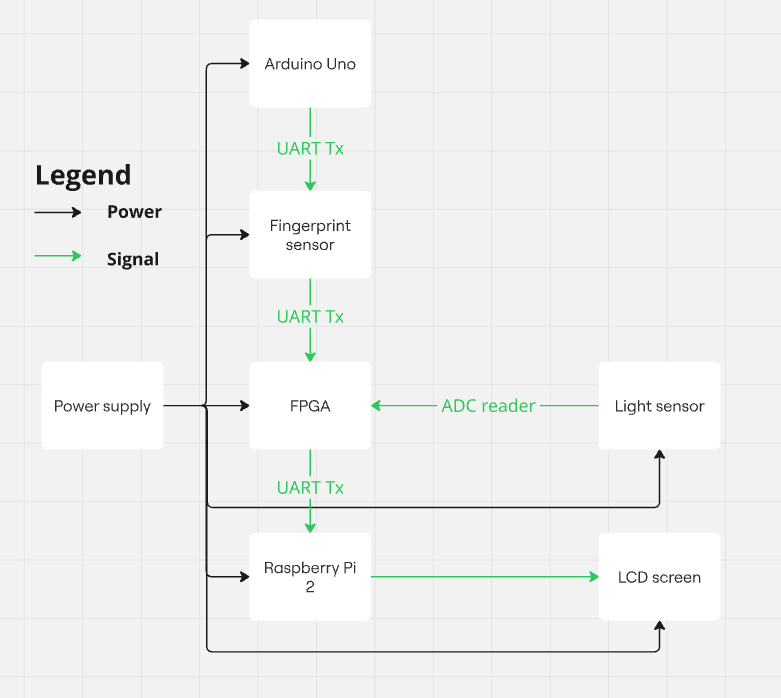


Fig 7: Power and Signal block diagram

Using the Uno, we are able to send the command to allow for the fingerprint to be captured. This prompts the R307 to capture an image and send the data to the FPGA through UART. This along with the data coming from the MCP3202 ADC, refreshing every second, provides us with the 2 starting seeds. The image data will need to be processed as there are 29952 bytes being sent to the FPGA. As such, this data needs to be processed to create a 32-bit number.

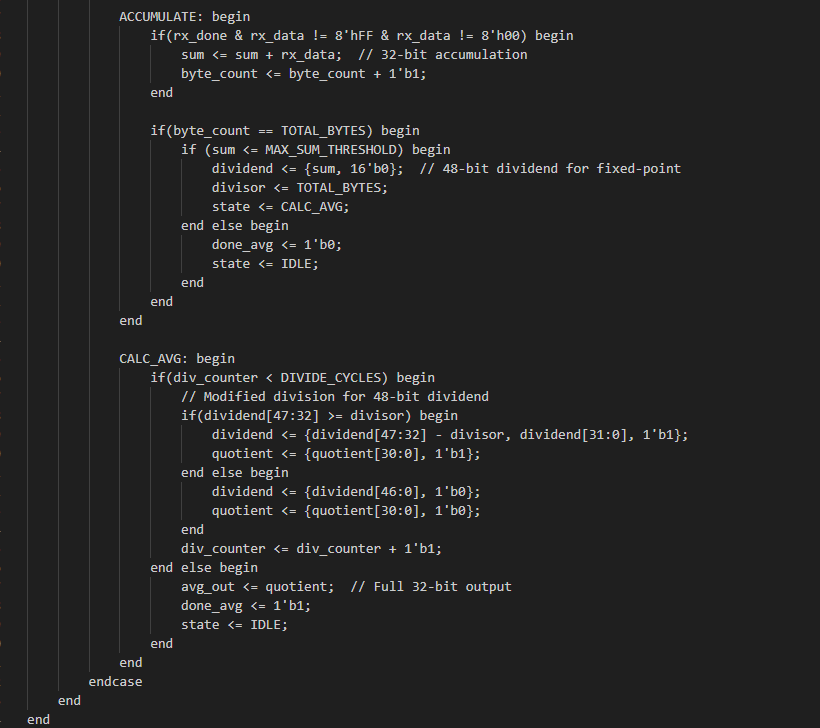


Fig 8: Processing Accumilate and Calcu avg state

Using the Processing module, we are able to receive the data, using the UART Rx module. We then proceed to sum the bytes together, ignoring the FF and 00 bytes which represent total darkness and total brightness. This data is stored till we reach an adjustable byte count, at which point we will divide it in cycles to achieve a 32-bit value to be used in the Hénon map.

As both the ADC and the Processing module have state machines that show when they are done, we used that to start the Hénon map module when both seeds are present.

Once data is confirmed, we place the data in a buffer where it can be checked to see if we have sent the same data previously, as seen in the figure below. It then sends the last 24 bits through the UART.

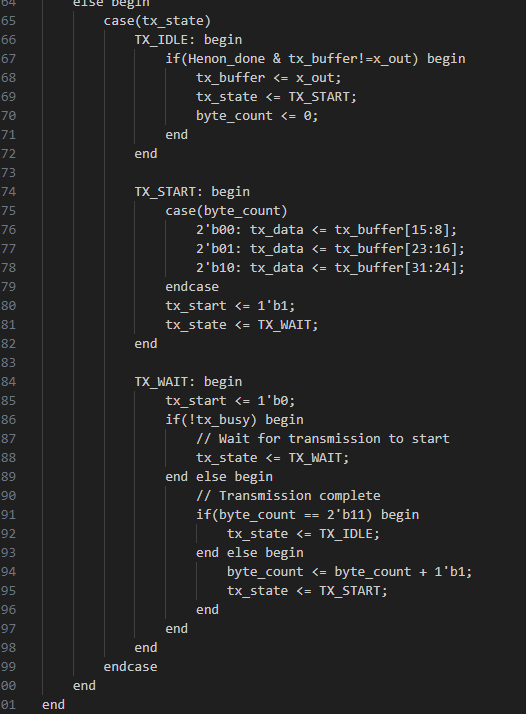


Fig 9: UART Tx to Rasberry Pi

The data is then sent in 3 separate 8-bits transmission to the Raspberry Pi to read and use for the Tarot card reading.

For all the communications, we used UART. However, as the components are adjusted to different baud rates, we ensured that the UART modules had a parameter that could be changed to allow for easier swapping of modules.

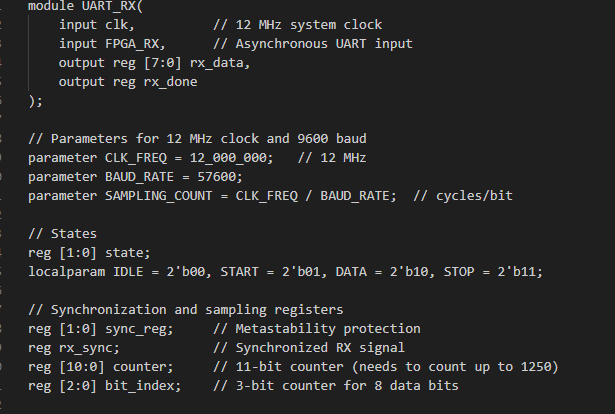


Fig 10: UART Rx parameters

# **Tarot card interpretation and implementation**

This digital tarot reading system uses a complete 78-card tarot deck, just like a traditional deck you’d hold in your hands. The deck splits into two parts: the major arcana (22 cards) and minor arcana (56 cards). The major arcana, like The Fool (beginnings), The Magician (skill), and The World (completion), focus on big life themes and spiritual lessons. The minor arcana, divided into four suits—Wands (action), Cups (emotions), Swords (thoughts), and Pentacles (material life)—cover everyday experiences. Each suit has Ace to Ten cards plus court cards (Page, Knight, Queen, King), offering insights into daily challenges, relationships, or personal growth. Together, these 78 cards provide a rich mix of grand and grounded perspectives.

1. *How the Cards Are Interpreted: Keywords to Story*

When three cards are drawn—one for past, one for present, one for future—the system uses keywords to capture each card’s core meaning. For example, The Fool might have “beginnings,” “spontaneity,” and “faith,” while the Ace of Cups has “love,” “new feelings,” and “overflow.” These keywords are woven into full sentences that reflect the card’s position in time, turning simple ideas into a flowing, personal story that connects the cards to your life.

1. *The Structure of the Reading: A Poetic Guide*

The reading follows a clear, mystical structure. It opens with a poetic line like, “The cosmic veil parts, revealing whispers of your path,” setting a reflective mood. Then, each card is presented with its position and a sentence blending its keywords. For the past, you might read, “The shadows of your past reflect the essence of ‘beginnings’ and ‘spontaneity’”; for the present, “The currents whisper of ‘manifestation’ and ‘skill’”; and for the future, “The horizon shimmers with ‘completion’ and ‘accomplishment.’” It closes with a thoughtful nudge, like, “Ponder these reflections from the loom of fate. The threads are shown, but the weaving is yours,” empowering you to shape your path.

1. *Why It Feels Like Tarot*

This system feels like a real tarot reading because it uses the full deck’s traditional meanings, randomly selects cards, and crafts a narrative that’s both poetic and open-ended. The English text mirrors the storytelling of a tarot reader, guiding you through past, present, and future with words that invite reflection and personal connection.

# **Rasberry Pi implentation**

The Raspberry Pi runs a Python script, which is the core backend logic for the tarot card reading user interface (UI). The system obtains serial data numbered from 1 to 78 from the FPGA and maps it to a specific tarot card. It then displays the card, Figure 11, along with pre-defined keywords tied to each card and the card image in a graphical interface seen in Figure 12. It bridges physical interaction (e.g., biometric sensors and light sensors) with an interpretive and visual experience.



Fig 11: “The Fool” example of a Tarot Card Image

Fig 12: User Interface showcasing both cards and text

## Key Libraries and Components

In the Python script, key libraries were used. tk*inter* was used for building the user interface (GUI). *PIL* *(Image, ImageTK)* supports the use of image loading and converting it for display in the GUI. Serial facilitates communication with hardware devices via the raspberry Pi’s UART interface. *glob, os, random, time:* Handle file searching, randomness, and timing functions.

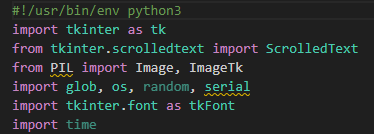
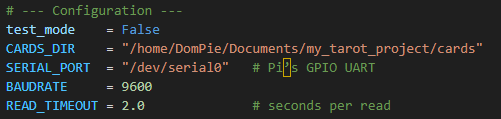


Fig 13: Imported libraries

## Configuration Parameters

Within the Python code, *CARDS\_DIR:* Specifies the folder path where tarot card images are stored. *SERIAL\_PORT:* Defines the serial communication port on the Pi (e.g., /dev/serial0). *BAUDRATE* and *READ\_TIMEOUT:* Define the speed and timeout for serial data reads and *test\_mode:* A toggle for bypassing hardware communication during testing.

Fig 14: Initialization of parameters

## Tarot Deck Initialization

The script builds a complete tarot deck including both Major Arcana (22 symbolic cards) and Minor Arcana (56 cards divided into 4 suits: Wands, Cups, Swords, Pentacles). Minor Arcana is generated programmatically using a list comprehension.

## Associated Keyword Mapping

Each card, particularly the Major Arcana, is associated with a set of unique keywords. These keywords serve to interpret the meaning of the card when drawn. This enhances the user experience by helping them understand the symbolism and message behind the chosen card.

## UI Design (tkinter-based)

A GUI is created using tkinter, incorporating:

A ScrolledText widget for displaying keywords. An Image display area for showing the chosen tarot card’s image. Font customization for readability. Cards are drawn from the deck and dynamically displayed along with associated keywords and images.

## Serial Data Input

The script receives data from the FPGA connected to the UART-sensor, i.e. a fingerprint scanner and light sensor. This input can be used as a seed or trigger for drawing a tarot card, adding a sense of randomness influenced by physical input.

## Image and Keyword Selection

The draw\_reading() function selects the tarot card from the deck that matches the serial data input. The corresponding card imagesnd keywords are displayed in the GUI and ensure that the images are scaled appropriately, seen in figure 10.

# **Arduino implementaion**

While the Arduino Uno is currently used to provide the start command for the Fingerprint sensor, the FPGA could potentially send the same signal, as the UART modules used are the same. However, for simplicity, we used the more reliable Arduino Uno to also ensure that the FPGA is not processing too much information.

# **Full system operation**

The complete system integrates multiple hardware components and software modules to create a seamless and secure biometric-driven tarot card experience. Each module - ranging from entropy collection to tarot visualization - operates in tandem, forming a cohesive pipeline that transforms a user’s fingerprint and environmental input into a personalized symbolic narrative.

## Intuitive UI/UX from User’s operation perspective

The User is prompted to simply place 1 on the fingerprint scanner and use another to press the “Begin Tarot” button on the touchscreen. Then the cards are drawn and displayed together with their narrative story. This requires no complicated thought or strategy on the User’s end and hence h/she can seamlessly enjoy the Tarot Reading experience.

## User Interaction and Entropy Generation

The system initiates user interaction via the R307 fingerprint sensor, triggered by a command from the Arduino Uno. Upon successful image capture, the raw fingerprint image is transmitted via UART to the FPGA.

Simultaneously, the MCP3202 ADC samples ambient light or temperature data, which is also sent to the FPGA. These two sensor values - fingerprint and analog sensor readings - form the seeds for the pseudo-random number generation process.

1. Fingerprint Seed: The image data is processed on the FPGA to discard outlier bytes (e.g., 0x00, 0xFF) and calculate an average brightness value, creating a 32-bit numeric seed.
2. Environmental Seed: The ADC data is scaled and converted to another 32-bit number. These two values collectively seed the Hénon map.

## FPGA PRNG Computation

The FPGA, specifically the CMOD A7, uses a quantized fixed-point implementation of the two-dimensional Hénon map to produce chaotic outputs. The FPGA's finite state machine iterates the map 16 times using the collected seeds to sufficiently diffuse the entropy. The result is a pseudo-random 32-bit value, where the least significant 24 bits are extracted and mapped to an index within the standard 78-card tarot deck.

The internal FSM modules handle synchronization between seed arrival, PRNG iteration, and output readiness.

The FPGA ensures that no duplicate data is used by maintaining a short-term memory buffer.

## Data Transmission to Raspberry Pi

Once the tarot index is generated, the FPGA transmits the 24-bit result to the Raspberry Pi via UART in 3 8-bit chunks. The UART modules on both devices are baud-rate-configurable, ensuring compatibility despite different hardware constraints.

## Tarot Card Display and Interpretation

On the Raspberry Pi 4, a Python script listens to the serial port for incoming tarot indices. Upon receipt:

The index is matched to a card in the tarot deck, including both Major and Minor Arcana. Associated keywords and symbolic meaning are retrieved and the tkinter-based GUI dynamically updates to show the chosen card's image and its interpretive text.

The result is a personalised tarot reading that feels meaningful, secure, and interactive—one that is influenced by both the unique biometric fingerprint and momentary environmental context.

# **Innovation and education hightlights**

## Technological Innovation

Utilizing an FPGA to implement a pseudo-random number generator (PRNG) based on the Hénon map introduces a novel approach to generating entropy, enhancing the unpredictability and security of the system. The fusion of chaotic mathematical models with tarot card symbolism creates a unique user experience, blending deterministic algorithms with interpretive art.   
  
At the heart of the Chaotic Tarot Reader, it is a fundamentally novel approach to biometric authentication.

While most fingerprint-based systems rely on software processing - leaving them exposed to memory dumps, reverse engineering, and replay attacks - our design offloads the entire fingerprint pipeline onto an FPGA via On-Chip Fingerprint Processing Where Raw image data from the R307 sensor is streamed directly into FPGA logic. A FPGA-resident SHA-256 engine hashes the fingerprint template in real time, with encrypted Storage in BRAM so that only the 256-bit hash ever resides in on-chip Block RAM (BRAM). As such, no unencrypted fingerprint data or templates ever cross the FPGA boundary.

The hardware implementation also eliminates OS - or firmware-level attack surfaces and cryptographic hashing prevents fingerprint reconstruction. Replay attacks are neutralized because each template is non-invertible and bound to the current session.

By processing, hashing, and storing biometric data wholly within the FPGA fabric, the system ensures end-to-end security at the hardware level - an industry-leading step beyond traditional software-based biometrics.

## Educational Value

The project encompasses various fields - electronics, computer science, mathematics, and art—providing a holistic educational tool that encourages learners to explore multiple disciplines.

Students gain practical experience with hardware components like FPGAs, ADCs, and microcontrollers, as well as software development for GUI applications, fostering a comprehensive understanding of system integration.

Having to interpret the output of chaotic systems and relating them to symbolic representations like tarot cards encourages creative thinking and offers a platform for discussions on determinism, randomness, and interpretation.

## Broader Implications

This project exemplifies how complex systems can be made accessible and engaging, serving as a model for educational tools that aim to demystify advanced concepts through interactive and personalized experiences.

# **Challenges, Lessons Learnt, Solutions**

## Software – Data Stream Syncronisation

A significant challenge in the software design stemmed from the need to synchronize two distinct data streams that operated at different frequencies from. the Fingerprint Sensor and CMOD 7A. This timing discrepancy created a critical issue where the Hénon map component either received an excessive number of starting signals in rapid succession or, conversely, failed to receive any signals at all. Such inconsistency disrupted the intended functionality of the pseudorandom number generator (PRNG).

To address this, we carefuly observed and tuned comunication protocol by measuring the actual output with an oscilloscope. , a precision diagnostic tool that enables the visualization of electrical signals in real time.

## Software – Data Transmission Integrity

Another notable issue arose when certain bits failed to transmit or receive correctly between system components. This problem jeopardized the integrity of the data flow, which is critical for the accurate generation of pseudorandom numbers. To diagnose the root cause, we reutilized the oscilloscope. By analyzing the waveforms, we were able to verify the correctness of the hardware connections and confirm whether the signals being sent adhered to the expected patterns.

This meticulous examination revealed inconsistencies in output signal, which we promptly rectified by ensuring secure connections and proper grounding troublemaking wires. The use of the oscilloscope was instrumental in identifying these discrepancies, allowing us to restore reliable data transmission and safeguard the system’s overall performance.

## Hardware - Fingerprint Sensor Overheating

During prolonged stress testing and troubleshooting sessions exceeding one hour, we encountered an undocumented error code, 0xFF, from the fingerprint sensor. This anomaly prompted a detailed investigation into its cause. Leveraging a thermal camera, we monitored the sensor’s temperature and discovered that its internal circuitry was overheating under sustained operational loads. The thermal imaging confirmed a direct correlation between elevated temperatures and the occurrence of the error code.

To mitigate this issue, we instituted a protocol whereby the system is powered down for a cooling period of 5 to 10 minutes following intensive use. This straightforward yet effective measure has proven successful in preventing overheating, thereby ensuring the fingerprint sensor’s consistent and reliable performance. Additionally, we explored passive cooling solutions, such as improved ventilation around the sensor, to further enhance its thermal management in future iterations.

## Hardware – 3D Printing Unavailability

A significant hurdle in our hardware development process emerged when we discovered that the university’s shared 3D printing facilities were fully booked for the next three weeks (due to other school projects running simultaneously like Capstone, 3D-Prog, Architecture, etc). This posed a critical challenge, as fabricating a physical prototype was essential for validating our design and ensuring proper integration of components. To address this, we took proactive measures by purchasing the required filament materials and arranging to borrow a personal 3D printer from a friend. This resourceful solution enabled us to proceed with the prototyping phase without significant delays, demonstrating our ability to adapt to scheduling constraints and maintain project progress. The successful production of the prototype reaffirmed the importance of flexibility and external collaboration in overcoming logistical obstacles.

## Hardware – Display Frying

We encountered another hardware setback when the original touchscreen display malfunctioned and became inoperable. To diagnose the issue, we employed an oscilloscope, which revealed that an overvoltage condition in one of the connecting wires was the root cause. After correcting the voltage issue and ensuring proper wiring integrity, we replaced the damaged screen with a borrowed unit from a friend. However, the replacement screen had slightly different dimensions compared to the original, requiring us to manually adjust the existing enclosure. Through careful filing and the strategic use of duct tape, we modified the case to accommodate the new screen, ensuring a secure fit and full functionality. This process highlighted our technical troubleshooting skills and adaptability, allowing us to navigate unexpected hardware challenges effectively while keeping the project on track.

##### References

[1] D. Murillo-Escobar, M. Á. Murillo-Escobar, C. Cruz-Hernández *et al.*, “Pseudorandom number generator based on novel 2D Hénon-Sine hyperchaotic map with microcontroller implementation,” *Nonlinear Dynamics*, vol. 111, pp. 6773–6789, Apr. 2023, doi: 10.1007/s11071-022-08101-2.

[2] A. Soleymani, M. J. Nordin, and E. Sundararajan, “A chaotic cryptosystem for images based on Hénon and Arnold cat map,” The Scientific World Journal, vol. 2014, Art. no. 536930, Aug. 2014, doi: 10.1155/2014/536930.

[3] R. Lozi, “Complexity leads to randomness in chaotic systems,” in Complexity, Metastability and Nonextensivity, S. Abe, H. J. Herrmann, P. Quarati, A. Rapisarda, and C. Tsallis, Eds. Singapore: World Scientific, 2011, pp. 79–98, doi: 10.1142/9789814338820\_0005.