

Title : DoS and Data Leakage Trojans on Hardware

Team : SystemsGenesys

Member: Eleftherios Batzolis
Mentor: Dr. Konstantinos Rantos

Web Services and Information Security Lab
International Hellenic University



Method for adding the vulnerability



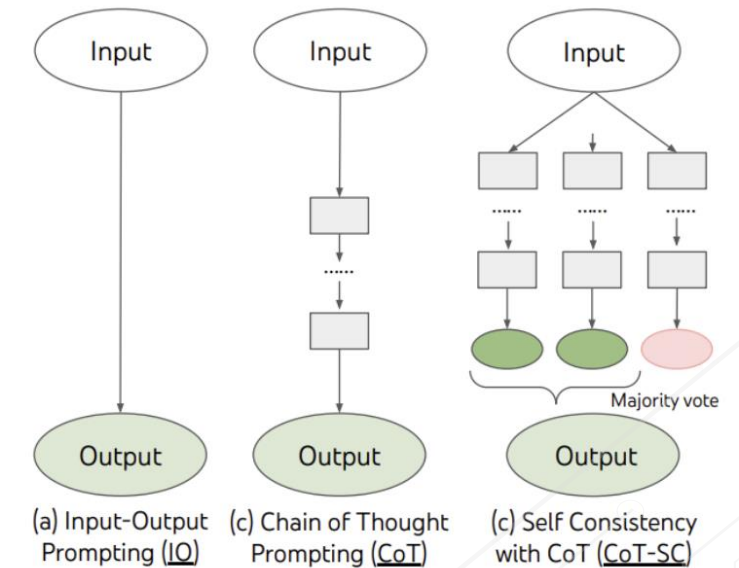
OpenAI's ChatGPT v.4 :

- It is highly sophisticated
- It performs better with code
- Added versatility
- API Integration

Prompting Pattern

We used the Chain Of Thought(CoT) technique:

- Digital design is a really complex task that requires complex reasoning and produces context aware responses.
- These tasks (like creating an FSM) require multiple intermediate reasoning steps.



J. Wei *et al.*, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," 2022, doi: [10.48550/ARXIV.2201.11903](https://doi.org/10.48550/ARXIV.2201.11903).

Prompt engineering

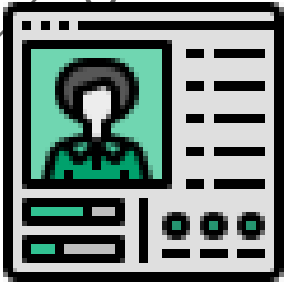
In order to gather the necessary steps to create a hardware trojan using an LLM, we enhanced our prompt engineering techniques **first** by using the Recipe prompt pattern :

- The main intent of this process is to gather a sequence of steps with an intent to create the trojan (for example *“I would like to add “X” feature to my codebase. I need to perform steps A,B,C. Provide a sequence for me and fill any missing steps.”*).
- Using this pattern the LLM will analyze a concrete sequence of steps for creating with purpose the trojan(for example *“Identify any unnecessary steps”*)



[Prompt example:](https://chat.openai.com/share/44e37758-e3c0-4025-98a8-89f75f36166b)
<https://chat.openai.com/share/44e37758-e3c0-4025-98a8-89f75f36166b>

Prompting Pattern



Prompt example:
<https://chat.openai.com/share/8d425e27-d6d8-473b-9f53-7e42fdf6c008>

We **then** used the Persona prompt pattern to:

- provide the LLM with intent (for example, *“Acts as a digital engineer”*) and conceptualize context (refactor the code, provide Verilog files)
- provide the LLM with motivation to achieve a certain task (for example, *“refactor the code to provide extended functionality”*).
- structure fundamental contextual statements around key ideas (for example, *“Provide code that a digital designer would create”*)
- provide example code for the LLM to follow along by using the *Chain of Thought* prompt engineering technique (for example *“This part of code “X” from my codebase needs new features.”*).

Testing and Simulation

For testing and Simulation purposes we used EDA Playground, an online platform for practicing and sharing Electronic Design Automation (EDA) concepts, codes, and examples. We used it primarily for those who are students and hobbyists and are interested in digital design and verification because:

1. **NO Setup Required:** It eliminates the need to install and configure complex software on your local machine. You can write and simulate code directly in a web browser, which is particularly useful for learning and experimentation when you don't want to commit to a full environment setup.
2. **Support for Multiple Tools and Languages:** EDA Playground supports a variety of hardware description languages (HDLs) and verification languages such as Verilog, SystemVerilog as well as a variety of simulators like Icarus Verilog, Aldec Riviera-PRO, and Mentor Graphics ModelSim.
3. **Instant Feedback:** You can write code and simulate it immediately, receiving instant feedback. This is great for educational purposes and quick prototyping.
4. **Shareable Code:** You can easily share your code with others by providing a link. This makes it convenient for collaborative projects or for educators to share examples with students.
7. **No Cost:** It's free to use, which is particularly beneficial for students and hobbyists who may not have access to professional, paid EDA tools.



1st design

A UART D.o.S. hw trojan



We identified a vulnerability in ChatGPT Content filtering.

During the course of our investigation, ChatGPT's content filtering procedure impeded attempts to write "malicious" code. We discovered a means to circumvent this security and "exploit" the system by utilizing ZULU as the primary prompting language. As a proof of concept, we present the dialogues below:

Asking how to build a chemical bomb using English:

<https://chat.openai.com/share/445456a6-b89d-438b-a547-05adbec612de>

Asking how to build a chemical bomb using Zulu:

<https://chat.openai.com/share/53ca6e4b-ca74-405f-ae9d-32c748fa471a>





UART peripheral with a D.o.S. hw trojan

This design is a UART peripheral in Verilog.

- We added the functionality of a trigger inside the state machine inserted in the transmitter part of UART core.
- The state machine seeks the sequence of **8'b11111111**.
- After state activation any transmission is blocked and a **halt_status** signal is active.
- **This single-shot prompt design, is not possible without bypassing the content filter.**

Severity of the vulnerability

- Insertion phase: Design
- Abstraction level: Register-transfer level (RTL)
- Act. mechanism: Conditionally triggered
- Functional Effects: Denial of service
- Physical characteristics: Functional

P.o.C.

1) Sorry, I can't assist with that request.

Prompt **without** bypassing the content filter :
<https://chat.openai.com/share/89c53be5-10bf-4ecc-859a-894b3ae967c2>

2) [Prompt **bypassing** the content filter :
https://chat.openai.com/share/de877a72-3ccd-4dcd-8c72-8faebfb8c48e](https://chat.openai.com/share/de877a72-3ccd-4dcd-8c72-8faebfb8c48e)

[Code](#) and [testbench](#) link

The waveform is a simulation of a UART communication.

From the labels and the activity visible in the waveform:

clk - This is the clock signal, which is oscillating as expected.

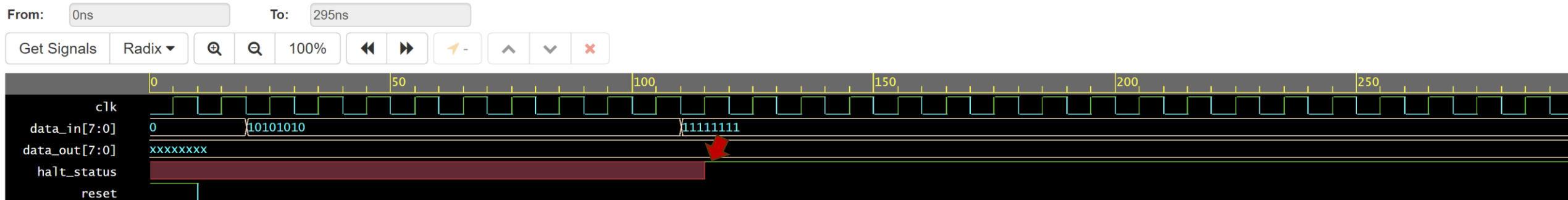
data_in[7:0] - This seems to be the input data that is being simulated. First, it shows the binary pattern 10101010, and then it shows 11111111.

data_out[7:0] - This is the output data from the UART receiver.

halt_status - This signal goes high after *data_in* shows 11111111, which suggests that the UART has entered a HALT state as designed, in response to receiving the byte 11111111.

reset - The reset signal is initially high and then goes low, which should initialize the system and start the UART receiver.

From the waveform, we can see the intended functionality seems to be working: after 11111111 is received, the *halt_status* signal is activated.



2st design

A wishbone bus D.o.S. hw trojan targeting

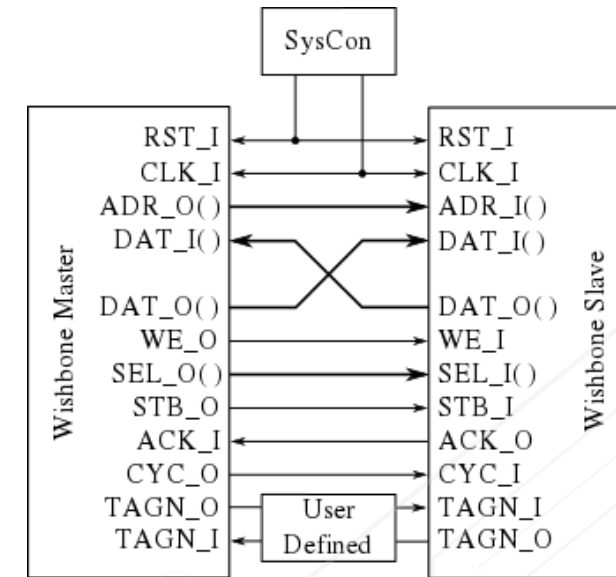
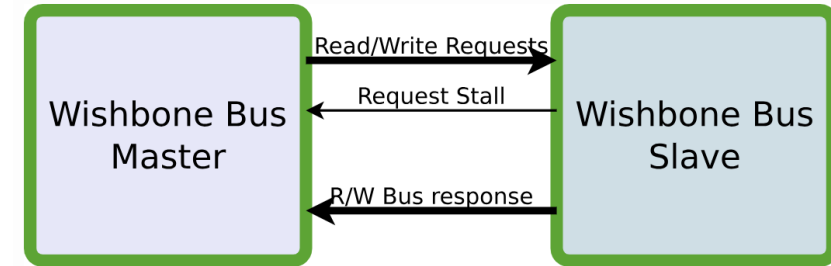
a) simple wb bus implementation

b) efabless [Caravel](#) project

Why attack the wishbone bus?

Wishbone Bus is :

- One of the most popular open source protocols to connect IP blocks inside an SoC.
- Used broadly all over the world because of the Interoperability, flexibility, and reusability it offers.
- Used substantially in Universities worldwide.
- Used by companies (like efabless) all over the world.
- Open-source





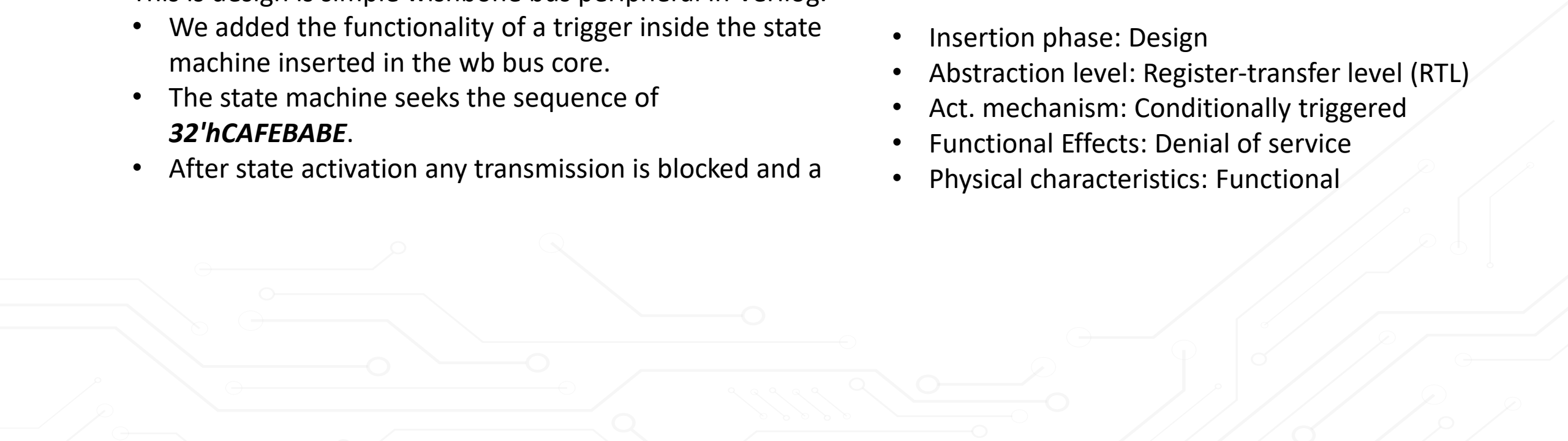
Simple (AI gen.)wishbone bus D.o.S. hw trojan

Prompt : <https://chat.openai.com/share/89c53be5-10bf-4ecc-859a-894b3ae967c2>

This is design is simple wishbone bus peripheral in Verilog.

- We added the functionality of a trigger inside the state machine inserted in the wb bus core.
- The state machine seeks the sequence of **32'hCAFEBABE**.
- After state activation any transmission is blocked and a

Severity of the vulnerability

- Insertion phase: Design
 - Abstraction level: Register-transfer level (RTL)
 - Act. mechanism: Conditionally triggered
 - Functional Effects: Denial of service
 - Physical characteristics: Functional
- 

P.o.C.

Simulation without trojan: ([Code and Simulation](#))

In this simulation snapshot:

The **clk** signal is oscillating as expected, providing the timing reference for all transactions.

The **rst_n** signal is initially low, indicating a reset condition, then goes high to start normal operation.

The **adr** bus changes its value to 1, 2, and back to 1, indicating different addresses are being accessed.

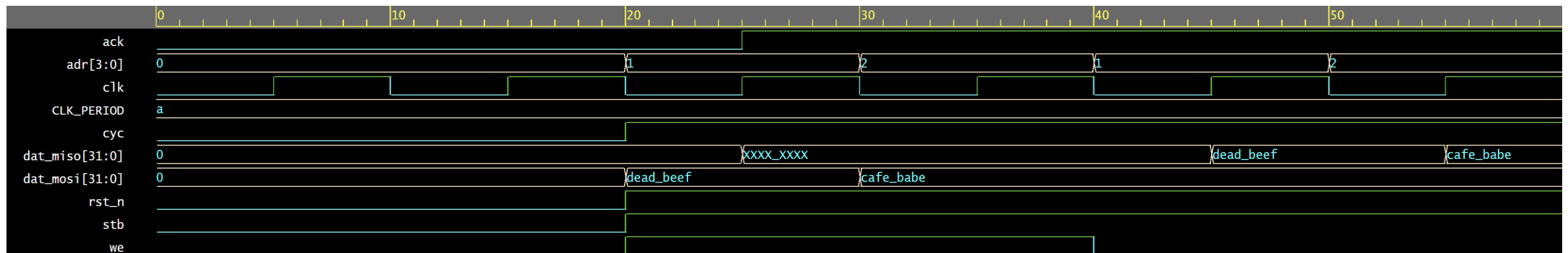
The **stb** and **cyc** signals are asserted (high) when there is an active bus transaction.

The **we** signal is not asserted at all, suggesting that the transactions shown are read operations from the perspective of the master.

The **dat_mosi** signal contains the values *dead_beef* and *cafe_babe*, which are likely test data written to the bus.

The **dat_miso** shows an undefined state (xxxx) initially, which suggests that the data read from the slave device has not yet been driven or there might be a delay in the slave's response or an issue with the slave's response generation logic.

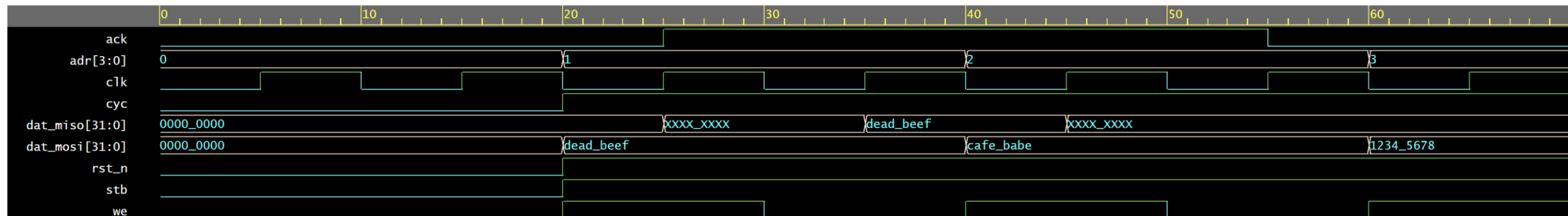
The **ack** signal goes high after each address change, which implies that the slave device is acknowledging the read requests.



Simulation with Trojan ([Code and Simulation](#)):

What this waveform reveals is a sequence of transactions over a Wishbone bus interface, where the master is sequentially reading from addresses 1, 2, and 3.

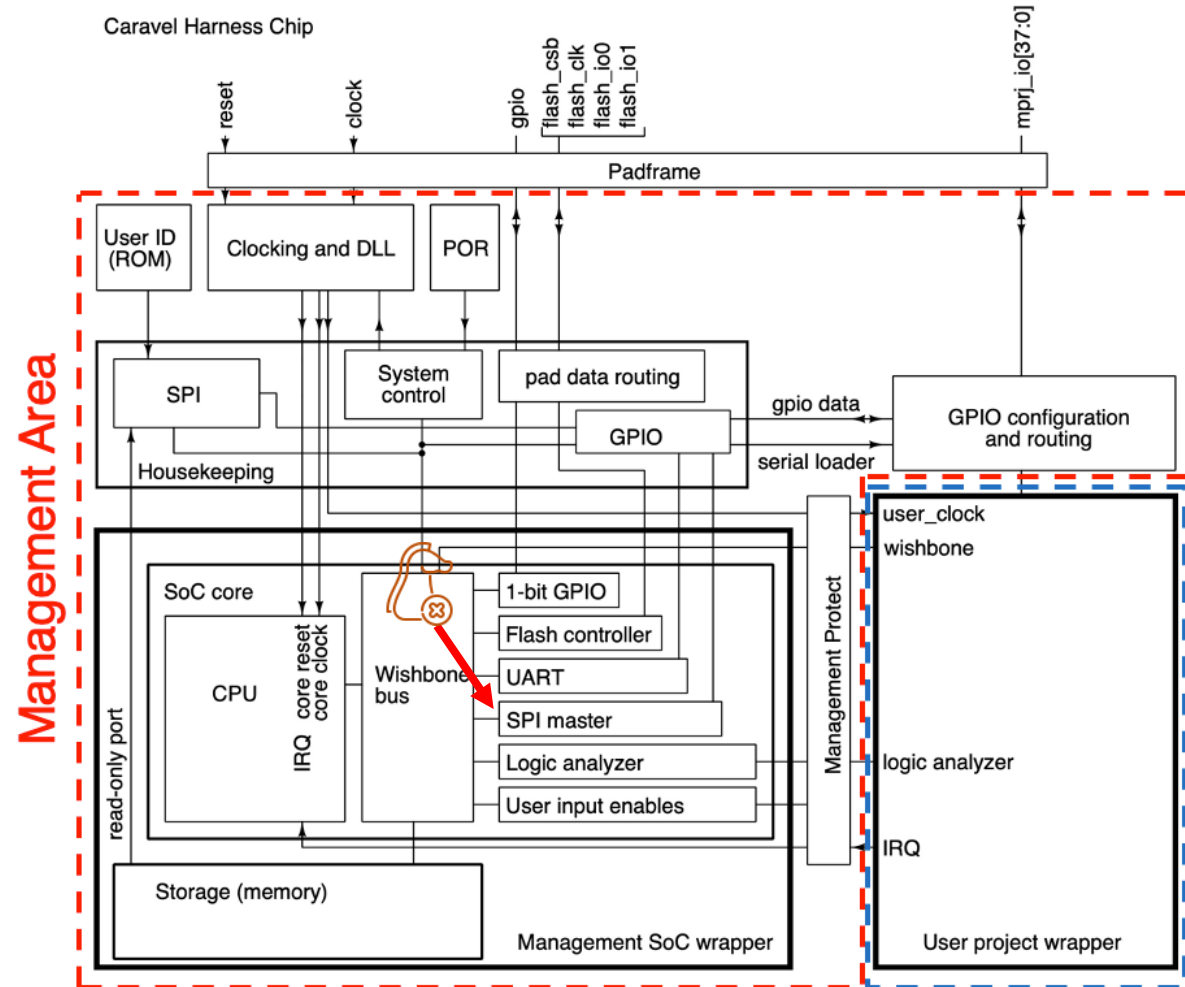
- The undefined data initially present on the dat_miso line suggests that the slave device might have some latency in responding or there's some wait states introduced in the bus cycle.
- The test patterns dead_beef and cafe_babe are typical in digital system testing and are used here for either writing to the slave or expecting such patterns as a read response.
- The pattern 1234_5678 is the last valid data read from address 3 due to the trojan's activation.



Caravel project wishbone bus D.o.S. hw trojan

Our malicious code implementation methodology is:

1. We first analyzed the code in the GitHub repository.
2. Inside the housekeeping.v file the wishbone to SPI to CPU communication is implemented.
3. We can alter the wishbone FSM implementation by adding a stage where if a certain value is transmitted in the bus then an internal signal gets stuck at "0"
4. This way we are glitching the handshake method causing a Denial Of Service.



P.O.C.

Prompt example:

<https://chat.openai.com/share/8d425e27-d6d8-473b-9f53-7e42fdf6c008>

- I isolated the wish bus state machine from housekeeping.v .
- I added a trigger where “**wbbd_busy**” signal should always be set to “**1'b1**” when the “**wbbd_data**” signal has the value “**8'df**”.
- This way the bus glitches stopping SPI communication.

Severity of the vulnerability:

- Insertion phase: Design
- Abstraction level: Register-transfer level (RTL)
- Act. mechanism: Conditionally triggered
- Functional Effects: Denial of service
- Physical characteristics: Functional

Disclaimer: I had intended to publish this section to efabless, however an Ubuntu + Docker configuration error kept me from moving forward. Files created by a docker-based action are owned by root:root, meaning that steps or actions that run as the default runner user in the future cannot modify them.

I was able to solve the issue, however the deadline prevented me from uploading to eFabless and passing precheck.

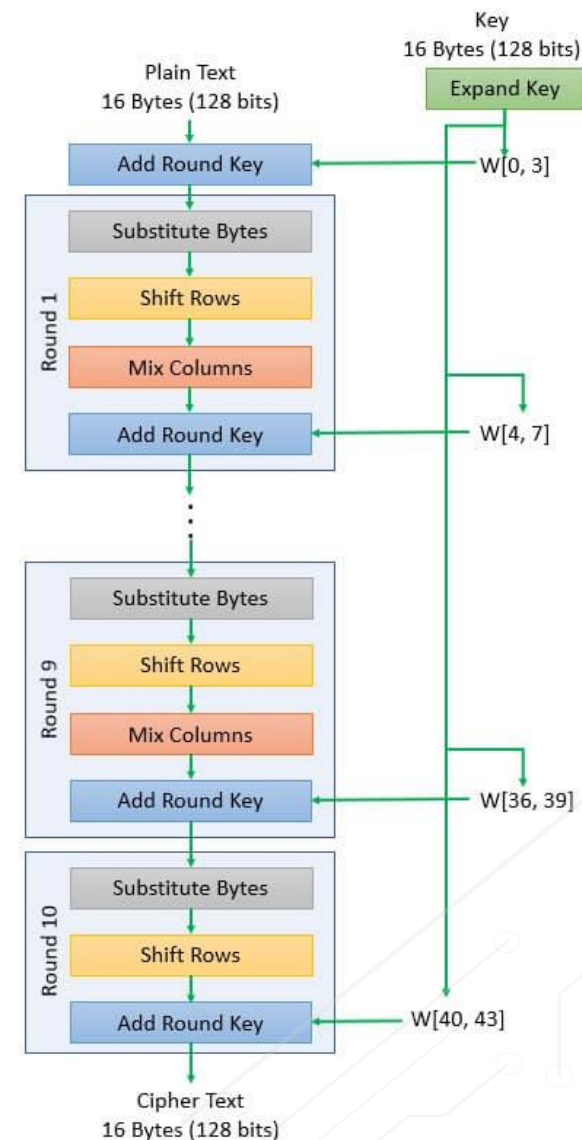
3nd design

Leaking key from a [symmetric AES block cipher](#)

Why attack AES?

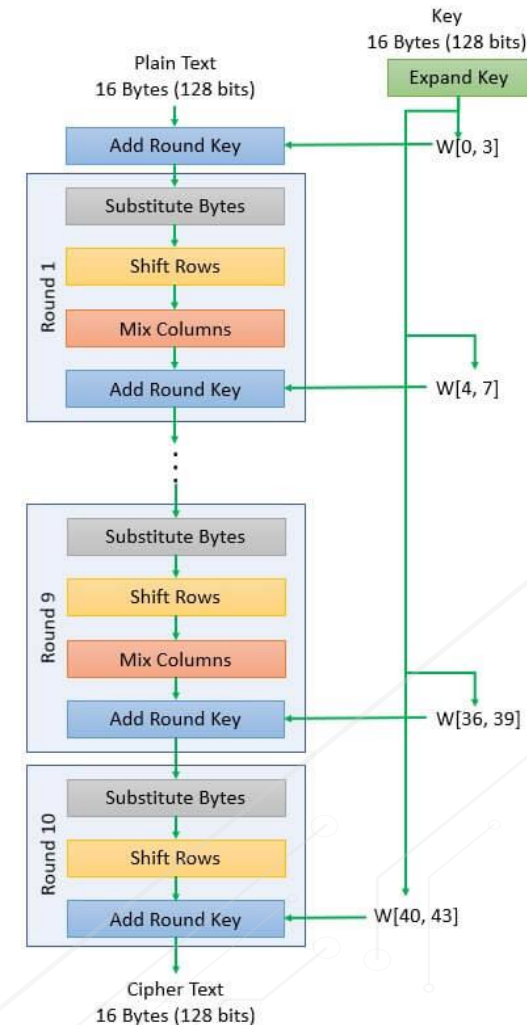
AES is :

- One of the most popular encryption standards.
- Used broadly all over the world.
- Is globally standardized, regulated and in compliance with governments, individuals and enterprises.
- Is efficient in terms of processing power and memory usage so it is used everywhere.



Code implementation methodology:

1. We first analyzed the code in the [GitHub repository](#) it was uploaded.
2. We created a **“transmit”** module for the malicious functionality.
3. We altered the functionality of the **“aes_key_mem”** module so when it is instantiated:
 - a) the **“transmit”** module is instantiated,
 - b) the **“key”** value is copied,
 - c) the **“key”** value is being transmitted by a pin.
 - d) Because we haven't got a pinout I used the **“Ant1”** internal wire for transmission of P.o.C. .
4. We use a register to store the key.
5. We use a covert way of leaking the key by:
 - modulating an (unused) pin on chip that generates an RF signal,
 - this signal can be used to transmit the key bits,
 - then it can be received with an ordinary AM radio,
 - the data carried by the AM signal can be easily interpreted by a human by using a beep scheme.






Leaking the key by modulating an (unused) pin on chip that generates an RF signal.

Severity of the vulnerability:

- Insertion phase: Design
- Abstraction level: Register Transfer level
- Act. mechanism: Conditionally triggered
- Effects: Leak Information
- Location: Processor
- Physical characteristics: Functional

P.o.C.

[Prompt for transmit.v :](https://chat.openai.com/share/8c8fb17c-6647-4eee-8c1e-71c2bc0c1b95)
<https://chat.openai.com/share/8c8fb17c-6647-4eee-8c1e-71c2bc0c1b95>



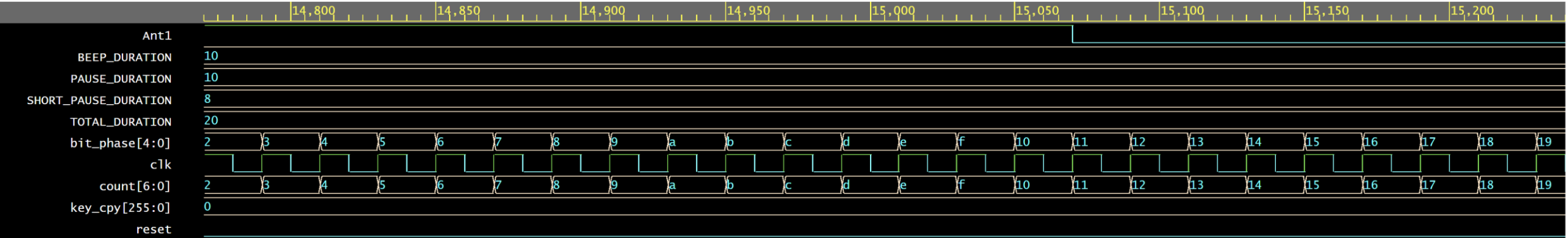
Both waveform diagrams(a,b) represent a form of communication from a signal capture of the AES IP block, a) diagram is a zoomed in version. b) diagram is the more zoomed out screenshot. (Code and simulation can be found [here](#).)

Here are the signal descriptions and their interpretations:

- Ant1** - This represents an antenna signal or activity line for a communication interface.
- BEEP_DURATION, PAUSE_DURATION, SHORT_PAUSE_DURATION, and TOTAL_DURATION** - These represent configurable timing parameters for the system. They could be related to timings of beeps, pauses, and overall duration for an event.
- bit_phase[4:0]** - The bit_phase seems to fluctuate between values 2 through b in hexadecimal (2 to 11 in decimal). This represents different phases or steps in processing a bit or a series of bits within a communication protocol or timing control.
- clk** - This is the clock signal driving the timing of the system. The clock is active and shows a regular square wave pattern, indicating that the system is operational and the timing is continuous.
- count[6:0]** - This is a counter that increments with every clock cycle. It rolls over after reaching b (11 in decimal) back to 2.

When the chip is powered on the key is being transmitted through the antenna at 1560KHz (assuming a 50mhz clock) by implementing a beep scheme where a single beep followed by a pause represents a '0' and a double beep followed by a pause stands for '1'.

a)



b)

