
FULVIO CASTELLO, EUGENIO RESSA, UMBERTO TOPPINO

HARDWARE-BASED CTF

CAPTURE-THE-FLAG CHALLENGES: AN OVERVIEW

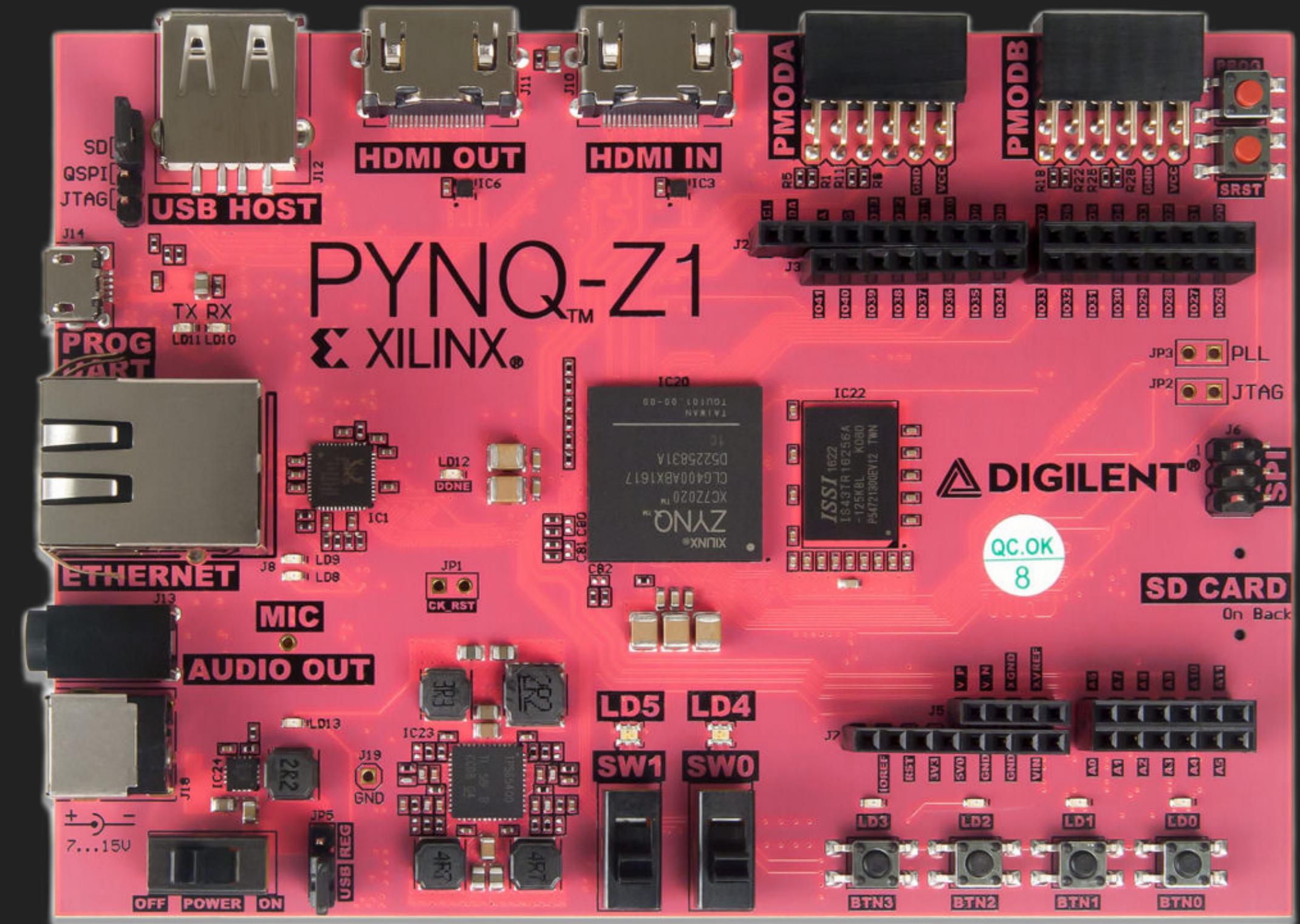
- ▶ Purpose: exploiting one or more vulnerabilities included inside the target platform by means of personal experience/skills, with the objective of capturing a “flag”
- ▶ Flag: a unique string that is formatted in a competition-specific manner. If acquired, it grants access to the desired asset, effectively certifying the success in the competition
- ▶ Types – three main modalities:
 - ▶ Attack/Defense ✕
 - ▶ Jeopardy ♦
 - ▶ King of the Hill ☈
- ▶ Topics: Binary, Crypto, Forensics, Hardware, Miscellaneous, Networking, Web...

GOAL OF THE PROJECT

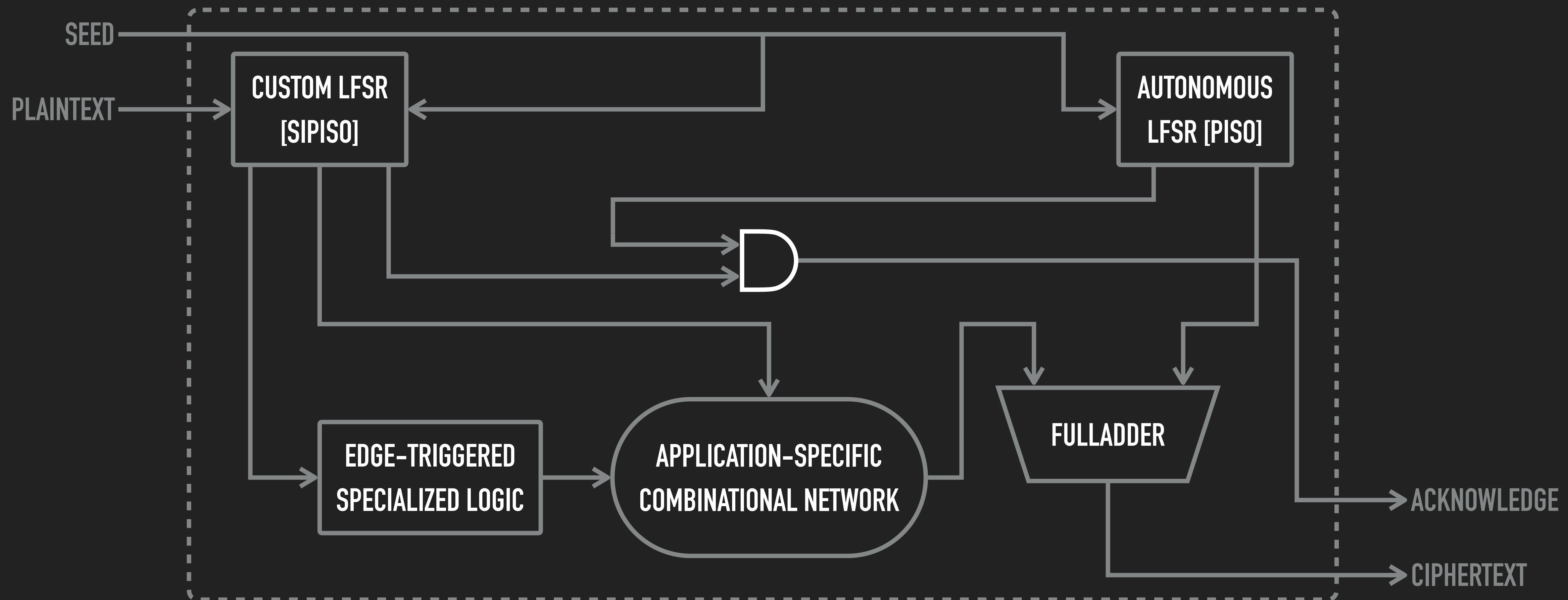
- ▶ Creating a brand new CTF challenge based on a target embedded platform, featuring the following characteristics:
 - ▶ Style – jeopardy
 - ▶ Theme – hardware
 - ▶ Design – 256-bit block cipher
 - ▶ Connectivity – public IP (+ port)
 - ▶ Interaction – online webpage.

CHOSEN HARDWARE ARCHITECTURE

- ▶ PYNO-Z1 Python Productivity board equipped with a Zynq-7000 XC7Z020-1CLG400C SoC:
 - ▶ 650 MHz dual-core Cortex-A9 processor
 - ▶ 512MB DDR3 with 16-bit bus @ 1050 Mbps
 - ▶ High-bandwidth peripheral controllers: 1 Gigabit Ethernet, USB 2.0, SDIO
 - ▶ Low-bandwidth peripheral controllers: SPI, UART, CAN, I2C
 - ▶ Remotely programmable via JTAG/Quad-SPI flash/microSD.



TOP-LEVEL DESIGN ▶ BLOCK CIPHER



LINKING FPGA \Leftrightarrow MCU

- ▶ Data exchange with the FPGA itself is done through the AXI4 peripheral, which includes a flexible amount of registers directly accessible from the external world
- ▶ The top-level component has to be instantiated inside said device in order to create and package an application-specific IP
- ▶ The result of this process is a bitstream – a collection of configuration files used to describe all the logic needed to actually implement the VHDL design into hardware
- ▶ The Overlay Python library subsequently handles the interface with the MCU, exposing all the desired functionalities in the form of easily accessible APIs.

REMOTE COMMUNICATION WITH THE BOARD ↗

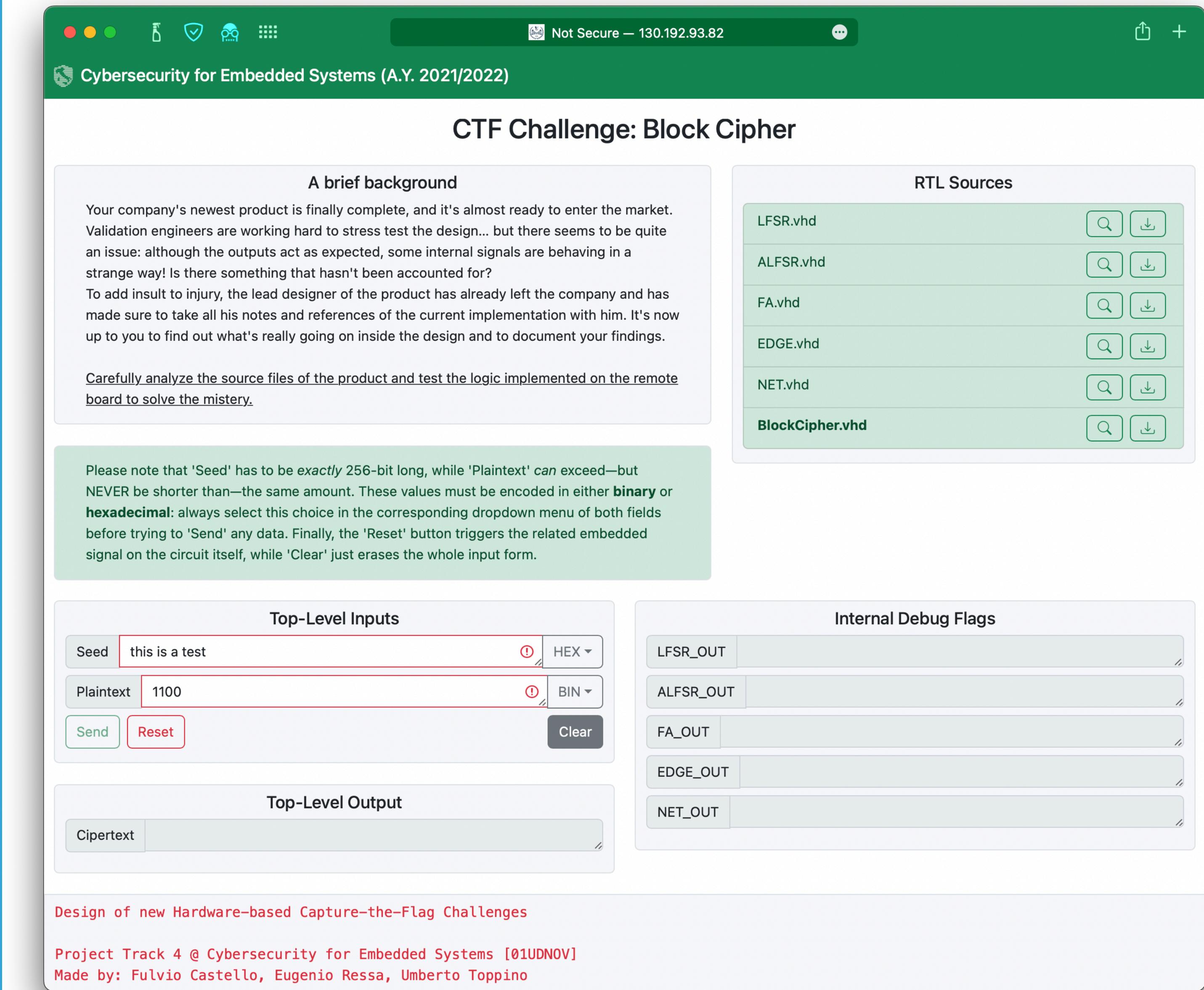
- ▶ The server side of the challenge is powered by CherryPy, an object-oriented Python web framework
- ▶ All the methods used to link FPGA & MCU are thereby exposed as publicly accessible static URLs
- ▶ The contestant can thus send/receive data by means of standard HTTP requests, directly interfacing with the AXI4 peripheral registers from remote
- ▶ Several front-end packets made of HTML/CSS/JS code are then returned, so that it becomes possible to interact with the challenge itself by means of an intuitive GUI.

CLIENT-SIDE GUI ↗

- ▶ The Graphical User Interface (GUI) is a webpage accessible to the competitors by means of any popular browser. It includes four main sections:
 - ▶ Challenge Description – a brief narrative that serves as a background for the CTF challenge itself, for the sake of player immersion
 - ▶ Source Files – a list of links used to preview and/or download all the components that make up the inspected VHDL design
 - ▶ Input Form – a number of text fields that allow the player to send data to different input ports of the top-level entity fitted onto the board
 - ▶ Output Forms – a group of read-only dynamic labels containing both intermediate and final results of the functionality implemented into the FPGA.

DEMO

- ▶ Seed == 256 bit
- ▶ Plaintext \geq 256 bit
- ▶ Ciphertext length is a consequence of the previous field
- ▶ Binary & HEX are both supported.



The screenshot shows a web browser window titled "Cybersecurity for Embedded Systems (A.Y. 2021/2022)" with the URL "Not Secure — 130.192.93.82". The main content is a challenge titled "CTF Challenge: Block Cipher".

A brief background:
Your company's newest product is finally complete, and it's almost ready to enter the market. Validation engineers are working hard to stress test the design... but there seems to be quite an issue: although the outputs act as expected, some internal signals are behaving in a strange way! Is there something that hasn't been accounted for? To add insult to injury, the lead designer of the product has already left the company and has made sure to take all his notes and references of the current implementation with him. It's now up to you to find out what's really going on inside the design and to document your findings.

Carefully analyze the source files of the product and test the logic implemented on the remote board to solve the mystery.

Please note that 'Seed' has to be *exactly* 256-bit long, while 'Plaintext' *can* exceed—but NEVER be shorter than—the same amount. These values must be encoded in either **binary** or **hexadecimal**: always select this choice in the corresponding dropdown menu of both fields before trying to 'Send' any data. Finally, the 'Reset' button triggers the related embedded signal on the circuit itself, while 'Clear' just erases the whole input form.

Top-Level Inputs:
Seed: this is a test (dropdown: HEX)
Plaintext: 1100 (dropdown: BIN)
Buttons: Send, Reset, Clear

Top-Level Output:
Ciphertext: [empty input field]

Internal Debug Flags:
LFSR_OUT
ALFSR_OUT
FA_OUT
EDGE_OUT
NET_OUT

RTL Sources:
LFSR.vhd
ALFSR.vhd
FA.vhd
EDGE.vhd
NET.vhd
BlockCipher.vhd

Design of new Hardware-based Capture-the-Flag Challenges
Project Track 4 @ Cybersecurity for Embedded Systems [01UDNOV]
Made by: Fulvio Castello, Eugenio Ressa, Umberto Toppino

DEMO

- ▶ Connect to **130.192.93.82**
- @ port **8080**
- ▶ ‘Send’: issue, await, fetch information
- ▶ ‘Reset’: trigger the related signal on the board
- ▶ ‘Clear’: erase all available user inputs.

Cybersecurity for Embedded Systems (A.Y. 2021/2022)

CTF Challenge: Block Cipher

A brief background

Your company's newest product is finally complete, and it's almost ready to enter the market. Validation engineers are working hard to stress test the design... but there seems to be quite an issue: although the outputs act as expected, some internal signals are behaving in a strange way! Is there something that hasn't been accounted for? To add insult to injury, the lead designer of the product has already left the company and has made sure to take all his notes and references of the current implementation with him. It's now up to you to find out what's really going on inside the design and to document your findings.

Carefully analyze the source files of the product and test the logic implemented on the remote board to solve the mystery.

Please note that 'Seed' has to be *exactly* 256-bit long, while 'Plaintext' *can* exceed—but NEVER be shorter than—the same amount. These values must be encoded in either **binary** or **hexadecimal**: always select this choice in the corresponding dropdown menu of both fields before trying to 'Send' any data. Finally, the 'Reset' button triggers the related embedded signal on the circuit itself, while 'Clear' just erases the whole input form.

Top-Level Inputs

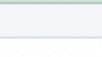
Seed	8F610A58815BA37A17FEFDA7B28DAB2C5E711C1EBB1C02 CEA042D54BEC4EA1E0	✓	HEX ▾
Plaintext	A320F385012525AADB218E3785CBBAA095024DA5D 8E81B1A10DAD62BD51740E	✓	HEX ▾

Send
Reset
Clear

Top-Level Output

Ciphertext	0xD6B69A42278C0DE71B7135C3E98F535288A583D4FA659FFD15 E1BD5785ACD7
------------	--

RTL Sources

LFSR.vhd	 
ALFSR.vhd	 
FA.vhd	 
EDGE.vhd	 
NET.vhd	 
BlockCipher.vhd	 

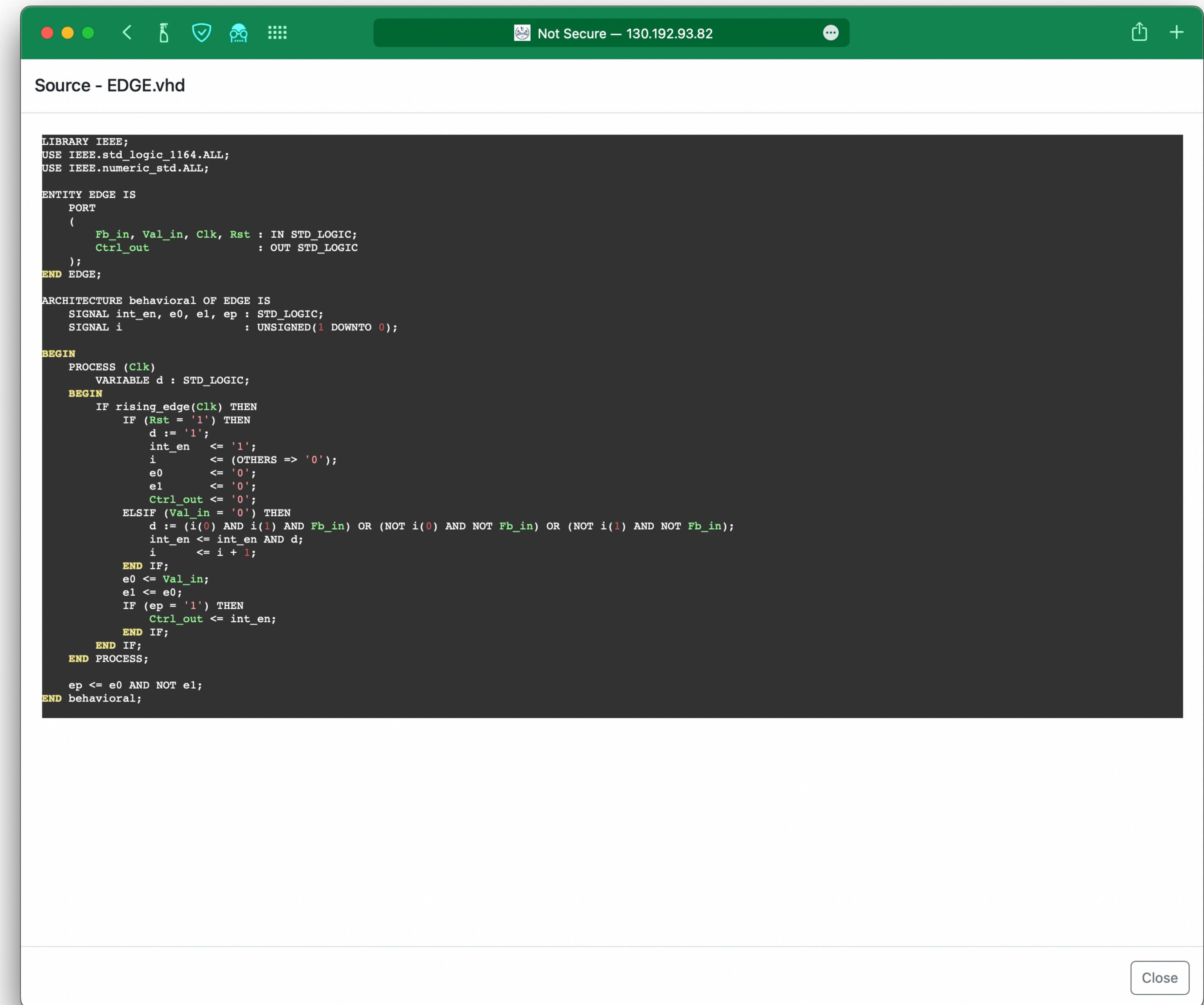
Internal Debug Flags

LFSR_OUT	0x9C0C8189BF9C7B15464FCC675990932C29E6BDDDEAB171C5 B33469098EAECF
ALFSR_OUT	0x2BDB34171B42BE369898404DC81331AA62432E24374CB937 D48097DADC296F
FA_OUT	0x886BBB8CFD4FEC44C2B73C6F81CDC46203510699D0A5DE1FAB 9949C4CD3CDE
EDGE_OUT	0x15B8689ABDE65A620D57DA6D44F993CB4A3CDE4457FA8A8 BB72F46F24D1421
NET_OUT	0x7D9E479686888AAFED120B2D2C8DD81939F99C4C02C6883 5832D08D583AC6

Design of new Hardware-based Capture-the-Flag Challenges

DEMO

- ▶ Preview VHDL netlists inside the browser
- ▶ Download source files to the local system
- ▶ Well-formatted, easy to read code...
- ▶ ...but (hopefully) hard to unriddle!



```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY EDGE IS
    PORT
    (
        Fb_in, Val_in, Clk, Rst : IN STD_LOGIC;
        Ctrl_out : OUT STD_LOGIC
    );
END EDGE;

ARCHITECTURE behavioral OF EDGE IS
    SIGNAL int_en, e0, e1, ep : STD_LOGIC;
    SIGNAL i : UNSIGNED(1 DOWNTO 0);

BEGIN
    PROCESS (Clk)
        VARIABLE d : STD_LOGIC;
    BEGIN
        IF rising_edge(Clk) THEN
            IF (Rst = '1') THEN
                d := '1';
                int_en <= '1';
                i <= (OTHERS => '0');
                e0 <= '0';
                e1 <= '0';
                Ctrl_out <= '0';
            ELSIF (Val_in = '0') THEN
                d := (i(0) AND i(1) AND Fb_in) OR (NOT i(0) AND NOT Fb_in) OR (NOT i(1) AND NOT Fb_in);
                int_en <= int_en AND d;
                i <= i + 1;
            END IF;
            e0 <= Val_in;
            e1 <= e0;
            IF (ep = '1') THEN
                Ctrl_out <= int_en;
            END IF;
        END PROCESS;
    ep <= e0 AND NOT e1;
END behavioral;
```

“AND THAT'S A WRAP! THANK
YOU ALL FOR YOUR ATTENTION.”

Project Track 4