Reverse Engineering: FPGA vs ASIC

Pedro Bernardo

December 2, 2020

Reverse Engineering: What is it?

"Reverse Engineering is the process by which an artificial object is deconstructed to reveal its designs, architecture, code or to extract knowledge from the object."
- *Wikipedia*

Reverse Engineering: Is it legal?

Yes!

- Protected in the US by the *Semiconductor Chip Protection Act*
- Similar legislation in Japan and the EU

Reverse Engineering: Who is it for?

Customers fall into two groups:

- Manufactoring companies (technical information)
- Lawyers or IP groups in companies (patent-related information)

ASIC

What is it?

Application-**S**pecific **I**ntegrated **C**ircuit

ASICs: Types of Reverse Engineering

- Product Teardown
- System-level Analysis
- Process Analysis
- Circuit Extration

ASICs: Product Teardowns

- Unit is disassembled
- Boards and sub-assemblies are photographed
- Components are catalogued and inventoried

ASICs: System-level Analysis

Electroninc systems can consist of:

- Hardware, software, firmware, communications, etc.

System analysis is useful for all of these.

ASICs: System-level Analysis - Hardware

Board is delayered (if needed).

Components and connections are identified.

Board schematics are re-created.

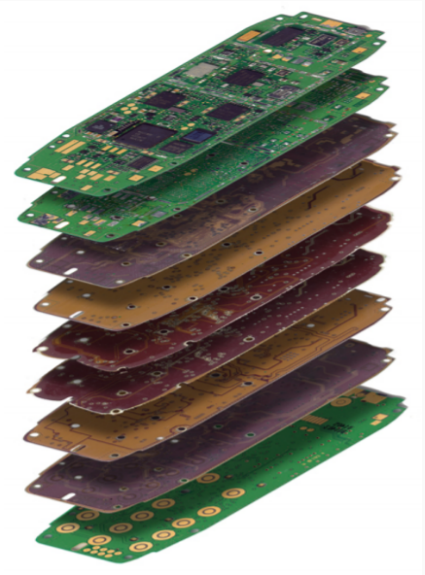


Figure 1: PCB layers from a phone

ASICs: System-level Analysis - Software

- Extraction of embedded code
- Sometimes protected with HW or SW locks
- Can often be disabled through a collection of techniques

ASICs: System-level Analysis - Software

- HW locks bypassed through IC microsurgery (requires Circuit Analysis first)
- Ecnrypted code requires encryption analysis, followed by decryption
- Disassemblers and decompilers (IDA, Ghidra) can be used to Reverse Engineer the software
- Code can be analyzed statically or dynamically

ASICs: Circuit Extraction

- Device Depot
- Device Delayering
- Imaging
- Anotations
- Verification and Schematic Creation
- Schematic Analysis and Organization

ASICs: Circuit Extraction

Figure 2: RE as it used to be done!

FPGA

FPGA: Why Reverse Engineer them?

Same reasons, and then some more!

FPGA: RE Motivation

- From bitstream to original design
- If not possible, data can still be extracted
  - Encryption keys
  - Look-Up-Table content

FPGA: RE Motivation

Threat of bitstream modification or injection:

- Since original design is not known, modifications to bitstream are difficult.

FPGA: Security

- Unlike ASICs, FPGA RE can be done with only a configuration file. Hence the bitstream format being secret

FPGA: Makeup

- Configurable Logic Blocks (CLBs),
  - LUT, MUXers, adders, Flip-Flops
- I/O Blocks (IOBs) and
  - Connection to external stimulus
- Interconnects (Switch Matrix)
  - Connect CLBs and IOBs to work together, creating the desired logic

FPGA: Makeup

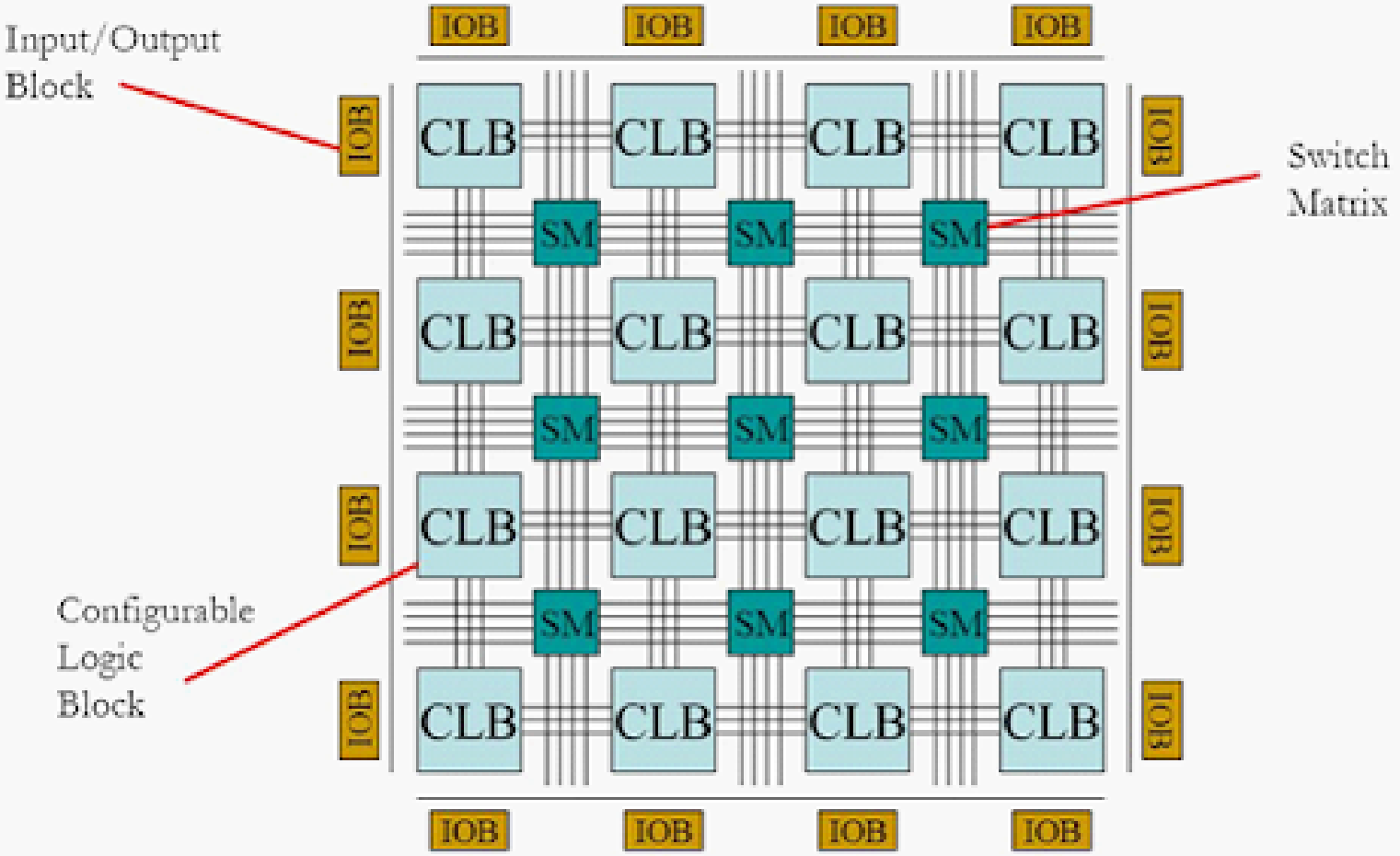


Figure 3: Illustration of FPGA fabric

FPGA: Reconfigurability Tradeoff

- ASICs are more power efficient
- ASICs more compact for the same circuit

FPGA: Bitstream

- FPGAs are configured using HDL
- The synthesis engine produces bitstream from HDL
- Bitstream represents the configuration options for the CLBs, IOBs and the switch matrix.

FPGA: Bitstream Synthesis

Figure 4: From HDL to the Bitstream

- Netlist contains list of components and their connections
- Map maps the netlist components into board components
- Place places the mapped components and assigns them specific locations in the board
- Route makes connections betweel all of the placed components.
- Bitstream file is created

FPGA: Approaches to Protection

- Obfuscation
- Encryption

FPGA: Approaches to Protection - Obfuscation

Process of modifying description or structure of a circuit in order to conceal it's functionality.
Can be:

- Passive
    - Alters HDL comprehensibility
- Active
    - Alters circuit functionality

FPGA: Approaches to Protection - Encryption

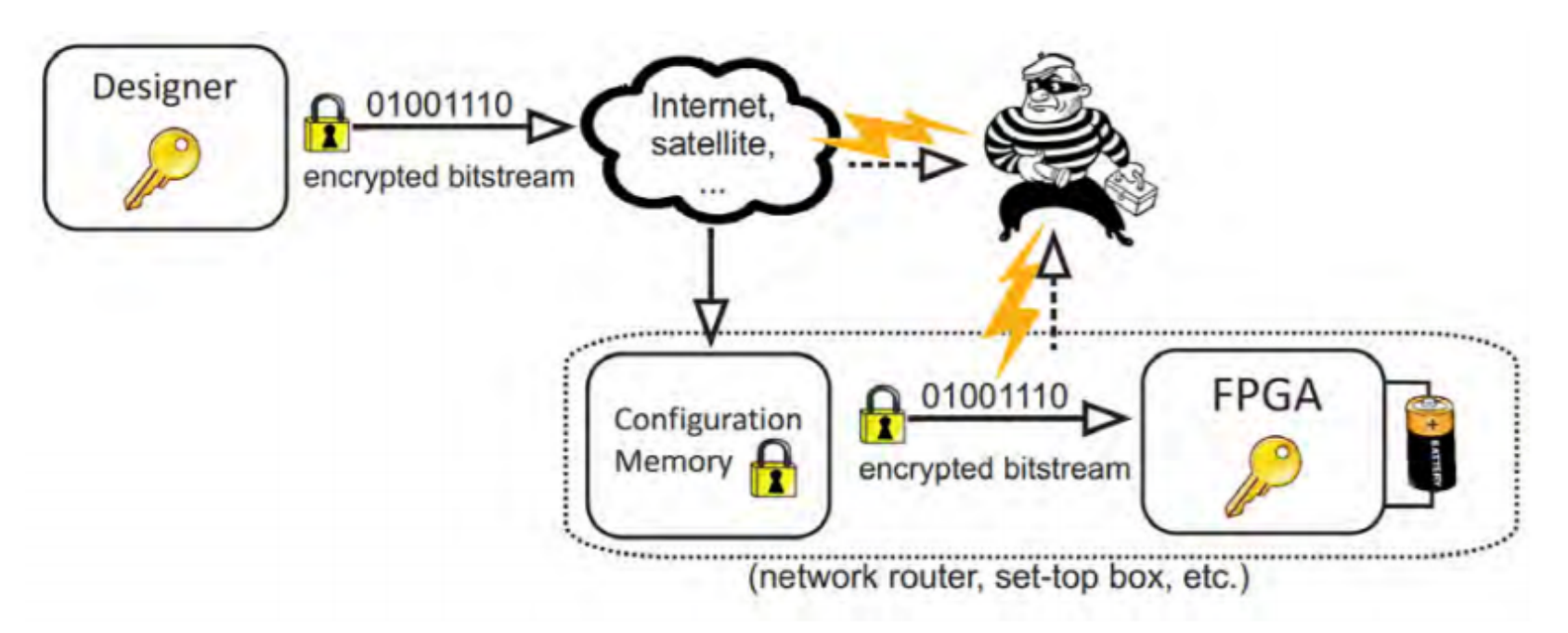


Figure 5: FPGA receiving encrypted bitstream form designer

FPGA: Approaches to Protection - Encryption

Encryption is not enough!

- Low-energy or small form-factor applications do not have encryption block.
- Remote upgrades must use the same key.
- Devices that stay in the field for a long time are susceptible to physical attakcs.

FPGA Reverse Engineering: Example

Research paper by Daniel Celebucki of the USAF.

Target system: *LatticeECP3 LFE3-35EA-8FN484C FPGA*

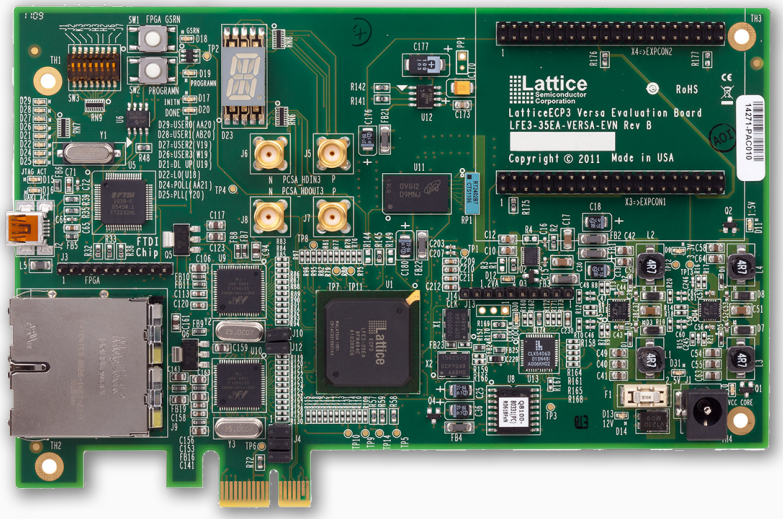


Figure 6: LatticeECP3 LFE3-35EA-8FN484C FPGA

FPGA Reverse Engineering: Assumptions

- Bitstream must not be encrypted
- Not obfuscated

May not seem so, but these are valid assumptions.

Not all FPGAs offer encryption options.

No obfuscation in the bitstream is the common case.

FPGA Reverse Engineering: The Process

- Using the provided tooling:
  - Most important tool is the Synthesis Software
- Select a specific configuration option to reverse engineer.
  - Specific pin set as input or output, logic function in a CLB, etc.

FPGA Reverse Engineering: The Process

- Exercise that configuration through all it's possible values
  - Usually done through scripting (TCL scripts)
  - Modifying HDL or LPF (XDC for Xilinx) files
  - Performing the bitstream synthesis
  - Lattice Diamond software (equivalent to Xinilx's Vivado)
- Analyse the bitstreams
  - Correlate the differences in the bitstreams to the differences in the description

FPGA Reverse Engineering: The Process

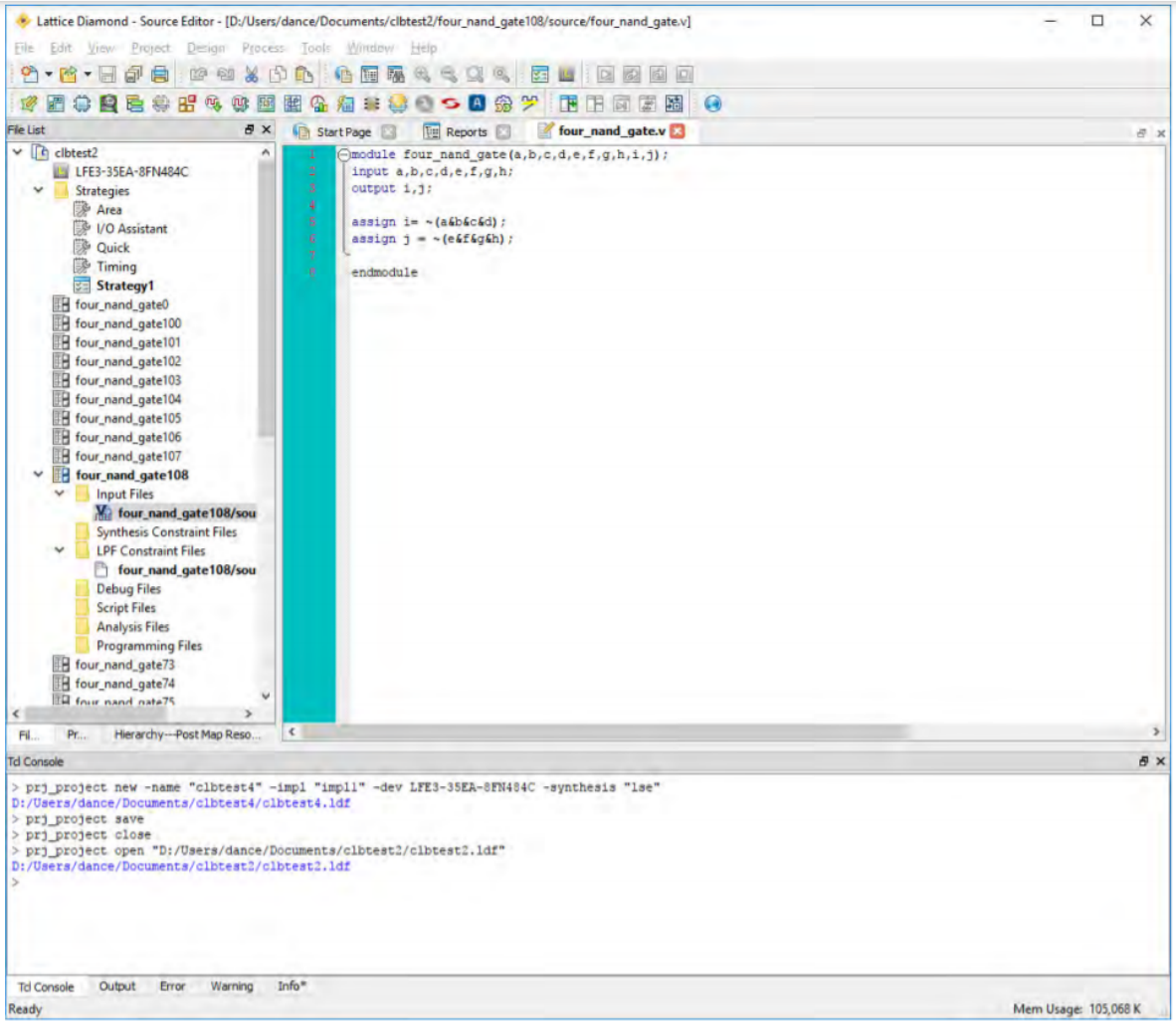


Figure 7: Lattice Diamond software GUI showing HDL editor, File List view, and TCL Console

FPGA Reverse Engineering: I/O Blocks

- Many fewer then CLBs or switching matrix options.
- Configuration options can be easily changed. (Usually through constraints files)

FPGA Reverse Engineering: I/O Blocks

The pullmode is responsible for describing how the signal will be interpreted at the pin. The pullmode can be set to four options:

- **Up**: Input is attachd to pull-up resistor (pin is set to 1)
- **Down**: Input is attachd to pull-down resistor (pin is set to 0)
- **Keeper**: Neither pull-up nor pull-down. Drives weak 0 or 1 matching the level of the last logic state present on the pad.
- **None**: Not set to any mode

FPGA Reverse Engineering: I/O Blocks

Algorithm 1 Pullmode Bitstream Generation

1: **for** Pin **p** in all I/O Pins **do**
2: **for** pullmode **val** in UP,DOWN,KEEPER,NONE **do**
3: Replace IOBUF line in LPF with "IOBUF PORT "a" PULLMODE=**val**"
4: **end for**
5: Replace Location line in LPF with "LOCATE COMP "<input/output pin name>" SITE **p**"
6: **end for**

Figure 8: Algorithm used to generate the pullmode bitstreams

FPGA Reverse Engineering: I/O Blocks

Table 1. Pullmode Groups

Group Number	Indices
Group 1	476 or 477
Group 2	440404 or 440405
Group 3	667296 or 667297
Group 4	895054 or 895055
Group 5	Offset Pattern: 1, 2, 433, 434, 435
Group 6	Offset Pattern: 427, 428, 433, 860, 861

FPGA Reverse Engineering: I/O Blocks

The Slew Rate and Drive were two other configurations that were reverse engineered for IOBs.

- Slew rate: maximum voltage change per unit time in a node of a circuit
- Can be either *fast* or *slow*
- Drive: applicable to all output and bidirectional pins and specifies the strength of the output signal in milliamps

FPGA Reverse Engineering: I/O Blocks

Three main goals to starting off with IOBs:

- Map all configuration options for each pin to their respective indices and values in the bitstream
- Determine whether a pin was an input or an ouput on the bitstream file
- Determine whether a pin was connected to logic blocks in the design

FPGA Reverse Engineering: Configurable Logic Blocks

- Usually harder to reverse, but next logical step
- Many more than the IOBs
- Not so easy to directly modify

FPGA Reverse Engineering: Configurable Logic Blocks

- LUTs are configured based on the HDL when synthesized
- LPF files are not used until the *map*, *place* and *route* take place
- Different (but equivalent logic) may be implemented by the synthesis engine, making the reversing process harder

FPGA Reverse Engineering: Configurable Logic Blocks

- Platform dependent solution: Lattice FPGAs have primitives supported by each device
- In this case, LUT4 primitive was used to configure the LUT with the intended configuration value
- LUTs are configured based on the output of their desired truth table.

FPGA Reverse Engineering: Configurable Logic Blocks

```
D C B A : Z
0 0 0 0 : 0
0 0 0 1 : 0
0 0 1 0 : 1
0 0 1 1 : 0

0 1 0 0 : 0
0 1 0 1 : 0
0 1 1 0 : 1
0 1 1 1 : 0

1 0 0 0 : 0
1 0 0 1 : 0
1 0 1 0 : 1
1 0 1 1 : 0

1 1 0 0 : 1
1 1 0 1 : 1
1 1 1 0 : 1
1 1 1 1 : 1

INIT = F444 (16)
```

Figure 9: LUT initialization value based on the desired truth table

FPGA Reverse Engineering: Configurable Logic Blocks

- Figure out the value the LUT was initialized to
- Remake the truth table
- Use the same strategy as before (bitstream generation and comparison)
- The 16 bit value was assumed to be stored in the bitstream
- 61 bitstreams were generated for the LUT
  - 0x0000-0x000F, 0x0010-0x00F0, 0x0100-0x0F00, 0x1000-0xF000

FPGA Reverse Engineering: Configurable Logic Blocks

- Enough to identify the bits responsible for LUT configuration
- But, confirmation is needed:
  - Generate bitstreams with different combinational logic (for that specific LUT)
  - Recover the truth table from the bitstreams
  - Confirm if the logic is the same

FPGA Reverse Engineering: Configurable Logic Blocks

- It was accurate. Most times...
- For one testcase, the expected was $AB + C\overline{D}$

FPGA Reverse Engineering: Configurable Logic Blocks

FPGA Reverse Engineering: Configurable Logic Blocks

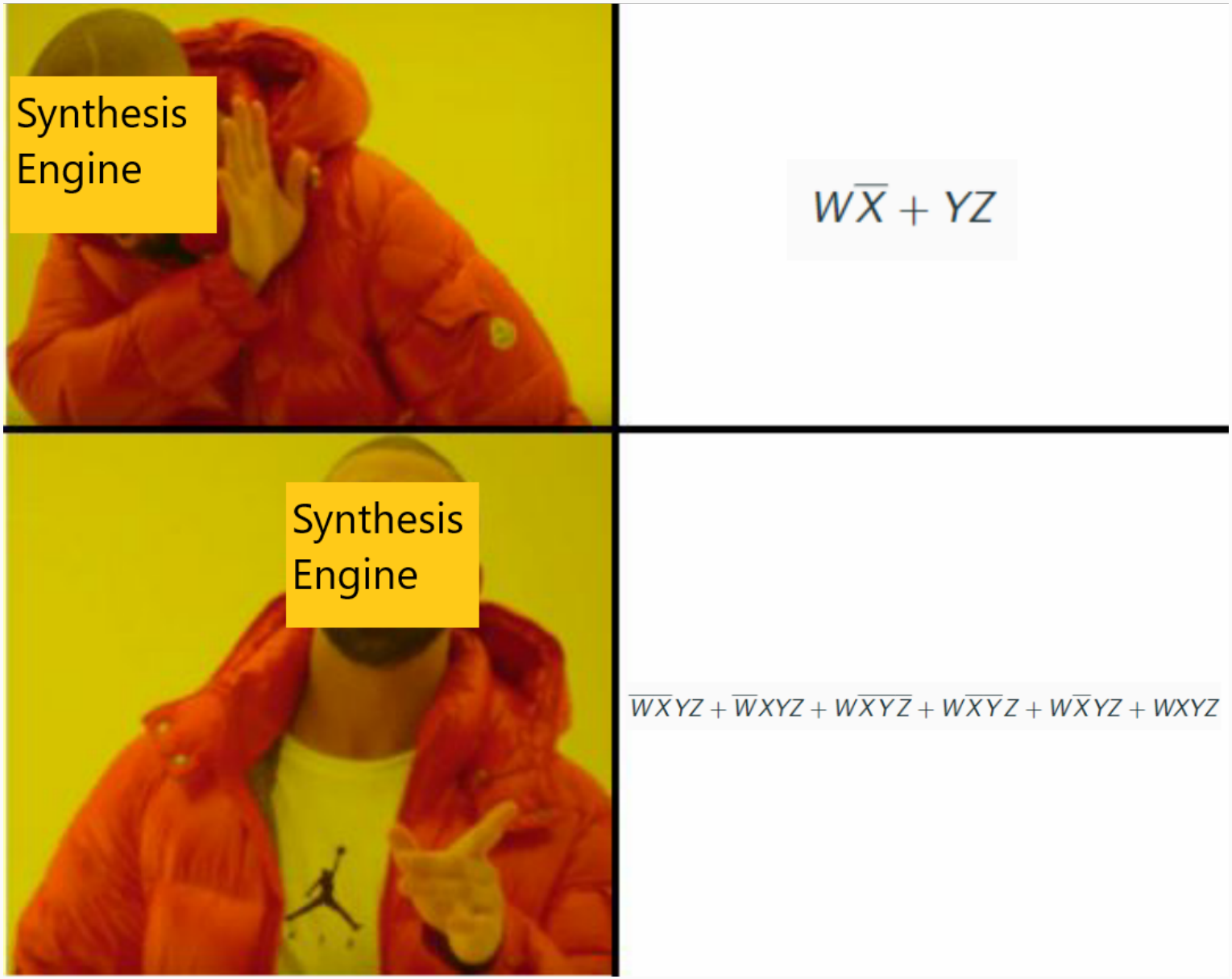
Which yields the following logic function:

$$\overline{WX}YZ + \overline{W}XYZ + W\overline{XYZ} + W\overline{XY}Z + W\overline{X}YZ + WXYZ$$

Which reduces to the expected

$$W\overline{X} + YZ$$

FPGA Reverse Engineering: Configurable Logic Blocks



FPGA Reverse Engineering: Switching Matrix

- Much harder to reverse due to higher number of configuration options
- Provides vital information about the circuit (CLB and IOB connections)
- Can be reversed using the same techniques

Questions?

References i

[1] Randy Torrance, Dick James. *The State-of-the-Art in IC Reverse Engineering*. https://www.iacr.org/archive/ches2009/57470361/57470361.pdf. Online; accessed 27 November 2020.

[2] Daniel J. Celebucki. *Methods of Reverse Egineering a bitstrem for Field Programmable Gate Array Protection*. https://apps.dtic.mil/dtic/tr/fulltext/u2/1055984.pdf. Online; accessed 28 November 2020.

[3] Thomas Weber. *HITB 19 - Reverse Engineering Custom ASICs By Exploiting Supply-Chain Leaks*. https://www.youtube.com/watch?v=iZRbf-gWFBo. Online; accessed 26 November 2020.

References ii

[4] Mathias Lasser. *34C3 - Reverse Engineering FPGAs*. https://www.youtube.com/watch?v=Y1Wwa8csFjk. Online; accessed 26 November 2020.

[5] Wikipedia. *Reverse Engineering*. https://en.wikipedia.org/wiki/Reverse_engineering. Online; accessed 29 November 2020.

[6] Wikipedia. *Application-specific integrated circuit*. https://en.wikipedia.org/wiki/Application-specific_integrated_circuit. Online; accessed 29 November 2020.

References iii

[7] embeddedbits.org *Extracting firmware from devices using JTAG*. https://embeddedbits.org/2020-02-20-extracting-firmware-from-devices-using-jtag/ Online; accessed 28 November 2020.