

Experiment 11

Reverse Engineering Bitstream

The objective of this experiment is to reverse-engineer bitstream.

Instructor: Dr. Swarup Bhunia

TAs: Shuo Yang and Reiner Dizon

Theory Background

1. FPGA and Bitstream Generation

Most logic circuits (microprocessor cores, memory controllers, accelerators, ...) are just an assembly of combinatorial logic functions and flip-flops (registers). Fig. 1 shows a typical floorplan of an FPGA (Max 10 here). An FPGA consists of I/O banks, logic array block (LAB), embedded memory, embedded multipliers, global clock network (GCLK), a phase-locked loop (PLL), analog-to-digital converters (ADC), etc. LABs are configurable logic blocks that consist of a group of logic resources. Each LAB consists of 16 logic elements (LEs), LE carry chains, LAB control signals, local interconnect, and register chains. LE is the smallest logic unit in MAX 10 devices. An FPGA can be programmed indefinitely to implement any of these circuits and connect it to the outside world. Any logic function is implementable using logic elements (LEs).

The configurable resources can be programmed to one or more predefined options, which is controlled by configuration bits called a bitstream. Bitstream is a sequence of bits which is used to configure an FPGA. Fig. 2 shows the FPGA design flow (from design to bitstream generation) and the FPGA design flow:

- i. At first, the Verilog/VHDL files are read and compiled.
- ii. The logic functions are broken down into LEs and registers connected together.
- iii. The output is called a technology-mapped netlist. It corresponds to the phases of logic synthesis and mapping.
- iv. The LEs are assigned to physical locations on the chip. This is called the placement phase.
- v. Connections between LEs are established through the switch boxes. This is called the routing phase.
- vi. A binary file called the bitstream is generated, which contains the contents of each LE and the configuration of each switch box.
- vii. The bitstream is loaded into an FPGA device.

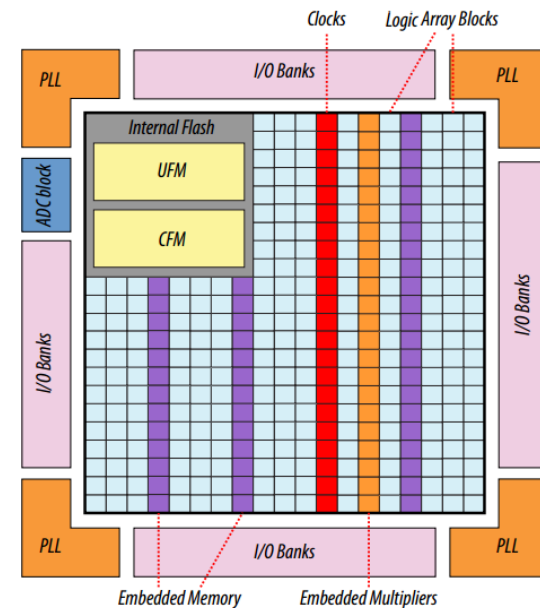


Fig. 1: Typical Device Floorplan for MAX 10 Devices [1].

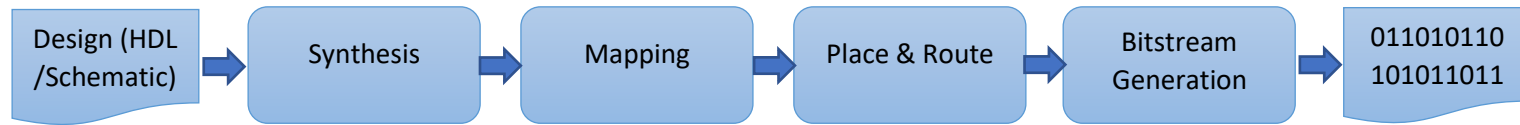


Fig. 2: FPGA design flow.

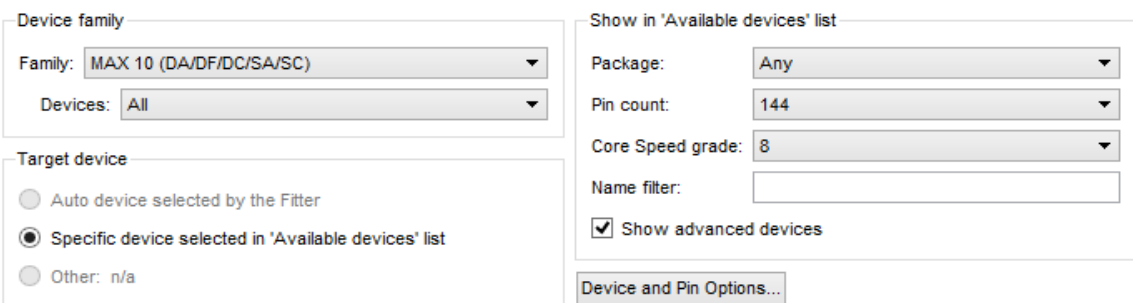
2. Bitstream Reverse Engineering

Reverse engineering bitstream can help in gathering bitstream details, which are otherwise withheld by vendors, necessary for creating a bitstream generation tool. A program, in an FPGA design flow, can't be analyzed from the bitstream. The bitstream format is proprietary and undocumented from the FPGA vendors. Even with understanding, it's very difficult to analyze the bitstream in order to clone an original design. Encryption can be encrypted or unencrypted. In the case of the encrypted bitstream, a key is used to encrypt the bitstream so that an attacker does not understand the contents of the bitstream. A symmetric key should be in the FPGA device (store permanently in a non-volatile memory) in order to decrypt the design. The decrypted version is used to configure an FPGA for a specific design. However, FPGA bitstream is unencrypted in most of the cases. Usually, security through obscurity is performed which is not always secret.

Experiment Set-up: Configuration

Perform following steps to reverse engineering bitstream.

- i. Device set-up: Assignment -> Device -> Device and pin options (Fig. 3)
- ii. Configuration: Configuration scheme: Internal configuration and configuration mode: Single uncompressed Image. Uncheck Generate compressed scheme. Uncheck **Force VCCIO to be compatible with the configuration I/O voltage**. Click **Device options** and uncheck **Verify project**
- iii. Programming Files: Click **programming Files** from left. Check Serial Vector Format, JEDEZ STAPL Format File, and Jam STAPL Byte Code 2.0 File.
- iv. Error Detection CRC: uncheck all options.



Device family

Family: MAX 10 (DA/DF/DC/SA/SC)

Devices: All

Target device

Auto device selected by the Filter

Specific device selected in 'Available devices' list

Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: 144

Core Speed grade: 8

Name filter:

Show advanced devices

Device and Pin Options...

Fig. 3: Setup for MAX 10 FPGA.

Instructions and Questions

In this experiment, we will learn how to reverse engineer logic primitives like NAND, NOR, XOR, etc.

PART I Reverse engineering bitstream with known bitstream

- i. Write a Verilog/VHDL code for a 2-input logic gate (AND/OR/NAND/XOR/NOR). Assign pin 44, pin 50, and pin 52 for input X, input Y, and output Z, where $Z=X$ (AND/OR/NAND/XOR/NOR) Y.
- ii. Compile code using Ctrl+L for each condition;
- iii. Locating logic design: Go to RTL viewer from Tools. Then go to Locate Chip Planner, as shown in Fig. 4. Make sure that 2-XOR gate is in coordinate (31, 0).
- iv. Generate bitstream (.sof file).
- v. Read the bitstream file with the given Matlab code bitgen.m.
- vi. Write a code (Matlab/C/python) to compare generated bitstream with the unknown bitstream.
- vii. Determine the logic gate of the provided bitstream.

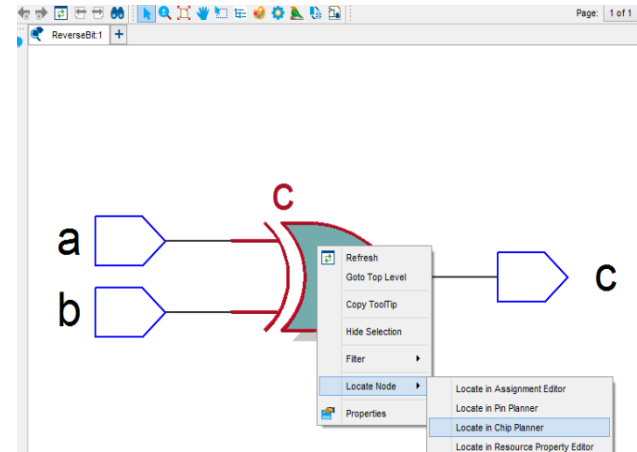


Fig. 4: Setup of the chip planner for MAX 10.

PART II Bitstream characteristic

- i. Write a Verilog/VHDL code for a 2-XOR gate. Generate bitstream. Assign pin44, pin 50, and pin 52 for input X1, input Y1, and output Z1, where $Z1=X1$ XOR Y1.
- ii. Write a Verilog/VHDL code for a 2-AND gate. The output is only 1-bit different. Assign pin 44, pin 50, and pin 52 for input X2, input Y2, and output Z2, where $Z2=X2$ OR Y2.
- iii. Locating logic design: Go to RTL viewer from Tools. Then go to Locate Chip Planner. Make sure that 2-OR gate and 2-XOR gate are in the same location. Also, make sure that they have similar physical I/O pins.
- iv. Generate bitstream for both logic gates.
- v. Read the bitstream file with the given Matlab code bitgen.m.
- vi. Write a code (Matlab/C/python) to compare two bitstreams.

- vii. Compare two bitstreams and measure the hamming distances.
- viii. Find the region where bitstream for Z1 and Z2 are different.
- ix. Change I/O pins and see the difference with reference bitstream generated from step vi. See the difference in the bitstream. What do you see? Explain your observation.

PART III: Bitstream characteristic 2

- i. Generate bitstream for $Z1=X1 \text{ XOR } Y1$.
- ii. Generate bitstream for $Z2=X2 \text{ NAND } Y2$.
- iii. Generate bitstream for $Z1= X1 \text{ XOR } Y1$ and $Z2=X2 \text{ NAND } Y2$ together.
- iv. Calculate bitstream (Z1) OR bitstream (Z2).
- v. Compare bitstream from step iii and iv. Make conclusion.

PART IV: Bitstream characteristic of a sequential circuit

- i. Generate bitstream for a positive edge-triggered flipflop.
- ii. Generate bitstream for a negative edge-triggered flipflop.
- iii. Highlight changes with the conclusion.

References and Further Reading

[1] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/max-10/m10_architecture.pdf