FPGA reverse-engineering challenge

Hackito Ergo Sum
Paris, April 8th-10th 2010

http://lekernel.net
http://www.hackitoergosum.org

FPGA security

- Advertised by manufacturers
- A design cannot be analyzed from the programming file (bitstream)
- Security features built on this assumption: anti cloning, evaluation designs, ...

Why?

- Bitstream format is proprietary and undocumented
- Even with understanding: analysis is difficult
- No encryption! (in most cases)
- Sounds like security through obscurity!
- But it worked so far?!?

A little background...

- Most logic circuits (microprocessor cores, memory controllers, accelerators, …) are just an assembly of combinatorial logic functions and flip-flops (registers)
- An FPGA can be programmed indefinitely to implement *any* of these circuits and connect it to the outside world

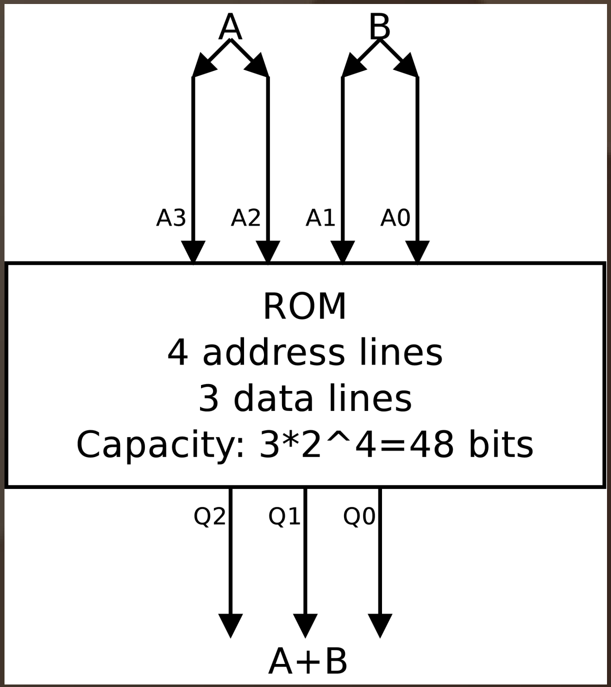
How to build such a reconfigurable device?

Building a reconfigurable architecture

- Memories (ROM, RAM, Flash, …) basically map an address to a word; for example address 0 maps to 5, address 1 maps to 3, etc.
- If you put 0000 (0) on the address pins the output is 0101 (5). If you put 0001 (1), the output is 0011 (3).
- Let's implement combinatorial logic with memory!
- You can implement any function with n inputs and m outputs with a ROM that has n address pins and m data output pins.
- Such a ROM is called a *look-up table* (LUT)

2-bit adder in ROM

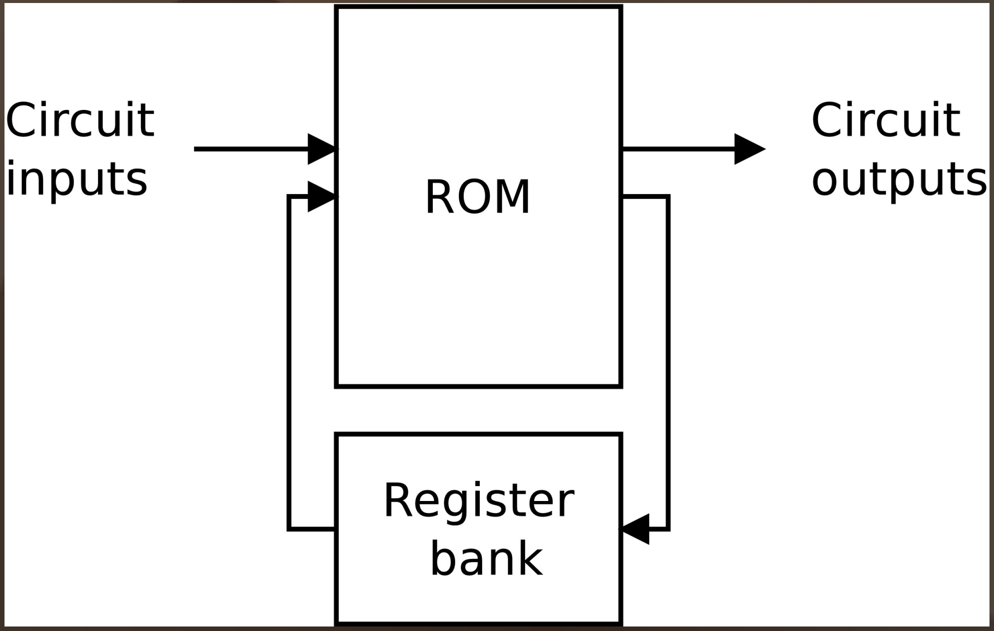
- 0000 → 000
- 0001 → 001
- 0100 → 001
- 0101 → 010
- 1101 → 100 ...



- It's like the addition tables of elementary school...
- Similarly you can implement AND, OR, XOR, any combination of them, etc.

What about the registers?

- Just connect them between some data and address pins...
- This architecture could be as powerful as a FPGA! Is it good?



Building a microprocessor

- 8 8-bit registers, 16-bit address bus, 8-bit data bus
- Address lines: 8*8=64 register inputs, 8-bit data input from the bus. Total 72.
- Data lines: 64 register outputs, 16-bit address, 8-bit data output to the bus. Total 88.
- Required LUT capacity: 88*2^72=47244640256TB
- oops...
- And a real processor is a LOT more complex than this!

But what about simple functions?

- In the previous estimate, the main problem is the big number of the address lines, i.e. the big number of inputs to the logic function in the LUT.
- Indeed, the LUT capacity grows exponentially with the number of inputs.
- If we keep the number of inputs to the logic function low (up to 6-7 in practice), LUTs remain usable.

Combining LUTs together

- A LUT with 4 input and 1 output (4-LUT) can be used to implement any logic function of 4 parameters.
- It contains 16 bits of memory.
- How would you implement *any* function of 5 parameters using 4-LUTs?

Shannon decomposition

```
eval(f, x1, x2, x3, x4, x5)
  if(f = 1)
    then return f(1, x2, x3, x4, x5)
    else return f(0, x2, x3, x4, x5)
```

f1(a, b, c, d) = f(1, a, b, c, d) and f2(a, b, c, d) = f(0, a, b, c, d) are logic functions of 4 parameters!

f(x1, x2, x3, x4) = (x1 & f1(x2, x3, x4)) | (~x1 & f2(x2, x3, x4)))

Needs:

- Two 4-LUT to implement f1 and f2
- One 4-LUT to implement the multiplexer

As a general rule...

- Any logic function is implementable using LUTs combined in this way
- Is it efficient?
- Let C(n) be the cost in 4-LUTs of a function with n inputs
- $C(4) = 1$
- $C(n+1) = 2*C(n) + 1$
- Exponential again!! Won't do better than the big ROM...

But!

- Some functions have better decomposition
- Example: a 7-input AND can be built with two 4-LUT only (instead of 15 using the previous method)
- The output of a LUT can be the input of several LUTs
- There are many other possible optimizations
- Complex problem, still a subject of research

LUT decomposition works in practice

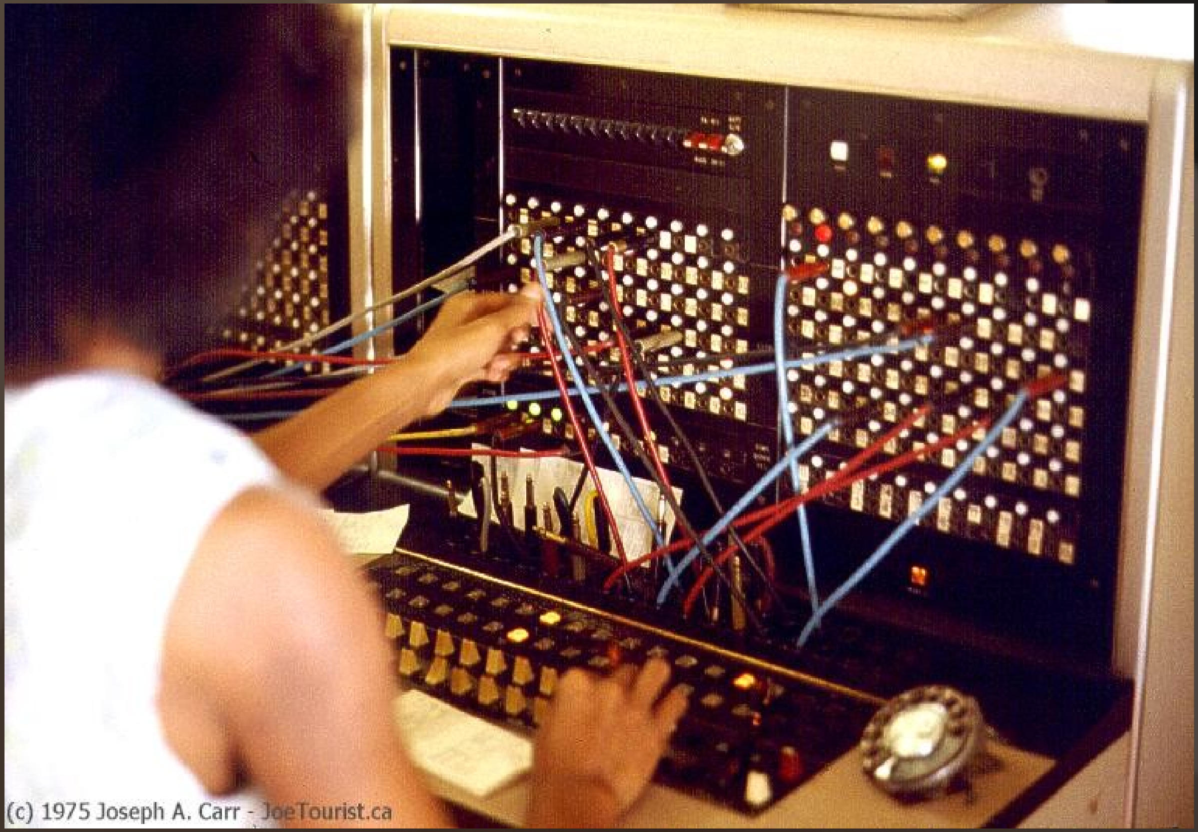
- That's what FPGAs do to implement your logic functions
- Heuristic optimization algorithms
- That's partly why your FPGA “compilations” take so long.
- In an FPGA the registers are distributed
- Each LUT has a D flip-flop at its output that can be enabled or disabled

How many inputs should the LUTs have?

- If the LUT has too few inputs, many of them will be needed to implement a complex logic function
- If the LUT has too many inputs, it will be more costly and a complex logic function could perhaps be broken down into smaller LUTs with better overall efficiency
- 4-LUT (Xilinx Virtex-4, Spartan-3, Altera Cyclone): most simple and common type.

How are the LUTs connected together?

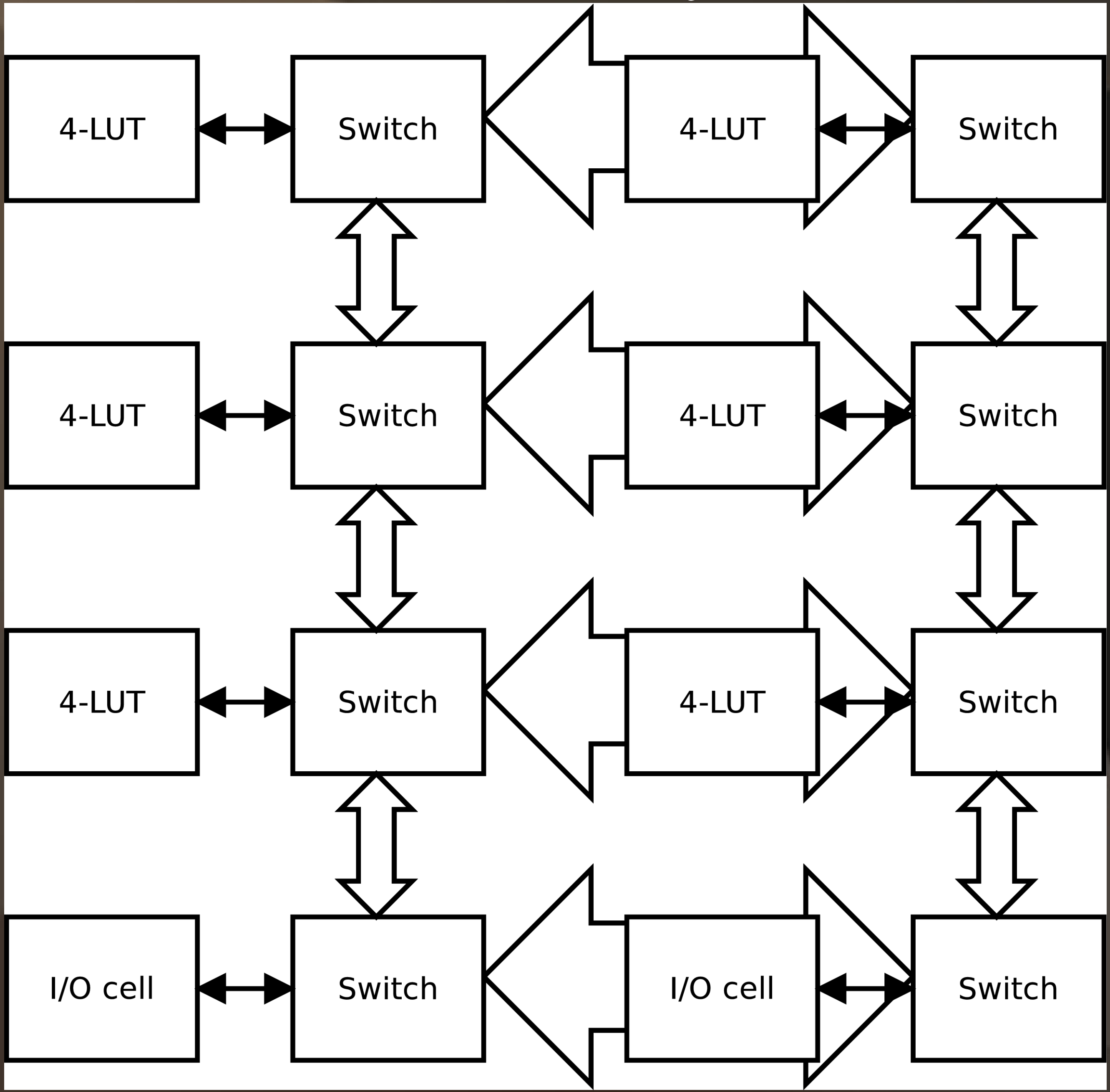
- Programmable interconnect provided by “switch boxes” inside the FPGA
- Similar to a telephone network with switchboards



How is the design connected to the outside world?

- Special FPGA cells are connected to the “telephone-like” network and send the signal to the actual pin on the chip
- Those are called “I/O cells”
- Each physical pin has its dedicated I/O cell which is connected to the network

Summary


4-LUT
Switch
4-LUT
Switch
4-LUT
Switch
4-LUT
Switch
4-LUT
Switch
4-LUT
Switch
I/O cell
Switch
I/O cell
Switch

The build flow

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

Enough theory...

- You know the basic theory behind the operation of FPGAs.
- Xilinx provides a tool called “FPGA Editor” that allows you to manually configure each of these components on their chips.
- Let's see how it works...

Resources

- Icarus Verilog can do some synthesis, but does not work well. http://www.icarus.com/eda/verilog
- Place and route algorithm and open source implementation by the university of Toronto: http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html

  This tool lacks architecture data about real FPGAs.
- Reverse engineering the Xilinx bitstream format (incomplete) http://www.ulogic.org
- FPGA Editor video tutorial http://www.billauer.co.il/xilinx-fpga-editor-video-tutorial-guide.html