Discussion Section 7

Sean Huang

February 19, 2021

Berkeley
UNIVERSITY OF CALIFORNIA

HW3Q3

Problem 3: Bit-stream Reverse Engineering

You have been recruited as a penetration tester for a company manufacturing FPGA-based secure endpoints for a private network. They have asked you to pose as a potential attacker and try to find vulnerabilities in their system. After thinking about the most likely avenues of attack, you decide to pose as a malicious actor who has acquired detailed information on the FPGA that is being used in the endpoints. After analyzing a sample of the device, you were able to determine a few properties of the system.

1. The device contains a collection of N-LUTs (the value of N is part of the mystery). The LUTs are numbered 0, 1, 2, ... Each LUT in the FPGA has the same number of inputs (same N).
2. Each LUT has an output labeled y_i, where i is the LUT number, and inputs labeled x_{i_j}, where i is the LUT number and j is the input number.
3. For programming, the LUTs are connected in a shift register. They are programmed with a configuration bit-stream shifted in from one LUT to the next. Recall from lecture that the configuration bit-stream programs the values of the latches for the truth table.
4. The encryption is a stream cipher. This particular implementation uses several LFSRs, **one of which is relying on some LUTs to perform an XOR operation**. If the XOR LUTs are exposed, it would seriously compromise the security of the device, so the company is very interested in knowing which LUTs are exposed.
5. One of the LUTs is programmed as a simple 2-bit AND gate (*Hint: this will be useful in figuring out the endianness and size of the LUTs.*).
6. Via a side-channel power analysis attack, you were able to determine part of the bitstream, shown here: `0x1111111169969669575757FF699696697F77FFFF`. This bitstream is fed in from right to left (i.e. `F` is fed first and `1` is fed last).

Berkeley
UNIVERSITY OF CALIFORNIA

HW3Q3

- 2-input AND Truth Table
- Consider this as a bitstream
  - 0001 = 0x1
  - 1000 = 0x8

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

HW3Q3

- What about implementing this in a 3-LUT?
  - Repeat the same truth table, only caring about the 2 LSBs
- Bitstream this
  - 00010001 = 0x11

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

HW3Q3

- What about implementing this in a 4-LUT?
  - Repeat the same truth table, only caring about the 2 LSBs
- Bitstream this
  - 0001000100010001 = 0x1111

A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

HW3Q3

- 2-input XOR Truth Table
- Consider this as a bitstream
  - 0110 = 0x6

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

HW3Q3

- What about implementing this in a 3-LUT?
  - Repeat the same truth table, only caring about the 2 LSBs
- Bitstream this
  - 01101001 = 0x69

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

HW3Q3

- What about implementing this in a 4-LUT?
  - Repeat the same truth table, only caring about the 2 LSBs
- Bitstream this
  - 0110100110010110 = 0x6996

A	B	C	D	X
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

HW3Q3

- Original bitstream
  - 0x11111111699696695757 57FF699696697F77FFFF
- String of 1s must be the 2-AND

HW3Q3

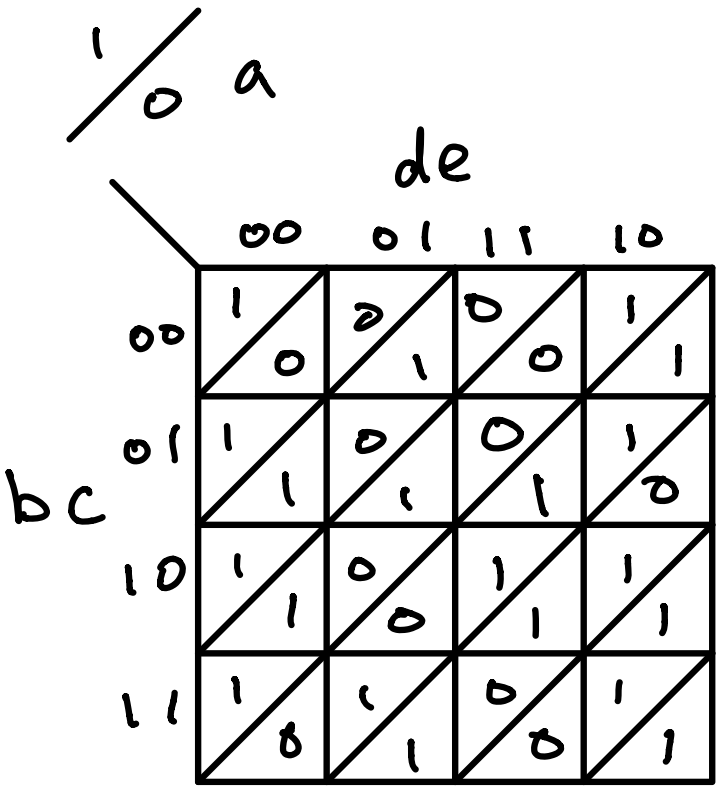
- Original bitstream
  - 0x11111111699696695757757FF699696697F77FFFF
- String of 1s must be the 2-AND
  - 8 hexadecimal digits = 32 bits -> 5-LUT
- 0x69969669 is a 5-input XOR
- Write out truth tables for remaining two operations and simplify to find expressions

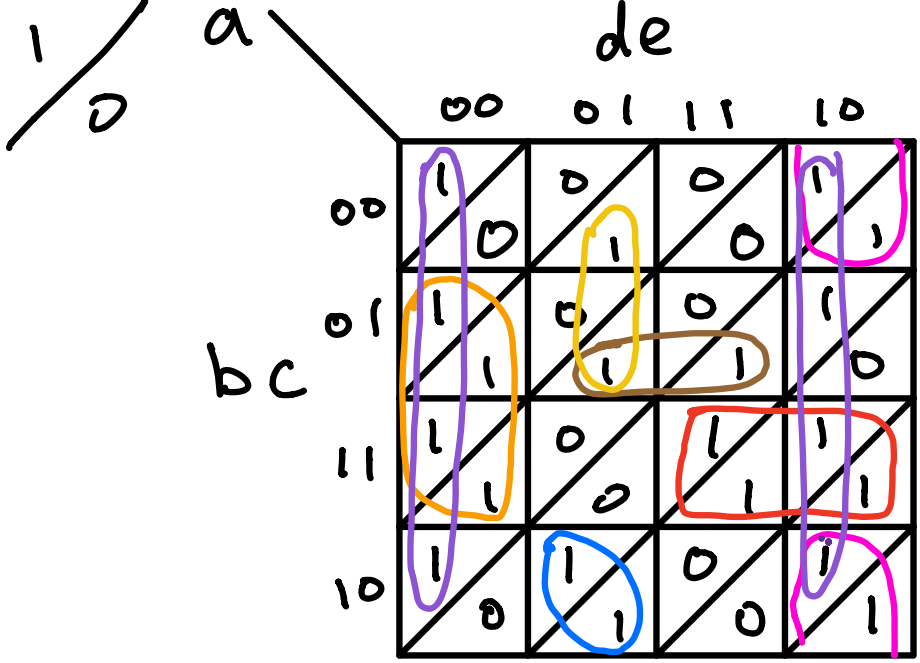
K-map

OAI

Karnaugh Maps

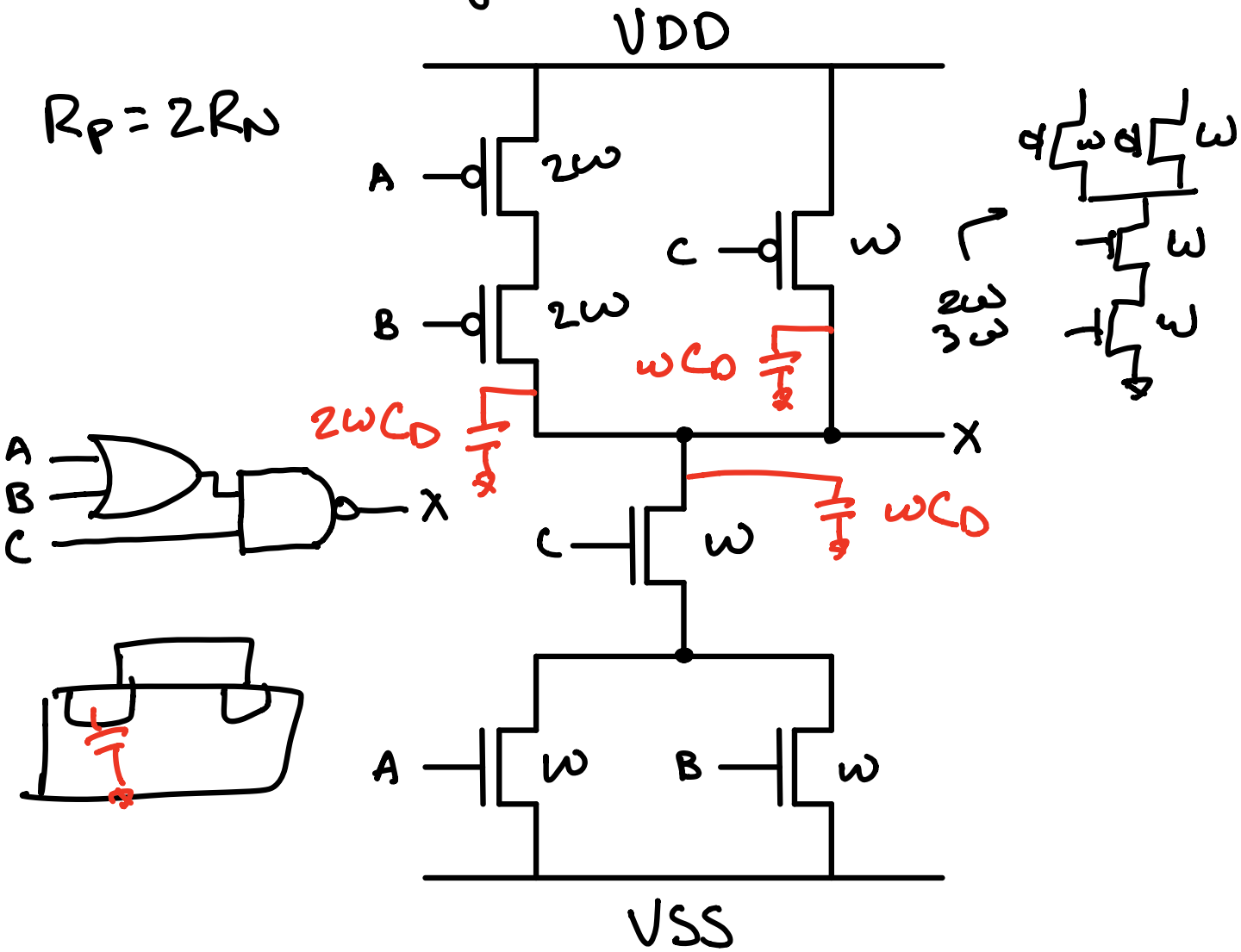
a	b	c	d	e	x
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	0
0	1	1	1	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	1	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	0	1	0
1	1	1	1	0	1
1	1	1	1	1	1



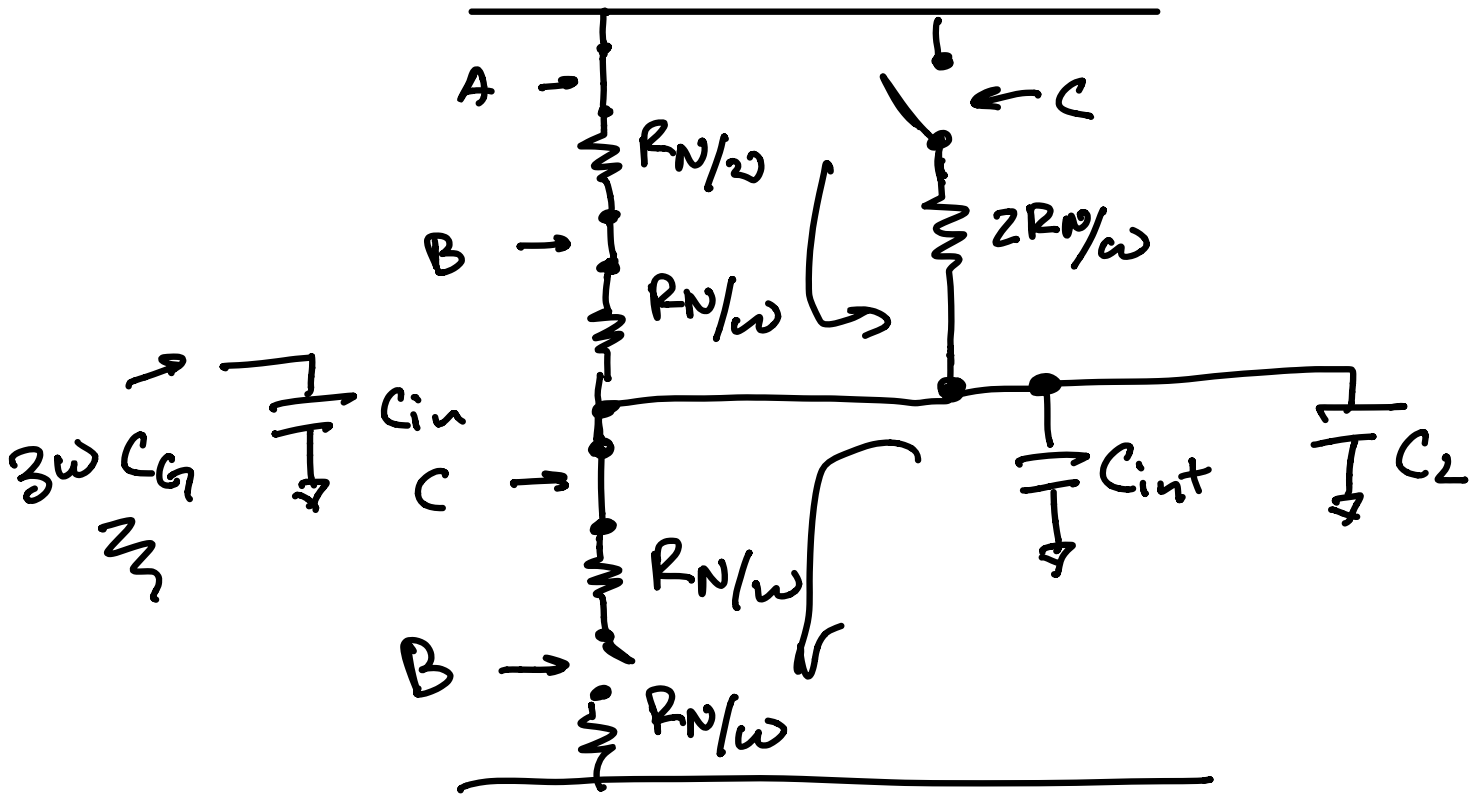


$y = ae' + cd'e' + bcd + c'de' + bc'd'e + a'b'd'e + a'b'ce$

OAI Gate Delay



Consider driving A, B = 0, C = 1



$$C_{int} = 2\omega C_D + \omega C_D + \omega C_D = 4\omega C_D$$

$$\gamma C_G$$

$$t_p = 0.69\, R\, (C_{int}\, C_L) \qquad \gamma$$

$$= 0.69\, 2 \left(R_N/\omega\right)\left(4\omega C_D + C_L\right)$$

$0.69 \frac{R_N}{3}(3\gamma\omega C_G)$

$$= 0.69 \left(\frac{2R_N}{\omega}\right)\left(4\gamma\omega C_G + C_L\right)$$

t_{p0}

$$= 0.69 \left(\frac{2R_N}{\omega}\right) 3\gamma\omega C_G \left(\frac{4}{3} + \frac{C_L}{3\gamma\omega C_G}\right)$$

$f = \frac{C_L}{C_{in}}$

$$= (0.69)2(3\gamma\omega C_G R_N)\left(\frac{4}{3} + \frac{C_L}{\gamma(3\omega C_G)}\right)$$

$$= 2\, t_{p0} \left(\frac{4}{3} + \frac{C_L}{\gamma C_{in}}\right)$$

$$= 2\, t_{p0} \left(\frac{4}{3} + \frac{f}{\gamma}\right)$$