Fast Exhaustive Search for Quadratic Systems in $\mathbb{F}_2$ on FPGAs

Charles Bouillaguet, Chen-Mou Cheng, Tung Chou,
Ruben Niederhagen, Bo-Yin Yang

August 15, 2013

Introduction

The MP Problem

Solving a system of m multivariate polynomial equations in n variables over $\mathbb{F}_q$ is called the *MP problem*.

The MP problem is an **NP-hard problem** even for multivariate *quadratic* systems and $q = 2$.

Introduction

Multivariate Public-Key Cryptography:

e.g. HFE, SFLASH, and QUARTZ

Provably-Secure Stream Ciphers:

e.g. QUAD

Introduction

Multivariate Public-Key Cryptography:

e.g. HFE, SFLASH, and QUARTZ

Provably-Secure Stream Ciphers:

e.g. QUAD

Algebraic Cryptanalysis:

Obtain a system of multivariate polynomial equations with the secret among the variables.

- Naturally breaks the above,
- does not break AES as first advertised,
- but does break, e.g., KeeLoq.

Complexity?

Introduction

Most Efficient Algorithm for $\mathbb{F}_2$:

Brute-force search, testing all 2^n possible inputs.

Previous Work:

On **GPUs** we can solve a quadratic system of

48+ equations in 48 variables in 21min.

Introduction

Most Efficient Algorithm for $\mathbb{F}_2$:

Brute-force search, testing all 2^n possible inputs.

Previous Work:

On **GPUs** we can solve a quadratic system of

48+ equations in 48 variables in 21min.

Research Question:

How would specifically designed hardware perform on this task?

We approach the answer by solving multivariate quadratic systems on reconfigurable hardware (**FPGAs**).

Gray-Code Approach

Full-Evaluation Approach

- Evaluate the whole equation for each possible input.
- Time Complexity: $O(2^n n^2)$
- Memory Complexity: $O(n)$

Gray-Code Approach

Full-Evaluation Approach

- Evaluate the whole equation for each possible input.
- Time Complexity: $O(2^n n^2)$
- Memory Complexity: $O(n)$

Gray-Code Approach

- Only re-compute those parts of the equation that have changed.
- Enumerate input vector in Gray-code order.
- Update solution using the derivatives of the involved variables.
- Time Complexity: $O(2^n m)$
- Memory Complexity: $O(n^2 m)$

Trade computation for memory.

Gray-Code Approach

$k = 01010_b$; $x_4 = 0$, $x_3 = 1$, $x_2 = 0$, $x_1 = 1$, $x_0 = 0$

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 + 1 + 1 + 0 + 1$$

Gray-Code Approach

$k = 01010_b$; $x_4 = 0$, $x_3 = 1$, $x_2 = 0$, $x_1 = 1$, $x_0 = 0$

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 + 1 + 1 + 0 + 1$$

$k = 01011_b$; $x_4 = 0$, $x_3 = 1$, $x_2 = 0$, $x_1 = 1$, $x_0 = 1$

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 + 1 + 1 + 1 + 1$$

Gray-Code Approach

$k = 01010_b$; $x_4 = 0$, $x_3 = 1$, $x_2 = 0$, $x_1 = 1$, $x_0 = 0$

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 + 1 + 1 + 0 + 1$$

$k = 01011_b$; $x_4 = 0$, $x_3 = 1$, $x_2 = 0$, $x_1 = 1$, $x_0 = 1$

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 + 1 + 1 + 1 + 1$$

$k = 01100_b$; $x_4 = 0$, $x_3 = 1$, $x_2 = 1$, $x_1 = 0$, $x_0 = 0$

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 + 1 + 0 + 0 + 1$$

Gray-Code Approach

$k = 01010_b$; $x_4 = 0, x_3 = 1, x_2 = 0, x_1 = 1, x_0 = 0$

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 + 1 + 1 + 0 + 1$$

$k = 01011_b$; $x_4 = 0, x_3 = 1, x_2 = 0, x_1 = 1, x_0 = 1$

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 + 1 + 1 + 1 + 1$$

$k = 01001_b$ in *Gray-code* order

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 1 + 0 + 1 + 1$$

Gray-Code Approach

$k = 01010_b$; $x_4 = 0, x_3 = 1, x_2 = 0, x_1 = 1, x_0 = 0$

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 + 1 + 1 + 0 + 1$$

$k = 01011_b$; $x_4 = 0, x_3 = 1, x_2 = 0, x_1 = 1, x_0 = 1$

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 + 1 + 1 + 1 + 1$$

$k = 01001_b$ in *Gray-code* order

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$
$$f = 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 1 + 0 + 1 + 1$$
$$f = f(01011_b) - 0 \cdot 1 - 1 + 0 \cdot 0 + 0$$

Gray-Code Approach

$k = 01010_b$; $x_4 = 0, x_3 = 1, x_2 = 0, x_1 = 1, x_0 = 0$

$$\begin{aligned} f &= x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1 \\ f &= 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 + 1 + 1 + 0 + 1 \end{aligned}$$

$k = 01011_b$; $x_4 = 0, x_3 = 1, x_2 = 0, x_1 = 1, x_0 = 1$

$$\begin{aligned} f &= x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1 \\ f &= 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 + 1 + 1 + 1 + 1 \end{aligned}$$

$k = 01001_b$ in *Gray-code* order

$$\begin{aligned} f &= x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1 \\ f &= 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 1 + 0 + 1 + 1 \\ f &= f(01011_b) - 0 \cdot 1 - 1 + 0 \cdot 0 + 0 \\ f &= f(01011_b) + \frac{\partial f}{\partial x_1}(01001_b) \end{aligned}$$

Gray-Code Approach

Full-Evaluation Approach

- Evaluate the whole equation for each possible input.
- Time Complexity: $O(2^n n^2)$
- Memory Complexity: $O(n)$

Gray-Code Approach

- Only re-compute those parts of the equation that have changed.
- Enumerate input vector in Gray-code order.
- Update solution using the derivatives of the involved variables.
- Time Complexity: $O(2^n m)$
- Memory Complexity: $O(n^2 m)$

Trade computation for memory.

Xilinx Spartan6 FPGA

Lookup Table (LUT) – LUT-6

Can be seen as

- logic: compute any logical expression in 6 variables,
- ROM: store 64bit, addressed by 6 address ports.

Can be used as two LUT-5 with identical input wires and independent output wires.

Xilinx Spartan6 FPGA

Resources

- 50% SLICEX
  - 4 LUT-6
  - 8 Flip-Flops
- 25% SLICEL
  - \+ wide multiplexers
  - \+ carry logic for large adders
- 25% SLICEM
  - \+ LUT can be used as shift registers
  - \+ LUT can be used as *RAM* sharing the same write address
- Block RAM, DSPs, IO, ...

Gray-Code Algorithm

24: **function** EVAL(s)
25: **while** $s.i < 2^n$ **do**
26: $s.i \leftarrow s.i + 1$;
27: $k_1 \leftarrow \mathrm{BIT}_1(s.i)$;
28: $k_2 \leftarrow \mathrm{BIT}_2(s.i)$;
29: **if** k_2 **valid then**
30: $s.d'[k_1] \leftarrow s.d'[k_1] \oplus s.d''[k_1, k_2]$;
31: **end if**
32: $s.y \leftarrow s.y \oplus s.d'[k_1]$;
33: **if** $s.y = 0$ **then**
34: **return** $\mathrm{shr}(s.i, 1) \oplus s.i$;
35: **end if**
36: **end while**
37: **end function**

Parallelization

Fix i Variables for 2^i Parallel Instances:

$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$

e.g. $i = 2$:

$$f_{00_b} = 0 \cdot x_2 + 0 \cdot x_0 + x_2x_1 + 0 + x_1 + x_0 + 1$$
$$f_{01_b} = 0 \cdot x_2 + 1 \cdot x_0 + x_2x_1 + 1 + x_1 + x_0 + 1$$
$$f_{10_b} = 1 \cdot x_2 + 0 \cdot x_0 + x_2x_1 + 0 + x_1 + x_0 + 1$$
$$f_{11_b} = 1 \cdot x_2 + 1 \cdot x_0 + x_2x_1 + 1 + x_1 + x_0 + 1$$

2^i independent equations (systems)

Parallelization

Fix i Variables for 2^i Parallel Instances:

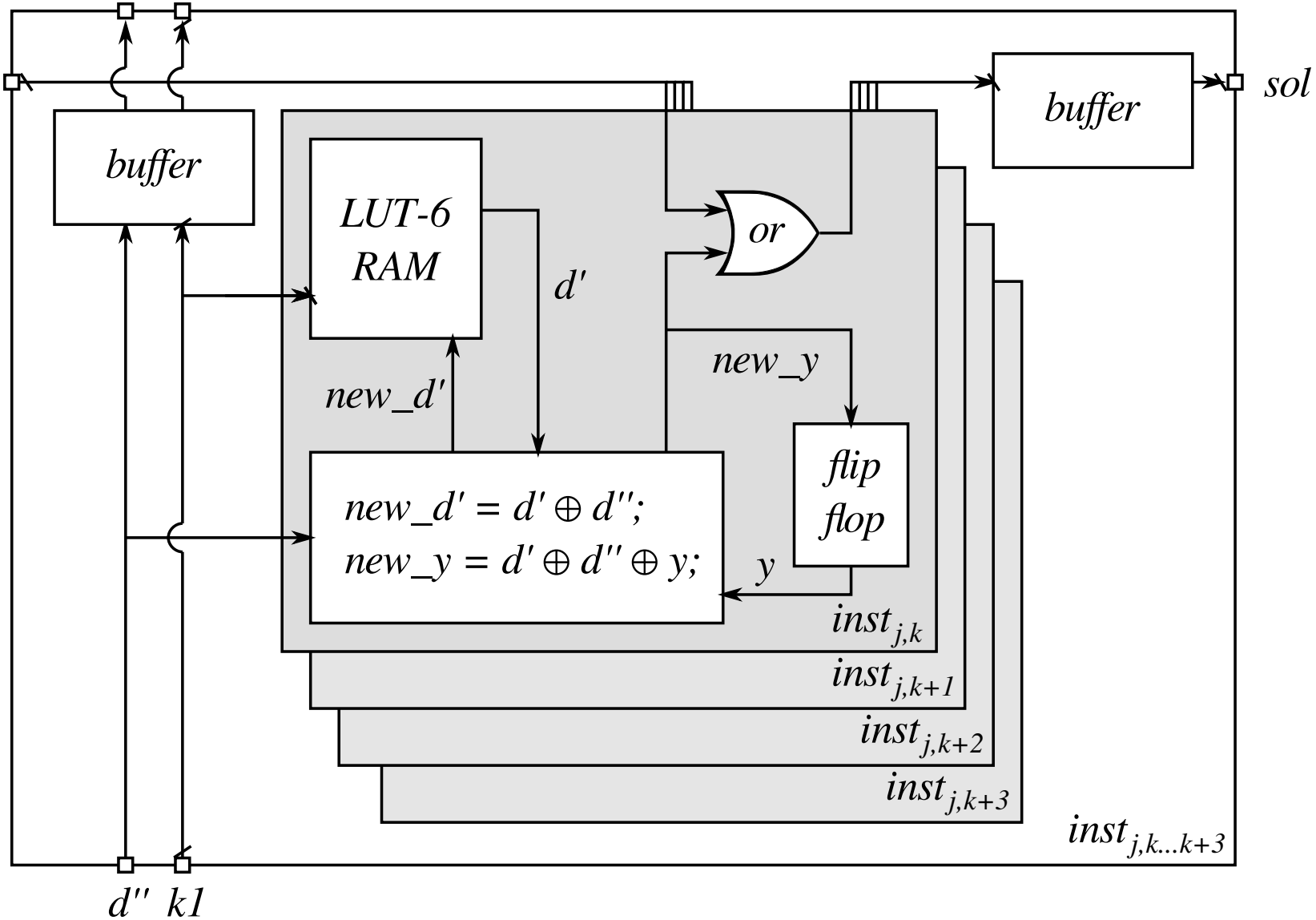
$$f = x_4x_2 + x_3x_0 + x_2x_1 + x_3 + x_1 + x_0 + 1$$

e.g. $i = 2$:

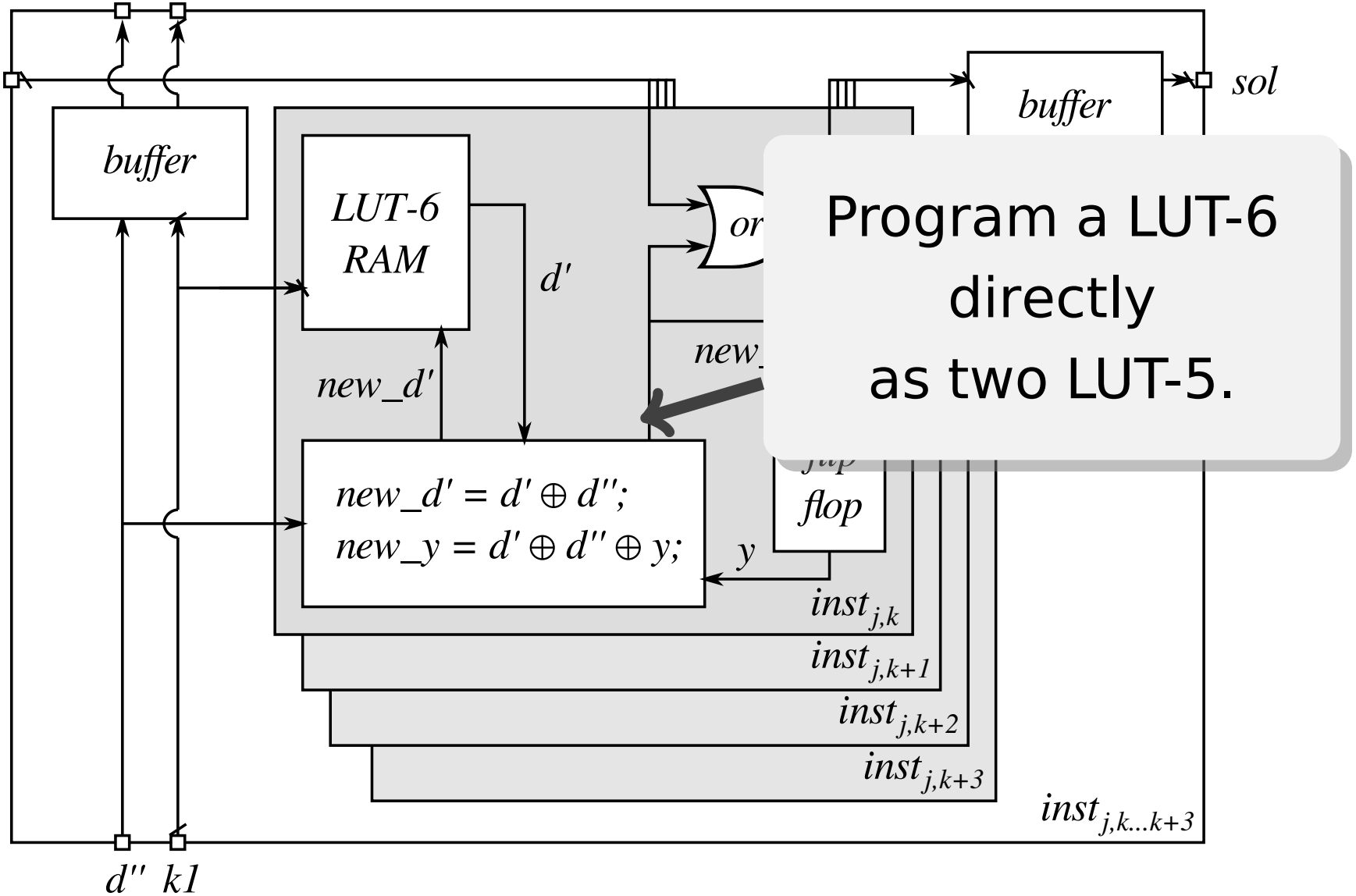
$$\begin{aligned}
f_{00_b} &= 0 \cdot x_2 + 0 \cdot x_0 + x_2x_1 + 0 + x_1 + x_0 + 1 \\
f_{01_b} &= 0 \cdot x_2 + 1 \cdot x_0 + x_2x_1 + 1 + x_1 + x_0 + 1 \\
f_{10_b} &= 1 \cdot x_2 + 0 \cdot x_0 + x_2x_1 + 0 + x_1 + x_0 + 1 \\
f_{11_b} &= 1 \cdot x_2 + 1 \cdot x_0 + x_2x_1 + 1 + x_1 + x_0 + 1
\end{aligned}$$

2^i independent equations (systems)
sharing the *same* quadratic terms!

Instance

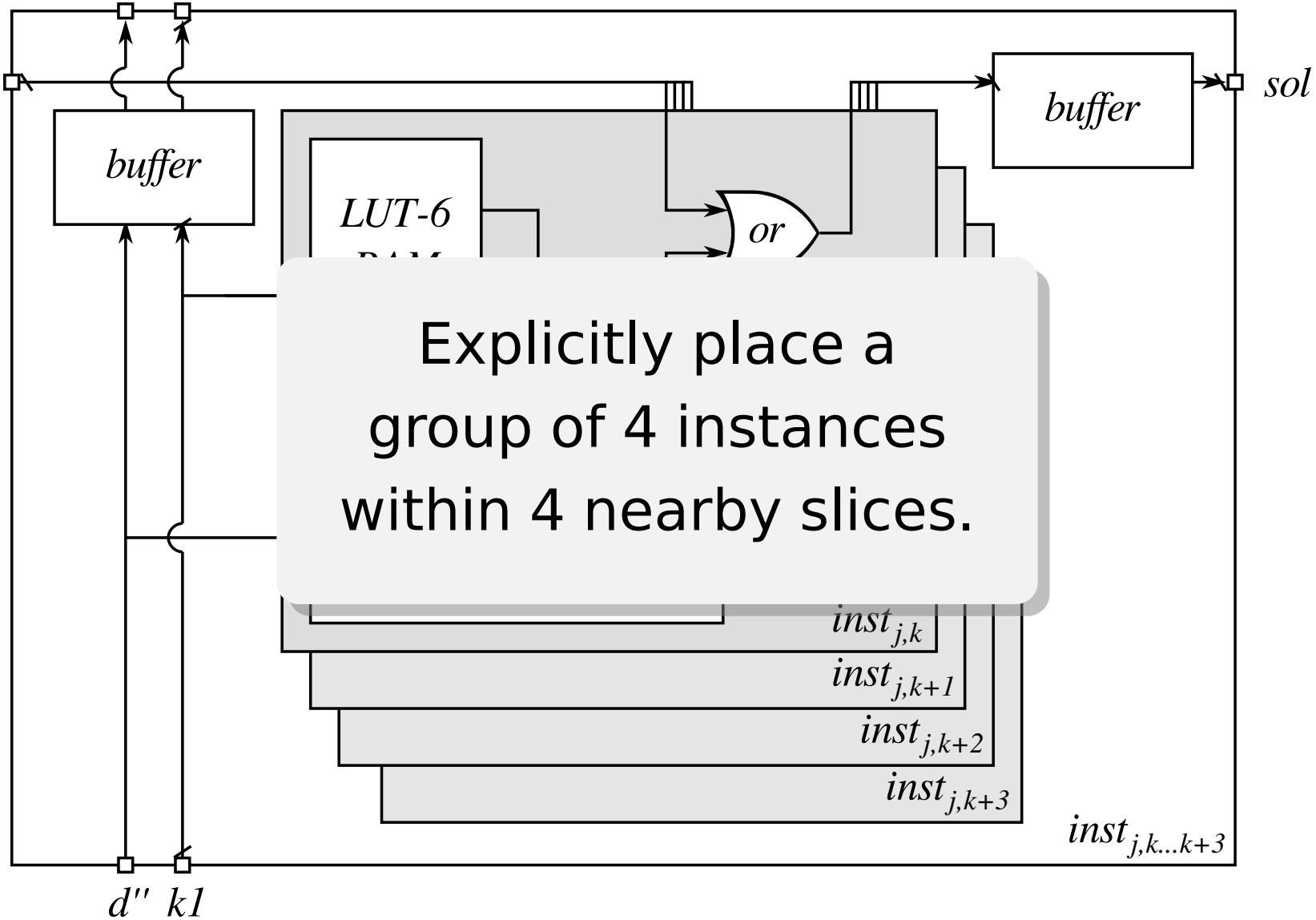

buffer
buffer
sol
LUT-6
RAM
or
d'
new_y
new_d'
flip
flop
new_d' = d' ⊕ d'';
new_y = d' ⊕ d'' ⊕ y;
y
inst j,k
inst j,k+1
inst j,k+2
inst j,k+3
inst j,k...k+3
d''
k1

Instance

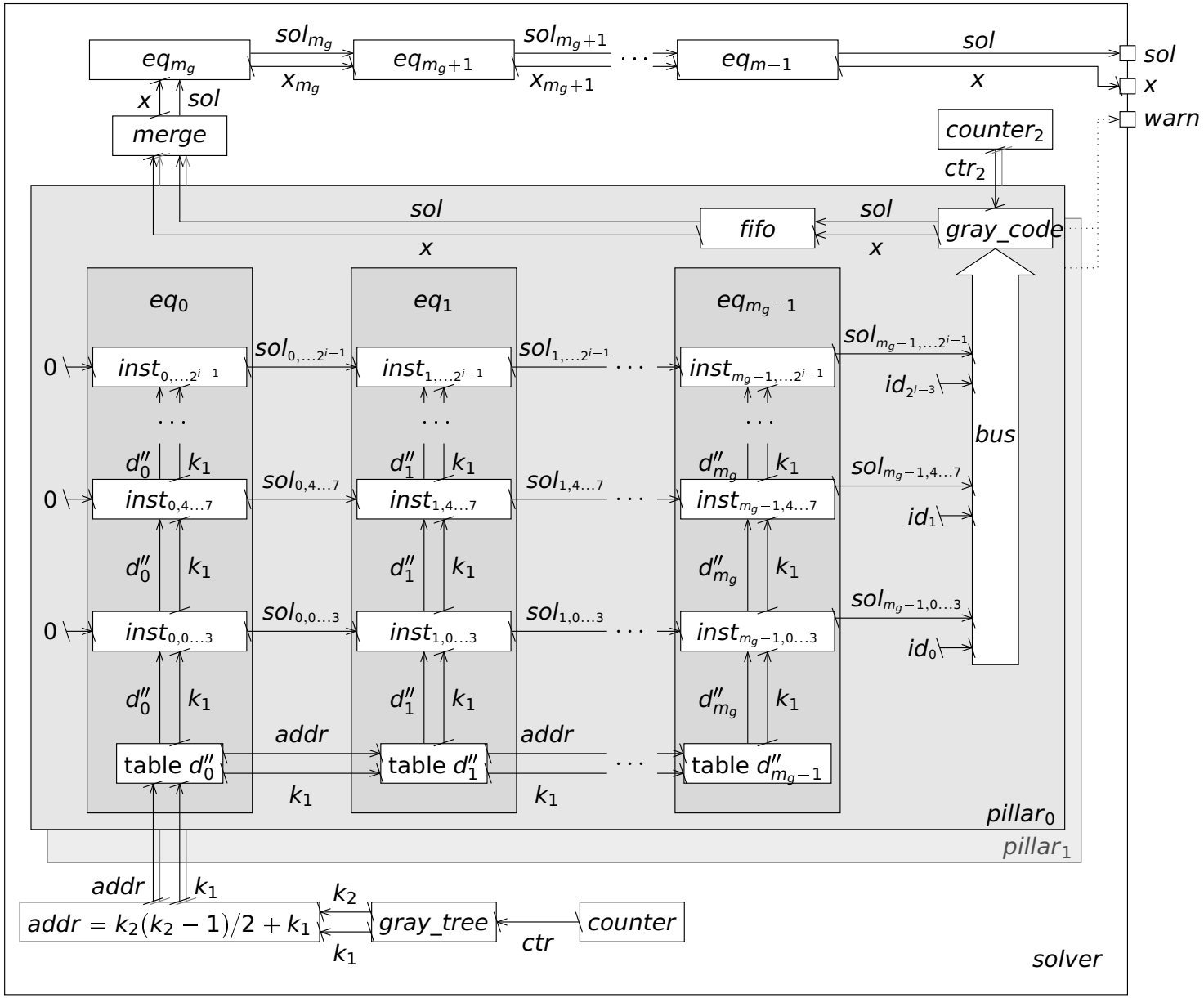


Program a LUT-6 directly as two LUT-5.

Instance


buffer
buffer
sol
LUT-6
or
Explicitly place a group of 4 instances within 4 nearby slices.
$inst_{j,k}$
$inst_{j,k+1}$
$inst_{j,k+2}$
$inst_{j,k+3}$
$inst_{j,k...k+3}$
d''
k1

Overall Architecture



Synthesis Tools

Xilinx Tools tend to fail for those parts of the design with the highest need for efficiency!

Synthesis Tools

Xilinx Tools tend to fail for those parts of the design with the highest need for efficiency!

Development Strategy

We require some kind of low-level, assembly-like HDL:

- Use Python scripts for code generation.
- Assign most of the logic explicitly to LUT-6.
- Fully pipeline the design.
- Explicitly place components to achieve 200MHz.
- Exchange LUT data without resynthesizing.

Synthesis Tools

Xilinx Tools tend to fail for those parts of the design with the highest need for efficiency!

Development Strategy

We require some kind of low-level, assembly-like HDL:

- Use Python scripts for code generation.
- Assign most of the logic explicitly to LUT-6.
- Fully pipeline the design.
- Explicitly place components to achieve 200MHz.
- Exchange LUT data without resynthesizing.

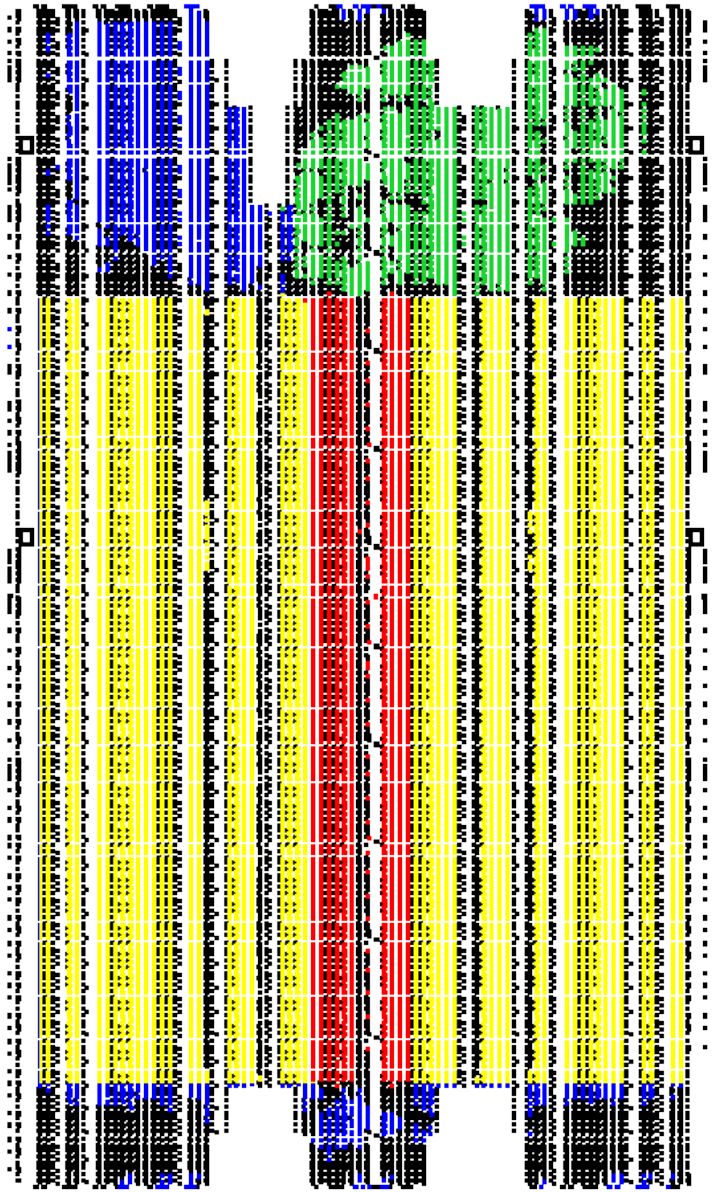
Missing in our tool chain:

- Explicit routing.
- Totally avoid Verilog/VHDL; program low levels directly.

Results

Parameters:

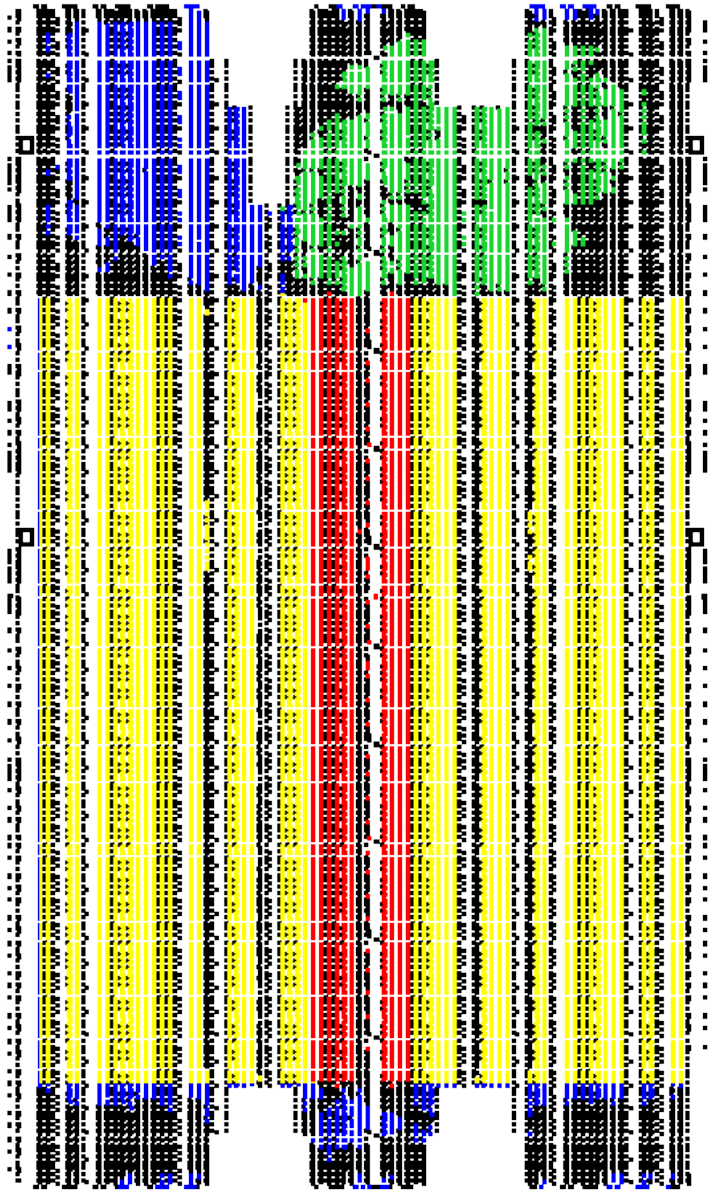
- $n = 54$
- $m = 54$
- 2^{10} instances, two pillars
- $m_g = 12$



Results

Performance:

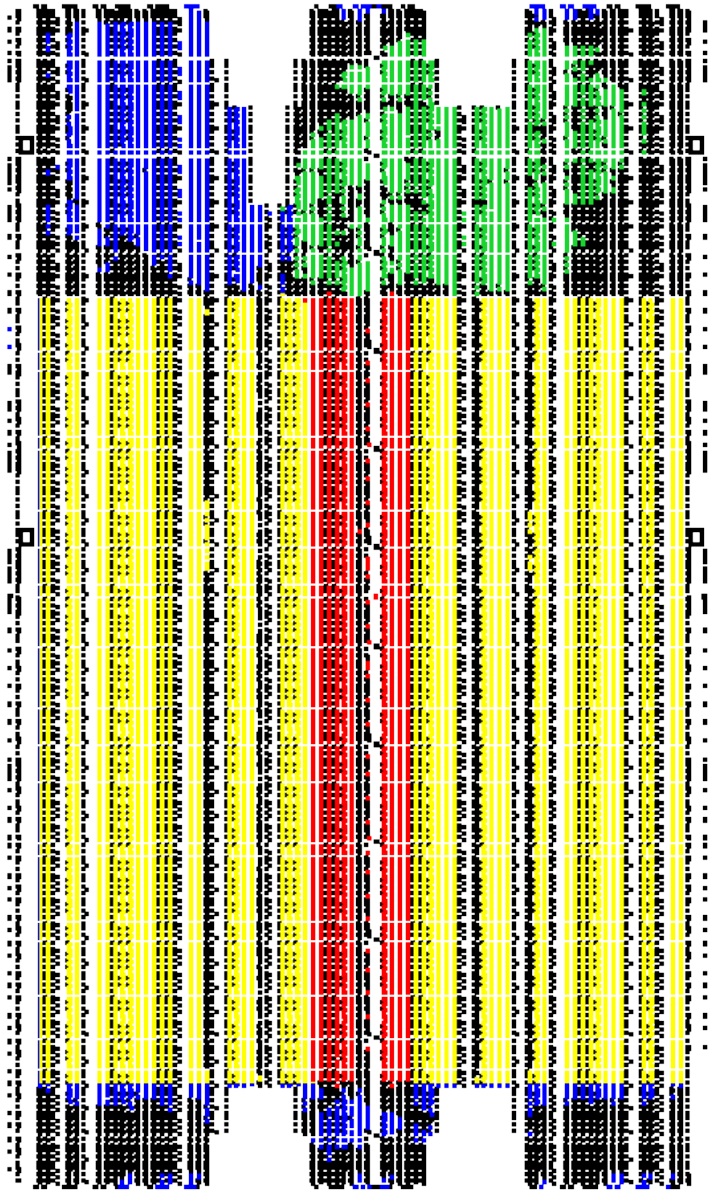
- 200MHz
- 8.8W (GPU: 235W)
- runtime: $2^{54-10}/200\text{MHz} = 24.43\text{h}$
- runtime for $n = 48$: $2^{48-10}/200\text{MHz} = 22.91\text{min}$ (GPU: 21min)
- total energy for $n = 48$: 3,4Wh (GPU: 82.3Wh)



Results

Large FPGA Clusters:

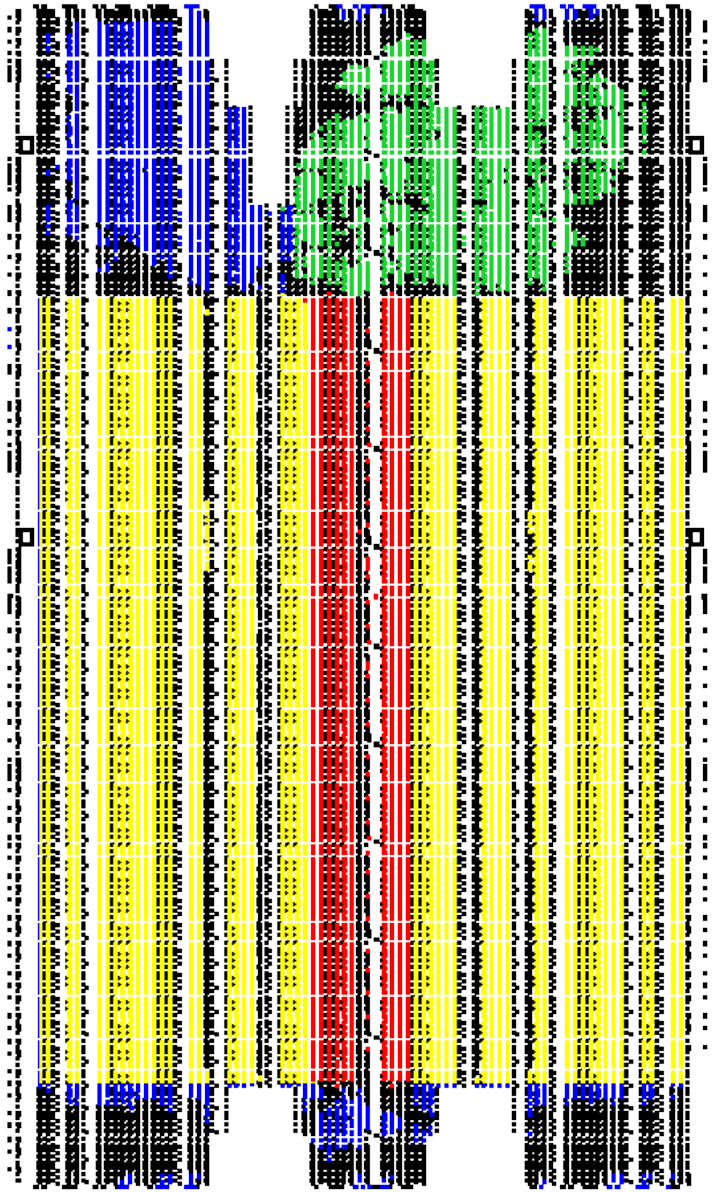
Same routing and placement for any equation system by simply exchanging LUT data.



Results

80-bit Security:

Solving a system of 80 variables requires 1042days on 65,536 Spartan-6 FPGAs at a total cost of about US$40 million.



Questions?