Optimizing Implementations of Lightweight Building Blocks

Jérémy Jean · Thomas Peyrin · Siang Meng Sim · Jade Tourteaux

ANSSI, France
NTU, Singapore

FSE 2018 @ Bruges, Belgium
March 6, 2018
Jeremy.Jean@ssi.gouv.fr
# Hardware Implementations

## Generalities

- **Two principal families:**
  - **FPGA:** Field-Programmable Gate Array
  - **ASIC:** Application Specific Integrated Circuit
- **Several-step process:** design, *synthesis*, place and route, ...
- **Library:** collection of (vectorial) Boolean operations

## Many Different Metrics (not exhaustive)

- **Area.** Size of the implementation (in #transistors)
- **Latency.** Time needed to the evaluation of the circuit
- **Energy.** Number of Joules required for the execution

## Our Goal

Given a function $F$, a library (i.e., collection of gates) and an optimization criteria (e.g., area minimization), find a small implementation of $F$ by interconnecting gates from the library.
Example of Simple Gates

<table>
<thead>
<tr>
<th>Gate</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>$c = a \lor b$</td>
</tr>
<tr>
<td>NOR</td>
<td>$c = \neg(a \lor b)$</td>
</tr>
<tr>
<td>XOR</td>
<td>$c = a \oplus b$</td>
</tr>
<tr>
<td>AND</td>
<td>$c = a \land b$</td>
</tr>
<tr>
<td>NAND</td>
<td>$c = \neg(a \land b)$</td>
</tr>
<tr>
<td>XNOR</td>
<td>$c = \neg(a \oplus b)$</td>
</tr>
</tbody>
</table>

Surface Metric Simplified

We normalize the area of library gates by the one of NAND2 $\Rightarrow$ GE
Ex.: on UMC 180nm, one NOT uses 6.451 $\mu$m$^2$ and NAND2 9.677 $\mu$m$^2$, so $||\text{NOT}|| = 0.67$ GE.

Sizes of Common Gates (in GE)

<table>
<thead>
<tr>
<th>Library</th>
<th>NAND</th>
<th>NOT</th>
<th>XOR</th>
<th>AND</th>
<th>NAND3</th>
<th>XOR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMC 180nm</td>
<td>1.00</td>
<td>0.67</td>
<td>3.00</td>
<td>1.33</td>
<td>1.33</td>
<td>4.67</td>
</tr>
<tr>
<td>TSMC 65nm</td>
<td>1.00</td>
<td>0.50</td>
<td>3.00</td>
<td>1.50</td>
<td>1.50</td>
<td>5.50</td>
</tr>
</tbody>
</table>
Contributions

Our Contribution: LIGHTER

- Synthesis of functions on small domain (4 and 8 bits)
- Graph-based algorithm finding efficient implementations
- Two cases: linear and nonlinear (bijective) functions
- C/C++ code available online:
  http://jeremy.jean.free.fr/pub/fse2018_layerImplementations.tar.gz

<table>
<thead>
<tr>
<th>Linear</th>
<th>Nonlinear</th>
</tr>
</thead>
<tbody>
<tr>
<td>New metric for XOR count</td>
<td>More general than linear case</td>
</tr>
<tr>
<td>Up to 8 bits permutations</td>
<td>Tuned for any HW library</td>
</tr>
<tr>
<td>Several new MDS matrices</td>
<td>Can target SW and HW</td>
</tr>
<tr>
<td>Improved implementations of known matrices</td>
<td>Applications to several 4-bit lightweight Sboxes</td>
</tr>
</tbody>
</table>
Graph-Based Algorithm for Logic Synthesis

Let $\mathcal{B}$ a set of Boolean instructions, e.g.:

$$x \leftarrow x \oplus y \quad x \leftarrow x \lor y \quad x \leftarrow x \land y \quad x \leftarrow \neg x$$

### Graph Modelling

- Let $G = (V, E)$ the weighted DAG with:
  - $V$: the set of all $n$-bit permutations
  - $E$: the set of edges between two vertices linked by an instruction of $\mathcal{B}$
  - an edge using instruction $o \in \mathcal{B}$ is weighted by $||o||$
- Finding a small $\mathcal{B}$-implementation of $F$ w.r.t. to $||.||$ is equivalent to finding (one of) the shortest path $Id \rightarrow F$ in $G$

### Norm Depends on Context

- **Hardware area optimization:** $\forall o \in \mathcal{B}, \quad ||o|| = \#\text{GE needed for } o$
- **Software optimization:** $\forall o \in \mathcal{B}, \quad ||o|| = 1$
  
  all instructions have the same cost $\Rightarrow$ minimize their number
- **FHE or masking:** $||\text{NonLinear}|| = 1000$ and $||\text{Linear}|| = 1$
Algorithm (High-Level)

**Graph Traversal Algorithm**

- Bidirectional Dijkstra’s Algorithm (MITM-like approach)
- Successors of $v \in V$: functions reachable from $v$ by applying instructions in $B$
- Need to invert some edges to get the implementation ($Id \rightarrow F'$)
Instruction Set Heuristic

Examples of Invertible Instructions in $\mathcal{B}$

Consequences

- Easy inversion of the backward subgraph (rooted on $F$)
- Makes the algorithm complexities smaller
- Optimality only proven for implementations using $\mathcal{B}$
- Returned implementation not optimal on any instruction set
Application to Nonlinear Functions
Nonlinear Permutations: Goal and Previous Work

Goal

For a given library, find small circuits of a given nonlinear permutation w.r.t. the area of the final circuit

SAT-Based Approach by [Sto16]

- Encode the minimization problem to SAT
- **Several metrics**: # HW gates, depth, # SW instructions, ...
- **But**: no distinction between the gates (constant norm)
- **Open problem**: Possible to encode any norm in SAT?

Sbox Implementations from Cipher Designers

- PRESENT ASIC implementation by A. Poschmann [BKL+07]
  ⇒ Size: **28 GE** optimized for area and delay (UMC 180nm)
  ⇒ Improved to **22.33 GE** by Osvik [YKPH11]
- PICCOLO [SIH+11] Sbox chosen for low area ⇒ Size: **24 GE** (130nm)
Method of Comparison

Fair Hardware Comparisons are Difficult

- We compare our algorithm to ABC [BM10] (state-of-the-art academic synthesizer)
- For a given Sbox, we perform synthesis in three cases:
  - Using our algorithm from the table description (LUT)
  - Using ABC from the LUT
  - Using ABC from the netlist generated by our algorithm
- Optimization: area only
- Libraries: UMC 180nm and TSMC 65nm

Notes on Instructions Used and Libraries

- Let $\mathcal{A}$ the set of all the gates from a given library
- Our algorithm: invertible combinations of gates in a subset of $\mathcal{A}$
- When running ABC, we let it use all gates in $\mathcal{A}$
- So: netlists from our algorithm can only be smaller on full $\mathcal{A}$
## Nonlinear Permutations: Results on UMC 180nm

<table>
<thead>
<tr>
<th>Sbox</th>
<th>ABC (from LUT)</th>
<th>Ours (from LUT)</th>
<th>ABC (from ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PICCOLO</td>
<td>21.00 GE</td>
<td>13.00 GE</td>
<td>no improvement</td>
</tr>
<tr>
<td>SKINNY</td>
<td>22.33 GE</td>
<td>13.33 GE</td>
<td>no improvement</td>
</tr>
<tr>
<td>TWINE</td>
<td>26.33 GE</td>
<td>21.67 GE</td>
<td>no improvement</td>
</tr>
<tr>
<td>PRESENT</td>
<td>24.33 GE</td>
<td>21.33 GE</td>
<td>no improvement</td>
</tr>
<tr>
<td>Rectangle</td>
<td>25.33 GE</td>
<td>18.33 GE</td>
<td>no improvement</td>
</tr>
<tr>
<td>LBlock $S_0$</td>
<td>20.33 GE</td>
<td>16.33 GE</td>
<td>no improvement</td>
</tr>
</tbody>
</table>

## Remarks

- **PRESENT**: 21.33 GE, smallest known to date (used in [JMPS17]), however long critical path
- **PICCOLO**: the original NOR/XOR is replaced by OR/XNOR (3 transistors saved)
### Results on UMC 180nm: PICCOLO

|-------------------------------|-------------------------------|

![Diagram of PICCOLO Sbox - 14 GE](image1)

![Diagram of PICCOLO Sbox - 13 GE (new)](image2)

---

11/16

Jérémy Jean (ANSSI) / Optimizing Implementations of Lightweight Building Blocks
Application to Linear Functions
Linear Permutations: How To Count XORs

A Simple XOR Count: \#XOR = 8 (d-XOR)

\[
\begin{bmatrix}
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
b_3 \\
b_2 \\
b_1 \\
b_0
\end{bmatrix}
= 
\begin{bmatrix}
b_3 \oplus b_2 \oplus b_0 \\
b_3 \oplus b_2 \oplus b_1 \\
b_3 \oplus b_2 \oplus b_1 \oplus b_0 \\
b_3 \oplus b_1
\end{bmatrix}
\]

An Improved XOR Count: \#XOR = 4 (s-XOR)

\[
\begin{bmatrix}
b_3 \\
b_2 \\
b_1 \\
b_0
\end{bmatrix}
\rightarrow
\begin{bmatrix}
b_1 \\
b_2 \\
b_0 \\
b_3
\end{bmatrix}
\rightarrow
\begin{bmatrix}
b_1 \\
b_2 \\
b_0 \\
b_3 \oplus b_1
\end{bmatrix}
\rightarrow
\begin{bmatrix}
b_1 \\
b_2 \\
b_0 \\
b_3 \oplus b_1
\end{bmatrix}
\rightarrow
\begin{bmatrix}
b_3 \oplus b_2 \oplus b_1 \\
b_3 \oplus b_2 \oplus b_1 \oplus b_0 \\
b_3 \oplus b_1
\end{bmatrix}
\rightarrow
\begin{bmatrix}
b_3 \oplus b_2 \oplus b_1 \\
b_3 \oplus b_2 \oplus b_1 \oplus b_0 \\
b_3 \oplus b_1
\end{bmatrix}
\]
## Linear Permutations: Results

### Local Optimization

We applied the graph algorithm to the linear permutations of GF(2^c) corresponding to field multiplications: \( x \rightarrow \alpha x, \alpha \in GF(2^c) \)

### Results for Field Elements

- Found **optimal s-XOR implementations** up to 12 XORs for matrix dimensions up to 8 (memory limitations)
- For more than 12, we provide a **heuristic** (non-optimal results)
- With more heuristics, we can consider **higher dimensions** (e.g., dimension 32 with the AES matrix, see next talk)

### Results for MDS Matrices

- Optimized s-XOR field elements \( \Rightarrow \) search for **new MDS matrices**
- For instance, for \( 4 \times 4 \) involutory matrices over GF(2^4):

  \[
  \begin{bmatrix}
  2 & 1 & 1 & 9 \\
  1 & 4 & f & 1 \\
  d & 9 & 4 & 1 \\
  1 & d & 1 & 2
  \end{bmatrix}
  \]

  is **s-XOR optimal**
Conclusion and Future Works

Versatile Tool LIGHTER

- For crypto implementers (HW/SW): Finds efficient implementations of known functions
- For cipher designers: Helps choose building blocks
- For special crypto applications: FHE, masking, ...

The tool can be parameterized to many different scenarios

Future Works

- More complex hardware gates (easy)
- Optimize for delay (easy/moderate)
- Search for new building blocks (no MITM) (moderate)
- Probabilistic version of the algorithm (moderate)
- Remove (or relax) heuristic on invertible instructions (hard)
Thank you for your attention!

http://jeremy.jean.free.fr/pub/fse2018_layer_implementations.tar.gz