Logic Synthesis and Automation for Memristive Memory Processing Unit

Shahar Kvatinsky
Viterbi Faculty of Electrical Engineering
Technion – Israel Institute of Technology

EPFL Workshop on Logic Synthesis and Emerging Technologies
September 2017
The External Memory Wall Problem
von Neumann (Architecture) Bottleneck

A bottleneck for both throughput and power!
And an Energy Bottleneck

<table>
<thead>
<tr>
<th>Operation (16-bit operand)</th>
<th>Energy/Op (45 nm)</th>
<th>Cost (vs. Add)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add operation</td>
<td>0.18 pJ</td>
<td>1X</td>
</tr>
<tr>
<td>Load from on-chip SRAM</td>
<td>11 pJ</td>
<td>61X</td>
</tr>
<tr>
<td><strong>Send to off-chip DRAM</strong></td>
<td><strong>640 pJ</strong></td>
<td><strong>3,556X</strong></td>
</tr>
</tbody>
</table>

Processing “In-Memory” (PIM)
Reducing Data Movement
Processing “In-Memory” (PIM) Reducing Data Movement

Prior Art

90’s
- Configuration PIM machine
- Active Pages
- SA connected to SIMD pipeline
- Automata Memory

Recent

CMOS Processing Units (PUs)

Data transfer is still required to/from DRAM and PUs

Real Computing within the Memory
Beyond von Neumann Architecture

Input Device → CPU

Control Unit
Arithmetic/Logic Unit

Memory

Output Device

Memory Processing Unit (MPU)
mMPU: Solving the von Neumann Bottleneck

Moving from DRAM to memristive memory

mMPU: performing computation **USING** the memristive memory cells
Agenda

• The need for non-von Neumann architectures
• **Memristive technologies**
• Memristive MPU (mMPU) architecture
• mMPU logic synthesis and automation
• Summary
Memristors
Emerging Nonvolatile Memory Technologies

Resistive RAM (RRAM)
- SanDisk
- SONY
- HP
- winbond
- Panasonic
- TOSHIBA
- Crossbar

STT MRAM
- EVERSPIN TECHNOLOGIES
- HITACHI
- Crocus Technology
- TOSHIBA
- QUALCOMM
- SAMSUNG

Phase Change Memory (PCM)
- SAMSUNG
- IBM
- STI
- Micron
- SK hynix
Memristor – Memory Resistor
Resistor with Varying Resistance

Decrease resistance
Agenda

• The need for non-von Neumann architectures
• Memristive technologies
• Memristive MPU (mMPU) architecture
• mMPU logic synthesis and automation
• Summary
MAGIC – Memristor Aided LoGIC

Example of MAGIC NOR

Initialize OUT to $R_{ON}$

$R_{ON} = \text{Logic ‘1’}$

$R_{OFF} = \text{Logic ‘0’}$

<table>
<thead>
<tr>
<th>IN$_1$</th>
<th>IN$_2$</th>
<th>NOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Increase resistance

$R_{OFF} >> R_{ON}$

Real MAGIC

Jung et al., Nano Research, July 2017

Bae et al., Nano Letters (accepted)

ASIC² winbond
Our lab (HfOx based)
MAGIC NOR in a Crossbar
MAGIC NOR in a Crossbar
MAGIC NOR in a Memristive Memory

Parallelism
SIMD

Hierarchy of Logical Functions

Matrix multiplication

MUL
POW
ADD
NOR
AND
COPY
NAND

Convolution

SQRT
DIV

MAGIC - NOR

Complete logic family
mMPU µArchitecture

R. Ben-Hur and S. Kvatsinsky, "Memory Processing Unit for In-Memory Processing," NANOARCH 2016
mMPU μArchitecture

R. Ben-Hur and S. Kvavinsky, "Memory Processing Unit for In-Memory Processing," NANOARCH 2016
mMPP μArchitecture

R. Ben-Hur and S. Kvatinsky, "Memory Processing Unit for In-Memory Processing," NANOARCH 2016
mMPU μArchitecture

R. Ben-Hur and S. Kvatinsky, "Memory Processing Unit for In-Memory Processing," NANOARCH 2016
Issues Involved in mMPU Architecture

- Memory Design
- mMPU Controller Design and Optimization
- Periphery Design
- Programming Model
- Software
- Real-PIM-System
- CPU
- mMPU Controller
- mMPU
- Applications
Agenda

• The need for non-von Neumann architectures

• Memristive technologies

• Memristive MPU (mMPU) architecture

• mMPU logic synthesis and automation

• Summary
mMPU Controller
Design and Automation

• Supports regular memory operations

• Optimized logic flow:
  – Parallelism (in-array and banks)
  – Cost function (latency, area, energy)

• Real-time memory mapping
Exploit Parallelism

- Reducing the number of gates is not enough
- Mapping determines the possible parallelism
SIMPLE MAGIC

Synthesis and In-memory Mapping of Logic Execution for Memristor-Aided LoGIC

- Both reducing the number of gates and mapping into the memristive memory

SIMPLE MAGIC

Customized standard cell library (.genlib)

module ckt(
  :
  :
endmodule

Logic function (.blif)

GATE inv
GATE nor2

ABC Synthesis Tool

NOR and NOT netlist (.v)

In-memory computation constraints

Performance Optimization

Spatial independent execution sequence

Real-time Address Mapping

Addresses constraints

Location specific execution sequence
SIMPLE MAGIC

Customized standard cell library (.genlib)

Logic function (.blif)

module ckt
GATE inv
GATE nor2
.
.
endmodule

ABC Synthesis Tool

NOR and NOT netlist (.v)

In-memory computation constraints

Performance Optimization

Spatial independent execution sequence

Real-time Address Mapping

Addresses constraints

Location specific execution sequence
Variables

• For every gate $j$:
  – Locations (rows/columns):
    • $C_{Aj}, C_{Bj}, C_{Ej}, R_{Aj}, R_{Bj}, R_{Ej}$
  – Clock cycle of execution: $T_j$
Optimization Function

\[ Latency_{\text{best mapping}} = \min \{ \max_j \{ T_j \} \} \]
Constrains - Locations

- I/Os of each gate have to be located in the same row and different columns, or vice versa

- For every gate $j$:
  $$[(C_{A_j} = C_{B_j} = C_{E_j}) \cap (R_{A_j} \neq R_{B_j} \neq R_{E_j})] \cup [(C_{A_j} \neq C_{B_j} \neq C_{E_j}) \cap (R_{A_j} = R_{B_j} = R_{E_j})]$$
Constrains - Locations

- Simultaneous execution of different gates only when they are aligned in the rows/columns

\[
A_j \\
B_j \\
E_j
\]

\[
A_k \\
B_k \\
E_k
\]
SIMPLE MAGIC

Customized standard cell library (.genlib)

Logic function (.blif)

module ckt(
  .
  .
endmodule

GATE inv
GATE nor2
  
ABC Synthesis Tool

NOR and NOT netlist (.v)

In-memory computation constraints

Performance Optimization

Spatial independent execution sequence

Real-time Address Mapping

Addresses constraints

Location specific execution sequence
# 1 Bit Full Adder

## Spatial Independent Execution Sequence

**Diagram:**

```
\[
\begin{array}{c}
A_j \\
B_j \\
\end{array}
\rightarrow
\begin{array}{c}
j \\
E_j \\
\end{array}
\]
```

<table>
<thead>
<tr>
<th>row</th>
<th>column 1</th>
<th>column 2</th>
<th>column 3</th>
<th>column 4</th>
<th>column 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td><strong>C\text{in}</strong></td>
<td><strong>A1</strong></td>
<td><strong>E1</strong></td>
</tr>
<tr>
<td>2</td>
<td><strong>A</strong> A2</td>
<td><strong>B</strong> A3</td>
<td></td>
<td><strong>E7</strong> B8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>E11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>E9 B11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>B9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A A5</td>
<td>B B5</td>
<td></td>
<td>E6 A7 A9</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>E2 A4</td>
<td>E3 B4</td>
<td>E5 B6</td>
<td>E4 A6 A10</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>E8 B10 A11 A12</td>
<td>E12 <strong>C\text{out}</strong></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>E10 S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Time of execution:**

- Cycle 1
- Cycle 2
- Cycle 3
- Cycle 4
- Cycle 5
- Cycle 6
- Cycle 7
- Cycle 8
- Cycle 9
- Cycle 10
SIMPLE MAGIC

Customized standard cell library (.genlib)

Logic function (.blif)

module ckt

GATE inv
GATE nor2

endmodule

ABC Synthesis Tool

NOR and NOT netlist (.v)

In-memory computation constraints

Performance Optimization

Spatial independent execution sequence

Real-time Address Mapping

Addresses constraints

Location specific execution sequence
1 Bit Full Adder
Location Specific Execution Sequence

- occupied cells
- available cells

- Location for 1Bit Full Adder
  (10 rows X 5 columns)
Experimental Results

# Computation steps

Benchmarks

5xp1  clip  cm150a  cm162a  cm163a  misex1  parity  x2

- Chakraborti et al. [1]
- Original
- ABC
- SIMPLE
Agenda

• The need for non-von Neumann architectures
• Memristive technologies
• Memristive MPU (mMPU) architecture
• mMPU logic synthesis and automation

• Summary
mMPU – Huge Potential for Real Processing In-Memory

• Orders of magnitude better performance & energy

• SIMPLE MAGIC – mMPU controller design

• Current work:
  – Reducing the running time of SIMPLE
  – Extending the optimization functions
  – Real-time mapping