Lake Memory Generator and SMT for Automated Memory Configuration

Maxwell Strange, Kavya Sreedhar, Nestan Tsiskaridze

AHA! Retreat
7/30/20
Evolution of Memory Tile

- Memory Tile
  - Line buffer
  - Line + Double Buffer
  - Unified Buffer

- Unified Buffer Implementation
  - Efficiency, cheaper ports, cleaner
  - Match the compiler
  - Lake
From Humble Beginnings… (Jade)

• For image applications
  • Only supports line buffer

• Uses two banks of single-width memory
• To provide simultaneous read and write for a FIFO
Need Double Buffering Too

- Add address generators to previous design
  - Use each bank as a buffer
  - Can map line buffer on this as well
Unified Buffer

- Data Streams In
- Data Streams Out

Clock Cycle
Data
Valid

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>3</td>
<td>-</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Unified Buffer
Wide Fetch Width

![Diagram showing the components of a wide fetch width scheme, including aggregation buffer, storage buffer (SRAM), and transpose buffer.]
Lake – From Perl to Python (May 2020 tape-out)

• Memory Generator
  • Parameterizable
    • Ports, # banks, fetch width, etc.
  • Modes
    • RAM
    • FIFO
    • Unified Buffer
• Chaining
  • Creating larger memories
  • Creating more ports
Garnet Memory Tile
## Synthesis Breakdown

<table>
<thead>
<tr>
<th>Module</th>
<th>General Idea</th>
<th>Area</th>
<th>% of Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interconnect</td>
<td>CB + SB</td>
<td>3324 μm²</td>
<td>15.8%</td>
</tr>
<tr>
<td>Tile_MemCore/MemCore</td>
<td>Everything in the tile other than the interconnect</td>
<td>17676 μm²</td>
<td>84.2%</td>
</tr>
<tr>
<td>... Logic</td>
<td>All logic in the memory core</td>
<td>8304 μm²</td>
<td>... 46.97%</td>
</tr>
<tr>
<td>... Configuration Space</td>
<td>Configuration registers and configuration bus demuxing</td>
<td>3856 μm²</td>
<td>... 21.81%</td>
</tr>
<tr>
<td>... SRAM Macros</td>
<td>2x 512x32 SRAM Macros</td>
<td>5516 μm²</td>
<td>... 31.2%</td>
</tr>
<tr>
<td>Tile_MemCore</td>
<td>Entire memory tile – interconnect, core logic, SRAM Macros, configuration space</td>
<td>21000 μm²</td>
<td>100%</td>
</tr>
</tbody>
</table>
# Main Memory Core Logic Synthesis Breakdown

<table>
<thead>
<tr>
<th>Module</th>
<th>General Idea</th>
<th>Area</th>
<th>% of Total MemCore Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO Mode</td>
<td></td>
<td>675 μm²</td>
<td>8.12%</td>
</tr>
<tr>
<td>SRAM Mode</td>
<td></td>
<td>95 μm²</td>
<td>1.14%</td>
</tr>
<tr>
<td>Unified Buffer Mode</td>
<td></td>
<td>7268 μm²</td>
<td>87.5%</td>
</tr>
<tr>
<td>... Accessors</td>
<td>Drive read + write to memory components</td>
<td>1966 μm²</td>
<td>... 27%</td>
</tr>
<tr>
<td>... Aggregation Buffers</td>
<td>Serial-in Parallel-out</td>
<td>800 (2 * 400) μm²</td>
<td>... 11%</td>
</tr>
<tr>
<td>... SRAM address generators</td>
<td>2x input + 2x output address generators (6 nested for loops, 16bit)</td>
<td>2620 (655 * 4) μm²</td>
<td>... 36%</td>
</tr>
<tr>
<td>... Transpose Buffers</td>
<td>Parallel-in Serial-out</td>
<td>800 (400 * 2) μm²</td>
<td>... 11%</td>
</tr>
<tr>
<td>Tile_MemCore/Memcore Logic</td>
<td>All the logic in the memory core</td>
<td>8304 μm²</td>
<td>100%</td>
</tr>
</tbody>
</table>
## Diet Lake

<table>
<thead>
<tr>
<th>Module</th>
<th>Area – Pre-Optimization</th>
<th>Area – Post Optimization</th>
<th>Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Space</td>
<td>3856 μm²</td>
<td>3174 μm²</td>
<td>17.6%</td>
</tr>
<tr>
<td>Unified Buffer Mode</td>
<td>7268 μm²</td>
<td>2983 μm²</td>
<td>58.9%</td>
</tr>
<tr>
<td>... Accessors</td>
<td>1966 μm²</td>
<td>378 μm²</td>
<td>80.77%</td>
</tr>
<tr>
<td>... SRAM address generators</td>
<td>2620 μm²</td>
<td>900 μm²</td>
<td>65.648%</td>
</tr>
<tr>
<td>Tile_MemCore/Memcore Logic</td>
<td>8304 μm²</td>
<td>4019 μm²</td>
<td>51.6%</td>
</tr>
</tbody>
</table>
Are We Done?

No

• Don't have a single source of truth
  • We talk with Qiaoyi (Joey) all the time about the hardware
  • Had to hand write the functional model (again, and again, and …)

• Don't really have a DSL
  • More like a set of parameters
Lake, the Next Generation

1. Develop Lake primitives that can be connected in a user-specified graph (the new Lake specification)

2. Leverage hwtypes & Magma
   - For functional model and HW SST
   - For Lake specification with Graph, Node, Edge classes
Tapeout Design: 2 input ports, 2 output ports.
Btor2 format allows specifying word-level model checking problems.

**Pono**: our Performant, Adaptable, and Extensible SMT-based Model Checker (formerly Cosa2).
Configuration Constraints

- Avoid non-determinism;
- Prune the search space;
- Data exchange only.
How far can we scale?

Memory Tile reads start @54
Timeout = 20 min

Better…?

Conv33
32x32 image
Intel Xeon E5-2620 v4, CPU 1.6GHz, 16GB.
Can we utilize more information from users/designers?
Internal sequence obtained from the polyhedral analysis.
Btor2

; BTOR description generated by Yosys 0.9+1706
for module LakeTop.
1 sort bitvec 16
2 input 1 addr_in[0]
3 input 1 addr_in[1]
4 input 1 chain_data_in[0]
5 input 1 chain_data_in[1]
6 sort bitvec 1
7 input 6 chain_idx_input
8 input 6 chain_idx_output
9 sort bitvec 2
10 input 9 chain_valid_in
11 input 6 clk
12 input 6 clk_en
13 sort bitvec 8

Configuration Constraints

Tile Interface/
Configuration Register List

Configuration Solver

Memory Tile Sequence

SRAM Sequence
How far can we scale?

Memory Tile reads start @54
SRAM reads start @51
Timeout = 20 min

Better…?
Btor2

; BTOR description generated by Yosys 0.9+1706 for module LakeTop.
1 sort bitvec 16
2 input 1 addr_in[0]
3 input 1 addr_in[1]
4 input 1 chain_data_in[0]
5 input 1 chain_data_in[1]
6 sort bitvec 1
7 input 6 chain_idx_input
8 input 6 chain_idx_output
9 sort bitvec 2
10 input 9 chain_valid_in
11 input 6 clk
12 input 6 clk_en
13 sort bitvec 8

Configuration Constraints

Tile Interface/
Configuration Register List

Address Configuration Registers

SRAM Sequence

Memory Tile Sequence

Configuration Solver
How far can we scale?

Memory Tile reads start @54
SRAM reads start @51
Timeout = 20 min

Better…?
New Modular Design

Memory Tile in Pono

- Aggregation Buffer
- Storage Buffer (SRAM)
- Transpose Buffer

Input: @0 @1 @2
Output: @0 @1 @2

Configuration Registers
# How far can we scale with a modular approach?

<table>
<thead>
<tr>
<th>Image</th>
<th>#Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>8x8</td>
<td>71</td>
</tr>
<tr>
<td>16x16</td>
<td>263</td>
</tr>
<tr>
<td>20x20</td>
<td>407</td>
</tr>
<tr>
<td>24x24</td>
<td>583</td>
</tr>
<tr>
<td>32x32</td>
<td>1031</td>
</tr>
<tr>
<td>40x40</td>
<td>1607</td>
</tr>
<tr>
<td>44x44</td>
<td>1943</td>
</tr>
<tr>
<td>48x48</td>
<td>2311</td>
</tr>
</tbody>
</table>

**Identity stream**

- Memory Tile reads start @8
- 2.8 GHz Quad-Core
- Intel Core i7, 16 GB

![Diagram with columns representing #Cycles for different Image sizes](image)
How far can we scale with a modular approach?

Identity stream

Memory Tile reads start @8
2.8 GHz Quad-Core
Intel Core i7, 16 GB

Propagating configuration solutions from agg & tb to sram.
Towards More Agile Design

- More complex and larger designs;
- More information to utilize from inside the design;
- More modular control over different modules of the design – each module controlled through their configuration registers.

Tool Summary

- A **Configuration Solver** based on **Pono** and **SMT** that is *fully automated* and *flexible*.
- The approach works!