A Digital Flow for Asynchronous VLSI Systems: Status Update

Udit Agarwal\textsuperscript{1}, Samira Ataei\textsuperscript{2}, Jiayuan He\textsuperscript{1}, Wenmian Hua\textsuperscript{2}, Yi-Shan Lu\textsuperscript{1}, Sepideh Maleki\textsuperscript{1}, Yihang Yang\textsuperscript{2}, Keshav Pingali\textsuperscript{1}, Rajit Manohar\textsuperscript{2}

\textsuperscript{1}University of Texas at Austin, \textsuperscript{2}Yale University

November 5, 2020 at WOSET 2020
A little bit about me...

• Yi-Shan Lu
  • PhD student at CS, UT Austin
  • Advisor: Prof. Keshav Pingali

• Research interests
  • Parallelization & language design for domain-specific computation
  • Current focus: EDA algorithms, timing analysis & simulation

• Selected honors
  • Graph Challenge Champion, HPEC 2017
  • Third Place Award, TAU Contest 2019
  • Second Place, CADathlon at ICCAD 2019
  • Participation Award, TAU Contest 2020
Asynchronous design flow at a glance

Updates

• Cyclone [1]
  • Asynchronous timer & power analyzer

• BiPart
  • Deterministic parallel hypergraph partitioner

• Dali [2]
  • A gridded cell placer

• AMC [4]
  • Asynchronous memory compiler

Cyclone\textsuperscript{[1]}: Comprehensive async timing & power analyzer

- Need to analyze cycles explicitly for asynchronous circuits
- Functionality enhancement
  - Supports QDI circuits with data & bundled-data logic timing constraints
  - Power analysis integrated with timer
- Performance improvement
  - Faster timing graph creation by exploiting module hierarchy
  - Effectively parallelized using Galois
    - Steady slew & delay computation
    - Longest-path forest construction in critical cycle ratio algorithm
    - Timing propagation
    - Timing constraint checking

Cyclone [1]: Performance on large circuits

<table>
<thead>
<tr>
<th>Name</th>
<th># pins</th>
<th>$p^*$ (ns)</th>
<th>M</th>
<th>Power (mW)</th>
<th>Max. cycle ratio</th>
<th>Full performance analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>YTO (s) (#t)</td>
<td>CPLEX (s)</td>
</tr>
<tr>
<td>bd203</td>
<td>495</td>
<td>0.443</td>
<td>1</td>
<td>0.521</td>
<td>0.010 (01)</td>
<td>0.010</td>
</tr>
<tr>
<td>s5387</td>
<td>88,292</td>
<td>4.388</td>
<td>3</td>
<td>22.602</td>
<td>0.969 (14)</td>
<td>3.090</td>
</tr>
<tr>
<td>ac97_ctrl</td>
<td>650,709</td>
<td>3.785</td>
<td>3</td>
<td>190.356</td>
<td>8.486 (21)</td>
<td>60.390</td>
</tr>
<tr>
<td>vga_lcd</td>
<td>5,689,435</td>
<td>7.046</td>
<td>1</td>
<td>911.437</td>
<td>100.112 (49)</td>
<td>2,267.920</td>
</tr>
</tbody>
</table>

Faster performance characterization by better cycle ratio algorithm

6-20X self speedup through parallelization for large designs

BiPart: Deterministic parallel hypergraph partitioner

Initial partitioning

Coarsening

G₀

G₁

G₂

G₃

Refinement

G₀

G₁

G₂

G₃

2X faster by using CSR format

15% cut size reduction by multiple scheduling policies for merging nodes
## BiPart: Comparison w/ Zoltan

<table>
<thead>
<tr>
<th>Graph</th>
<th># Nodes</th>
<th># Hedges</th>
<th># Edges</th>
<th>BiPart (1) (sec)</th>
<th>BiPart (14) (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random-15M</td>
<td>15.0M</td>
<td>17.0M</td>
<td>280.6M</td>
<td>431.9</td>
<td>64.85</td>
</tr>
<tr>
<td>Random-10M</td>
<td>10.0M</td>
<td>15.0M</td>
<td>115.0M</td>
<td>198.6</td>
<td>35.07</td>
</tr>
<tr>
<td>WB</td>
<td>9.8M</td>
<td>6.9M</td>
<td>57.2M</td>
<td>20.89</td>
<td>7.32</td>
</tr>
<tr>
<td>NLPK</td>
<td>3.5M</td>
<td>3.5M</td>
<td>96.8M</td>
<td>17.49</td>
<td>5.88</td>
</tr>
<tr>
<td>Xyce</td>
<td>1.9M</td>
<td>1.9M</td>
<td>9.5M</td>
<td>3.20</td>
<td>0.94</td>
</tr>
<tr>
<td>Circuit1</td>
<td>1.9M</td>
<td>1.9M</td>
<td>8.9M</td>
<td>2.90</td>
<td>0.98</td>
</tr>
<tr>
<td>Leon2</td>
<td>62.7K</td>
<td>1.7M</td>
<td>6.8M</td>
<td>1.89</td>
<td>1.14</td>
</tr>
<tr>
<td>webbase</td>
<td>1.1M</td>
<td>800.8K</td>
<td>2.4M</td>
<td>0.67</td>
<td>0.46</td>
</tr>
<tr>
<td>Sat14</td>
<td>1.0M</td>
<td>1.0M</td>
<td>3.1M</td>
<td>58.76</td>
<td>9.90</td>
</tr>
<tr>
<td>RM07</td>
<td>381.7K</td>
<td>381.7K</td>
<td>37.5M</td>
<td>3.08</td>
<td>0.92</td>
</tr>
</tbody>
</table>

- **Zoltan quality**
- **Zoltan speed**
- No points dominated by Zoltan
- 6-7X self speedup

**Zoltan cannot finish Random-15M (OoM)**
Dali [2]:
A gridded cell placer

Dali: A gridded cell placement flow
Core problem: weighted wire-length optimization

- Removes cell overlaps
- Avoid large cell displacement
- Create mini-rows
- Clean design rule violations related to N/P-wells
- Create N/P-wells & place well tap cells
- Connect VDD/GND pins to power supply

Design → Global placement → Forward-backward legalization → Well legalization → Power grid design → Layout

Existing techniques
• Model wire-length as a quadratic function
• Obtain a rough placement

Dali\textsuperscript{[2]}: Comparison to standard-cell methodology

### Placement region treatment

<table>
<thead>
<tr>
<th>Properties</th>
<th>Standard cell</th>
<th>Gridded cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows in the placement region</td>
<td>Predefined/Static</td>
<td>Dynamic</td>
</tr>
<tr>
<td>N/P-well</td>
<td>Preplaced</td>
<td>Mini-row-based</td>
</tr>
<tr>
<td>Well-tap cells</td>
<td>Preplaced</td>
<td>Mini-row-based</td>
</tr>
</tbody>
</table>

### Placement flow

<table>
<thead>
<tr>
<th>Placement stage</th>
<th>Standard cell placer</th>
<th>Dali</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global placement</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Detailed placement</td>
<td>Yes</td>
<td>Within mini-row after WL</td>
</tr>
<tr>
<td>Legalization</td>
<td>Align to rows</td>
<td>Align to routing grid</td>
</tr>
<tr>
<td>Well-legalization</td>
<td>Implicit by abutment/filler cells</td>
<td>Mini-row construction</td>
</tr>
<tr>
<td>Power routing</td>
<td>Implicit by abutment/filler cells</td>
<td>Placement-based</td>
</tr>
</tbody>
</table>

Dali [2]: Used to tape out chips in 65nm process

AMC \cite{Ataei2019}:
Asynchronous memory compiler

- Memory compiler
  - Enables automatic generation of memory layouts to minimize the development costs of ASIC & processor designs

- Asynchronous memory compiler (AMC)
  - The first open-source memory compiler that generates Asynchronous pipelined SRAMs with high throughput and best-case latency
  - Provides GDSII layout, SPICE netlist, LEF and LIB files, and Verilog model of SRAM for variable size and configurations
  - v1.0 available on GitHub w/ a reference implementation for SCMOS: https://github.com/asyncvlsi/AMC

AMC[^4]:
Features

• **Supported memory functions**
  • Three types of operations: read, write & read-modify-write
  • Synchronous interface
  • SRAM BIST (built-in self test) based on March C-algorithm
  • Power-gating option
  • Write-masking option

• **Ease of use**
  • Support for different memory cell layouts & different bank orientations
  • Portable to new technology nodes; successful on 0.5um, 65nm, 28nm & 12nm FINFET technologies

AMC \textsuperscript{[4]}: Power-gating option

- Power-gating provides two operation modes for asynchronous SRAMs:
  - SLEEP or low power mode
  - WAKE-UP or active mode

(a) Ring-style power-gating and (b) Daisy-chain SLEEP signal distribution

---

\textsuperscript{[4]} S. Ataei, R. Manohar. AMC: An asynchronous memory compiler. In \textsc{Async} 2019.
AMC \[^{[4]}\]: Write-masking option

- Write-masking determines the data bits to write during the memory write mode
  - When the write mask pin \( k \) is high (WM\([k]\) =1), the corresponding data bit (DIN\([k]\)) is selected, and its data is written to the memory
  - When the write mask pin is low, no data is written for that bit and memory cell retains its previous value

32KB SRAM in 12nm FinFET technology (a) without write-masking \(1x\) and (b) with write-masking \(1.04x\)

Conclusions

• Updates in our async design tool chain
  • Cyclone, asynchronous timer & power analyzer
  • BiPart, deterministic parallel hypergraph partitioner
  • Dali, gridded cell placer
  • AMC, asynchronous memory compiler

• Future works
  • Make the flow timing driven
  • Improve the flow in terms of QoR & runtime
  • Support for more async logic families
Thanks!

Visit our GitHub repository at http://github.com/asyncvlsi/