Towards Layout-Friendly High-Level Synthesis

Jason Cong  UCLA
Bin Liu  UCLA
Guojie Luo  Peking University
Raghu Prabhakar  UCLA
Outline

- High-level synthesis and layout-friendly architecture
- Evaluation of the impact of high-level decisions
- Evaluation of metrics for scheduling/binding
- Conclusion
High-Level Synthesis

- Synthesis as a model refinement process
- Mature RTL-to-layout flow today
- Behavior model: one level above RTL
  - C/C++/SystemC/Matlab, etc.
- High-level synthesis
  - Untimed behavioral model to cycle-accurate RTL
  - Typically: C to Verilog
A Typical Synthesis Flow from Behavior Level

Compiler transformation
• Program -> CDFG

Scheduling
• CDFG -> FSMD

Binding
• FSMD -> RTL

RTL Synthesis, P&R …

$t_1 = a + b;$
$t_2 = c \times d;$
$t_3 = e + f;$
$t_4 = t_1 \times t_2;$
$z = t_4 - t_3;$
A Short History of High-Level Synthesis

- 1980s—early 1990s: research and prototype
  - Synopsys Behavioral Compiler, etc.
  - Mostly from behavioral VHDL/Verilog
- Late 1990s: early commercialization
  - C-based languages (C/C++/SystemC) as input
  - AutoESL (Xilinx), Cadence, Forte, Mentor (Calypto), NEC, Synfora (Synopsys), Synopsys
  - Growing interest driven by design complexity and time-to-market pressue
xPilot: Behavioral-to-RTL Synthesis Flow [SOCC’2006]

- Behavioral spec. in C/C++/SystemC
- Frontend compiler
- SSDM
- RTL + constraints
- FPGAs/ASICs

**Advanced transformation/optimizations**
- Loop unrolling/shifting/pipelining
- Strength reduction / Tree height reduction
- Bitwidth analysis
- Memory analysis …

**Core behavior synthesis optimizations**
- Scheduling
- Resource binding, e.g., functional unit binding
- Register/port binding

**μ-Arch-generation & RTL/constraints generation**
- Verilog/VHDL/SystemC
- FPGAs: Altera, Xilinx
- ASICs: Magma, Synopsys, …
AutoPilot Compilation Tool (based UCLA xPilot system)

- Platform-based C to FPGA synthesis
- Synthesize pure ANSI-C and C++, GCC-compatible compilation flow
- Full support of IEEE-754 floating point data types & operations
- Efficiently handle bit-accurate fixed-point arithmetic
- More than 10X design productivity gain
- High quality-of-results

Developed by AutoESL, acquired by Xilinx in Jan. 2011
AutoPilot Results: Sphere Decoder (from Xilinx)

- Wireless MIMO Sphere Decoder
  - ~4000 lines of C code
  - Xilinx Virtex-5 at 225MHz
- Compared to optimized IP
  - 11-31% better resource usage

<table>
<thead>
<tr>
<th>Metric</th>
<th>RTL Expert</th>
<th>AutoPilot Expert</th>
<th>Diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>32,708</td>
<td>29,060</td>
<td>-11%</td>
</tr>
<tr>
<td>Registers</td>
<td>44,885</td>
<td>31,000</td>
<td>-31%</td>
</tr>
<tr>
<td>DSP48s</td>
<td>225</td>
<td>201</td>
<td>-11%</td>
</tr>
<tr>
<td>BRAMs</td>
<td>128</td>
<td>99</td>
<td>-26%</td>
</tr>
</tbody>
</table>

TCAD April 2011 (keynote paper)
“High-Level Synthesis for FPGAs: From Prototyping to Deployment”
AutoPilot Results: DQPSK Receiver (from BDTI)

- Application
  - DQPSK receiver
  - 18.75 Msamples @ 75 MHz clock speed
- Area better than hand-coded

<table>
<thead>
<tr>
<th>Application</th>
<th>Hand-coded RTL</th>
<th>AutoPilot</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQPSK receiver</td>
<td>5.9%</td>
<td>5.6%</td>
</tr>
</tbody>
</table>

Xilinx XC3SD3400A chip utilization ratio (lower the better)

BDTi evaluation of AutoPilot
http://www.bdti.com/articles/AutoPilot.pdf
AutoPilot Results: Optical Flow (from BDTI)

- **Application**
  - Optical flow, 1280x720 progress scan
  - Design too complex for an RTL team

- **Compared to high-end DSP:**
  - 30X higher throughput, 40X better cost/fps

<table>
<thead>
<tr>
<th>Chip Unit Cost</th>
<th>Highest Frame Rate @ 720p (fps)</th>
<th>Cost/performance ($/frame/second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xilinx Spartan3ADSP XC3SD3400A chip</td>
<td>$27</td>
<td>183</td>
</tr>
<tr>
<td>Texas Instruments TMS320DM6437 DSP processor</td>
<td>$21</td>
<td>5.1</td>
</tr>
</tbody>
</table>

BDTi evaluation of AutoPilot
http://www.bdti.com/articles/AutoPilot.pdf
Impact on Quality of Result

◆ Big impact on QoR due to drastically different architectures
  ▪ Parallel/sequential/pipelined
  ▪ Different ways to map operations to control states
  ▪ Different ways to share functional units/registers/interconnects

◆ Opportunity to select from multiple possible implementations
  ▪ Instead of struggling with a sub-optimal RTL
  ▪ Need metrics/models to decide which implementation is superior

◆ Performance/throughput/area can be estimated reasonably well in HLS
  ▪ Frequency/congestion is quite hard
  ▪ Some RTL structures lead to long interconnect delay after layout
Interconnect Estimation: the Challenge

- Estimation of interconnect timing and congestion is hard at a high level
  - Long wires/congestion occur during layout

- Incorporate layout in synthesis?
  - Reasonable, but time consuming.
  - May not be necessary if we just want to estimate if one solution is better than the other
  - Try to get the more layout-friendly solution

- In this work
  - Experimentally evaluate the impact of HLS decisions on congestion
  - Evaluate some possible metrics without doing layout
Outline

- High-level synthesis and layout-friendly architecture
- Evaluation of the impact of high-level decisions
- Evaluation of metrics for scheduling/binding
- Conclusion
# Experiment Setup

- **Varying strategies in HLS**
  - Impacts of compiler transformation and synthesis engine (scheduling & binding) evaluated separately
  - 5 DSP benchmarks (lots of multiplication/addition, simple or no control flow) for synthesis engine

## Binding objective

<table>
<thead>
<tr>
<th></th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Total area</td>
</tr>
<tr>
<td>2</td>
<td>Total area</td>
</tr>
<tr>
<td>3</td>
<td>#R (total number of registers)</td>
</tr>
<tr>
<td>4</td>
<td>#M</td>
</tr>
<tr>
<td>5</td>
<td>#M</td>
</tr>
<tr>
<td>6</td>
<td>#M and #R</td>
</tr>
<tr>
<td>7</td>
<td>#M and #R</td>
</tr>
<tr>
<td>8</td>
<td>#M</td>
</tr>
<tr>
<td>9</td>
<td>#M and #A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test</th>
<th>Number of lines in C</th>
<th>Number of nodes in CDFG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>96</td>
<td>78</td>
</tr>
<tr>
<td>Test2</td>
<td>20</td>
<td>90</td>
</tr>
<tr>
<td>Test3</td>
<td>97</td>
<td>160</td>
</tr>
<tr>
<td>Test4</td>
<td>16</td>
<td>50</td>
</tr>
<tr>
<td>Test5</td>
<td>87</td>
<td>390</td>
</tr>
</tbody>
</table>
The RTL Implementation Flow for Routability Evaluation

- C program
  - high-level synthesis by xPilot (with different strategies)
  - Verilog code
- RTL elaboration by Quartus
  - Logic synthesis by ABC
  - Pack & place by VPACK+VPR
  - Routing by VPR
- Evaluation
Implementation Flow Setup

◆ Target platform: island-style FPGA
  ▪ 10 4-LUTs per CLB, with routing channels between CLBs (span = 1 CLB)
  ▪ The number of routing tracks per channel (channel width) is constant

◆ Configurations of the toolchain
  ▪ Logic synthesis by ABC with default settings
  ▪ Packing by T-VPACK with default settings
  ▪ Wirelength-driven placement by VPR using simulated-annealing
  ▪ Routing by VPR using negotiation-based routing and directed search
    • The channel width is variable and determined by binary search

◆ Post-layout characteristics
  ▪ Maximum channel width (CW_max)
  ▪ Average wirelength (WL_avg)
    = average #tracks per net
Impact of the Synthesis Engine

◆ 60 RTLs generated for each design
  ▪ 6 scheduling strategies, 10 binding strategies
  ▪ Some are equivalent

◆ Results: min/max for each metric
  ▪ Clearly, very different although behaviorally equivalent

<table>
<thead>
<tr>
<th>design</th>
<th>CW_max</th>
<th>CW_avg</th>
<th>WL_tot</th>
<th>WL_avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>test1</td>
<td>46/86</td>
<td>30/44</td>
<td>36K/166K</td>
<td>9/12</td>
</tr>
<tr>
<td>test2</td>
<td>60/98</td>
<td>38/47</td>
<td>61K/251K</td>
<td>11/13</td>
</tr>
<tr>
<td>test3</td>
<td>40/70</td>
<td>25/33</td>
<td>25K/103K</td>
<td>7/9</td>
</tr>
<tr>
<td>test4</td>
<td>52/86</td>
<td>31/40</td>
<td>47K/196K</td>
<td>10/12</td>
</tr>
<tr>
<td>test5</td>
<td>62/122</td>
<td>39/55</td>
<td>94K/562K</td>
<td>12/17</td>
</tr>
</tbody>
</table>
Impact of the Synthesis Engine (min vs max)

- **CW_max**
- **CW_avg**
- **WL_tot**
- **WL_avg**
Impact of Compiler Transformations

◆ A matrix multiplication example

```java
outer_loop: for (i = 0; i < 8; i++) {
    middle_loop: for (j = 0; j < 8; j++) {
        Result[i][j] = 0;
        inner_loop: for (k = 0; k < 8; k++)
            Result[i][j] += X[i][k] * Y[k][j];
    }
}
```

◆ Different ways to transform/pipeline the code, partition memory

<table>
<thead>
<tr>
<th>loop</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Keep all loops, pipeline</td>
<td>As is</td>
</tr>
<tr>
<td>inner loop</td>
<td></td>
</tr>
<tr>
<td>2 Unroll inner loop, pipeline</td>
<td>Partition X into columns and Y into rows</td>
</tr>
<tr>
<td>middle loop</td>
<td>to allow simultaneous accesses</td>
</tr>
<tr>
<td>3 Unroll inner and middle</td>
<td>Partition X and Y into scalars, partition</td>
</tr>
<tr>
<td>loop, pipeline outer loop</td>
<td>Result into columns</td>
</tr>
</tbody>
</table>
## Impact of Compiler Transformations

<table>
<thead>
<tr>
<th>solution</th>
<th>CW$_{\text{max}}$</th>
<th>CW$_{\text{avg}}$</th>
<th>WL$_{\text{tot}}$</th>
<th>WL$_{\text{avg}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>18</td>
<td>4543</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>24</td>
<td>43610</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>36</td>
<td>223042</td>
<td>10</td>
</tr>
</tbody>
</table>
High-level synthesis and layout-friendly architecture

Evaluation of the impact of high-level decisions

Evaluation of metrics for scheduling/binding

Conclusion
Structural Metrics in High-Level Synthesis

- Can we estimate layout-friendliness from netlist structure?
  - How effective are they?

- Number of datapath nets
  - A straightforward estimation of interconnect complexity

- Total multiplexor inputs
  - Extensively in HLS systems since the 1980s
  - M. C. McFarland. Reevaluating the design space for register transfer hardware synthesis. *ICCAD’ 87.*
  - Big multiplexor often cause congestion/timing issues
  - Less multiplexors -> less interconnects
  - Regarded as a good first-order metric
Structural Metrics in High-Level Synthesis

- **Graph adhesion**: a metric based on cut size
  - Originally developed for use in logic synthesis (gate-level)
  - Measured as sum of all-pair min-cut size (SAPMC)
    - Smaller cut size implies better structure

- The adhesion metric grows quadratically with circuit size

- To avoid a bias on area, we can do normalization
  - Average min-cut size (AMC) between all pairs $\frac{n(n-1)}{2}$
Structural Metrics in High-Level Synthesis

- **Spreading score**: a metric based on graph embedding
  - Observation: long wires are often trouble makers
  - If a netlist can be placed without introducing long wires, it is good

- **Example**
  - Mesh: known to be layout-friendly
  - Mesh with local interconnects
  - Mesh with long interconnects
    - Hard to make the wires short

- **Intuition**: prefer topologies that spread easily
Spreading score is defined as the optimal value of the following problem:

$$\sum_{i=1}^{n} w_i \|p_i\|^2$$

subject to

$$\sum_{i=1}^{n} w_i p_i = 0$$

$$\|p_i - p_j\| \leq l_{ij}, \quad \forall (i, j) \in E$$

- $p_i$ is the location of the $i$th component.
- $l_{ij}$ is the length budget for the edge $(i, j)$.
- $w_i$ is the weight (area).

The formulation asks the following question:

- How far can the components spread away from their center of gravity without introducing wires longer than $l_{ij}$?
- Relaxed version can be solved optimally in polynomial time.
- Also normalize by $n(n-1)/2$.
Single-Variable Regressions (CW_max)

- CW_max vs. Spreading
- CW_max vs. AMC
- CW_max vs. #net
- CW_max vs. mux_input
Single-Variable Regressions ($\text{WL}_{\text{avg}}$)

$\text{WL}_{\text{avg}}$ vs. Spreading

$\text{WL}_{\text{avg}}$ vs. AMC

$\text{WL}_{\text{avg}}$ vs. #net

$\text{WL}_{\text{avg}}$ vs. mux_input
Multi-Variable Linear/Quadratic Regressions

- Randomly select 70% data as the training data, and 30% data as the testing data

- Predicting CW_max
  - (2-var) \( CW_{\text{max}} = \text{regress} (\text{Spreading}, \text{AMC}) \)
    - Linear: \( CW_{\text{max}} = k_1 \times \text{Spreading} + k_2 \times \text{AMC} + k_0 \)
    - Quadratic: \( CW_{\text{max}} = k_1 \times \text{Spreading} + k_2 \times \text{AMC} + k_{11} \times \text{Spreading}^2 + k_{22} \times \text{AMC}^2 + k_{12} \times \text{Spreading} \times \text{AMC} + k_0 \)
  - (3-var) \( CW_{\text{max}} = \text{regress} (\text{Spreading}, \text{AMC}, \text{num_nets}) \)

- Predicting WL_avg
  - (2-var) \( WL_{\text{avg}} = \text{regress} (\text{Spreading}, \text{AMC}) \)
  - (3-var) \( WL_{\text{avg}} = \text{regress} (\text{Spreading}, \text{AMC}, \text{num_nets}) \)
Predicting $CW_{max}$

$$= \text{regress} (\text{Spreading}, \text{AMC})$$

<table>
<thead>
<tr>
<th>training</th>
<th>linear</th>
<th>quadratic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>error</td>
<td>%</td>
</tr>
<tr>
<td>test1</td>
<td>4.76</td>
<td>7.0%</td>
</tr>
<tr>
<td>test2</td>
<td>8.15</td>
<td>10.7%</td>
</tr>
<tr>
<td>test3</td>
<td>3.73</td>
<td>6.8%</td>
</tr>
<tr>
<td>test4</td>
<td>4.48</td>
<td>7.0%</td>
</tr>
<tr>
<td>test5</td>
<td>5.50</td>
<td>6.4%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>testing</th>
<th>linear</th>
<th>quadratic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>error</td>
<td>%</td>
</tr>
<tr>
<td>test1</td>
<td>6.10</td>
<td>9.3%</td>
</tr>
<tr>
<td>test2</td>
<td>10.37</td>
<td>14.4%</td>
</tr>
<tr>
<td>test3</td>
<td>7.20</td>
<td>13.7%</td>
</tr>
<tr>
<td>test4</td>
<td>5.16</td>
<td>6.9%</td>
</tr>
<tr>
<td>test5</td>
<td>11.33</td>
<td>15.9%</td>
</tr>
</tbody>
</table>
Predicting $CW_{max} = \text{regress}(\text{Spreading}, \text{AMC}, \text{num\_nets})$

<table>
<thead>
<tr>
<th>training</th>
<th>linear</th>
<th>quadratic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>error</td>
<td>%</td>
</tr>
<tr>
<td>test1</td>
<td>3.16</td>
<td>4.6%</td>
</tr>
<tr>
<td>test2</td>
<td>1.47</td>
<td>1.9%</td>
</tr>
<tr>
<td>test3</td>
<td>2.30</td>
<td>4.2%</td>
</tr>
<tr>
<td>test4</td>
<td>3.80</td>
<td>6.0%</td>
</tr>
<tr>
<td>test5</td>
<td>3.08</td>
<td>3.5%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>testing</th>
<th>linear</th>
<th>quadratic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>error</td>
<td>%</td>
</tr>
<tr>
<td>test1</td>
<td>4.33</td>
<td>6.1%</td>
</tr>
<tr>
<td>test2</td>
<td>3.55</td>
<td>5.1%</td>
</tr>
<tr>
<td>test3</td>
<td>2.66</td>
<td>4.9%</td>
</tr>
<tr>
<td>test4</td>
<td>4.42</td>
<td>6.3%</td>
</tr>
<tr>
<td>test5</td>
<td>4.66</td>
<td>6.8%</td>
</tr>
</tbody>
</table>
Predicting $WL_{avg}$

$= \text{regress}(\text{Spreading}, \text{AMC})$

| training | linear | | | quadratic | | |
|----------|--------|--------|--------|--------|--------|
|          | error  | %      | error  | %      |
| test1    | 0.39   | 3.5%   | 0.33   | 3.0%   |
| test2    | 0.34   | 2.9%   | 0.27   | 2.3%   |
| test3    | 0.18   | 2.1%   | 0.18   | 2.1%   |
| test4    | 0.57   | 5.3%   | 0.41   | 3.8%   |
| test5    | 0.81   | 5.7%   | 0.51   | 3.6%   |

| testing  | linear | | | quadratic | | |
|----------|--------|--------|--------|--------|--------|
|          | error  | %      | error  | %      |
| test1    | 0.54   | 5.1%   | 0.73   | 7.1%   |
| test2    | 0.44   | 3.7%   | 0.46   | 3.8%   |
| test3    | 0.49   | 5.9%   | 0.60   | 7.2%   |
| test4    | 0.38   | 3.6%   | 0.45   | 4.4%   |
| test5    | 0.89   | 7.0%   | 0.93   | 7.0%   |
Predicting $WL_{avg}$

$= \text{regress}(\text{Spreading, AMC, num\_nets})$

<table>
<thead>
<tr>
<th>training</th>
<th>linear</th>
<th>quadratic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>error</td>
<td>%</td>
</tr>
<tr>
<td>test1</td>
<td>0.39</td>
<td>3.5%</td>
</tr>
<tr>
<td>test2</td>
<td>0.33</td>
<td>2.8%</td>
</tr>
<tr>
<td>test3</td>
<td>0.18</td>
<td>2.1%</td>
</tr>
<tr>
<td>test4</td>
<td>0.53</td>
<td>5.0%</td>
</tr>
<tr>
<td>test5</td>
<td>0.80</td>
<td>5.6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>testing</th>
<th>linear</th>
<th>quadratic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>error</td>
<td>%</td>
</tr>
<tr>
<td>test1</td>
<td>0.56</td>
<td>5.2%</td>
</tr>
<tr>
<td>test2</td>
<td>0.47</td>
<td>3.9%</td>
</tr>
<tr>
<td>test3</td>
<td>0.44</td>
<td>5.3%</td>
</tr>
<tr>
<td>test4</td>
<td>0.45</td>
<td>4.3%</td>
</tr>
<tr>
<td>test5</td>
<td>0.89</td>
<td>6.9%</td>
</tr>
</tbody>
</table>
Concluding Remarks

◆ Predicting impact on layout in HLS is hard

◆ Some structural metrics show good correlation with results after layout in some cases.
  ▪ But consistency is an issue

◆ We would like explicit models to guide HLS perturbation
  ▪ Structural metrics not applicable
  ▪ Machine learning?
backups
Why High-Level Synthesis?

◆ Management of design complexity
  ▪ Higher level abstraction -> less details -> more efficiency in design and verification
  ▪ C/C++ easier to code than VHDL/Verilog
    • Sequential code is easier to create/maintain/verify

◆ Design reused and exploration
  ▪ The same code can be synthesized on different platforms, targeting different frequency, resource limits, etc.
  ▪ It is easy to generate different architectures

◆ Easier integration
  ▪ Specify software/hardware in a unified model
Levels of Abstraction in IC Design

- System level: C, English
- Behavior-level: C, C++, SystemC
- RT level: datapath, controller
- Gate-level: logic synthesis

Levels of Abstraction:
- System level
- Behavior-level
- RT level
- Gate-level

Logical Synthesizer

Behavior Synthesizer

Place & Route

Layout

SW/HW Co-design

Levels of Abstraction in IC Design
Example

\[ x_1 \times x_2 + x_3 \times x_4 + x_5 + x_6 + x_7 \]

AMC = 1
Mux size = 0
Spreading score = 0.417

AMC = 2.2
Mux size = 4
Spreading score = 0.083