HMFlow: Accelerating FPGA Compilation with Hard Macros for Rapid Prototyping

Christopher Lavin, Marc Padilla, Jaren Lamprecht, Philip Lundrigan
Brent Nelson and Brad Hutchings

FCCM
May 2, 2011
Design Cycle

Debug

Compile

Edit
The Problem

FPGA Compilation Times Severely Limit Turns Per Day

Reduced Productivity

Higher Development Costs
Without regard for circuit quality…

how fast can we make FPGA compilation?
Approach

• Let’s emulate software compilation
  – Pre-compiled libraries
    • In hardware, these are called “hard macros”
  – Rapidly link, or assemble hard macros to create designs

• Goals
  – Design to implementation in seconds
  – Find out: How fast can designs using hard macros be compiled for an FPGA?
What is a Hard Macro?

- A pre-placed and pre-routed module
- Can be placed multiple places on the FPGA fabric
- Hard-macro based design
  - Skips
    - Synthesis (XST)
    - technology mapping (NGDBuild)
    - Packing (MAP)
  - Design assembled from hard macros into final implementation
How To Create a Hard Macro?

• Use Xilinx tools to create the macro’s circuitry
  • Custom area constraints generated on-the-fly
    • Provide sufficient area for cells (LUTs, DSPs, BRAMs)
  • Placed and routed NCD extracted for hard macro creation

• NCD translated into XDL, converted to Hard Macro
  • Hard macro ports are added
  • Illegal primitives removed (TIEOFFs, IOBs, etc)
  • GND and VCC nets removed
RapidSmith: Create Your Own CAD Tools

- Makes writing CAD tools easier
  - Uses XDL as input/output format
  - Targets real Xilinx FPGAs
- Over 200 APIs for manipulating XDL netlists
- Java-based and open source
  - Available at: http://rapidsmith.sourceforge.net
Approach: HMFlow

INPUT DESIGNS

HARD MACRO SOURCES

.mdl

Design Parser & Mapper

Design Stitcher

XDL Hard Macro Placer

XDL Router

.xdl

COMPLETELY PLACED & ROUTED XDL

Built on RapidSmith
What is System Generator?

- A Xilinx hardware library blockset for the Simulink environment in MATLAB

HMFlow supports over 75% of the System Generator block set
How Do You Run HMFlow?
One Design, Two Flows

HMFlow → Placed & Routed Design (XDL) → XDL 2 NCD → Placed & Routed Design (NCD)
How does HMFlow work?

**XDL Design**

- **Hard Macro Cache**
- **Add**
- **Mux**
- **Addressable Shift Register**

**System Generator**

1. **select**
   - From Workspace1
2. **data_in**
   - From Workspace
3. **address**
   - From Workspace2

**AddSub**

**Mux**

**Gateway In**

**Gateway In1**

**Gateway In2 Addressable Shift Register**

**Gateway Out**
Stitching the Design

- Inserts IOBs and Clk circuitry
- Examines original design and creates network connections
- Design is then ready for placement and routing
 Placement

- Xilinx PAR does not handle hard macro-based designs well
  - Developed fast heuristic for placement
    - Everything is only placed once, rapid results
    - Hard macros with BRAMs/DSPs are placed first
    - Next, largest to smallest hard macros are placed
    - Placement attempts to place each block next to its most highly connected neighbor
  - Example: places 757 blocks in 219ms
  - Developed interactive hand placer / visualizer tool
Hard Macro Debug Placer

757 blocks placed in 219 ms
Routing

• PathFinder is just *too slow* for our goals
  • (Remember: compilation speed at all costs)

• Maze Router
  – First come first served routing resource
    • Very fast router
  – Uses ‘congestion avoidance’ instead of negotiation
    • Analyzes design previous to routing
    • Critical resources reserved to avoid conflicts
    • Depends on chip not being fully utilized
Xilinx vs. HMFlow Runtimes

Runtime (minutes)

10-12X Speedup
Runtime Distribution of HMFlow + XDL2NCD

- XDL2NCD: 84%
- XDL Export: 1%
- Router: 14%
- Placer: 0%
- Stitcher: 0%
- HM Cache: 1%
- Simulink Parser: 0%
Maximum Clock Rate of Designs

2-4X Slowdown
Conclusion

• RapidSmith
  – Java, open source XDL CAD tool framework
  – Required foundation for **HMFlow**

• **HMFlow** provides hard macro-based design
  – 10-12X speedup over fastest Xilinx flow
  – Scalable to very large designs
  – Clock rate 2-4X decrease
    • Still 10,000’s times faster than simulation

• XDL2NCD conversion time: outstanding issue

• Come see RapidSmith/HMFlow @ Demo Night
Related and Future Work

• Related Work: Hard Macros / XDL
  – Next talk: Automatic HDL-based generation of homogeneous hard macros for FPGAs
  – USC-ISI’s Torc: Tools for Open Reconfigurable Computing
    • Open source project with similar goals to RapidSmith

• Our Future Work
  – Continue support of RapidSmith
  – **HMFlow**: Larger hard macros
    • Maintain rapid compilation **AND** high clock rates
    • LabVIEW FPGA designs
Placement Object Reduction by Using Hard Macros

Hard Macro Instances

Primitive Instances

Object Reduction 10-20X
Over 75% of blocks supported in most commonly used System Generator packages
# Benchmark Runtimes

<table>
<thead>
<tr>
<th>Design Name</th>
<th>Simulink Parser</th>
<th>BlockGen</th>
<th>Stitcher</th>
<th>Placer</th>
<th>Router</th>
<th>XDL Export</th>
<th>HMFlow</th>
<th>XDL2NCD</th>
<th>HMFlow Total</th>
<th>Xilinx Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>pd_control</td>
<td>0.09</td>
<td>0.74</td>
<td>0.19</td>
<td>0.02</td>
<td>0.22</td>
<td>0.06</td>
<td>1.31</td>
<td>2.8</td>
<td>4.1</td>
<td>65.6</td>
</tr>
<tr>
<td>polyphaseFilter</td>
<td>0.09</td>
<td>0.75</td>
<td>0.22</td>
<td>0.02</td>
<td>1.41</td>
<td>0.11</td>
<td>2.59</td>
<td>4.0</td>
<td>6.6</td>
<td>60.3</td>
</tr>
<tr>
<td>aliasingDDC</td>
<td>0.11</td>
<td>0.77</td>
<td>0.22</td>
<td>0.02</td>
<td>1.45</td>
<td>0.13</td>
<td>2.69</td>
<td>7.4</td>
<td>10.1</td>
<td>62.2</td>
</tr>
<tr>
<td>dualDivider</td>
<td>0.31</td>
<td>0.89</td>
<td>0.20</td>
<td>0.05</td>
<td>2.41</td>
<td>0.22</td>
<td>4.08</td>
<td>6.3</td>
<td>10.3</td>
<td>96.6</td>
</tr>
<tr>
<td>computeMetric</td>
<td>0.28</td>
<td>0.89</td>
<td>0.64</td>
<td>0.05</td>
<td>6.36</td>
<td>0.61</td>
<td>8.83</td>
<td>17.1</td>
<td>25.9</td>
<td>160.8</td>
</tr>
<tr>
<td>fft1024</td>
<td>0.24</td>
<td>0.94</td>
<td>0.30</td>
<td>0.05</td>
<td>4.95</td>
<td>0.38</td>
<td>6.84</td>
<td>10.3</td>
<td>17.2</td>
<td>119.3</td>
</tr>
<tr>
<td>filtersAndFFT</td>
<td>0.33</td>
<td>0.98</td>
<td>0.80</td>
<td>0.19</td>
<td>12.3</td>
<td>0.75</td>
<td>15.4</td>
<td>20.3</td>
<td>35.6</td>
<td>254.0</td>
</tr>
<tr>
<td>frequencyEstimator</td>
<td>0.44</td>
<td>1.50</td>
<td>0.58</td>
<td>0.22</td>
<td>18.1</td>
<td>1.17</td>
<td>22.0</td>
<td>107.3</td>
<td>129.3</td>
<td>373.5</td>
</tr>
<tr>
<td>dualFilter</td>
<td>0.47</td>
<td>1.31</td>
<td>1.20</td>
<td>0.44</td>
<td>34.7</td>
<td>1.66</td>
<td>39.8</td>
<td>140.4</td>
<td>180.1</td>
<td>469.0</td>
</tr>
<tr>
<td>trellisDecoder</td>
<td>0.66</td>
<td>1.72</td>
<td>1.42</td>
<td>0.55</td>
<td>54.0</td>
<td>2.50</td>
<td>60.9</td>
<td>115.1</td>
<td>176.0</td>
<td>824.6</td>
</tr>
<tr>
<td>filterFFTCM</td>
<td>0.52</td>
<td>1.94</td>
<td>1.64</td>
<td>0.98</td>
<td>69.9</td>
<td>3.05</td>
<td>78.1</td>
<td>541.2</td>
<td>619.3</td>
<td>1021</td>
</tr>
<tr>
<td>multibandCorrelator</td>
<td>0.83</td>
<td>1.80</td>
<td>1.84</td>
<td>1.86</td>
<td>73.3</td>
<td>5.78</td>
<td>85.4</td>
<td>506.7</td>
<td>592.1</td>
<td>786.2</td>
</tr>
<tr>
<td>signalEstimator</td>
<td>0.84</td>
<td>2.33</td>
<td>2.16</td>
<td>1.53</td>
<td>107.5</td>
<td>15.4</td>
<td>129.8</td>
<td>869.2</td>
<td>999.0</td>
<td>1509</td>
</tr>
</tbody>
</table>

All times are recorded in seconds
## Benchmark Attributes

<table>
<thead>
<tr>
<th>Design Name</th>
<th>Slices</th>
<th>BRAMs</th>
<th>DSP48s</th>
<th>Primitive Instances</th>
<th>Hard Macro Instances</th>
<th>Hard Macro Defs</th>
<th>Nets</th>
<th>Xilinx Clk Speed</th>
<th>HMFlow Clk Speed</th>
<th>HMFlow Speedup</th>
<th>Time2NCD Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>pd_control</td>
<td>150</td>
<td>1</td>
<td>0</td>
<td>200</td>
<td>21</td>
<td>12</td>
<td>368</td>
<td>147</td>
<td>129</td>
<td>50.00x</td>
<td>15.85x</td>
</tr>
<tr>
<td>polyphaseFilter</td>
<td>680</td>
<td>8</td>
<td>4</td>
<td>777</td>
<td>79</td>
<td>30</td>
<td>1638</td>
<td>275</td>
<td>108</td>
<td>23.24x</td>
<td>9.19x</td>
</tr>
<tr>
<td>aliasingDDC</td>
<td>806</td>
<td>1</td>
<td>3</td>
<td>876</td>
<td>78</td>
<td>25</td>
<td>1628</td>
<td>191</td>
<td>107</td>
<td>23.14x</td>
<td>6.17x</td>
</tr>
<tr>
<td>dualDivider</td>
<td>1832</td>
<td>0</td>
<td>6</td>
<td>1951</td>
<td>542</td>
<td>39</td>
<td>4004</td>
<td>141</td>
<td>79</td>
<td>23.69x</td>
<td>9.34x</td>
</tr>
<tr>
<td>computeMetric</td>
<td>2551</td>
<td>56</td>
<td>40</td>
<td>2799</td>
<td>332</td>
<td>64</td>
<td>7447</td>
<td>143</td>
<td>57</td>
<td>18.21x</td>
<td>6.20x</td>
</tr>
<tr>
<td>fft1024</td>
<td>2553</td>
<td>8</td>
<td>12</td>
<td>2656</td>
<td>313</td>
<td>48</td>
<td>5889</td>
<td>215</td>
<td>74</td>
<td>17.43x</td>
<td>6.94x</td>
</tr>
<tr>
<td>filtersAndFFT</td>
<td>5203</td>
<td>25</td>
<td>31</td>
<td>5325</td>
<td>588</td>
<td>92</td>
<td>11590</td>
<td>191</td>
<td>74</td>
<td>16.54x</td>
<td>7.13x</td>
</tr>
<tr>
<td>frequencyEstimator</td>
<td>6988</td>
<td>31</td>
<td>72</td>
<td>7152</td>
<td>757</td>
<td>249</td>
<td>16919</td>
<td>167</td>
<td>60</td>
<td>16.97x</td>
<td>2.89x</td>
</tr>
<tr>
<td>dualFilter</td>
<td>11173</td>
<td>33</td>
<td>26</td>
<td>11283</td>
<td>901</td>
<td>93</td>
<td>25961</td>
<td>183</td>
<td>46</td>
<td>11.80x</td>
<td>2.60x</td>
</tr>
<tr>
<td>trellisDecoder</td>
<td>16973</td>
<td>61</td>
<td>53</td>
<td>17269</td>
<td>1328</td>
<td>196</td>
<td>42195</td>
<td>82</td>
<td>35</td>
<td>13.55x</td>
<td>4.69x</td>
</tr>
<tr>
<td>filterFFTCM</td>
<td>18883</td>
<td>81</td>
<td>12</td>
<td>19126</td>
<td>920</td>
<td>149</td>
<td>49037</td>
<td>148</td>
<td>37</td>
<td>13.08x</td>
<td>1.65x</td>
</tr>
<tr>
<td>multibandCorrelator</td>
<td>19732</td>
<td>52</td>
<td>23</td>
<td>19901</td>
<td>1472</td>
<td>90</td>
<td>47993</td>
<td>140</td>
<td>34</td>
<td>9.21x</td>
<td>1.33x</td>
</tr>
<tr>
<td>signalEstimator</td>
<td>23841</td>
<td>126</td>
<td>47</td>
<td>24091</td>
<td>1448</td>
<td>390</td>
<td>60727</td>
<td>104</td>
<td>34</td>
<td>11.62x</td>
<td>1.51x</td>
</tr>
</tbody>
</table>
Benchmark Resource Usage as a Percentage of a Virtex4 LX200

![Bar chart showing resource usage percentages for various benchmark programs. The chart includes categories for slices, BRAMs, and DSPs. Each program is represented by a different color and bar height, indicating the percentage of resource usage.]