Microsoft Research

Each year Microsoft Research hosts hundreds of influential speakers from around the world including leading scientists, renowned experts in technology, book authors, and leading academics, and makes videos of these lectures freely available. 2016 © Microsoft Corporation. All rights reserved.
Live Hardware Development at UCSC

Jose Renau

Department of Computer Engineering, University of California, Santa Cruz
http://masc.soe.ucsc.edu
Architects typically care about 2-axes
Now, I mostly work on 3rd axis
Challenges

- Chip design is very slow complex task
  - Simulations are slow
  - RTL simulation/emulation/verification is painful
  - Synthesis is even slower
  - ...
My new focus on complexity

Live Development Environment
“Live” opposite of “Batch”

- Live
  - Maximize “developer” utilization
  - Response under few seconds
  - Change code while running

- Batch
  - Maximize “computer” utilization
  - Submit job and check “hours” later for results
Why we need Live?

Human mnemonic memory
Typically 10-15 seconds

Initial delays over 2 secs have impact in Quality of Experience (QoE)

Breaks over a few seconds, require to “rebuild” memory
Who is going Live with computers?

- Music and Visual effects: Live Coding
  - Fast response to code changes
  - Music parties

- Teaching Programming
  - Interactive browser, programming

- Some Programming Languages
  - REPL == Read Eval Print Loop
Current ASIC/FPGA Flow

module Shift_Reg (clk, parameter N=16);
  input clk, reset, en;
  output reg [N-1:0] Q;
endmodule

RTL Coder

Live Hardware Development at UCSC
Current ASIC/FPGA Flow

Minutes (hours) iteration
Current FPGA flows for a “no change”
Changes tend to be local

```plaintext
assign w = a & b;
assign x = c | d;
assign y = !x;
assign z = w & c;
```
Changes tend to be local

```
assign w = a & b;
assign x = c | d;
assign y = !x;
assign z = w & c;
```

```
assign w = a & b;
assign x = c | d;
assign y = !x;
assign z = !(w | c)
```
Changes tend to be local

```
assign w = a & b;
assign x = c | d;
assign y = !x;
assign z = w & c;
```

```plaintext
assign w = a & b;
assign x = c | d;
assign y = !x;
assign z = !(w | c)
```
Selecting changes: Commits

Commits on Apr 26, 2017

- Merge pull request #51 from master
- Fix higher address in MMU translation
- Fix MMU way indexing
- Fix register access widths for shadow registers
- Style: Espresso and Prontospresso fixes
- Add missing pins and signals in LSU
Selecting changes: Commits

Commits on Apr 26, 2017

- Fix higher address bits in MMU translation
- Fix MMU way indexing
- Fix register access widths for shadow registers
- Style: Espresso and Prontespresso fixes
- Add missing pins and signals in LSU
Selecting changes: Commented out code

```verilog
137 always_comb begin
138 | if(start) begin
139 | `divp busy next = 1'h1;
140 | end else begin
141 | //if(divp op ready && divp op valid && ~div_in_retry) begin
142 | | if(divp op ready && ~divider s retry) begin
143 | | | `divp busy next = 1'b0;
144 | | end else if(divp busy) begin
145 | | | `divp busy next = 1'b1;
146 | | end else begin
147 | | `divp busy next = 1'b0;
148 | end
149 | end
150 end
```
Goal: Partition graph (smaller problem)
Outline

- Live Synthesis
- Live Place & Route for FPGA
- Live Simulation
Live Synthesis
LiveSynth: Flow Overview

Setup Phase
Initial Synthesis
- HDL
  - Elaboration
    - spec0
  - Synthesis
    - impl0
- Place & Route
LiveSynth: Flow Overview

**Setup Phase**
- Initial Synthesis
  - HDL
  - Elaboration
  - Synthesis
    - spec0
    - impl0
  - Place & Route

**Setup Pass**
- Functional Match
  - FIBs

**Live Phase**
- Interactive
  - ADL
  - AEI elaboration
  - ASynthesis
  - Nelist Diff
  - Nelist Stitch
  - Nelist Stitch

**Place & Route**
Functional match

CandidateInvariant Points

(a) Specification

(b) Implementation
Functional match

SAT Solve to verify

Candidate Invariant Points

(a) Specification
(b) Implementation

- A
- B
- C

- A
- B
- C

- Y
- Y

- Y
- Y

- FB1
- FB1

- Y
- Y
Functional match

SAT Solve to verify

Candidate Invariant Points

(a) Specification

(b) Implementation

Slow, but only done once
Fine to miss some invariant points
Netlist diff at Live speeds

(a) Original

(b) Modified

<table>
<thead>
<tr>
<th>Cone</th>
<th>Different?</th>
<th>Gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>FALSE</td>
<td>{}</td>
</tr>
<tr>
<td>FIB1</td>
<td>FALSE</td>
<td>{}</td>
</tr>
</tbody>
</table>
Netlist diff at Live speeds

(a) Original
(b) Modified

<table>
<thead>
<tr>
<th>Cone</th>
<th>Different?</th>
<th>Gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>TRUE</td>
<td>{NOT1, OR1}</td>
</tr>
<tr>
<td>FIB1</td>
<td>FALSE</td>
<td>{}</td>
</tr>
</tbody>
</table>
Netlist diff at Live speeds

(a) Original

(b) Modified

<table>
<thead>
<tr>
<th>Cone</th>
<th>Different?</th>
<th>Gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>TRUE</td>
<td>{NOT1, OR1}</td>
</tr>
<tr>
<td>FIB1</td>
<td>FALSE</td>
<td>{AND1}</td>
</tr>
</tbody>
</table>
How big are the changes?

Functionally Invariant cones
- No change during synthesis
- Can be plugged in and out without any effort

<table>
<thead>
<tr>
<th>Cone Size</th>
<th>fpu</th>
<th>mips</th>
<th>or1200</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;200</td>
<td>1769</td>
<td>1237</td>
<td>643</td>
</tr>
<tr>
<td>200-300</td>
<td>99</td>
<td>73</td>
<td>172</td>
</tr>
<tr>
<td>300-400</td>
<td>938</td>
<td>35</td>
<td>156</td>
</tr>
<tr>
<td>400-500</td>
<td>1</td>
<td>2</td>
<td>185</td>
</tr>
<tr>
<td>500-600</td>
<td>649</td>
<td>11</td>
<td>74</td>
</tr>
<tr>
<td>600-800</td>
<td>34</td>
<td>316</td>
<td>63</td>
</tr>
<tr>
<td>800-1000</td>
<td>33</td>
<td>29</td>
<td>58</td>
</tr>
<tr>
<td>1000-1500</td>
<td>5</td>
<td>124</td>
<td>56</td>
</tr>
<tr>
<td>1500-2000</td>
<td>1</td>
<td>550</td>
<td>0</td>
</tr>
<tr>
<td>2000-3000</td>
<td>0</td>
<td>421</td>
<td>0</td>
</tr>
<tr>
<td>3000-4000</td>
<td>0</td>
<td>302</td>
<td>0</td>
</tr>
<tr>
<td>&gt;4000</td>
<td>0</td>
<td>115</td>
<td>0</td>
</tr>
</tbody>
</table>
ASIC (DC) synthesis under 60 secs

delta Commercial synth
diff
stitch
ASIC (DC) synthesis under 60 secs

Baseline: 29 minutes
Worst case: 5 minutes
Typical: 1 minute
Speedup results

- LiveSynth yields speedups of up to 80x for a single code change
- The average speedup was 10x across designs and changes
- LLIR shows a maximum of 30x and average of 3x speedup
- Rapid Recompile is mostly flat around 2x
QoR degradation

- In most cases there is less than 1% difference when compared to full synthesis
- The maximum observed difference was ~4%
Live FPGA
What About Place/Route? (FPGA)

- Altera/Intel not a great job
- Xilinx does great at placement bad at routing

Goal
- Go over the cones of logic
- Find if same LUTs (with different code) can be re-used
  - Structure
- Replace/Re-Route only due to new LUT structure
Just Synthesis

Full had ~10-20 minutes, synthesis under 10 secs
Place without Structural Optimization

Synth + Place is OK even without structural patch
Route without Structural Optimization

Synth + Place + Route is OK even without structural patch
Structural improves even more

Structural is 15-40x faster than Vivado
Total Synth+Place+Route for FPGAs

The match overhead is small.
Live Simulation
Checkpoint and Hot Reload

- Goal: Under 2 seconds Hot Reload RTL Simulation [micro18?]
Very Fast Simulations

- Very fast too
  - HotReload: Simulate a 16 core RISC-V in under 2 seconds
  - LiveSim is even faster than Verilator without Hot Reload
Live status

- Live Synthesis [dac17]
  - Working on heuristic decomposition to further reduce the nets
- Live ASIC benchmarks: Anubis [IWLS17]
- Live FPGA Bitstream [ICCAD 18 submission]

- Live Hardware Accelerator performance feedback
  - [MICRO 18 submission]

- Live ASIC Placement
  - Several options, but analytical reusing anchors converges fast [DAC 19]
- Live ASIC Routing
  - Only a small fraction needs to be re-routed [DAC 20?]

- Pyrope, a new HDL
Live Summary

- My hammer
  - Live development cycle for hardware (1-20 seconds goal)
  - Hammer...
    - the ASIC synthesis
    - the simulation
    - the FPGA bitstream
    - the performance projections
    - ...

- No lost of quality, no approximate models

- Just get good/fast incremental with Live time scale
“Afterlive”

- Live is a must have, but we working...
- A more expressive/re-usable HDL language
- More automatic/formal proofs
- Open Source ASIC/FPGA/Simulation flow (working on it)
Questions?

Jose Renau

Department of Computer Engineering,
University of California, Santa Cruz
http://masc.soe.ucsc.edu

- **LiveSynth: Towards an Interactive Synthesis Flow**, Rafael Trapani Possignolo, and Jose Renau, Design Automation Conference (DAC), June 2017.