Architecture Exploration for HLS-Oriented FPGA Debug Overlays

Al-Shahna Jamal, Jeffrey Goeders, Steve Wilton
What this talk is about…

Recent work: Source-level, in-system debugging of HLS circuits

- Debug instrumentation is inserted at compile time
- Changing this instrumentation (to trace new data) requires a *recompile*
Recent work: Source-level, in-system debugging of HLS circuits
- Debug instrumentation is inserted at compile time
- Changing this instrumentation (to trace new data) requires a recompile

In this work: Debug instrumentation still inserted at compile time BUT can be configured at runtime (fast customization)

Impact: Achieves software like compile times (~1sec) between debug iterations
Outline

• Motivation for In-System Debug

• Previous Work: In-System Debug Framework for HLS
  • Debug Instrumentation at compile time

• This paper: HLS Debug Overlay to allow customization at runtime

• Evaluation

• Future Work
Motivation for In-System Debug

Previous Work: In-System Debug Framework for HLS
  - Debug Instrumentation at compile time

This paper: HLS Debug Overlay to allow customization at runtime

Evaluation

Future Work
Software designers need a full ecosystem of tools:
  • Testing, debugging, optimization....

**Debugging:** When do we have to do in-system debug?
  • Simulation may take too long
  • Bug may be dependent on system interactions, IO traffic, etc.

For certain bugs we have to perform in-system debug, observing the actual hardware
Hardware Debug Tools

Not practical for a software designer!
• Motivation for In-System Debug

• Previous Work: In-System Debug Framework for HLS
  • Debug Instrumentation at compile time

• This paper: HLS Debug Overlay to allow customization at runtime

• Evaluation

• Future Work
Previous Work: In-System Debug Framework for HLS

Capture system-level bugs → Need to run at-speed, on-chip

Solution: Record and Replay

1. User selects variables, tool determines signals, inserts instrumentation
2. Compile
3. Execute and record
4. Stop and retrieve
5. Software-like debug using recorded data

Limited on-chip memory → Need to select what we want to record and use memory efficiently

```c
void qSort(int *arr) {
    int piv, beg[N], end[N];
    int i=0;
    int L, R, swap;
    ...
}
```
Previous Work: Taking Advantage of HLS Scheduling

- Recorded signals change each cycle
- 50x-100x more memory efficient than traditional Embedded Logic Analyzer (ELA) approach

Outline

• Motivation for In-System Debug

• Previous Work: In-System Debug Framework for HLS
  • Debug Instrumentation at compile time

• This paper: HLS Debug Overlay to allow customization at runtime

• Evaluation

• Future Work
HLS Overlays: Software-like Debug Turn-Around Times

Debug Scenario
between debug iterations (fast)

User Circuit instrumented with Flexible Overlay

Overlay Configuration Bits

once, at compile time (slow)
Workflow Using the Debug Overlay

- Design (.v) + Insert Overlay Instrumentation
- Compile to Bitstream (*lengthy*)
- Personalize Overlay (*fast*)
- Run
  - View/Analyze Captured Data
  - Found root cause?
  - Fix Error

Debug Turn
Key: The more general/flexible the overlay – the larger the area overhead

Our Approach: determine a set of useful capabilities, and architect an overlay that is just flexible enough to implement these
What can this overlay do?

**Our approach**: determine a set of **useful capabilities**, and architect an overlay that is **just flexible enough** to implement these.

1. **Selective Variable Tracing**
   - Select user visible variables to trace

2. **Selective Function Tracing**
   - Select region of code to trace

3. **Conditional Buffer Freeze**
   - Specify a condition on the circuit that, when true, causes recording in the trace buffer to halt.
Selective Variable Tracing: User Perspective

Select/de-select variables from pane in Debug GUI
Architecture to Support Capability

Debug Scenario
- between debug iterations (fast)

User Circuit instrumented with Flexible Overlay

Overlay Configuration Bits

once, at compile time (slow)
Selective Variable Tracing Architecture – Initial Ideas…

Could have a configurable memory that enables which RTL signals (that map to C code variables) we want to trace. Program this memory at runtime...

Aside: Intel’s In-System Memory Content Editor
Could associate a bit in Config RAM with each RTL signal that corresponds to a C code variable...
Selective Variable Tracing Architecture – Initial Ideas…

Could associate a bit in Config RAM with each RTL signal that corresponds to a C code variable…
Key: Every bit is associated with a state in the user circuit.
Selective Variable Tracing Architecture: Variant B

User Circuit

Trace Scheduler

Trace Buffer

Line Packer

Config RAM

recode_state

current_state

num_words

packed_data

r_1

r_n

r_3

r_1

S_7, S_1, S_4, S_3

S_7

S_1

S_4

S_3

S_2

S_6

S_5

S_0

S_4

mem

r_{12}

r_{10}

ctrl

r_{9}

ctrl

ctrl

ctrl

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

S_0

S_1

S_2

S_3

S_4

S_5

S_6

S_7

S_0

S_1

S_2

S_3

S_4

S_5

S_6

S_7

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl

r_{9}

mem

r_{12}

r_{10}

ctrl

r_{3}

r_{1}

ctrl
Variant B: Line Packer – Architectural Parameter “G”

• **G**: granularity

• Increasing G splits the incoming trace data into smaller words – more fine grained packing

• Increasing G also increases the steering logic/area overhead

[Diagram showing trace data and trace buffer with labels and arrows indicating data flow and granularity parameter G.]
Variant B: Line Packer – Architectural Parameter “G”

- **G**: granularity

- Increasing G splits the incoming trace data into smaller words – more fine grained packing

- Increasing G also increases the steering logic/area overhead
Variant B: Line Packer – Architectural Parameter “G”

- **G**: granularity
- Increasing G splits the incoming trace data into smaller words – more fine grained packing
- Increasing G also increases the steering logic/area overhead
Variant B – Multi-Bit Configuration ROM

Multi-Bit RAM

User Circuit

Trace Scheduler

Line Packer

Trace Buffer

Config RAM

recode_state

current_state

num_words

packed_data

time

Trace Buffer

S7, S1
S4, S3
r9
r10
ctrl
ctrl
ctrl

S4
mem
r12
r10
ctrl

S1
r3
r1
ctrl

S7
r9

S0

r0

S1

r1

S2

r0

S3

r1

S4

r1

S5

r0

S6

r1

S7

r0
Selective Function Tracing: User Perspective

Select Functions from pane in Debug GUI
Selective Function Tracing: Same architecture!
Conditional Buffer Freeze – User Perspective

Condition

\( a < 0 \), line 94
Conditional Buffer Freeze

User Circuit

Trace Scheduler

Config RAM

Communication and Control Logic

Line Packer

Conditional Freeze Buffer Unit(s)

Trace Buffer

current_state

r_n, ..., r_1

num_words

w

w

data_mask target_value state op

Comparator

set 1'b1

1'b0

ip_full

packed_data

trace_buffer_disable
Conditional Buffer Freeze – Architectural Parameter “C”

- Increase C units to express a more complex condition
- Example: Stop tracing when err flag 1 OR err flag 2 goes high
- “Stop write controller” receives signals from all C units – OR trigger function
• Motivation for In-System Debug

• Previous Work: In-System Debug Framework for HLS
  • Debug Instrumentation at compile time

• This paper: HLS Debug Overlay to allow customization at runtime

• Evaluation

• Future Work
Evaluation: Run-Times

Compile Time vs. Overlay Personalization Time (seconds)

- Previous Work: User Circuit + Instrumentation (286 seconds)
- Current Work: User Circuit + Overlay (314 seconds)
- Configuring Overlay (1 second)
Variant A Overlay – Impact on Area

Baseline debug instrumentation is 20% size of the user circuit*

Variant A increases the size by 39 ALMs on average, and 1 M9K – cheap!

Architecture vs. Trace Window Length

Architectural enhancements improve trace window length
Area goes up dramatically for high granularity in line packer.
Overhead: Conditional Units

Area increases with number of C units with small decrease in Fmax
How can a FPGA vendor use these results?

Provide a library of overlays.

Depending on the user’s debugging needs, and resources available – select appropriate library:
- **Economy Library**: cheaper overlay (i.e. only selective variable tracing)
- **Deluxe Library**: supports more capabilities (i.e. conditional trigger functions)

Can also take advantage of:
- User input / estimates to user
- Variable reconstruction
Outline

• Motivation for In-System Debug

• Previous Work: In-System Debug Framework for HLS
  • Debug Instrumentation at compile time

• This paper: HLS Debug Overlay to allow customization at runtime

• Evaluation

• Future Work
Currently, the user selects the overlay + capabilities to insert.

• Next step – create a tool that automatically determines the type of overlay to insert based on estimated unused resources

The overlay is passive (i.e. only monitors the user circuit)

• Investigate limited controllability
• Allow for simple “what if” scenarios
Achieved software like compile times between debug turns in a limited context via an HLS oriented overlay

- Can personalize the overlay at runtime without a recompile

- Overlay supports a set of capabilities (selective variable/function tracing, conditional buffer freeze)

- Overheads are significant (335 ALMs for Variant B/G=2 line packer, 249 ALMs for C=1 unit) on top of the Baseline instrumentation

Worth it for the option to have software like compile times during debug
Thank you
Additional
## Previous Work – Instrumentation Overhead

<table>
<thead>
<tr>
<th>Circuit</th>
<th>User Module (ALMs)</th>
<th>Instrumentation (100%)</th>
<th>Proportion in Debug Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Fixed hlsd (ALMs)</td>
<td>Trace Scheduler (ALMs)</td>
</tr>
<tr>
<td>adpcm</td>
<td>7019</td>
<td>480</td>
<td>1749</td>
</tr>
<tr>
<td>aes</td>
<td>7135</td>
<td>479</td>
<td>754</td>
</tr>
<tr>
<td>blowfish</td>
<td>3038</td>
<td>528</td>
<td>1187</td>
</tr>
<tr>
<td>dfadd</td>
<td>3605</td>
<td>495</td>
<td>1115</td>
</tr>
<tr>
<td>dfdiv</td>
<td>6000</td>
<td>532</td>
<td>1124</td>
</tr>
<tr>
<td>dfmul</td>
<td>1881</td>
<td>483</td>
<td>675</td>
</tr>
<tr>
<td>dfsin</td>
<td>11864</td>
<td>529</td>
<td>2904</td>
</tr>
<tr>
<td>gsm</td>
<td>4147</td>
<td>473</td>
<td>782</td>
</tr>
<tr>
<td>jpeg</td>
<td>18735</td>
<td>506</td>
<td>2781</td>
</tr>
<tr>
<td>mips</td>
<td>1441</td>
<td>505</td>
<td>419</td>
</tr>
<tr>
<td>motion</td>
<td>6470</td>
<td>520</td>
<td>524</td>
</tr>
<tr>
<td>sha</td>
<td>1720</td>
<td>514</td>
<td>334</td>
</tr>
<tr>
<td>combined</td>
<td>66522</td>
<td>583</td>
<td>13525</td>
</tr>
<tr>
<td>Mean</td>
<td><strong>10736</strong></td>
<td><strong>509</strong></td>
<td><strong>2114</strong></td>
</tr>
</tbody>
</table>

Roughly ¼ is debug instrumentation.