Quantifying Observability for In-System Debug of High-Level Synthesis Circuits

Jeffrey Goeders
Steve Wilton

a place of mind

UBC

ALTERA

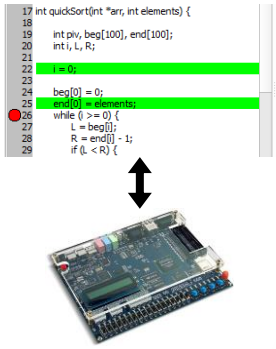
What this talk is about…

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

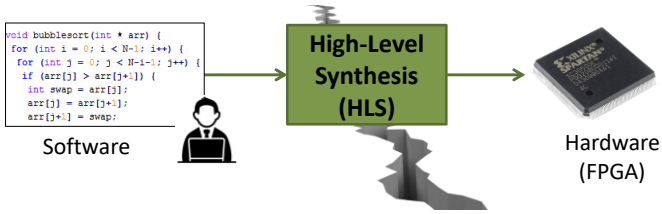
How do you measure the effectiveness of a debug tool?

This work: Quantifying observability into an HLS circuit

Use the metric to explore debugging techniques and trade-offs



High-Level Synthesis



Software designers need more than a compiler

- They need tools for testing, debugging, optimization….

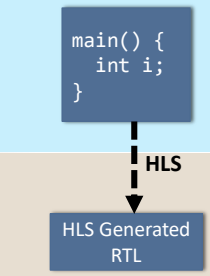
My PhD work: **Debugging HLS circuits**

Why this is challenging:

1. Circuit looks nothing like the original software
2. Debugging hardware is difficult – limited observability into chip

3

Bugs in HLS systems



Kernel-level bugs
- Self-contained
- Easy to reproduce

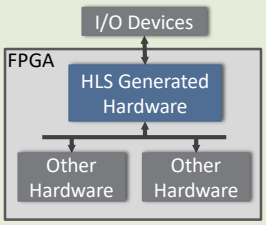
Debug C code on workstation (gdb).

Software

RTL Verification
- Verify RTL correctness
- Catch tool usage errors

Run C/RTL co-simulation on workstation.

Simulation



System-Level Bugs
- Bugs in interfaces
- Dependent on I/O traffic
- Hard to reproduce, or require long run times

Debug on FPGA

(Requires observing internals of FPGA)

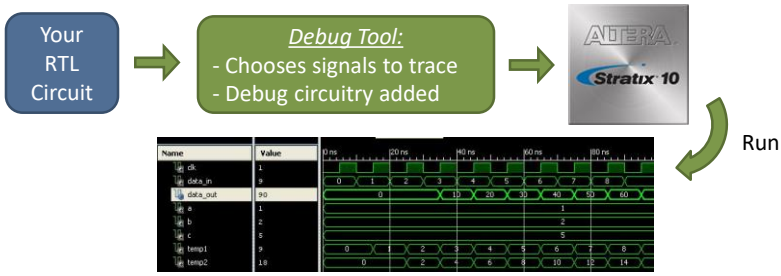
How do you observe these bugs?

Hardware

4

Can We Use Hardware Debug Tools?

Embedded Logic Analyzer
(SignalTap/Chipscope):



Designer is forced to debug using the RTL, which is nothing like the 'C' code

Our Approach

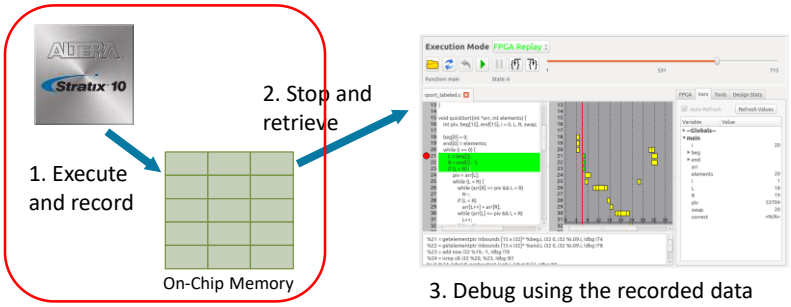
1. A software-like debugger running on a workstation
   - Single-stepping, breakpoints, inspect variables
2. Interacting with the circuit on the FPGA
   - Capture system-level bugs in the real operating environment

Key: If we want to capture system bugs, the circuit needs to execute at normal speed (MHz)

- Makes 'interactive debugging' impossible

Solution: <u>Record and Replay</u>

- Record circuit execution on-chip, retrieve, debug using the recorded data

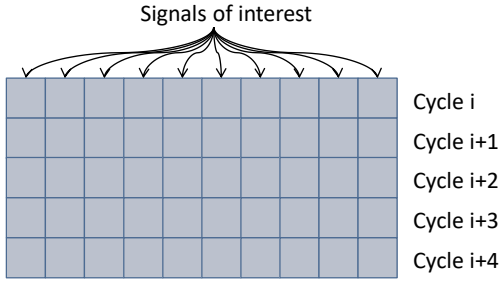


Limited on-chip memory: Can only observe a small portion of entire exectuion

7

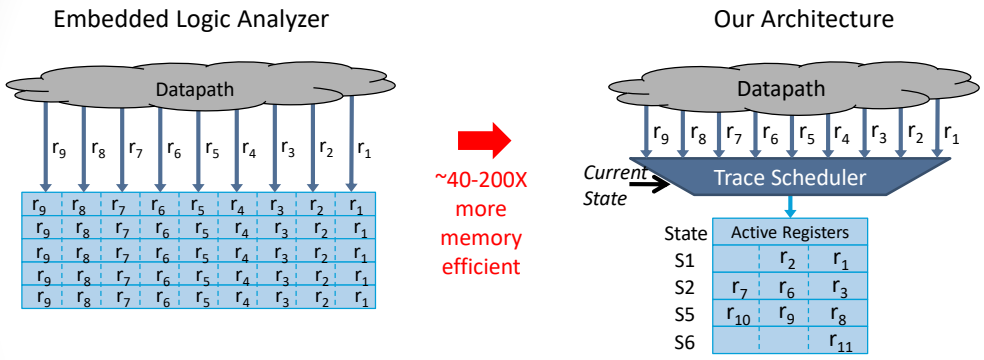
Embedded Logic Analyzers

- Example: Chipscope/Signaltap
- Record (trace) signals into on-chip memory
- Trace Buffers
  - Memory configured as a cyclic buffer
  - Each cycle, store samples of all signals of interest



8

Leveraging the HLS Information



Dynamically change which signals are recorded each cycle

- HLS schedule is used to only record variable updates
- Longer execution trace → Find bugs faster

9

HLS Observability

Usually not possible to provide "complete observability"

- Limited on-chip memory
- What data should be given to the user? What should be ignored?

Why have an observability metric?

- Compare and contrast debug techniques; understand relative strengths
- Toward debug techniques tailored to the design/bug

Observability metrics have been proposed for RTL circuits

- Issue: 'RTL' observability not meaningful in the software domain

Need an observability metric for HLS circuits, based upon the original software code.

10

Observability Metric

What does our metric measure?

- **As a user steps through a program, how often are the values of variable accesses available?**

Why this approach?

- Recent debug work: software-like debug experience

We define Observability as:

$$Observability = Availability \cdot Duration$$

How many cycles is the data available for?

What percentage of variable accesses have recorded values available to the user?

Observability Metric

$$Observability = Availability \cdot Duration$$

$$Availability\ (A) = \frac{\sum_{i \in var} f_i \cdot v_i}{\sum_{i \in var} f_i \cdot a_i}$$

v_i: Variable accesses with known value
a_i: Total number of variable accesses
f_i: Variable favorability coefficient

$Duration = e_{tb} \cdot Memory\ Size$ (kb)

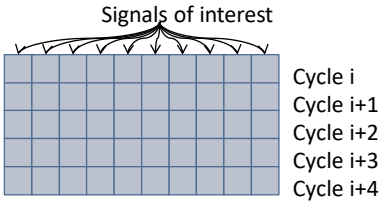
e_{tb}: Memory efficiency (cycles captured per kB of memory)

$$Observability\ per\ kb = A \cdot e_{tb}$$

Observability provided by an Embedded Logic Analyzer

$Observability\ per\ kb = A \cdot e_{tb}$

- A = 100%
- $e_{tb} = \frac{1k}{\#\ Bits\ Traced}$



Methodology:

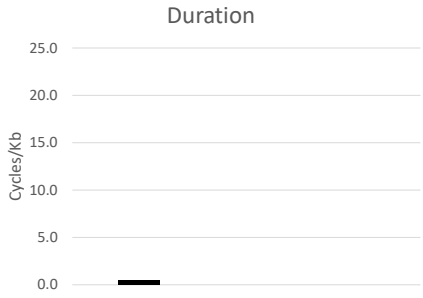
- CHStone benchmarks, LegUp 4.0
- Record ALL 'C' variables

Result:

- $Observability\ per\ kb = 100\% \cdot 0.5\ cycles/kb$

13

Observability Results



		Availability		*Duration*	*vs. ELA*
1.	Embedded Logic Analyzer	100%	·	0.5cyl/kb	1x

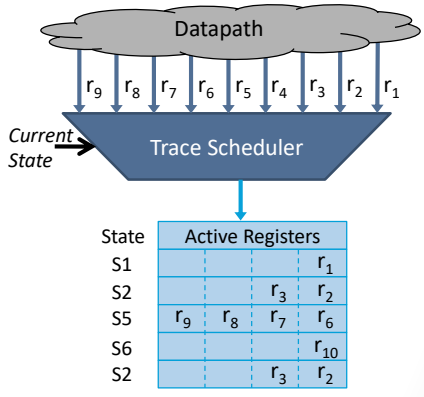
14

Observability of Dynamic Tracing Scheme

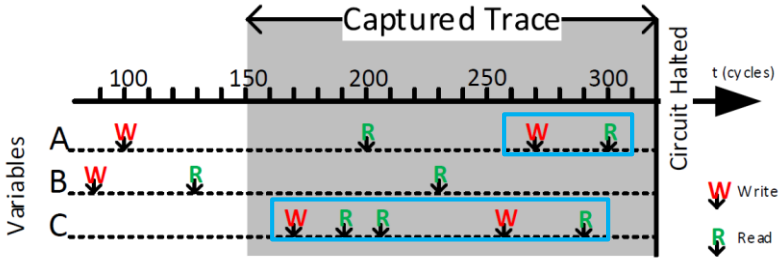
Our recent work:

- Use HLS schedule to only record variable updates

If we record all variable updates, is *Availability* 100%?



Issue with Only Recording Updates



Variables updates may occur outside of captured trace

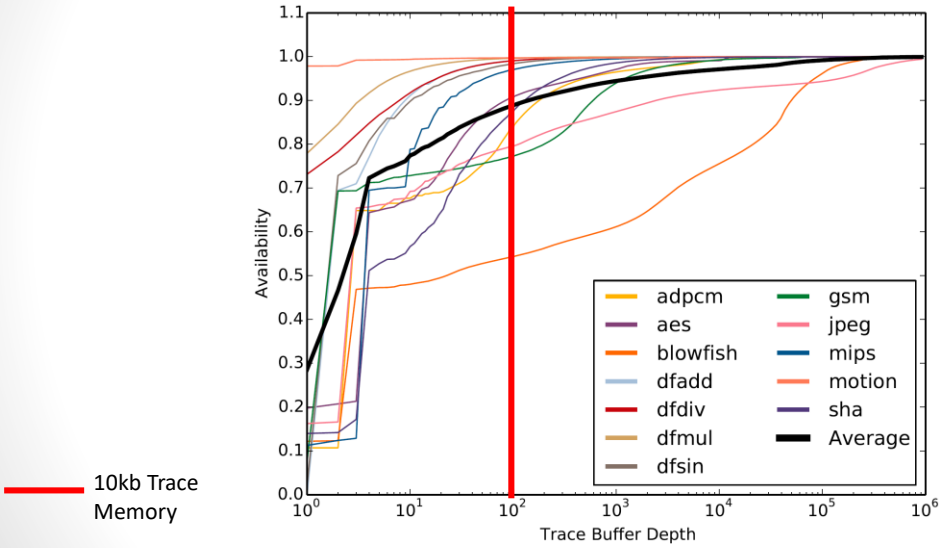
- During debug, these variable values are not available to the user

$$A = {}^{7}/_{9} = 78\%$$

More likely to occur if:

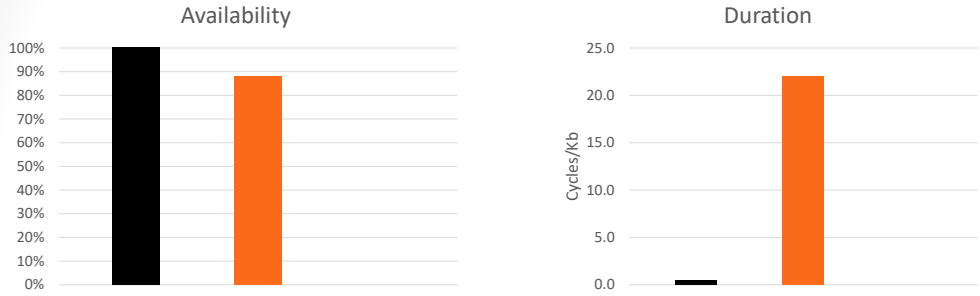
- Long gaps of time from update to access
- Trace buffers are small

Availability (%) – Record Updates Only



17

Observability Results



		Availability		*Duration*	*vs. ELA*
1.	Embedded Logic Analyzer	100%	·	0.5cyl/kb	1x
2.	Record “Updates”	88%	·	22.0cyl/kb	38x

18

Which variables cause this issue?

```
#define N 100

int matrix_multiply(int * fifo_in) {
  int i, j, k, sum;
  int A[N][N], B[N][N], C[N][N];

for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    A[i][j] = *fifo_in;

for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    B[i][j] = *fifo_in;;

  for (i = 0; c < m; c++) {
    for (j = 0; d < q; d++) {
      sum = 0;
      for (k = 0; k < p; k++) {
        sum += A[i][k]*B[k][j];
      }
      C[i][j] = sum;
    }
  }
  return 0;
}
```

Local/Scalar Variables:

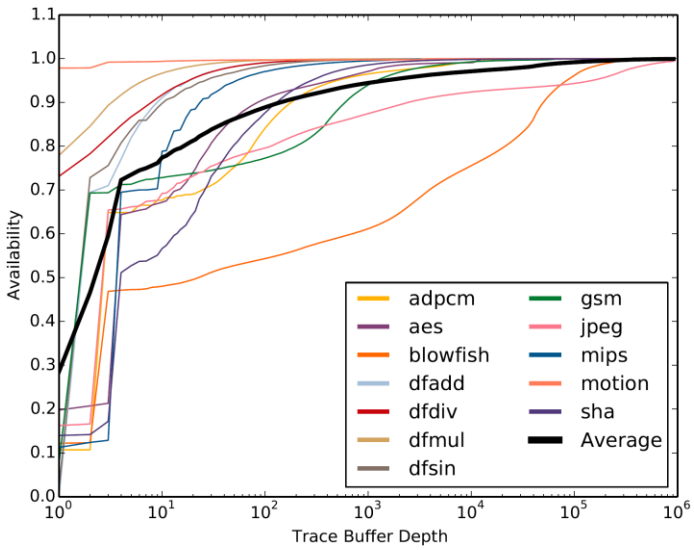
- Shorter lifespan, often accessed soon after updating
- Typically mapped to registers in the hardware

Global/Vector Variables:

- Longer lifespan, may be accessed long after being initialized/updated
- Typically mapped to memories in the hardware

19

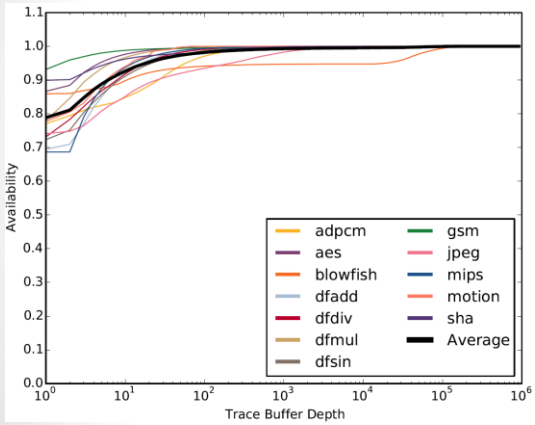
Availability (%) – Record Updates Only



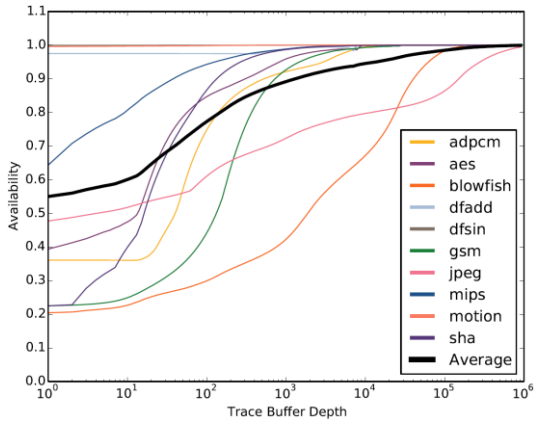
20

Availability (%) – Record Updates Only

Variables in Registers



Variables in Memory



Recording "Updates Only" works well for variables in registers, but has issues for variables in memory

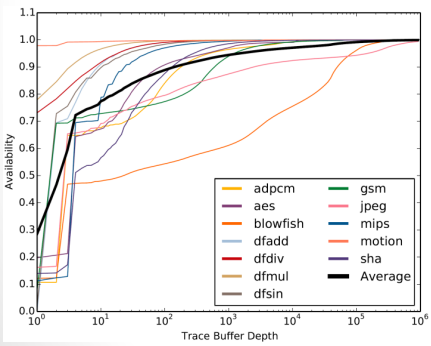
Availability (%) – Record Updates + Memory Reads

Record when variables are read as well as written

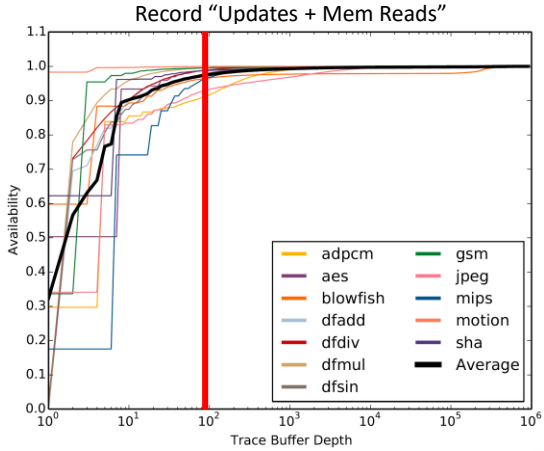
- First, consider memory reads only
- Provides better availability (at a cost of duration)

10kb Trace Memory

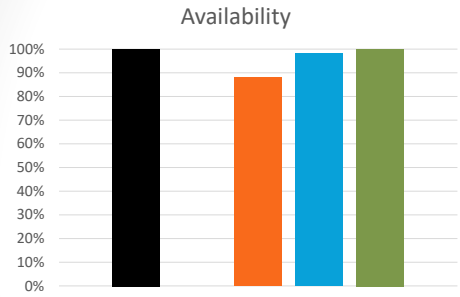
Record "Updates Only"

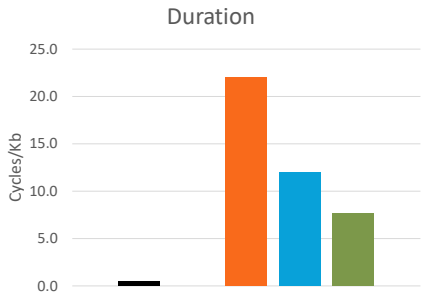


Record "Updates + Mem Reads"



Observability Results





		Availability		*Duration*	*vs. ELA*
1.	Embedded Logic Analyzer	100%	·	0.5cyl/kb	1x
2.	Record "Updates"	88%	·	22.0cyl/kb	38x
3.	Record "Updates + Mem Reads"	98%	·	12.0cyl/kb	24x
4.	Record "Updates + Reads"	100%	·	7.7cyl/kb	14x

23

Observing a Subset of Variables

What happens to observability if we only observe a subset of variables? 10%? 90%?

Selecting RTL signals for an Embedded Logic Analyzer → Predictable effect on observability

Selecting 'C' variables to observe → non-uniform effect on observability:

- Bit-width minimization
- 1 Variable in C code → Many signal in hardware:
  - LLVM SSA form creates new register/signal for each assignment
- Many Variables in C code → 1 Signal in hardware:
  - Function parameters
  - In-lining

24

Variable Selection Experiment

Test different variable selection methods and measure availability and duration

Methodology:

- Sweep % of signal traced, from 10% to 100%
- Record “Updates Only”

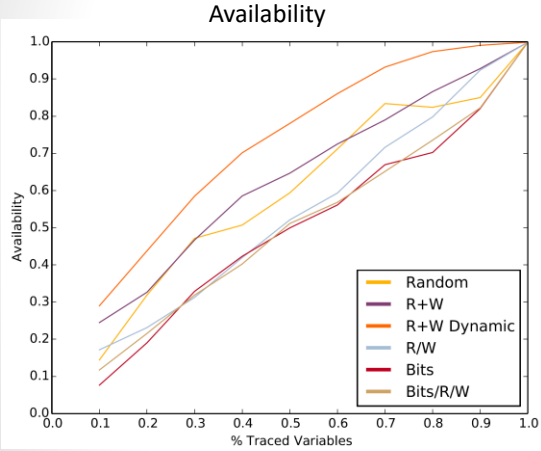
Variable selection methods:

1. Random: Random selection of variables
2. R+W Static: Variables that are read or written most often (Static analysis)
3. R+W Dynamic: Variables that are read or written most often (Dynamic analysis)
4. R/W: Select variables with highest read/write ratio.
5. Bit Width: Select variables with smallest bit width

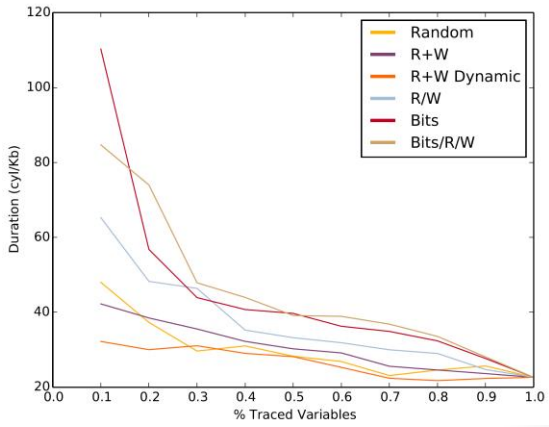
25

Variable Selection Results

Availability



Duration



26

Impact of Results

Different signal-tracing techniques provide observability trade-offs

- Record updates only → Long duration, some variable values unavailable to user

Selecting variables for observation → non-uniform cost

Can we tailor HLS debugging methods to:

- Circuit characteristics?
- Type of bug/issue?

Vision: Automatic analysis for optimal debugging technique

27

Summary

- HLS users require a full eco-system of tools, including effective debuggers
- Metric for in-system observability of an HLS circuit
- Debugging techniques provided varied observability characteristics
- This is an important step to:
  - Developing effective HLS debuggers
  - Understanding what techniques are best suited for certain debug problems

28