Using Round-Robin Tracepoints to Debug Multithreaded HLS Circuits on FPGAs

Jeffrey Goeders
Steve Wilton

NSERC CRSNG
ALTERA
INVIONICS

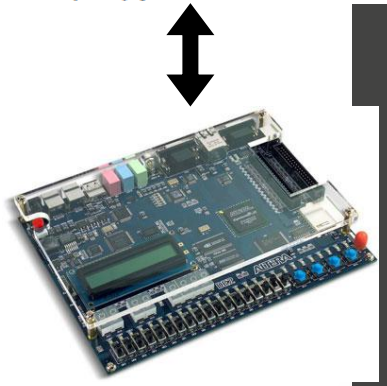
What this talk is about…

How to allow designers to debug HLS circuits?

In past work we developed a debugging tool:

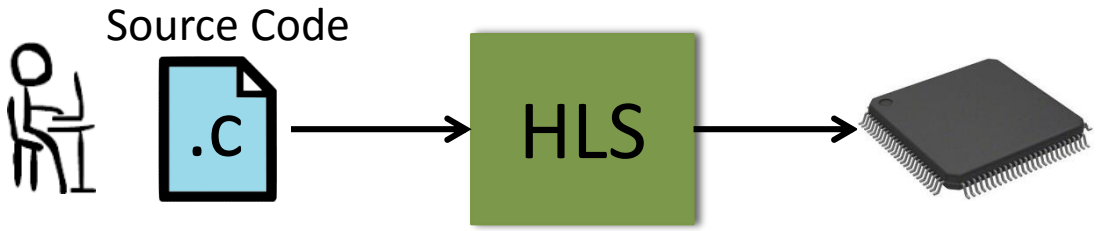
- Software-like debugger
- Interfaces with an HLS circuit
- Only worked for single-threaded C code

```
17 int quickSort(int *arr, int elements) {
18
19     int piv, beg[100], end[100];
20     int i, L, R;
21
22     i = 0;
23
24     beg[0] = 0;
25     end[0] = elements;
26     while (i >= 0) {
27         L = beg[i];
28         R = end[i] - 1;
29         if (L < R) {
```



What needs to change to support multithreaded code?

High-level synthesis



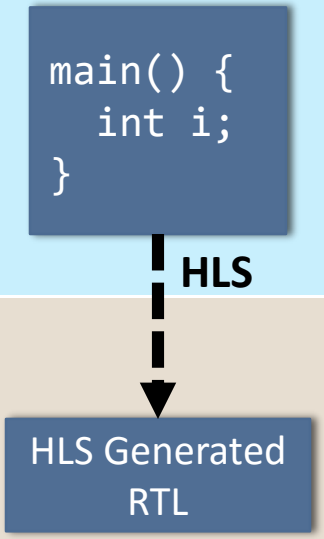
1. Faster development
2. Software designers target FPGAs

Software designers need a full ecosystem of tools

- Testing
- Debugging
- Optimization
- ….

In this talk, I am going to focus on debugging

Bugs in HLS systems



Kernel-level bugs

- Self-contained
- Easy to reproduce

Debug C code on workstation (gdb).

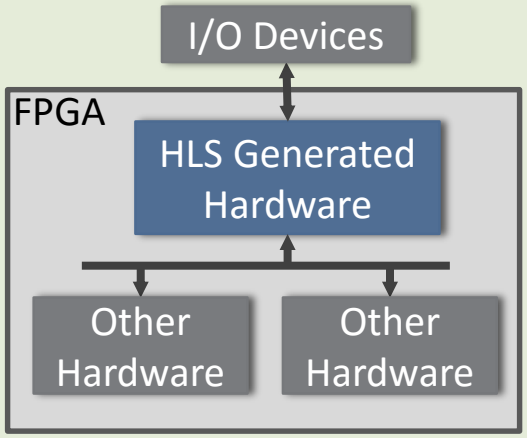
Software

RTL Bugs & RTL Verification

- Incorrect use of HLS tools

Run C/RTL co-simulation on workstation.

Simulation



System-Level Bugs

- Bugs in interfaces
- Dependent on interaction timings
- Hard to reproduce
- Require long run-times

Debug on FPGA

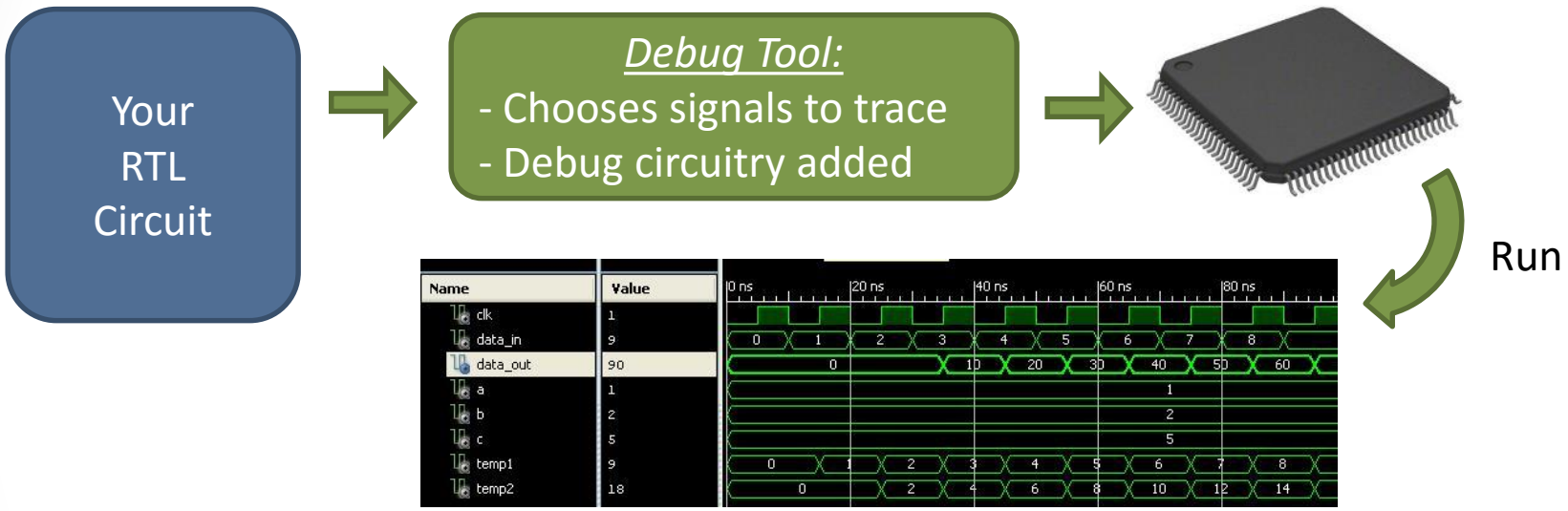
(Requires observing internals of FPGA)

These are the bugs we are targeting.

Hardware

General Hardware Debug

Commercial ELA tools (SignalTap II / Chipscope Pro):



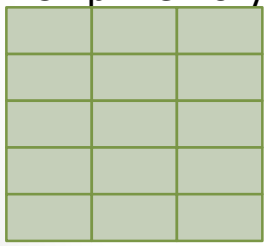
Can we use these tools to debug HLS circuits? **No**

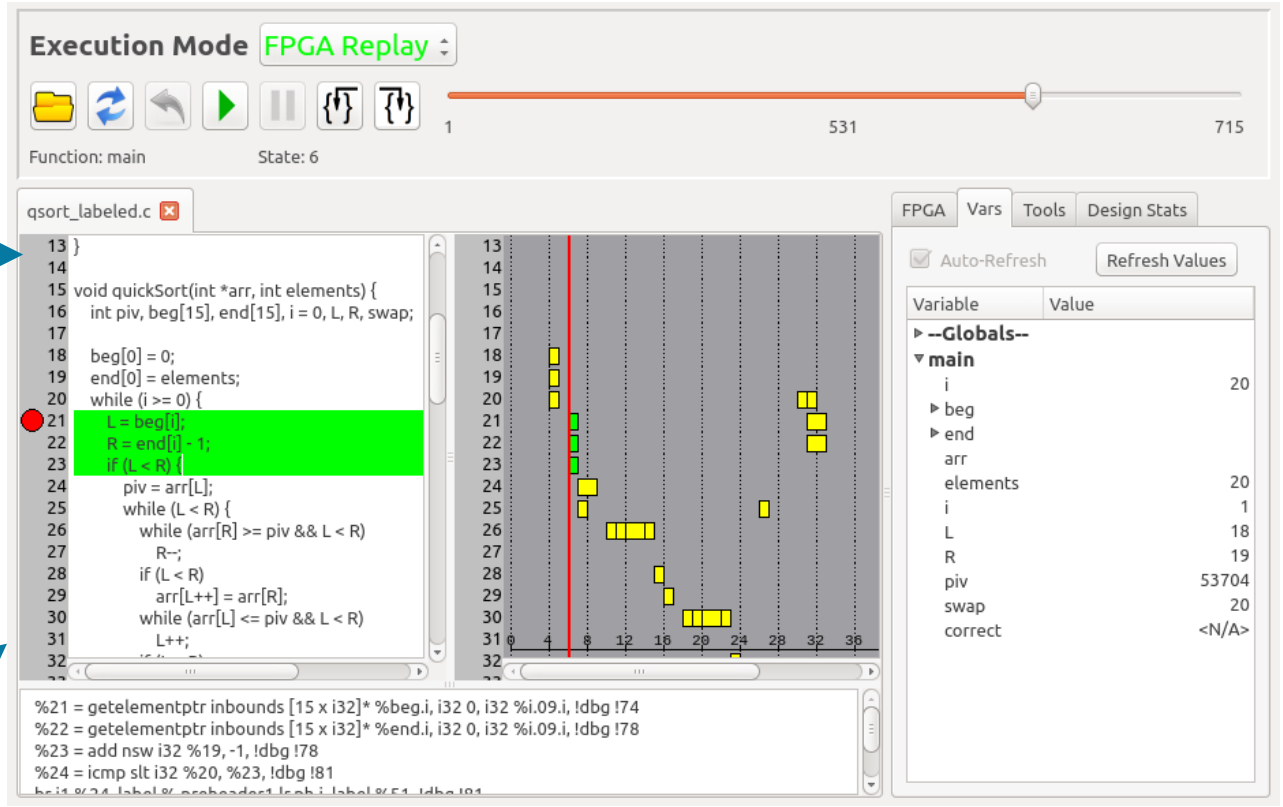
1. *Software designers?* Beyond their expertise
2. *Hardware designers?* Forces them to give up higher-level abstraction
3. *It's very hard.* HLS transformations means RTL doesn't look like the C code

Previous Work

Trace Buffer

(On-Chip Memory)

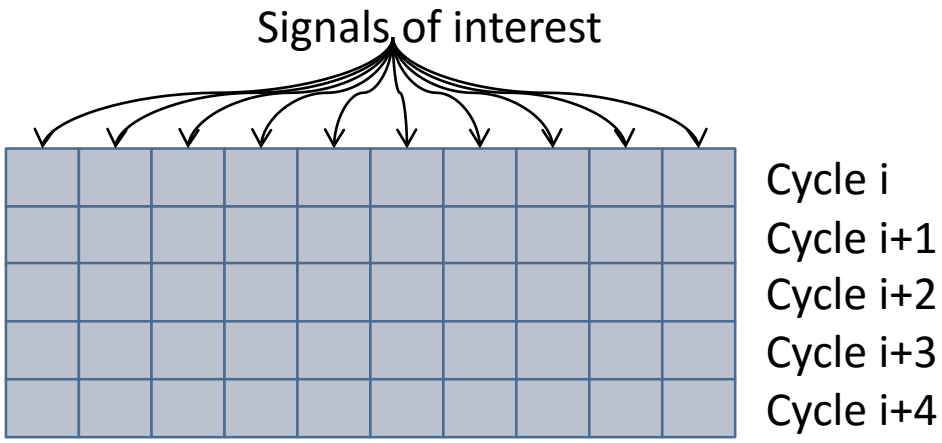




- Software-like debugging experience
- Circuit runs at full speed (record, then retrieve)
- Only works with single-threaded C code

Recording Execution: Trace Buffers

To record live circuit execution we use trace buffers:



Trace buffers require a lot of memory, and on-chip memory is scarce

- Can't record entire circuit execution

Width x Depth tradeoff:

- The more signals we record, the shorter the execution trace

What's New? Multithreaded HLS Circuits

User provides multithreaded source code (OpenCL, pthreads)

- Hardware modules execute in parallel

This is the future of HLS!

- Exploits parallelism of FPGA
- Altera OpenCL SDK, Xilinx SDAccel

Leads to more complicated in-system bugs:

- Thread communication/synchronization
- Race conditions
- Deadlocks
- Performance issues (thread imbalance, starvation, etc.)

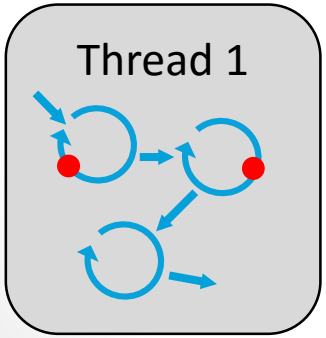
Debugging Multithreaded HLS Circuits

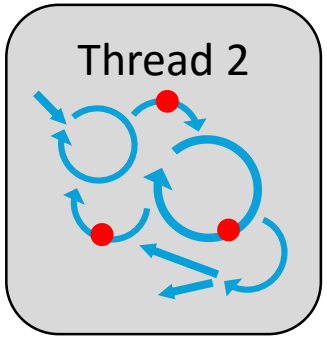
Need to record long run-times to expose these bugs

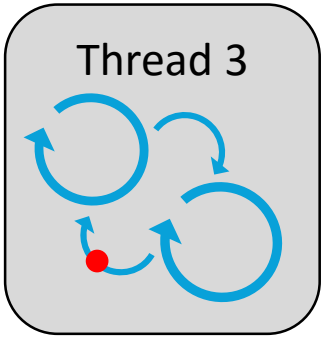
- Must record fewer signals
- Can't do GDB/Eclipse-like debug
- Need a debug framework for partial observability

Solution: **Tracepoint-based debugging**

- Like a breakpoint, but instead of stopping it just adds to a log.
- One or more tracepoints per thread
- *Optionally* record variables at the tracepoint.







Tracepoint Log

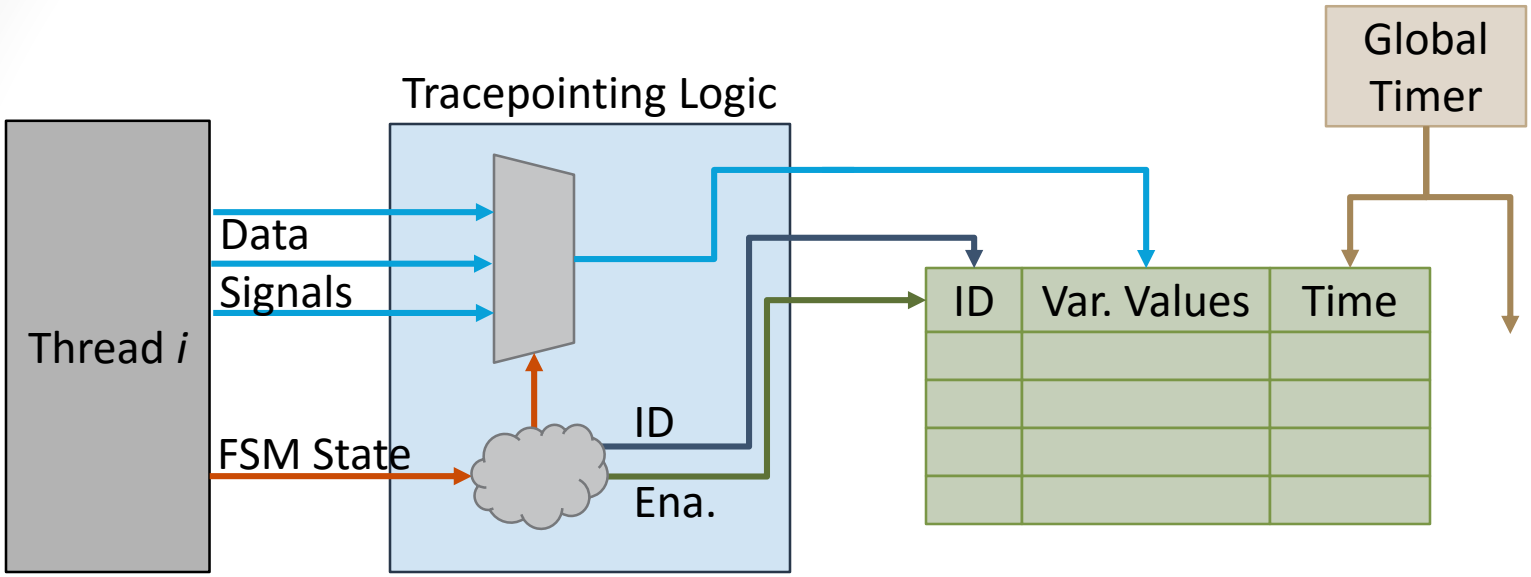
Tracepoint data is provided to the user as a timeline:

Thread ID	Cycle #	Tracepoint	Variables Logged
1	26452	main(): 113	mutex = 0
2	28591	foo(): 41	x = 7, y = 3
4	28037	bar(): 3	<None>
…	…	…	…

We add debug circuitry to record tracepoints into trace buffers:

- Which tracepoint – ID
- Time
- Variable values

Basic Tracepoint Architecture

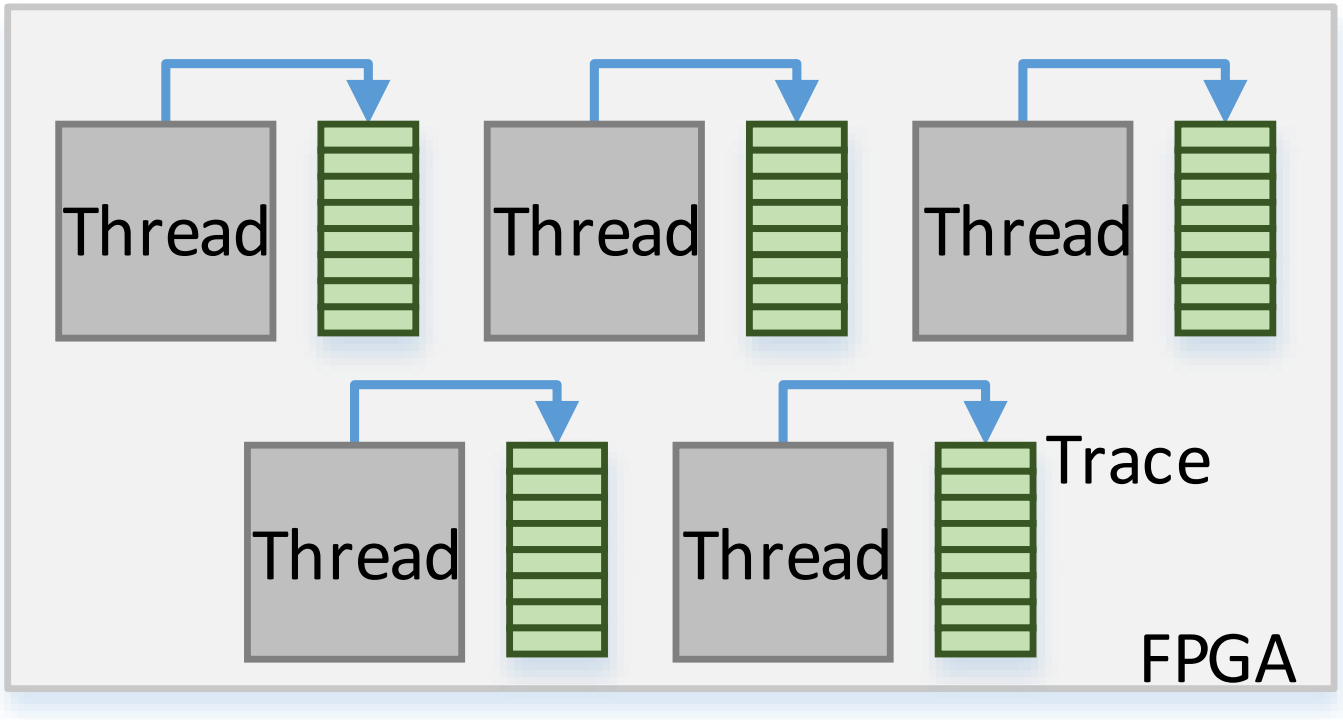


This is a direct extension of previous work, except only a subset of variables are recorded

Much more efficient than an ELA (Chipscope Pro, SignalTap II)

- Variable values are multiplexed
- Add buffer entries buffer only when needed

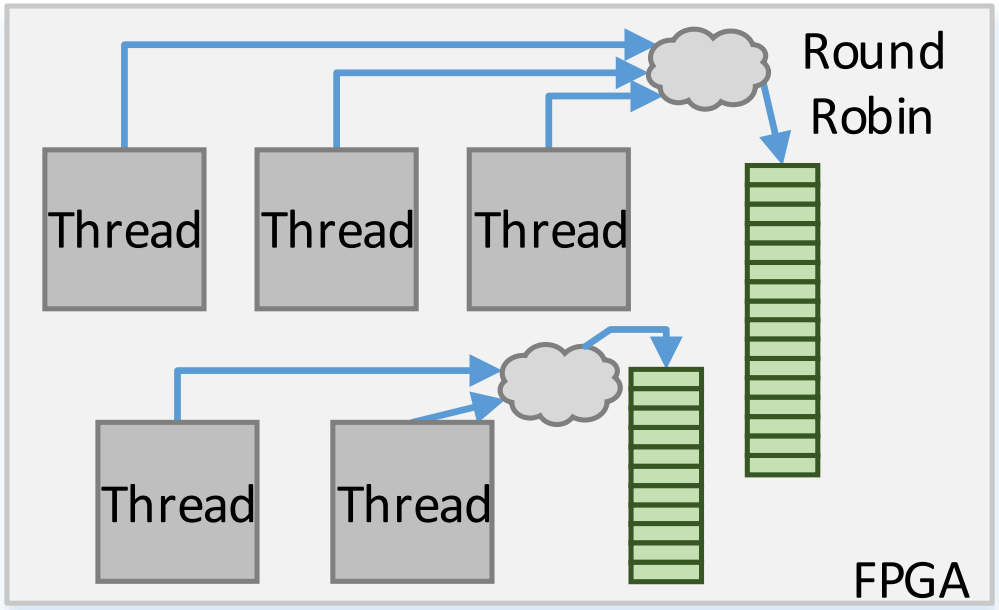
The architecture is duplicated to every thread in the system:



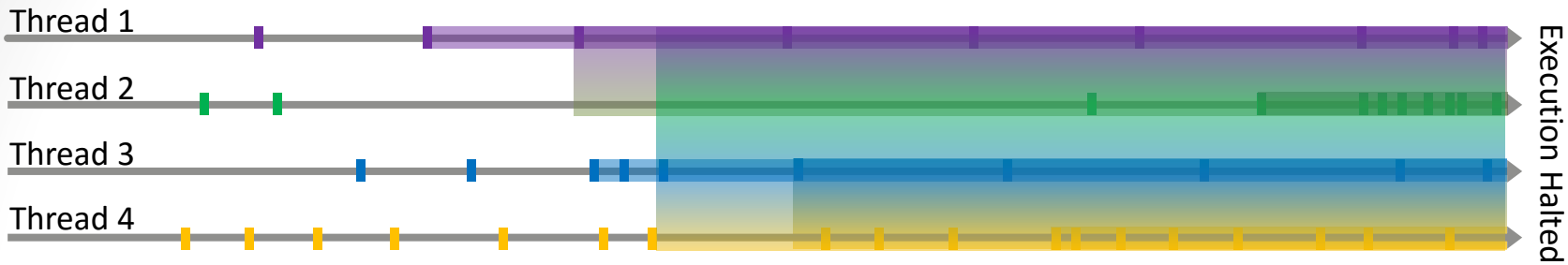
Giving each thread their own trace buffer is not ideal:

- Each thread will fill buffers at different rates
- Hard to predict this before-hand
- Leads to wasted memory space

Buffer Sharing



Effects of Multiple Trace Buffers



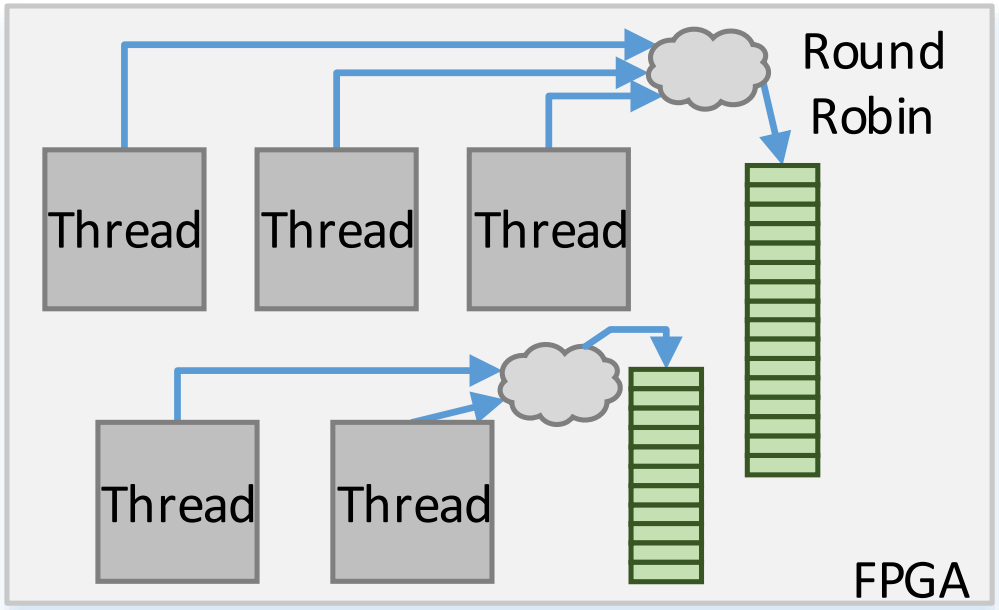
Case 1: 8-entry buffer per thread
Case 2: Two 16-entry buffers
Case 3: One 32-entry buffer

Full-System Execution Trace

For multiple trace buffers, execution trace length is the **minimum** captured in any buffer

Key: More buffer sharing = longer execution traces

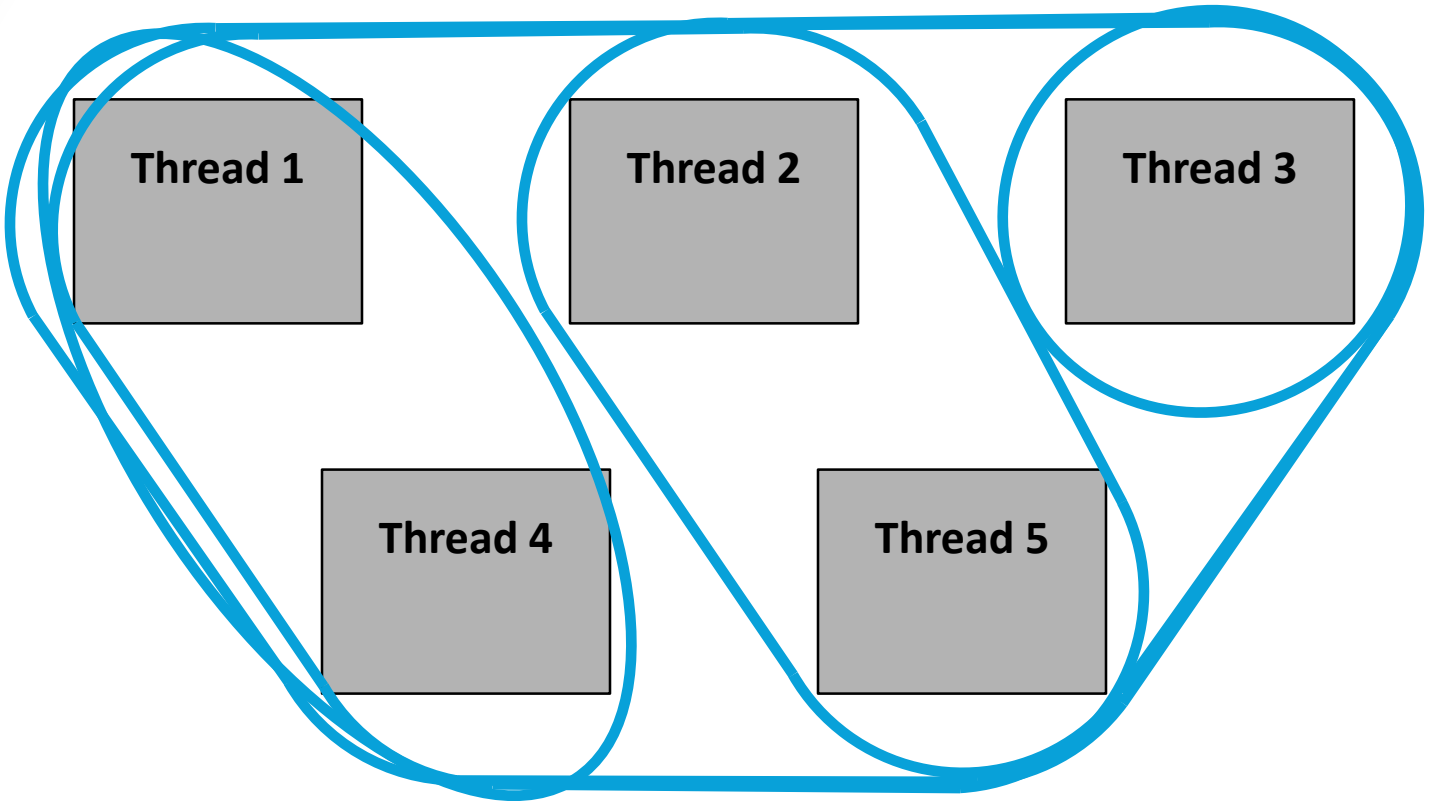
Buffer Sharing



To support buffer sharing, we need to handle:

1. Grouping threads
2. Arbitration

Grouping Threads



Want to group as aggressively as possible

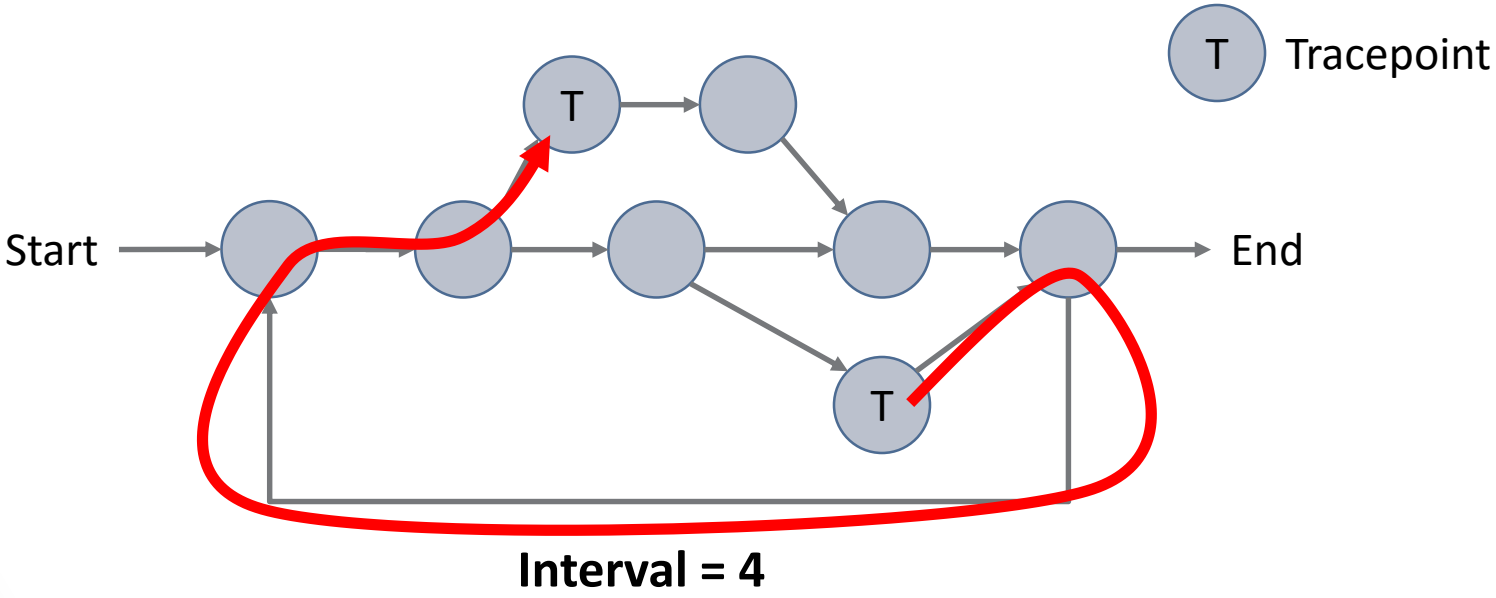
- If group encounters too many tracepoints, memory will be overwhelmed

Need to know how often a thread will encounter tracepoints…

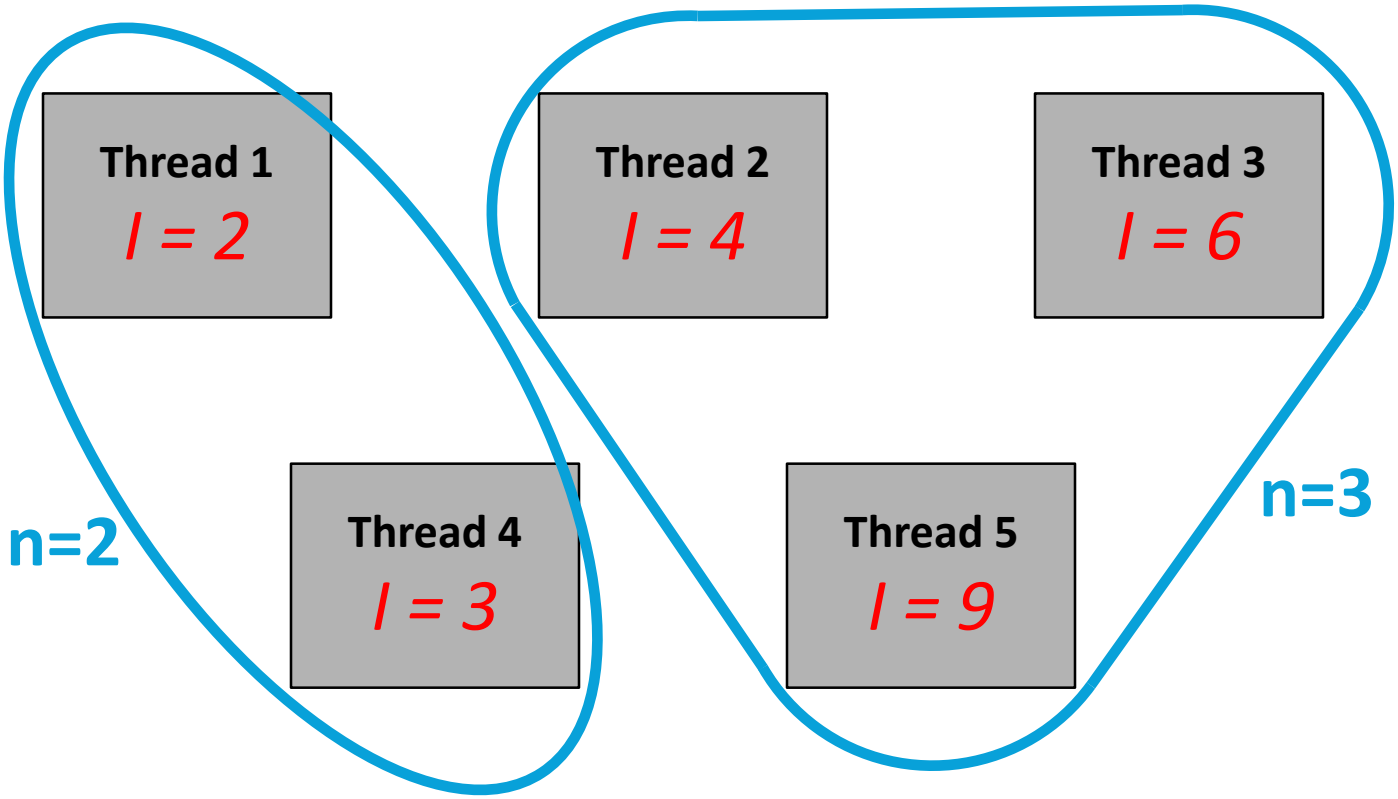
Tracepoint Interval

Use Dijkstra's algorithm on the CDFG

- Tracepoint Interval: minimum # of cycles between tracepoint hits



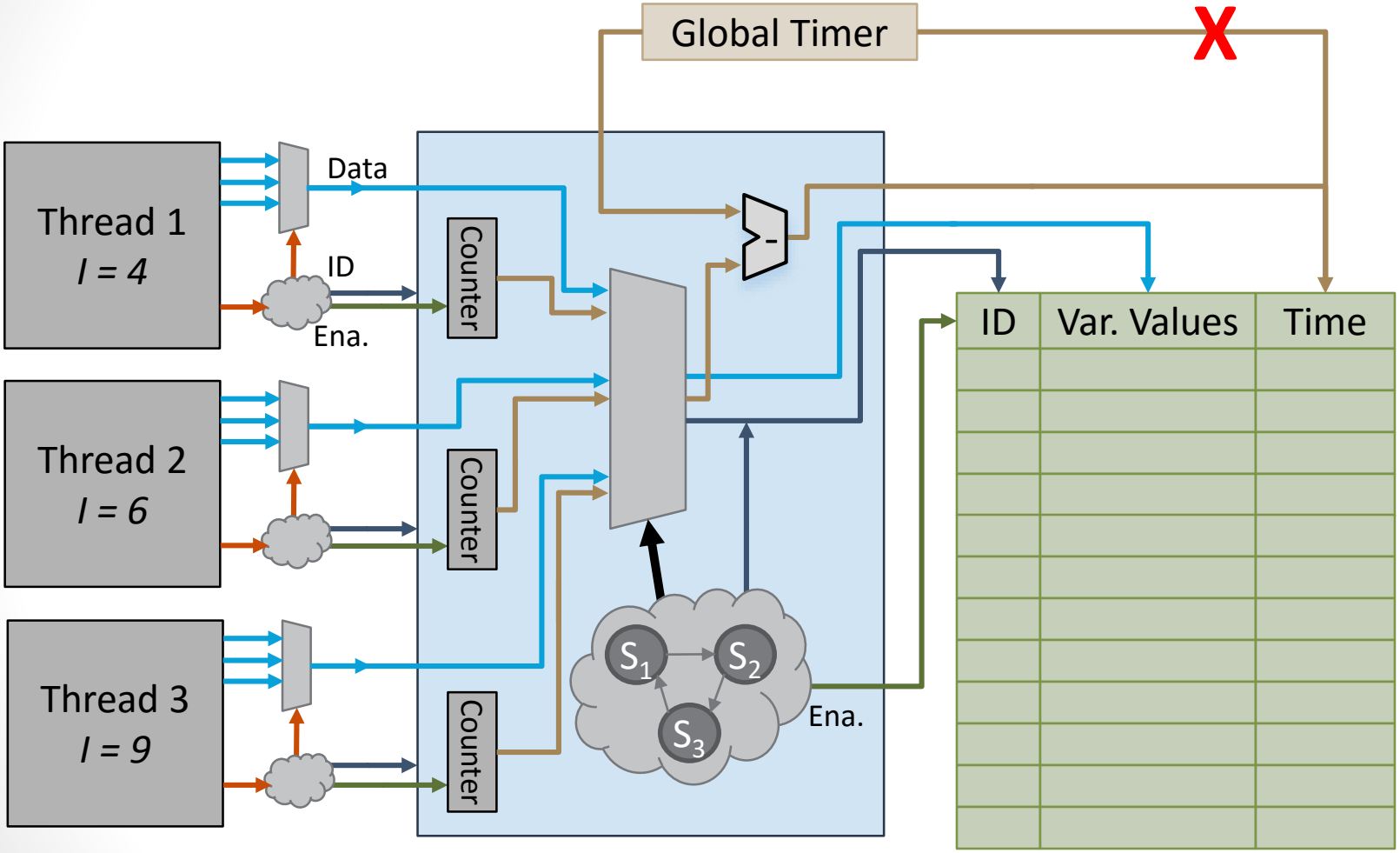
Grouping Threads



Group such that: ***Interval ≥ n***

- Allows for simple round robin arbitration

Round-Robin Arbitration Circuit

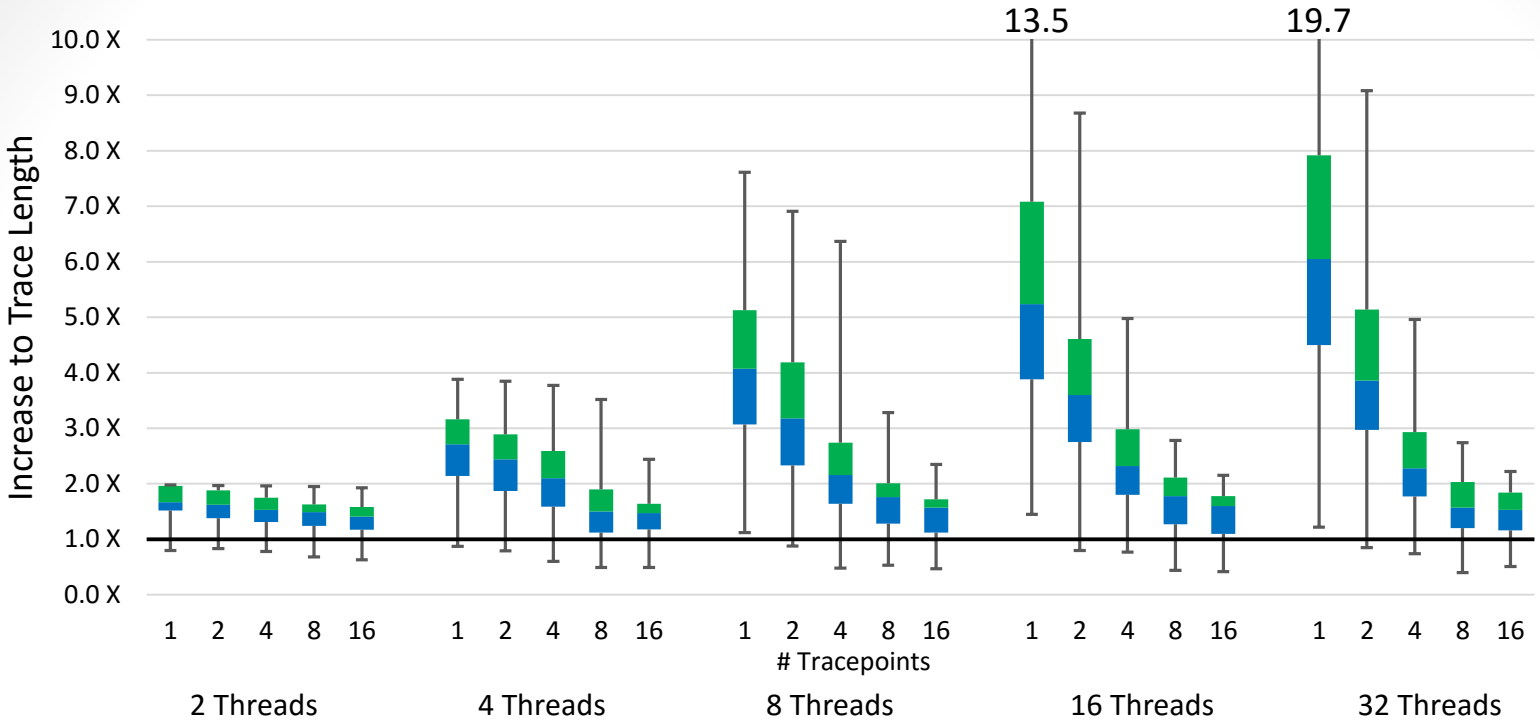


Benefit of Buffer Sharing?

Metric: How much does buffer sharing increase the length of the recorded execution trace?

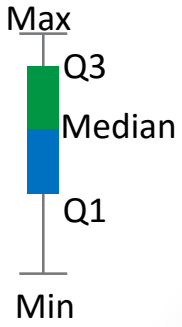
$$\frac{\#\ Cycles\ recorded\ \boldsymbol{with}\ Buffer\ Sharing}{\#\ Cycles\ recorded\ \boldsymbol{without}\ BufferSharing}$$

Longer trace length → Easier for designers to find multithreaded bugs



1000 trials per configuration:

- Randomly create synthetic benchmark from CHStone
- LegUp to synthesize
- Group threads and measure improvement



8 threads, 1 tracepoint each: 4.1X increase to trace length

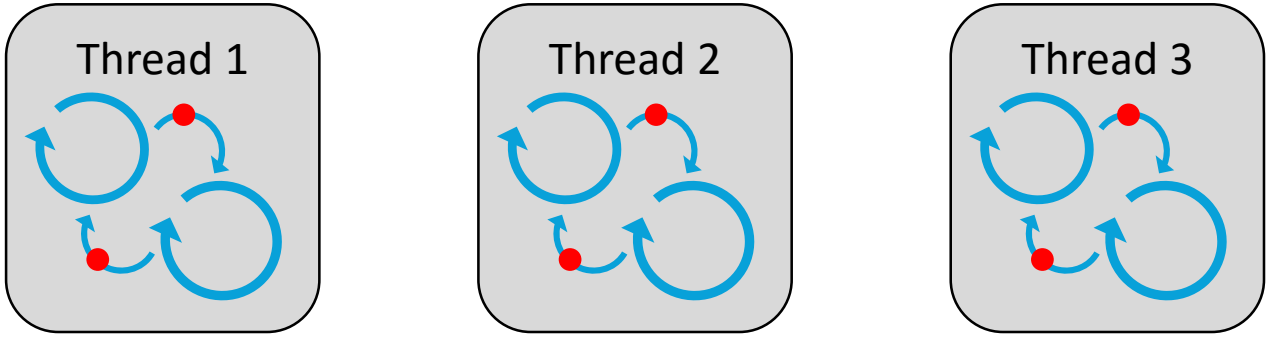
Issues and Limitations

Works best with few tracepoints per thread

- More tracepoints = smaller tracepoint interval, less buffer sharing.

Buffer sharing only benefits heterogeneous threads (i.e. task-parallel)

- In a data-parallel system (e.g. OpenMP) all threads encounter tracepoints at the same rate

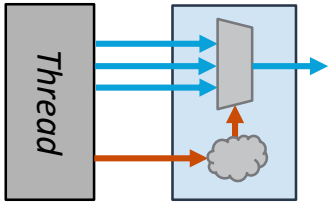


Task-parallel programs are more likely to need in-system debug

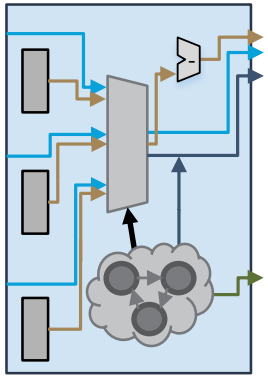
- Thread balancing, starvation, synchronization, etc.

Area Overhead

Tracepointing Logic



Round-Robin Logic



Data Width	# Tracepoints				
	1	2	4	8	16
16	0	17	19	53	90
32	0	33	35	101	170
64	0	64	67	197	330

Stratix IV ALUTs, per thread

Data Width	# Threads				
	2	4	8	16	32
16	29	20	22	21	23
32	37	24	30	29	32
64	53	32	46	45	49

Stratix IV ALUTs, per thread

8 threads, with two 32-bit tracepoints each: 63 ALUTs/per thread.

Summary

- HLS may change the face of hardware design for FPGAs
  - But, only if we have a suitable eco-system
  - Need to find elusive bugs in multithreaded systems
- Tracepoint-based debugging architecture for multithreaded HLS circuits
  - Run in-system, at-speed
  - Familiar to software designers
- Round-robin architecture
  - Record a longer execution trace: Find bugs faster

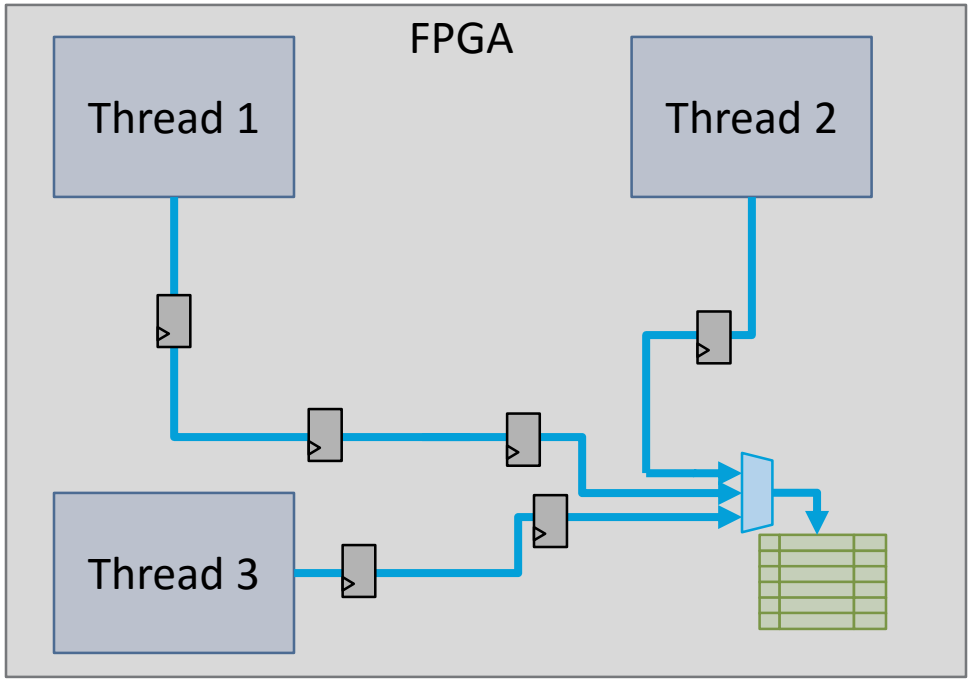
Timing Considerations

Grouped threads may not be physically adjacent

- Long routing lengths will affect timing.

Pipeline the data signals

- Subtract the pipeline depth from *time* during offline re-ordering



Pipeline Model