DESIGN AUTOMATION CONFERENCE

JULY 10 - 14, 2022

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Bringing Source-Level Debugging Frameworks to Hardware Generators

Keyi Zhang, Zain Asgar, Mark Horowitz
Computer Science Department
Stanford University

The Good, the Bad and the Ugly

- The Good:
  - Huge leaps in front-end design tools productivity
    - Hardware generator frameworks embedded in a host programming languages, such as Chisel
    - High level synthesis tools that turn C/C++ into RTL
  - More software-oriented concepts/constructs
    - Object-oriented programming
    - Functional programming
    - Software/hardware co-design

The Good, the Bad and the Ugly

- The Good:
  - Huge leaps i
    - Hardware g ...ages, such as Chisel
    - High level s
  - More softwa
    - Object-orie
    - Functional
    - Software/h

```
// SLT, SLTU
val slt =
  Mux(io.in1(xLen-1) === io.in2(xLen-1), io.adder_out(xLen-1),
  Mux(cmpUnsigned(io.fn), io.in2(xLen-1), io.in1(xLen-1)))
io.cmp_out := cmpInverted(io.fn) ^ Mux(cmpEq(io.fn), in1_xor_in2 === UInt(0), slt)

// SLL, SRL, SRA
val (shamt, shin_r) =
  if (xLen == 32) (io.in2(4,0), io.in1)
  else {
    require(xLen == 64)
    val shin_hi_32 = Fill(32, isSub(io.fn) && io.in1(31))
    val shin_hi = Mux(io.dw === DW_64, io.in1(63,32), shin_hi_32)
    val shamt = Cat(io.in2(5) & (io.dw === DW_64), io.in2(4,0))
    (shamt, Cat(shin_hi, io.in1(31,0)))
  }
val shin = Mux(io.fn === FN_SR  || io.fn === FN_SRA, shin_r, Reverse(shin_r))
```

Chisel code for RocketChip

59 DESIGN AUTOMATION CONFERENCE

The Good, the Bad and the Ugly

- The Bad and the Ugly:
  - Generated RTL is obfuscated due to compiler optimizations
    - Low-level RTL
    - Loses designer intent
  - Verification has to be done at RTL level for integration tests
  - Productivity gain from front-end design is lost in verification

59 DESIGN AUTOMATION CONFERENCE

The Good, the Bad and the Ugly

- The Bad
  - Generate
    - Low-le
    - Loses
  - Verificati
  - Producti

```
wire [63:0] _in2_inv_T_1 = ~io_in2; // @[ALU.scala 61:35]
wire [63:0] in2_inv = io_fn[3] ? _in2_inv_T_1 : io_in2; // @[ALU.scala 61:20]
wire [63:0] in1_xor_in2 = io_in1 ^ in2_inv; // @[ALU.scala 62:28]
wire [63:0] _io_adder_out_T_1 = io_in1 + in2_inv; // @[ALU.scala 63:26]
wire [63:0] _GEN_1 = {{63'd0}, io_fn[3]}; // @[ALU.scala 63:36]
wire  _slt_T_7 = io_fn[1] ? io_in2[63] : io_in1[63]; // @[ALU.scala 68:8]
wire  slt = io_in1[63] == io_in2[63] ? io_adder_out[63] : _slt_T_7; // @[ALU.scala 67:8]
wire  _io_cmp_out_T_2 = ~io_fn[3]; // @[ALU.scala 44:26]
wire  _io_cmp_out_T_4 = _io_cmp_out_T_2 ? in1_xor_in2 == 64'h0 : slt; // @[ALU.scala 69:41]
wire  _T_2 = io_fn[3] & io_in1[31]; // @[ALU.scala 76:46]
wire [31:0] _T_4 = _T_2 ? 32'hffffffff : 32'h0; // @[Bitwise.scala 72:12]
wire [31:0] hi = io_dw ? io_in1[63:32] : _T_4; // @[ALU.scala 77:24]
wire  hi_1 = io_in2[5] & io_dw; // @[ALU.scala 78:33]
wire [4:0] lo = io_in2[4:0]; // @[ALU.scala 78:60]
wire [5:0] shamt = {hi_1,lo}; // @[Cat.scala 30:58]
wire [31:0] lo_1 = io_in1[31:0]; // @[ALU.scala 79:34]
wire [63:0] shin_r = {hi,lo_1}; // @[Cat.scala 30:58]
wire  _shin_T_2 = io_fn == 4'h5 | io_fn == 4'hb; // @[ALU.scala 81:35]
wire [63:0] _shin_T_6 = {{32'd0}, shin_r[63:32]}; // @[Bitwise.scala 103:31]
wire [63:0] _shin_T_8 = {shin_r[31:0], 32'h0}; // @[Bitwise.scala 103:65]
wire [63:0] _shin_T_10 = _shin_T_8 & 64'hffffffff00000000; // @[Bitwise.scala 103:75]
wire [63:0] _shin_T_11 = _shin_T_6 | _shin_T_10; // @[Bitwise.scala 103:39]shin_T_16 | _shin_T_20; // @[Bitwise.scala
103:39]
wire [63:0] _GEN_3 = {{8'd0}, _shin_T_21[63:8]}; // @[Bitwise.scala 103:31]
```

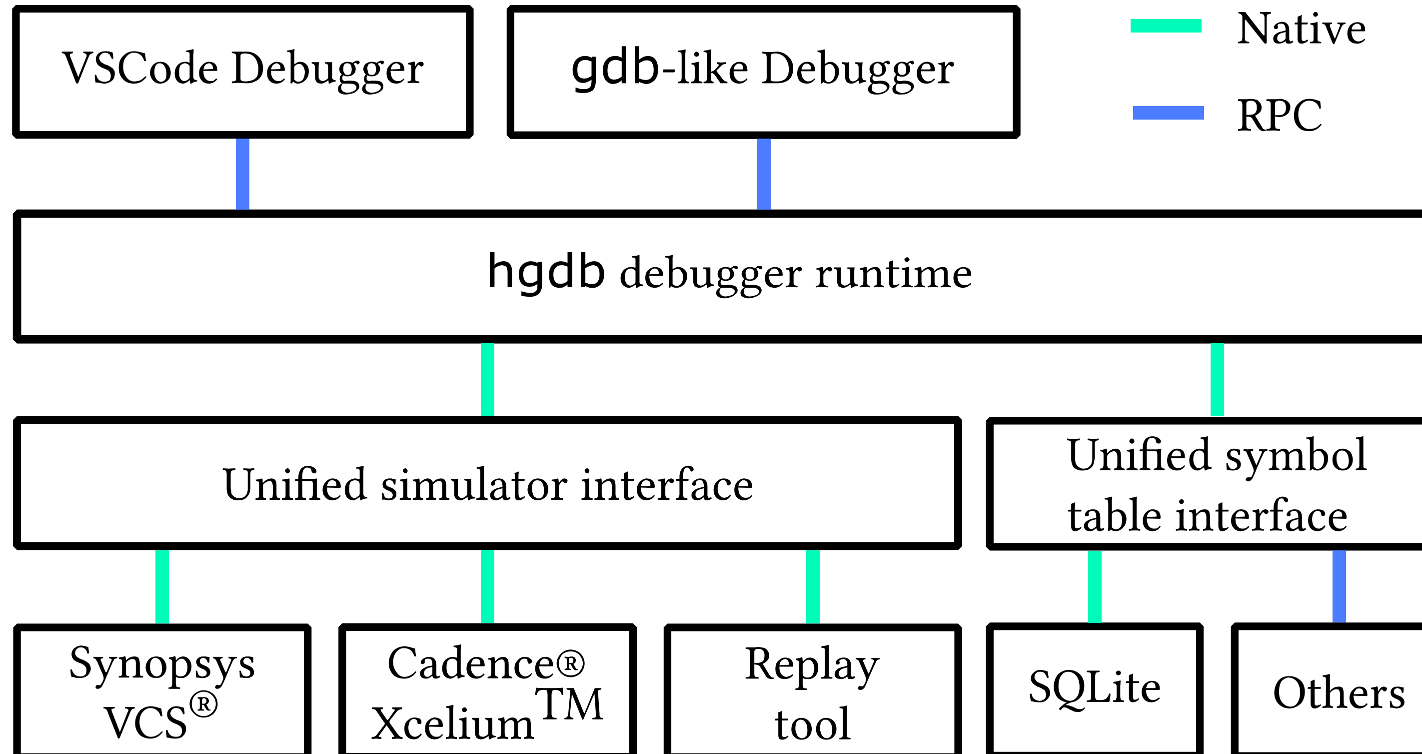
Generated RTL from the code shown before

59 DESIGN AUTOMATION CONFERENCE

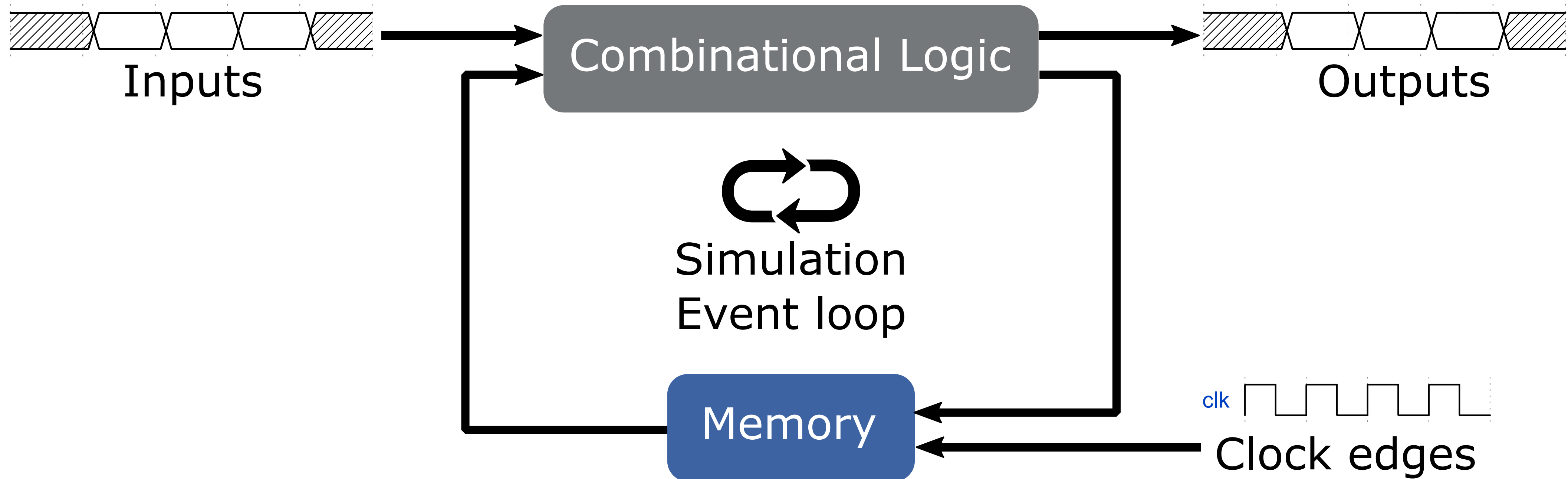
Introducing hgdb

- Source-level debugging
- Minimal performance overhead
- No RTL changes required
- Two complete debuggers
  - VSCode
  - gdb-inspired console debugger
- All major simulators
  - Big 3
  - Verilator
  - iverilog
- FSDB and VCD Replay
  - Reverse debugging!

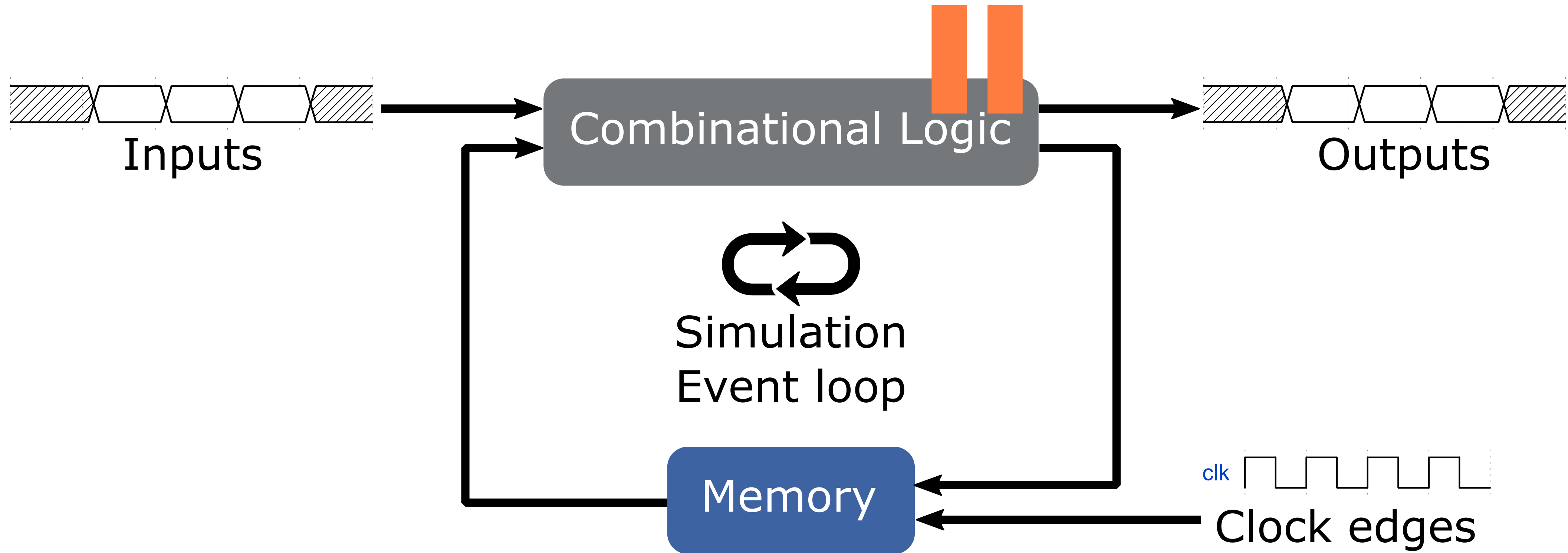
System design



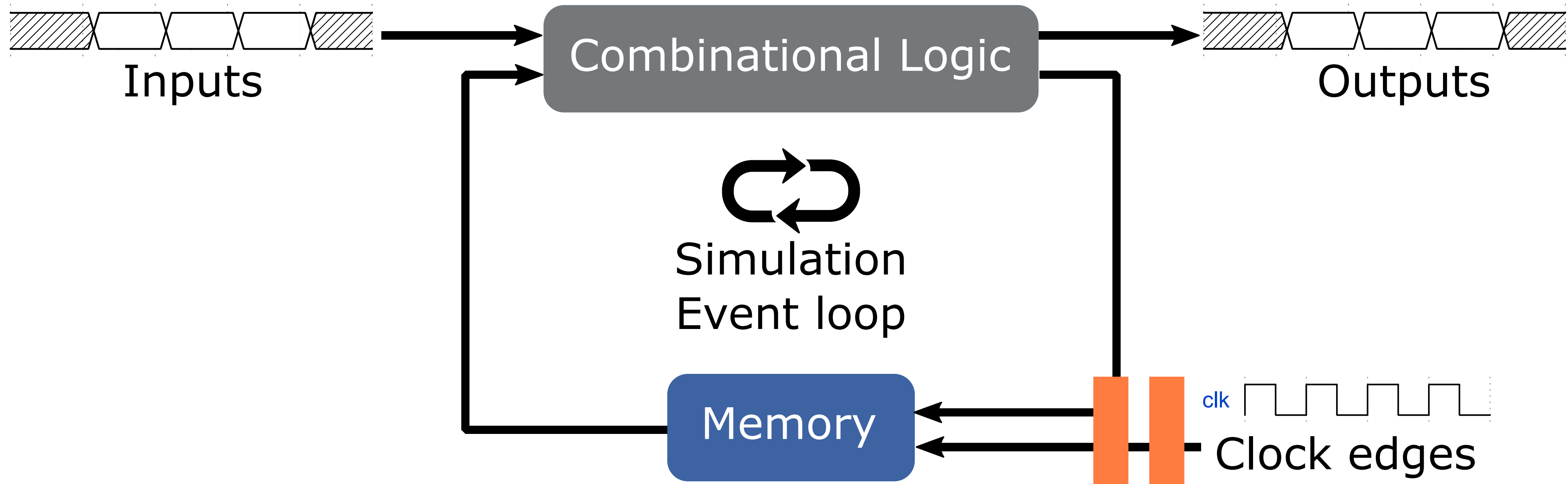
Low overhead breakpoint emulation



Low overhead breakpoint emulation



Low overhead breakpoint emulation



59 DESIGN AUTOMATION CONFERENCE

Emulate with correct semantics

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

Stack frame:

data	1	2
	3	4
sum	4	
i	4	

DESIGN AUTOMATION CONFERENCE 59

SSA and loop unrolling to rescue

- Insight:
  - Static single assignment (SSA) and loop unrolling are commonly used in compiler optimization.
  - Use these transformation artifacts to help debugging.

59 DESIGN AUTOMATION CONFERENCE

SSA and loop unrolling to rescue

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

Original Code

SSA and loop unrolling to rescue

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  if (data[0] % 2) sum += data[0];
  if (data[1] % 2) sum += data[1];
  if (data[2] % 2) sum += data[2];
  if (data[3] % 2) sum += data[3];
end
```

Code after loop unrolling

59 DESIGN AUTOMATION CONFERENCE

SSA and loop unrolling to rescue

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Code after Single-Static-Assignment transformation

59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation with SSA

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

One to many mapping due to loop unrolling

Breakpoint emulation with SSA

data	1	2
	3	4
sum ← sum0	0	

Stack frame

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Use SSA mapping to construct stack frame

59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation with SSA

data	1	2
	3	4
sum ← sum1	1	

Stack frame

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Use SSA mapping to construct stack frame

DESIGN AUTOMATION CONFERENCE 59

Breakpoint emulation with SSA

data	1	2
	3	4
sum sum2	1	

Stack frame

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Use SSA mapping to construct stack frame

59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation with SSA

data	1	2
	3	4
sum ← sum3	4	

Stack frame

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Use SSA mapping to construct stack frame

59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation with SSA

data	1	2
	3	4
sum ← sum0	0	
i	0	

Stack frame

```
logic [31:0] s
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Storing static values into symbol table when unrolling the loop

Breakpoint emulation with SSA

data	1	2
	3	4
sum ← sum1	1	
i	1	

Stack frame

```
logic [31:0] s
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Storing static values into symbol table when unrolling the loop

Breakpoint emulation with SSA

data	1	2
	3	4
sum ← sum2	1	
i	2	

Stack frame

```
logic [31:0] s
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Storing static values into symbol table when unrolling the loop

59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation with SSA

data	1	2
	3	4
sum ← sum3	4	
i	3	

Stack frame

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Storing static values into symbol table when unrolling the loop

59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation with SSA

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

data[0] % 2

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Using SSA transformation to compute “enable condition”

59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation with SSA

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

data[1] % 2

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Using SSA transformation to compute “enable condition”

59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation with SSA

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

data[2] % 2

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Using SSA transformation to compute “enable condition”

59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation with SSA

```
logic [31:0] sum;
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

data[3] % 2

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Using SSA transformation to compute “enable condition”

59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation with SSA

data	1	2
	3	4
sum ← sum0	0	
i	0	

Stack frame

```
logic [31:0] s
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Put everything together: only two breakpoints are enabled

59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation with SSA

data	1	2
	3	4
sum ← sum2	1	
i	2	

Stack frame

```
logic [31:0] s
logic [31:0] data[4];

always_comb begin
  sum = 0;
  for (int i = 0; i < 4; i++) begin
    if (data[i] % 2) begin
      sum += data[i];
    end
  end
end
```

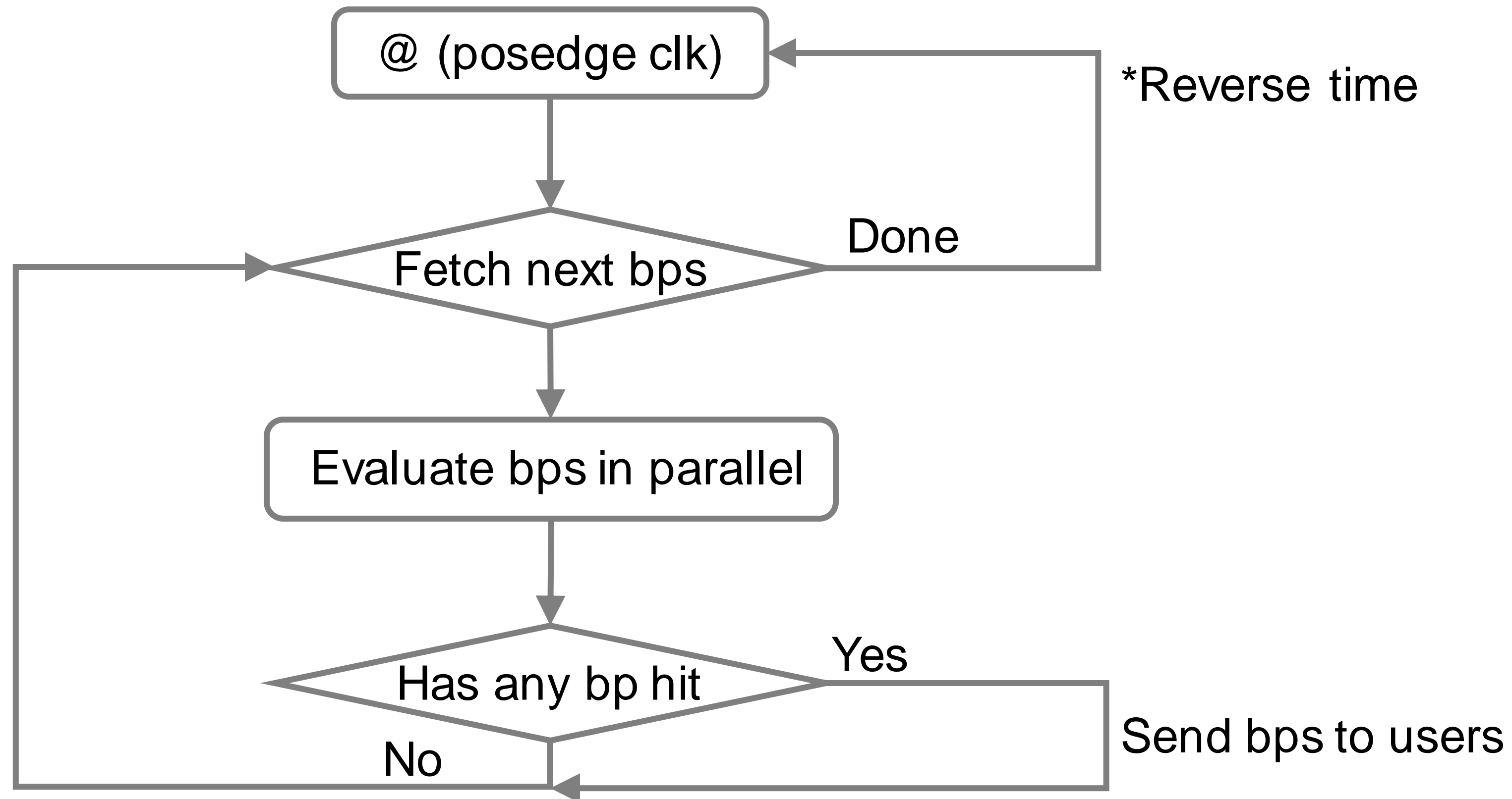
```
logic [31:0] sum, sum0, sum1, sum2, sum3, sum4;
logic [31:0] data[4];

assign sum0 = 0;
assign sum1 = data[0] % 2? data[0]: sum0;
assign sum2 = data[1] % 2? sum1 + data[1]: sum1;
assign sum3 = data[2] % 2? sum2 + data[2]: sum2;
assign sum4 = data[3] % 2? sum3 + data[3]: sum3;
assign sum =  sum4;
```

Put everything together: only two breakpoints are enabled

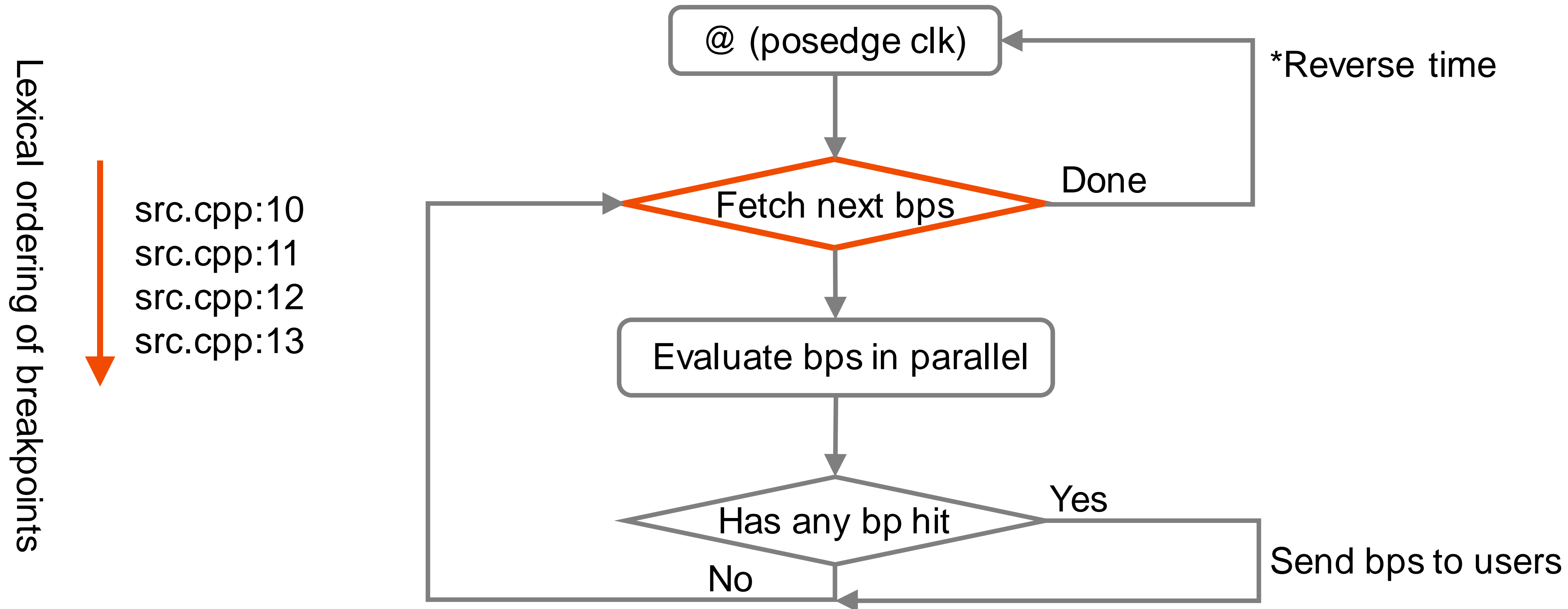
59 DESIGN AUTOMATION CONFERENCE

Breakpoint emulation loop



59 DESIGN AUTOMATION CONFERENCE

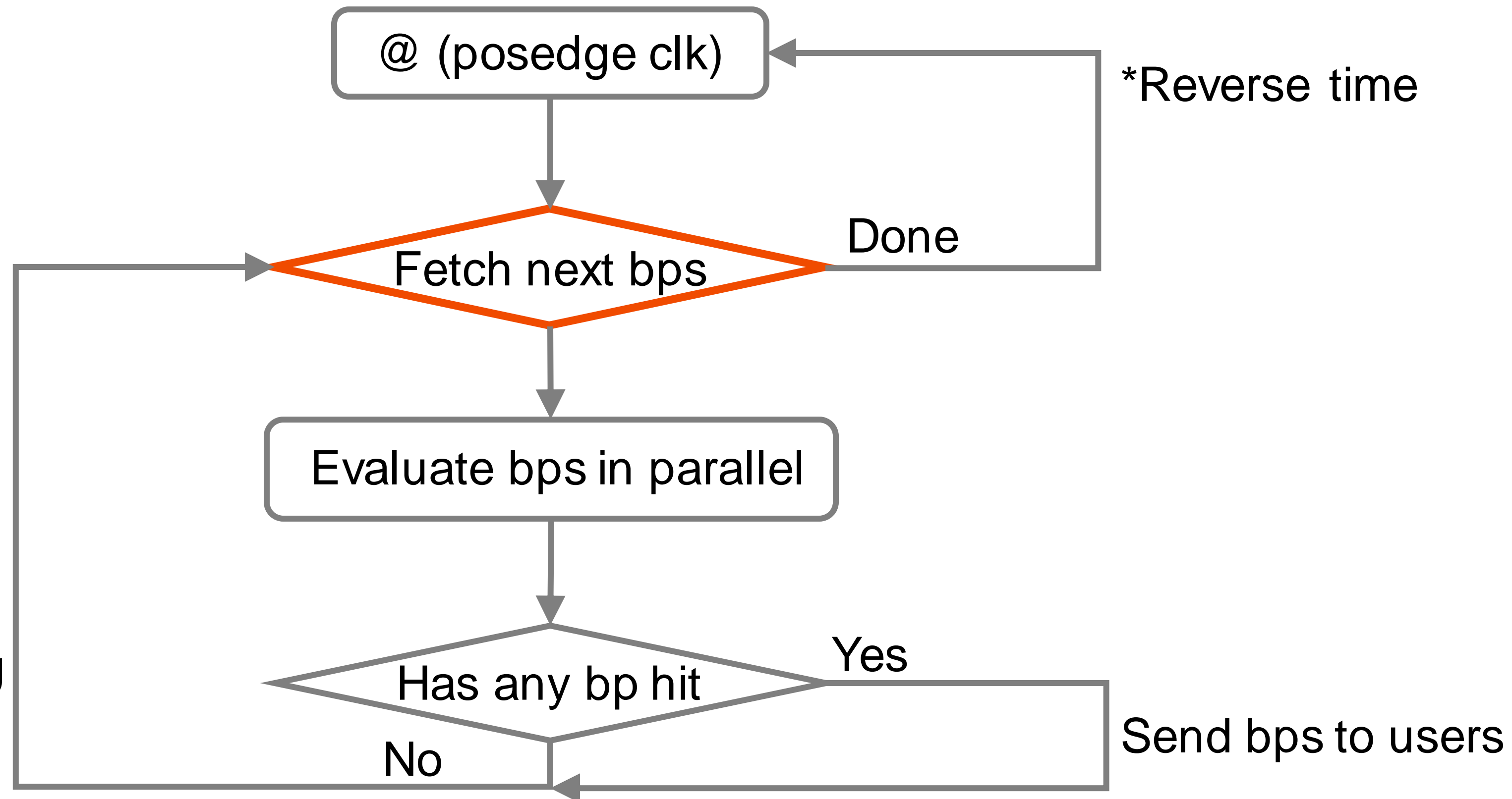
Breakpoint emulation loop



Breakpoint emulation loop

src.cpp:10
src.cpp:11
src.cpp:12
src.cpp:13

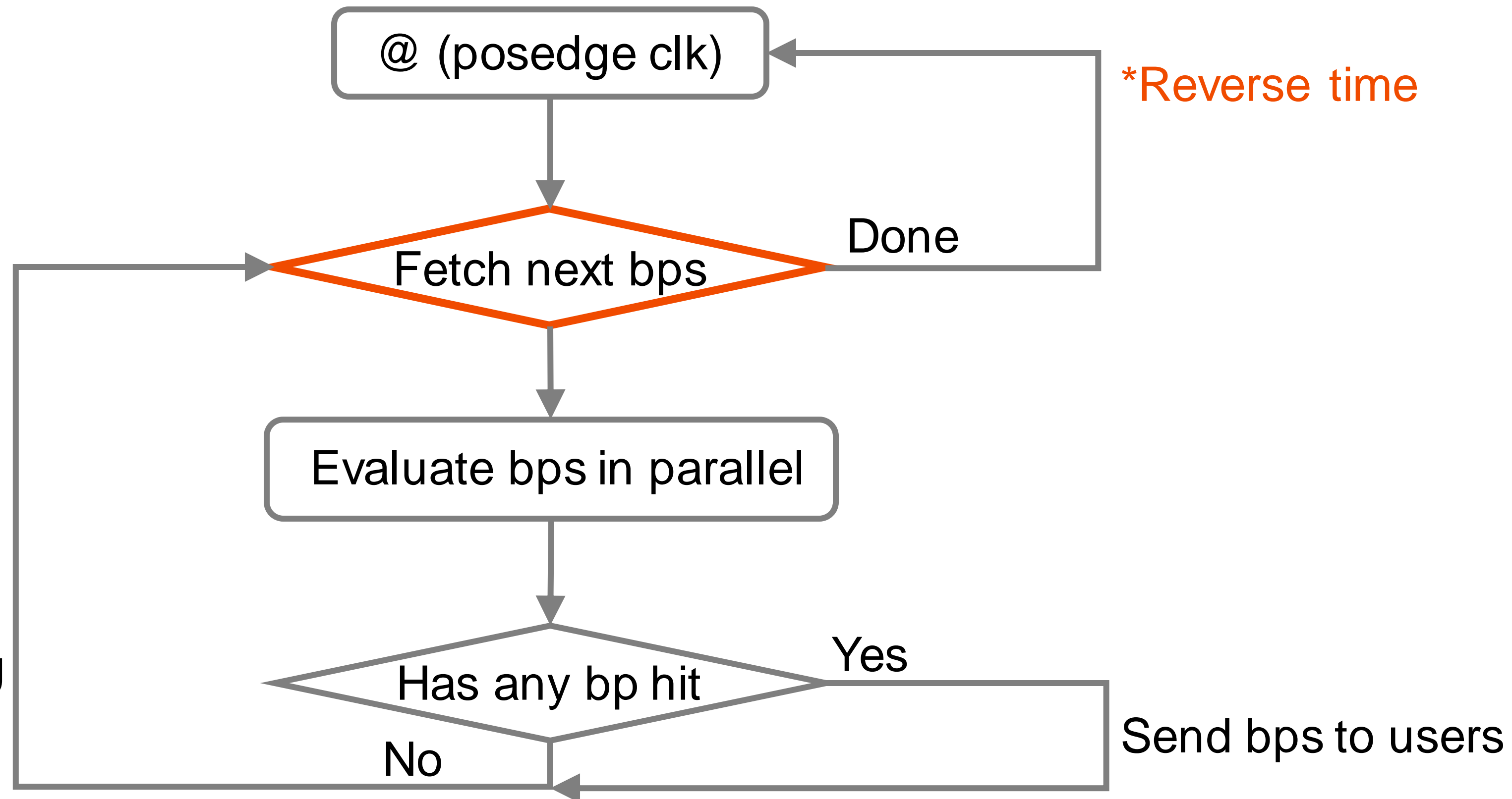
Intra-cycle reverse debugging



Breakpoint emulation loop

src.cpp:10
src.cpp:11
src.cpp:12
src.cpp:13

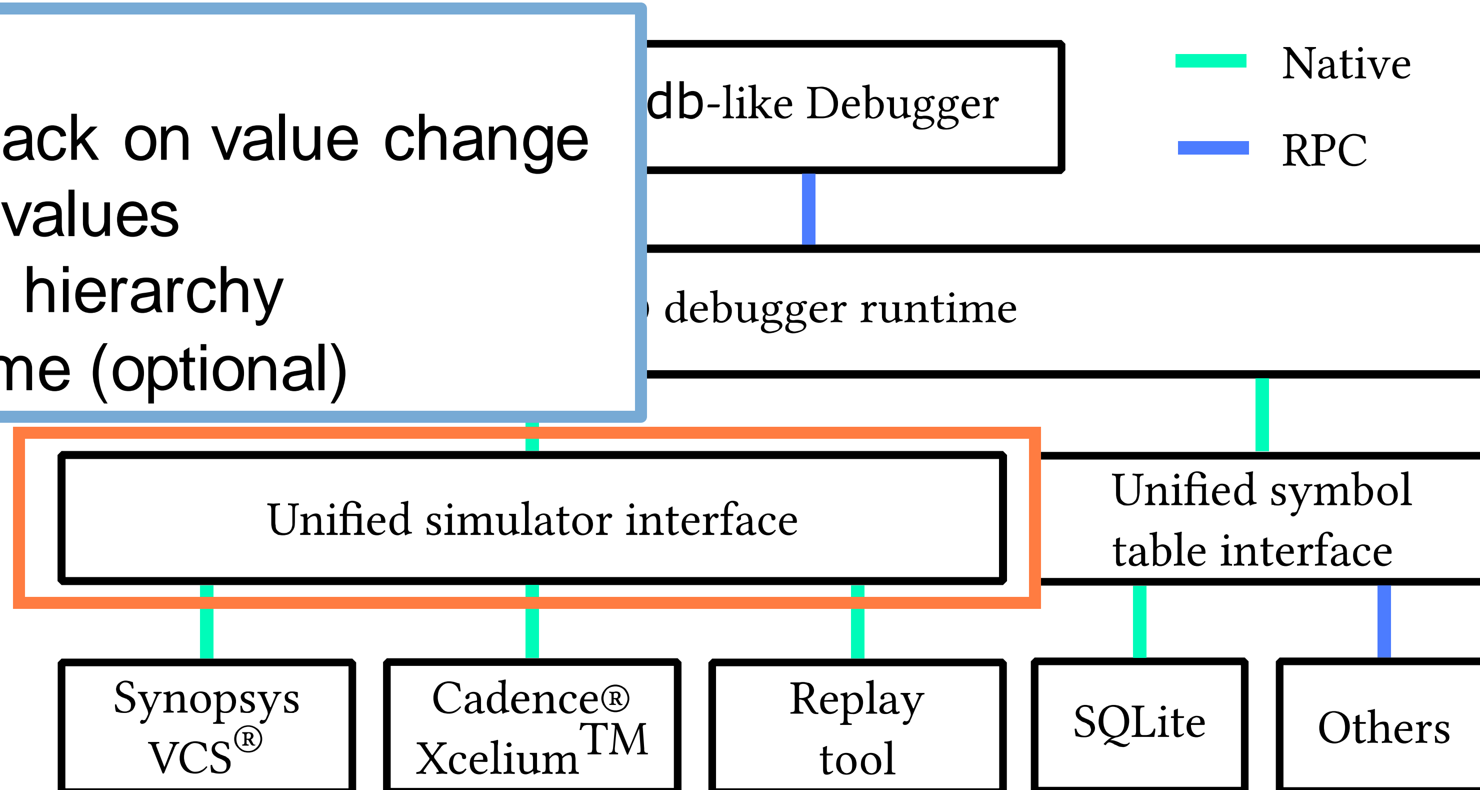
Inter-cycle reverse debugging



Unified simulator interface

Primitives:

- Place callback on value change
- Get signal values
- Get design hierarchy
- Reverse time (optional)

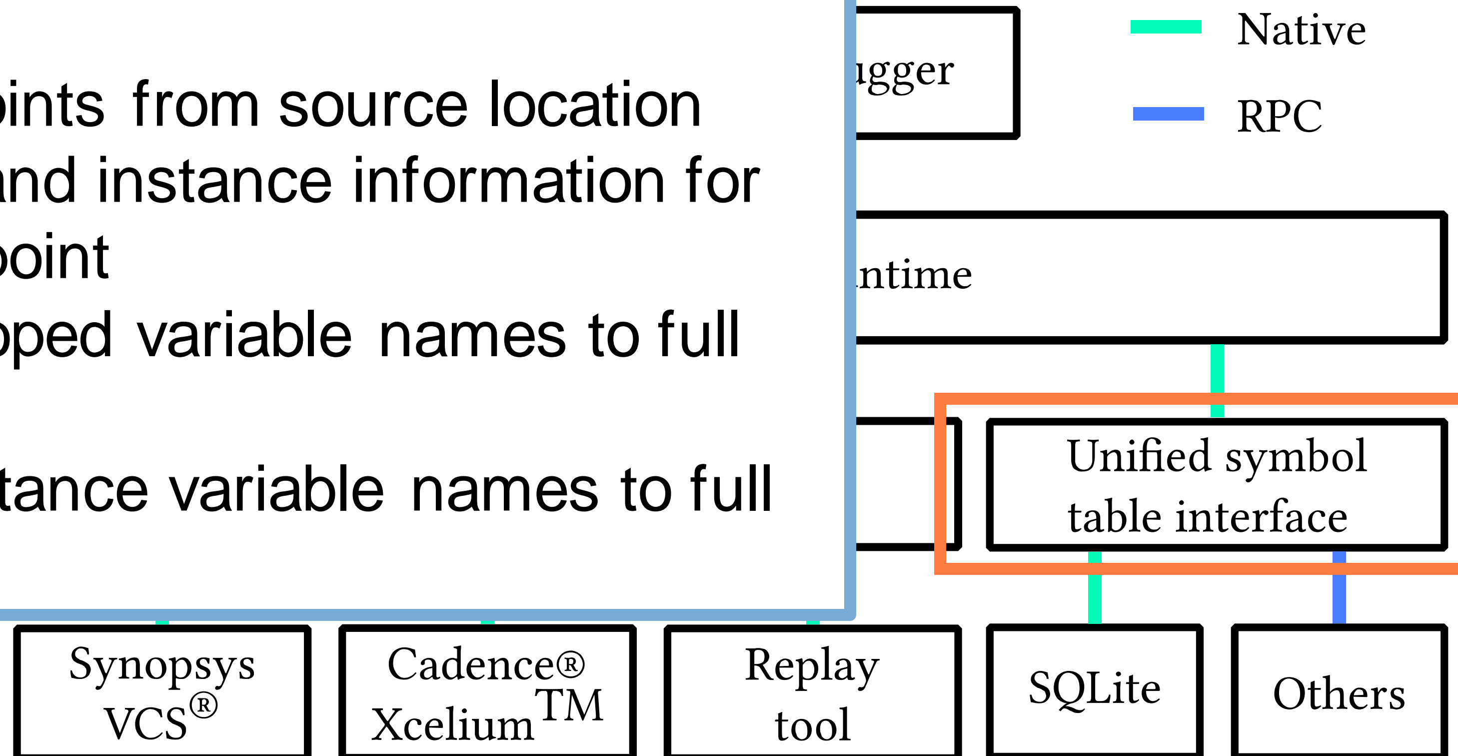


59 DESIGN AUTOMATION CONFERENCE

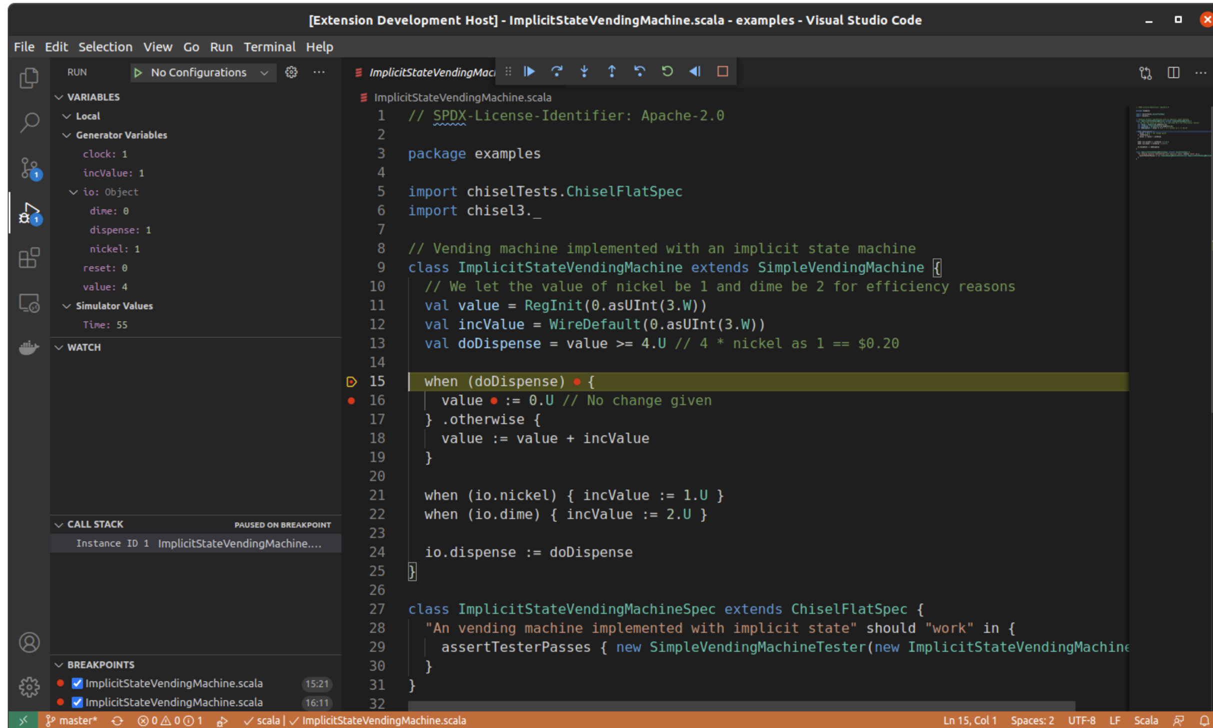
Unified simulator interface

Primitives:

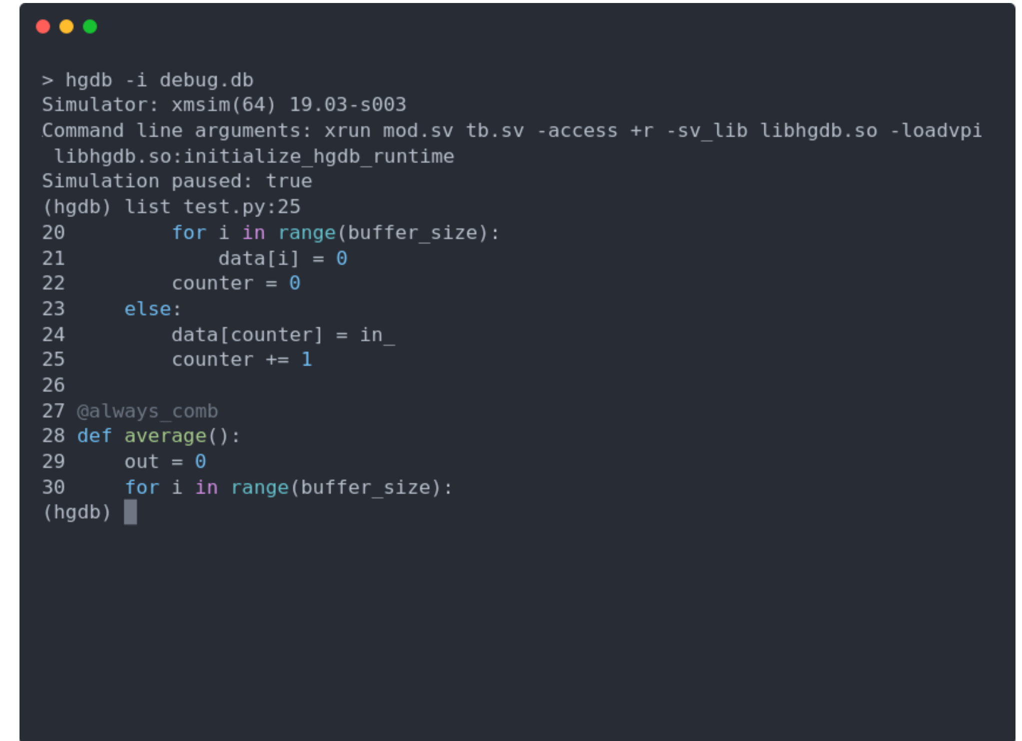
- Get breakpoints from source location
- Get scope and instance information for each breakpoint
- Resolve scoped variable names to full name
- Resolve instance variable names to full name



Hgdb debuggers



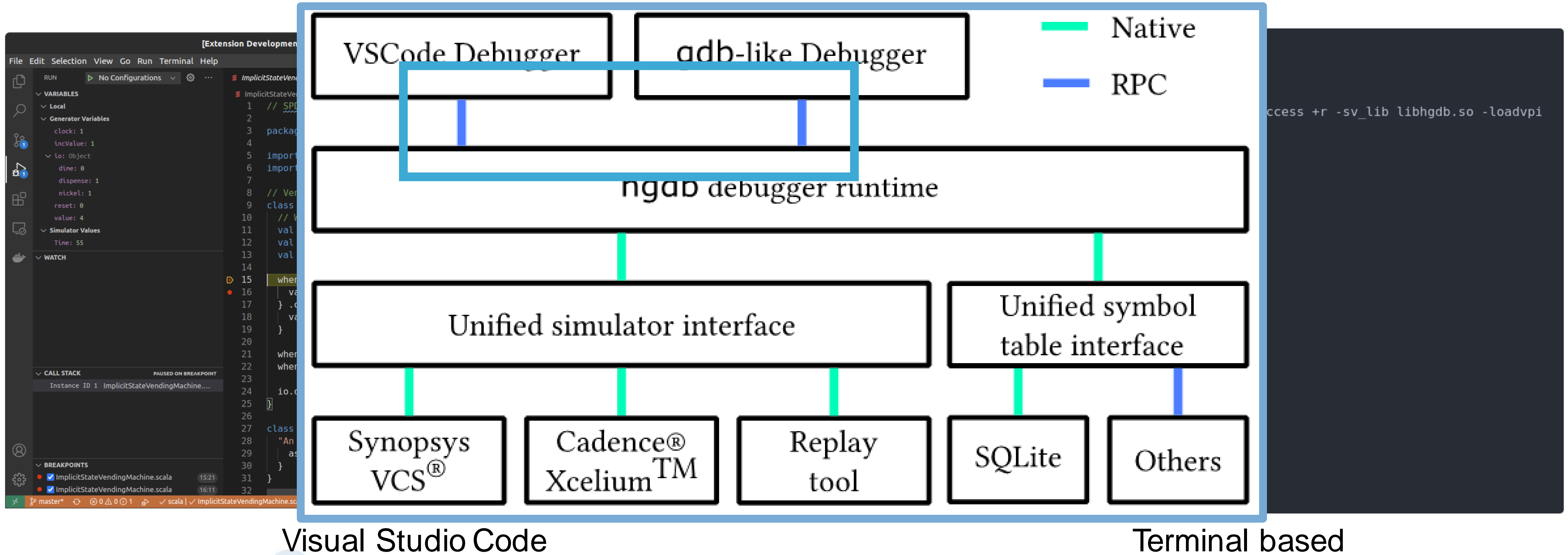
Visual Studio Code



Terminal based

59 DESIGN AUTOMATION CONFERENCE

Hgdb debuggers



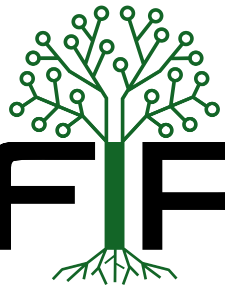
Visual Studio Code

Terminal based

59 DESIGN AUTOMATION CONFERENCE

Integration with hardware generators

CHISEL



FIRRTL

Chisel/Firrtl

CIRCT

MLIR/CIRCT

XILINX VITIS™

Vitis HLS

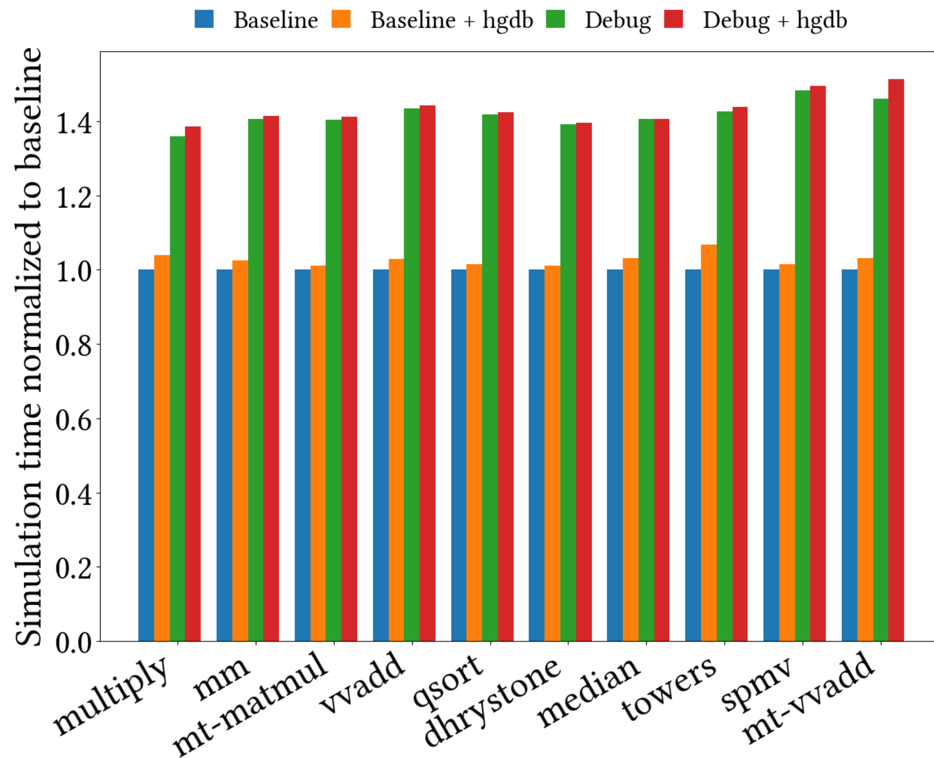
Working with Firrtl compiler

```
Input: CircuitState
Output: Table
Annotations ← {};
foreach node ∈ CircuitState do // First pass
    if node is statement then
        node.enable ← ComputeEnableCondition(node);
    end
    Annotation ← Annotations ∪ {node}
end
// FIRRTL transformations;
IRNodes ← {};
foreach node ∈ Annotations do
    if node ∈ CircuitState then
        IRNodes ← IRNodes ∪ node;
    end
end
Table ← ComputeSymbolTable(IRNodes);
```

- First pass:
  - Insert annotation and compute enable condition
  - (Debug mode) insert DontTouchAnnotation
- Second pass:
  - Collect annotations and only compute symbol table if the IRNode still exists

59 DESIGN AUTOMATION CONFERENCE

Performance evaluation



- Rocketchip built-in benchmark
- Debug mode refers to passes disable compiler optimization
- 5% performance overhead

59 DESIGN AUTOMATION CONFERENCE

Conclusion

- Hardware generators are new, and debugging infrastructure is missing
- Hgdb connects hardware generator frameworks and existing simulators
  - Works with all major simulator vendors
  - Brings source level debugging
- Hgdb is an open-source framework from Stanford AHA! center
  - Contributions are welcome!

https://github.com/Kuree/hgdb