Introducing **h**ardware **g**enerator **d**ebu**g**ger

and how we can debug at source-level

Keyi Zhang
Computer Science Department
Stanford University

What are hardware generators?

First generation:

- “Preprocessor”-based string templating
- Using Perl or Python to preprocess the SystemVerilog to overcome the language limit.

What are hardware generators?

First gene

- “Prep
- Using he
  langu

```
module `mname` (
clk, clk_en, reset,
//; for(my $i=0; $i<$pe_output_count; $i++) {
pe_output_`$i`,
//; }
//; for(my $i=0; $i<$sides; $i++) {
//;  for(my $j=0; $j<$number_of_outputs; $j++) {
out_`$i`_`$j`,
in_`$i`_`$j`,
//;  }
//; }
```

What are hardware generators?

First generation:

- “Preprocessor”-based string templating
- Using Perl or Python to preprocess the SystemVerilog to overcome the language limit.

Second generation”

- Embedded domain specific languages that use host languages features to improve design productivity
- Chisel (scala), Manga (Python), PyMTL (Python)
- “Mini” compiler tool chains with IR

What are hardware generators?

First gene

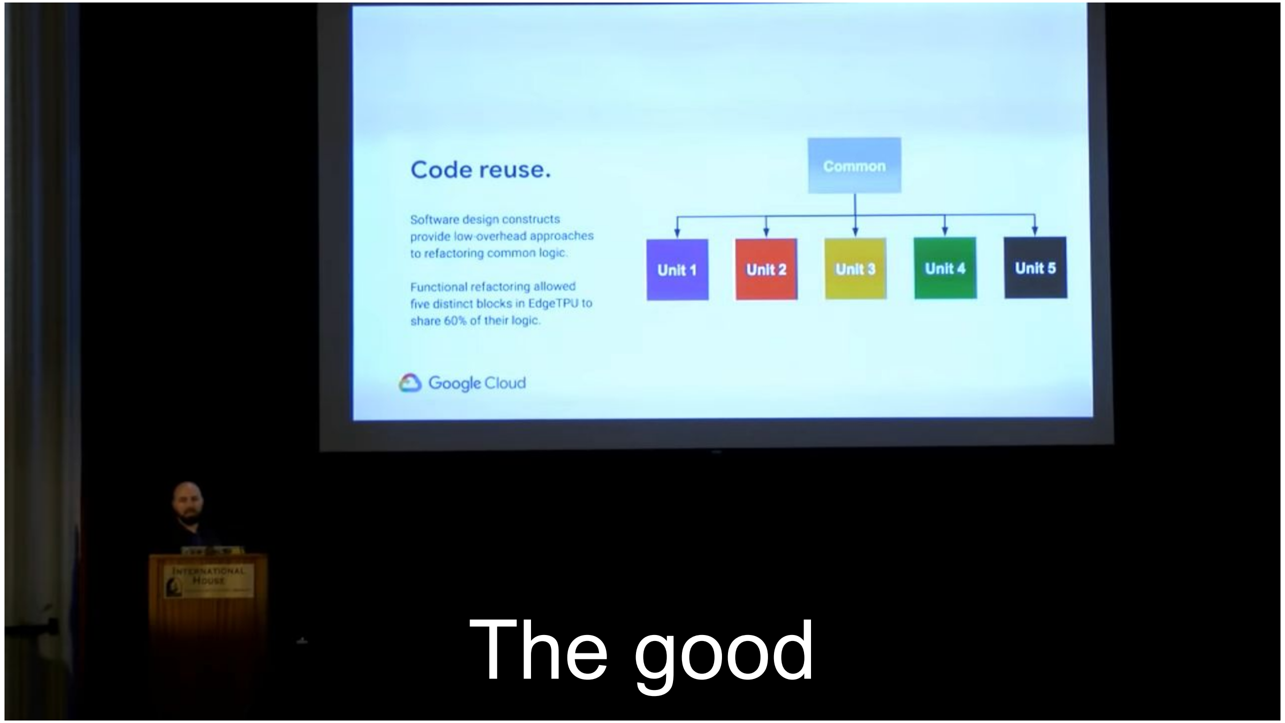
- “Prep
- Using … he
  langu

Second g

- Embe … res to
  impro
- Chise
- “Mini” compiler tool chains with IR

```
class DoubleBufferFifo[T <: Data](gen: T, depth: Int) extends Fifo(gen: T,
depth: Int) {

  private class DoubleBuffer[T <: Data](gen: T) extends Module {
    val io = IO(new FifoIO(gen))

    val empty :: one :: two :: Nil = Enum(3)
    val stateReg = RegInit(empty)
    val dataReg = Reg(gen)
    val shadowReg = Reg(gen)
...
```

Google tried Chisel 3 years ago

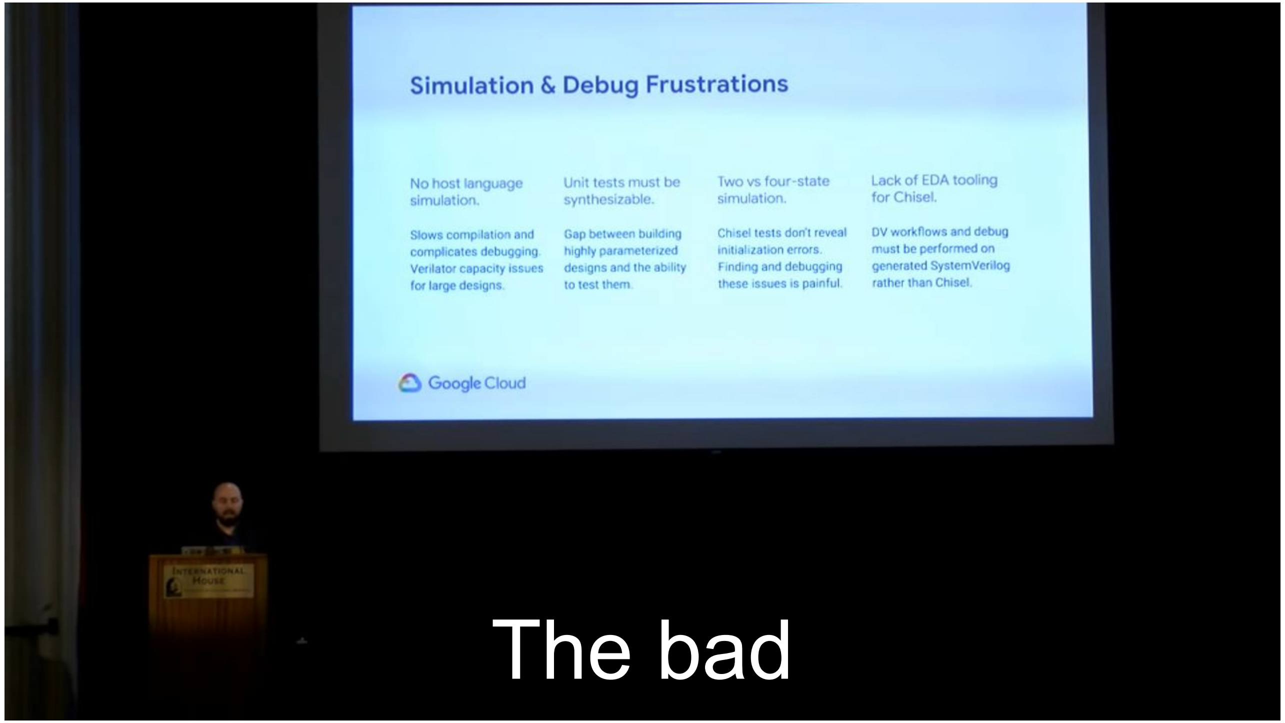


Code reuse.

Software design constructs provide low-overhead approaches to refactoring common logic.

Functional refactoring allowed five distinct blocks in EdgeTPU to share 60% of their logic.

Google tried Chisel 3 years ago



Simulation & Debug Frustrations

No host language simulation.

Slows compilation and complicates debugging. Verilator capacity issues for large designs.

Unit tests must be synthesizable.

Gap between building highly parameterized designs and the ability to test them.

Two vs four-state simulation.

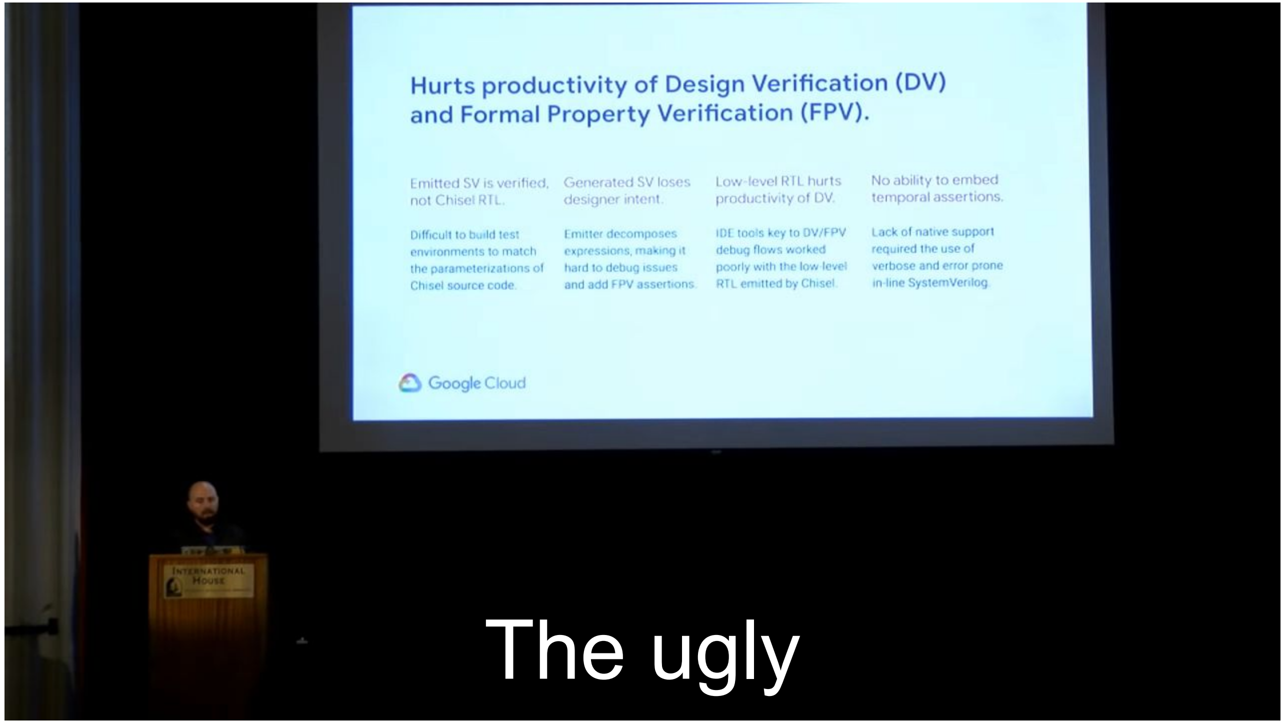
Chisel tests don't reveal initialization errors. Finding and debugging these issues is painful.

Lack of EDA tooling for Chisel.

DV workflows and debug must be performed on generated SystemVerilog rather than Chisel.

The bad

Google tried Chisel 3 years ago



Hurts productivity of Design Verification (DV) and Formal Property Verification (FPV).

Emitted SV is verified, not Chisel RTL.	Generated SV loses designer intent.	Low-level RTL hurts productivity of DV.	No ability to embed temporal assertions.
Difficult to build test environments to match the parameterizations of Chisel source code.	Emitter decomposes expressions, making it hard to debug issues and add FPV assertions.	IDE tools key to DV/FPV debug flows worked poorly with the low level RTL emitted by Chisel.	Lack of native support required the use of verbose and error prone in-line SystemVerilog.

Google Cloud

The ugly

Let’s address the bad part: simulation and debugging

Latest RocketChip + Chisel3

```
// 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))
```

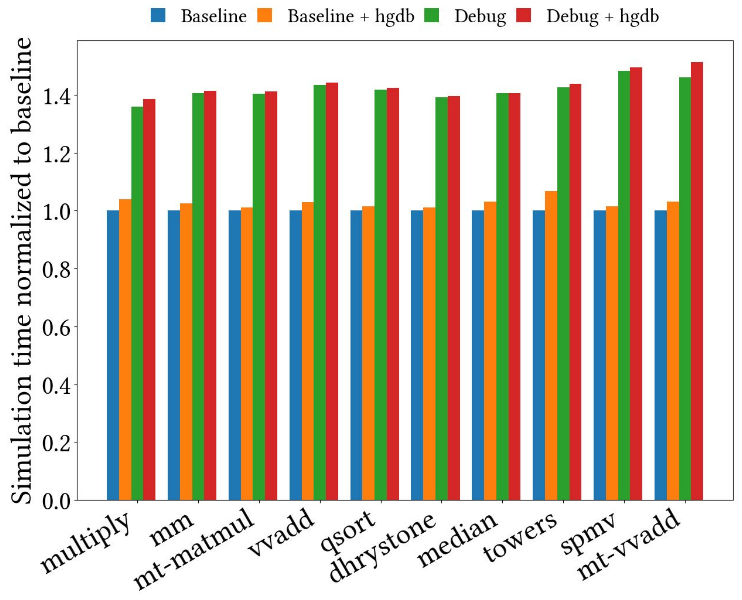
Let’s address the bad part: simulation and debugging

Latest RocketChip + Chisel3

```
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]
wire [63:0] _GEN_2 = {{16'd0}, _shin_T_11[63:16]}; // @[Bitwise.scala 103:31]
wire [63:0] _shin_T_16 = _GEN_2 & 64'hffff0000ffff; // @[Bitwise.scala 103:31]
wire [63:0] _shin_T_18 = {_shin_T_11[47:0], 16'h0}; // @[Bitwise.scala 103:65]
wire [63:0] _shin_T_20 = _shin_T_18 & 64'hffff0000ffff0000; // @[Bitwise.scala 103:75]
wire [63:0] _shin_T_21 = _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]
```

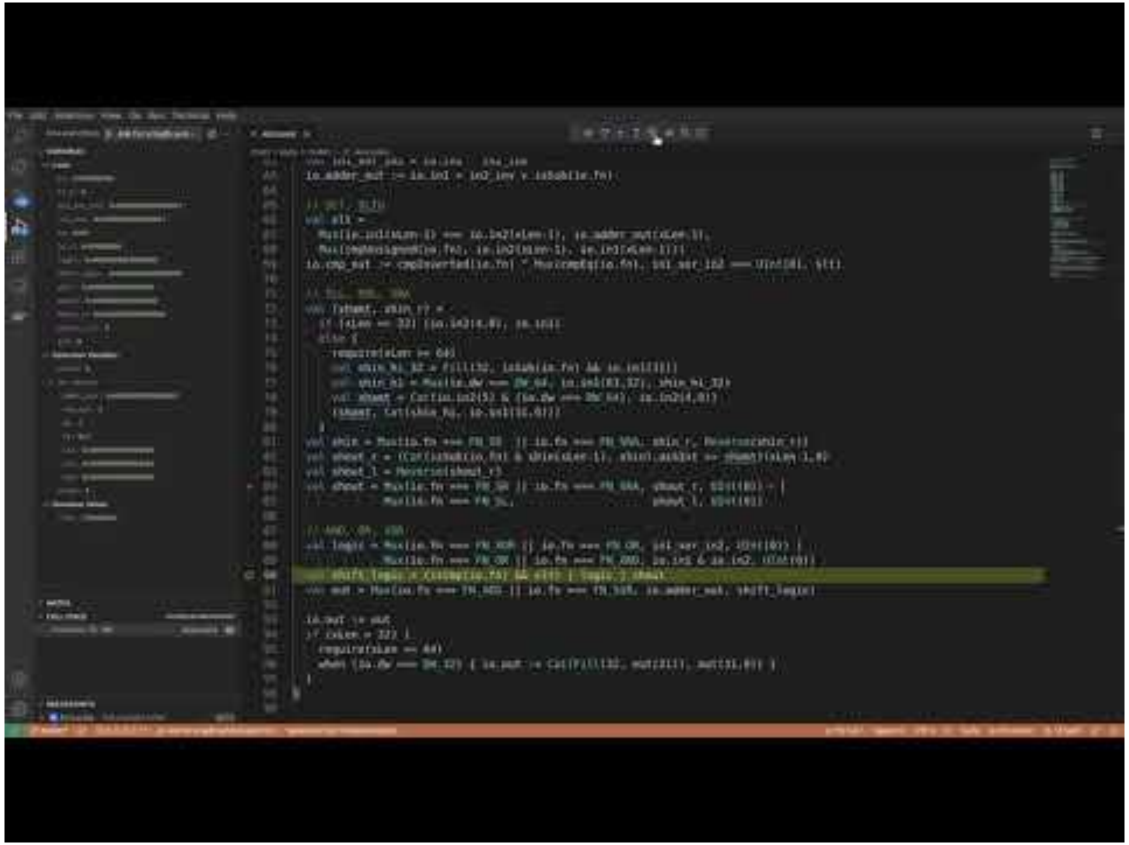
hgdb: batteries included

- Source-level debugging
- Minimal performance overhead
- No RTL changes required [1]
- Two complete debuggers
  - VSCode
  - gdb-inspired console debugger
- All major simulators
  - Big 3
  - Verilator
  - iverilog
- FSDB and VCD Replay
  - Reverse debugging!



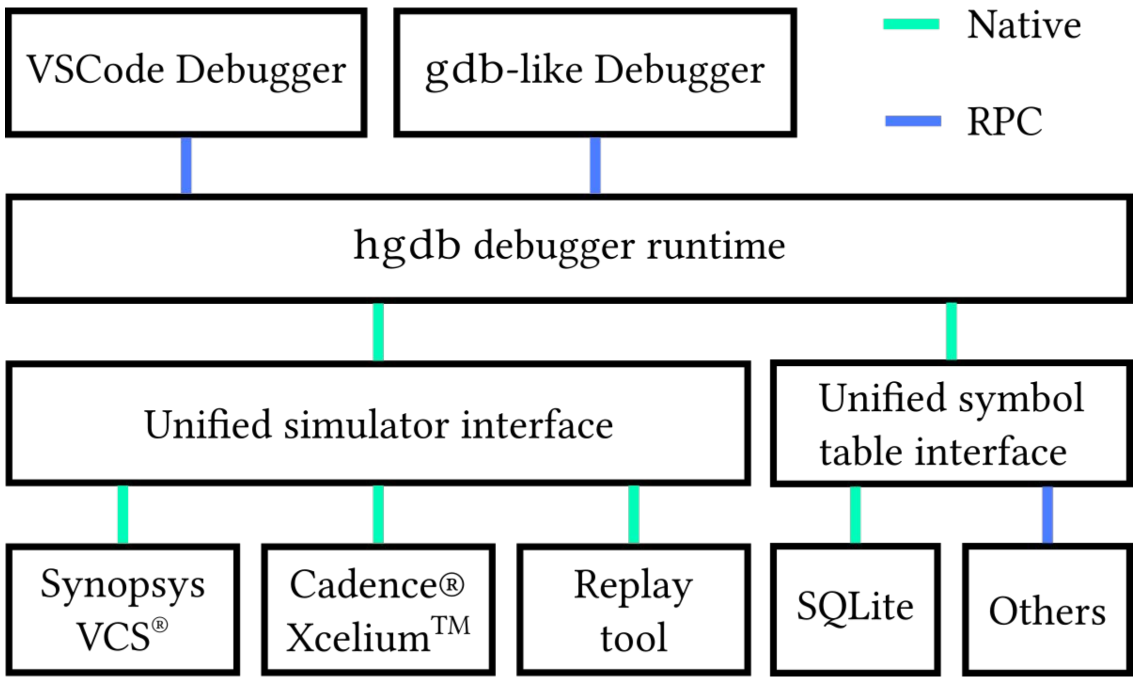
RocketChip Benchmark

What hgdb brings to the table

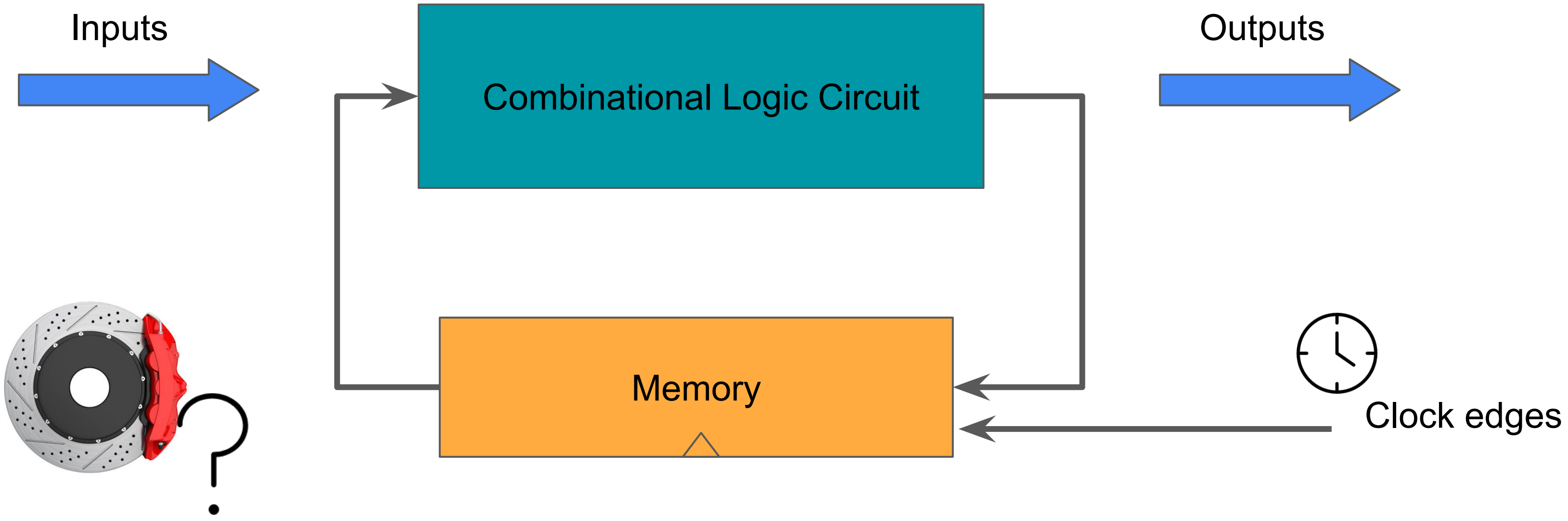


```
[illegible]
io.adder_out := io.in1 + in2_inv + isSub(io.fn)

// 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))
val shout_r = (Cat(isSub(io.fn) & shin(xLen-1), shin).asSInt >> shamt)(xLen-1,0)
val shout_l = Reverse(shout_r)
val shout = Mux(io.fn === FN_SR || io.fn === FN_SRA, shout_r, UInt(0)) |
            Mux(io.fn === FN_SL,                     shout_l, UInt(0))

// AND, OR, XOR
val logic = Mux(io.fn === FN_XOR || io.fn === FN_OR, in1_xor_in2, UInt(0)) |
            Mux(io.fn === FN_OR || io.fn === FN_AND, io.in1 & io.in2, UInt(0))
val shift_logic = (isCmp(io.fn) && slt) | logic | shout
val out = Mux(io.fn === FN_ADD || io.fn === FN_SUB, io.adder_out, shift_logic)

io.out := out
if (xLen > 32) {
  require(xLen == 64)
  when (io.dw === DW_32) { io.out := Cat(Fill(32, out(31)), out(31,0)) }
}
}
```

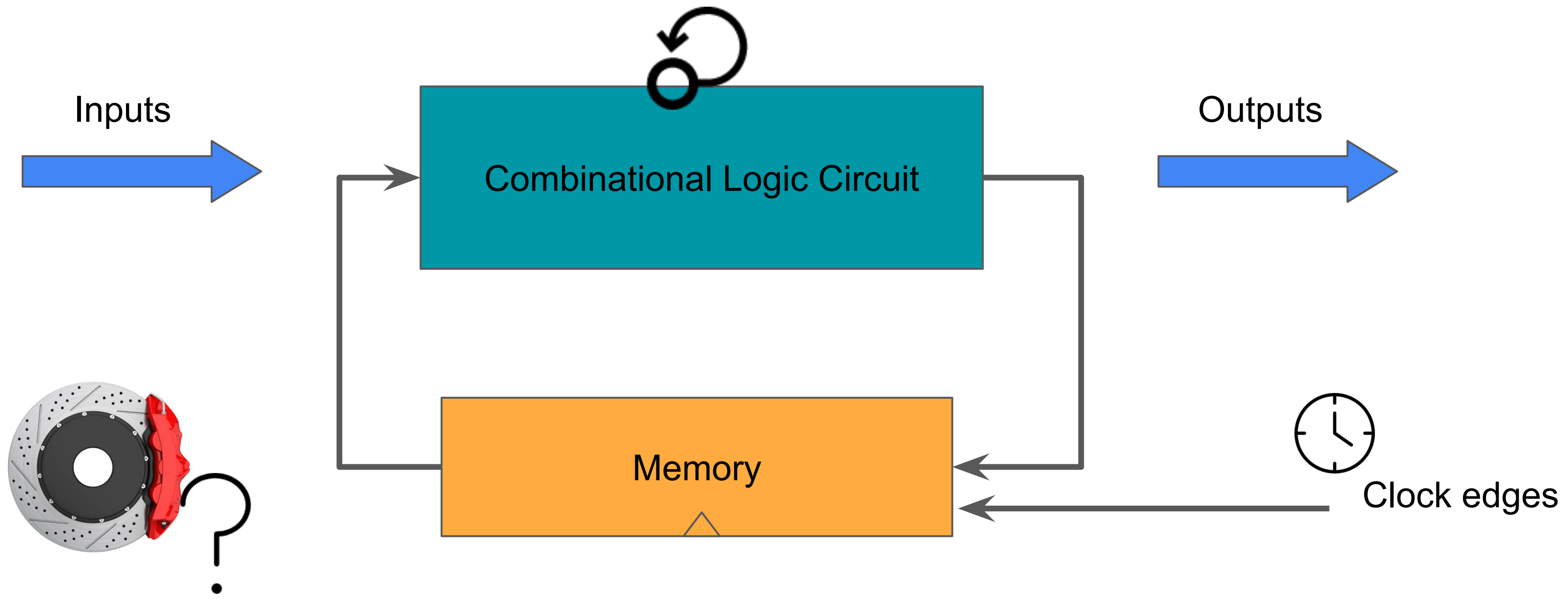
How hgdb works



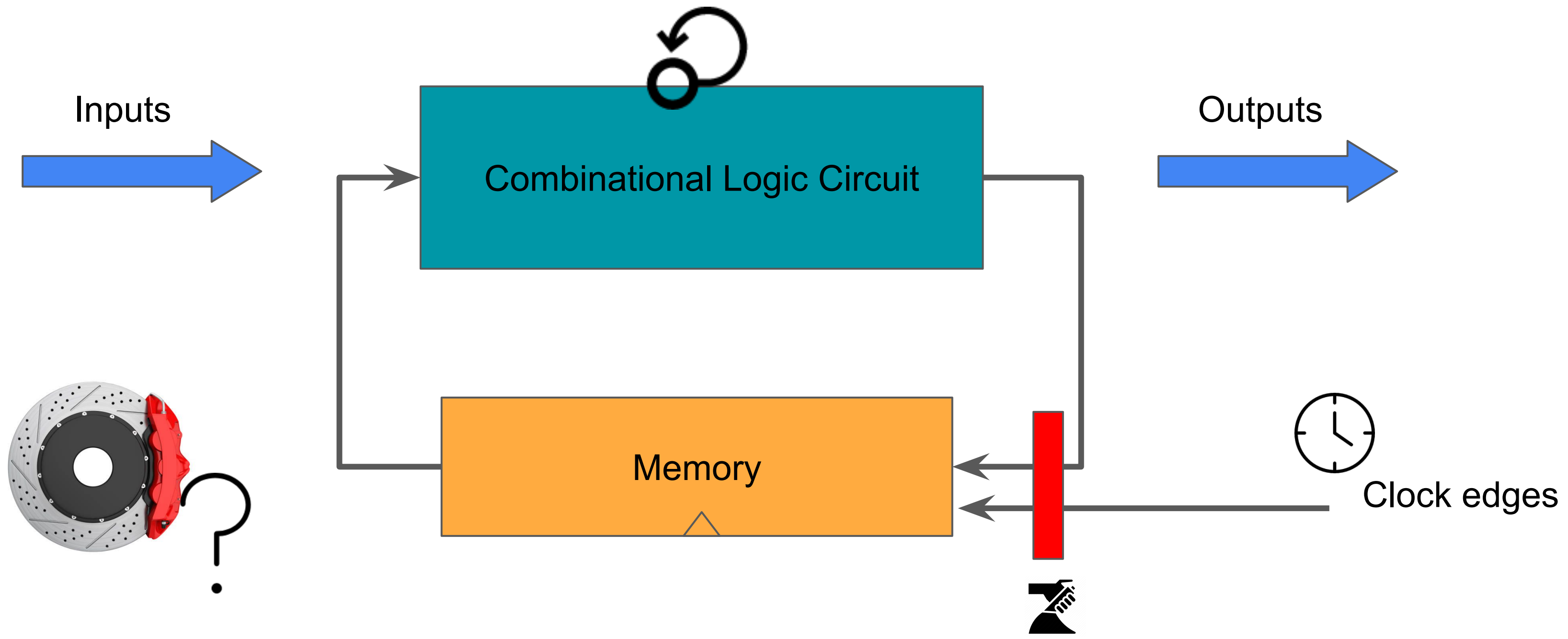
Keeps the overhead low: breakpoint emulation



Keeps the overhead low: breakpoint emulation



Keeps the overhead low: breakpoint emulation



But does it work all the time?

No. Unfortunately. 🙁

Combinational logic

```
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
```

Partial results lost when the state stablizles

Loop unrolling and SSA to rescue

Static single assignment

```
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
```

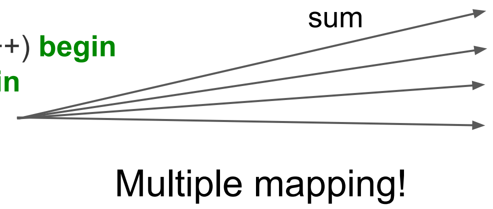
Loop unroll

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

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

SSA Transform

Loop unrolling and SSA 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
```



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

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

Original form

Final form

Loop unrolling and SSA 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
```

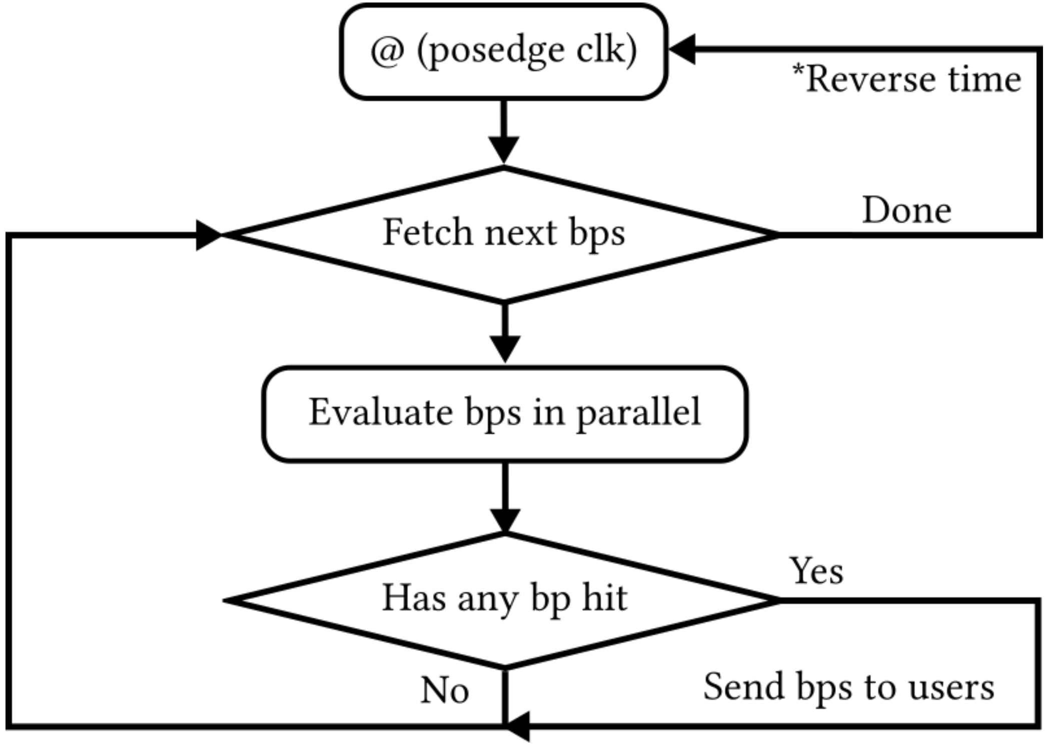
Enable condition

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

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

Original form

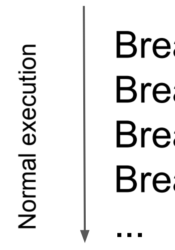
Final form

Breakpoint emulation loop

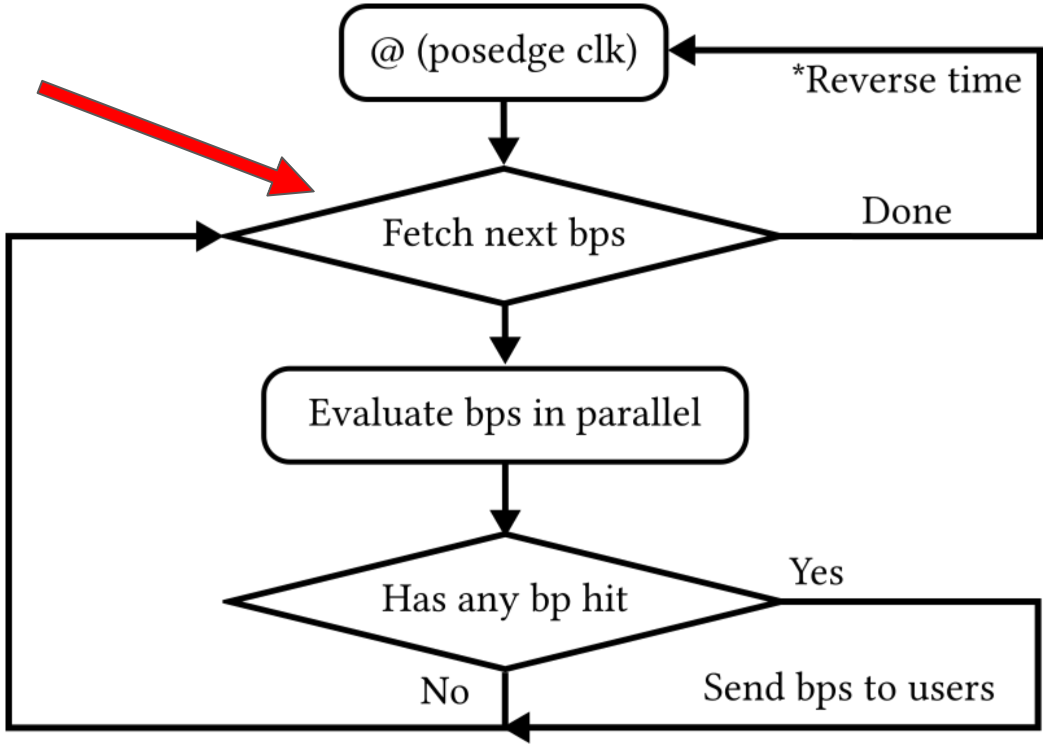


Breakpoint emulation loop

Fetch ordering!

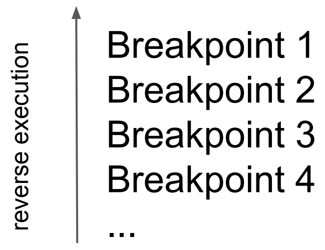


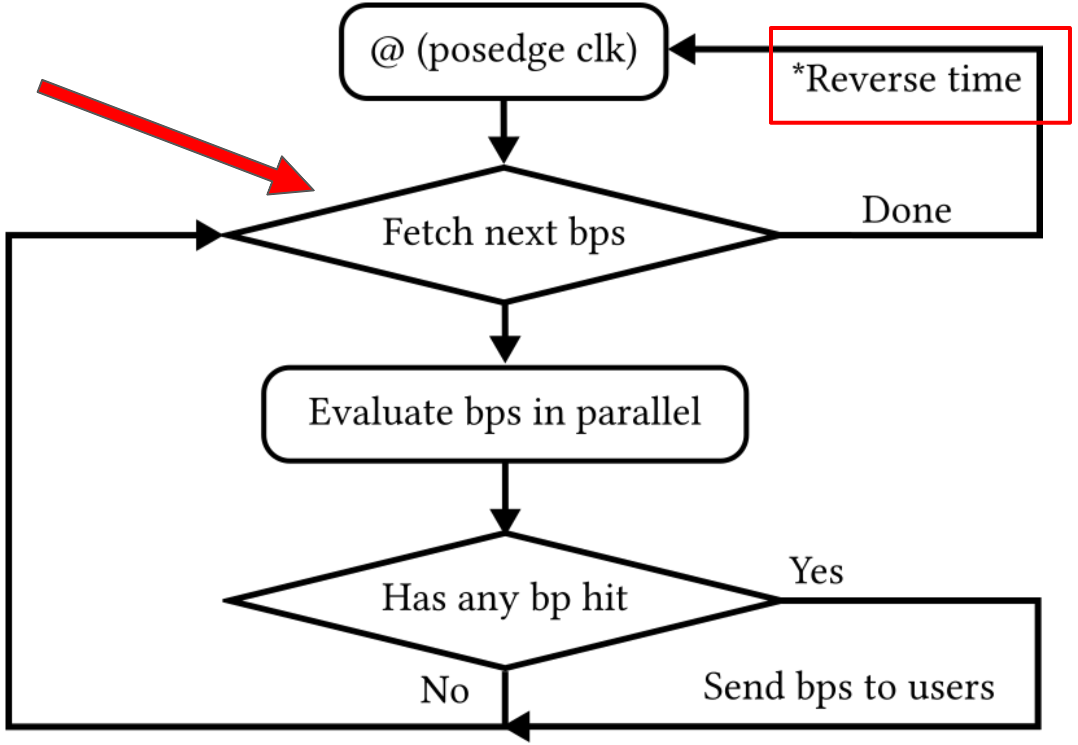
Breakpoint 1
Breakpoint 2
Breakpoint 3
Breakpoint 4
...



Breakpoint emulation loop

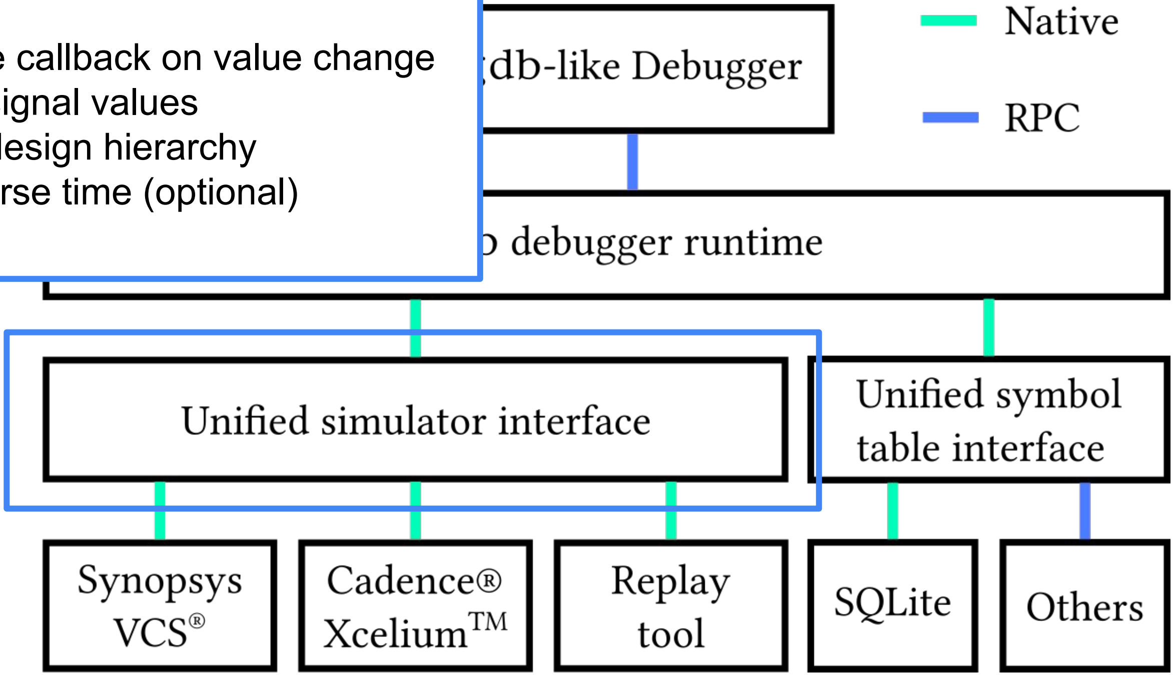
Fetch ordering!





Unified simulator interface

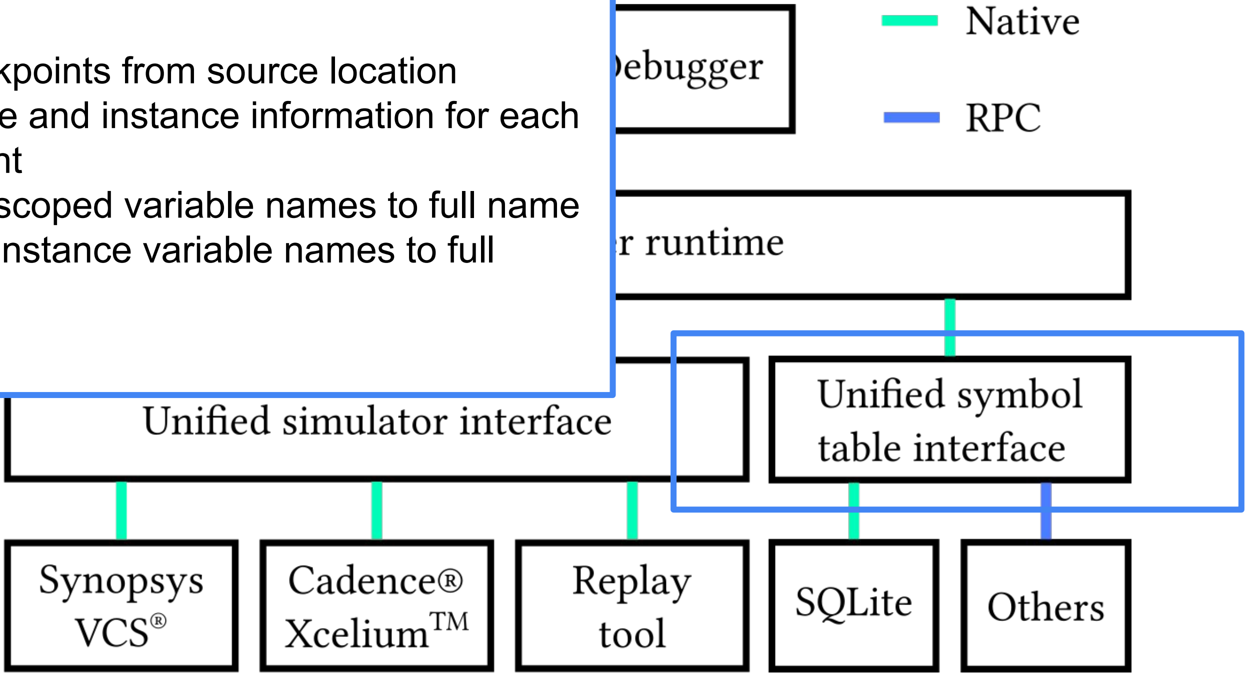
Primitives:
- Place callback on value change
- Get signal values
- Get design hierarchy
- Reverse time (optional)



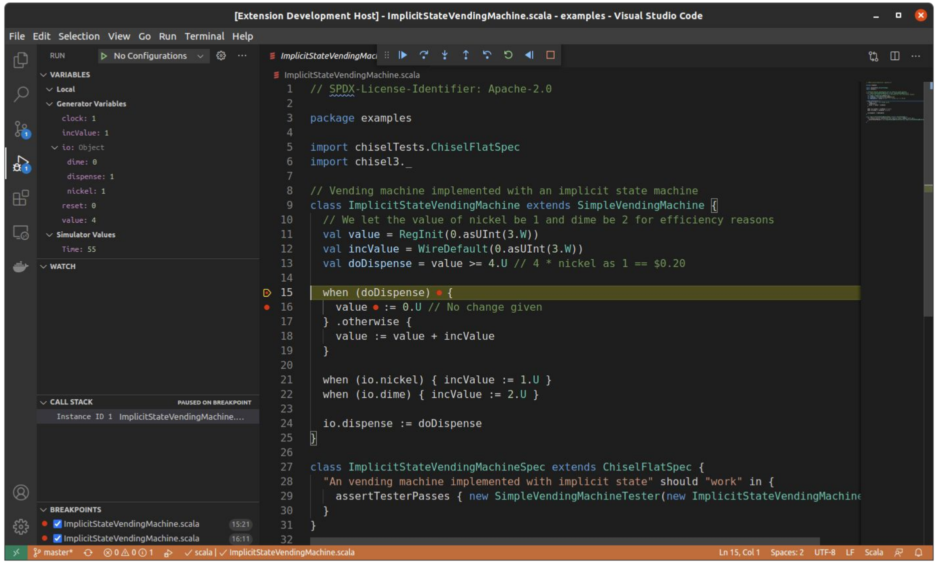
Unified symbol table interface

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



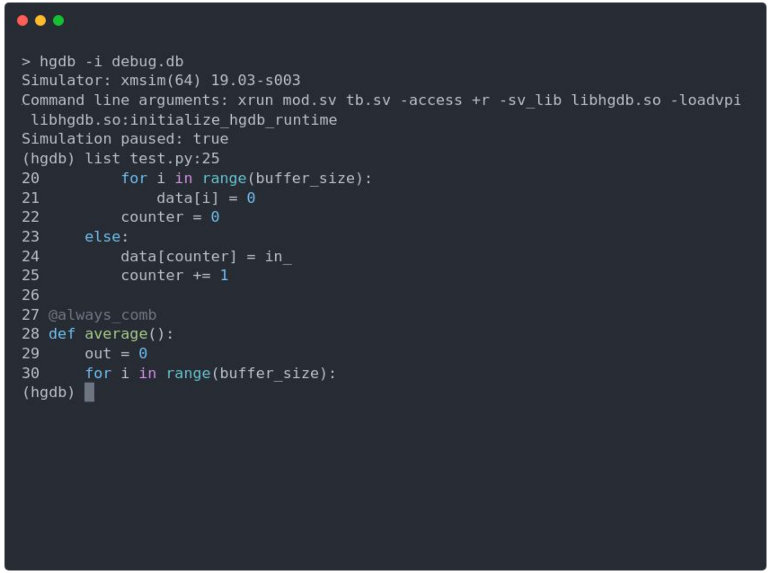
Hgdb debuggers


[Extension Development Host] - ImplicitStateVendingMachine.scala - examples - Visual Studio Code
File
Edit
Selection
View
Go
Run
Terminal
Help
RUN
No Configurations
ImplicitStateVendingMachine.scala
master*
scala | ✓ ImplicitStateVendingMachine.scala
Ln 15, Col 1
Spaces: 2
UTF-8
LF
Scala

```
VARIABLES
  Local
  Generator Variables
    clock: 1
    incValue: 1
    io: Object
      dime: 0
      dispense: 1
      nickel: 1
    reset: 0
    value: 4
  Simulator Values
    Time: 55
WATCH

CALL STACK                                   PAUSED ON BREAKPOINT
    Instance ID 1  ImplicitStateVendingMachine....

BREAKPOINTS
  ● ✓ ImplicitStateVendingMachine.scala   15:21
  ● ✓ ImplicitStateVendingMachine.scala   16:11
```

```scala
// SPDX-License-Identifier: Apache-2.0

package examples

import chiselTests.ChiselFlatSpec
import chisel3._

// Vending machine implemented with an implicit state machine
class ImplicitStateVendingMachine extends SimpleVendingMachine {
  // We let the value of nickel be 1 and dime be 2 for efficiency reasons
  val value = RegInit(0.asUInt(3.W))
  val incValue = WireDefault(0.asUInt(3.W))
  val doDispense = value >= 4.U // 4 * nickel as 1 == $0.20

  when (doDispense) {
    value := 0.U // No change given
  } .otherwise {
    value := value + incValue
  }

  when (io.nickel) { incValue := 1.U }
  when (io.dime) { incValue := 2.U }

  io.dispense := doDispense
}

class ImplicitStateVendingMachineSpec extends ChiselFlatSpec {
  "An vending machine implemented with implicit state" should "work" in {
    assertTesterPasses { new SimpleVendingMachineTester(new ImplicitStateVendingMachine
  }
}
```

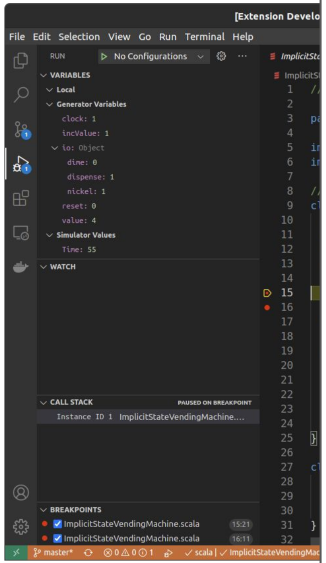
Visual Studio Code



```
> hgdb -i debug.db
Simulator: xmsim(64) 19.03-s003
Command line arguments: xrun mod.sv tb.sv -access +r -sv_lib libhgdb.so -loadvpi
 libhgdb.so:initialize_hgdb_runtime
Simulation paused: true
(hgdb) list test.py:25
20          for i in range(buffer_size):
21              data[i] = 0
22          counter = 0
23      else:
24          data[counter] = in_
25          counter += 1
26
27 @always_comb
28 def average():
29     out = 0
30     for i in range(buffer_size):
(hgdb) 
```

Terminal based

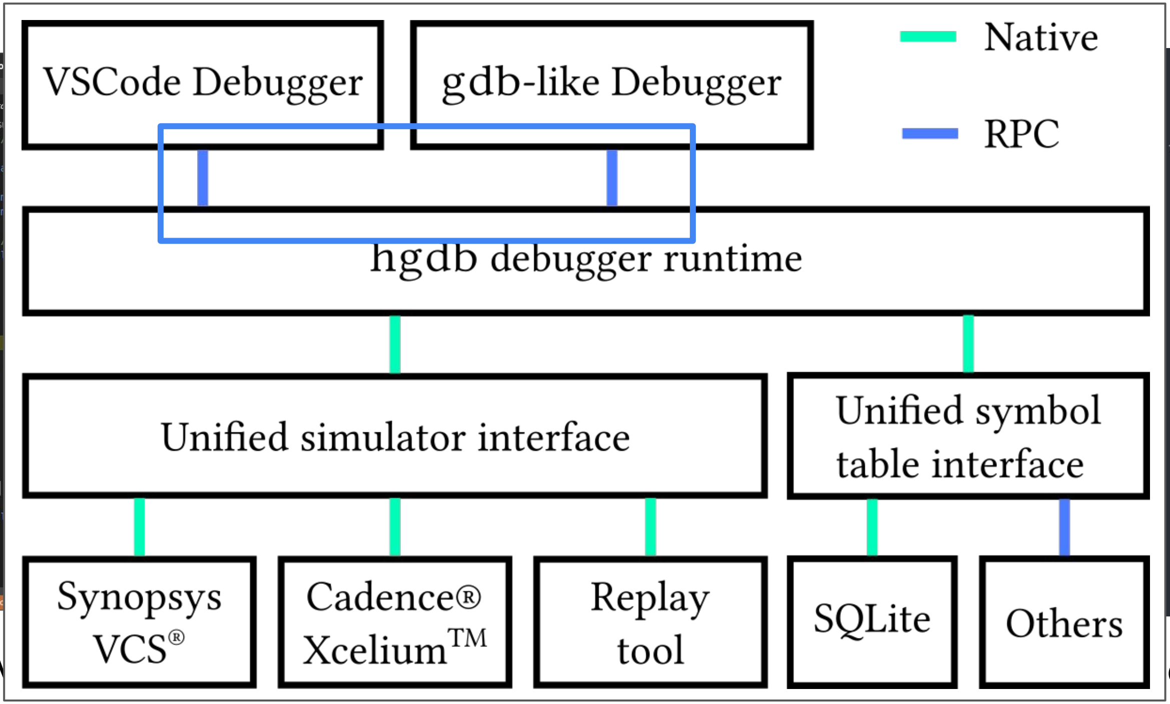
Hgdb debuggers

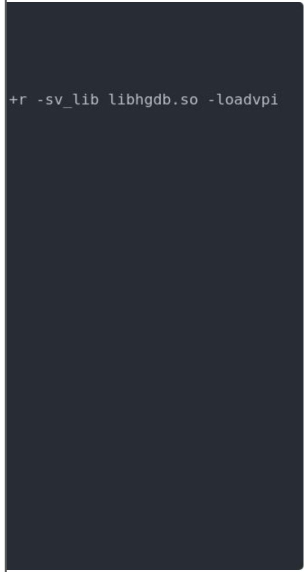


```
Local
  Generator Variables
    clock: 1
    incValue: 1
    io: Object
      dime: 0
      dispense: 1
      nickel: 1
    reset: 0
    value: 4
  Simulator Values
    Time: 55
```

```
Instance ID 1  ImplicitStateVendingMachine....
```

```
ImplicitStateVendingMachine.scala  15:21
ImplicitStateVendingMachine.scala  16:11
```





```
+r -sv_lib libhgdb.so -loadvpi
```

V

ed

How hgdb works with Chisel

```
node _T = bits(io.fn, 3, 3) @[ALU.scala 40:29]
node _T_1 = bits(io.in1, 31, 31) @[ALU.scala 76:55]
node _T_2 = and(_T, _T_1) @[ALU.scala 76:46]
node _T_3 = bits(_T_2, 0, 0) @[Bitwise.scala 72:15]
node _T_4 = mux(_T_3,
                UInt<32>("h0ffffffff"),
                UInt<32>("h00")) @[Bitwise.scala 72:12]
node _T_5 = eq(io.dw, UInt<1>("h01")) @[ALU.scala 77:31]
node _T_6 = bits(io.in1, 63, 32) @[ALU.scala 77:48]
node hi = mux(_T_5, _T_6, _T_4) @[ALU.scala 77:24]
```

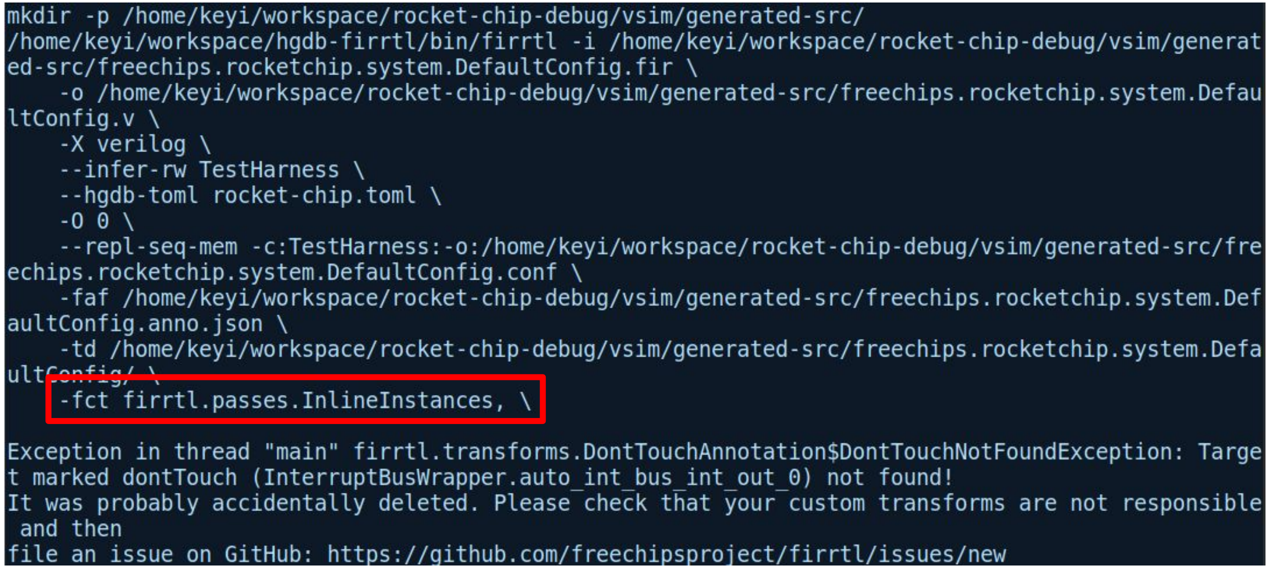
How hgdb works with Chisel

```
node _io_adder_out_T = add(io.in1, in2_inv) @[ALU.scala 63:26]
node _io_adder_out_T_1 = tail(_io_adder_out_T, 1) @[ALU.scala 63:26]
node _io_adder_out_T_2 = bits(io.fn, 3, 3) @[ALU.scala 40:29]
node _io_adder_out_T_3 = add(_io_adder_out_T_1, _io_adder_out_T_2)
@[ALU.scala 63:36]
node _io_adder_out_T_4 = tail(_io_adder_out_T_3, 1) @[ALU.scala 63:36]
io.adder_out <= _io_adder_out_T_4 @[ALU.scala 63:16]
```

```
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]
assign io_out = ~io_dw ? _io_out_T_2 : out; // @[ALU.scala 96:28 ALU.scala 96:37 ALU.scala 93:10]
assign io_adder_out = _io_adder_out_T_1 + _GEN_1; // @[ALU.scala 63:36]
```

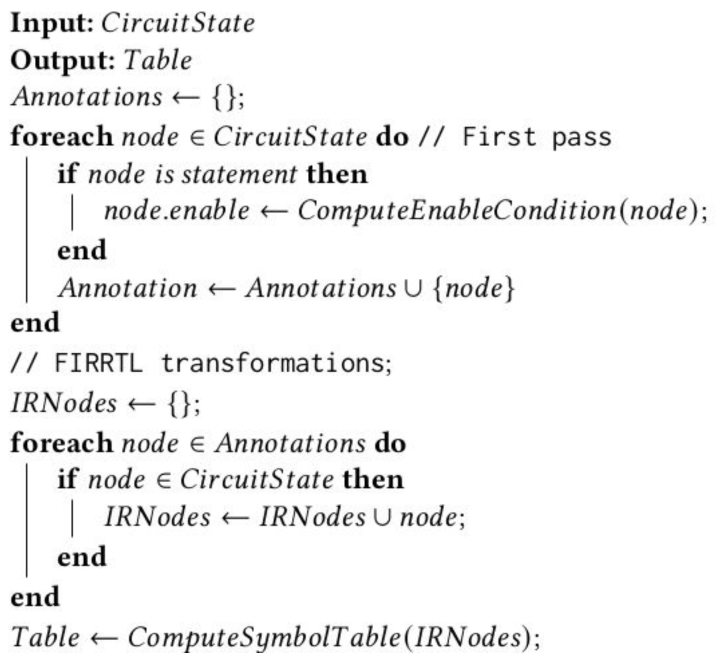
How hgdb works with Chisel

Let’s insert DontTouchAnnotation everywhere!



```
mkdir -p /home/keyi/workspace/rocket-chip-debug/vsim/generated-src/
/home/keyi/workspace/hgdb-firrtl/bin/firrtl -i /home/keyi/workspace/rocket-chip-debug/vsim/generat
ed-src/freechips.rocketchip.system.DefaultConfig.fir \
    -o /home/keyi/workspace/rocket-chip-debug/vsim/generated-src/freechips.rocketchip.system.Defau
ltConfig.v \
    -X verilog \
    --infer-rw TestHarness \
    --hgdb-toml rocket-chip.toml \
    -O 0 \
    --repl-seq-mem -c:TestHarness:-o:/home/keyi/workspace/rocket-chip-debug/vsim/generated-src/fre
echips.rocketchip.system.DefaultConfig.conf \
    -faf /home/keyi/workspace/rocket-chip-debug/vsim/generated-src/freechips.rocketchip.system.Def
aultConfig.anno.json \
    -td /home/keyi/workspace/rocket-chip-debug/vsim/generated-src/freechips.rocketchip.system.Defa
ultConfig/ \
    -fct firrtl.passes.InlineInstances, \

Exception in thread "main" firrtl.transforms.DontTouchAnnotation$DontTouchNotFoundException: Targe
t marked dontTouch (InterruptBusWrapper.auto_int_bus_int_out_0) not found!
It was probably accidentally deleted. Please check that your custom transforms are not responsible
 and then
file an issue on GitHub: https://github.com/freechipsproject/firrtl/issues/new
```

It finally works!



```
Input: $CircuitState$
Output: $Table$
$Annotations \leftarrow \{\}$;
foreach $node \in CircuitState$ do // First pass
    if $node$ is statement then
        $node.enable \leftarrow ComputeEnableCondition(node)$;
    end
    $Annotation \leftarrow Annotations \cup \{node\}$
end
// FIRRTL transformations;
$IRNodes \leftarrow \{\}$;
foreach $node \in Annotations$ do
    if $node \in CircuitState$ then
        $IRNodes \leftarrow IRNodes \cup node$;
    end
end
$Table \leftarrow ComputeSymbolTable(IRNodes)$;
```

First pass:

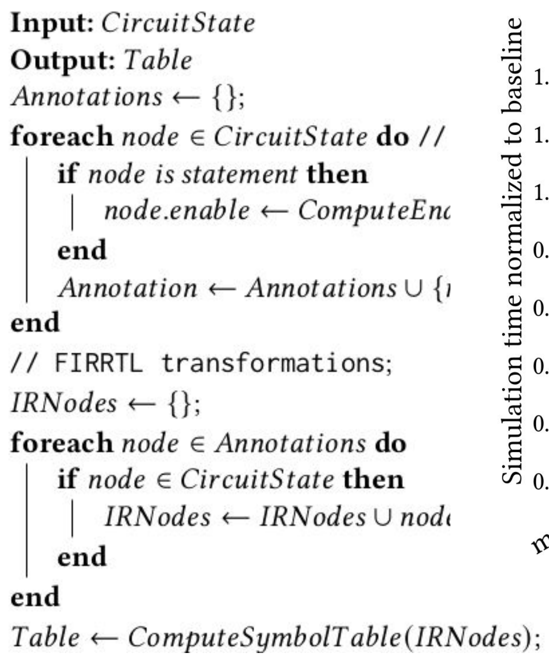
- Insert annotation and compute enable condition
- (Debug mode) insert DontTouchAnnotation

Second pass:

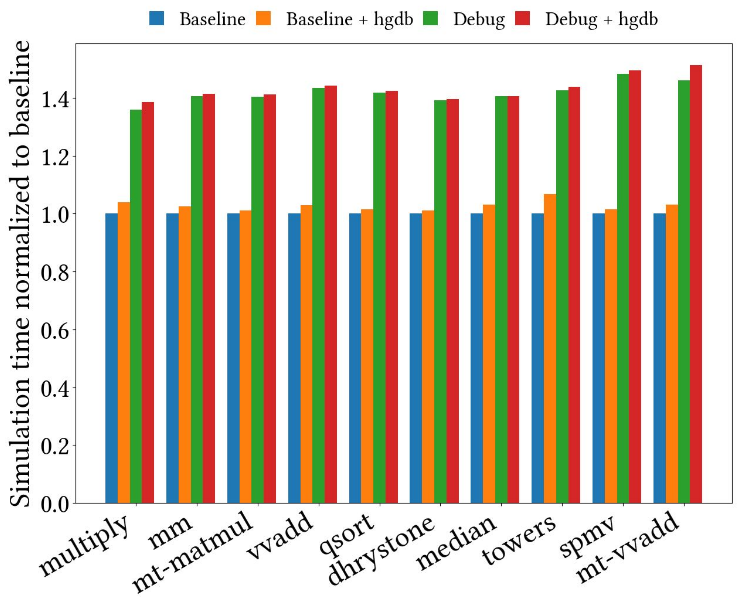
- Collect annotations and only compute symbol table if the IRNode still exists

https://github.com/Kuree/hgdb-firrtl

It finally works!



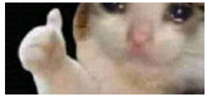
```
Input: $CircuitState$
Output: $Table$
$Annotations \leftarrow \{\}$;
foreach $node \in CircuitState$ do //
    if $node$ is statement then
        $node.enable \leftarrow ComputeEn$
    end
    $Annotation \leftarrow Annotations \cup \{$
end
// FIRRTL transformations;
$IRNodes \leftarrow \{\}$;
foreach $node \in Annotations$ do
    if $node \in CircuitState$ then
        $IRNodes \leftarrow IRNodes \cup node$
    end
end
$Table \leftarrow ComputeSymbolTable(IRNodes)$;
```



ompute enable

ontTouchAnnotation

only compute
ode still exists



https://github.com/Kuree/hgdb-firrtl

Future work in progress

- Work presented to LLVM/CIRCT and SiFive
  - Integrate hgdb to CIRCT system
  - Leverage LLVM compiler toolchain
- Watchpoint
  - We believe watchpoint can be effectively emulated with extra information from the IR
  - Similar to breakpoint emulation
  - Can replace driver tracing if used with reverse debugging
- Symbol table redesign
  - Current implementation “too flat”.
  - More akin to DWARF but simpler and easy to read/write

Conclusion

- Hgdb bridges the gap between hardware generator frameworks and existing simulators
  - Efficient
  - Works with all major simulator vendors
  - What you debug is what you tape out
- Hardware generators are fairly new, and debugging infrastructure is almost non-existing.
  - CIRCIT is the future???
- Hgdb is an open-source framework
  - Contributions are welcome!

https://github.com/Kuree/hgdb

Thank you