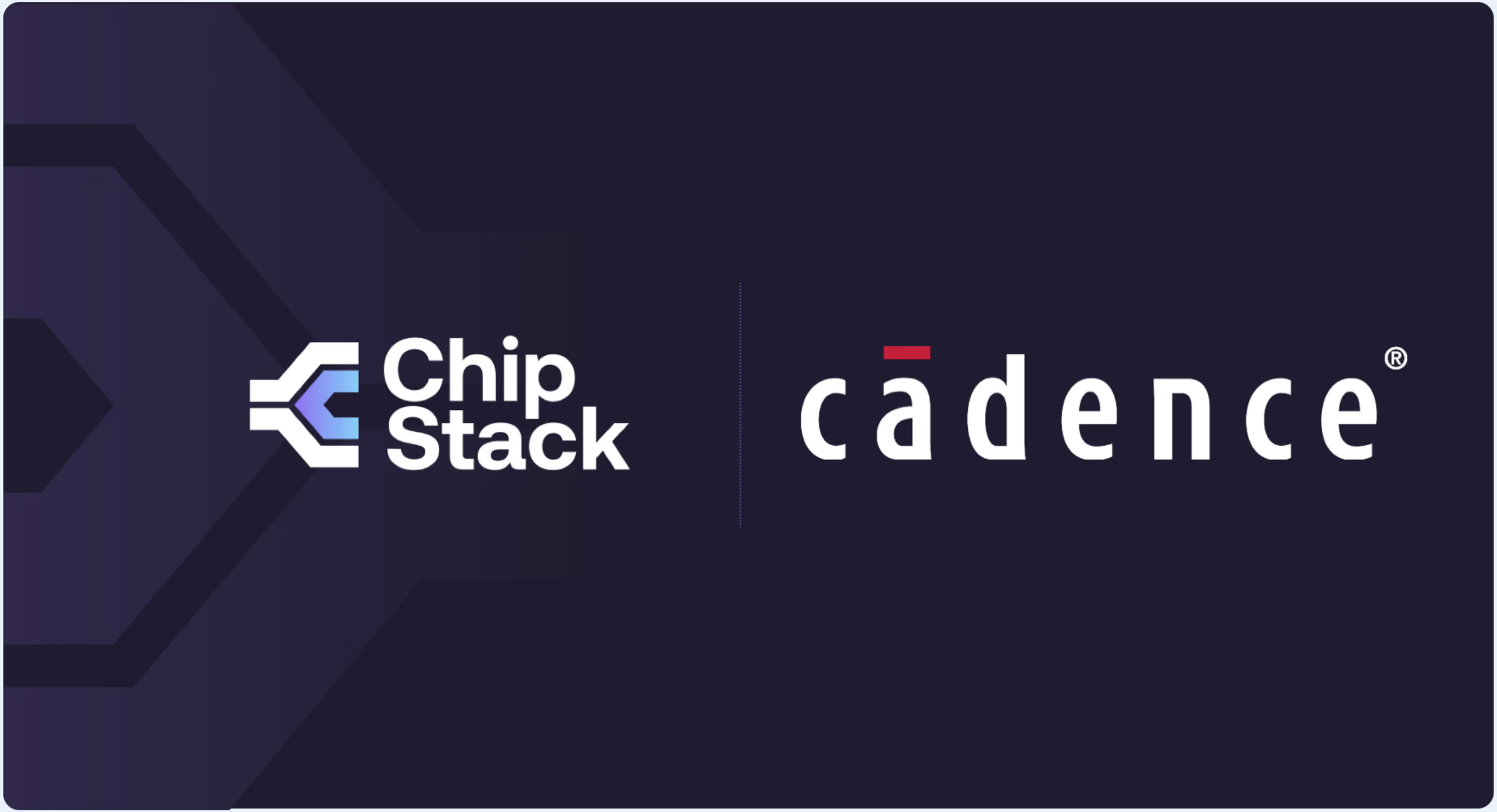
2026
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

DUET: Agentic Design Understanding via Experimentation and Testing

Gus Smith (Southmountain Research; formerly @ ChipStack)

Sandesh Adhikary, Vineet Thumuluri, Karthik Suresh, Vivek Pandit,
Kartik Hegde, Hamid Shojaei, Chandra Bhagavatula (Cadence; formerly @ ChipStack)


Chip Stack
cadence®

AI agents struggle to understand hardware designs.

We hypothesize agents can build better Design Understanding through Experimentation and Testing (DUET) using simulators, waveform viewers, and other tools.

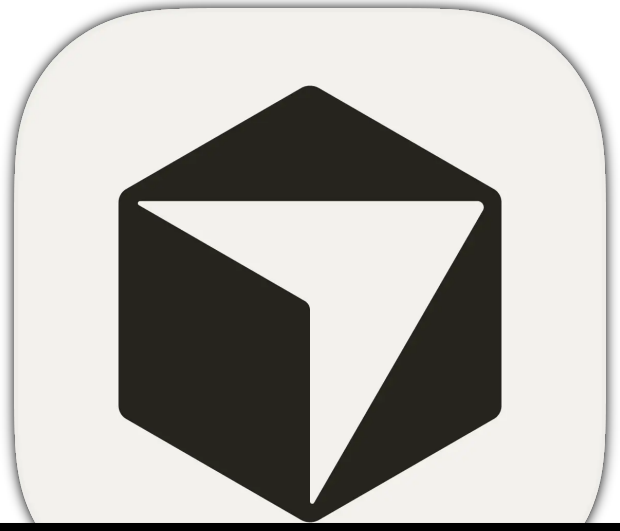
We demonstrate that the DUET methodology improves agents' ability on downstream tasks.

AI agents struggle to understand hardware designs.

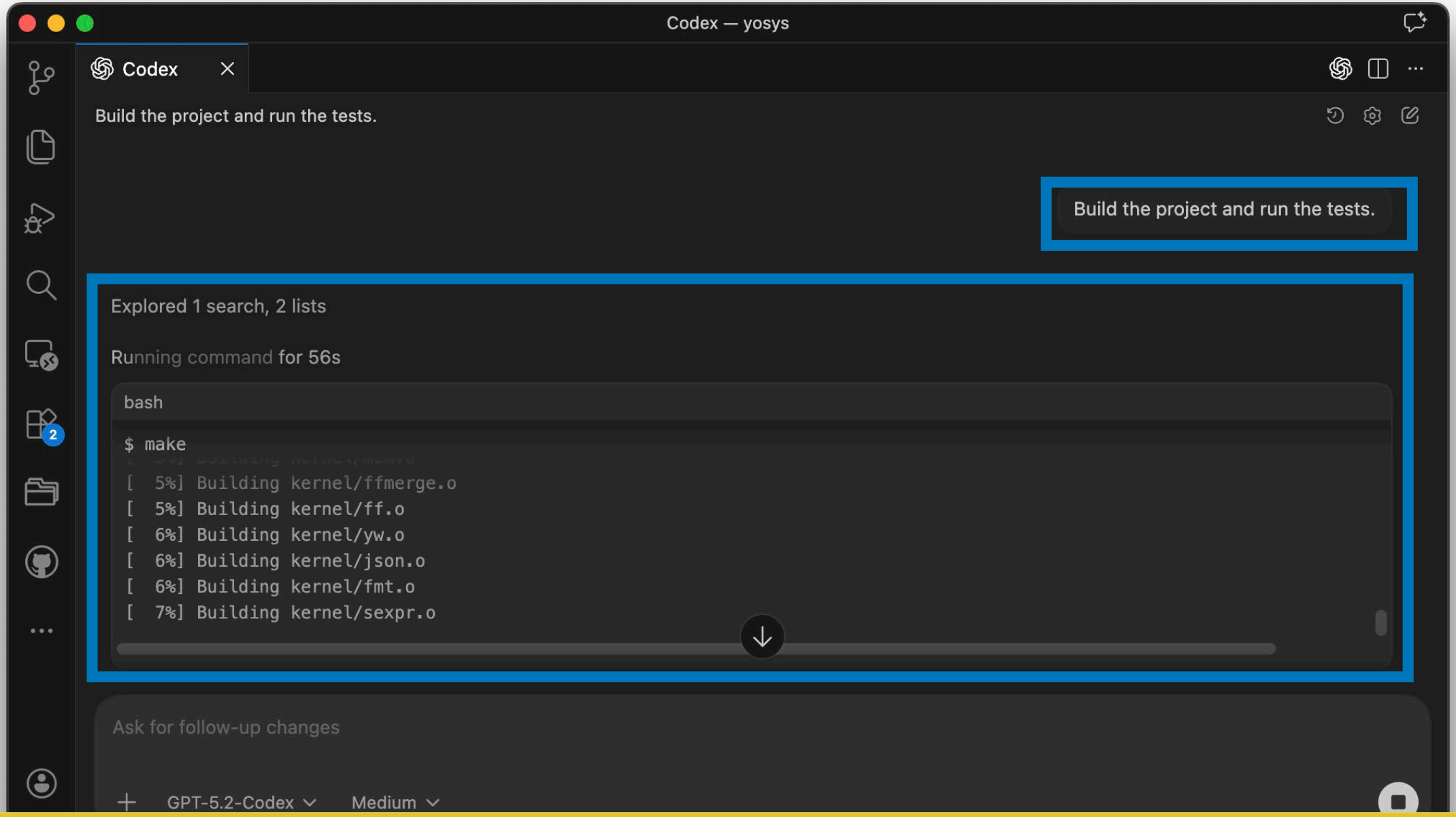
We hypothesize agents can build better Design Understanding through Experimentation and Testing (DUET) using simulators, waveform viewers, and other tools.

We demonstrate that the DUET methodology improves agents' ability on downstream tasks.

AI agents are becoming more and more common.

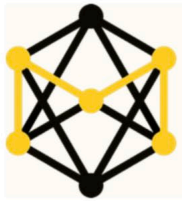


Beyond just a simple chat interface, AI agents run on your machine, allowing them to automate local (or even remote) development tasks.


Codex — yosys
Codex
Build the project and run the tests.
Build the project and run the tests.
Explored 1 search, 2 lists
Running command for 56s
bash
$ make
[5%] Building kernel/ffmerge.o
[5%] Building kernel/ff.o
[6%] Building kernel/yw.o
[6%] Building kernel/json.o
[6%] Building kernel/fmt.o
[7%] Building kernel/sexpr.o
Ask for follow-up changes
GPT-5.2-Codex
Medium

But AI agents still struggle with hardware design tasks.

ChipAgents AI



Moores Lab AI

Chip Stack

SILIMATE

…and many, many others!

cadence®

SYNOPSYS®

Revisiting VerilogEval: A Year of Improvements in Large-Language Models for Hardware Code Generation

NATHANIEL PINCKNEY, NVIDIA Corporati
CHRISTOPHER BATTEN, NVIDIA Corporati
MINGJIE LIU, NVIDIA Corporation, USA
HAOXING REN, NVIDIA Corporation, USA
BRUCEK KHAILANY, NVIDIA Corporation, U

Comprehensive Verilog Design Problems: A Next-Generation Benchmark Dataset for Evaluating Large Language Models and Agents on RTL Design and Verification

Nathaniel Pinckney
NVIDIA
npinckney@nvidia.com

Chenhui Deng
NVIDIA
cdeng@nvidia.com

Chia-Tung Ho
NVIDIA
chiatungh@nvidia.com

Yun-Da Tsai
NVIDIA
yundat@nvidia.com

Mingjie Liu
NVIDIA
mingjiel@nvidia.com

Wenfei Zhou
NVIDIA
wenfeiz@nvidia.com

Brucek Khailany
NVIDIA
bkhailany@nvidia.com

Haoxing Ren
NVIDIA
haoxingr@nvidia.com

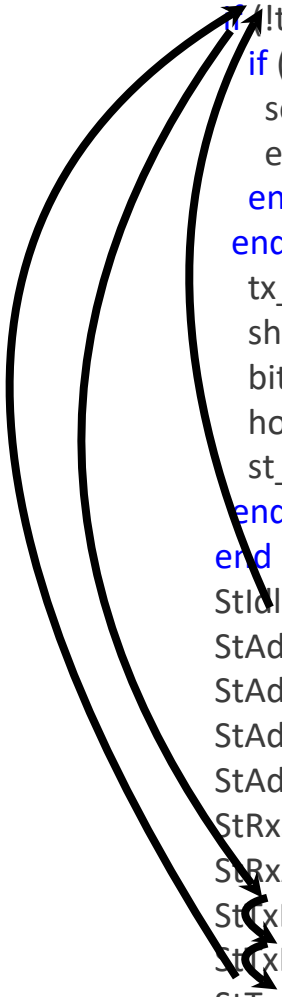
Agents struggle for a number of reasons.

Verilog is likely underrepresented in the training data.

Large design files fill up the context window.

…but also, RTL is just difficult!

```
always_comb begin
  unique case (st_q)
    StTxLoad: begin
      if (!tx_valid_i) begin
        if (stretch_en_i) begin
          scl_oe_d         = 1'b1;
          event_tx_stretch_o = 1'b1;
        end
      end else begin
        tx_ready_d = 1'b1;
        shreg_d    = tx_data_i;
        bitcnt_d   = 4'd0;
        holdcnt_d  = '0;
        st_d       = StTxBitSetup;
      end
    end
    StIdle:          ...
    StAddrShift:     ...
    StAddrAckWaitLow: ...
    StAddrAckHold:   ...
    StAddrAckPulse:  ...
    StRxShift:       ...
    StRxAck:         ...
    StTxBitSetup:    ...
    StTxHold:        ...
    StTxAckWait:     ...
  endcase
end
```



The syntax of RTL is complex and nonlinear.

```
      bitcnt_d   = 4'd0;
      holdcnt_d  = '0;
      st_d       = StTxBitSetup;
    end
  end
  StIdle:          ...
  StAddrShift:     ...
  StAddrAckWaitLow: ...
  StAddrAckHold:   ...
  StAddrAckPulse:  ...
  StRxShift:       ...
  StRxAck:         ...
  StTxBitSetup:    ...
  StTxHold:        ...
  StTxAckWait:     ...
 endcase
end
```

Describe the TX path clock stretching feature.

feature.

The module supports clock stretching on the target side by using open-drain control of SCL: when stretching is active, it drives SCL low (scl_oe_o=1), preventing the bus clock from rising until the controller is ready to proceed.

Preconditions (as referenced):

- target_enable_i = 1
- rw_bit_q = 1 (read / target→host), and the address ACK phase has completed

This is missing a crucial detail!

- rw_bit_q = 1 (read / target→host), and the address ACK phase has completed

Using your description, write a formal verification testbench for the feature.

attempt 1…

attempt 2…

Agent will struggle on downstream tasks when relying on this incomplete description!

AI agents struggle to understand hardware designs.

We hypothesize agents can build better Design Understanding through Experimentation and Testing (DUET) using simulators, waveform viewers, and other tools.

We demonstrate that the DUET methodology improves agents' ability on downstream tasks.

But humans struggle to understand designs too.

[sram_ctrl] Consider refactoring tlul_adapter_sram #7462

Open

This module has grown quite a bit over time and seems a bit difficult to read and debug by now.
Also, the byte write logic inside `tlul_sram_byte` needs additional request FIFOs which could potentially be consolidated with the rest of the `tlul_adapter_sram` FIFOs if we were to refactor this module.

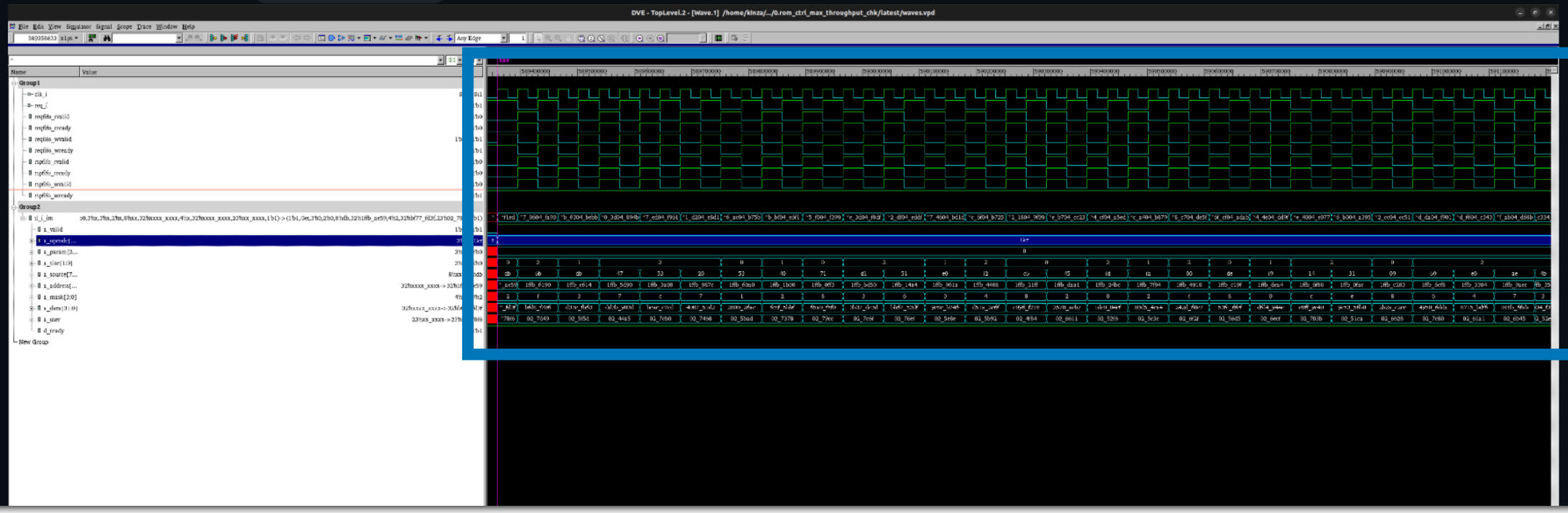
What do humans do when they don’t understand a design?

Use tools!

Open [rom_ctrl,rtl] Potential inefficiency in TL handling #25712

out. But that doesn't really apply to `rom_ctrl`: the ROM will always respond in exactly a cycle, so it's not physically possible to queue up two requests!

However, setting `Outstanding` to 1 causes low throughput, giving waves that look like this:



[spi_device] Combinational loop #5774

Closed

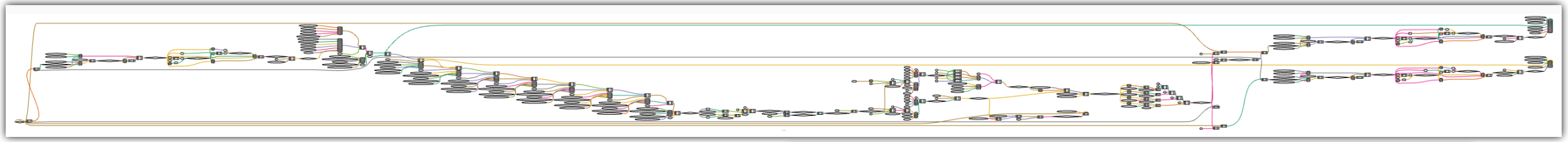
Verilator lint reports a genuine combinational loop in the code. The Verilator report is:

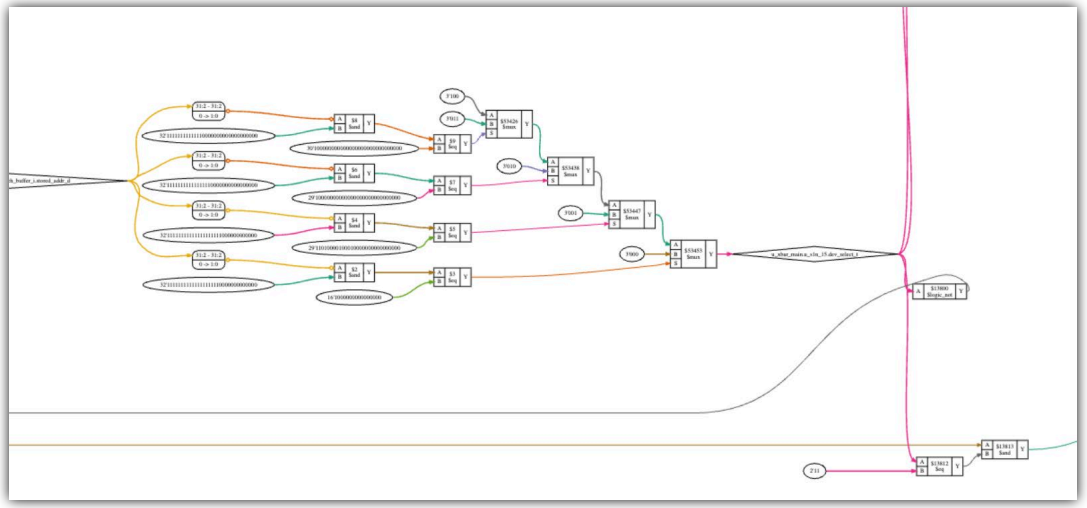
```
%Warning-UNOPTFLAT: ../src/lowrisc_ip_spi_device_0.1/rtl/spi_cmdparse.sv:101:18: Signal unc i:
  101 |   sel_datapath_e sel_dp;
      |                  ^~~~~~
                ../src/lowrisc_ip_spi_device_0.1/rtl/spi_cmdparse.sv:101:18:      Example pat
                ../src/lowrisc_ip_spi_device_0.1/rtl/spi_device.sv:481:3:      Example path:
                ../src/lowrisc_ip_spi_device_0.1/rtl/spi_device.sv:139:23:      Example path:
                ../src/lowrisc_ip_spi_device_0.1/rtl/spi_s2p.sv:44:3:      Example path: ALW
                ../src/lowrisc_ip_spi_device_0.1/rtl/spi_s2p.sv:42:14:      Example path: sp:
```

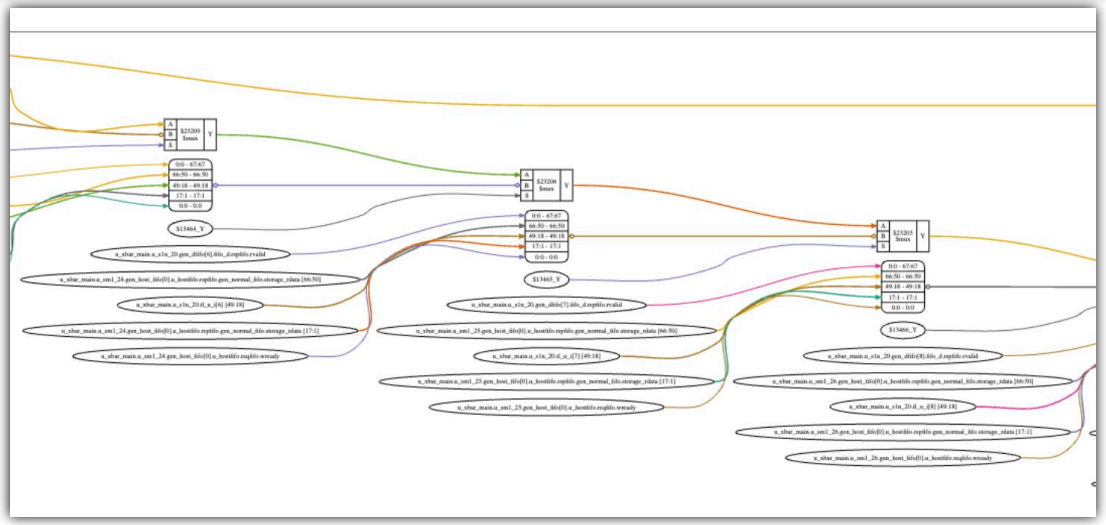
Logic loop in crossbar? #1657

Closed

#1700







Humans need tools to understand complex hardware designs…

So why not give AI agents tools as well?

AI agents struggle to understand hardware designs.

We hypothesize agents can build better Design Understanding through Experimentation and Testing (DUET) using simulators, waveform viewers, and other tools.

We demonstrate that the DUET methodology improves agents' ability on downstream tasks.

Describe the TX path clock stretching feature.

`simulate()`

`view_vcd()`

I will exercise the TX path clock stretching feature first.

Calling `simulate()` with `testbench.sv`...

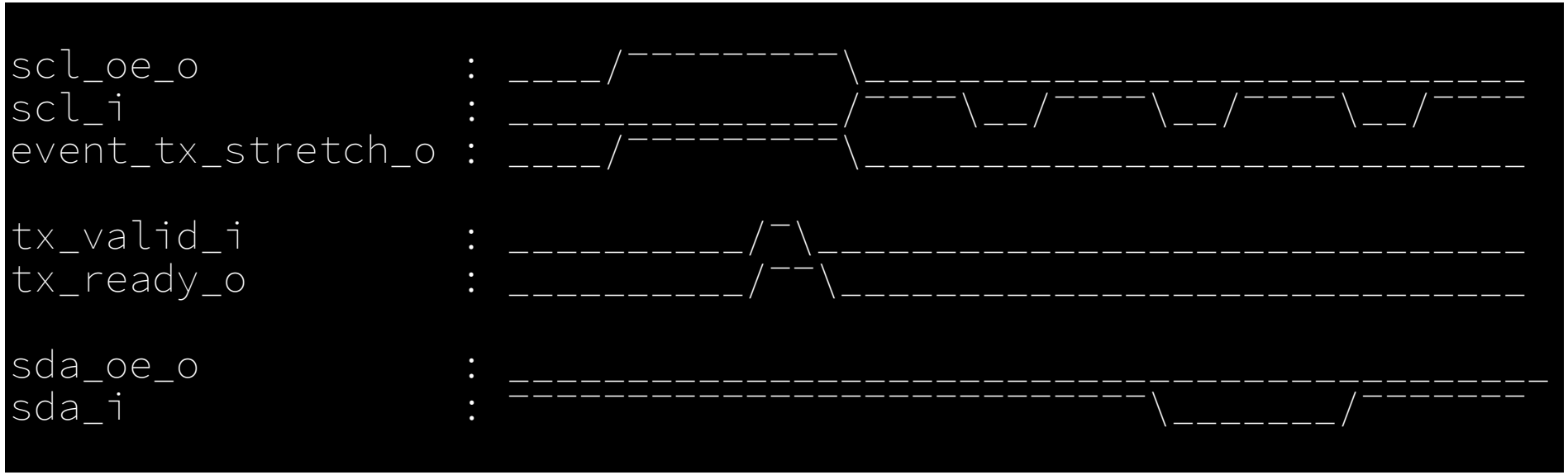
`testbench.sv`

```
module tb_i2c_target_tx_stretch;
  logic clk, rst;
  initial clk = 1'b0;
  always #5 clk = ~clk;
  ...
  i2c_target_controller dut (...);
  ...
  initial begin
    ...
    wait_clks(5);
    rst = 1'b0;
    wait_clks(5);
    target_enable_i = 1'b1;
    target_addr_i   = 7'h42;
    stretch_en_i    = 1'b1;
    i2c_idle();
    i2c_start();
    ...
```

Calling `view_vcd()`...

```
target_enable_i = 1'b1;
target_addr_i   = 7'h42;
stretch_en_i    = 1'b1;
i2c_idle();
i2c_start();
...
```

Calling `view_vcd()`...



```
sda_oe_o            : ___________________________________________
sda_i               : ________________________________\_______/_______
```

This isn't what I expected! I will adjust my understanding and try a new testbench.

Calling `simulate()` with `testbench.sv`...

...agent iterates a few more times...

The module supports clock stretching on the target side by using open-drain control of SCL: when stretching is active, it drives SCL low (scl_oe_o=1), preventing the bus clock from rising until the controller is ready to proceed.

Preconditions (as referenced):

- target_enable_i = 1
- rw_bit_q = 1 (read / target→host), and the address ACK phase has completed
- **keep scl_i low long enough for thd_dat_i during AddrAckWait/Hold before releasing high for AddrAckPulse.**

Crucial detail now present!

This example clearly lays out the DUET methodology.

In short: encourage dynamic experimentation with tools anywhere where deep design understanding is needed.

DUET goes beyond just “debugging with tools” — it is applicable in non-debugging contexts, e.g. documentation generation.

Tools:

- **Simulation tool:** simulates a testbench and returns the logs to the agent.
- **Waveform viewer:** allows agent to view and filter large waveforms
- **Formal tool:** runs a formal verification testbench with Jasper and returns the proof status of asserts, coverpoints, and assumes.
- **Design modification tool:** allows agent to temporarily modify the design itself.
- **Yosys-based scripting tool:** allows agent to write Yosys scripts to e.g. extract relevant signals or find logic cones

In general, **simulation is the most commonly used,** likely because of available training data.

Other tools will be used more when the agent is provided thorough examples.

AI agents struggle to understand hardware designs.

We hypothesize agents can build better Design Understanding through Experimentation and Testing (DUET) using simulators, waveform viewers, and other tools.

We d roves

This methodology can and should be applied anywhere where deep design understanding is needed!

AI agents struggle to understand hardware designs.

We hypothesize agents can build better Design Understanding through Experimentation and Testing (DUET) using simulators, waveform viewers, and other tools.

We demonstrate that the DUET methodology improves agents' ability on downstream tasks.

To evaluate DUET, we build an automated formal verification flow and measure its performance with and without DUET.

Goal: given a design and a set of functional properties, verify the properties via Jasper.

Inputs: a simple round-robin arbiter design + 10 properties.

Baseline flow:

1. For each property, the agent:
    i. Generates a formal testbench for the property.
    ii. Runs Jasper on the formal testbench and views the report from Jasper.
    iii. Returns to (i) and iterates until every assertion passes or until it hits its iteration limit.
    iv. Returns the final testbench for the property (passing or non-passing).
2. Verification terminates after the agent has processed each property in the verification plan.

The DUET flow simply adds DUET-based debugging in the loop:

1. For each property, the agent:
    i. Generates a formal testbench for the property.
    ii. Runs Jasper on the formal testbench and views the report from Jasper.
    iii. If verification failed, the agent is encouraged to use tools to debug.
    iv. Returns to (i) and iterates until every assertion passes or until it hits its iteration limit.
    v. Returns the final testbench for the property (passing or non-passing).
2. Verification terminates after the agent has processed each property in the verification plan.

Tools:

- **Simulation and formal tools** described previously.
- **Counterexample replication tool:** a special, more directed case of the simulation tool which uses simulation via Verilator to reproduce a Jasper counterexample.

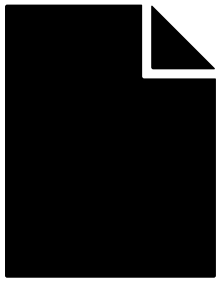
Property	BASELINE	DUET	Improved
1	unproven	proven	✓
2	unproven	unproven, refined	~
3	proven	proven	
4	proven	proven	
5	unproven	unproven	
6	unproven	proven	✓
7	vacuous	proven	✓
8	unproven	unproven	
9	proven	proven	
10	unproven	unproven	
	3/10	6/10	

Property	BASELINE	DUET	Improved
1	unproven	proven	✓
2	unproven	unproven, refined	∼
3	proven	proven	
4	proven	proven	
5	unproven	unproven	
6	unproven	proven	✓
7	vacuous	proven	✓
8	unproven	unproven	
9	proven	proven	
10	unproven	unproven	

Case Study

Property	BASELINE	DUET	Improved
1	unproven	proven	✓
2	unproven	unproven, refined	~
3	proven	proven	
4	proven	proven	
5	unproven	unproven	
6	unproven	proven	✓
7	vacuous	proven	✓
8	unproven	unproven	
9	proven	proven	
10	unproven	unproven	

Property 001: `grant_subset_of_request`
Goal: Prove that in every cycle, `grant` is a subset of `request`



`arbiter.sv`

`simulate()`

`verify()`

`repl_cex()`

arbiter.sv simulate() verify() repl_cex()

Calling `verify()` with `testbench.sv`…

`testbench.sv`

```
module arbiter_grant_subset_sva #(...) (...);
  property p_grant_subset;
    @(posedge clk) (grant & ~request) == '0;
  endproperty
  assert property (p_grant_subset);
endmodule
```

Jasper returns counterexample…

Jasper returns counterexample…

Calling `repl_cex()` with counterexample…

```
module tb_arbiter_ce;
  ...
  arbiter #(...) dut (...);
  initial clk = 1'b0; always #5 clk = ~clk;
  int cyc; initial cyc = 0; always @(posedge clk) cyc <= cyc + 1;
  ...
  always @(posedge clk) begin
    if (cyc == 2) begin
      rst <= 1'b1; request <= 6'h22; #1;
      assert (grant[4] == 1'b1 && request[4] == 0);
    end
  end
endmodule
```

```
endmodule
```

Calling `verify()` with `testbench.sv`...

`testbench.sv`

```
module arbiter_grant_subset_sva #(...) (...);
  property p_grant_subset_prev_req;
  @(posedge clk)
    1'b1 |-> ##1 ((grant & ~$past(request)) == '0);
  endproperty
  assert property (p_grant_subset_prev_req);
endmodule
```

Verifies!

AI agents struggle to understand hardware designs.

We hypothesize agents can build better Design Understanding through Experimentation and Testing (DUET) using simulators, waveform viewers, and other tools.

We demonstrate that the DUET methodology improves agents' ability on downstream tasks.

In summary: in any situation where agents need deep design understanding, use dynamic, tool-based experimentation.

Thank you!