2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

OCTOBER 26-27, 2021

Netlist Paths

A TOOL FOR FRONTEND NETLIST ANALYSIS

JAMIE HANLON – GRAPHCORE LTD

GRAPHCORE

accellera SYSTEMS INITIATIVE

Overview

- Context: problem and current solutions
- Overview of Netlist Paths
- Application example 1: critical timing paths
- Application example 2: structural unit tests
- Summary

Context

- Modern RTL designs use a variety of EDA tools to assist in Frontend development and debug.
  - Simulation, linting, power analysis, synthesis, CDC/RDC, formal.
- Custom tooling / flows can provide ways to enhance RTL development.
- Fast feedback to designers improves productivity.
- Standard EDA tools are proprietary (licensed) and TCL is difficult to interoperate with other languages.
- There is a lack of a simple, generic tool that provides access to a source-level netlist model.

Current tooling options

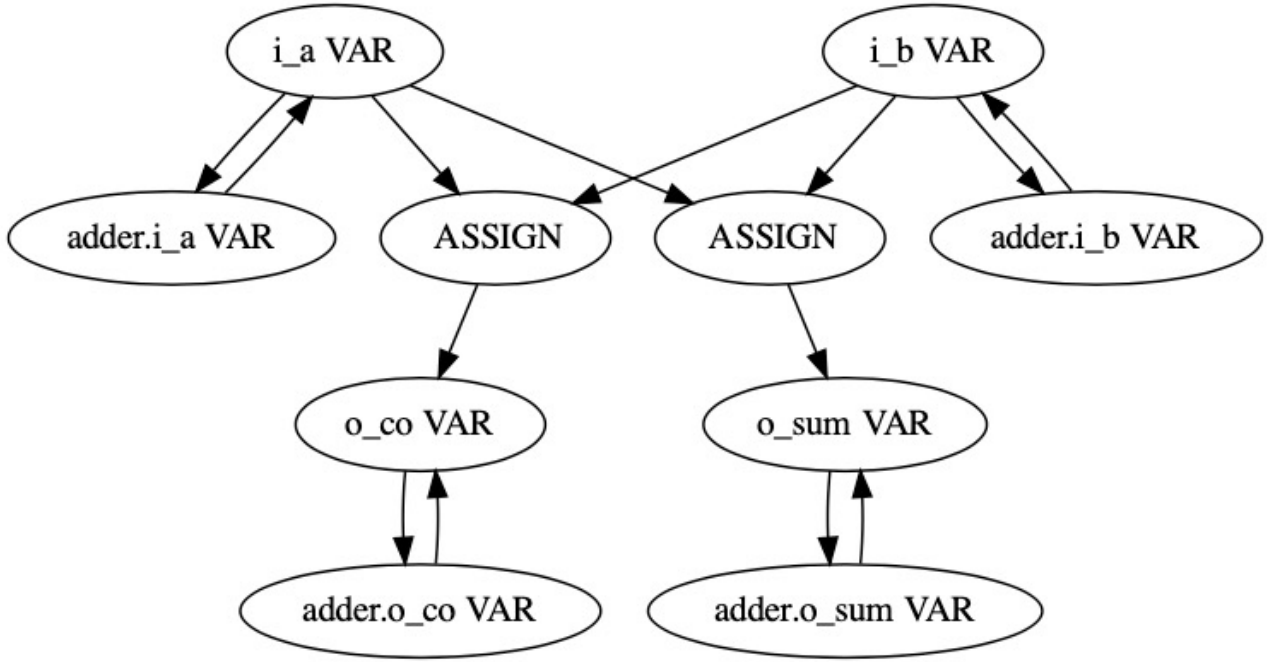
- Simulation (eg VCS, Questa, Xcelium)
  - TCL interfaces are designed around driving the tool, not providing access to the underlying models.
  - Synopsys NPI for Verdi is a C API for data structure access, but each model has limitations.
- Synthesis (eg Fusion Compiler, Genus)
  - RTL can just be elaborated, but source correspondence is degraded.
- Open-source Verilog tools can be modified and repurposed.
  - Parsers: Verible, Slang, Surelog.
  - Synthesis: Yosys.
  - Simulation: Verilator, Icarus.

Netlist Paths

- Open-source library for source-level netlist analysis.
- Uses Verilator to produce a flattened elaborated XML netlist of a design.
- Design connectivity is represented as a directed graph.
- C++ library implementation for performance.
- CLI for straightforward interactive use.
- Python API for easy integration into custom scripting and infrastructure.
- Deployed successfully at Graphcore, aiding the development of high-performance ASIC RTL.

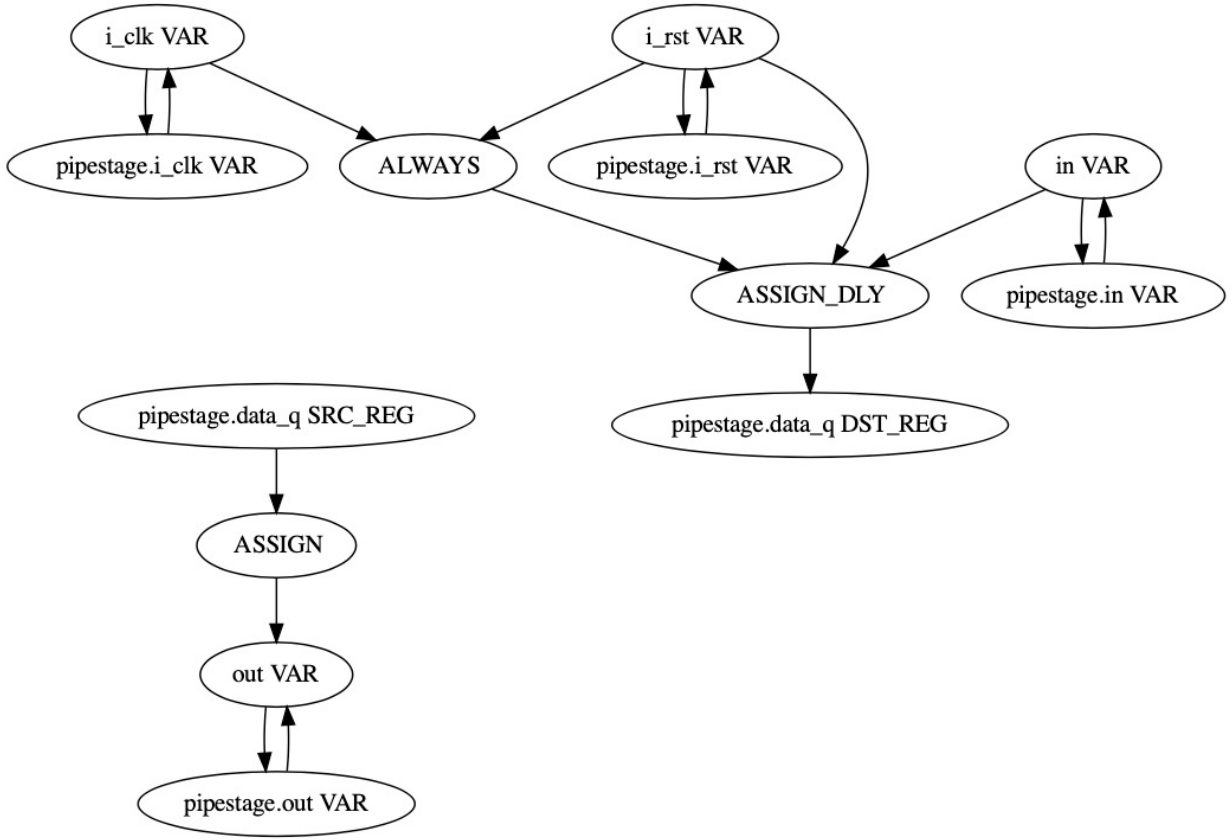
Example design analysis: adder

```
module adder
  #(parameter p_width = 32)(
    input  logic [p_width-1:0] i_a,
    input  logic [p_width-1:0] i_b,
    output logic [p_width-1:0] o_sum,
    output logic               o_co
  );
  assign {o_co, o_sum} = i_a + i_b;
endmodule
```



Example design analysis: register bank

```
module pipestage #(parameter p_width=32) (
    input  logic                 i_clk,
    input  logic                 i_rst,
    input  logic [p_width-1:0] in,
    output logic [p_width-1:0] out
  );
  logic [p_width-1:0] data_q;
  always_ff @(posedge i_clk or posedge i_rst)
    if (i_rst)
      data_q <= '0;
    else
      data_q <= in;
  assign out = data_q;
endmodule
```



Netlist Paths features

- Lookup of variables and datatypes.
  - Matching exactly or with regular expressions or wildcards.
- Path queries:
  - Point to point
  - Fan out from a start point
  - Fan in to an end point
  - Through registers
- Constraints can be added to refine a particular query, to establish correspondences with other tools.
  - Through points must be visited
  - Avoid points must not be visited

Application: critical timing paths

Point	Ref name	Net
cts_inv_15095706023/CK->X	CKINV_Y2	ctsbuf_net_909743178
ctobgt_inst_716854/CK->X	CKINV	ctobgt_76
cts_inv_7287698215/CK->X	CKINV	ctsbuf_net_125742394
cts_inv_7283698211/CK->X	CKINV_CB	ctsbuf_net_124742393
u_idecode_ibuf_rd_sdata_q_reg_9_/CK->Q2	**FSDPRBQM4SS_V2Y**	**u_idecode_ibuf_rd_sdata_q_28_**
HFSBUF_231_469368/A->X	BUF_SM	HFSNET_21922
u_idecode_ctmi_18981/A1->X	ND2_CB	u_idecode_ctmn_16434
ctmi_293593/A->X	ND2B_V1	ctmn_462686
ctmi_164819/A1->X	NR2	ctmn_462752
ctmTdsLR_1_510627/B2->X	NR3B_Y2	N625606
A434661/B2->X	AN3B	ctmn_462803
ctmTdsLR_2_721789/A1->X	NR3_Y2	copt_net_743354
...	...	...
ctmTdsLR_1_721788/A2->X	AOI21	N624302
ctmTdsLR_2_645468/A2->X	ND2	**dec_mx_instr_instrn_3_**
ctmTdsLR_2_721674/A1->X	NR2	copt_net_743315
ctmTdsLR_1_721660/A->X	AN3B	ZBUF_209_47
ctmTdsLR_2_729835/A->X	OR3B_CB	copt_net_746628
ctmTdsLR_1_729834/B->X	AO21B	**u_lsu_i0_op2_mux_2_**
ctmi_306787/A1->X	AN2	ctmn_484205
ctmTdsLR_3_722448/A2->X	OAI21_V1Y2	copt_net_743586
ctmTdsLR_3_635081/A->X	INV	tmp_net720478
ctmTdsLR_4_635082/A1->X	ND4	tmp_net720480
ctmTdsLR_1_722190/C->X	AOAI211	u_lsu_i0_op0_mux_3_
u_lsu_e0_addr0_op1_op2_q_reg_16_/D	**FSDPRBQM4SS_V2Y**	

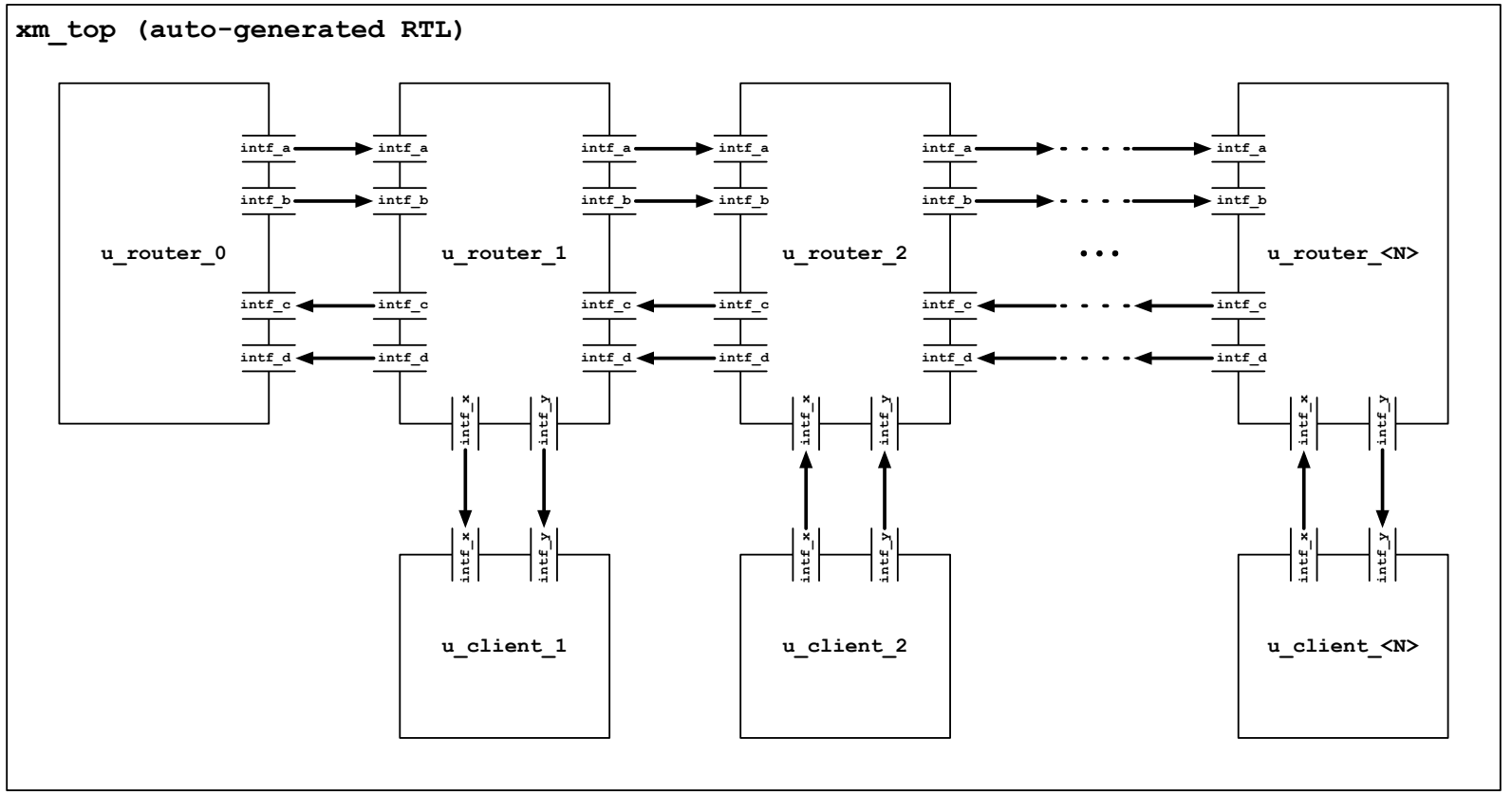
Application: critical timing paths

```
$ netlist-paths xm_tilecpu.netlist.xml \
    --ignore-hierarchy-markers --regex \
    --from u_idecode_ibuf_rd_sdata_q \
    --to u_lsu_e0_op0_q \
    --through dec_mx_instr$ \
    --through i0_op2_mux
Name                                  Type DType          Statement Location
------------------------------------- ---- -------------- --------- ------------------
xm_tilecpu.u_idecode.ibuf_rd_sdata_q REG  [69:0] logic  ASSIGN    xm_idecode.sv:709
...                                   ...  ...    ...    ...       ...
xm_tilecpu.dec_mx_instr               VAR  packed struct ASSIGN    xm_tilecpu.sv:1045
...                                   ...  ...    ...    ...       ...
xm_tilecpu.u_mrf_rf.r1_data           VAR  [32:0] logic  ASSIGN    xm_mrf_rf.sv:1593
...                                   ...  ...    ...    ...       ...
xm_tilecpu.dec_instrn_stall           VAR  logic         ASSIGN    xm_idecode.sv:1290
...                                   ...  ...    ...    ...       ...
xm_tilecpu.u_lsu.i0_op0_mux           VAR  [31:0] logic  ASSIGN    xm_lsu.sv:602
xm_tilecpu.u_lsu.e0_op0_q             REG  [31:0] logic
```

Application: structural unit tests

- Common practice to generate top-level integration RTL.
- Early development is done often without verification support.
- Simulations are expensive to run and slow.
- Netlist paths can be used to implement unit tests to check correct connectivity in the RTL.
  - Python API enables rapid development and integration versus setting up a TCL-based flow with a synthesis tool.

Application: structural unit tests



Application: structural unit tests

```
import py_netlist_paths

# Enforce signal names must match exactly.
py_netlist_paths.Options.get_instance().set_match_exact()
# Allow start and end points to be at nets rather than registers.
py_netlist_paths.Options.get_instance().set_restrict_start_points(False)
py_netlist_paths.Options.get_instance().set_restrict_end_points(False)
# Load the netlist XML.
netlist = py_netlist_paths.Netlist('xm_top.netlist.xml')

for i in range(N - 1):
    # Check the router is connected to the next router along.
    for signal in intf_a + intf_b + intf_c + intf_d:
        # Construct hierarchical paths to the signals.
        point_a = f'xm_top.u_router_{i}.{signal}'
        point_b = f'xm_top.u_router_{i+1}.{signal}'
        # Create a waypoint object for the start and end points.
        waypoint = py_netlist_paths.Waypoints(point_a, point_b)
        # Check that the two points are connected together as expected.
        assert netlist.path_exists(waypoint)
```

Application: structural unit tests

- Performing an equivalent task with Fusion Compiler is slower.
- The Netlist Path's check can be added to the CI test suite with minimal impact on overall runtime.
- Ease of setup and integration with Python has improved code quality and engineering productivity.

Complexity	Run Time (seconds)	
Number of router instances	Fusion Compiler	Netlist Paths
4	252	20
8	295	26
16	345	50
32	463	77
64	721	195

Other use cases

- Checking of patterns to match design objects for QoR reporting.
- CDC/RDC checks to catch simple violations.
- Design documentation.

Summary

- Netlist Paths is a lightweight, open-source tool that provides access to a source-level netlist.
- Allows simple, custom tooling to be built in Python and easily integrated into existing Python infrastructure.
- Successfully deployed at Graphcore to provide fast feedback to the designer.
- These applications have simplified team interactions, increased RTL quality and designer productivity.

Thank you

Jamie Hanlon and Samuel Kong

- jamie@graphcore.ai
- samuelk@graphcore.ai
- www.graphcore.ai

We’re hiring in Bristol & Cambridge!

GRAPHCORE