Verification with cocotb

Paul T. Keener

HEPIC 2024

1 May 2024

What is cocotb?

cocotb is a *COroutine* based *COsimulation TestBench* environment for verifying VHDL and SystemVerilog RTL using Python.

Taken from the documentation.

It is completely free

It is completely simulator agnostic, supporting anything that uses the standard VPI, VHPI, or FLI interfaces.

Most simulators (Verilator support is currently experimental)

Example: Makefile

```
# Makefile
# following example at https://docs.cocotb.org/en/stable/quickstart.html

# defaults
SIM ?= xcelium
TOPLEVEL_LANG ?= verilog

VERILOG_SOURCES += $(PWD)/adder_pipeline.v
# use VHDL_SOURCES for VHDL files

# TOPLEVEL is the name of the top-level module in your Verilog or VHDL file
TOPLEVEL = tb

# MODULE is the basename of the Python test file (which I named tb.py)
MODULE = tb

# include cocotb's make rules to take care of the simulator setup
include $(shell cocotb-config --makefiles)/Makefile.sim
```

Example: Verilog Module

```
module adder
(
 input wire clk,
 input wire [7:0] ain,
 input wire [7:0] bin,
 output reg [8:0] out
 );
    initial out = 0;
    reg [7:0] apl = 0;  // pipelined copy of ain, bin
    reg [7:0] bpl = 0;
    reg [8:0] q = 0;    // adder result
    reg [8:0] qpl = 0;  // pipelined copy of q
    always @ (posedge clk) begin
        apl <= ain;
        bpl <= bin;
        q   <= apl + bpl;
        qpl <= q;
        out <= qpl;
    end
endmodule
```

Example: Verilog TestBench

```
module tb;
    // Create a 100 MHz system clock: it is possible to do this from
    // the cocotb Python code, but I find it convenient (and more
    // CPU-efficient) to use a Verilog testbench for this.  Cocotb's
    // own examples tend not to have a separate Verilog testbench.
    reg clk = 0;
    initial begin
        while (1) begin
            clk = 0;
            #5;
            clk = 1;
            #5;
        end
    end

    initial begin
        $dumpfile("tb.vcd");
        $dumpvars(0, tb);
    end

    // Inputs (reg) and outputs (wire) for adder instance
    reg [7:0] ain = 0;
    reg [7:0] bin = 0;
    wire [8:0] out;
    adder a (.clk(clk), .ain(ain), .bin(bin), .out(out));
endmodule
```

Example: Python Driver

```
import cocotb
from cocotb.triggers import FallingEdge

import random

@cocotb.test()
async def my_test(dut):

    for i in range(10):
        await FallingEdge(dut.clk)
    dut._log.info("hello world")
    print("hello world - ordinary print function (no time stamp in log)")
    a_list = []
    b_list = []
    o_list = []
    for i in range(100):
        await FallingEdge(dut.clk)
        a = random.randint(0, 0xff)
        b = random.randint(0, 0xff)
        o = a + b
        a_list.append(a)
        b_list.append(b)
        o_list.append(o)
        dut.ain.value = a
        dut.bin.value = b
        if i < 5: continue  # don't test until pipeline has filled
        assert dut.a.apl.value == a_list[-2]
        assert dut.a.bpl.value == b_list[-2]
        assert dut.a.q.value == o_list[-3]
        assert dut.a.qpl.value == o_list[-4]
        assert dut.a.out.value == o_list[-5]
    for i in range(10):
        await FallingEdge(dut.clk)
```

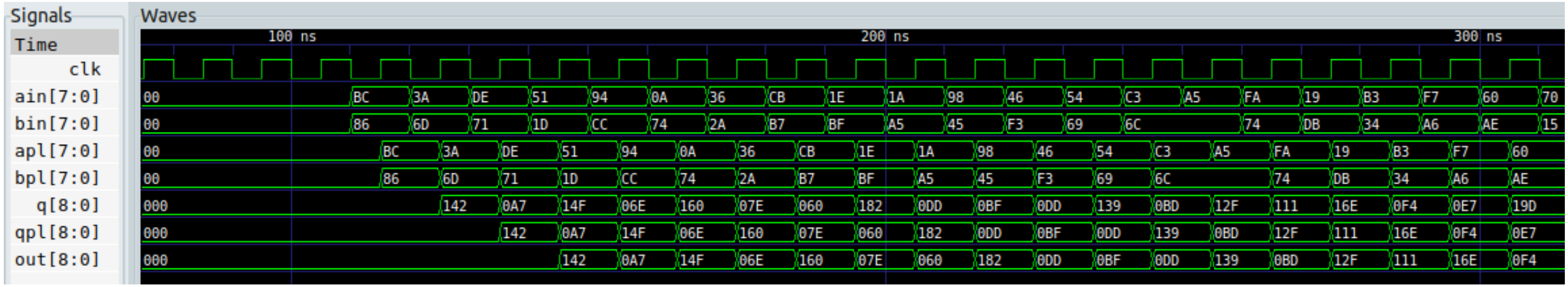
Example: Running cocotb (1/2)

```
TOOL:   xrun(64)        20.09-s005: Started on Apr 30, 2024 at 23:35:06 EDT
xrun(64): 20.09-s005: (c) Copyright 1995-2020 Cadence Design Systems, Inc.
Loading snapshot worklib.tb:v ..................... Done
     -.--ns INFO     cocotb.gpi                ..mbed/gpi_embed.cpp:100  in set_program_name_in_venv       Using Python virtual
environment interpreter at /home/u2/keener/cocotb/env/bin/python
     -.--ns INFO     cocotb.gpi                ../gpi/GpiCommon.cpp:105  in gpi_print_registered_impl      VPI registered
xcelium> source /cad/XCELIUM2009/tools/xcelium/files/xmsimrc
xcelium> run
     -.--ns INFO     cocotb.gpi                ..mbed/gpi_embed.cpp:240  in _embed_sim_init               Python interpreter initialized
and cocotb loaded!
     0.00ns INFO     cocotb                         __init__.py:220  in _initialise_testbench_          Running on xmsim(64) version
20.09-s005
     0.00ns INFO     cocotb                     __init__.py:227  in _initialise_testbench_         Running tests with cocotb v1.5.1 from
/home/u2/keener/cocotb/env/lib64/python3.6/site-packages/cocotb
     0.00ns INFO     cocotb                     __init__.py:247  in _initialise_testbench_         Seeding Python random module with
1714534507
     0.00ns WARNING  cocotb                     __init__.py:266  in _initialise_testbench_         Pytest not found, assertion rewriting
will not occur
     0.00ns INFO     cocotb.regression          regression.py:127  in __init__                       Found test tb.my_test
     0.00ns INFO     cocotb.regression          regression.py:472  in _start_test                    Running test 1/1: my_test
```

Example: Running cocotb (2/2)

```
     0.00ns INFO     cocotb.test.my_test..  decorators.py:313  in _advance               Starting test: "my_test"
   100.00ns INFO     cocotb.tb                             tb.py:19   in my_test            hello world
hello world - ordinary print function (no time stamp in log)
  1200.00ns INFO     cocotb.regression           regression.py:364  in _score_test        Test Passed: my_test
  1200.00ns INFO     cocotb.regression           regression.py:488  in _log_test_summary  Passed 1 tests (0 skipped)
  1200.00ns INFO     cocotb.regression           regression.py:557  in _log_test_summary  **************************************************************************
                                                                                           ** TEST        PASS/FAIL  SIM TIME(NS)  REAL TIME(S)  RATIO(NS/S) **
                                                                                           **************************************************************************
                                                                                           ** tb.my_test    PASS         1200.00          0.02     57005.95  **
                                                                                           **************************************************************************

  1200.00ns INFO     cocotb.regression           regression.py:574  in _log_sim_summary   ********************************************************************************************
                                                                                           **                                 ERRORS : 0                                 **
                                                                                           ********************************************************************************************
                                                                                           **                               SIM TIME : 1200.00 NS                        **
                                                                                           **                              REAL TIME : 0.03 S                            **
                                                                                           **                        SIM / REAL TIME : 39607.55 NS/S                     **
                                                                                           ********************************************************************************************

  1200.00ns INFO     cocotb.regression           regression.py:259  in tear_down          Shutting down...
```

Example: Waveform



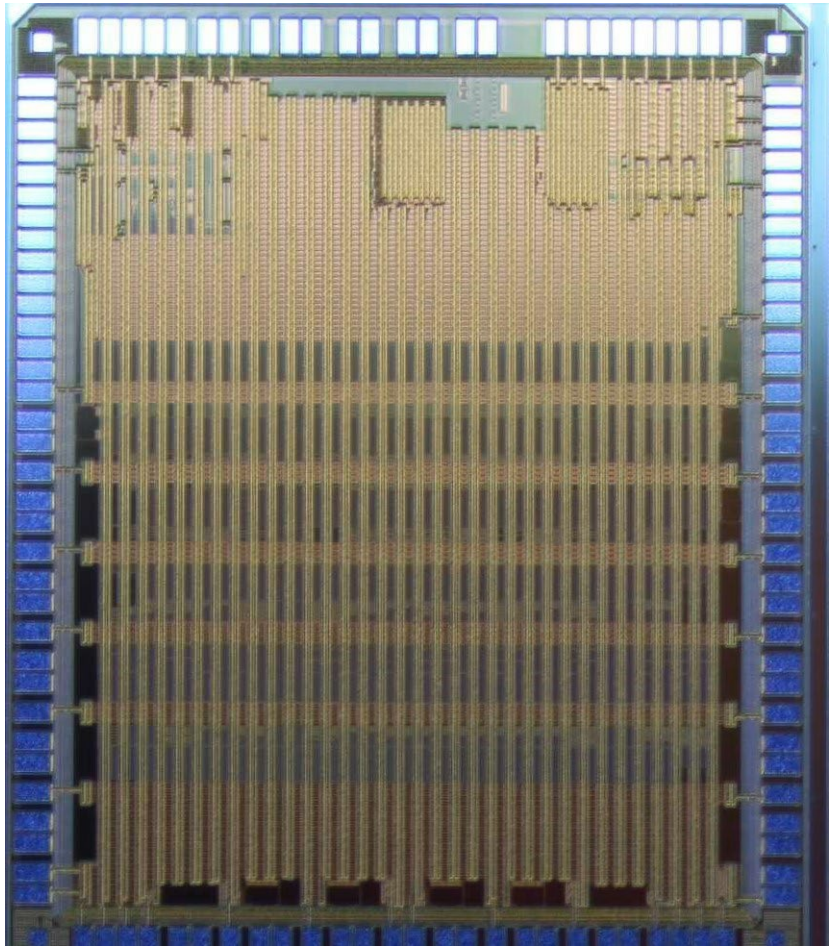
Why use cocotb?

1. Students already know it
   - Students generally don't come in knowing any HDL
2. It brings in a whole useful ecosystem of analysis and visualization packages *for free*
3. It is a "gateway drug" to teach students HDL
   - Students can immediately be useful in verification efforts, but as they get further they will tend to learn HDL from the design they are verifying
4. The python that gets written for verification is reusable, either literally or conceptually, in control and DAQ code for the physical ASIC

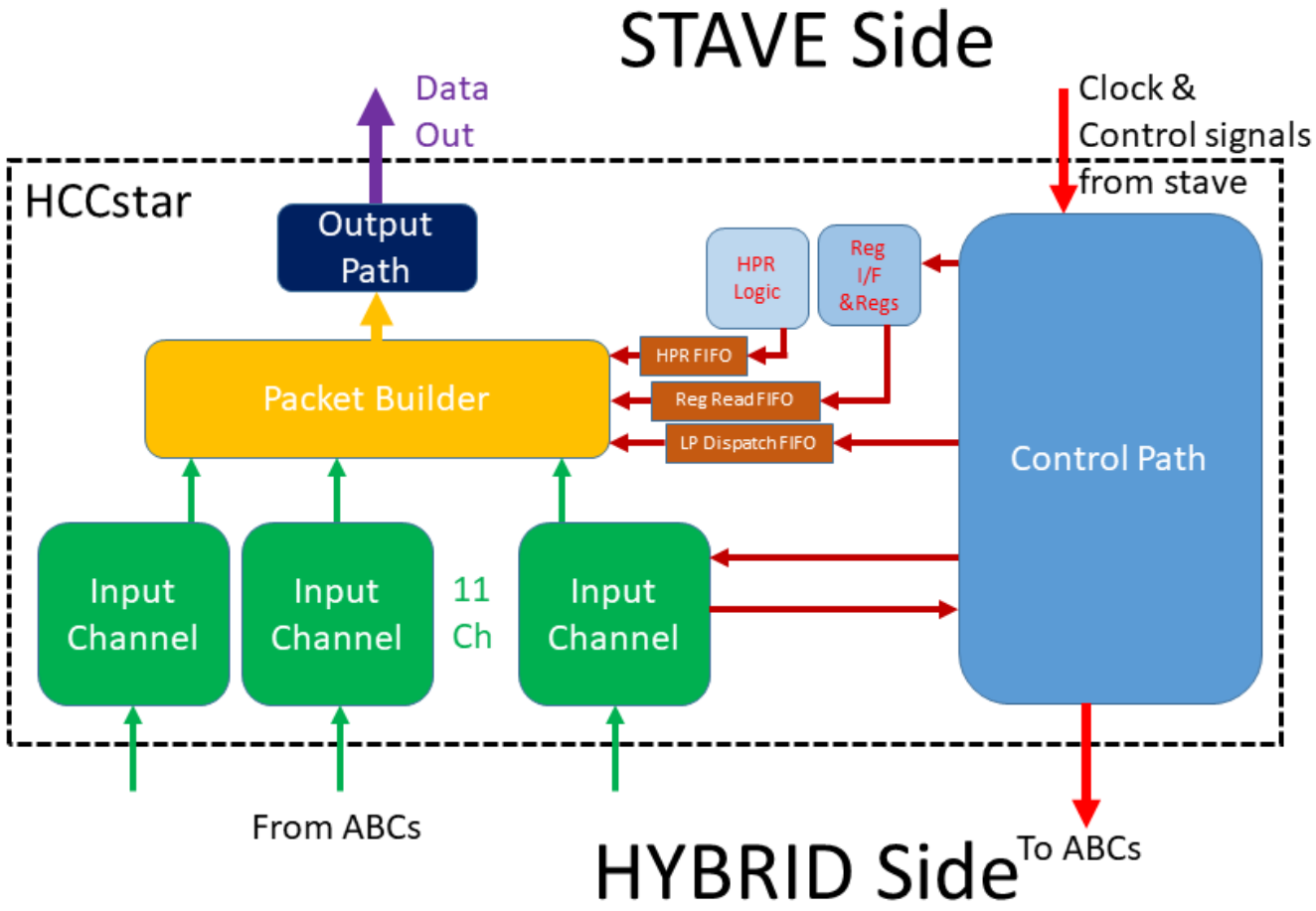
Non-Trivial Example: ATLAS ITk Strip HCCStar

Hybrid Controller Chip

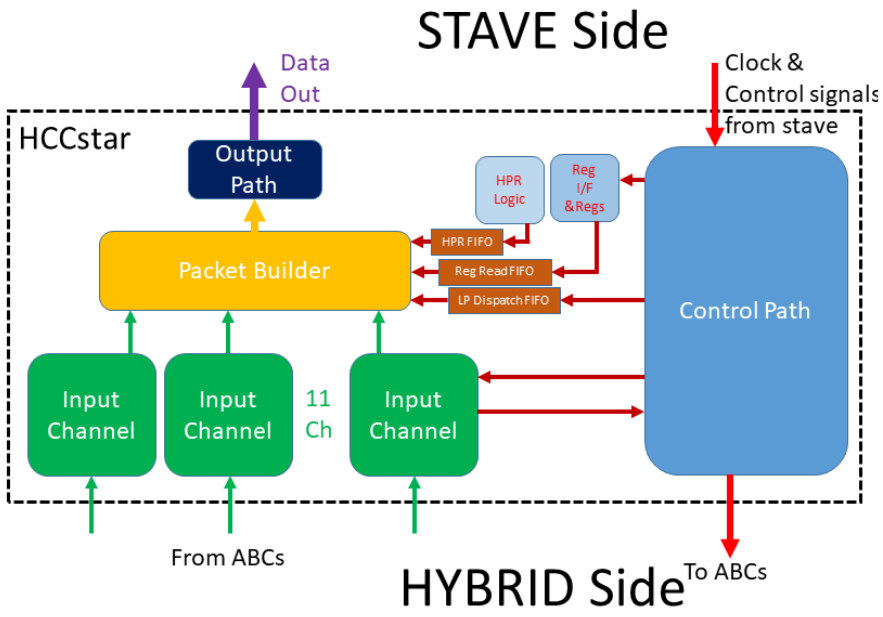
- GF 130nm
- TMR with CERN TMRG tool
- 12.5 Kilo Lines of RTL (untriplicated)
- 188 kgates, triplicated, synthesis
- 17 KLOC – python
  - 50% infrastructure
  - 50% tests
- ~120 separate tests

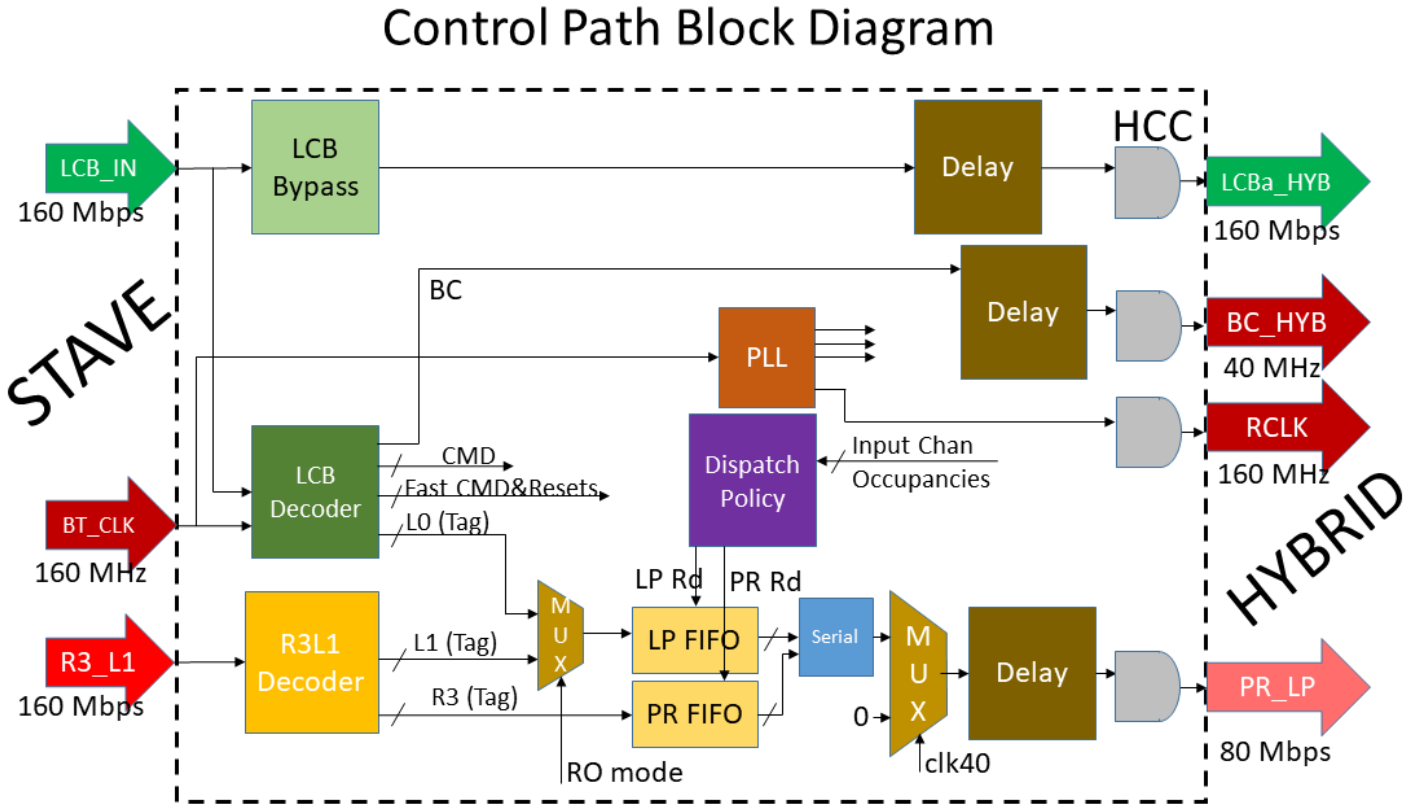


HCCStar Top Level

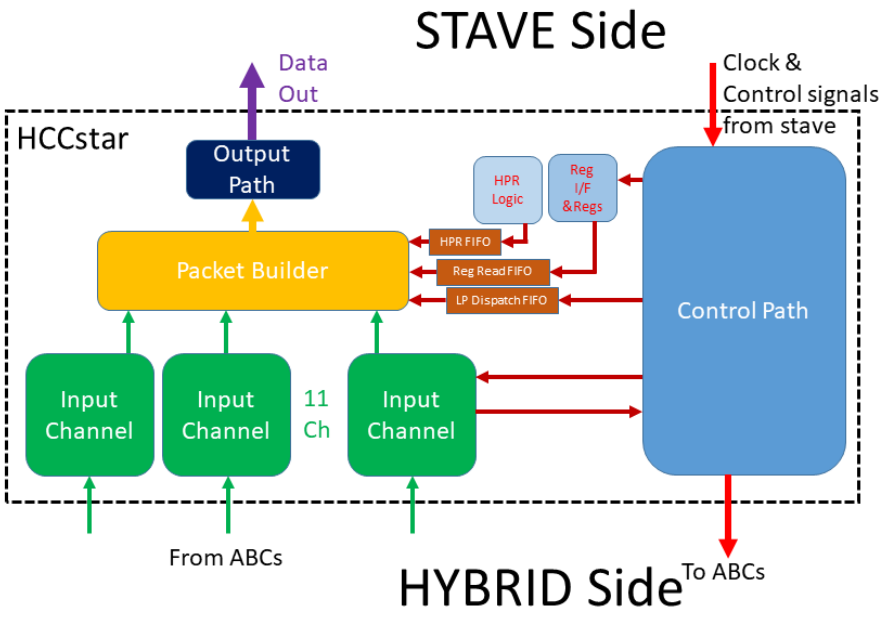


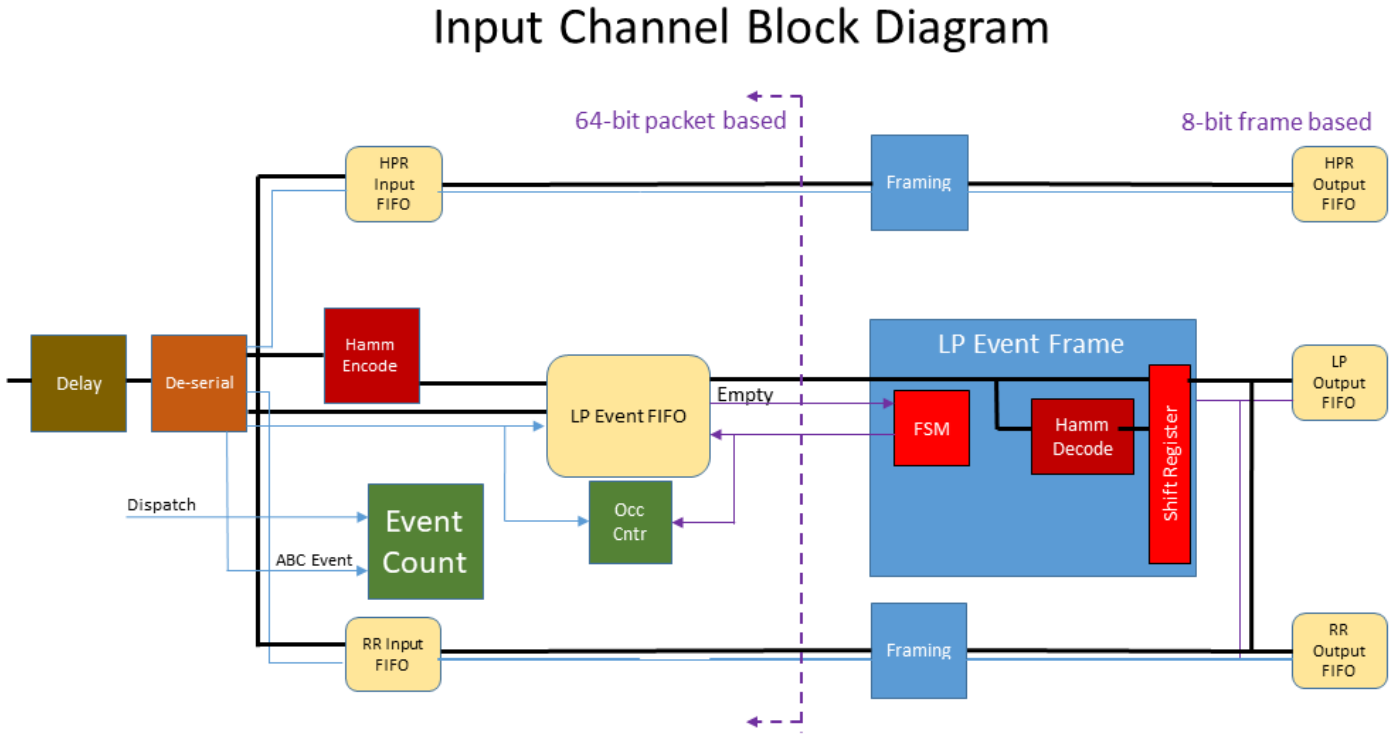
HCCStar Control Path



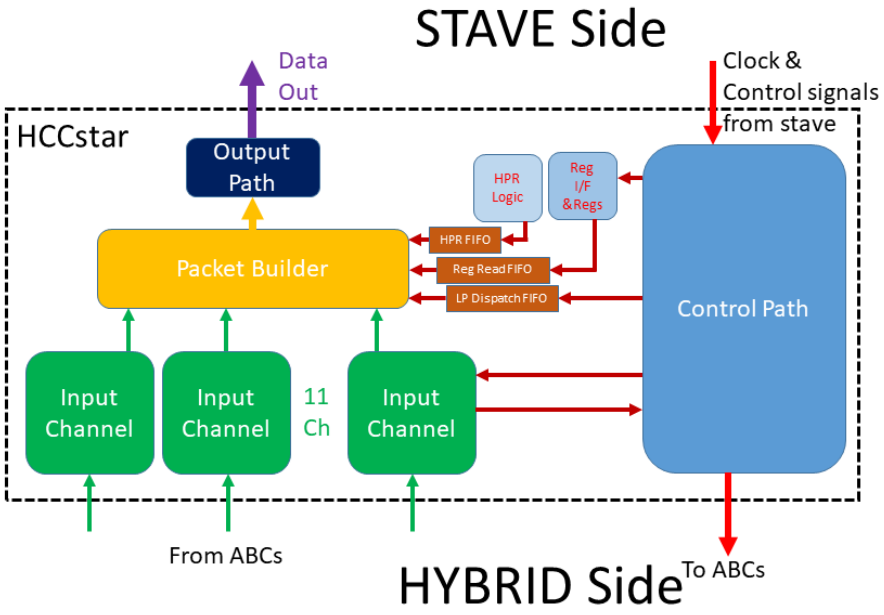


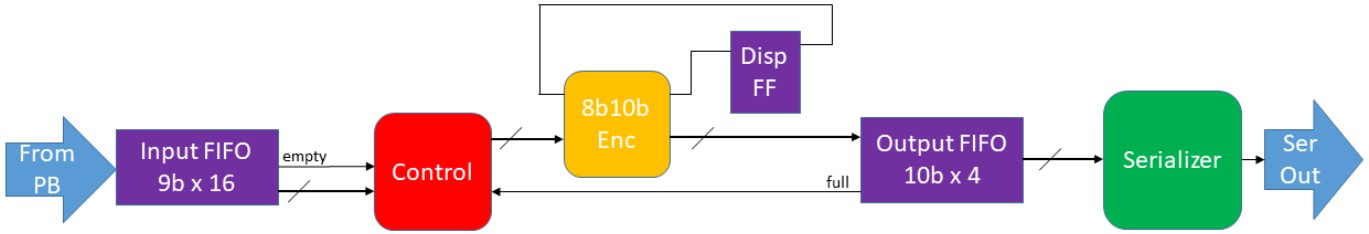
HCCStar Input Channel





HCCStar Output Path

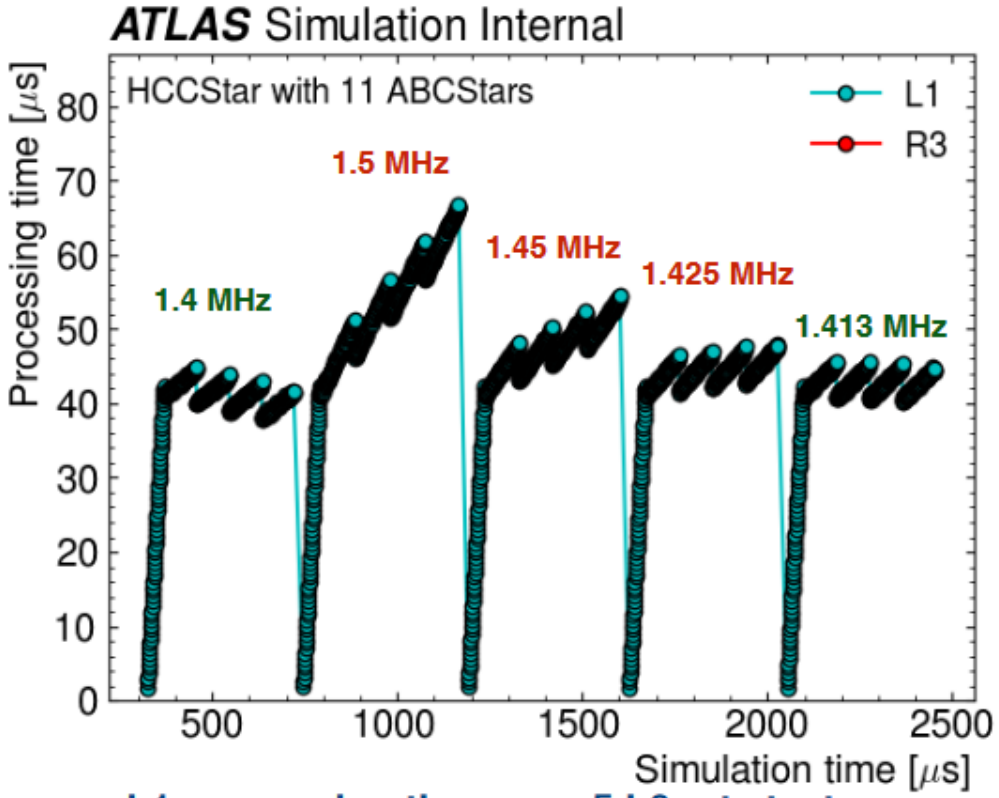




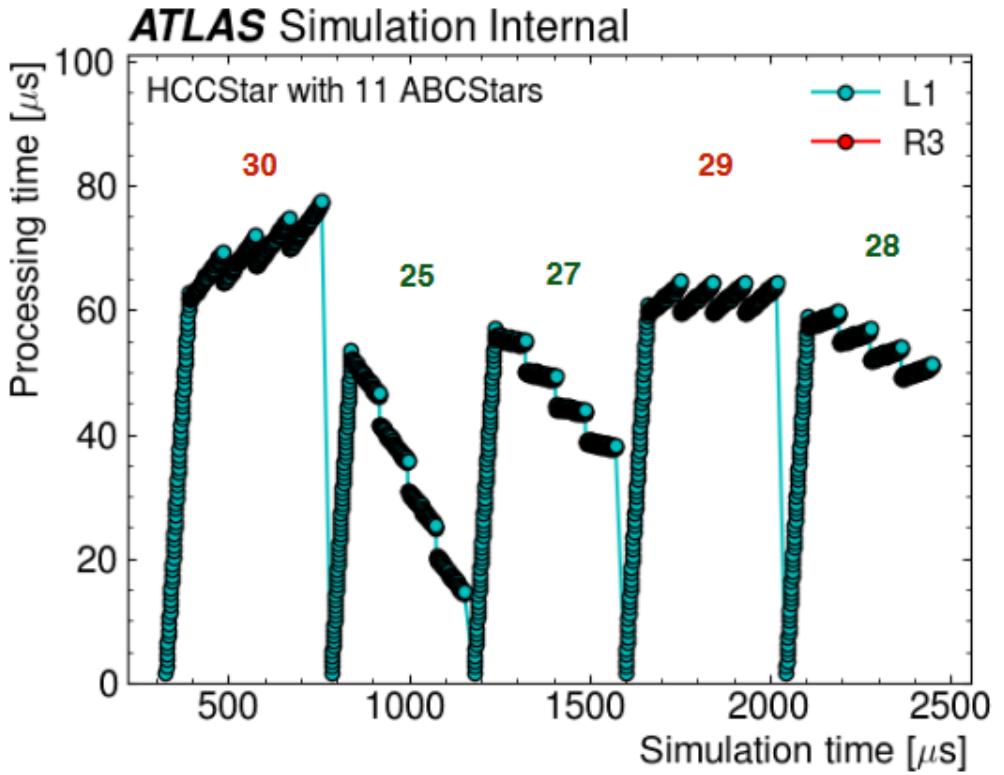
HCC Verification Example Code

```
@CocotbTest("HCC_401", skip=True)
def test_hcc_401_fake_unecoded_output(dut):
    """ Start in un-encoded mode and run some basic operations."""
    tester = hccstar.HCCStarTester(dut, out_encoding=0)
    tester.add_fake_abc(6)
    yield tester.start()

    tester.packetmon.set_automatic_packet_checking()

    # Reset and configure the chip.
    yield tester.configure_standalone()

    tester.enqueue_reg_write_hcc(0b0010, 40, 0x40)
    yield tester.lcbdrv.wait()
    yield triggers.ClockCycles(tester.clock, 100)

    # Send a L0A, then an ABC register read.
    tester.enqueue_L0A(1, 0b0100, bcr=False)
    yield tester.lcbdrv.wait()
    yield tester.get_parsed_packet('TYP_LP', ns_timeout=2500, fail=True)

    tester.enqueue_reg_read_abc(0b1111, 0b1111, 40)
    yield tester.lcbdrv.wait()
    yield tester.get_parsed_packet("TYP_ABC_RR", ns_timeout=2500, fail=True)
    …
```

HCC Verification Analysis Example



L1 processing time over 5 L0 rate tests



L1 processing time over 5 cluster occupancy tests

From HCCStar FDR presentation on Module simulation by J. Dandoy

HCC SEE Verification

Need to verification triplication strategy

Generate list of FF and LAT from synthesis tool

Generate list of wires with pyverilog (free Verilog parser)

Modify std cell library Verilog models to enable triggerable SEU

HCC Rapid SEU Test

Rapid SEU test

Inject SEU in a random FF every 2nd or 3rd clock

Verify triplicated FF all have the same state on the clock after the SEU injection

HCC SEU/SET Test

Inject an SEU in a random FF and look for unexpected behavior

Inject an (unphysically long) SET in a random wire and look for unexpected behavior

Read wire state, force opposite state and force back

SEE campaign targets

Type	Registers	Register SEEs	Wires	Wire SEEs
Triplicated	18378	1.8M	71533	7.15M
Non-Triplicated	24104	2.4M	178208	17.82M

Continuous Integration

Use gitlab runner on local machine to run RTL and postsynth verification on commits

Automated nightly and weekly verification runs (RTL, postsynth, pnr)

- HTML coverage reports using imc
- Text output

Coverage Top Level Summary Report, Instance-Based

User	bjr
Host	lxhiggs.hep.upenn.edu
Report Id	html_coverage
Report date	Sat 21 Aug 2021 01:29:40 EDT
Report options	-detail -html -metrics overall|block|expression|toggle|fsm -out html_coverage
Coverage database path	hccstar/verif/top/cov_work/scope/test/
Coverage model files	hccstar/verif/top/cov_work/scope/icc_0c94aca3_00000000.ucm
Coverage data files	hccstar/verif/top/cov_work/scope/test//icc_0c94aca3_00000000.ucd
Coverage database date	Sat 21 Aug 2021 01:29:12 EDT

Legend and Help

Refinement files applied before generating HTML report

CCF files applied before generating HTML report

Overall Instance-Based Coverage

Overall Average	Overall Covered	Block Average	Block Covered	Expression Average	Expression Covered	Toggle Average	Toggle Covered	Fsm Average	Fsm Covered	name
96.54%	96.08% (58216/60590/5041)	99.54%	97.19% (6049/6224/310)	94.11%	93.19% (1150/1234/52)	94.50%	96.02% (51017/53132/4679)	n/a	n/a	HCCStarWrapper

Coverage Color Legend

0	<25	<50	<75	<100	100	n/a	Not Scored

Conclusion

cocotb is a very capable verification framework

with a low barrier to entry

and brings with it a large ecosystem of tools