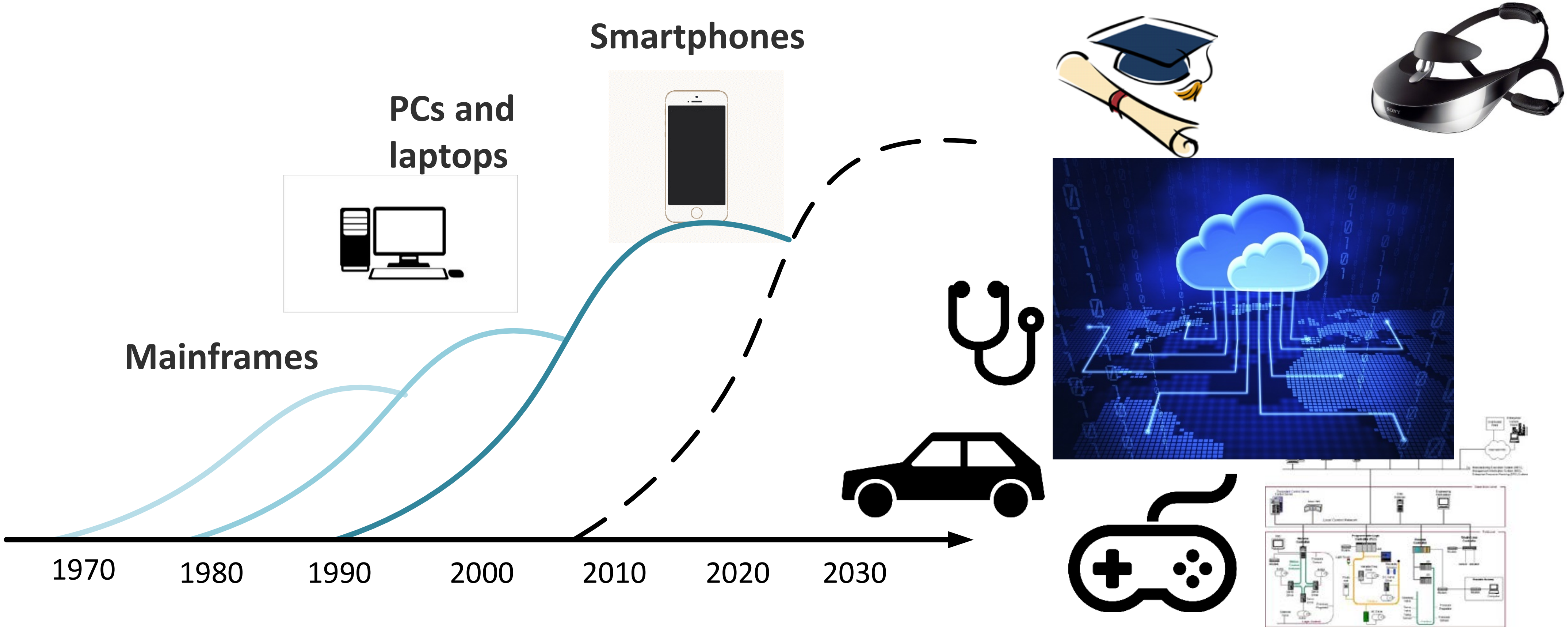
# Agile Hardware Design with a Generator-Based Methodology

Elad Alon

UC Berkeley

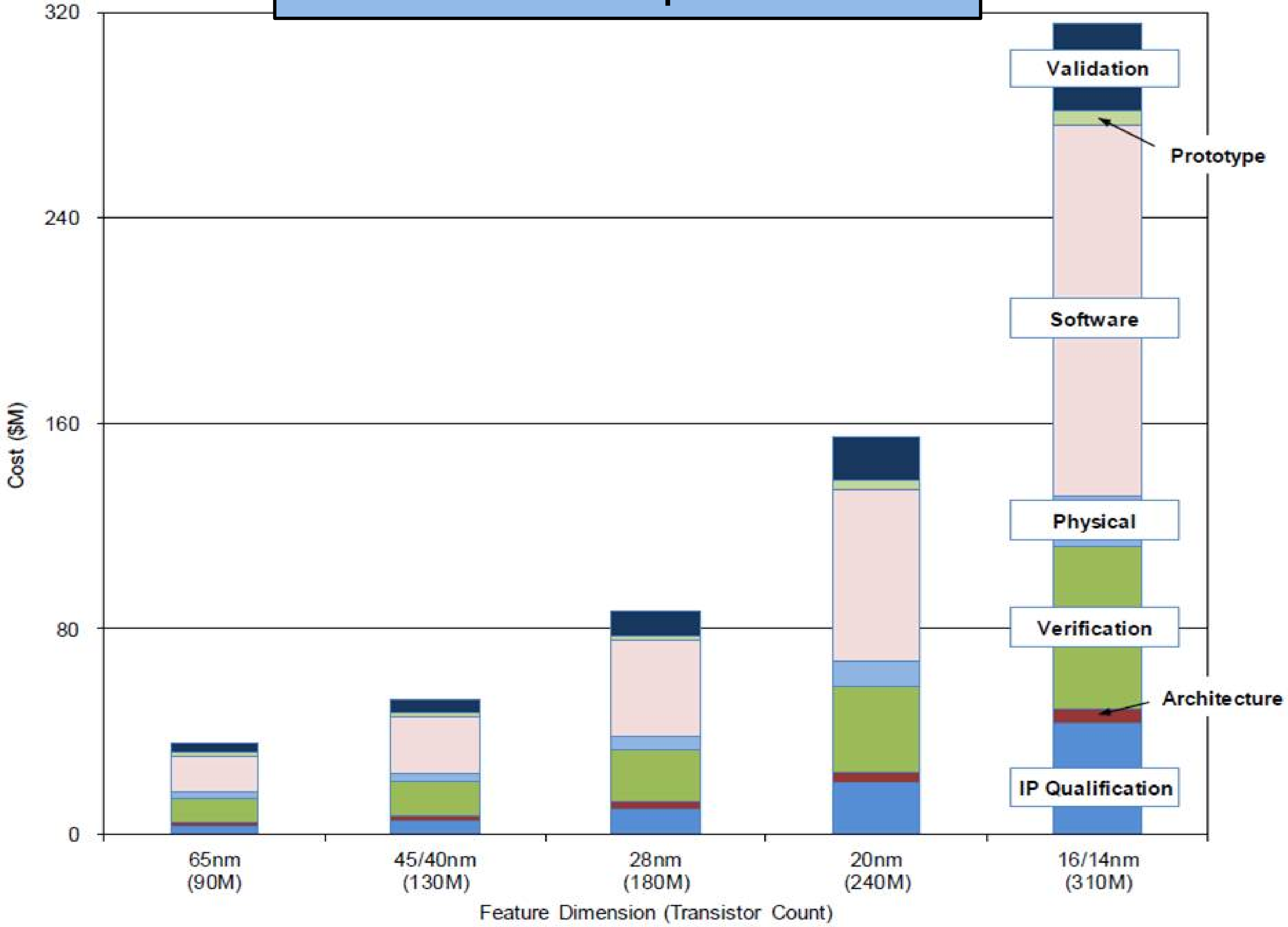
# Driving Applications Are Diversifying



- **Custom silicon needed for the cloud and a diverse set of applications with varying demands**

# ASICs Are Expensive

~$300M development cost

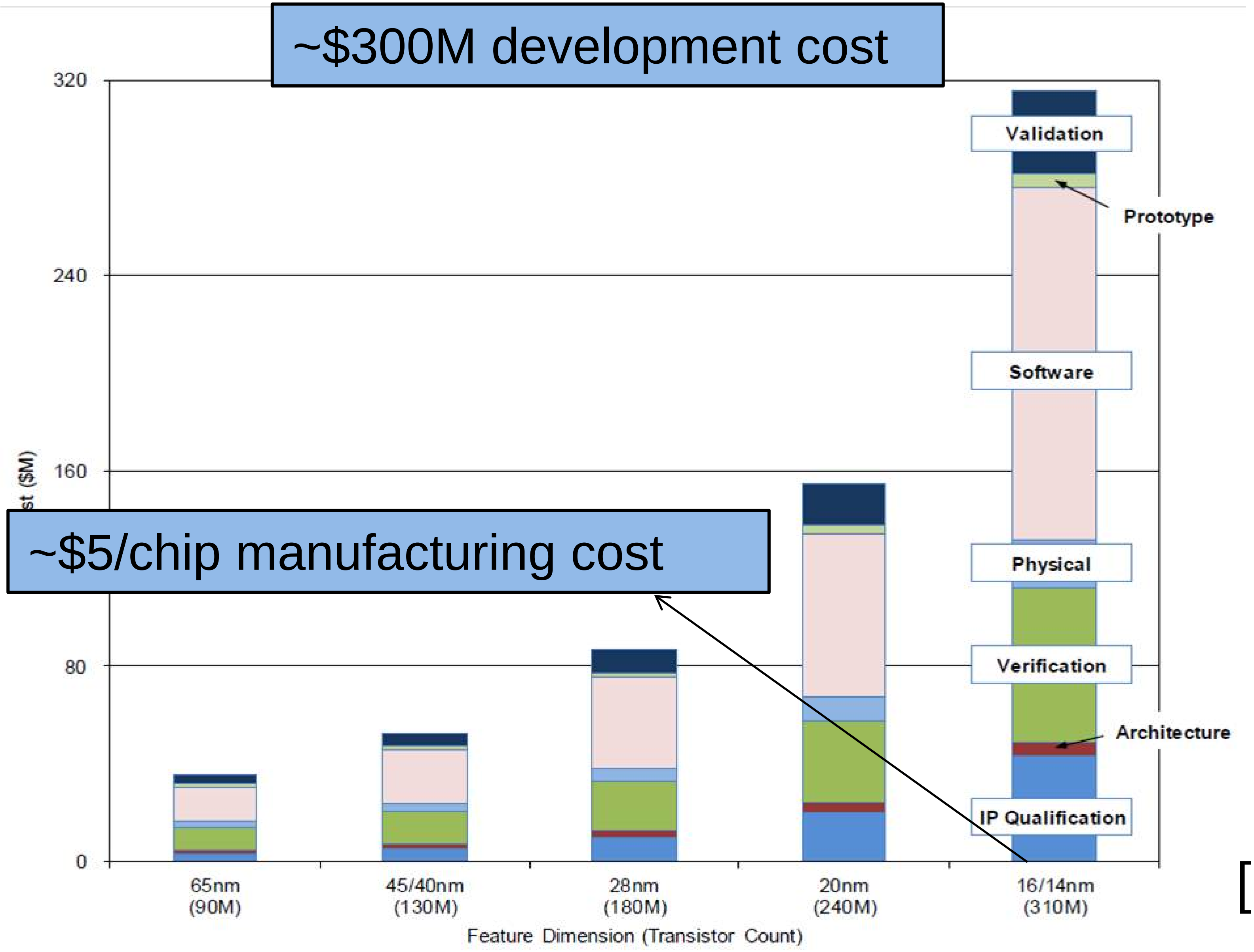


[Source: IBS]

# ASICs Are Expensive

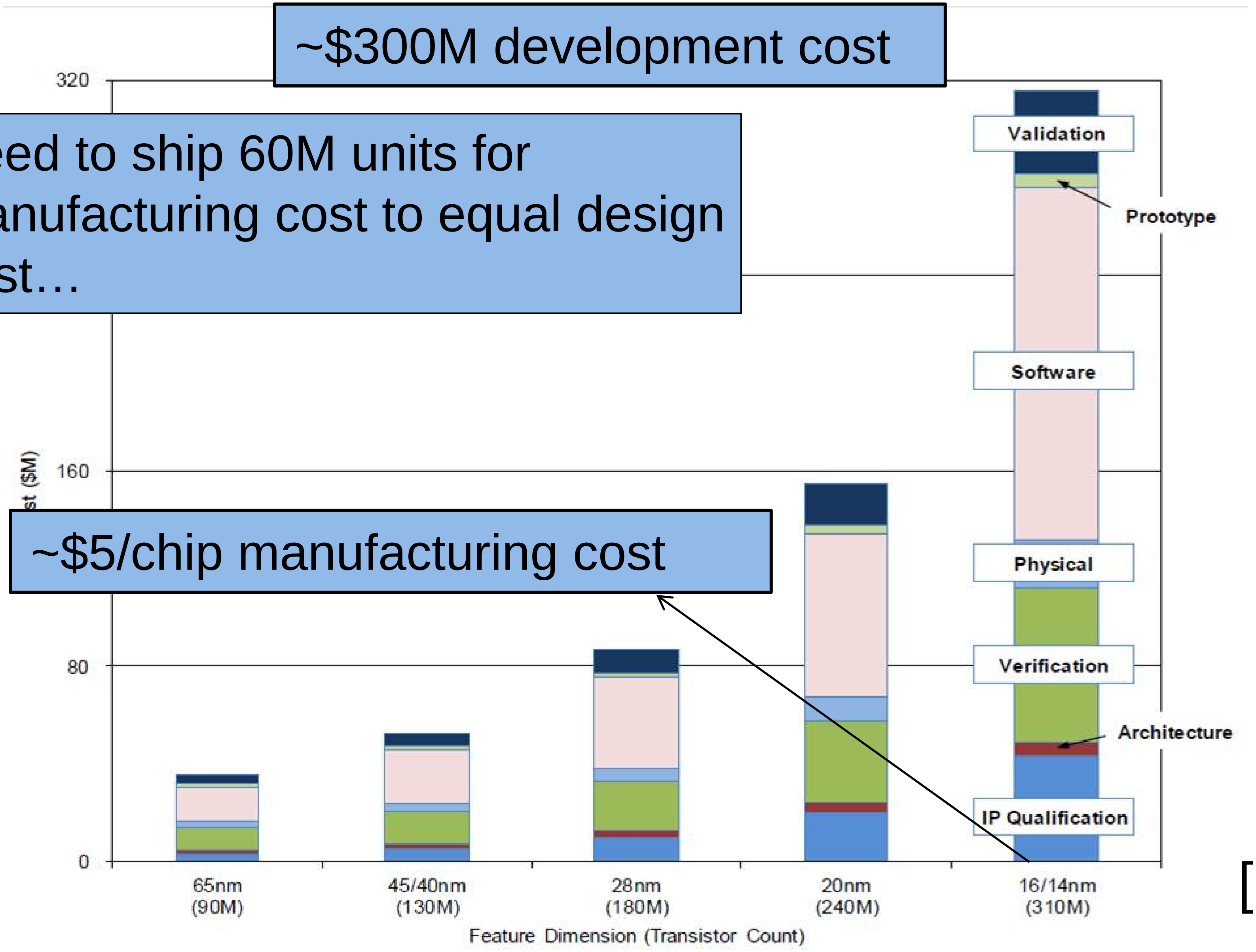
~$300M development cost

~$5/chip manufacturing cost



[Source: IBS]

# ASICs Are Expensive



~$300M development cost

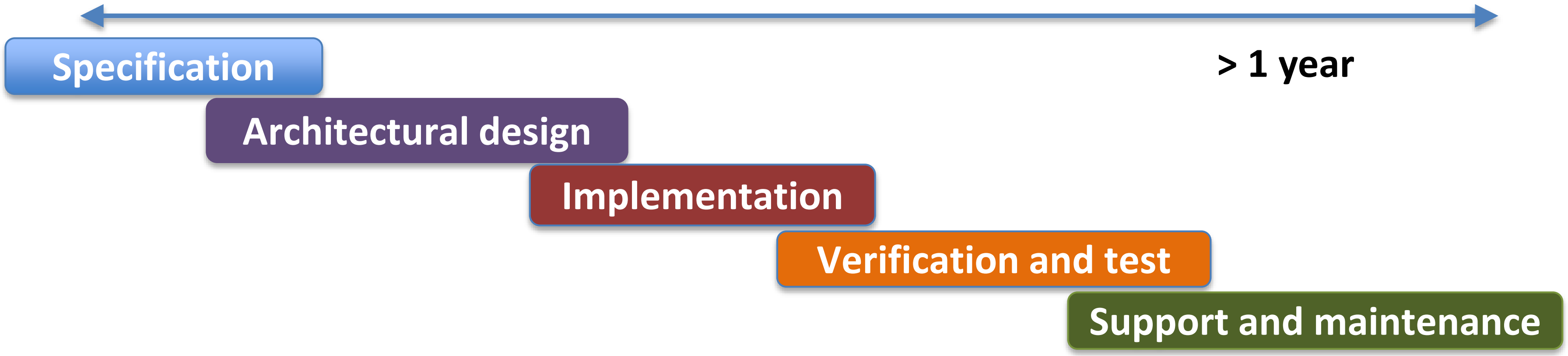
Need to ship 60M units for manufacturing cost to equal design cost…

~$5/chip manufacturing cost

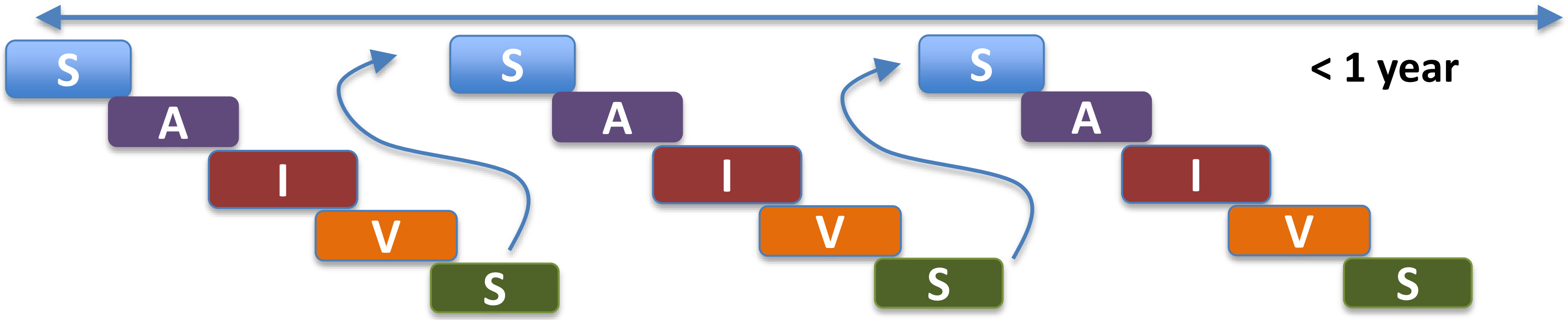
[Source: IBS]

# Software Had a Similar Problem

- “Waterfall” development:



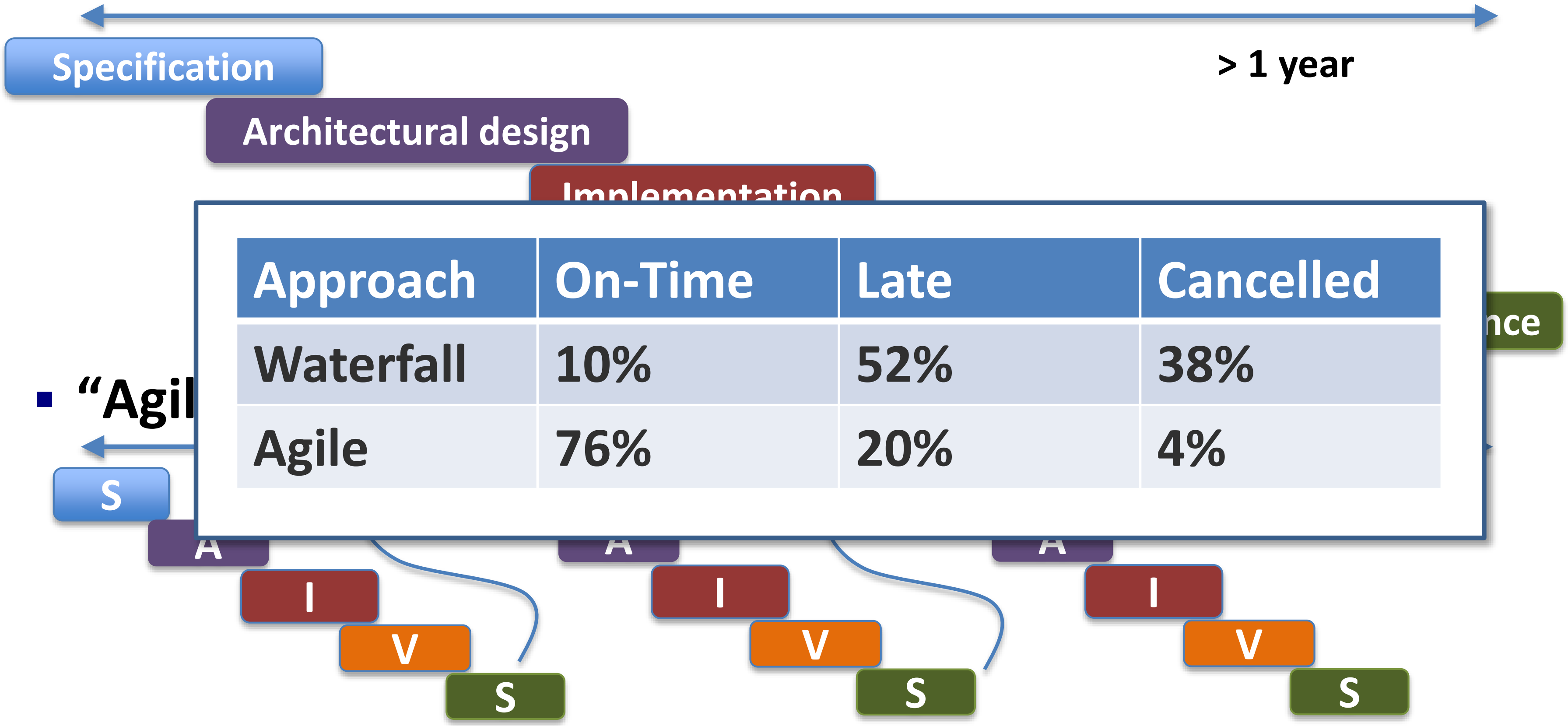
- “Agile” development:



Fox, Patterson, 2013.

# Software Had a Similar Problem

- “Waterfall” development:

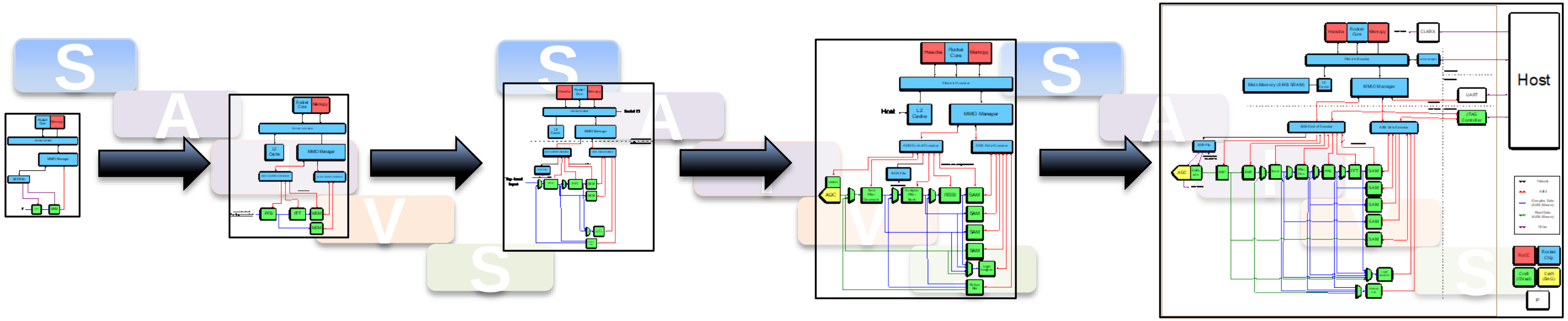


- “Agil

| Approach | On-Time | Late | Cancelled |
|---|---|---|---|
| Waterfall | 10% | 52% | 38% |
| Agile | 76% | 20% | 4% |

Fox, Patterson, 2013.

# Can Chip Design Be Agile?



- **Need methodologies and flows for:**
  - Scalable designs
  - Rapid design turn-around
  - Aggressive re-use
  - Agile verification and validation

# The Key Missing Piece

- **Dearth of re-use is the dominant problem**
  - Yes, lots of IP is out there
  - But that IP is largely “black-box” - hard to extend, modify, verify

- **Approach: don’t deliver instances – capture designer methodology in generators!**
  - Facilitates re-use via **parameterization** and **incremental extension** (of the generator – not the instance)

- **So how do we do this, and how well does it really work?**
  - Answering this question is the goal of our DARPA CRAFT team

# CRAFT Team

Elad Alon (PI), Borivoje Nikolic (co-PI),
Jonathan Bachrach, Koushik Sen

Berkeley
UNIVERSITY OF CALIFORNIA

Richard Berger

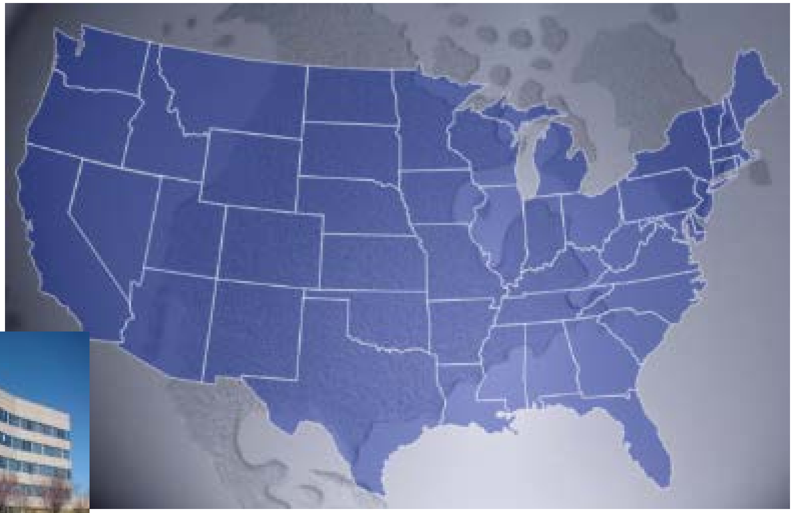
Silviu Chiricescu

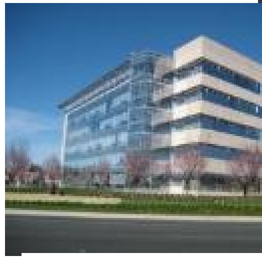
BAE SYSTEMS

Steven Shauck,

Matthew Doerflein

NORTHROP GRUMMAN





cādence™
Research Center

Mike Stellfox, Joseph Cole

cādence™
Design Center

Contains unpublished material – do not redistribute

# Outline

- Agile Design

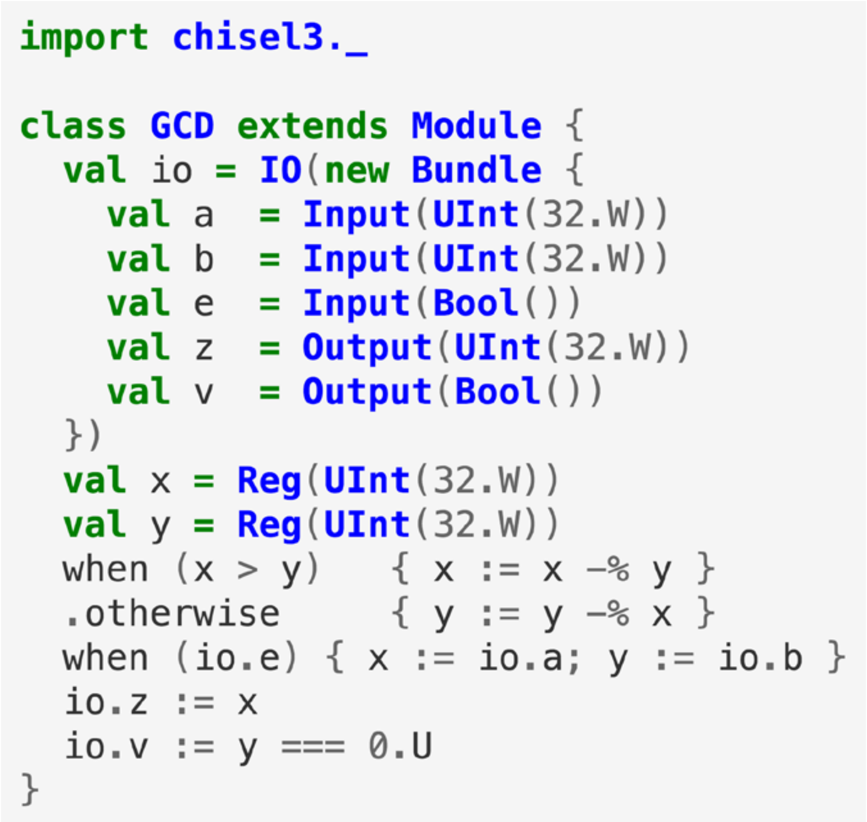
- **Generators for Digital: Chisel, FIRRTL, and Hammer**

- Generators for Analog: BAG
- Overall Flow + Verification
- Generators and SoCs
- Effort Data and Looking Forward

# Digital Generators: Chisel

- **Constructing Hardware In Scala Embedded Language**

- **Chisel is a Hardware Construction Language:**
  - Software library whose classes represent hardware primitives
  - Methods connect the classes together
  - So executing the software constructs a graph representing the RTL



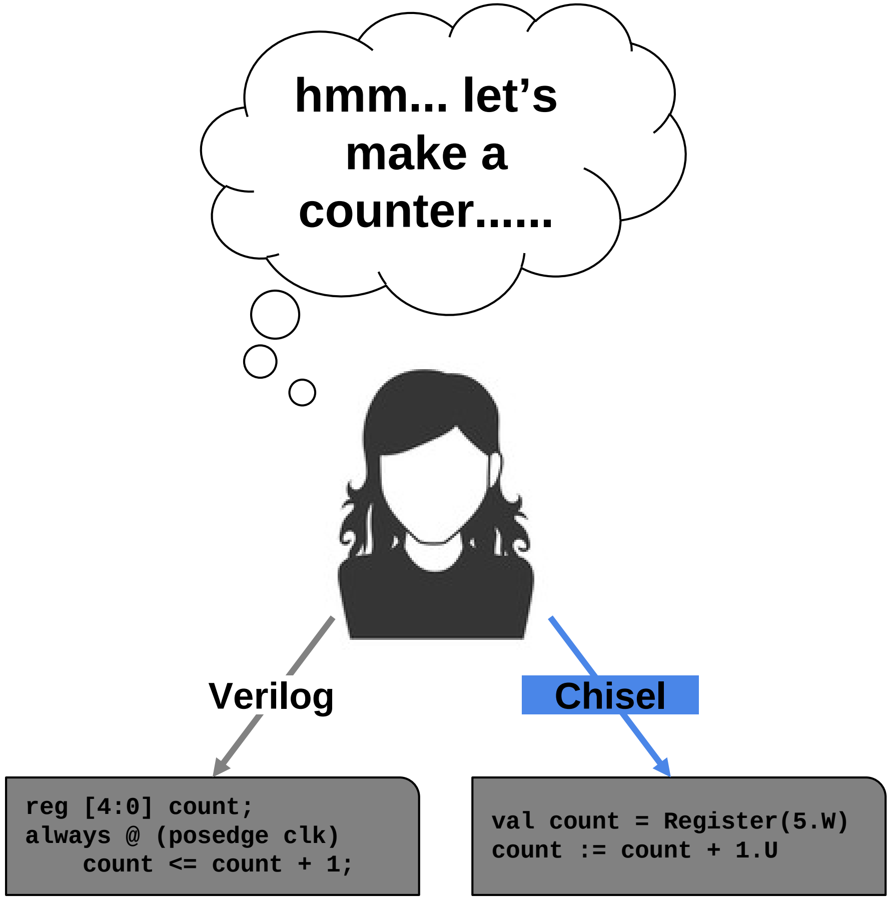
```
import chisel3._

class GCD extends Module {
  val io = IO(new Bundle {
    val a  = Input(UInt(32.W))
    val b  = Input(UInt(32.W))
    val e  = Input(Bool())
    val z  = Output(UInt(32.W))
    val v  = Output(Bool())
  })
  val x = Reg(UInt(32.W))
  val y = Reg(UInt(32.W))
  when (x > y)   { x := x -% y }
  .otherwise     { y := y -% x }
  when (io.e) { x := io.a; y := io.b }
  io.z := x
  io.v := y === 0.U
}
```

J. Bachrach et al., *DAC* 2012

# Chisel Provides Same Control as RTL

- **Same hardware abstraction level**
  - Compiler is NOT fancy
  - Very different from high-level synthesis

- **Higher software abstraction level**
  - Powerful parameterization, functional and object-oriented programming, static typing
  - **Huge base of existing software libraries**



hmm... let's make a counter......

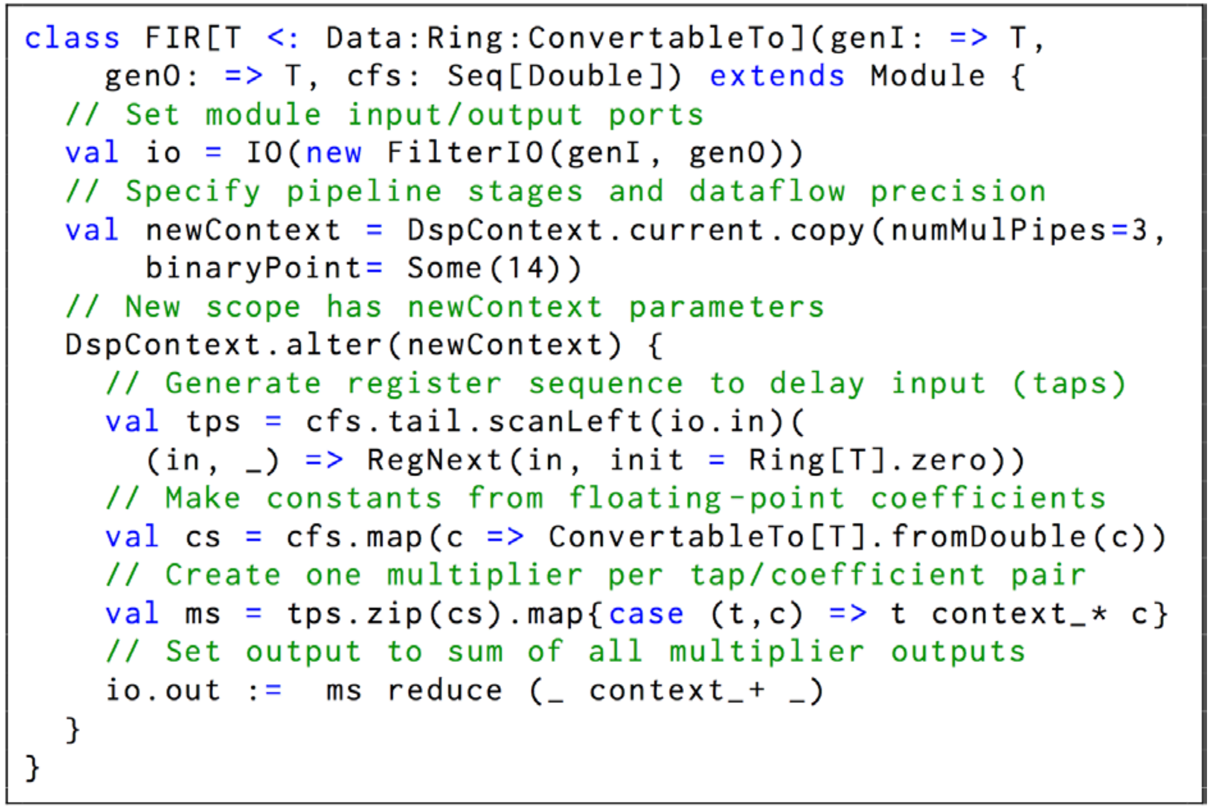
Verilog

```
reg [4:0] count;
always @ (posedge clk)
    count <= count + 1;
```

Chisel

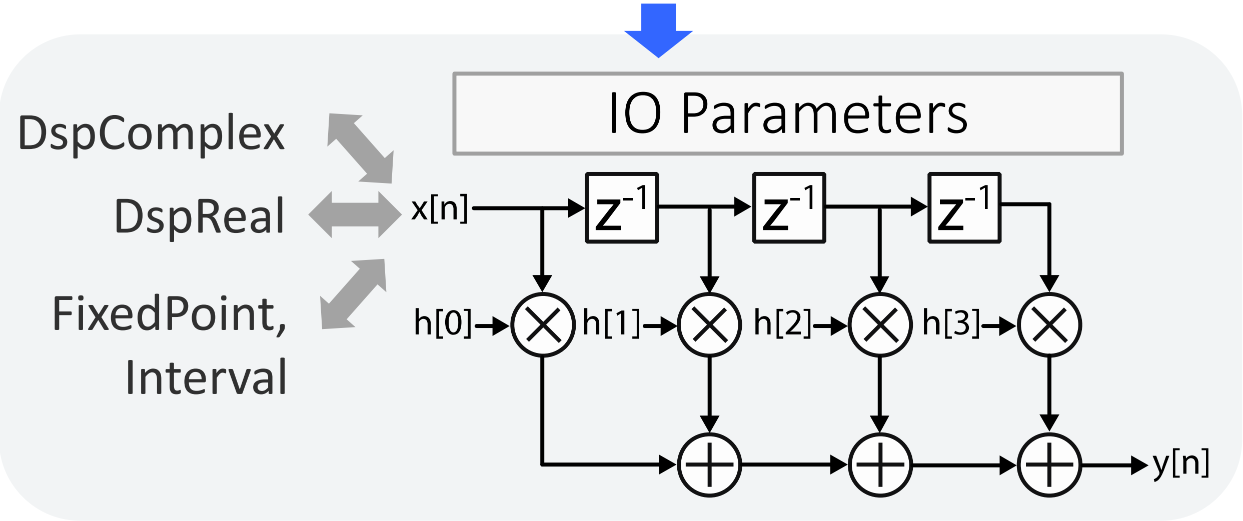
```
val count = Register(5.W)
count := count + 1.U
```

# ChiselDSP



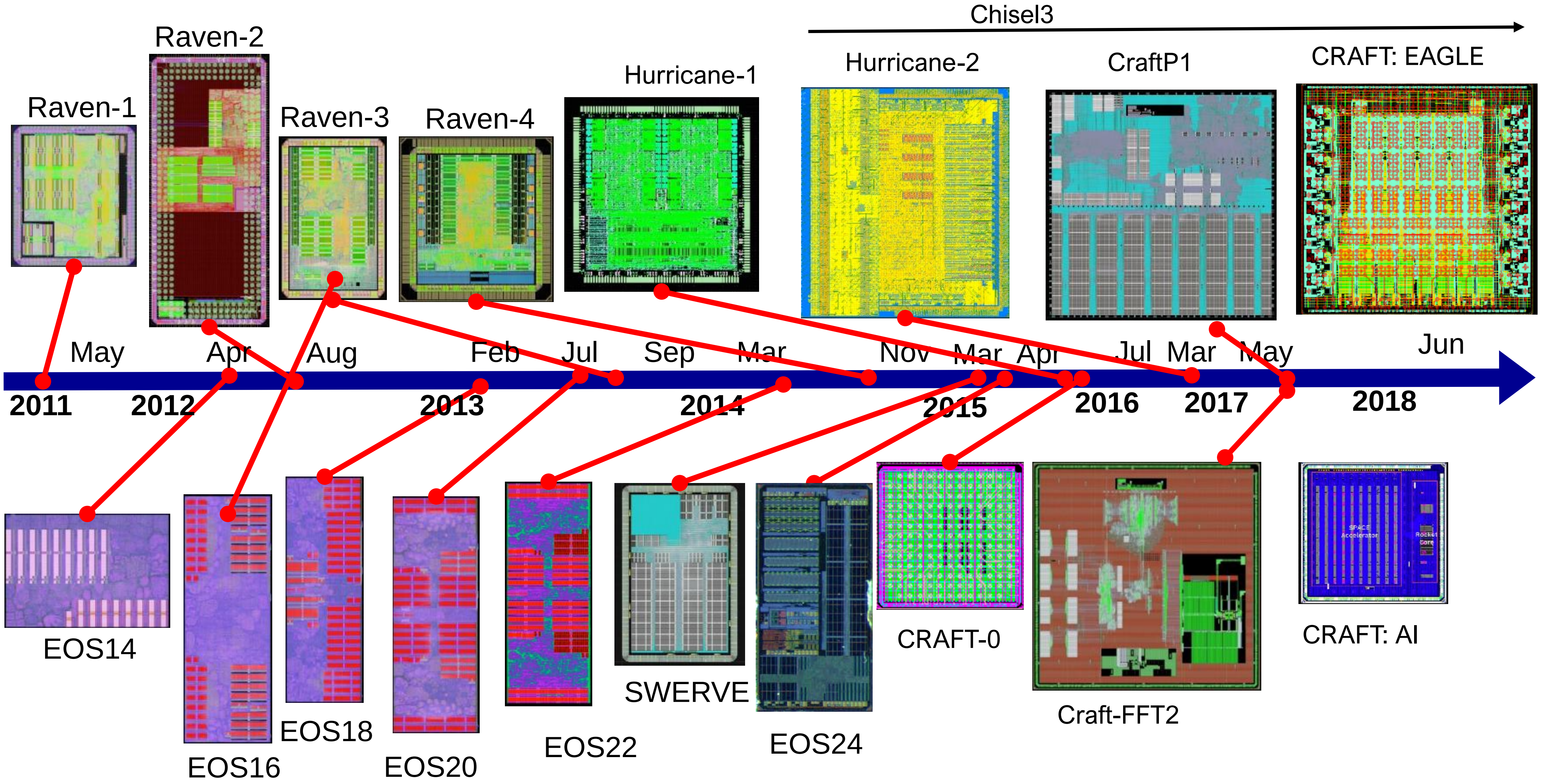
```
class FIR[T <: Data:Ring:ConvertableTo](genI: => T,
    genO: => T, cfs: Seq[Double]) extends Module {
  // Set module input/output ports
  val io = IO(new FilterIO(genI, genO))
  // Specify pipeline stages and dataflow precision
  val newContext = DspContext.current.copy(numMulPipes=3,
      binaryPoint= Some(14))
  // New scope has newContext parameters
  DspContext.alter(newContext) {
    // Generate register sequence to delay input (taps)
    val tps = cfs.tail.scanLeft(io.in)(
      (in, _) => RegNext(in, init = Ring[T].zero))
    // Make constants from floating-point coefficients
    val cs = cfs.map(c => ConvertableTo[T].fromDouble(c))
    // Create one multiplier per tap/coefficient pair
    val ms = tps.zip(cs).map{case (t,c) => t context_* c}
    // Set output to sum of all multiplier outputs
    io.out :=  ms reduce (_ context_+ _)
  }
}
```

$$y[n] = \sum_{k=0}^{N} h[k]x[n-k]$$



- **Supports number representation agnostic generator design**
  - Datatypes and associated operators can real/complex, fixed/floating-point without rewriting any of the core generator code

# Chisel-Designed Chips



Chisel3

Raven-2

Hurricane-1

Hurricane-2

CraftP1

CRAFT: EAGLE

Raven-1

Raven-3

Raven-4

May Apr Aug Feb Jul Sep Mar Nov Mar Apr Jul Mar May Jun

2011 2012 2013 2014 2015 2016 2017 2018

EOS14

EOS16

EOS18

EOS20

EOS22

SWERVE

EOS24

CRAFT-0

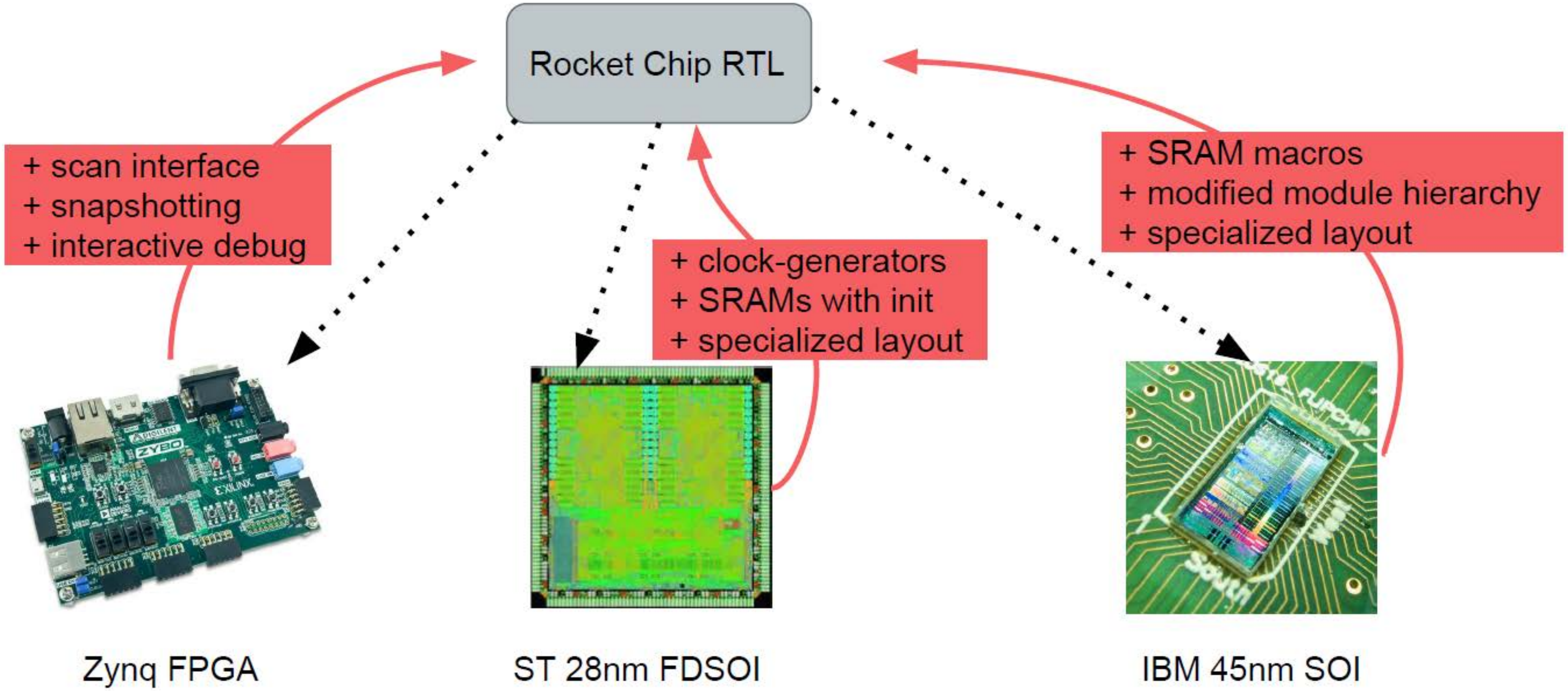
Craft-FFT2

CRAFT: AI

Raven, Hurricane: ST 28nm FDSOI, SWERVE: TSMC 28nm EOS: IBM 45nm SOI, CRAFT: 16nm TSMC

# Important Note About Reuse

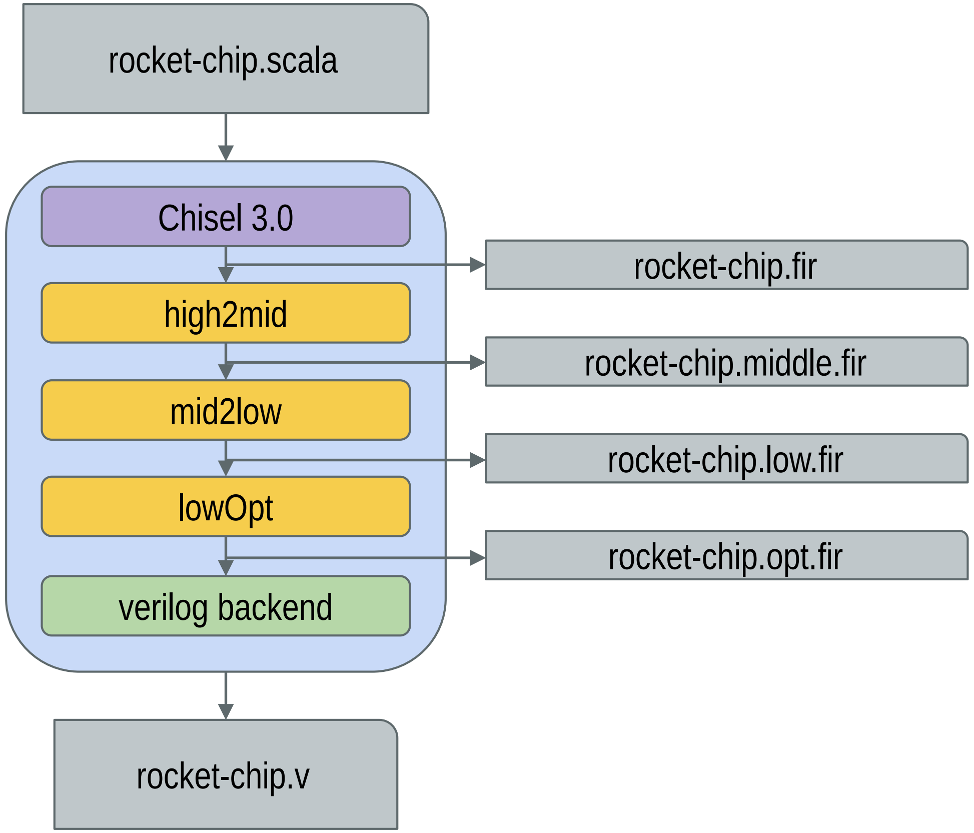
- **Downstream constraints may require changes to RTL**



- **Separating concerns between RTL and platform-specific optimizations maximizes reuse**

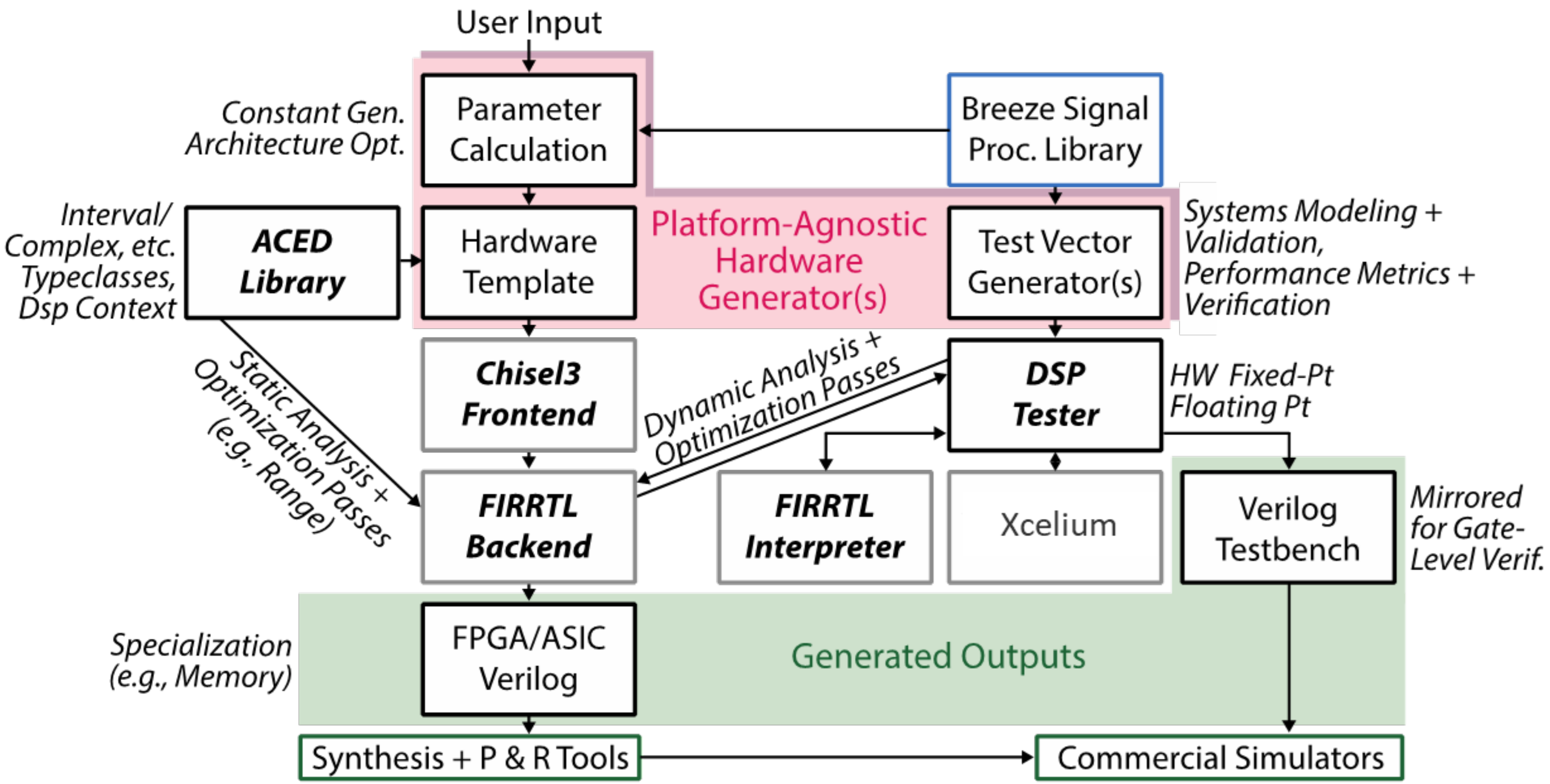
# Borrow From Software Again

- **FIRRTL: Flexible IR for RTL**
  - Basically “LLVM for hardware”
- **Frontend parser translates RTL source into IR**
- **Transformations on IR enable project- or platform-specific changes**
  - Without altering the original RTL
  - Transformations are composable
- **Backend emits the final design**



A. Izraelevitz et al., *ICCAD* 2017

# A Chisel Environment for DSP Generator Design



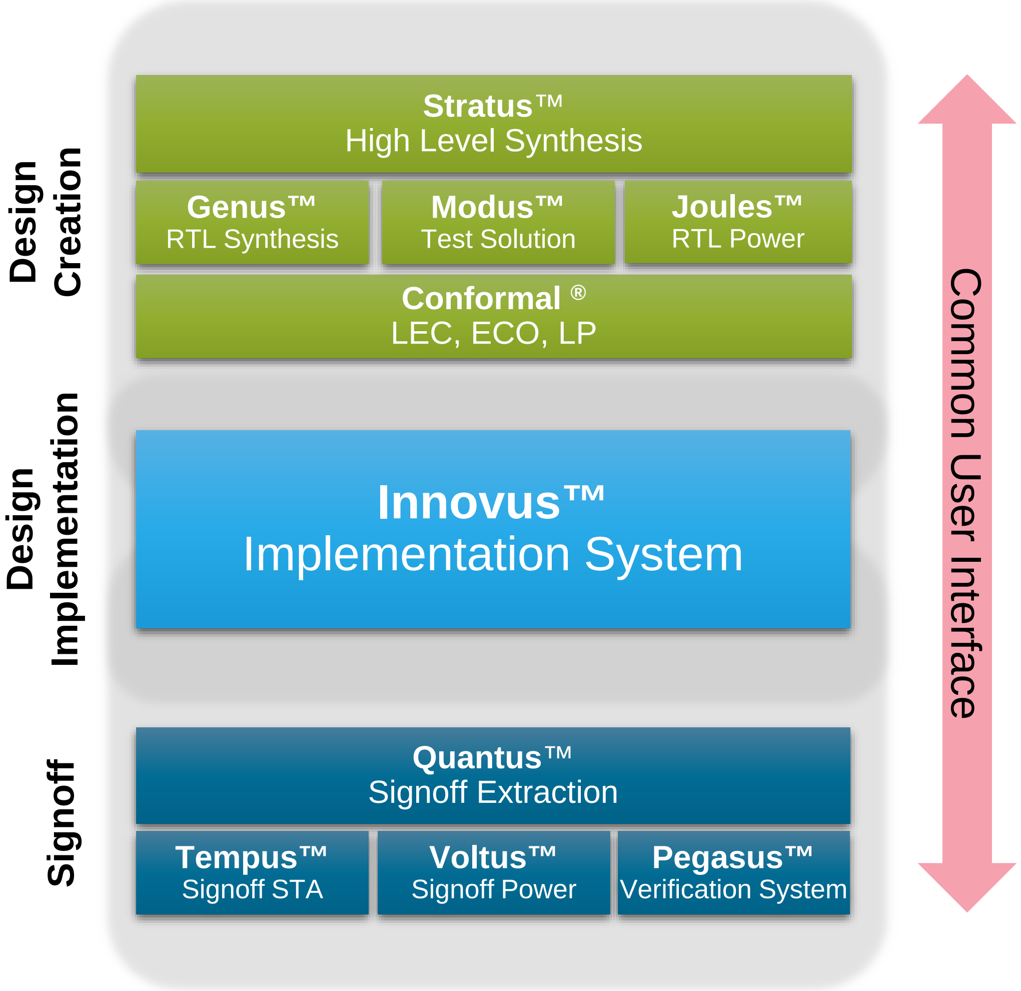
User Input
Constant Gen.
Architecture Opt.
Parameter Calculation
Breeze Signal Proc. Library
Interval/ Complex, etc. Typeclasses, Dsp Context
ACED Library
Hardware Template
Platform-Agnostic Hardware Generator(s)
Test Vector Generator(s)
Systems Modeling + Validation, Performance Metrics + Verification
Chisel3 Frontend
Dynamic Analysis + Optimization Passes
DSP Tester
HW Fixed-Pt Floating Pt
Static Analysis + Optimization Passes (e.g., Range)
FIRRTL Backend
FIRRTL Interpreter
Xcelium
Verilog Testbench
Mirrored for Gate-Level Verif.
Specialization (e.g., Memory)
FPGA/ASIC Verilog
Generated Outputs
Synthesis + P & R Tools
Commercial Simulators

- **Combination of FIRRTL and ChiselDSP capabilities enable single, unified environment for algorithm and hardware design as well as development**

A. Wang et al., *DAC* 2018

# Digital Physical Design

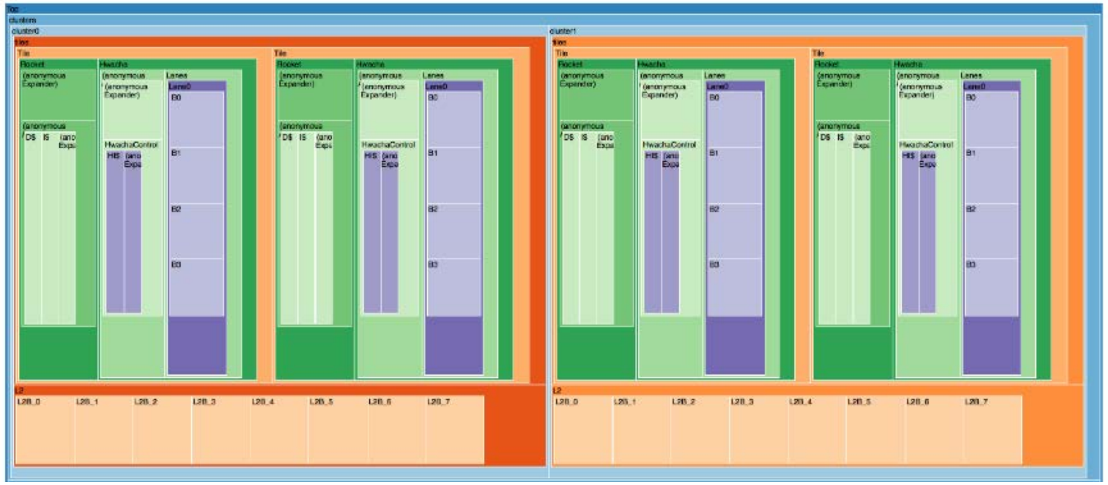
- **CAD tools highly automated and very powerful**
  - And Cadence’s recently introduced common user interface further streamlines RTL to GDS flow by enabling a single environment across tools

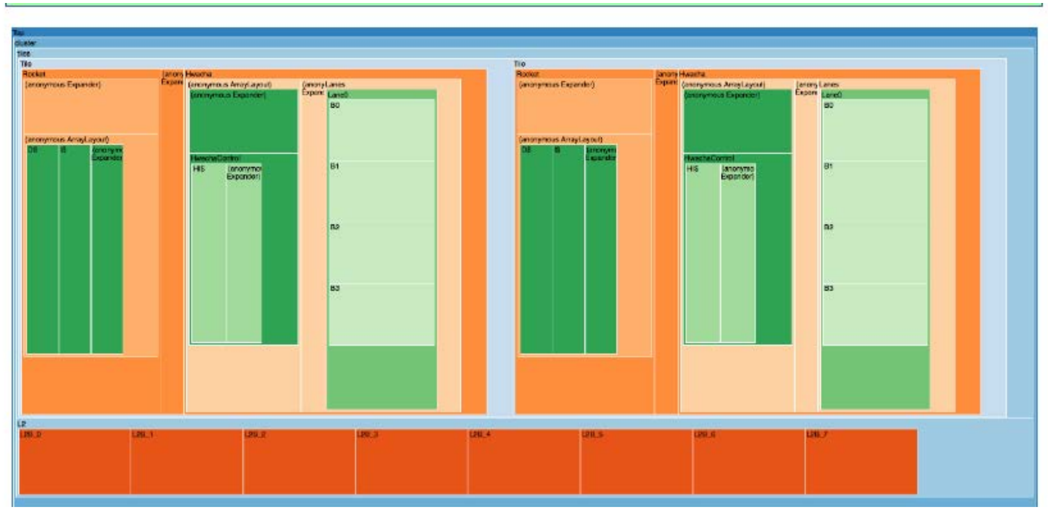


# Generators for Physical Design

- **Nonetheless, designer knowledge/expertise still critical**
  - Follow the same basic approach and capture expert designers' methodologies as executable code

- **Hammer is a newly developed framework to enable this**
  - Example floorplans generated by Hammer:

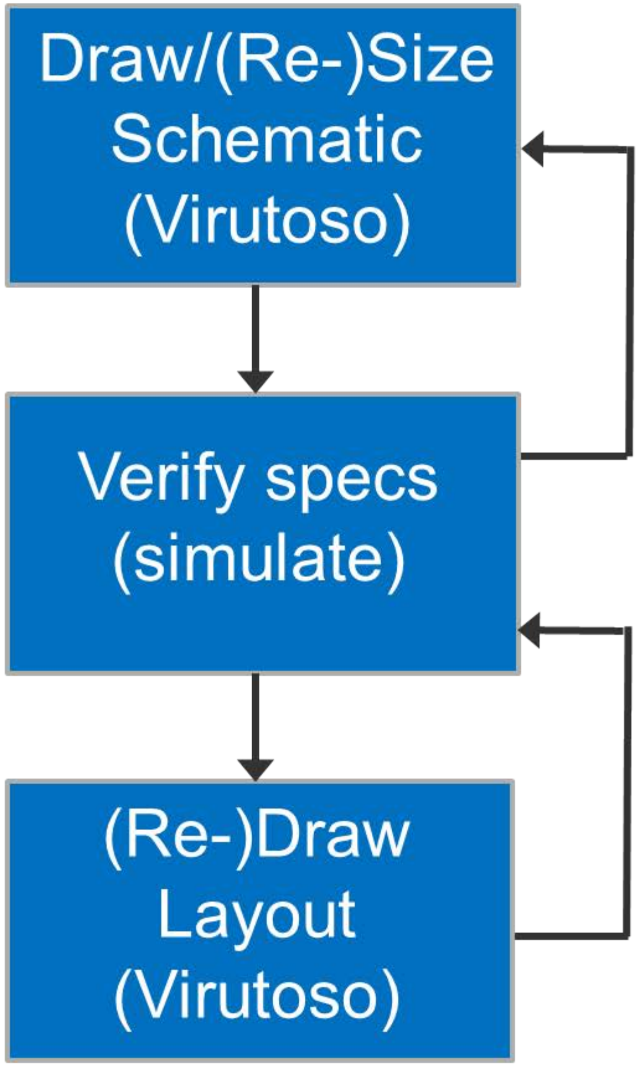




# Outline

- Agile Design
- Generators for Digital: Chisel, FIRRTL, and Hammer
- Generators for Analog: BAG
- Overall Flow + Verification
- Generators and SoCs
- Effort Data and Looking Forward

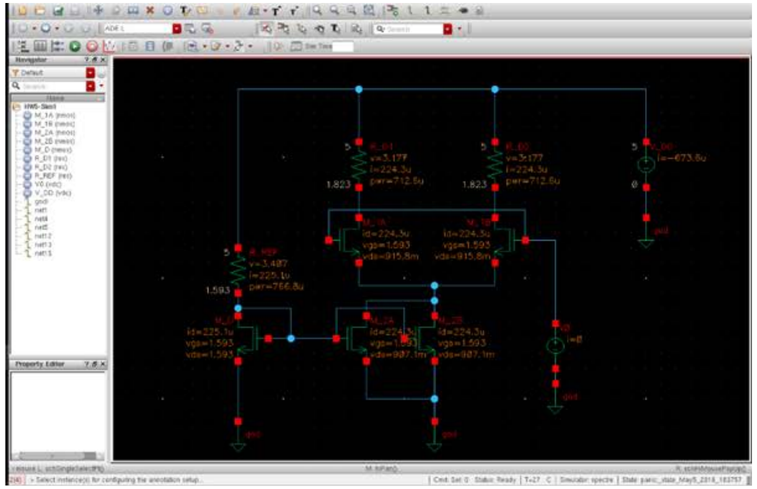
# Core Analog Design Loop

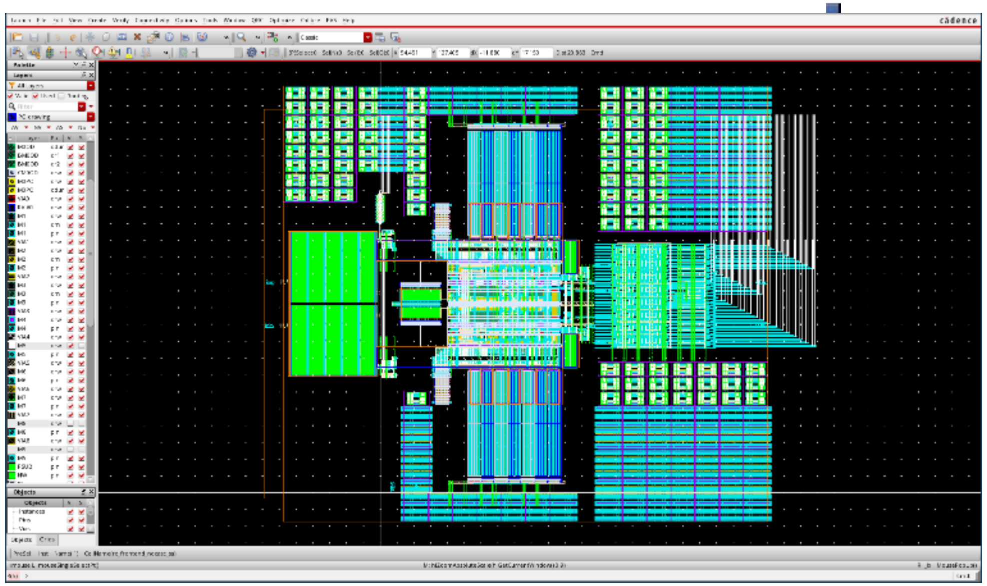


Draw/(Re-)Size Schematic (Virutoso)

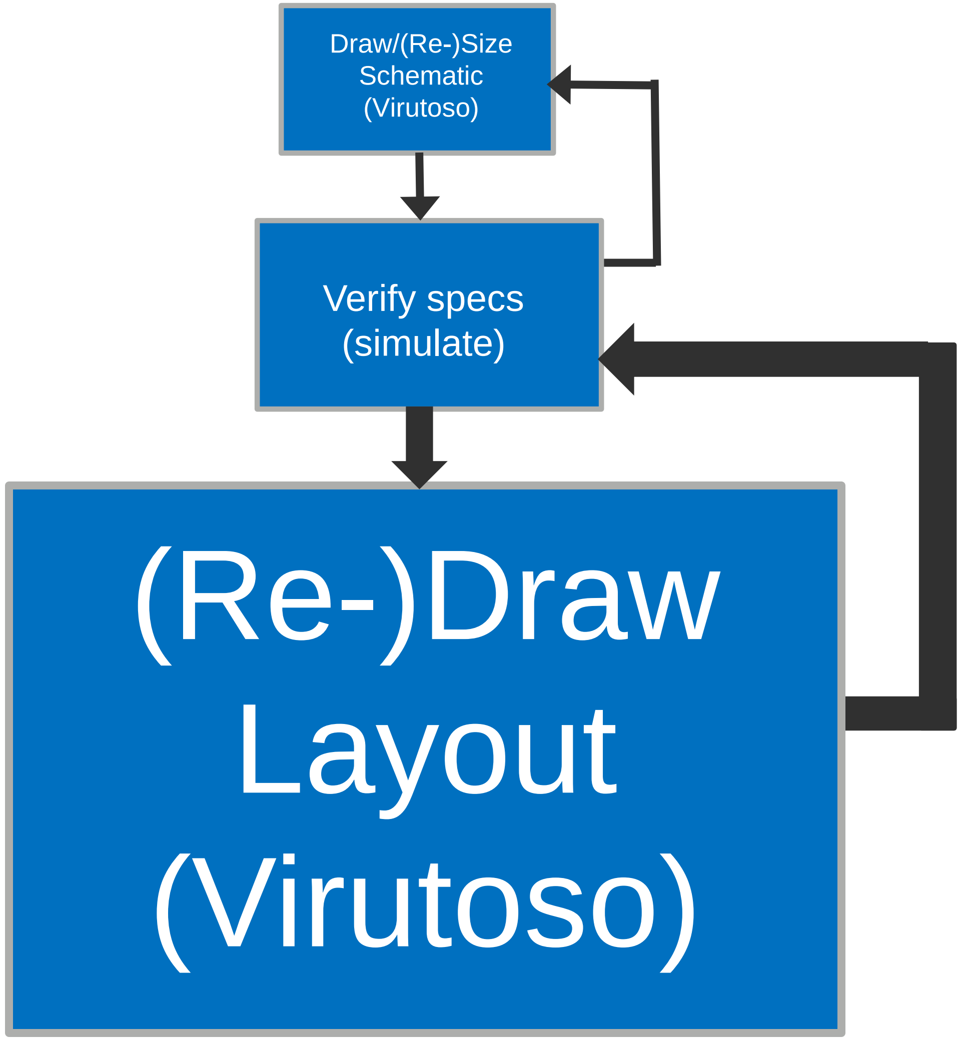
Verify specs (simulate)

(Re-)Draw Layout (Virutoso)





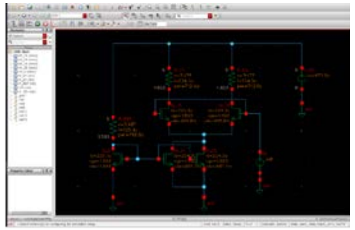
# A More Representative Depiction…

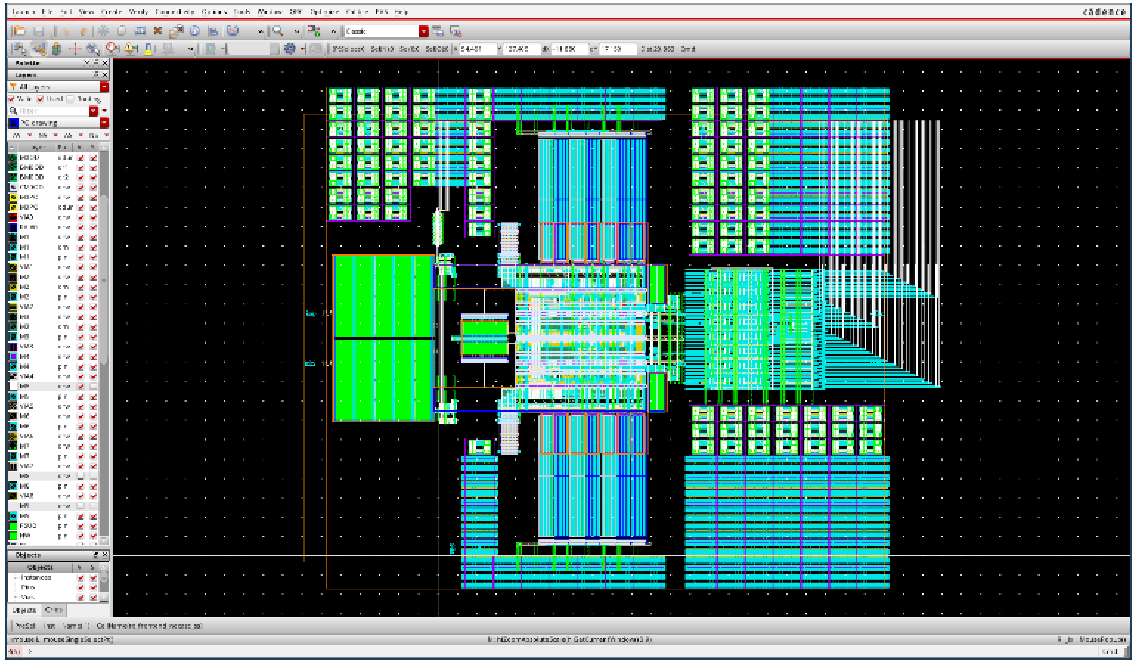


Draw/(Re-)Size Schematic (Virutoso)

Verify specs (simulate)

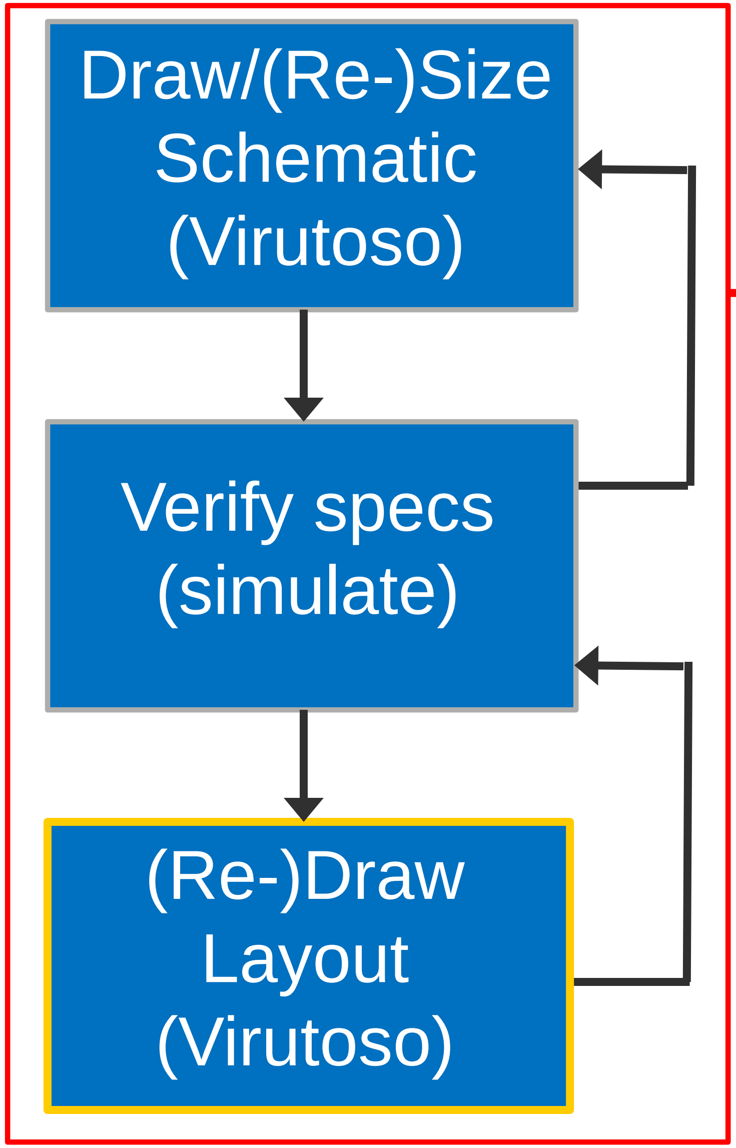
(Re-)Draw Layout (Virutoso)





# Reminder: Proposed Approach

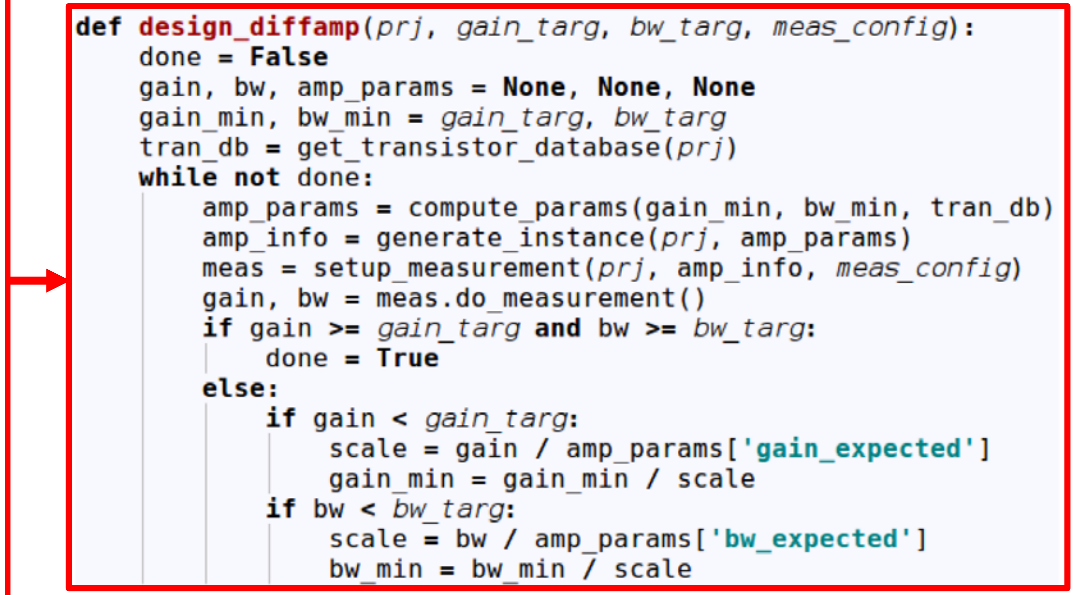
- **Reuse the core analog design loop itself**
  - Not the results of it



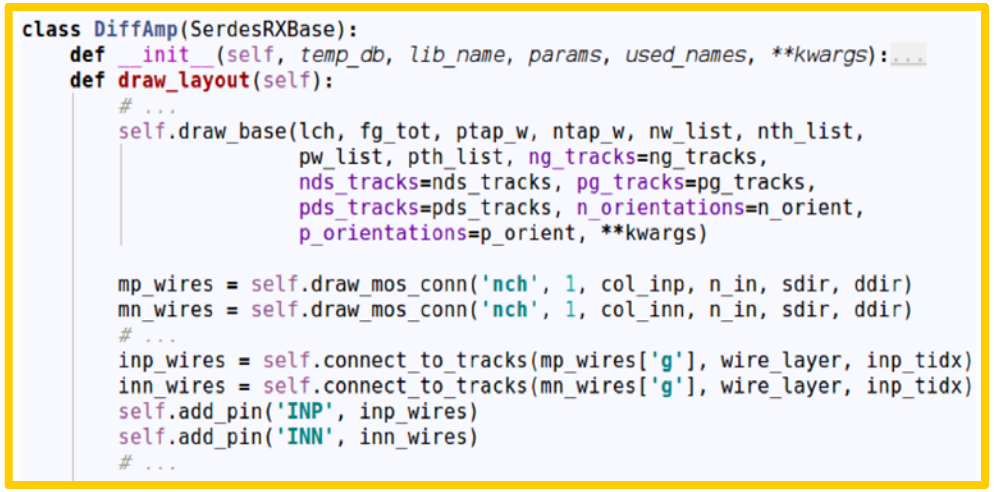
Draw/(Re-)Size Schematic (Virutoso)

Verify specs (simulate)

(Re-)Draw Layout (Virutoso)



```python
def design_diffamp(prj, gain_targ, bw_targ, meas_config):
    done = False
    gain, bw, amp_params = None, None, None
    gain_min, bw_min = gain_targ, bw_targ
    tran_db = get_transistor_database(prj)
    while not done:
        amp_params = compute_params(gain_min, bw_min, tran_db)
        amp_info = generate_instance(prj, amp_params)
        meas = setup_measurement(prj, amp_info, meas_config)
        gain, bw = meas.do_measurement()
        if gain >= gain_targ and bw >= bw_targ:
            done = True
        else:
            if gain < gain_targ:
                scale = gain / amp_params['gain_expected']
                gain_min = gain_min / scale
            if bw < bw_targ:
                scale = bw / amp_params['bw_expected']
                bw_min = bw_min / scale
```

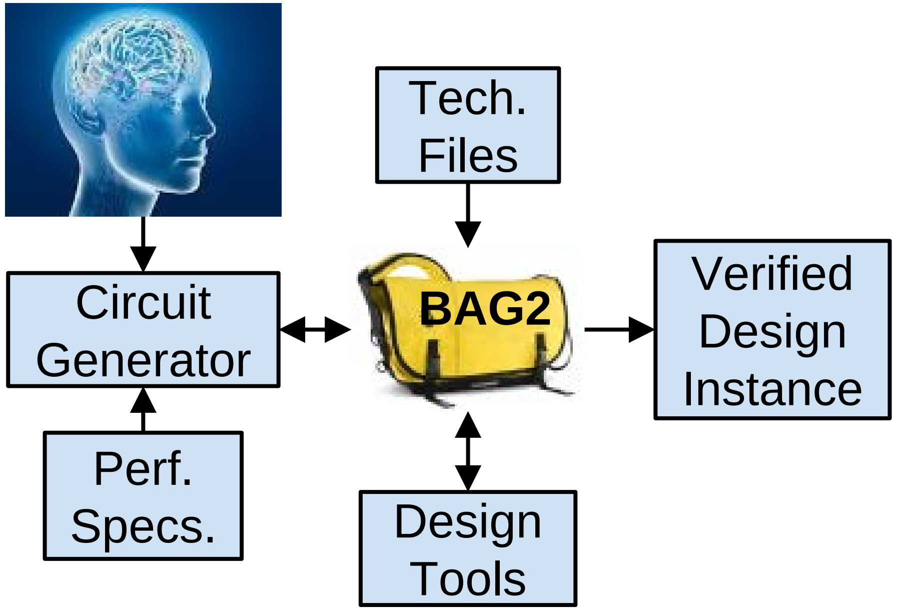


```python
class DiffAmp(SerdesRXBase):
    def __init__(self, temp_db, lib_name, params, used_names, **kwargs):...
    def draw_layout(self):
        # ...
        self.draw_base(lch, fg_tot, ptap_w, ntap_w, nw_list, nth_list,
                       pw_list, pth_list, ng_tracks=ng_tracks,
                       nds_tracks=nds_tracks, pg_tracks=pg_tracks,
                       pds_tracks=pds_tracks, n_orientations=n_orient,
                       p_orientations=p_orient, **kwargs)

        mp_wires = self.draw_mos_conn('nch', 1, col_inp, n_in, sdir, ddir)
        mn_wires = self.draw_mos_conn('nch', 1, col_inn, n_in, sdir, ddir)
        # ...
        inp_wires = self.connect_to_tracks(mp_wires['g'], wire_layer, inp_tidx)
        inn_wires = self.connect_to_tracks(mn_wires['g'], wire_layer, inp_tidx)
        self.add_pin('INP', inp_wires)
        self.add_pin('INN', inn_wires)
        # ...
```

# Berkeley Analog Generator (BAG)

- **Open-source Python-based framework allowing executable specification of design procedure**

- **I.e., BAG takes care of the “plumbing”**
  - BAG’s Python scripts interact directly with user’s Virtuoso instance

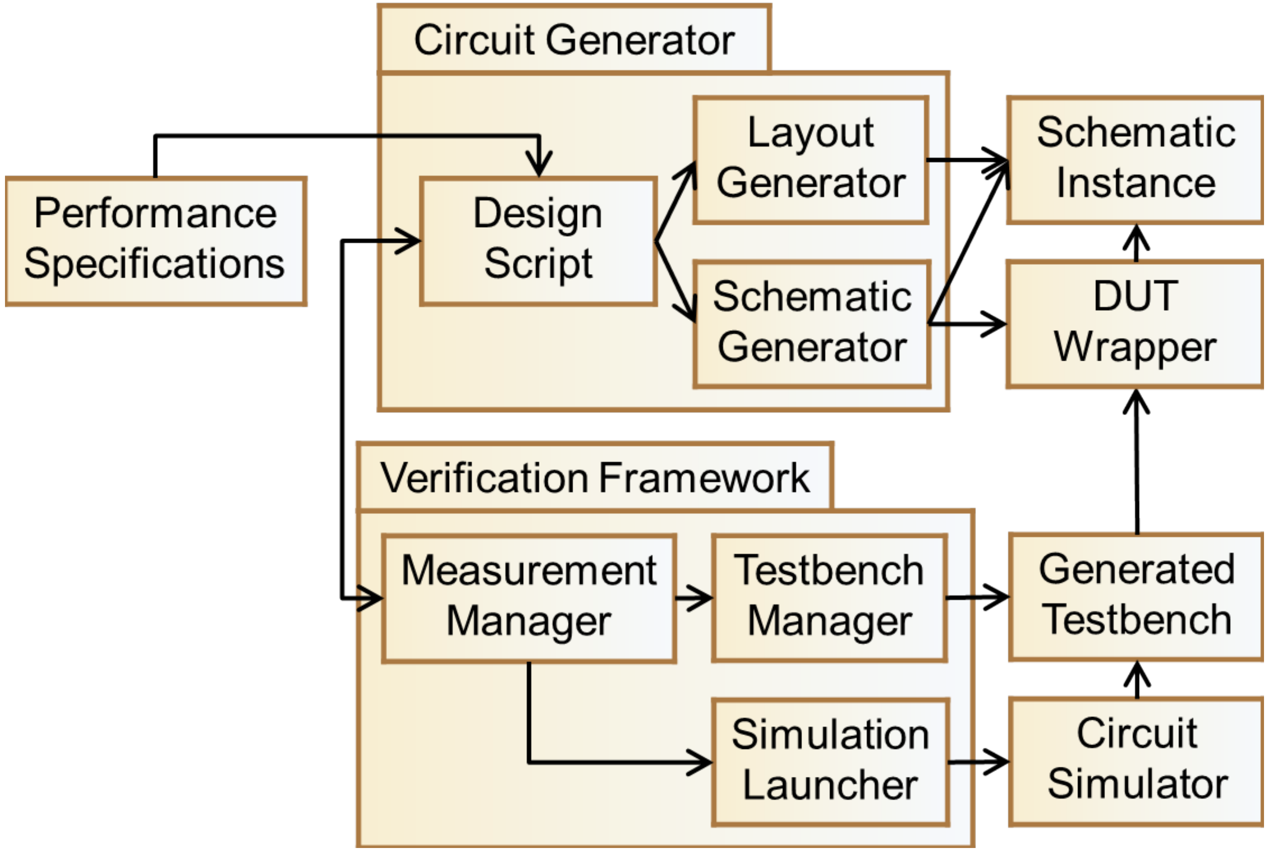


J. Crossley et al., *ICCAD* 2013
E. Chang et al., CICC 2018

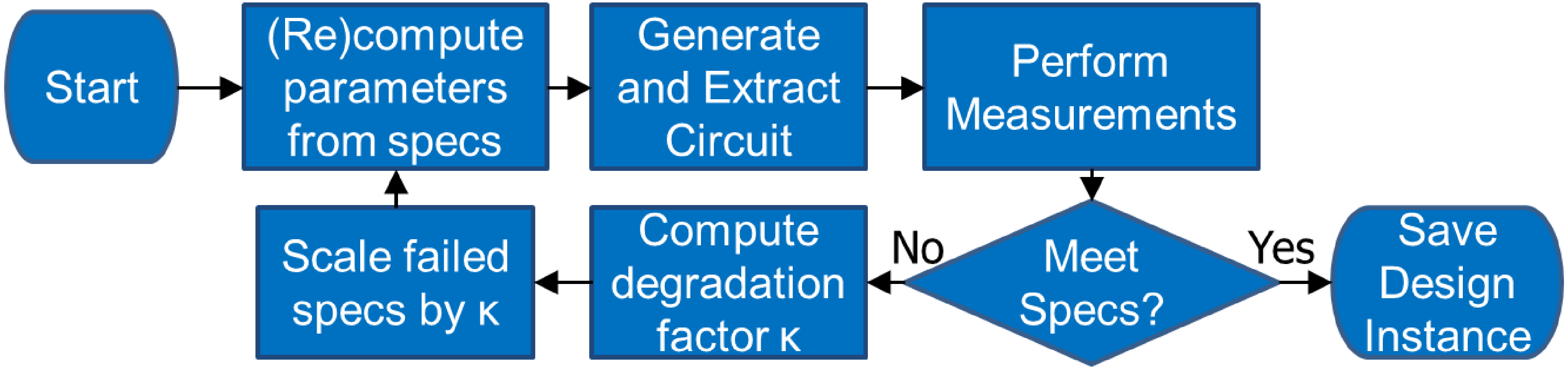
# Full Generator Overview

- **Schematic/layout generators produce actual views based on low-level structural parameters**

- **Design script captures algorithm that translates input specifications to structural parameters**

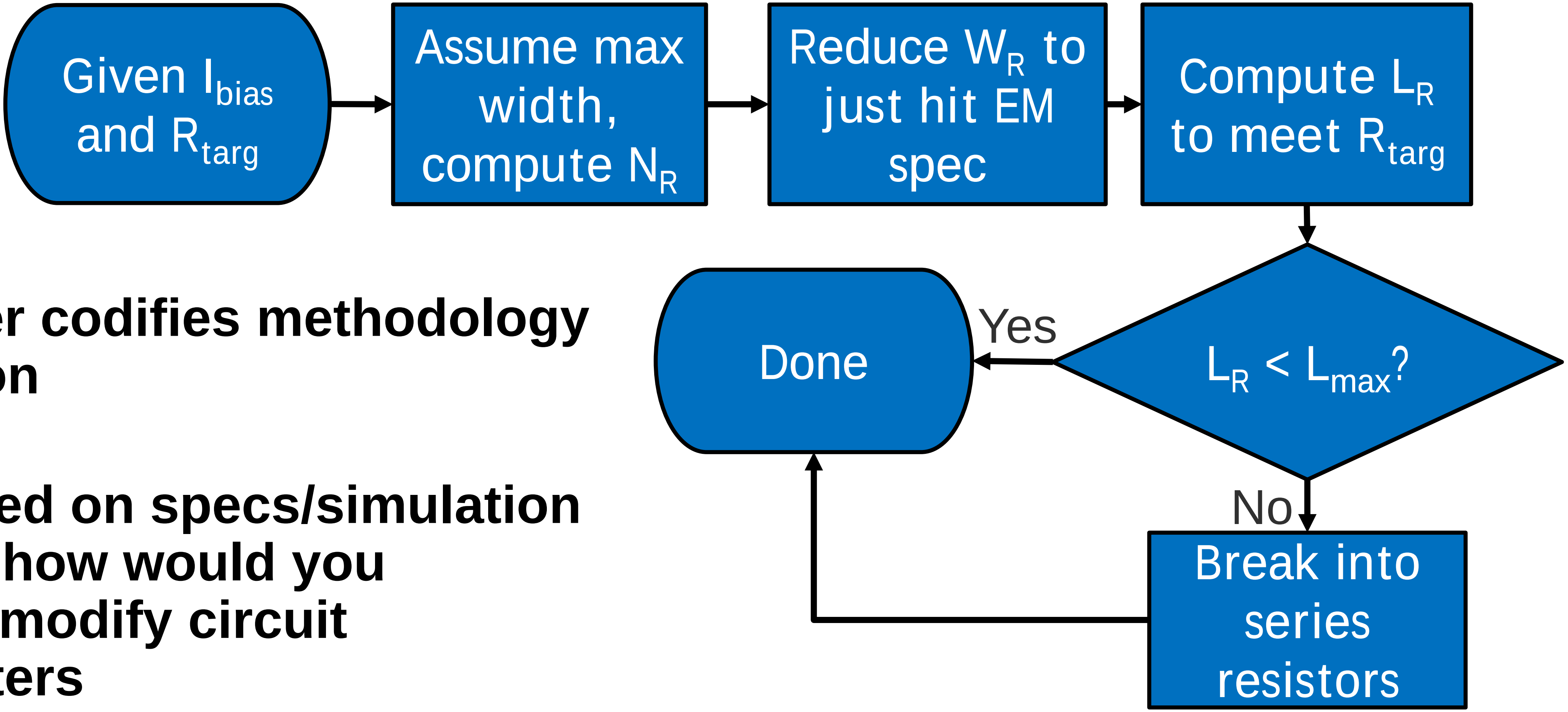


# Design Script Example


Start
(Re)compute parameters from specs
Generate and Extract Circuit
Perform Measurements
Meet Specs?
Yes
Save Design Instance
No
Compute degradation factor κ
Scale failed specs by κ

- **Designer codifies methodology in Python**
- **I.e., based on specs/simulation results, how would you choose/modify circuit parameters**
  - Design equations, transistor lookup tables, binary search, etc.

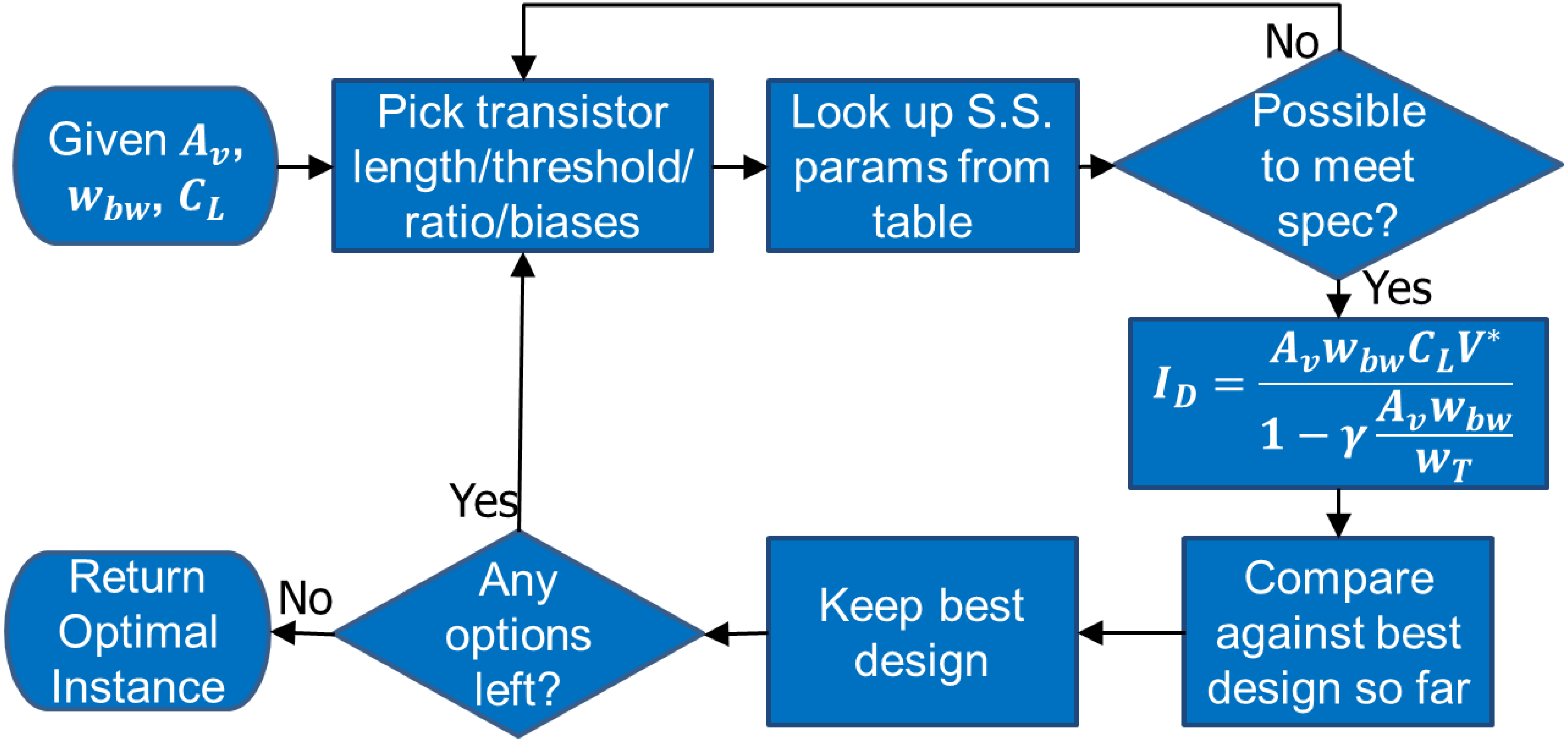
# Example 1: EM-Driven Resistor Sizing


Given $I_{bias}$ and $R_{targ}$
Assume max width, compute $N_R$
Reduce $W_R$ to just hit EM spec
Compute $L_R$ to meet $R_{targ}$
$L_R < L_{max}$?
Yes
Done
No
Break into series resistors

- **Designer codifies methodology in Python**

- **I.e., based on specs/simulation results, how would you choose/modify circuit parameters**

# Example 2: Diff. Amp. Sizing



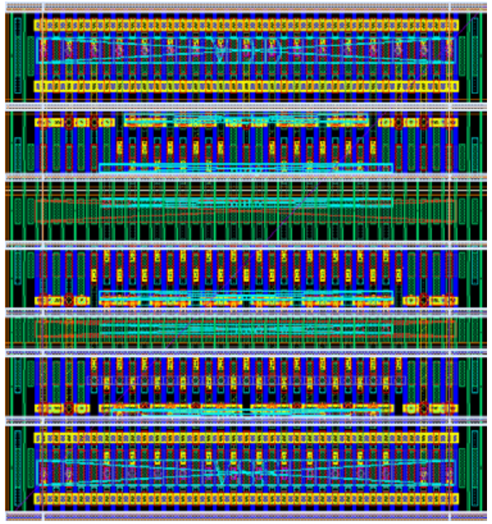
Given $A_v$, $w_{bw}$, $C_L$
Pick transistor length/threshold/ ratio/biases
Look up S.S. params from table
Possible to meet spec?
No
Yes
$I_D = \dfrac{A_v w_{bw} C_L V^*}{1 - \gamma \dfrac{A_v w_{bw}}{w_T}}$
Compare against best design so far
Keep best design
Any options left?
Yes
No
Return Optimal Instance

# Methodology for Layout?

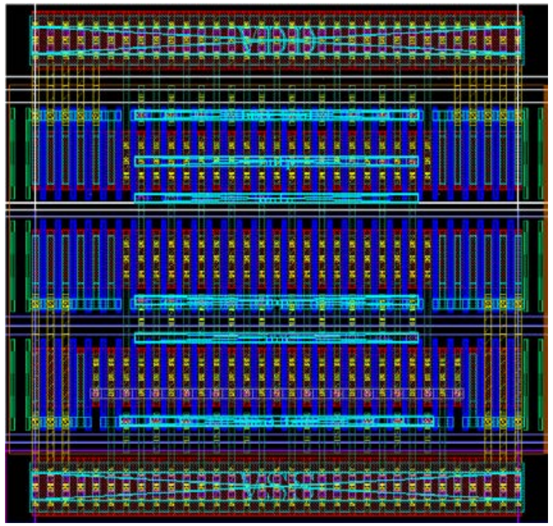
- **Hard to “reuse” a bunch of polygon drawing commands...**
- **Really want to re-use the “floorplan strategy”**
  - I.e., how to construct the floorplan as a function of the parameters
- **Two key realizations enabling portability and parameterization:**
  - Focus on capturing “conceptual” floorplan and electrical constraints instead of process-specific geometry details
  - Enforce a routing (and hence device) grid to simply DRC issues (even in advanced processes)

# Floorplan Invariance Example

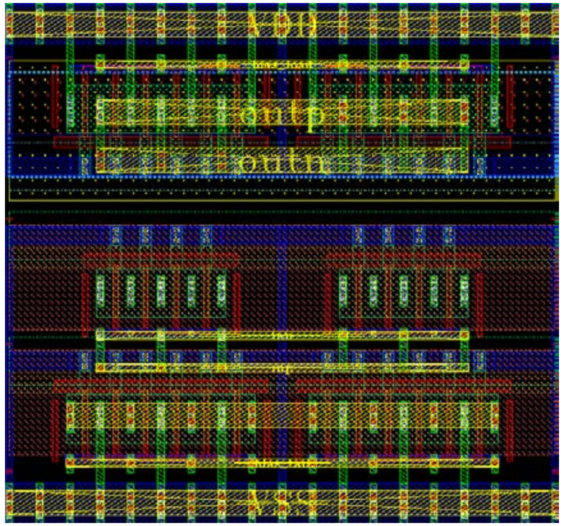
**TSMC 16nm**



**TSMC 28nm**



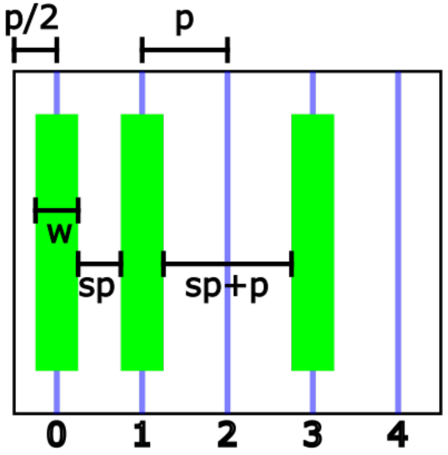
**GF 45nm RFSOI**



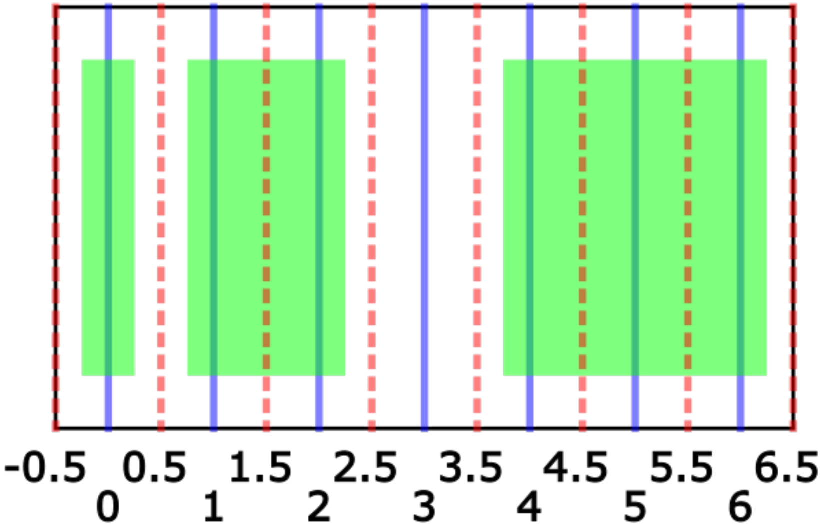
- **Rows of transistors, internal connections vertical, external connections horizontal**
- **General structure driven by electrical constraints, and remains invariant across technologies**

# Addressing DRC Explosion

- **Advanced processes push even custom layouts to be template/grid based**

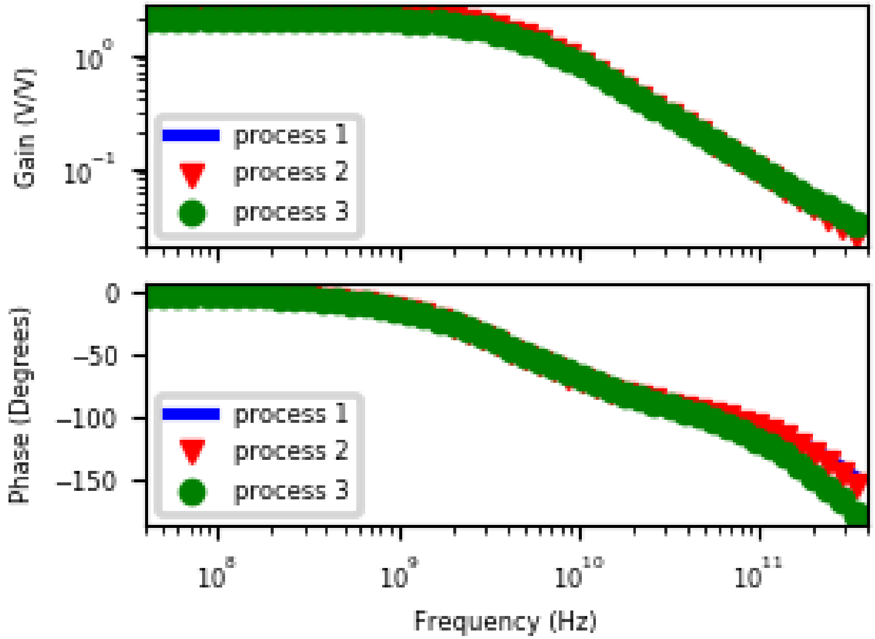


- **BAG improves portability by enforcing a layer-by-layer, customizable routing grid**

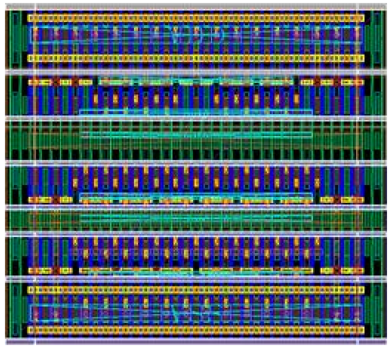


# Example: Diff. Amp. Generator

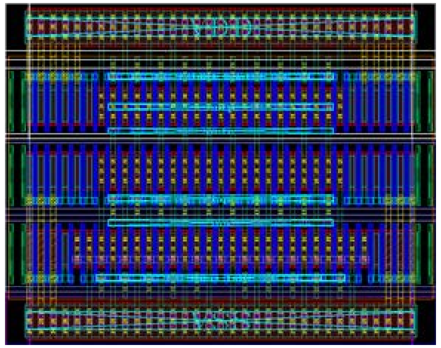
- **Input specs: Gain, BW**
- **Same code produces all three (DRC/LVS clean) instances**
  - Only code difference is implementation of process-specific primitives



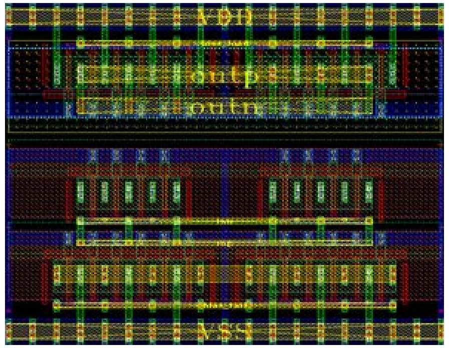
TSMC 16nm



TSMC 28nm



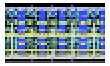
GF 45nm RFSOI



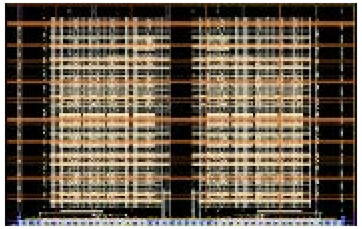
| Gain (V/V) | BW (GHz) |
|---|---|
| 2.07 | 4.12 |
| 2.22 | 4.15 |
| 2.17 | 4.03 |

# Some More Generators We’ve Built So Far

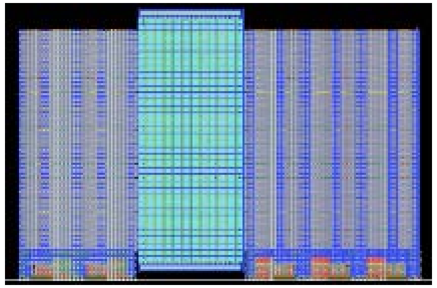
Comparator



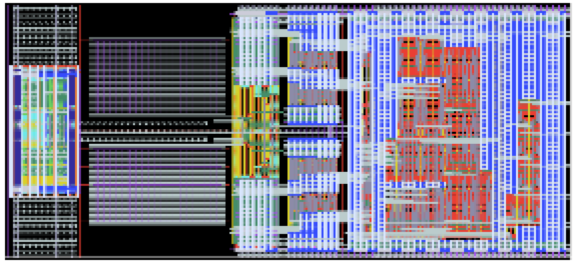
Switch-Cap DAC

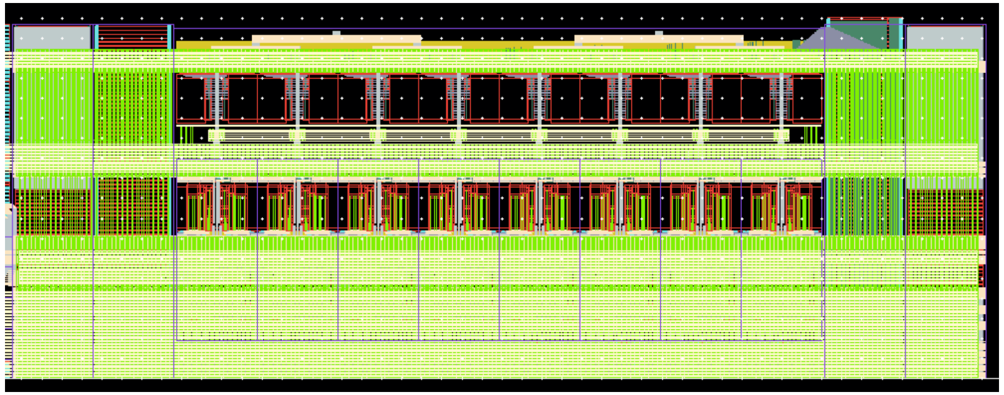


R-ladder DAC

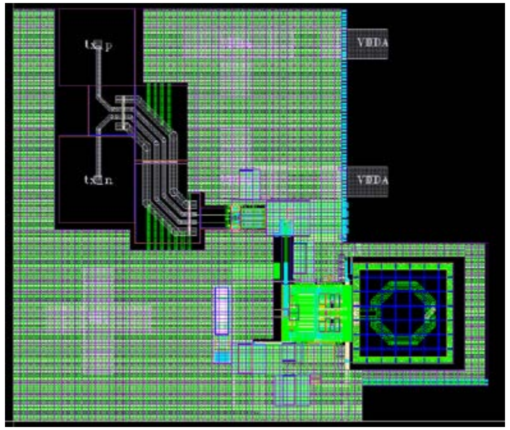


SAR ADC

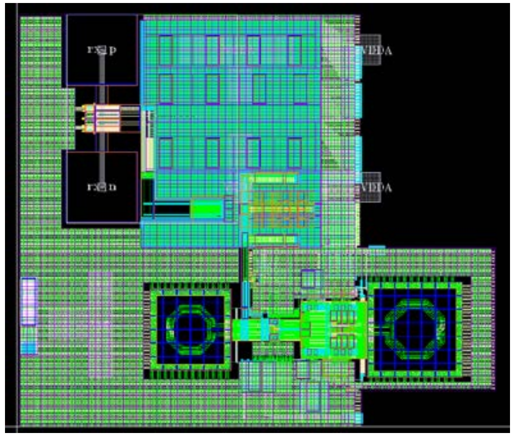




Time-Interleaved SAR ADC



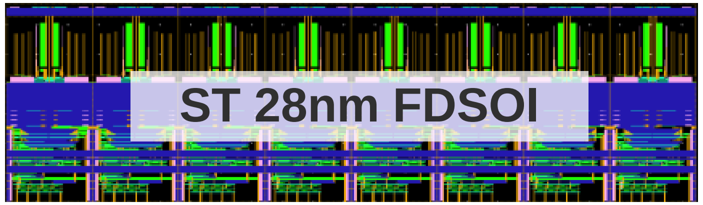
SerDes TX

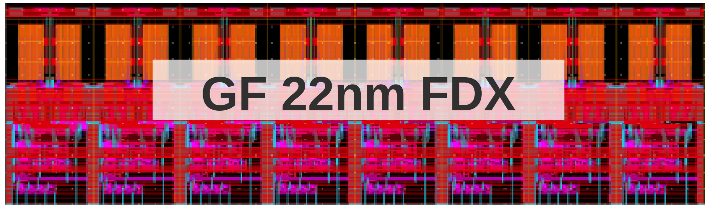


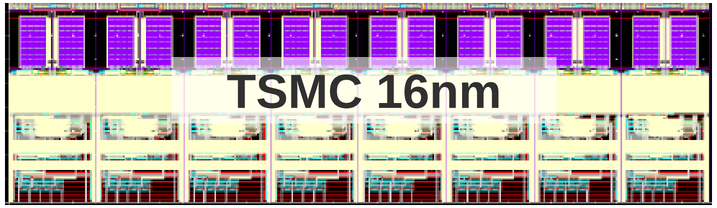
SerDes RX

# Yes, These Are Portable and Parameterized Too

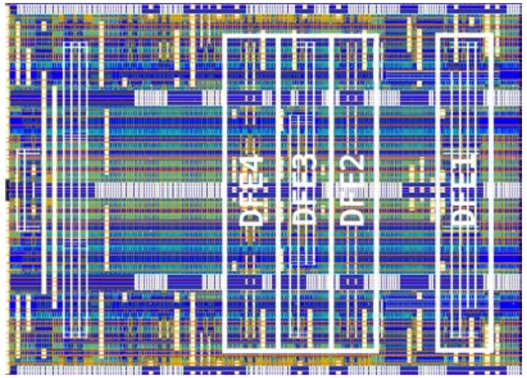
## ADC Core

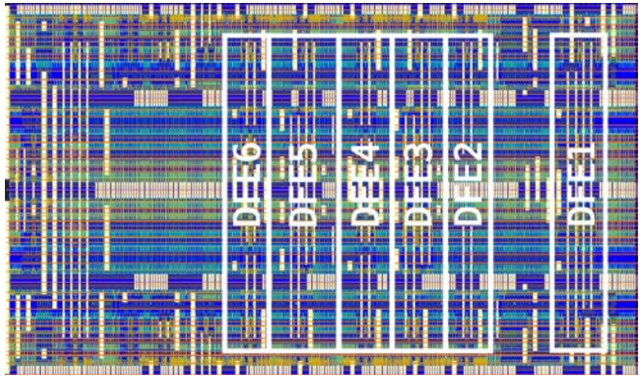




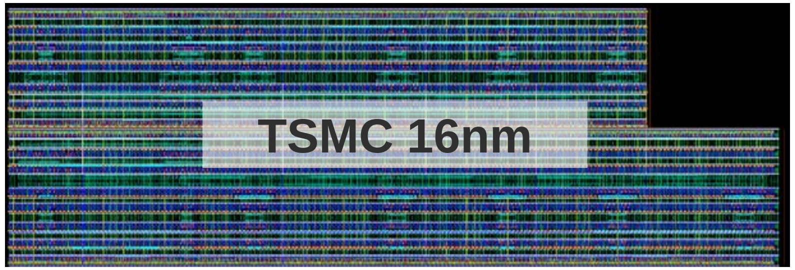


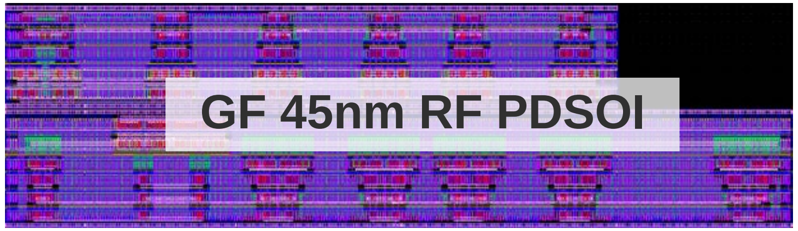
## SerDes RX Core
(variable taps)





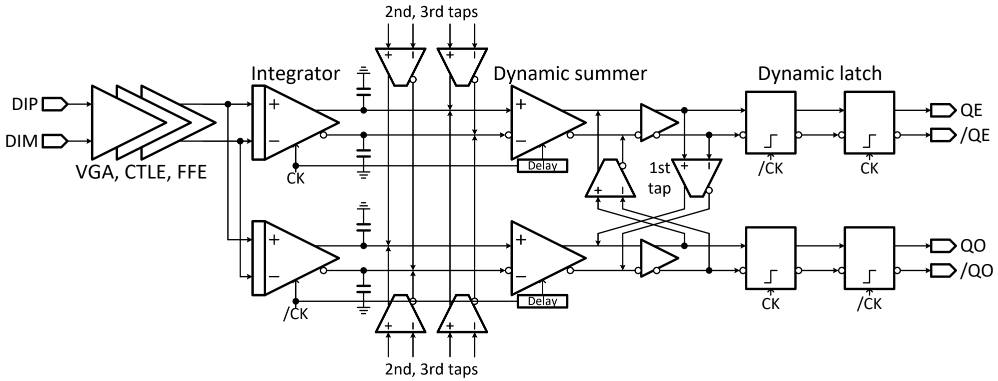
## SerDes RX Datapath



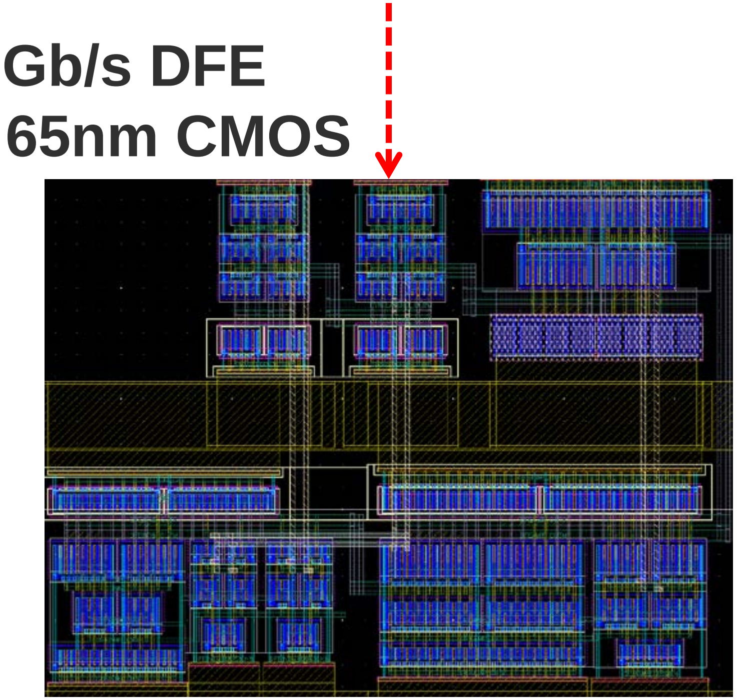


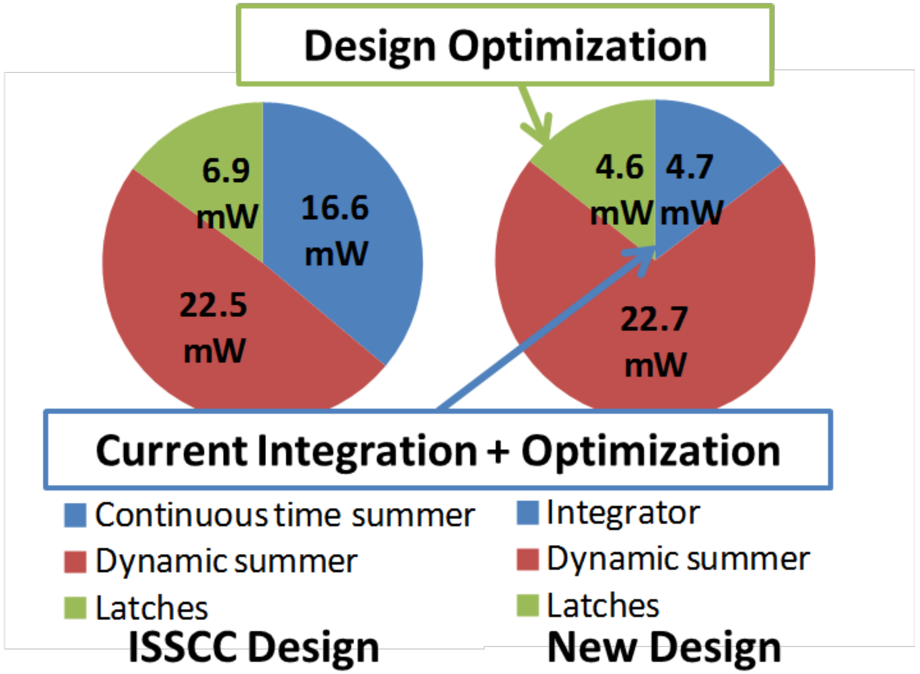
# Performance?

- **Generated circuits can be even better than custom…**
  - Computers are much better at iteration than humans



**60Gb/s DFE in 65nm CMOS**





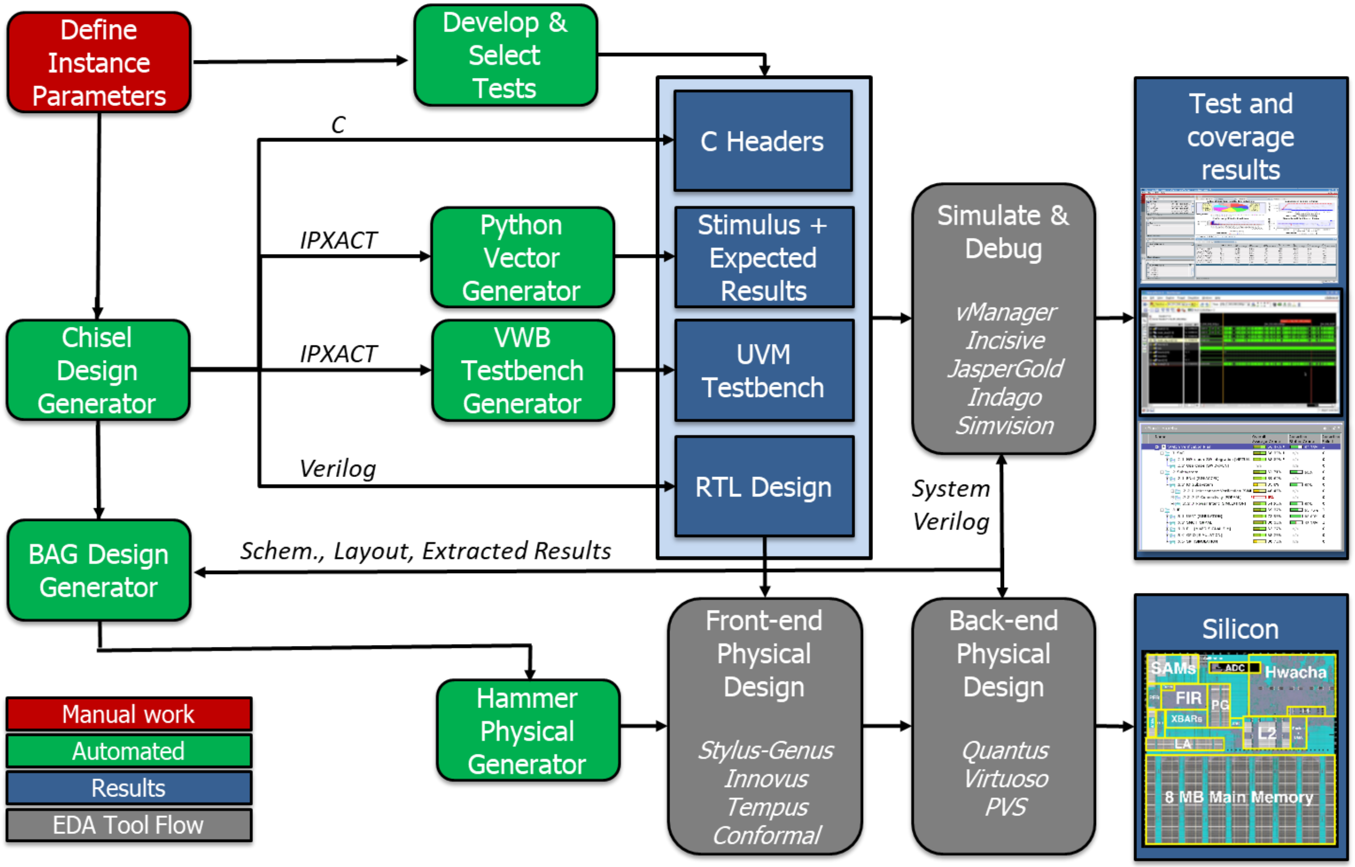
# Outline

- Agile Design
- Generators for Digital: Chisel
- Generators for Analog: BAG

- **Overall Flow + Verification**

- Generators and SoCs
- Effort Data and Looking Forward

# CRAFT Generator-Based Flow



Define Instance Parameters

Develop & Select Tests

C

C Headers

IPXACT

Python Vector Generator

Stimulus + Expected Results

Chisel Design Generator

IPXACT

VWB Testbench Generator

UVM Testbench

Verilog

RTL Design

Simulate & Debug

*vManager*
*Incisive*
*JasperGold*
*Indago*
*Simvision*

Test and coverage results

*System Verilog*

BAG Design Generator

*Schem., Layout, Extracted Results*

Hammer Physical Generator

Front-end Physical Design

*Stylus-Genus*
*Innovus*
*Tempus*
*Conformal*

Back-end Physical Design

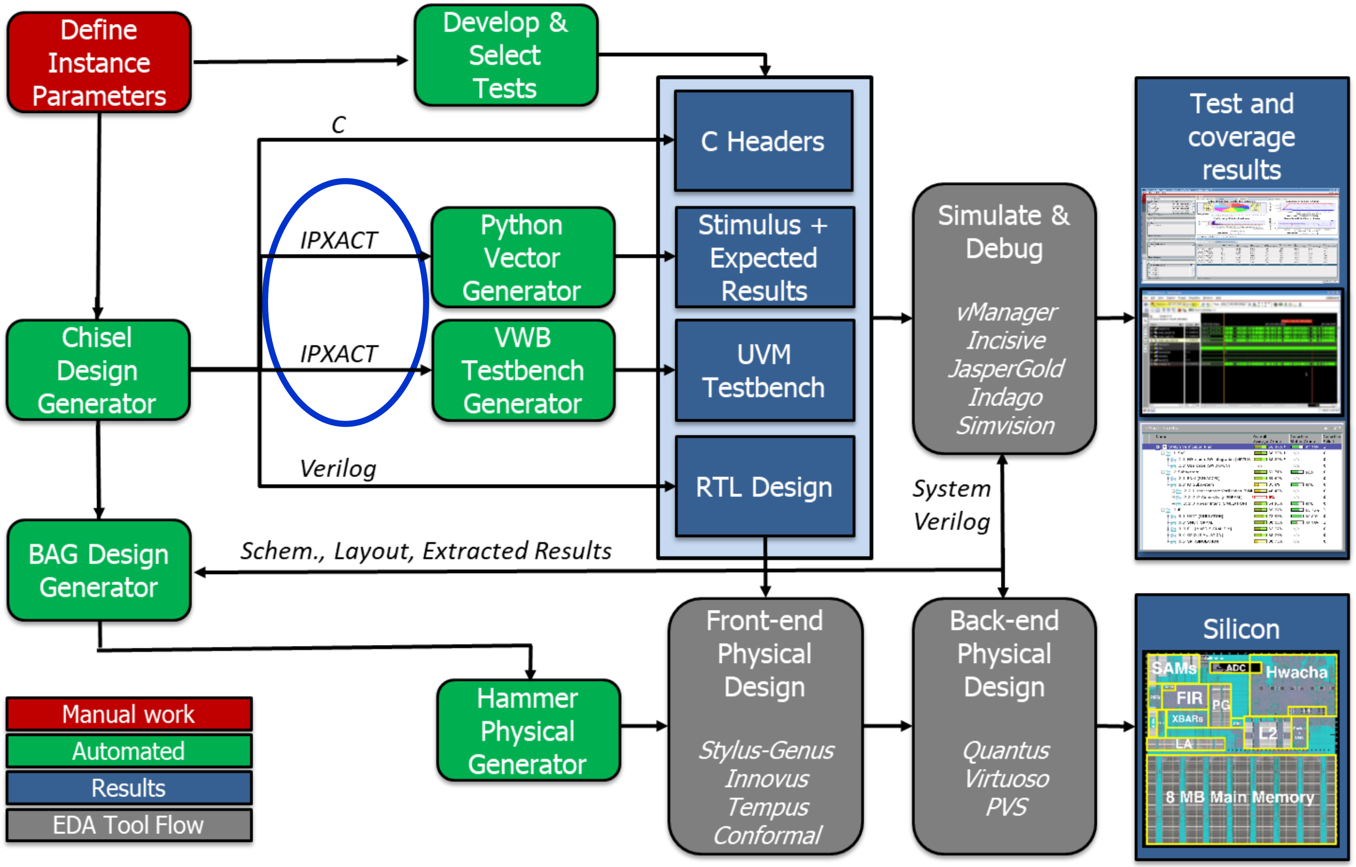
*Quantus*
*Virtuoso*
*PVS*

Silicon

Manual work
Automated
Results
EDA Tool Flow

- **Verification costs just as much as or even more than design**
  - If generating the designs, need to generate the verification environment and tests too

# CRAFT Generator-Based Flow

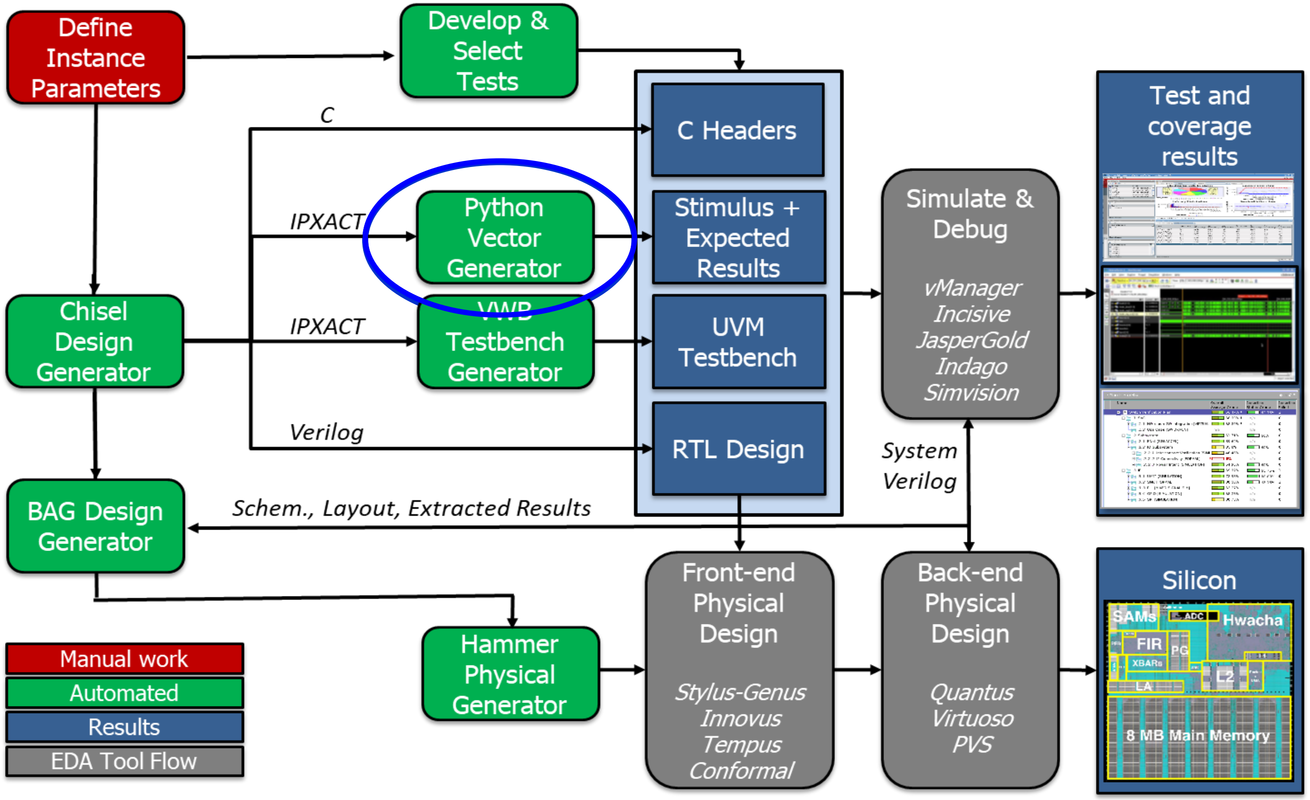


- IP-XACT used to pass parameters
    - So that verification generators know what the instance actually is

# Verification Generators

- **Really two parts to this:**
  - (1) Producing and checking the correct vectors

→ **Python Vector Generator**



# Verification Generators

- **Really two parts to this:**
  (1) Producing and checking the correct vectors
  (2) Hooking everything up correctly

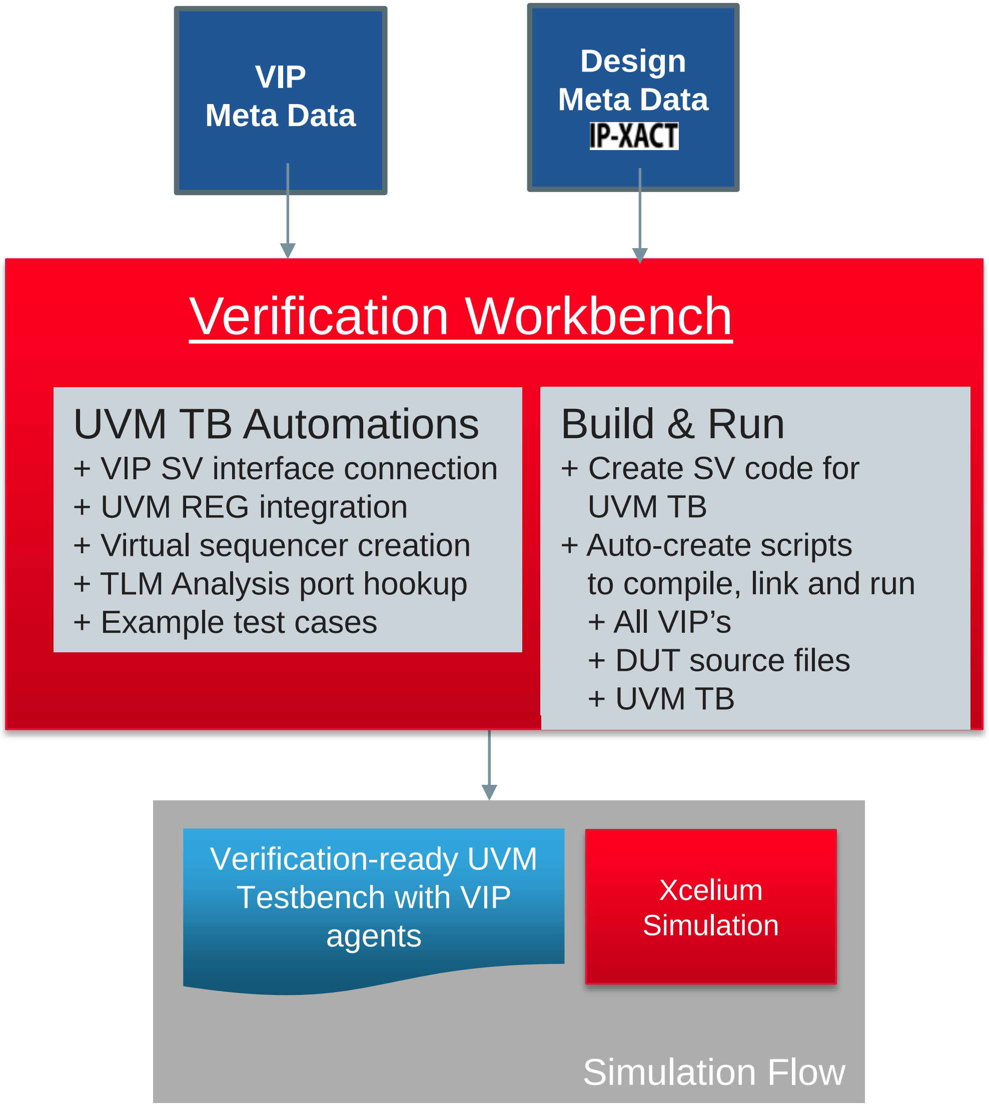
# Verification Generators

- **Really two parts to this:**

  (1) Producing and checking the correct vectors

  (2) Hooking everything up correctly

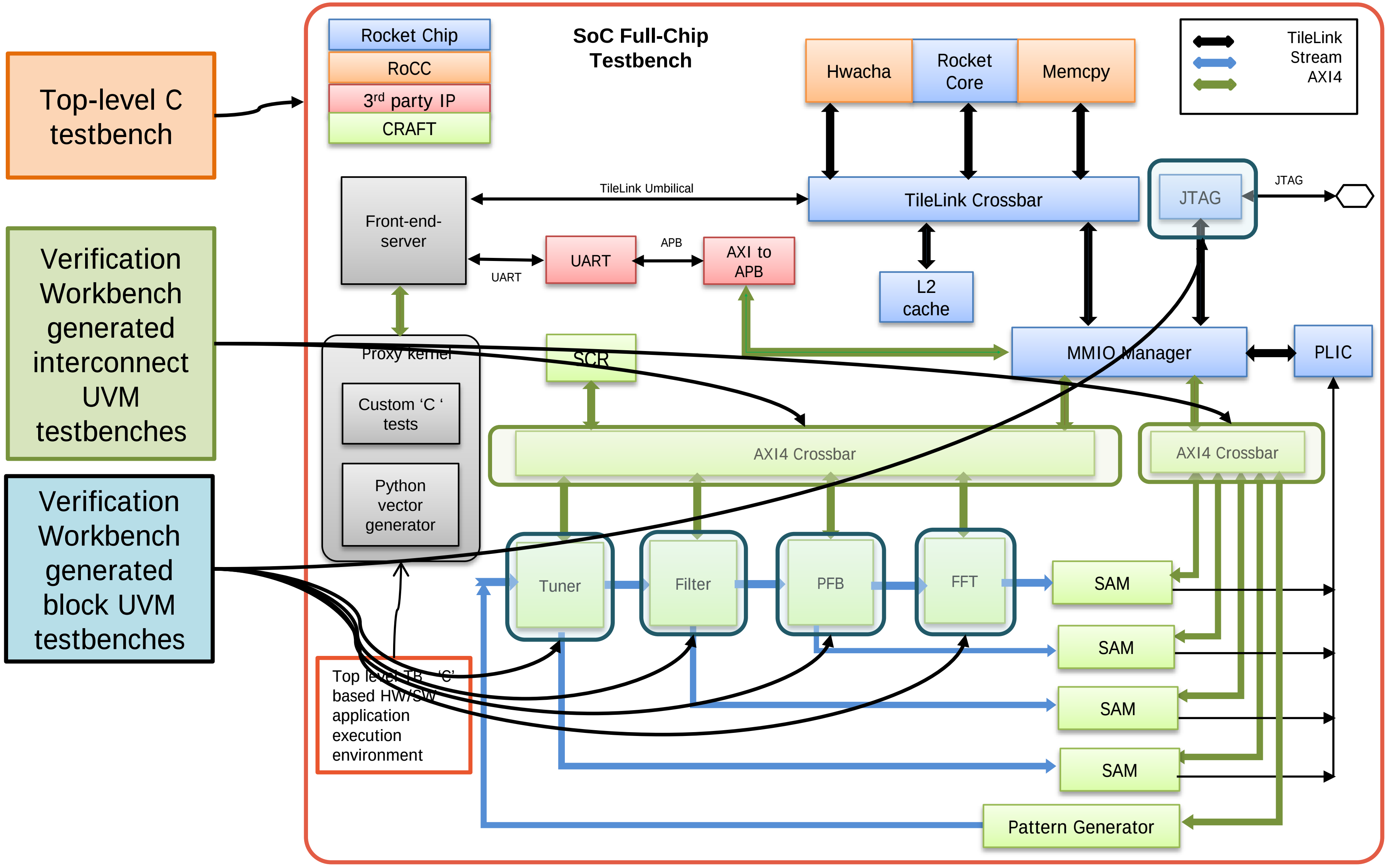
- **When done manually, (2) can actually be the dominant time sink (and source of errors)**
  - Fortunately, Cadence’s Verification Workbench automates this as well
  - And automatically provides VIPs for standard interfaces, analysis of crossbars, …



UVM TB Automations
+ VIP SV interface connection
+ UVM REG integration
+ Virtual sequencer creation
+ TLM Analysis port hookup
+ Example test cases

Build & Run
+ Create SV code for UVM TB
+ Auto-create scripts to compile, link and run
  + All VIP’s
  + DUT source files
  + UVM TB

# Applies At Multiple Levels of Hierarchy



Top-level C testbench

Verification Workbench generated interconnect UVM testbenches

Verification Workbench generated block UVM testbenches

Top level TB 'C' based HW/SW application execution environment

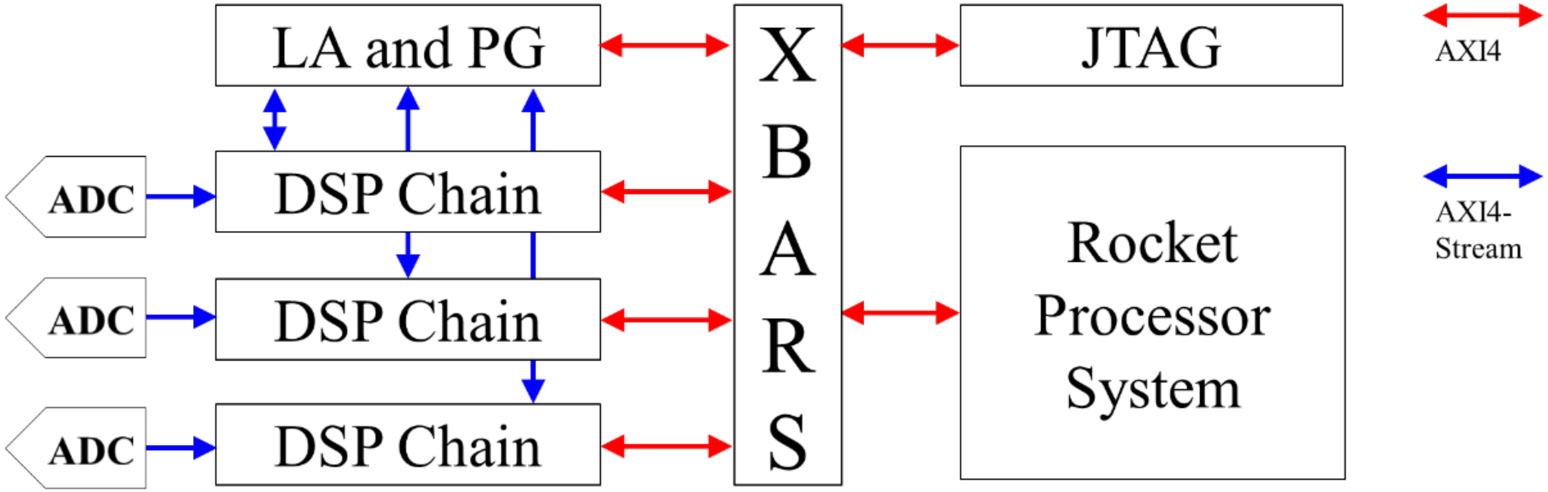
# Outline

- Agile Design
- Generators for Digital: Chisel
- Generators for Analog: BAG
- Overall Flow + Verification

- **Generators & SoCs**

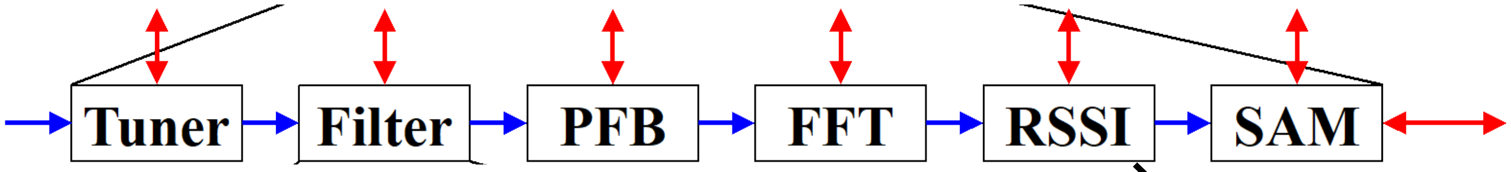
- Effort Data and Looking Forward

# SoC Generator: High-Level View

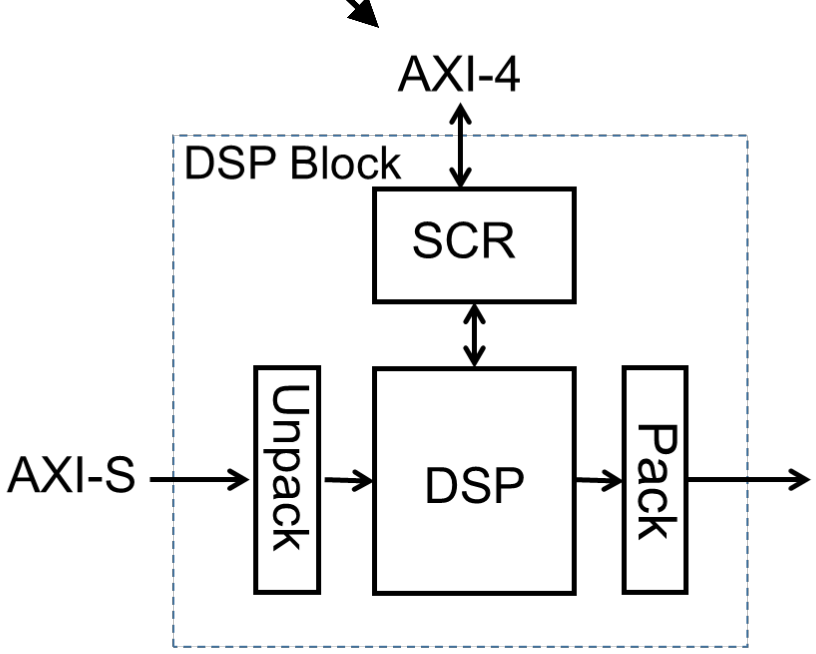


- **Rocket processor system can include:**
  - RISC-V processor(s), memory sub-system (L1/L2 caches, SRAM main memory), vector co-processor, DMA engine, serial adaptor, UART (3rd party IP), …
- **DSP chains parameterized in terms of:**
  - Composition, bitwidths, pipeline depths, number of lanes, data types, connections to PG/LA, …

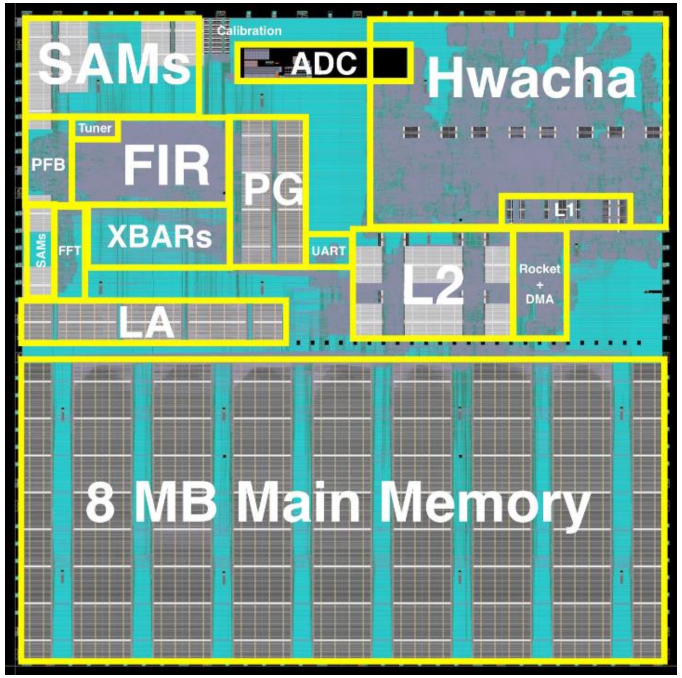
# Enabling Verification and Extension



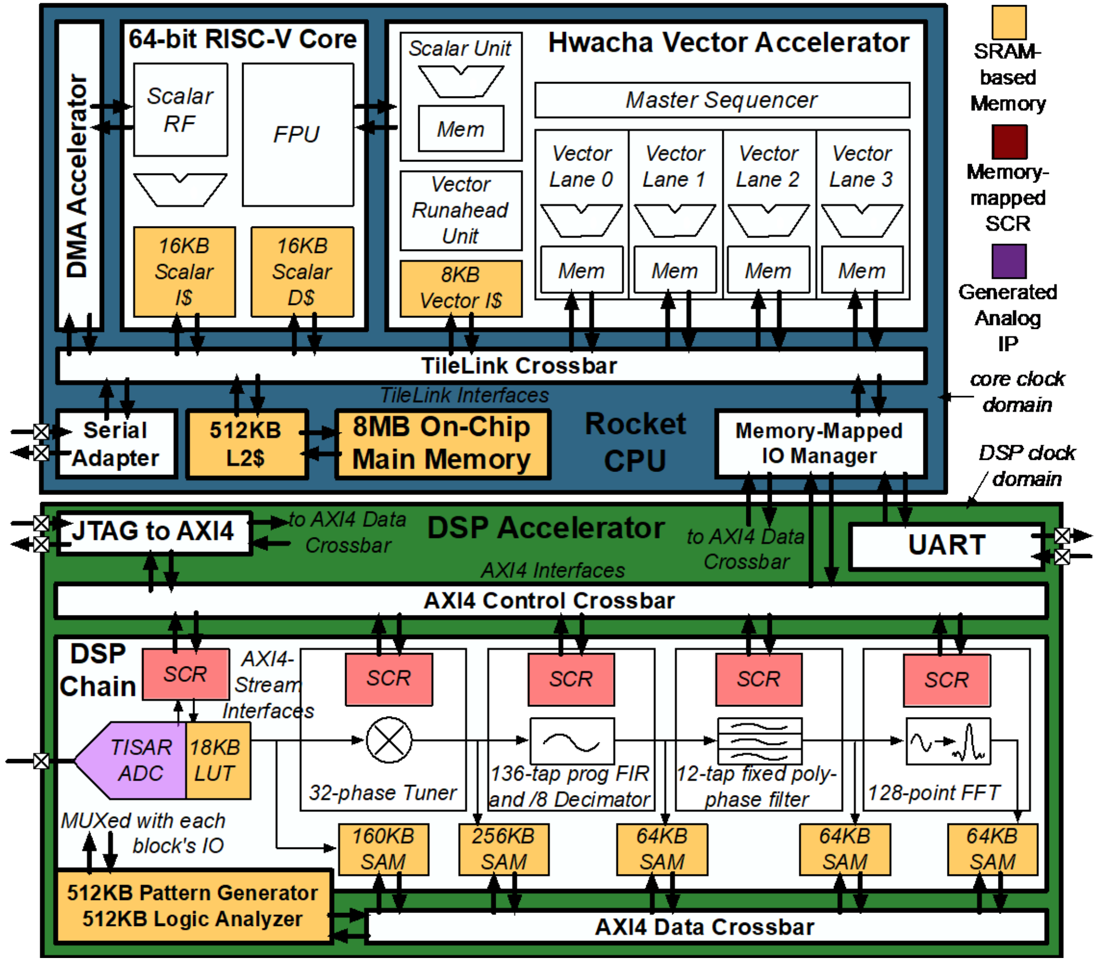
- **SoC architecture and generator designed to enable verification and extension:**
  - All DSP block inputs/outputs use AXI-Stream for data, AXI-4 for control
  - “DSPBlock” generator automatically packs/unpacks data, adds SCR
  - “DSPChain” generator automatically (configurably) adds Streaming to AXI Memory (SAM), expands crossbars, adds testing MUXes, …



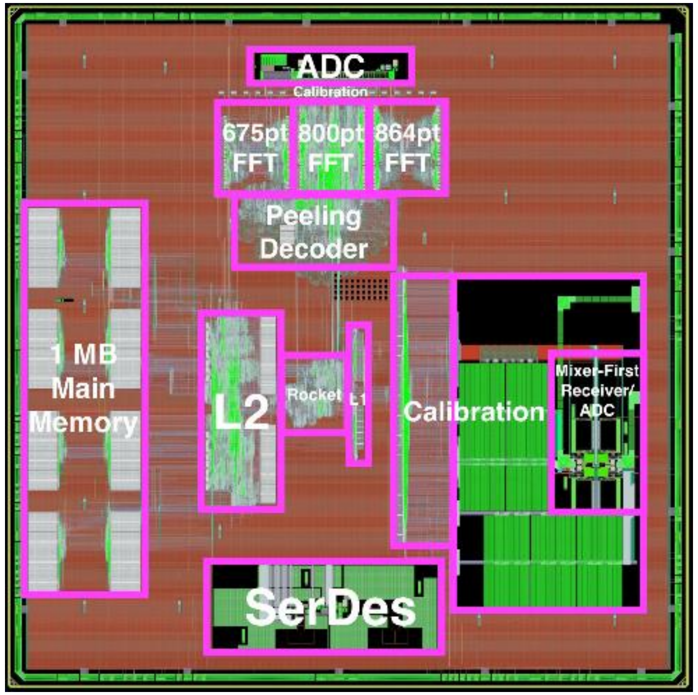
# Generated TSMC 16nm SoC: CraftP1

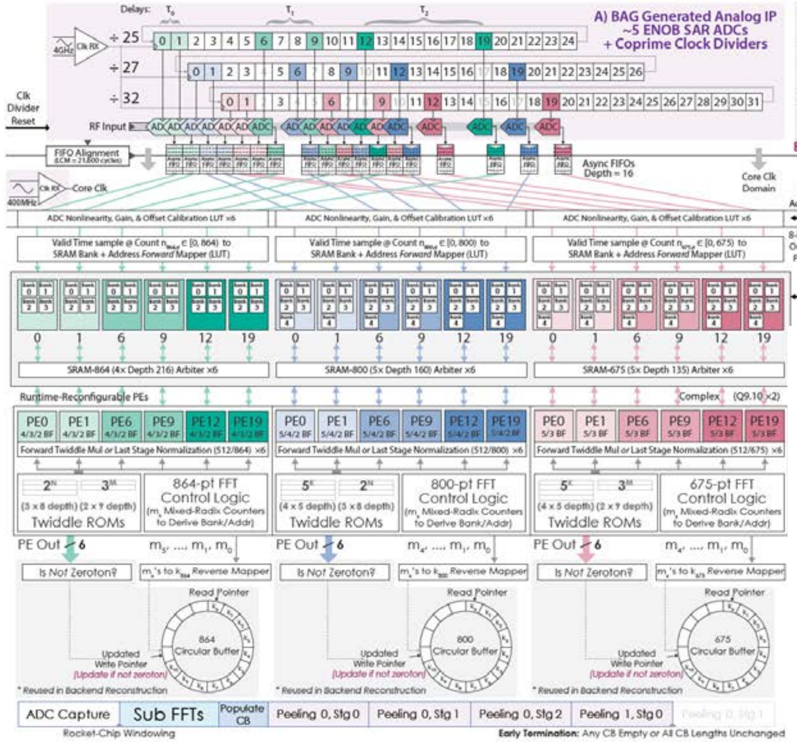


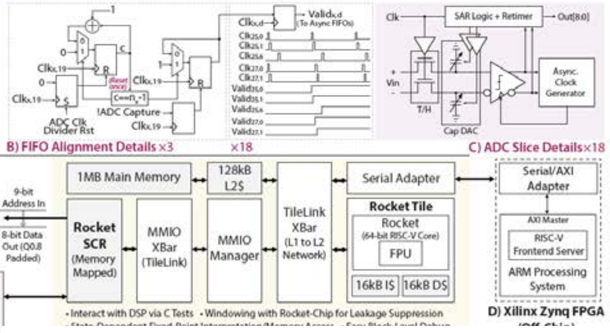
- ~8GS/s, ~6 ENOB 40mW ADC
- EW DSP @ 200-400MHz: tuner, FIR, PFB, FFT – all testable via PG/LA
- 200-400MHz Rocket core, 4-lane vector proc., DMA accelerator, 512kB L2 cache, 8MB SRAM

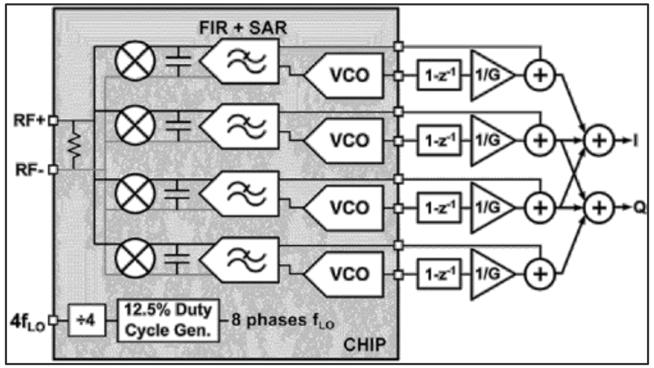


# Generated TSMC 16nm SoC: FFT2

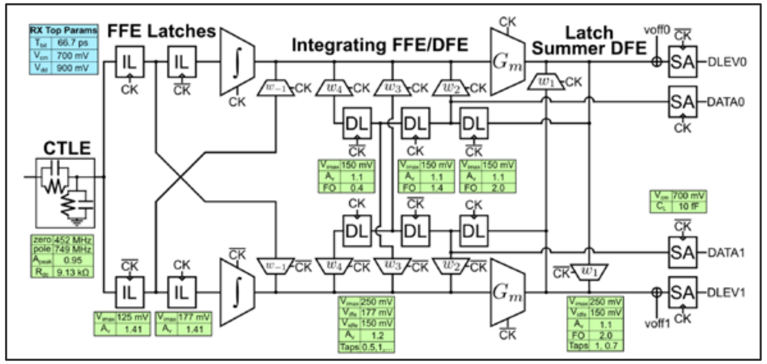






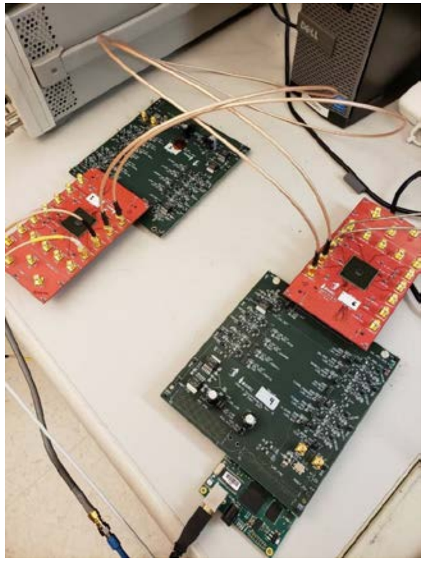


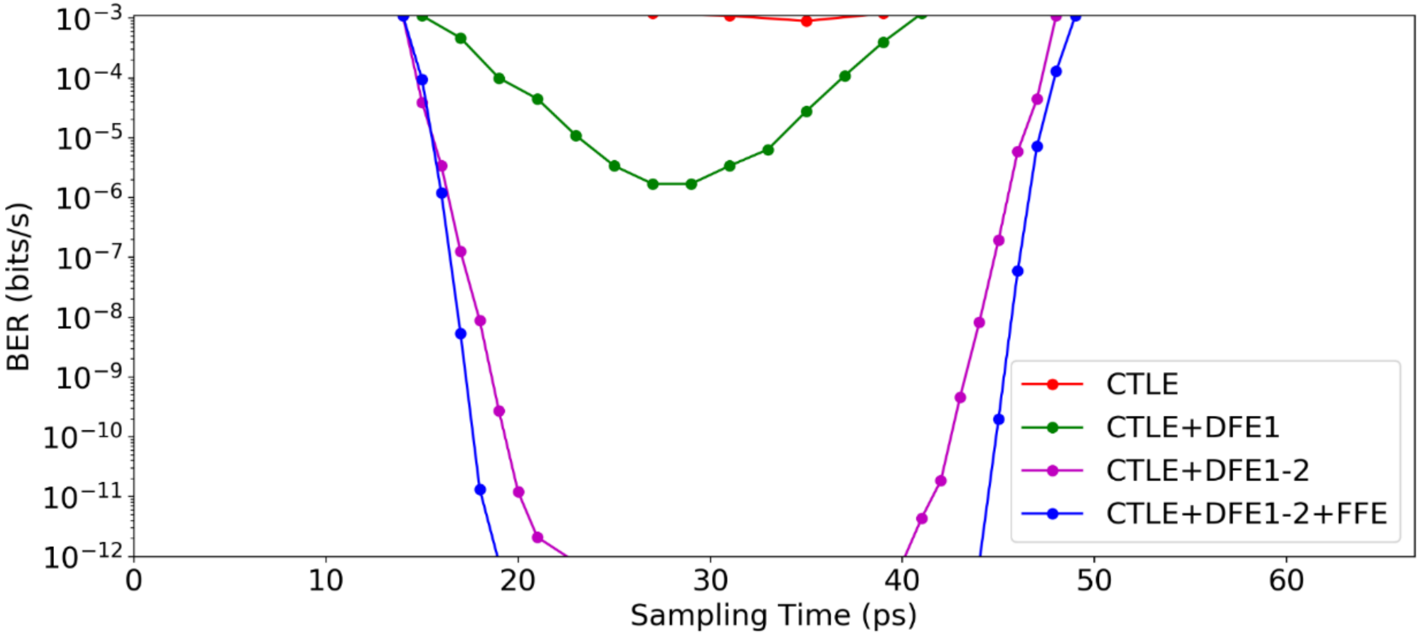
- Sparse sampling ADC, 500MHz sparse FFT, Peeling decoder DSP
- Generated 15Gb/s SerDes front-end

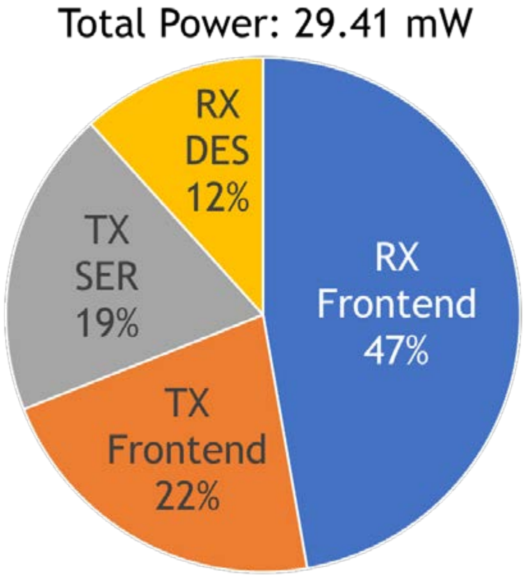


# Yes, The Silicon Worked

**SerDes Frontend:**







E. Chang et al., *VLSI* 2018
A. Whitcombe et al., *VLSI* 2018
A. Wang et al., *ESSCIRC* 2018
S. Bailey et al., *ASSCC* 2018

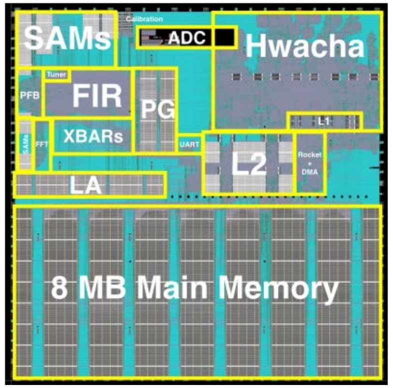
# Outline

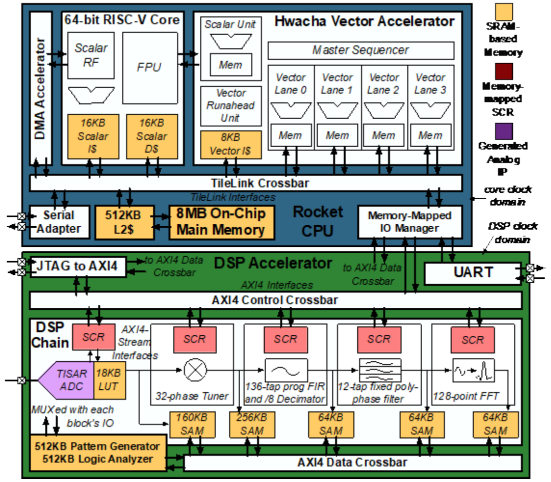
- Agile Design
- Generators for Digital: Chisel
- Generators for Analog: BAG
- Overall Flow + Verification
- Generators & SoCs

- **Effort Data and Looking Forward**

# Phase 1 Chips and Generators

## 10.3M Gate Signal Analysis SoC (CraftP1)





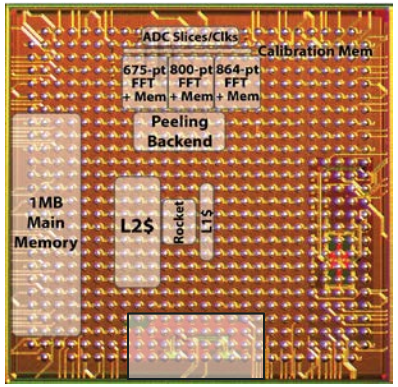
RISC-V Generator

DSP Generators

Accel. Generators

ADC Generator

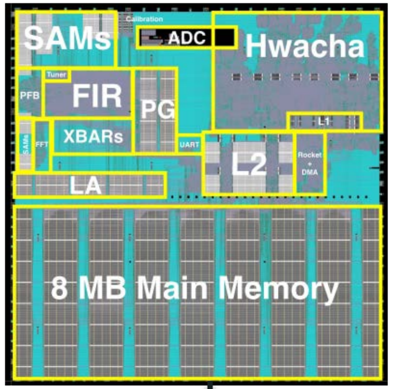
## 4.2M Gate Sparse Recon. SoC

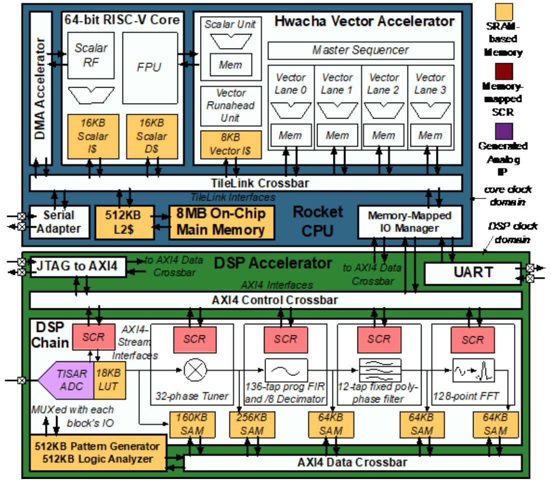


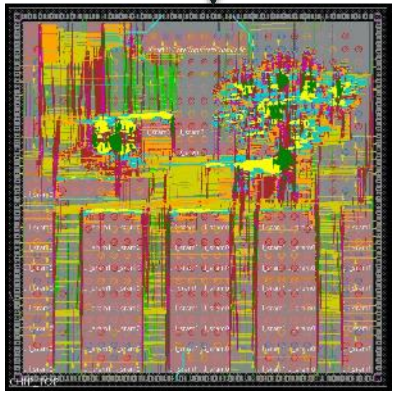
SerDes Frontend Generator

# Phase 1 Chips and Generators

## 10.3M Gate Signal Analysis SoC (CraftP1)







**Ported** signal analysis SoC to **GF 14nm** with **5X less effort**

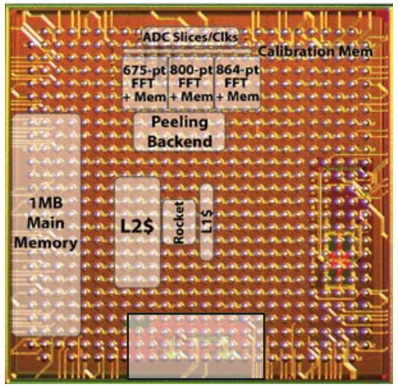
RISC-V Generator

DSP Generators

Accel. Generators

ADC Generator

## 4.2M Gate Sparse Recon. SoC

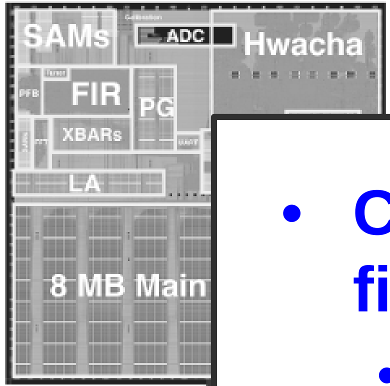


SerDes Frontend Generator

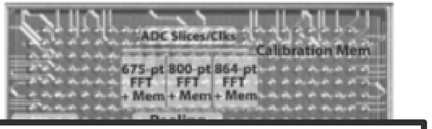
# Phase 1 Chips and Generators

10.3M Gate Signal Analysis SoC (CraftP1)

4.2M Gate Sparse Recon. SoC



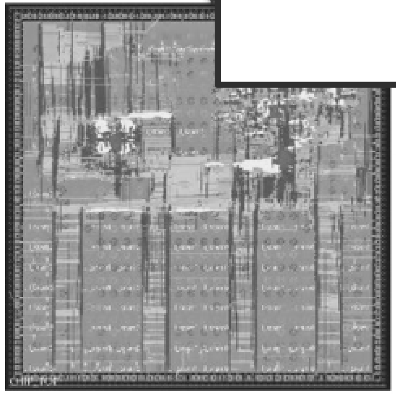
RISC-V Generator



- **CraftP1 SoC effort ~2-2.5X lower than effort estimator figures for a 28nm design!**
  - **Even though CraftP1 numbers included tech. setup and substantial methodology/flow development**

d

Generator

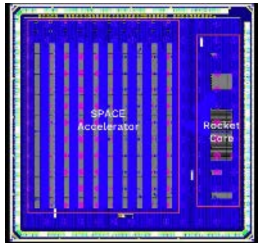


Ported signal analysis SoC to GF 14nm with 5X less effort

# Phase 2 Chips and Generators

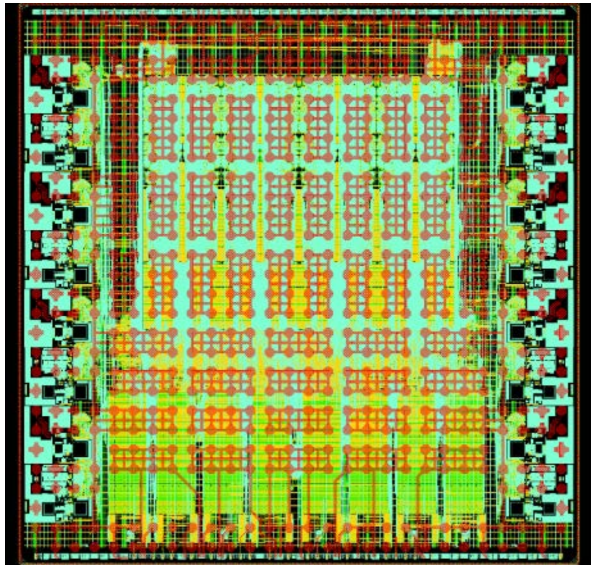
24.6M Gate Multiprocessor SoC (EAGLE)

2.3M Gate AI Accel.



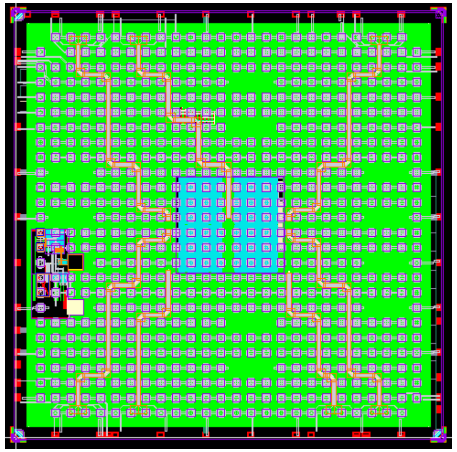
RISC-V Generator

Accel. Generators



Full SerDes Generator

Massive MIMO RF FE

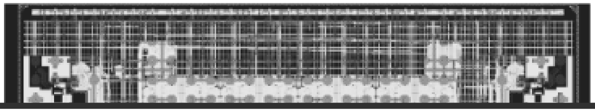


ADC/DAC Generators

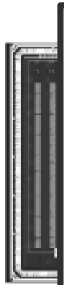
RF Frontend Generators

# Phase 2 Chips and Generators

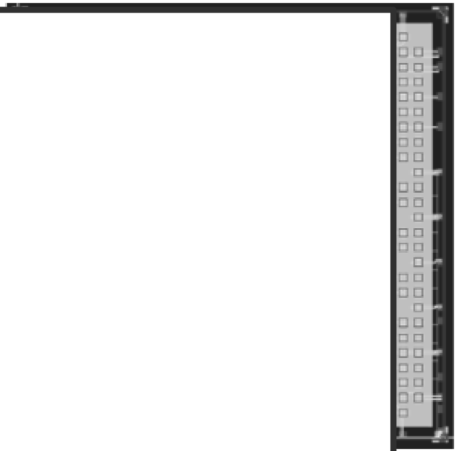
24.6M Gate Multiprocessor SoC (EAGLE)



2.3M Gate AI Accel.



Massive MIMO RF FE



- CraftP1 vs. EAGLE – ~2X <u>less</u> eng. effort for:
  - ~2.5X more gates
  - ~3X higher core frequency
  - ~4X higher analog/mixed-signal complexity

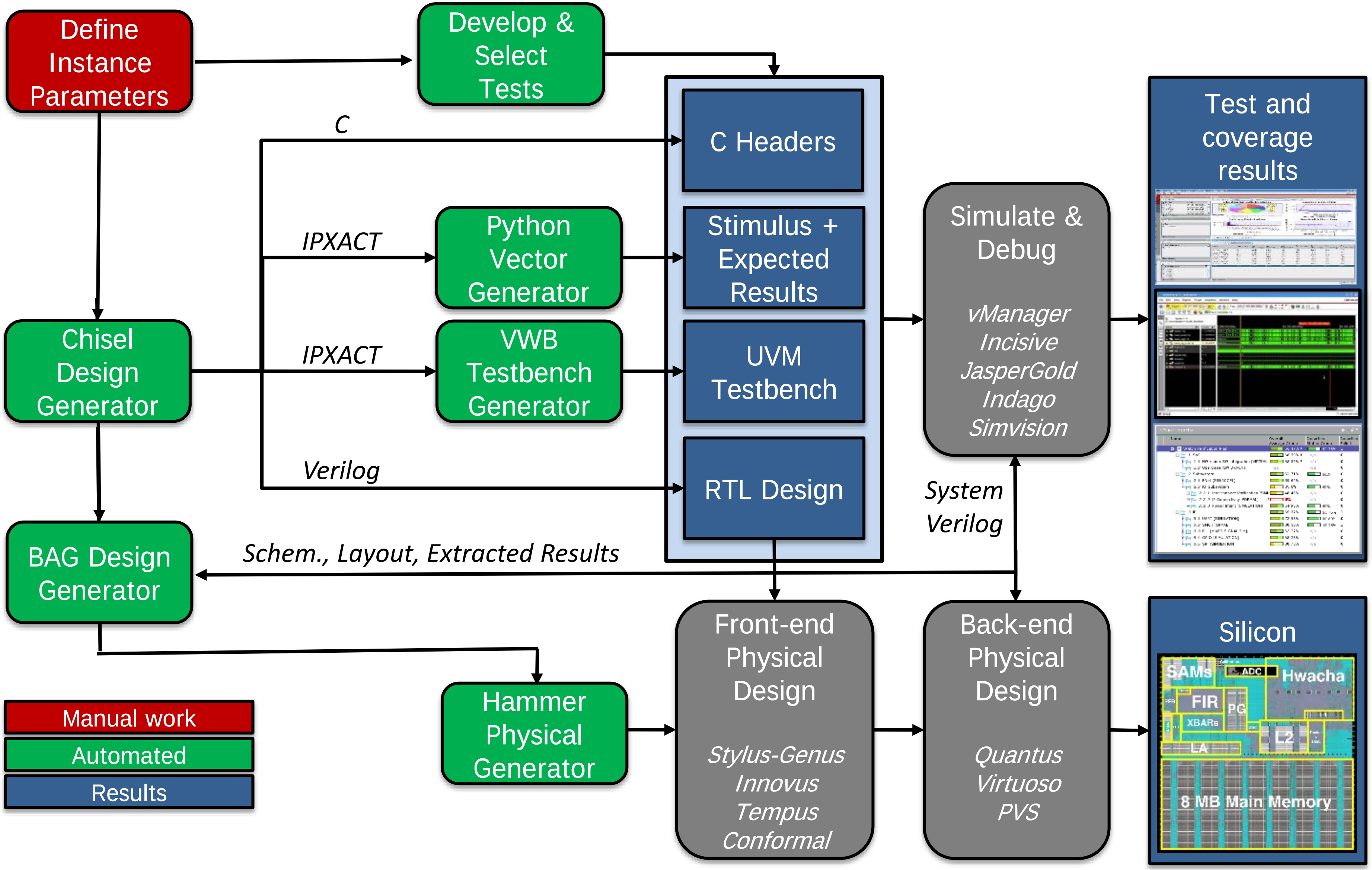
- Clearly demonstrates generator re-use benefits

Generator

Generators

RF Frontend Generators

# Another Teaser



Define Instance Parameters

Develop & Select Tests

C

C Headers

IPXACT

Python Vector Generator

Stimulus + Expected Results

Chisel Design Generator

IPXACT

VWB Testbench Generator

UVM Testbench

Verilog

RTL Design

Simulate & Debug

*vManager*
*Incisive*
*JasperGold*
*Indago*
*Simvision*

Test and coverage results

*System Verilog*

BAG Design Generator

*Schem., Layout, Extracted Results*

Hammer Physical Generator

Front-end Physical Design

*Stylus-Genus*
*Innovus*
*Tempus*
*Conformal*

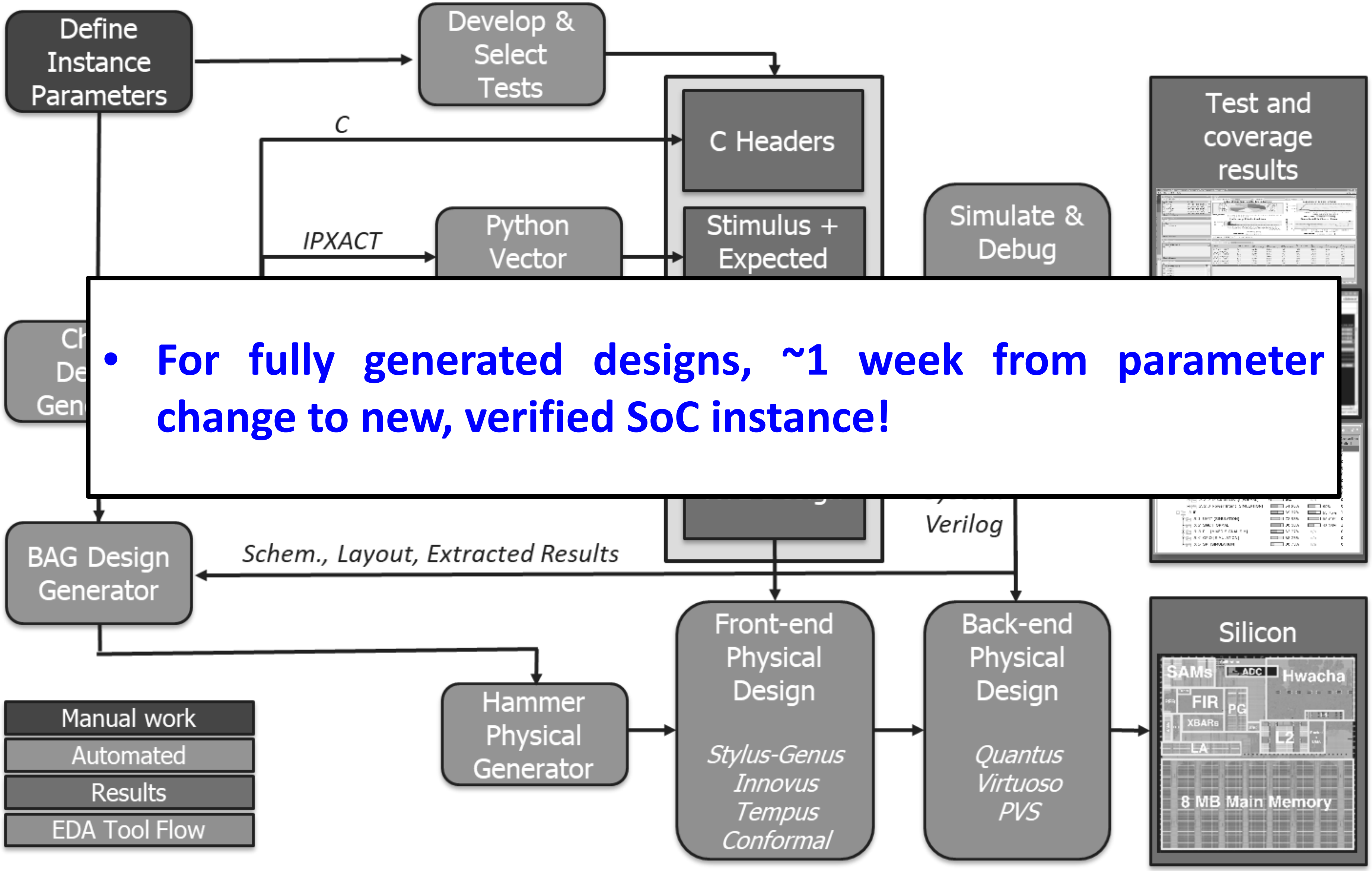
Back-end Physical Design

*Quantus*
*Virtuoso*
*PVS*

Silicon

| | |
|---|---|
| 🟥 | Manual work |
| 🟩 | Automated |
| 🟦 | Results |

# Another Teaser



- **For fully generated designs, ~1 week from parameter change to new, verified SoC instance!**

# Wrap-Up: Common “Questions”

- **“Are you really saying that hardware designers – even analog and layout engineers – should be writing re-usable code?”**
  - Yes – hire a Berkeley undergrad if you need help with coding

- **“What if I don’t know what my methodology is?”**
  - If you designed a circuit, you must have had some kind of methodology
  - Generators force you to “record” (think more carefully about) what you actually did (in code)

- “**What about future technologies with ML-GM, AI-FETs, and SIBL?”**
  - Key is once again to figure out what your own methodology actually is

# How You Can Get Started

- **Open-source boot-camps available here:**
  - Chisel: https://github.com/ucb-bar/generator-bootcamp
  - BAG: https://github.com/ucb-art/BAG2_cds_ff_mpt

- **Reach out to me if you are interested in participating in a live/hosted bootcamp**

- **All generators developed under CRAFT are open-source and/or available for government use**
  - Again, reach out to me if you are interested in finding out more
  - My email: elad@berkeley.edu

# Acknowledgments

- **DARPA CRAFT**
- **UCB – CDN – NGC – BAE Team:**
  - UCB: Stevo Bailey, Paul Rigge, Angie Wang, Eric Chang, Colin Schmidt, John Wright, Richard Lin, Adam Izraelevitz, Jaeduk Han, Howard Mao, Albert Ou, Zhongkai Wang, Chick Markley, Nathan Narevsky, Woorham Bae, Kosta Trotskovsky, Marko Kosunen, Edward Wang, Pengpeng Lu, Brian Richards, Jonathan Bachrach, Borivoje Nikolic
  - NGC: Steven Shauck, Sergio Montano, Justin Norsworthy, Munir Razzaque, Wen Hau Ma, Akalu Lentiro, Matthew Doerflein
  - Cadence: Darin Heckendorn, Chirag Goyal, Rudy Mason, Jim McGrath, Franco DeSeta, Mark Snowden, Ronen Shoham, Mike Stellfox, Eric Naviasky, Dan Fuhrman, Joseph Cole
  - BAE: Silviu Chiricescu, Richard Berger, Kendall Farnham
- **BWRC and ADEPT sponsors**