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:
  - Specification
  - Architectural design
  - Implementation
  - Verification and test
  - Support and maintenance

- “Agile” development:
  - Specification
  - Architecture design
  - Implementation
  - Verification and test
  - Support and maintenance

Fox, Patterson, 2013.
Software Had a Similar Problem

- “Waterfall” development:
  - Specification
  - Architectural design
  - Implementation
  - Verification and test
  - Support and maintenance

- “Agile” development:
  - Specification
  - Architectural design
  - Implementation
  - Verification and test
  - Support and maintenance

<table>
<thead>
<tr>
<th>Approach</th>
<th>On-Time</th>
<th>Late</th>
<th>Cancelled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall</td>
<td>10%</td>
<td>52%</td>
<td>38%</td>
</tr>
<tr>
<td>Agile</td>
<td>76%</td>
<td>20%</td>
<td>4%</td>
</tr>
</tbody>
</table>

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

Richard Berger
Silviu Chiricescu

Steven Shauck,
Matthew Doerflein

Mike Stellfox, Joseph Cole

Contains unpublished material – do not redistribute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

```scala
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

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

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

hmm... let’s make a counter......
ChiselDSP

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

```
class FIR[T <: Data:Ring:ConvertibleTo](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_* _)
  }
}
```

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

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

- 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…

(Re-)Draw Layout
(Virutoso)
Reminder: Proposed Approach

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

- 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

- Designer codifies methodology in Python
- I.e., based on specs/simulation results, how would you choose/modify circuit parameters

Given $I_{\text{bias}}$ and $R_{\text{targ}}$

Assume max width, compute $N_R$

Reduce $W_R$ to just hit EM spec

Compute $L_R$ to meet $R_{\text{targ}}$

$L_R < L_{\text{max}}$?

Yes

Done

No

Break into series resistors
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?

Yes → $I_D = \frac{A_v w_{bw} C_L V^*}{1 - \gamma \frac{A_v w_{bw}}{w_T}}$

No → Return Optimal Instance

Yes → Any options left?

No → Keep best design → Compare against best design so far
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

- 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

<table>
<thead>
<tr>
<th>Gain (V/V)</th>
<th>BW (GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.07</td>
<td>4.12</td>
</tr>
<tr>
<td>2.22</td>
<td>4.15</td>
</tr>
<tr>
<td>2.17</td>
<td>4.03</td>
</tr>
</tbody>
</table>
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
- ST 28nm FDSOI
- GF 22nm FDX
- TSMC 16nm

SerDes RX Core (variable taps)

SerDes RX Datapath
- TSMC 16nm
- GF 45nm RF PDSOI
Performance?

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

![Diagram of 60Gb/s DFE in 65nm CMOS](image)
Outline

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

• Overall Flow + Verification

• Generators and SoCs
• Effort Data and Looking Forward
• Verification costs just as much as or even more than design
  • If generating the designs, need to **generate** the verification environment and tests too
• **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, …
Applies At Multiple Levels of Hierarchy
Outline

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

• Generators & SoCs

• Effort Data and Looking Forward
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)

4.2M Gate Sparse Recon. SoC

- **RISC-V Generator**
- **DSP Generators**
- **Accel. Generators**
- **ADC Generator**
- **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

4.2M Gate Sparse Recon. SoC

- RISC-V Generator
- DSP Generators
- Accel. Generators
- ADC Generator
- SerDes Frontend Generator
Phase 1 Chips and Generators

10.3M Gate Signal Analysis SoC (CraftP1) 4.2M Gate Sparse Recon. SoC

• 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

Ported signal analysis SoC to GF 14nm with 5X less effort
Phase 2 Chips and Generators

2.3M Gate AI Accel.

RISC-V
Generator

Accel.
Generators

Full SerDes
Generator

Massive MIMO RF FE

24.6M Gate Multiprocessor SoC (EAGLE)

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 less eng. effort for:
  - ~2.5X more gates
  - ~3X higher core frequency
  - ~4X higher analog/mixed-signal complexity

  Clearly demonstrates generator re-use benefits
Another Teaser
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:
  • 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