Twine: A Chisel Extension for Component-Level Heterogeneous Design

Shibo Chen, Yonathan Fisseha, Jean-Baptiste Jeannin, Todd Austin

University of Michigan
The Death of Homogeneous Designs

Classical technological drivers are failing.

CPU Performance Scaling is Dead.

Cost of Design is Exploding.

The Death of Homogeneous Designs

Classical technological drivers are failing.

- CPU Performance Scaling is Dead.
- Cost of Design is Exploding.
- Homogeneous Designs are No Longer Cost-Effective.


The Era of Heterogeneous Designs

Increasing Amount of Hardware Designed, Customized, and Tailored for Specific Applications.

Customized SoC

- Amazon Announces Graviton2 SoC Along With New AWS Instances: 64-Core Arm With Large Performance Uplifts
- Microsoft’s Reported Plans to Design Its Own CPUs: 5 Thoughts
- Apple unveils M1, its first system-on-a-chip for Mac
- Google said to be preparing its own chips for use in Pixel phones and Chromebooks

Application-specific Hardware

- Cerebras’ wafer-size chip is 10,000 times faster than a GPU
- To foil hackers, this chip can change its code in the blink of an eye
- Startup Rolls Out On-Device Voice Command Chip That Runs on 1 mW
- Tesla vaunts creation of ‘the best chip in the world’ for self-driving
Meeting Distinct Requirements

Various Algorithms + Diverse Settings + Different Technologies

Distinct Performance, Area, Power, and Cost Requirements

Different Designs, Topologies, Functionalities
Meeting Distinct Requirements

Various Algorithms

Diverse Settings

Different Technologies

Distinct Performance, Area, Power, and Cost Requirements

How Do Developers Create New Designs?

Different Designs, Topologies, Functionalities
The Zen of Heterogeneous Design

Reuse
- Use Component Libraries
- Reuse Components in Other Designs

Customize
- Add New Stage
- Add New Functionalities
- Target New Application

Existing Components
- Replicate Functional Units
- Widen Memory Bandwidth
- Add Additional Layers

New Components
- Vectorize Functional Units
- Pipeline Operations
- Change Data Format

Scale

Reconfigure
Modern Hardware Design Languages Should Help Developers Efficiently Complete These Jobs:

- Use Component Libraries
- Reuse Components in Other
- Widen Memory Bandwidth

- Add New Stage
- Add New Functionalities
- Pipeline Operations
- Change Data Format

Scale

Existing Components

Reuse

New System

Customize

Reconfigure
Our Solution: Twine

Twine is a Chisel extension for component-level heterogeneous designs.

Twine supports essential features for heterogeneous design:

- Standardize Control Interfaces (reusability, scalability)
- High-level Operator for Composability (scalability, reconfigurability, customizability)
- Automate Control Coordination & Data Type Conversion (scalability, reconfigurability)
- Low Level Access to Chisel Primitives (reconfigurability, customizability)
Content

• Motivation

• Twine Features
  • Standard Control Interfaces
  • High-level Operator for Composability
  • Control Coordination & Type Conversion Automation

• Implementation & Circuit Generation

• Experiments & Results

• Limitations & Future work

• Conclusion
Content

• Motivation

• Twine Features
  • Standard Control Interfaces
  • High-level Operator for Composability
  • Control Coordination & Type Conversion Automation

• Implementation & Circuit Generation

• Experiments & Results

• Limitations & Future work

• Conclusion
Twine Standard Control Interfaces

* Interfaces define how a component communicates.

* Standardizing interfaces is a common practice.
  * Many standard interfaces for coarse-grained components (e.g., AXI, PCIe).
  * Too heavy for intra-accelerator communication.

* Naive approach: without standard control interfaces
  * Inspect, examine, and adapt component interfaces every time.
  * Automation is not straightforward, requiring significant designer effort and debugging.

* Better approach: standard control interfaces
  * Make component behaviors more predictable.
  * Enable high-level automation.
Twine Standard Control Interfaces

• Declaration of a Twine Module Interface
  
  ```scala
  val in  = IO(new ModuleInputType) // All data in-flow ports
  val out = IO(new ModuleOutputType) // All data out-flow ports
  val ctrl = IO(new ModuleCtrlType) // One of four standard control Interfaces
  ```

• Four Standard Control Interfaces in Twine
  
  - TightlyCoupledIOCtl
  - ValidIOCtl
  - DecoupledIOCtl
  - OutOfOrderIOCtl

  Flexibility
  - High
  - Low

  Complexity
  - High
  - Low
Content

• Motivation

• Twine Features
  • Standard Control Interfaces
  • High-level Operator for Composability
  • Control Coordination & Type Conversion Automation

• Implementation & Circuit Generation

• Experiments & Results

• Limitations & Future work

• Conclusion
High-level Operator for Composability

• New \textit{flow} operator >>> to distinguish from the original Chisel wire connection
  • Producer >>> Consumer
  • Supports all levels of granularity
  • \textit{moduleA} >>> \textit{moduleB}, \textit{wireA} >>> \textit{wireB}, \textit{Bundle(wireA, wireB)} >>> \textit{moduleA}

• Focus on producer/consumer relations
  • \textit{Producer}: module that outputs completed values
  • \textit{Consumer}: module that takes values as inputs (or needs to know when a value has been taken)

• Automatically inferred from the dataflow of the design
Content

• Motivation

• Twine Features
  • Standard Control Interfaces
  • High-level Operator for Composability
  • Control Coordination & Type Conversion Automation

• Implementation & Circuit Generation

• Experiments & Results

• Limitations & Future work

• Conclusion
Automate Control Coordination & Data Type Conversion

• Automatically generate system-level control logic
  • Inferred based on dataflow and producer/consumer relations
  • Mix-and-match across different interfaces
  • Ability to manually control preserved
Automate Control Coordination & Data Type Conversion

• Automatically generate system-level control logic
  • Inferred based on dataflow and producer/consumer relations
  • Mix-and-match across different interfaces
  • Ability to manually control preserved

• Data Type Conversion
  • Auto conversion between different data types (e.g., floating points <-> integers)
  • Auto conversion between different port width (useful for vectorized components)
Automate Data Type Conversion

- Simple conversion logic is combinational and transparent
  - e.g., Unsigned Integers <-> Signed Integers, Bitwidth expansion

- Complex conversion logic serves as a full converter module
  - Floating point to integer conversion
  - Serializer and de-serializer for vectorized components
Put Them Together

Assume there are modules A, B, C, and D. Module C is a vector module.

Implement one of four control interfaces →
- Predictable
- Enable high-level automation → More Reusable

Chisel Compatible
High Level Specification

in >>> A
A >>> B >>> D.1
A >>> C >>> D.2
D >>> out

Design Automation
- Control Coordination
- Data Type Conversion
Content

• Motivation
• Twine Features
• Implementation & Circuit Generation
• Experiments & Results
• Limitations & Future work
• Conclusion
Build Upon Existing Infrastructure & Preserve All Features

Chisel Utils (Pipe, etc.)

Chisel Syntax, Semantics, and Primitives

Twine Utils

Overloaded Operators

Twine Syntax, Semantics, Types, and Primitives
Twine Elaboration Pipeline

**Twine**

- **Interpret intermodular relations**
- **Insert simple data type conversion logic**
- **Insert complex data type conversion modules**
- **Analyze producer/consumer/stakeholder relations**
- **Update intermodular relation data**
- **Perform Sanity Checks**

**Original Chisel Elaboration**

**Twine Elaboration**
Content

• Motivation

• Twine Features

• Implementation & Circuit Generation

• Experiments & Results
  • Productivity Improvement Experiment
  • Design Quality Experiment

• Limitations & Future work

• Conclusion
Experiment: Productivity Improvement

- Prototyped a database query accelerator similar to Q100 (ASPLOS ‘14)
- Conducted design space exploration in Verilog, Chisel, and Twine

SQL Query

```
SELECT id, sales1+sales2, unit
FROM table
WHERE unit != 0
GROUPBY id
```

Design Translator

Hardware

Components & Filters

![Graph showing latency and area normalized to baseline for different vectorizations and memory widths](image)
Experiment: Productivity Improvement

- Much fewer lines of code (~1/3 of the designs in Chisel)
- Number of lines changed between designs is low

![Graph showing productivity improvement](graph.png)
Experiment: Design Quality

- Reproduced RISCV-MINI, a three-stage RISCV core in Twine
- Components interfaced with DecoupledIOCctrl

<table>
<thead>
<tr>
<th></th>
<th>Area*</th>
<th>Clock Period*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chisel</td>
<td>727004.94</td>
<td>0.85 ns</td>
</tr>
<tr>
<td>Twine</td>
<td>725937.90</td>
<td>0.82 ns</td>
</tr>
<tr>
<td>Change</td>
<td>-0.14%</td>
<td>-3.5%</td>
</tr>
</tbody>
</table>

*Based on IBM 45nm CMOS Process

RISCV-MINI: https://github.com/ucb-bar/riscv-mini
Content

• Motivation
• Twine Features
• Implementation & Circuit Generation
• Experiments & Results
• Limitations & Future work
• Conclusion
Limitations

• Inflexible processing granularity for vectorized modules

• Missed opportunities in inter-module optimizations
  • Possible out-of-order execution or forwarding across the module boundary
Future Research Directions

• Better verification and debugging capabilities for Twine
  • Utilize the producer/consumer relations to speed up verification process

• Flexible & customizable interface protocol framework
  • User-defined interfaces and elaboration process
Content

• Motivation
• Twine Features
• Implementation & Circuit Generation
• Experiments & Results
• Limitations & Future work
• Conclusion
Conclusion

Twine

• **Twine is a Chisel extension that supports**
  • reusable standard component control interfaces
  • high-level operator for composability
  • control coordination & data type conversion automation

• **Twine boosts developer productivity for heterogeneous designs.**
  • 1/3 of lines of codes compared to Chisel

• **Twine provides similar design quality comparing to Chisel.**

• Visit [https://github.com/Twine-Umich/Twine](https://github.com/Twine-Umich/Twine) to download Twine.
Q & A

Twine is an open-source project. To download Twine, please visit https://github.com/Twine-Umich/Twine

All feedbacks are welcomed!