



JUNE 23-27, 2024

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Bringing Digital IP Development into the 21st Century

The software development world continues to evolve at a breakneck pace, while digital hardware design is little changed since the 1990s

Warren Savage, Rocksavage Technology, Inc.

Prahlad Menon, Fraunhofer USA Center Mid-Atlantic



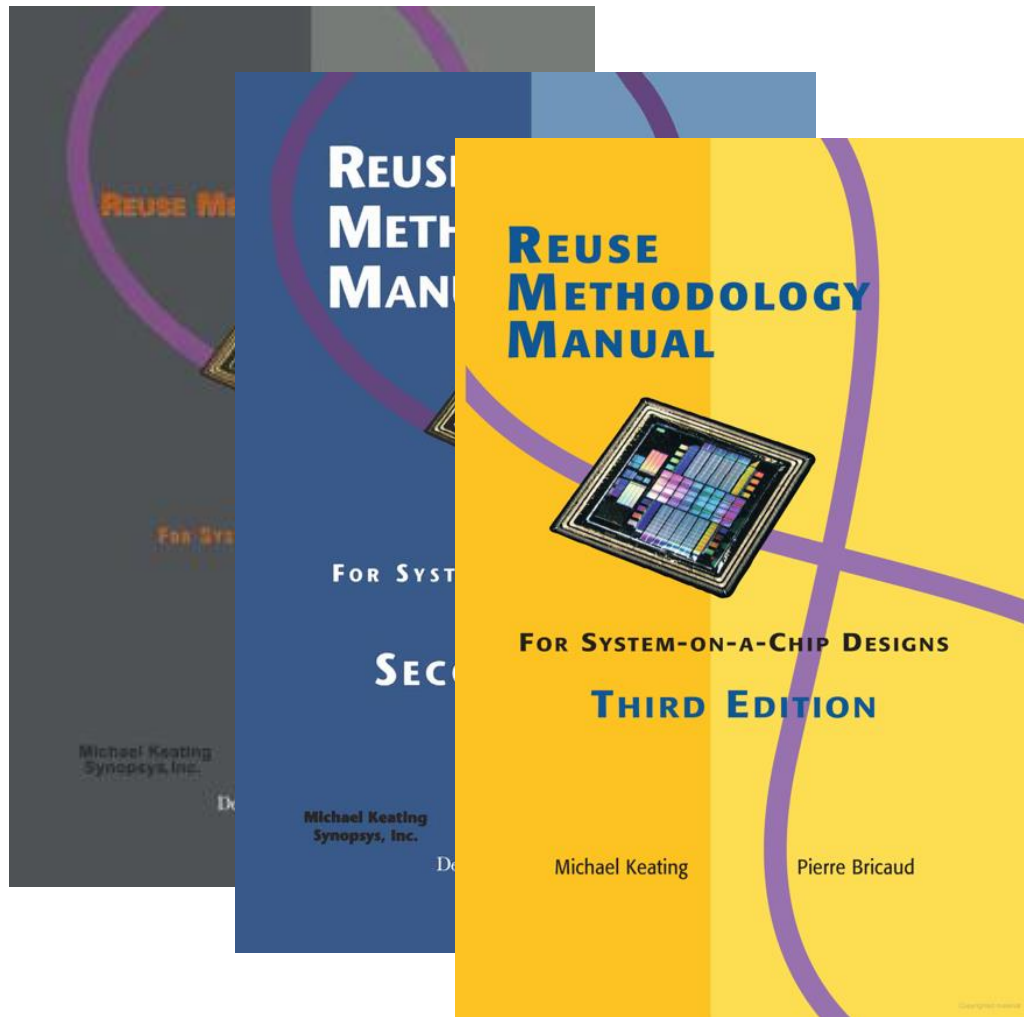
Motivation

The Reuse Methodology Manual was written 25 years ago and last updated 21 years ago

We are still designing IP the same way– with the same tools

New tools, methodologies, cloud-computing, and AI techniques can modernize the way we build the next generation of IP

Open-source EDA tools have matured to commercial viability for complex IP design



A New Paradigm – Hardware Generation

Area	Open Source Tools
Language	Scala + Chisel
IDE	VS Code
Code formatting	VS Code + scalafmt extension
Code linting	VS Code + scala extension
Code documentation	VS Code + scaladoc
Code Generation	CIRCT LLVM + FIRRTL
Build	Scala Build Tool (sbt)
Code coverage	Scala scoverage
Parameter checking	Scala assertions
Simulation	Scalatest + Chiseltest + Verilator
Verilog code coverage	Scalatest + Chiseltest + Verilator
Synthesis	Yosys + ABC
Static timing	OpenSTA
User documentation	LaTeX to PDF/HTML
CI/CD	GitHub + GitHub Actions

Source Code



RTL Generation



Verification/Analysis



Final Deliverables



Neat Things about Scala

- An expressive language
 - Procedural
 - Object-Oriented
 - Functional
- Rich set of data structures
- Code coverage
- Parameter checking
- Code coverage checks
- Self-documenting features (scaladoc)

The screenshot displays the Scala API documentation for the `DynamicFifo` package, located at `tech.rocksvage.chiselware`. The page features a sidebar on the right with a package tree showing the hierarchy: `root` → `tech` → `rocksvage` → `chiselware` → `DynamicFifo`. The main content area lists the package members, categorized into Type Members and Value Members. The Type Members section includes a `case class BaseParams` with parameters for data width, FIFO depth, external RAM, coverage, and BB files, a `class DynamicFifo` described as a synchronous FIFO and controller, a `class DynamicFifoTb`, and a `class SramBb` used as a Verilog simulation model. The Value Members section lists `object GenVerilog` extending `App` and `object Main` extending `App` to generate Verilog. A search bar at the top of the member list is labeled 'Filter all members'.

tech.rocksvage.chiselware
DynamicFifo

package **DynamicFifo**

Filter all members

Type Members

case class **BaseParams**(dataWidth: Int = 8, fifoDepth: Int = 8, externalRam: Boolean = false, coverage: Boolean = false, bbFiles: List[String] = List("your_sram.v"))
Default parameter settings for Dynamic FIFO

class **DynamicFifo**
A synchronous FIFO and FIFO controller with dynamic flags

class **DynamicFifoTb**

class **SramBb**
Blackbox to hold Verilog simulation model

Value Members

object **GenVerilog** extends App

object **Main** extends App
Generate Verilog

Packages
root
tech
rocksvage
chiselware
DynamicFifo
BaseParams
DynamicFifo
DynamicFifoTb
GenVerilog
Main
SramBb

Neat Things about Chisel

- Concise hardware semantics
- Correct-by-construction Verilog
- In-lining Verilog

Verilog

```
always @(posedge clock) begin
  if (reset) begin
    state <= 0;
  end else if (enable) begin
    state <= next;
  end
end
```

Chisel

```
val state = RegInit(0)

when(enable) {
  state := next
}
```

Neat Things about Chisel Test

- Simple peek/poke/expect primitives allow building of arbitrarily complex functions
- Fork/join allows for threading to simulate real-world operating scenarios
- Randomization features allow for constrained random verification test cases
- Native support for
 - Verilator
 - iVerilog
 - VCS

```
testDataBuffer.foreach { data =>
  fork {
    if (!isFull()) {
      if (math.random() < 0.75) {
        expectedData.enqueue(data)
        stack.push(1)
        push(data)
        checkFlags()
      }
    }
  }.fork {
    if (!isEmpty()) {
      if (math.random() < 0.55) {
        val expected = expectedData.dequeue()
        stack.pop()
        pop(expected)
        checkFlags()
      }
    }
  }.join()
```

```
// Randomize input variables
val validDataWidths = Seq(4, 5, 6, 7, 8, 16, 32, 64, 128)
val validFifoDepths = Seq(4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048)
val dataWidth       = validDataWidths(Random.nextInt(validDataWidths.length))
val fifoDepth       = validFifoDepths(Random.nextInt(validFifoDepths.length))
val useExternalRam  = (fifoDepth * dataWidth) > 1024
val testDataSize    = fifoDepth * Random.between(10, 20)
val fullThreshold   = (fifoDepth * Random.between(75, 95) / 100).toInt
val emptyThreshold  = (fifoDepth * Random.between(3, 25) / 100).toInt
```

> A Seamless Integrated Flow

Chisel Source Code

Generated Verilog

Post-Synthesis Analysis

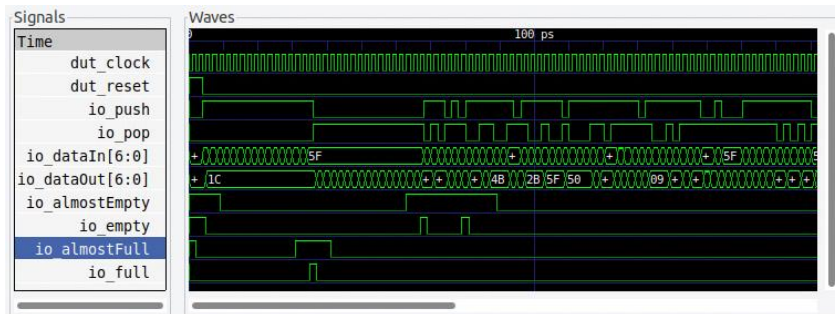
```
196
197 /** Generate Verilog */
198 object Main extends App {
199   val myParams = BaseParams(
200     externalRam = true, dataWidth = 128, fifoDepth = 32)
201   ChiselStage.emitSystemVerilog(
202     new DynamicFifo(myParams),
203     firtoolOpts = Array(
204       "--lowering-options=disallowLocalVariables",
205       "--disallowPackedArrays",
206       "--disable-all-randomization", "--strip-debug-info",
207       "--verilog", "--split-verilog", "-o=generated")
208   )
209 }
```

```
1 // Generated by CIRCT firtool-1.47.0
2 module DynamicFifo(
3   input      clock,
4   input      reset,
5   input      io_push,
6   input      io_pop,
7   input [127:0] io_dataIn,
8   input [4:0]  io_almostEmptyLevel,
9   input [127:0] io_ramDataOut,
10  output [127:0] io_dataOut,
11  output      io_empty,
12  output      io_full,
13  output      io_almostEmpty,
14
```

```
[tws:.../chiselWare/dynamicfifo/syn]$ cat area.rpt timing.rpt
small_false_8_8 = 769 gates
medium_false_32_64 = 19283 gates
large_false_64_256 = 152808 gates
small_true_64_256 = 355 gates
medium_true_128_128 = 477 gates
large_true_256_2048 = 502 gates
small_false_8_8 = 2.93 slack (MET)
medium_false_32_64 = 2.69 slack (MET)
large_false_64_256 = 2.80 slack (MET)
small_true_64_256 = 2.80 slack (MET)
medium_true_128_128 = 2.70 slack (MET)
large_true_256_2048 = 2.77 slack (MET)
[tws:.../chiselWare/dynamicfifo/syn]$
```

Constrained Random Verification

Scala Code Coverage Analysis



Class	Source file	Lines	Methods	Statements	Invoked	Coverage	Branches	Invoked	Coverage
BaseParams	BaseParams.scala	42	1	6	6	100.00 %	0	0	100.00 %
DynamicFifo	DynamicFifo.scala	193	4	211	210	99.53 %	4	4	100.00 %
DynamicFifoTb	DynamicFifoTb.scala	46	4	102	101	99.02 %	0	0	100.00 %
GenVerilog	GenVerilog.scala	50	1	40	40	100.00 %	0	0	100.00 %
Main	DynamicFifo.scala	212	1	18	18	100.00 %	0	0	100.00 %
SramBb	SramBb.scala	28	4	56	55	98.21 %	0	0	100.00 %

Automated Regressions in Cloud



GitHub Actions

✓ create-vm

2m 55s

✓ build-and-test

1m 45s

✓ delete-vm

20s

Documentation: Just More Code

LaTeX

```
\subsection{Interface Timing}

DynamicFifo has a simple, synchronous interface. The timing diagram shown below
in Figure~\ref{fig:timing} represents an instantiation with the following
parameters.

\begin{lstlisting}[language=Scala]
val myFifo = new DynamicFifo(
  externalRAM = true,
  dataWidth = 16,
  fifoDepth = 5)
\end{lstlisting}

You, 7 months ago | 1 author (You)

\begin{figure}[h]
  \includegraphics[width=\textwidth]{images/timing.png}
  \caption{Timing Diagram}\label{fig:timing}
\end{figure}

The \textit{almostEmptyLevel} port is driven by external logic to a static value
of 1 after reset and the \textit{almostFull} port is driven to 3.

Beginning in the third clock cycle, 5 words of data are pushed into the
FIFO. The status flags show the FIFO going from empty to full.

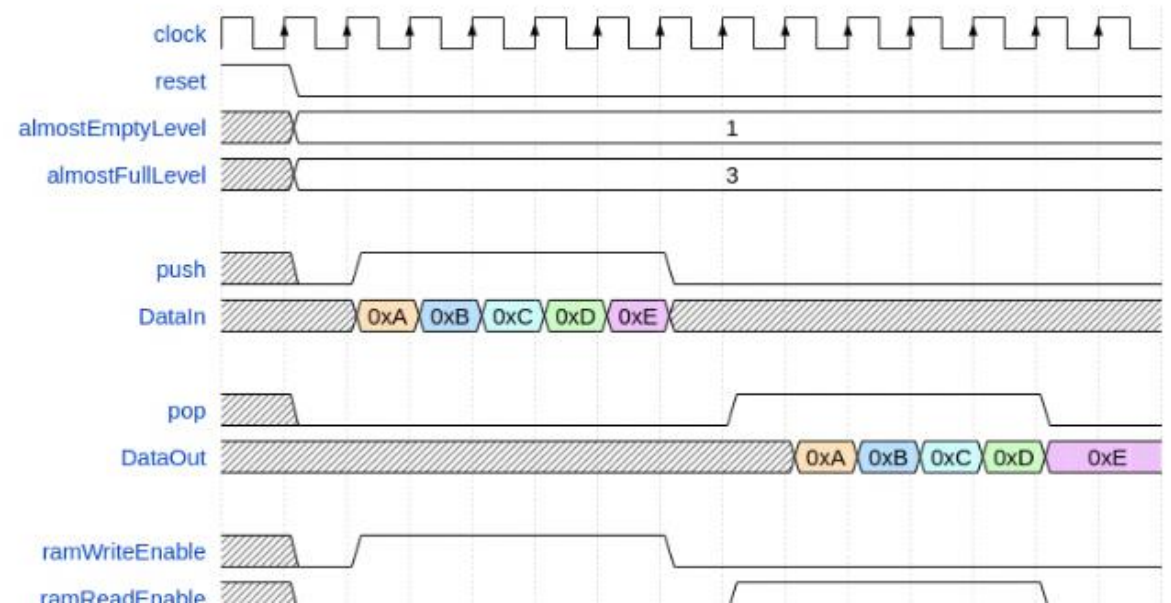
The FIFO is then fully emptied when the \textit{pop} port is held high for
5 clock cycles. The status flags show the FIFO going from full to empty again.
```

PDF (or HTML)

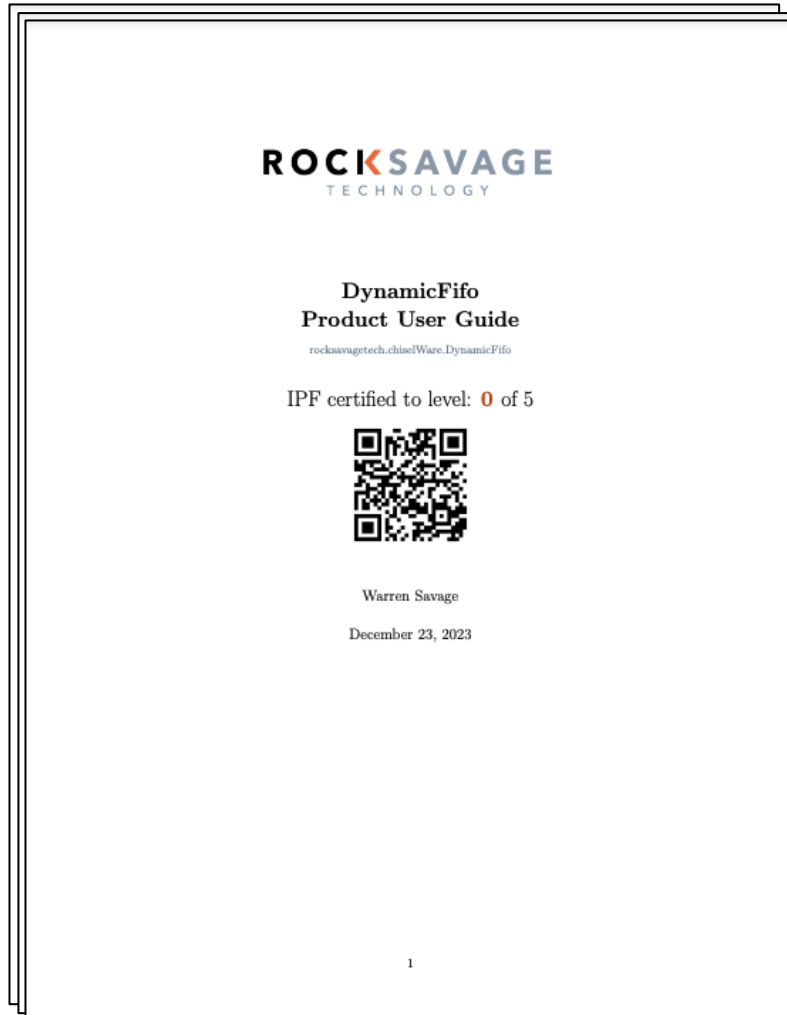
4.2 Interface Timing

DynamicFifo has a simple, synchronous interface. The timing diagram shown below in Figure 2 represents an instantiation with the following parameters.

```
val myFifo = new DynamicFifo(
  externalRAM = true,
  dataWidth = 16,
  fifoDepth = 5)
```



Gen AI: Automated Design Reviews



Automated checks can be coded for:

- Completeness of deliverables
- Confirming marketing claims against what is actually in the code
- Providing feedback on user documentation

Early results from

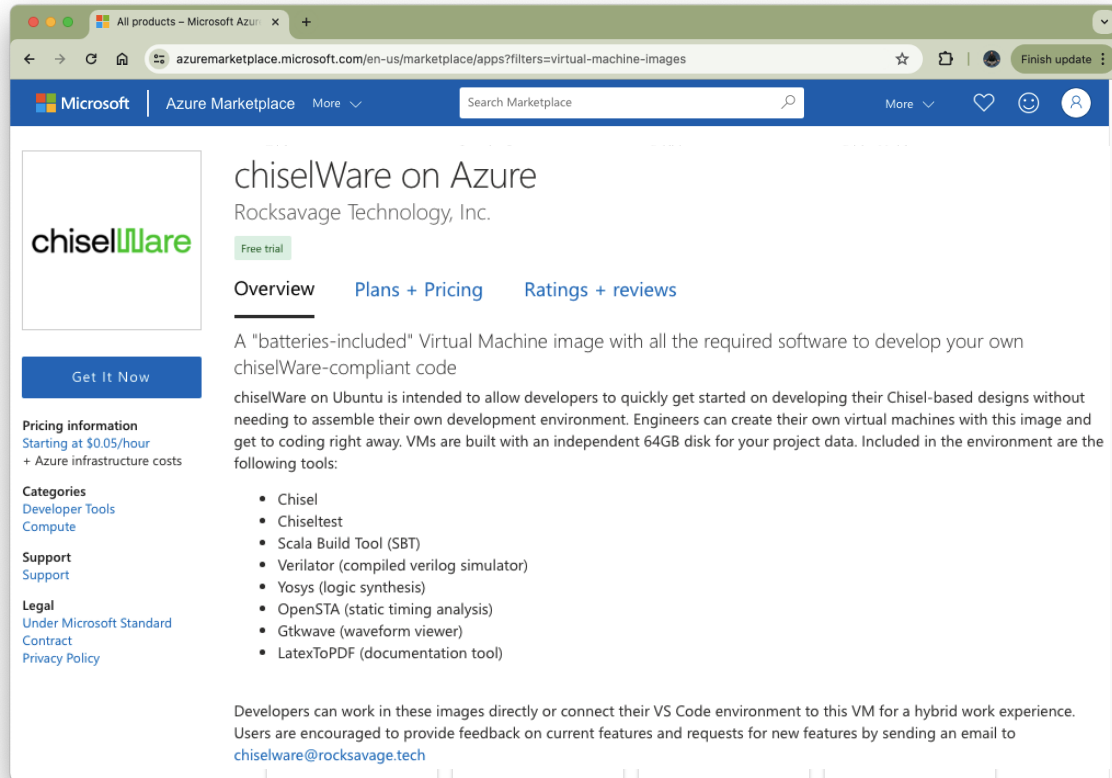


Analysis of Documentation

- **Comprehensiveness:** The guide appears to cover critical areas like port and parameter descriptions, operational theory, simulation, and synthesis. This indicates a comprehensive approach to documentation.
- **Errata and Known Issues:** Including a section on errata and known issues is excellent for transparency. It's crucial that these points are regularly updated and clearly connected to the relevant parts of the guide.
- **Technical Depth:** The guide seems to delve into technical details, which is essential for a user manual. It would be important to ensure that these sections are accessible to users with varying levels of expertise.
- **Simulation and Synthesis:** These sections are particularly valuable for users to understand the practical aspects of using DynamicFifo, especially for complex tasks.
- **Visual Aids and Examples:** The effectiveness of the documentation could be enhanced by including visual aids, diagrams, and real-world examples, though I cannot confirm their presence from the text alone.

Getting Started in the Cloud

Microsoft Azure Marketplace



The screenshot shows the Azure Marketplace page for "chiselWare on Azure" by Rocksavage Technology, Inc. The page includes a "Free trial" badge, tabs for "Overview", "Plans + Pricing", and "Ratings + reviews". The "Overview" tab is active, displaying a description of the "batteries-included" Virtual Machine image. It lists the tools included in the environment: Chisel, Chiseltest, Scala Build Tool (SBT), Verilator, Yosys, OpenSTA, Gtkwave, and LatexToPDF. Pricing information indicates it starts at \$0.05/hour plus Azure infrastructure costs. Categories include Developer Tools and Compute. Support and legal information are also provided.

chiselWare on Azure
Rocksavage Technology, Inc.

Free trial

Overview Plans + Pricing Ratings + reviews

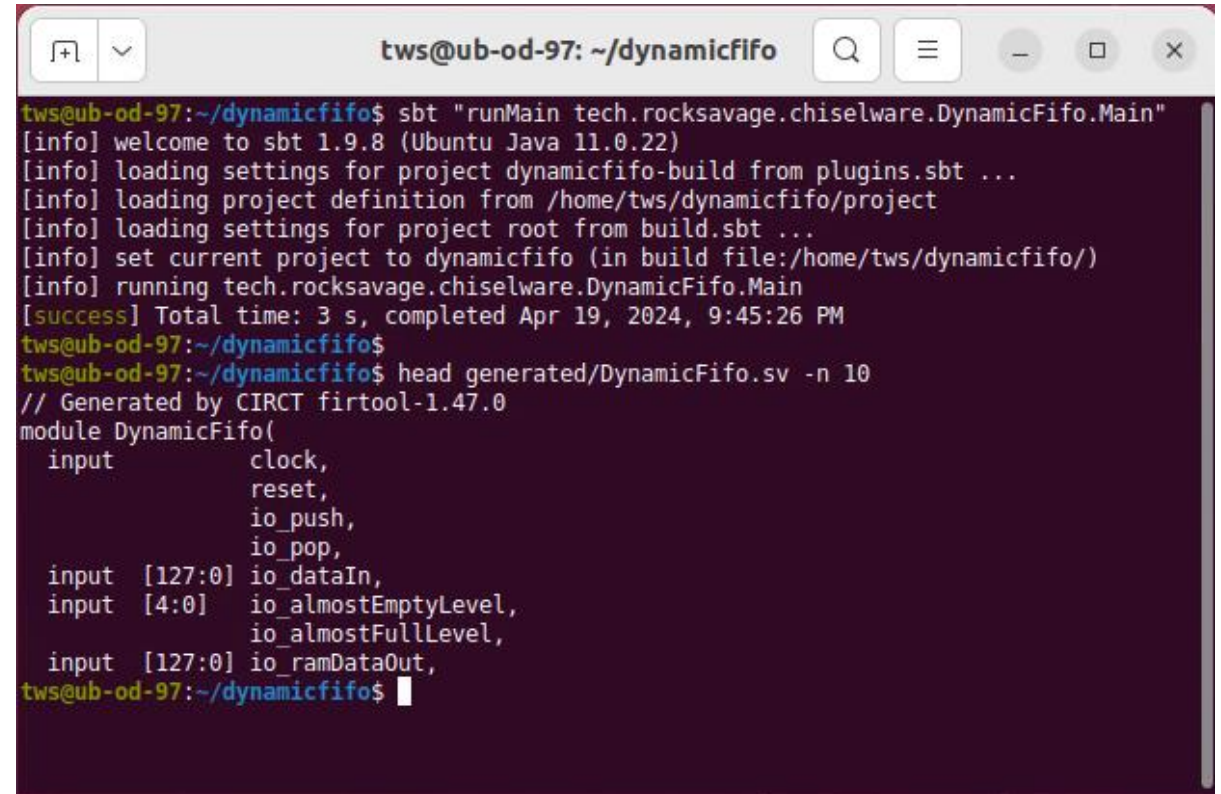
A "batteries-included" Virtual Machine image with all the required software to develop your own chiselWare-compliant code

chiselWare on Ubuntu is intended to allow developers to quickly get started on developing their Chisel-based designs without needing to assemble their own development environment. Engineers can create their own virtual machines with this image and get to coding right away. VMs are built with an independent 64GB disk for your project data. Included in the environment are the following tools:

- Chisel
- Chiseltest
- Scala Build Tool (SBT)
- Verilator (compiled verilog simulator)
- Yosys (logic synthesis)
- OpenSTA (static timing analysis)
- Gtkwave (waveform viewer)
- LatexToPDF (documentation tool)

Developers can work in these images directly or connect their VS Code environment to this VM for a hybrid work experience. Users are encouraged to provide feedback on current features and requests for new features by sending an email to chiselware@rocksavage.tech

Azure VM preconfigured with software



The screenshot shows a terminal window on an Azure VM. The user runs the command `sbt "runMain tech.rocksavage.chiselware.DynamicFifo.Main"`, which successfully executes the program. The output shows the SBT version (1.9.8) and the project name (dynamicfifobuild). The user then runs `head generated/DynamicFifo.sv -n 10`, which displays the first 10 lines of the generated Verilog code.

```
tws@ub-od-97: ~/dynamicfifobuild
tws@ub-od-97:~/dynamicfifobuild$ sbt "runMain tech.rocksavage.chiselware.DynamicFifo.Main"
[info] welcome to sbt 1.9.8 (Ubuntu Java 11.0.22)
[info] loading settings for project dynamicfifobuild from plugins.sbt ...
[info] loading project definition from /home/tws/dynamicfifobuild/project
[info] loading settings for project root from build.sbt ...
[info] set current project to dynamicfifobuild (in build file:/home/tws/dynamicfifobuild/)
[info] running tech.rocksavage.chiselware.DynamicFifo.Main
[success] Total time: 3 s, completed Apr 19, 2024, 9:45:26 PM
tws@ub-od-97:~/dynamicfifobuild$ head generated/DynamicFifo.sv -n 10
// Generated by CIRCT firtool-1.47.0
module DynamicFifo(
  input          clock,
                reset,
                io_push,
                io_pop,
  input [127:0] io_dataIn,
  input  [4:0]  io_almostEmptyLevel,
                io_almostFullLevel,
  input [127:0] io_ramDataOut,
tws@ub-od-97:~/dynamicfifobuild$
```

Get Started with a Working Project



<https://github.com/rocksavagetechnology/dynamicfifobuild>

A Vision: chiselWare

- **Objective:** Create a library of high-quality, useful IP cores that people want to use
- Step 1: Define what it means to be chiselWare-compliant
- Step 2: Establish a scalable infrastructure for designing and maintaining cores
- Step 3: Publish meaningful projects for adoption to a developer community
- Step 4: Curate submitted cores
- Step 5: Release curated cores
- Step 6: Support end-users of these cores
- Interested in being part of this project? Email: chiselware@rocksavage.tech