Chiffre: A Configurable Hardware Fault Injection Framework for RISC-V Systems

Schuyler Eldridge Alper Buyuktosunoglu Pradip Bose

2nd Workshop on Computer Architecture Research with RISC-V

Motivation: Selective Latch Hardening

Given a hardware design, which latches require hardening?

- All of them? ..... *Power/Area expensive!*
- Some of them? ..... *Difficult to find!*

A closed-form solution is impractical, however, empirical approaches that inject directly into a hardware description work well:

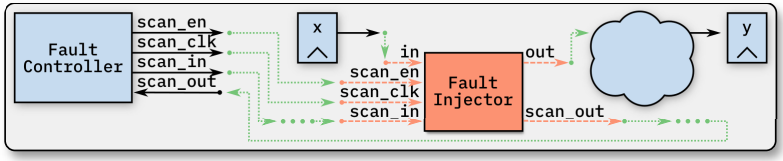
- A manually instrumented Illinois Verilog Model [1]
- The CLEAR methodology [2]

However, there is no existing automated, open-source workflow for enabling injection experiments at speed.

[1] M. Maniatakos, N. Karimi, C. Tirumurti, A. Jas, and Y. Makris. Instruction-level impact analysis of low-level faults in a modern microprocessor controller. *IEEE Trans. Computers*, 60(9):1260–1273, 2011.

[2] E. Cheng, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, et al. Clear: Cross-layer exploration for architecting resilience: Combining hardware and software techniques to tolerate soft errors in processor cores. In *Design Automation Conference (DAC)*, 2016.

Manual or Automated Verilog Instrumentation is Doable...




```
module MyModule( input clk,
                 , input scan_en, input scan_clk
                 , input scan_in, output scan_out
                 );
  reg x, y;
  wire not_x;
  wire x_faulty;
  MyInjector injector( .clock(clock), .in(x), .out(x_faulty),
                       .scan_en(scan_en), .scan_clk(scan_clk),
                       .scan_in(scan_in), .scan_out(scan_out) );
  assign not_x = ~x_faulty; // Right hand side replacement
  always @(posedge clk) begin
    y <= not_x;
  end
endmodule
```

… Except for All the Corner Cases!

- `genvar i; generate`
- `` `ifdef MYDEFINE ``
- `task`, `function`
- `disable`
- Name collisions

The difficulty lies in the complexity of the language

In software, we'd just use a compiler…

LLVM – A Modern Compiler

1. Frontend converts a high level language to an Intermediate Representation
2. The LLVM compiler optimizes the program
3. A backend emits a binary

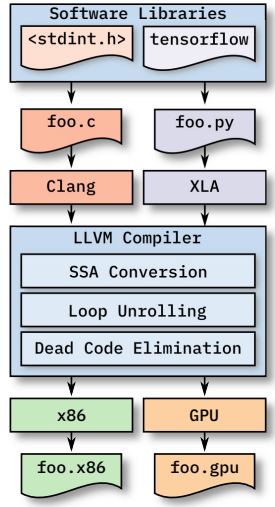


Figure: LLVM Compiler Flow

FIRRTL – A Modern Circuit Compiler (Emitting Verilog)

1. Frontend converts a high level language to the FIRRTL IR[1]
2. The FIRRTL compiler optimizes the circuit
3. A backend emits Verilog

[1] P. S. Li, A. M. Izraelevitz, and J. Bachrach. Specification for the firrtl language. Technical Report UCB/EECS-2016-9, EECS Department, University of California, Berkeley, Feb 2016.

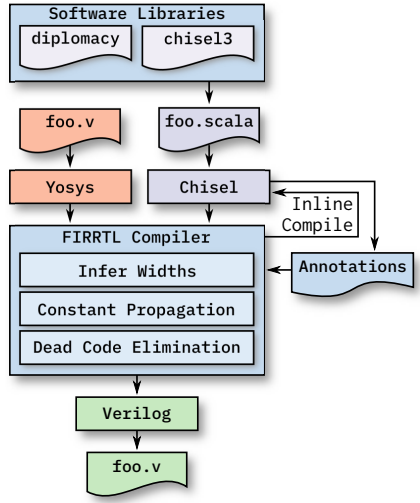


Figure: LLVM Compiler Flow

Chiffre– Automated Fault Injection Instrumentation

1. Chisel [1] frontend emits fault annotations
2. FIRRTL transforms add fault injectors, scan chain
3. The existing backend emits Verilog

[1] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović. Chisel: Constructing hardware in a scala embedded language. In *49th ACM/EDAC/IEEE Design Automation Conference (DAC), 2012*.

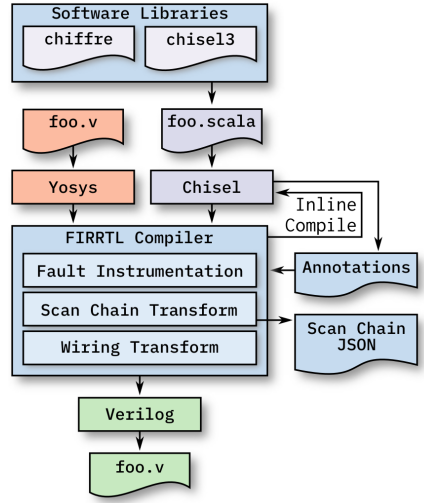
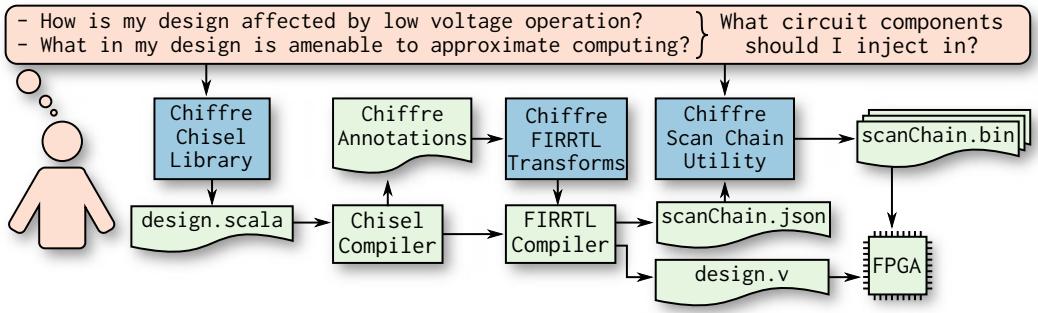


Figure: Chiffre Flow

Chiffre Framework Overview

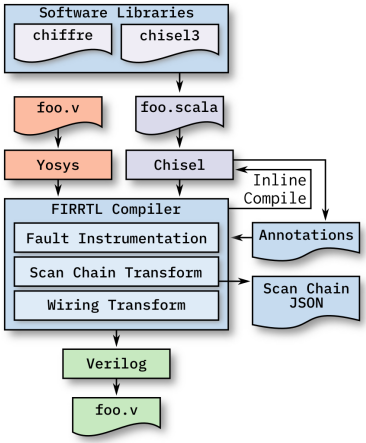


1. Chiffre Chisel library
2. Chiffre FIRRTL Transforms
3. Chiffre Scan Chain Utility

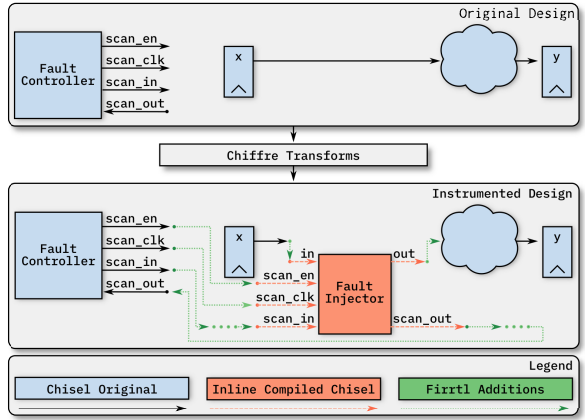
Chiffre Chisel Library

```
import chisel3._
import chiffre._

/* A controller for injectors on the "main" scan chain */
class MyController extends Module with ChiffreController {
  val io = IO(new Bundle{})
  lazy val scanId = "main"
  /* MyController body with scan chain logic not shown */
}

/* A module with faulty components */
class MyModule extends Module with ChiffreInjectee {
  val io = IO(new Bundle{})
  val x = Reg(UInt(1.W))
  val y = Reg(UInt(4.W))
  val z = Reg(UInt(8.W))
  isFaulty(x, "main", classOf[inject.LfsrInjector32])
  isFaulty(y, "main", classOf[inject.StuckAt])
  isFaulty(z, "main", classOf[inject.CycleInjector32])
}
```


Chiffre FIRRTL Transforms



(a) Chiffre Flow



(b) Chiffre Instrumentation

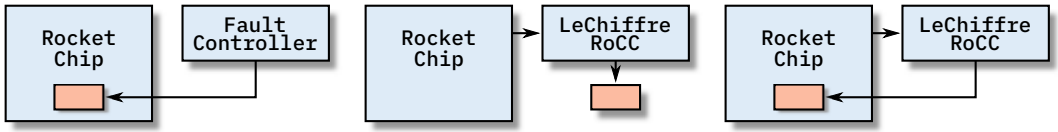
More details in the paper...

Chiffre Scan Chain Utility

Given a scan chain description, set all configurable fields

More details in the paper...

Fault Injection Experiments Enabled by Chiffre



(a) Internal injection (b) External injection (c) Self injection

We provide an example RoCC injector, **`LeChiffre`**, in the Chiffre Chisel Library

Example of Self Injection: Rocket “Jailbreak”

Use **LeChiffre** to self-inject into the `cycle` CSR

```
;;; Test that passes if the cycle counter decreases
;;; (This violates the RISC-V specification!)
test:
  rdcycle a0
  rdcycle a1
  bgt a0, a1, pass
  goto fail

pass:
  li a0, 0
  j done
fail:
  li a0, 1

done:
  ret
```

More information in our open-source test: `faulty-cycle.S`

Future Work

- SRAM injection
- Improved scan chain (or network), like Strober [1], MIDAS [2], FireSim [3]
- Fault injection experiment automation
- Recording/quantifying injection effects

[1] D. Kim, A. Izraelevitz, C. Celio, H. Kim, B. Zimmer, Y. Lee, J. Bachrach, and K. Asanović. Strober: fast and accurate sample-based energy simulation for arbitrary rtl. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, 2016.

[2] D. Kim, C. Celio, D. Biancolin, J. Bachrach, and A. Krste. Evaluation of risc-v rtl with fpga-accelerated simulation. In *1st Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2017.

[3] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra, Q. Huang, K. Kovacs, B. Nikolic, R. Katz, J. Bachrach, and K. Asanović. Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud. In *Proceedings of the 45th International Symposium on Computer Architecture (ISCA)*, 2018.

Open-source Availability

Chiffre is open-source (alpha) software/hardware!

Apache v2 License

`https://github.com/IBM/chiffre`

Acknowledgments

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. This document is Approved for Public Release, Distribution Unlimited.