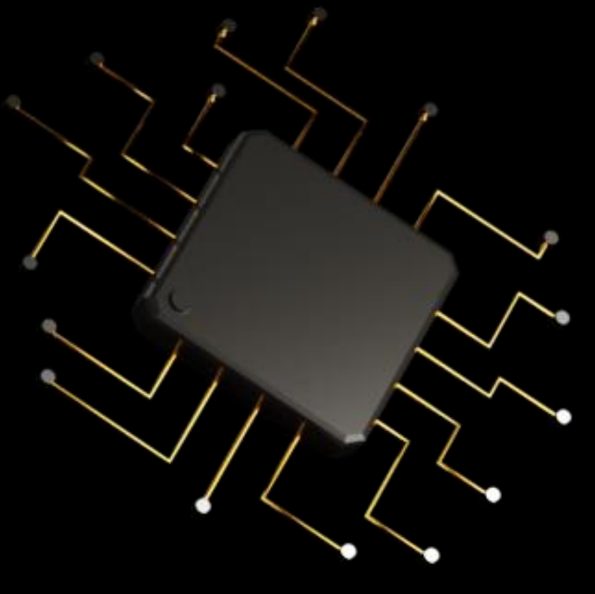
RISC-V SoC Hardware Vulnerability Detection Toolset



Team 41

Mason Korkowski, Micah Mundy, Gerald Edeh, Kolton Keller, Eva Kohl, Savva Zeglin, Magnus Anderson

Client/Advisor: Henry Duwe

What is HACK@DAC?

- Hardware Capture the Flag Competition
- Teams compete to find flaws in a buggy SoC design
- Encourages creation of automated bug detection tools
- Promotes security in hardware

Problem

- System on Chip (SoC) designs are becoming more and more prevalent.
- More advanced SoCs → higher complexity → more vulnerabilities
- The problem of detecting these vulnerabilities is *especially* challenging.
- Project goal: Construct a toolset that can be used for hardware capture-the-flag competitions such as HACK@DAC

Final Product

Toolset consisting of:

Signal Tracer

- Quickly determine signal dependencies without manually searching files

Signals That Never Change Detector

- Easily detect unchanging signal bugs in SoC design

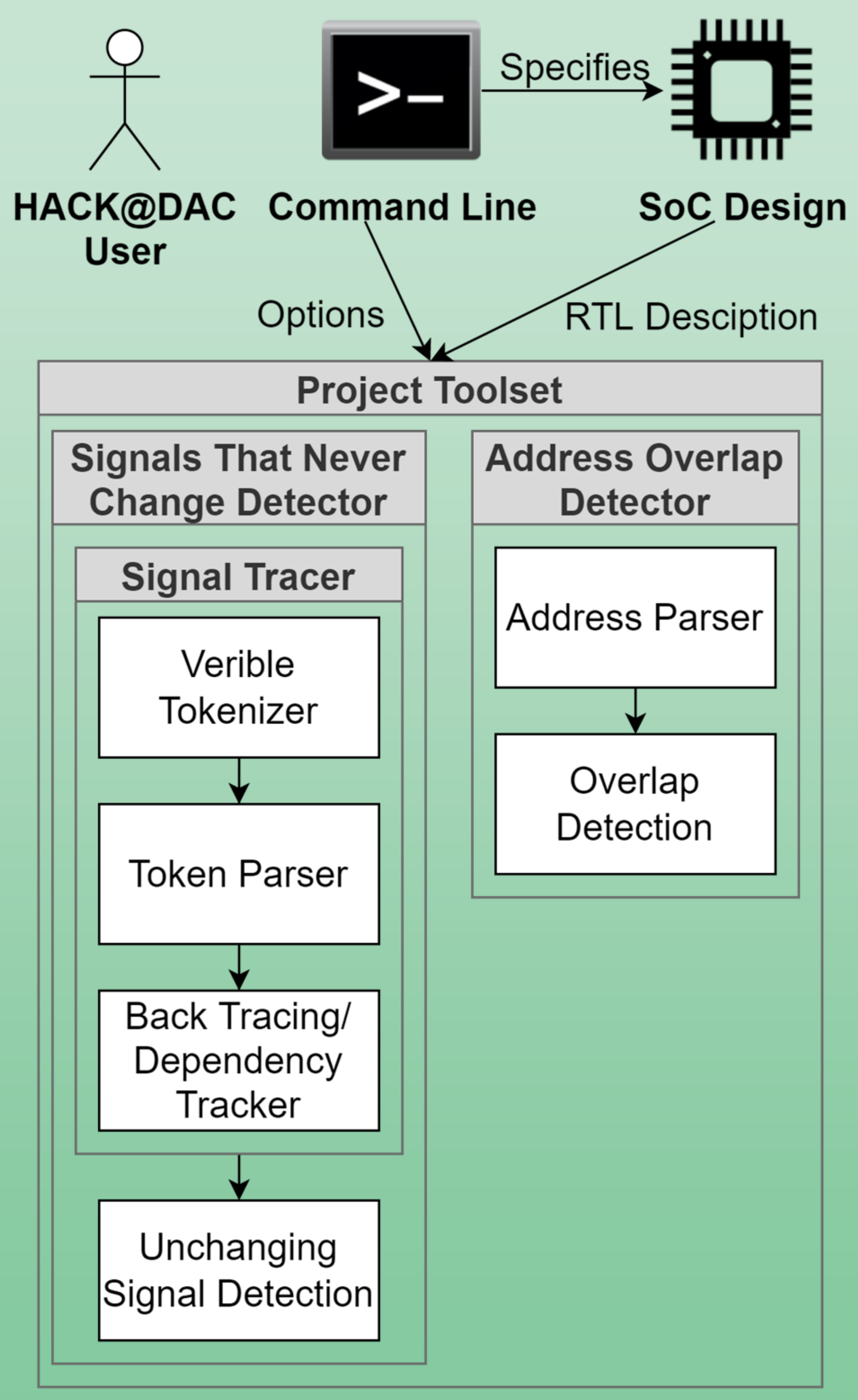
Address Overlap Detector

- Swiftly locate overlapping peripheral addresses

Design Requirements

- Find hardware-induced vulnerabilities in the RTL implementations of SoC designs
- Accessible via command line
- Run on a virtual machine with access to 4 cores and 16GB RAM
- Follow all HACK@DAC rules and regulations

Concept Diagram



Users and Uses

- Primary Users:
  - HACK@DAC participants
  - First line of attack
  - Reduce time to detect/exploit bugs
- Secondary Users:
  - Hardware Developers
  - First line of defense
  - Signal tracing and bug detection

Project Resources

- Google Drive Tools Suite
- Gitlab
- Software IDE(s)
- Questasim / other hardware simulation software
- Virtual machine for simulation, other tasks
- Personal computers for use of above tools and software

Testing

- Unit Testing:
  - Junit was used on the Java programs
  - Each program was tested on mock data and bug injection.
- System Testing:
  - Address Overlap Detector: Verified through injected bugs.
  - The Signals That Never Change Detector: Tested with a custom-made bug injector. Over 60% of injections found successfully

Signal Tracer

- Input:
  - Directory of SoC RTL design, top level module name, signal to trace
- Verible
  - External library which tokenizes Verilog Files
- Token Parser / Back Tracing / Dependency Tracker
  - Java Program
  - The tokens from Verible are parsed in order to locate the given signal
  - Each signal/module the signal depends on is used to build a dependency hierarchy
- Output:
  - Tree structure which describes every dependency of the signal

Address Overlap Detector

- Input:
  - Peripheral File of SoC design
- Address Parser
  - Bash Script
  - Core address configuration files are evaluated and output to a file
- Overlap Detection
  - Java program
  - Analyzes output from Address Parser
  - Detects any overlapping addresses
- Output:
  - Information regarding address overlap findings

Signals That Never Change Detector

- Input:
  - Directory of SoC RTL design, top level module name
- Unchanging Signal Detection
  - Java Program
  - Use Signal Tracer for each signal in the design
    - Determines if signal's dependencies are such that it will never change
- Output:
  - Information to user about signals whose value is unchanging