THE UNIVERSITY OF TEXAS AT DALLAS

Department of Electrical and Computer Engineering

Making Legacy Hardware Robust against Side Channel Attacks via High-Level Synthesis

Md Imtiaz Rashid, and **Benjamin Carrion Schaefer**

Mdimtiaz.rashid@utallas.edu, **schaferb@utdallas.edu**

ASIA SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE

30th Asia and South Pacific Design Automation Conference

ASP-DAC 2025

January 21, 2025, Tokyo

Outline

- Introduction
  - What, why and how?
- Background information
  - High-Level Synthesis
    - How does it work?
    - RTL to C for HLS compiler and modernization methodology overview
  - Side Channel Attacks (SCAs)
- Proposed Flow
  - SCA-aware RTL to C compiler
  - Security-aware HLS design space exploration
- Experiments
  - Experimental Setup
  - Experimental Results
- Conclusion

Introduction

What ?

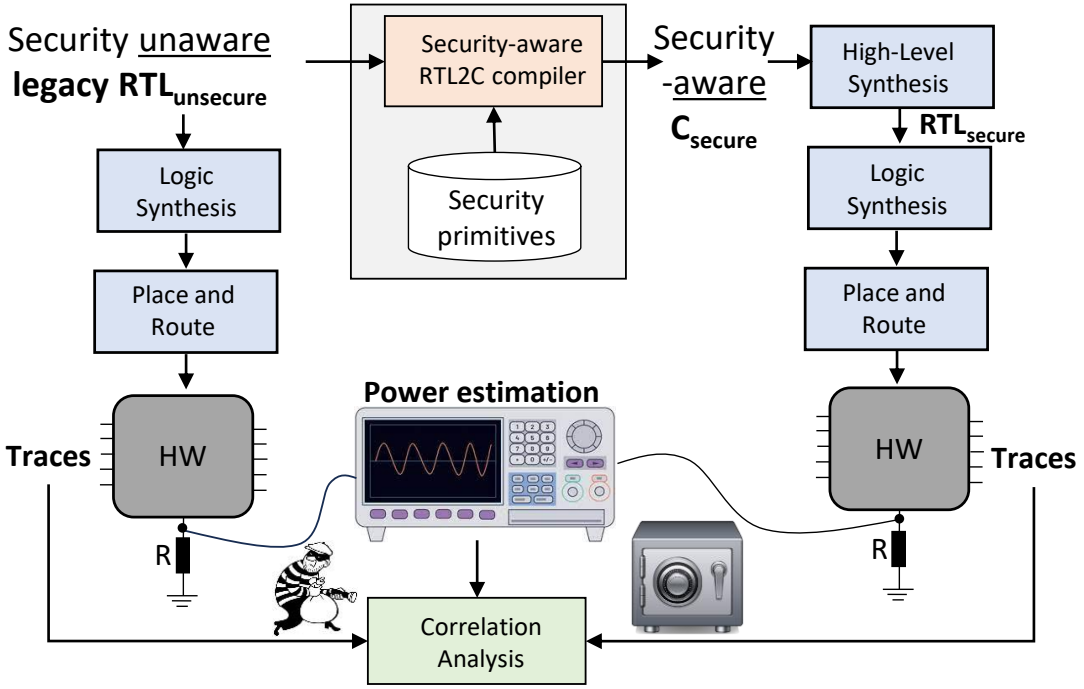
Legacy RTL code (Verilog or VHDL) is mostly **HW security unaware** → Need design flows to "modernize" this legacy RTL

Why ?

To protect older HW assets, e.g., from the military that can updated infrequently

How ?

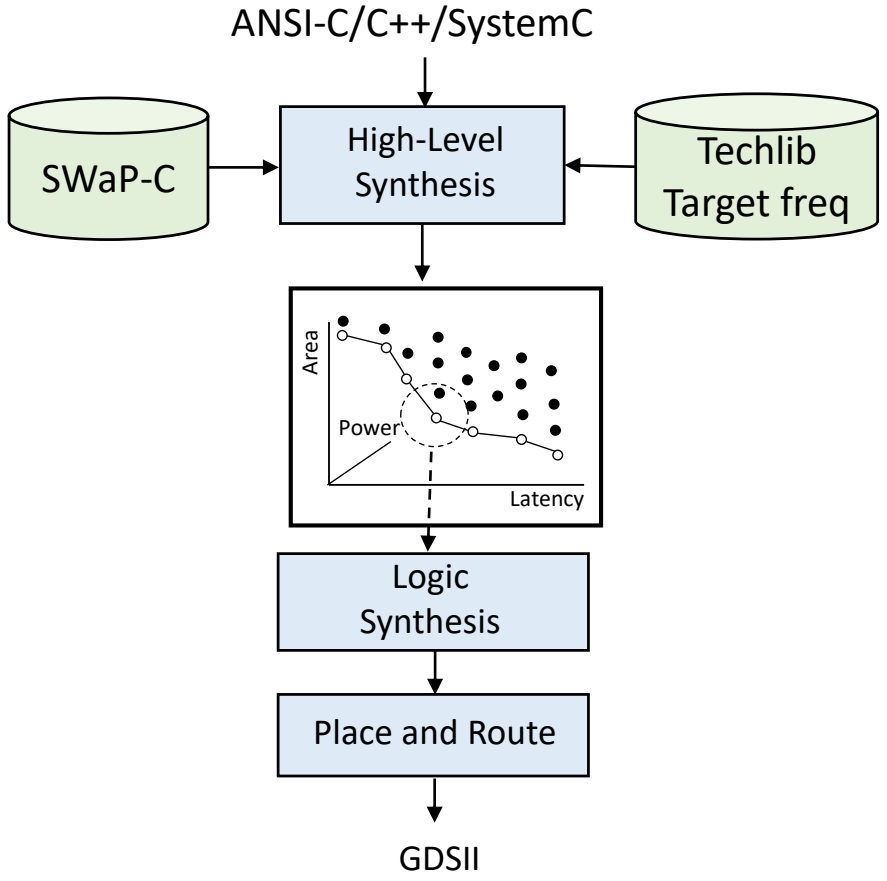
Through an RTL to C compiler coupled with security primitives at the behavioral level



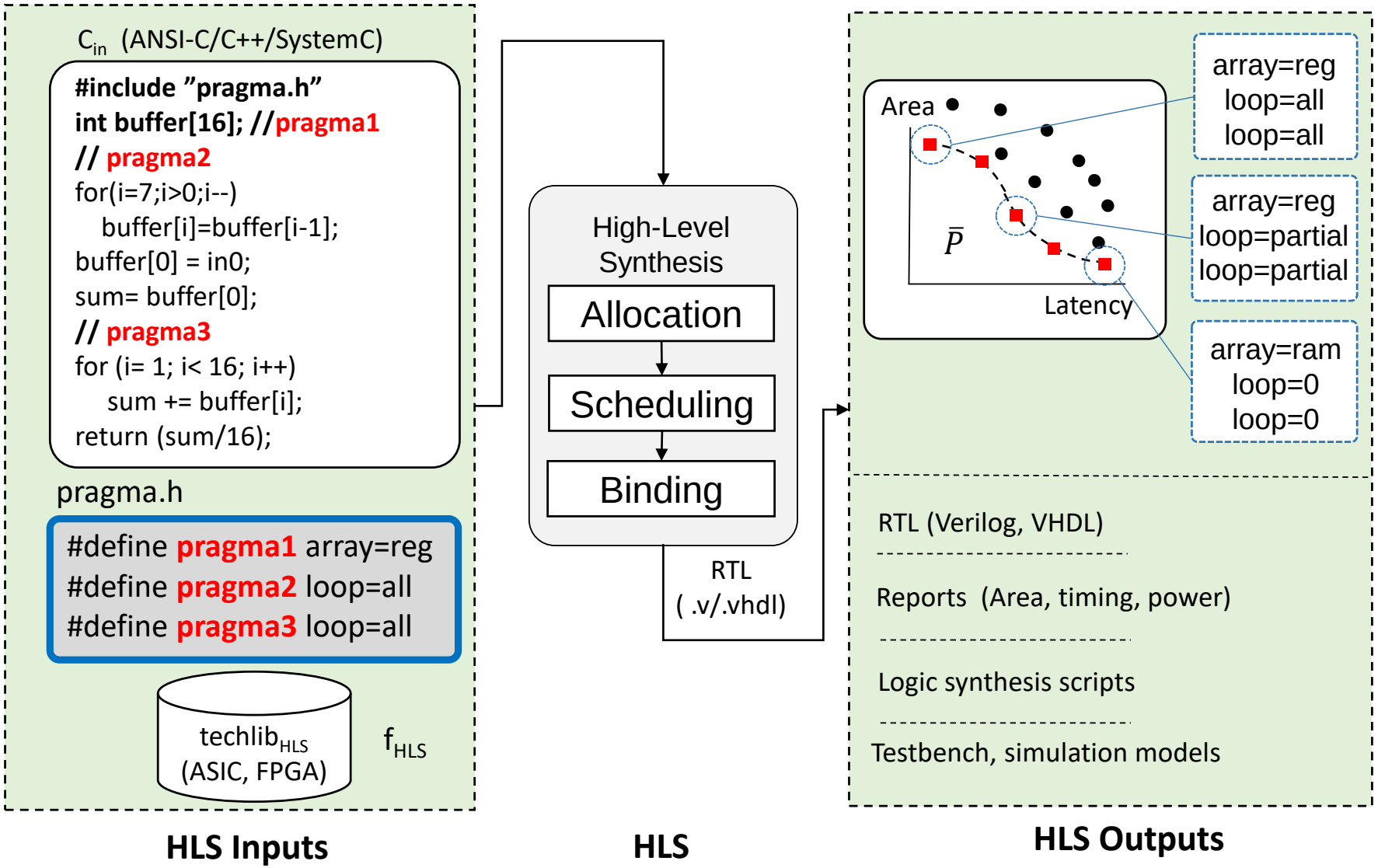
High-Level Synthesis

- **High-Level Synthesis**
  - Design circuits using software languages
  - Definition:

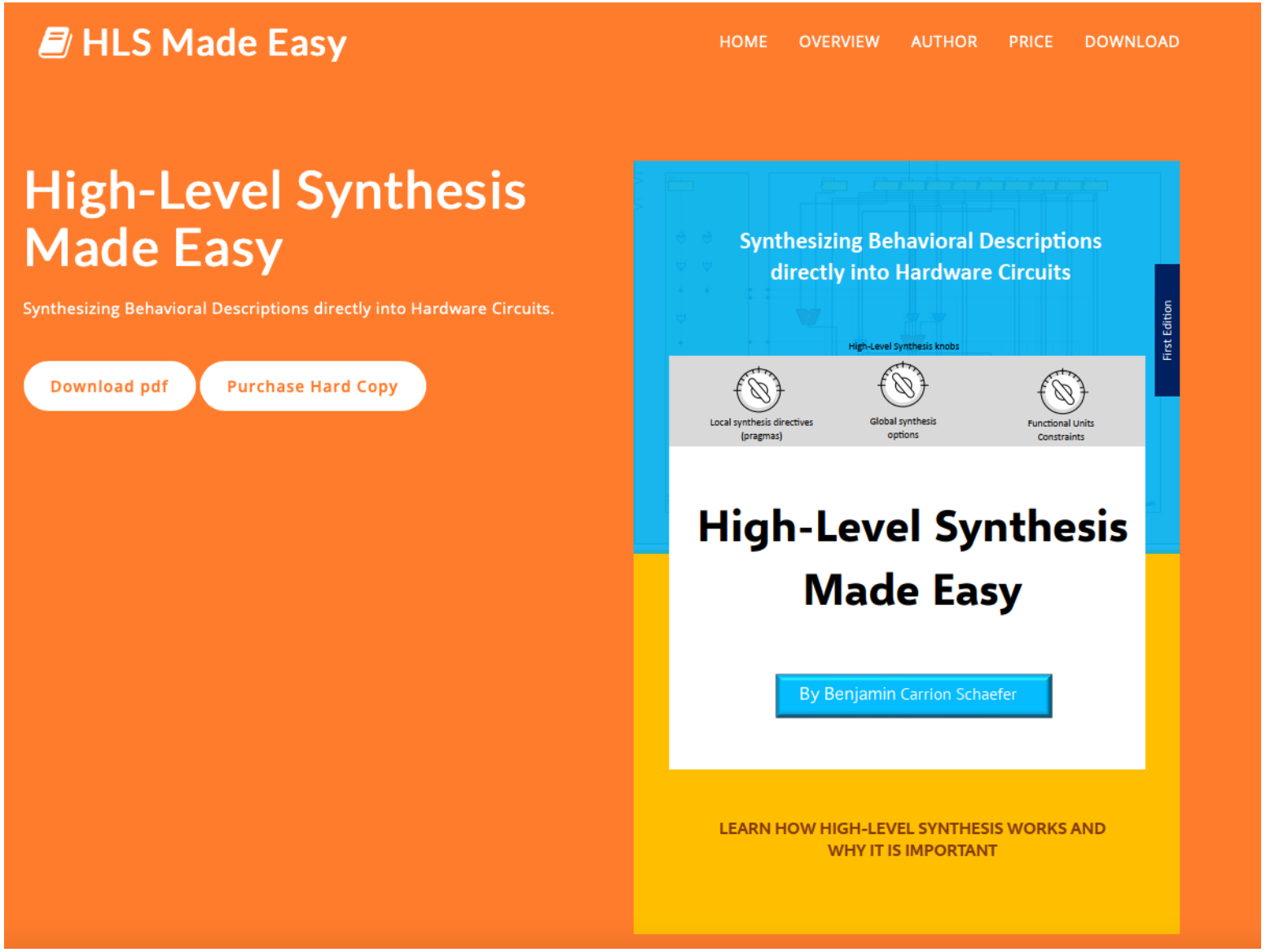
    "Automatic conversion of a behavioral, untimed description into efficient hardware that implements that behavior"
  - Benefits
    1. Software programmability and hardware performance
    2. Faster verification
    3. Allows to easily re-target any behavioral description to new technologies and newer design constraints (i.e., area, power, performance)



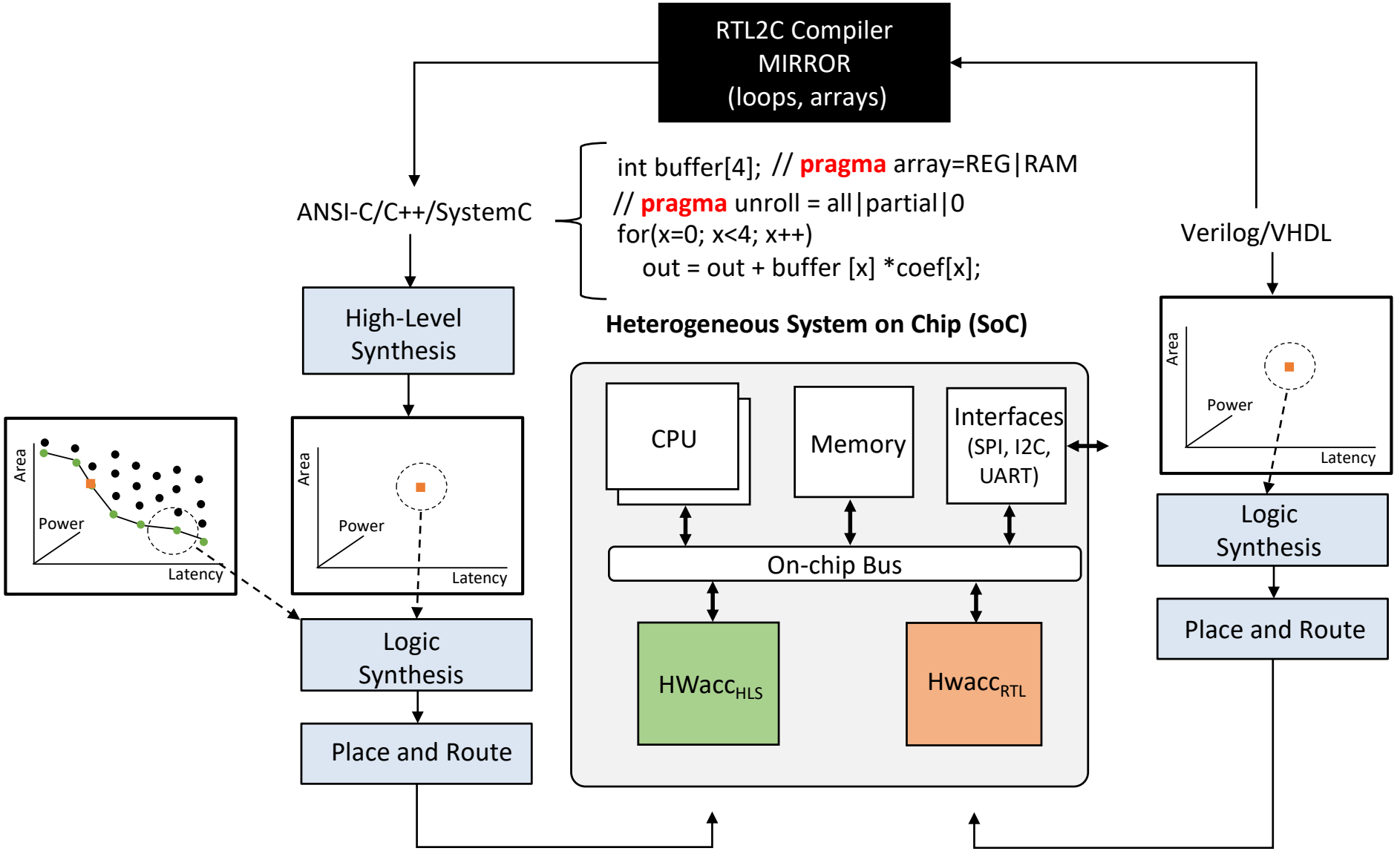
High-Level Synthesis in Practice



High-Level Synthesis Made Easy - www.hlsbook.com

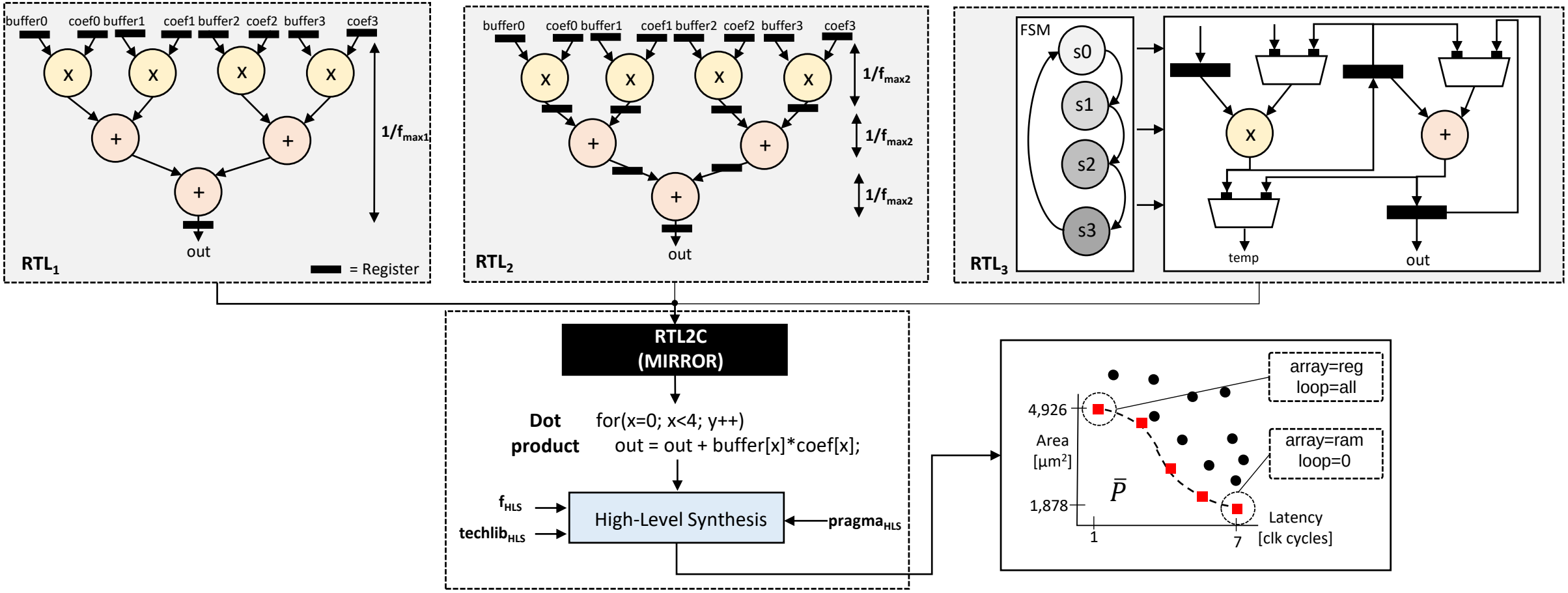


RTL to C Compiler



Motivational Example

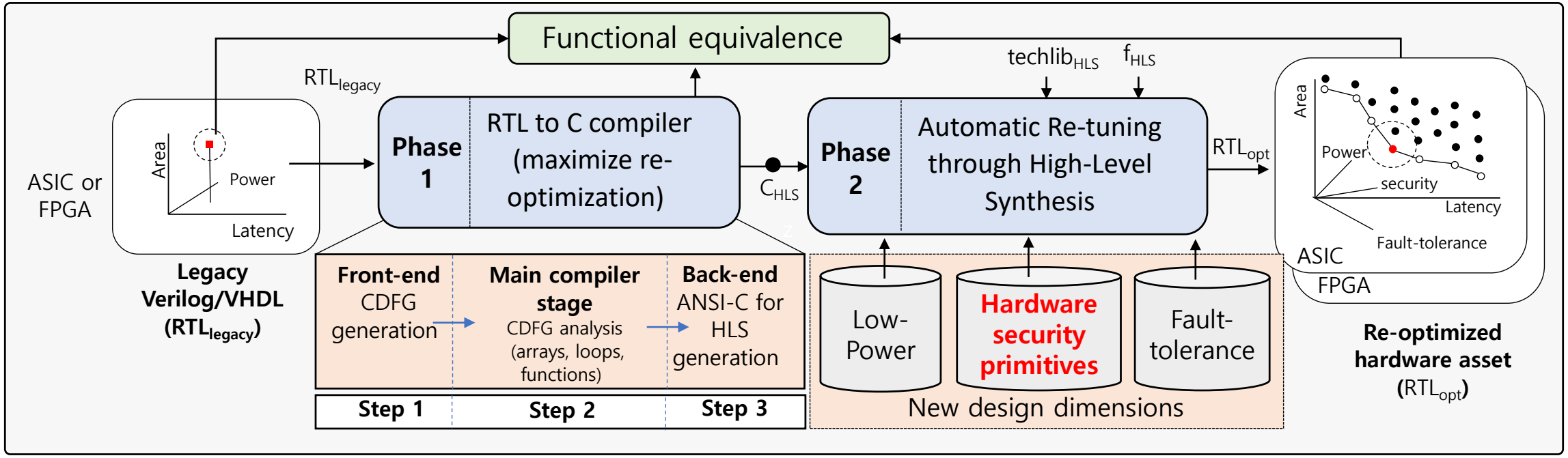
MIRROR: MaxImizing the Re-usability of RTL thrOugh RTL to C CompileR



Md Imtiaz Rashid and B. Carrion Schafer, **MIRROR: MaxImizing the Re-usability of RTL thrOugh RTL to C CompileR**, Design, Automation, and Test in Europe (DATE), pp. 1-6, 2023

Md Imtiaz Rashid and B. Carrion Schafer, **Robust and Efficient RTL to C Compiler Optimized for High-Level Synthesis**, IEEE Transactions of Computer Aided Design (TCAD), pp.1-9, 2024

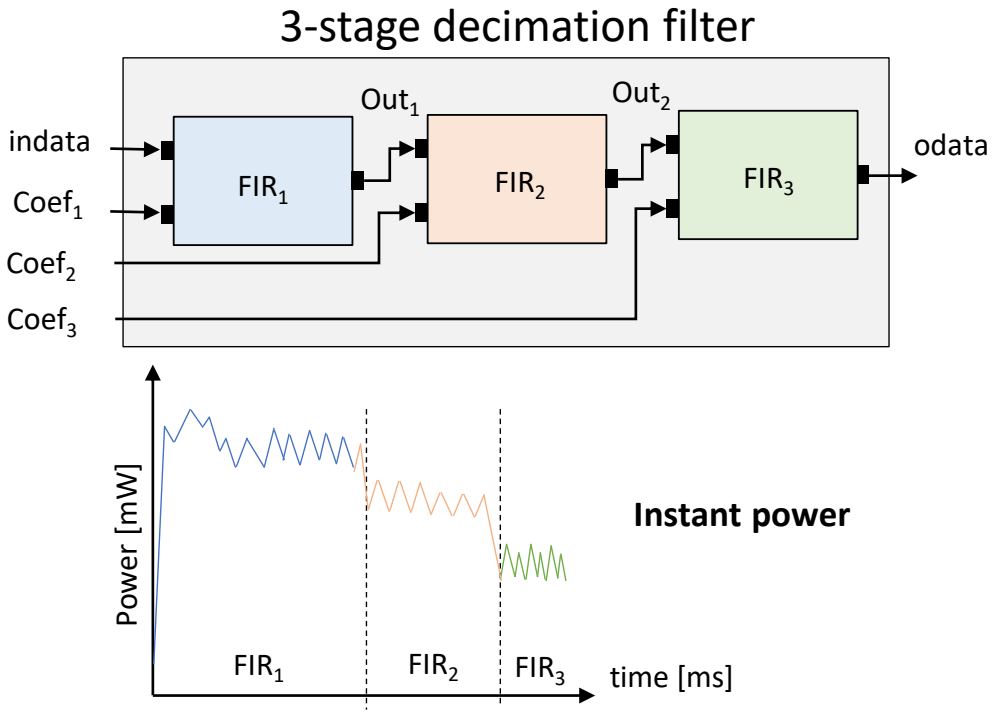
Envisioned Design Modernization Methodology



- **Input :** Legacy RTL Code (RTL_{legacy})
- **Output:** Re-optimized RTL code (RTL_{opt})
- **Flow:**
  - Phase 1: RTL2C Compiler optimized for maximum re-optimization
  - Phase 2: Automatic Re-tuning through High-Level Synthesis
- Functional equivalence between RTL_{legacy} and RTL_{opt}

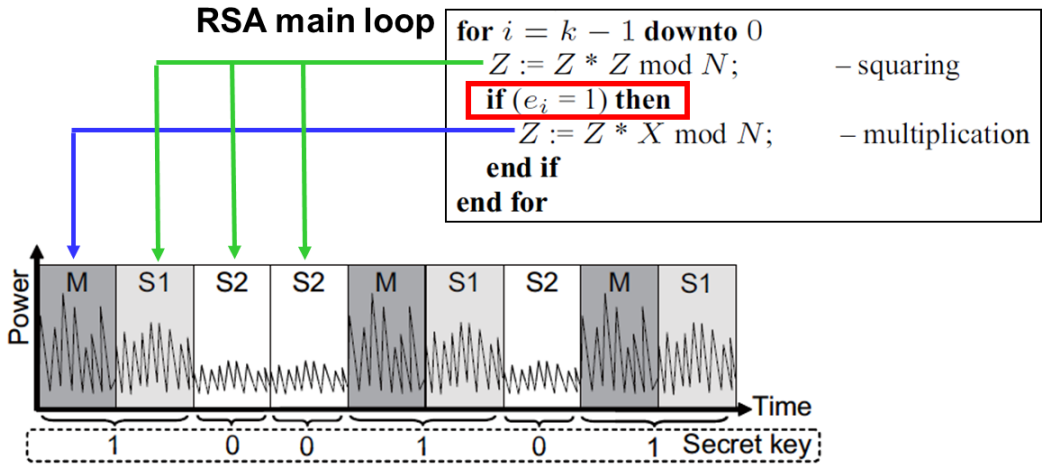
Side Channel Attacks (SCA)

- SCA: Extract information that is typically mathematically impossible to extract by measuring side channel parameters like power, timing, electro-magnetic emanations, etc..
- SCA have been mainly studied in the context of leaking encryption keys
- Example:
  - 3-stage decimation filter → attacker wants to know when filter has finished to start eavesdropping 'clean' signal
- How to measure the vulnerability?
  - t-stats
  - Side-channel Vulnerability Factor (SVF) = Correlation between sensitive application's execution pattern and side channel observations



Hardware Security : Side Channel Attacks

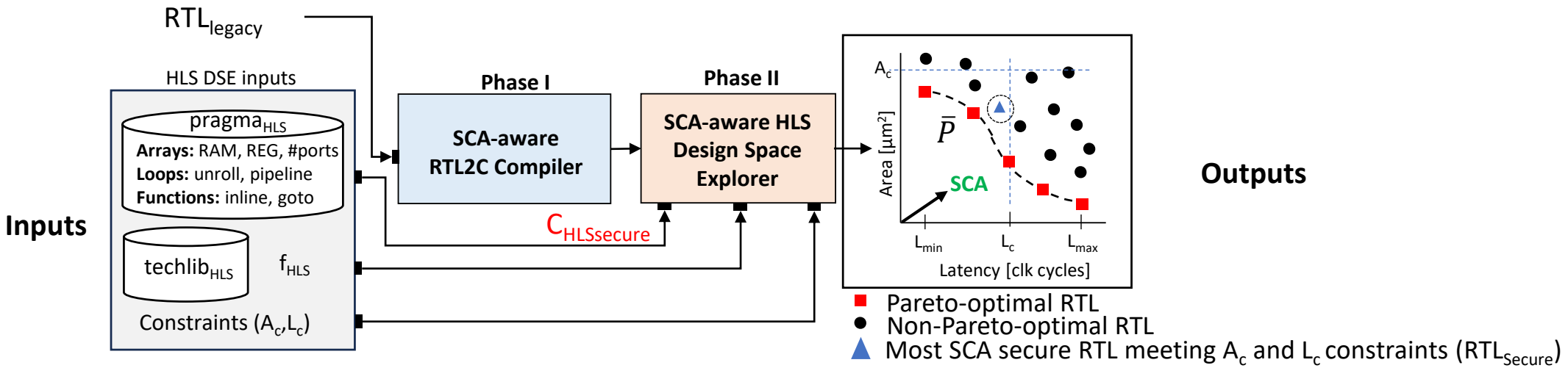
- Proposed method:
  - Leverage MIRROR (RTL2C) compiler to re-optimize legacy side channel attack unaware RTL
  - Balance or power profiles by updating behavioral description for HLS



Countermeasure

```
for (i = k - 1; i >= 0; i--) {
    Z = Z * Z mod N;
    if (e[i] = 1)
        Z = Z * X mod N;
    else
        Z = Z * 1 mod N; // dummy
}
```

Proposed Flow



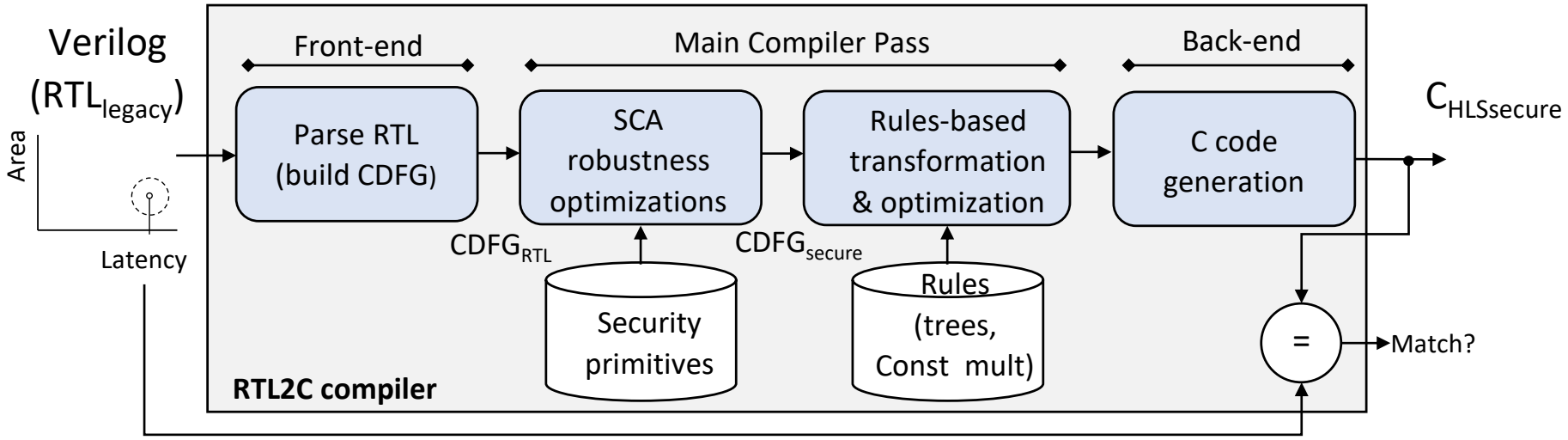
- Composed of two phases :

  Phase I: SCA-aware RTL2C compiler

  Phase II: SCA-aware HLS Design Space Exploration (DSE)

Phase I: SCA-aware RTL2C compiler

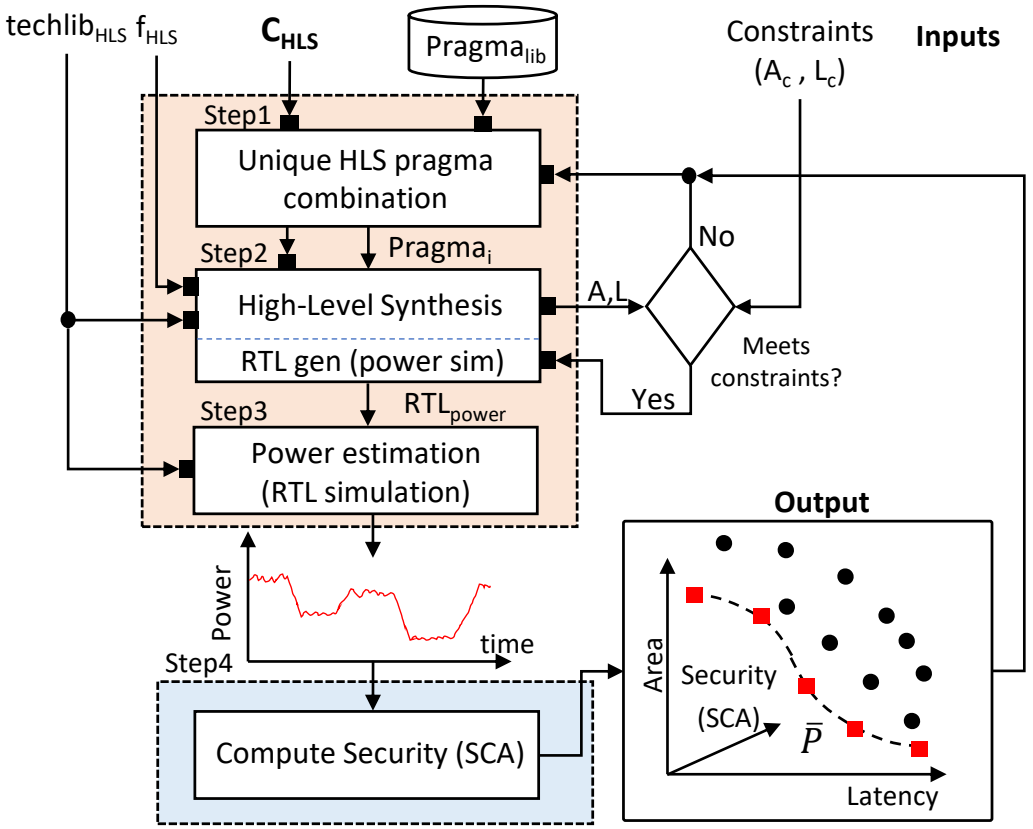
- Front-end: Parses RTL and generates CDFG
- Main Compiler pass: Applies HW security primitives
  - Reduces power fluctuations ➔ Conditional branches equally long
  - Breaks correlations between operations and power consumption ➔ adds additional dummy paths activated based on inputs similar to HW Trojans



Phase II: Security-aware HLS Design Space Explorer

- HLS DSE is a multi-objective optimization problem

- **Step 1:** Set pragma combination
- **Step 2:** High-Level Synthesis
- **Step 3:** Power estimation
- **Step 4:** Security Computation (SVF from VCD file)



Experimental Setup

Tools

- HLS Tools : NEC CyberWorkBench v.6.1
- Logic Synthesis tool: Synopsys Design Compiler v.0-2018.06-SP1
- Target technology: Nangate Opencell 45nm
- Target synthesis frequency: 200 MHz
- Compiler: Written in Python 3.6

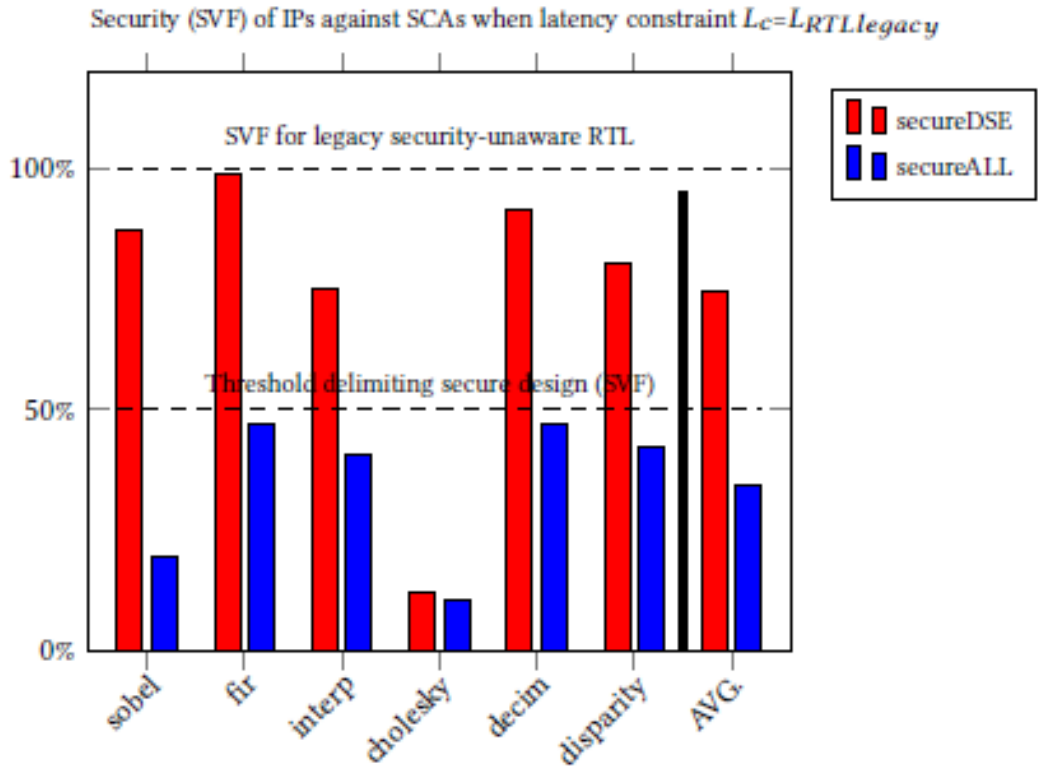
Evaluation

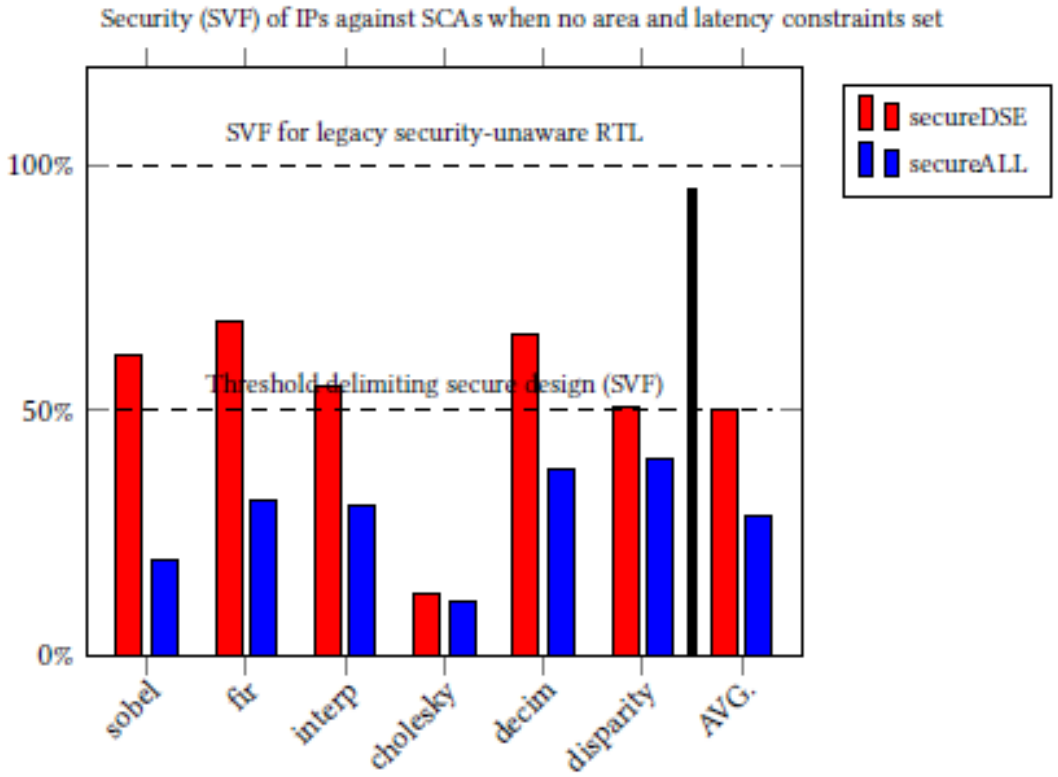
- Benchmarks: Six (6) S2CBench Benchmark suite
- Two methods: SecureDSE (only secure-aware HLS DSE)

  SecureALL (Secure—aware RTL2C compiler + secure-aware HLS DSE)

Security Vulnerability Factor (SVF) used to measure SCA robustness (SVF=100% original security-unaware RTL, 50% considered secure threshold)

Experimental Result – SVF Comparision





Observation 1: Exploring only HLS synthesis directives does not lead to a SCA secure design
Observation 2: RTL2C+HLS DSE (secureALL) does lead to secure designs. SVF improved by 65.5

Area Overhead Summary

Benchmark	Legacy vs. SecureDSE [%]	Legacy vs. SecureALL [%]
sobel	3.21	5.72
fir	1.34	7.25
Interpolation	4.15	7.15
Cholesky	4.41	5.84
Decimation	5.12	11.56
disparity	7.65	13.43

- Average area overhead of 4.31% fore SecureDSE and 8.49% for SecureALL methods

Conclusions

- Proposed an automated RTL “modernization” flow based on an RTL to C compiler that generates C code optimized for HLS
- Compiler is extended to make it security-aware (SCA)
- Investigate if SCA-aware HLS design space explorer is good enough vs. using the security-aware compiler with explorer
  - Results show that only security-aware exploration is not good enough

Thank You