Framework for Evaluating an IC's Resiliency to Fabrication-Time Attacks (U)

Tim Trippel

Summer Research Intern – Final Presentation

August 29th, 2017

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

About Me

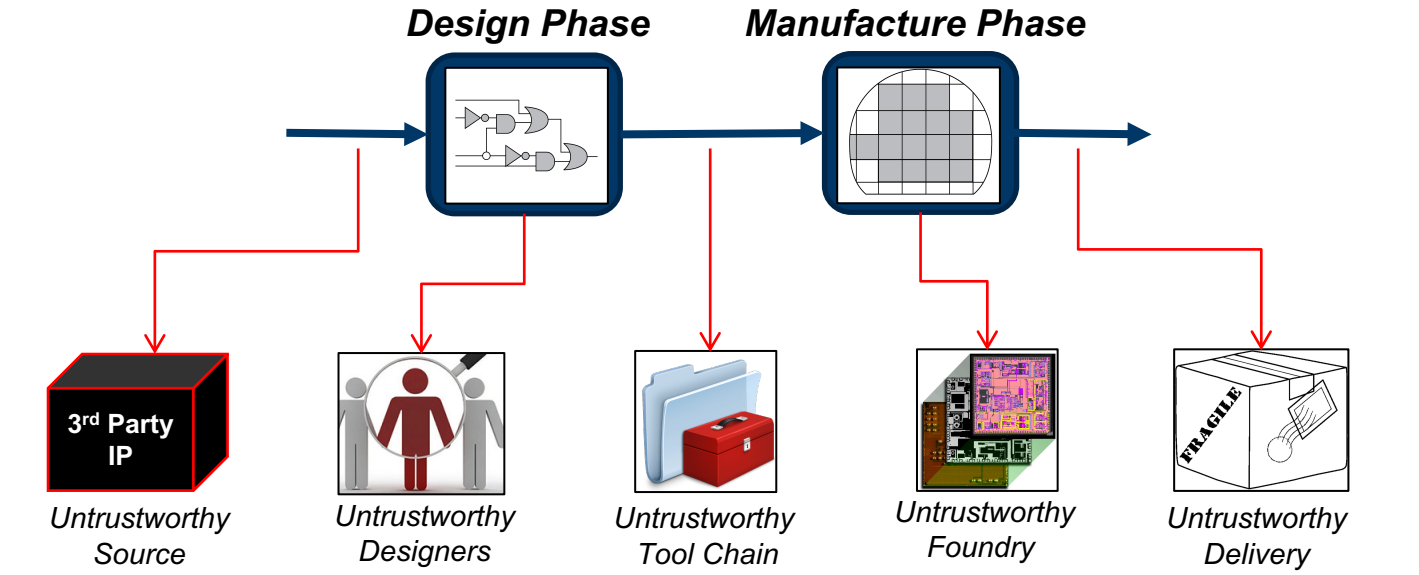
- **Purdue University**
  - **BS in Computer Engineering**
- **University of Michigan**
  - **MSE in Computer Science Engineering**
  - **PhD in Computer Science Engineering**
    - **Focus in Hardware Security**
- **Author of the "WALNUT" paper on controlling MEMS accelerometers with acoustics**
- **Worked under Matthew Hicks & Kevin Bush**

Overview

- Motivation
- Problem
- Preparation
- Approach
- Design
- Future Work

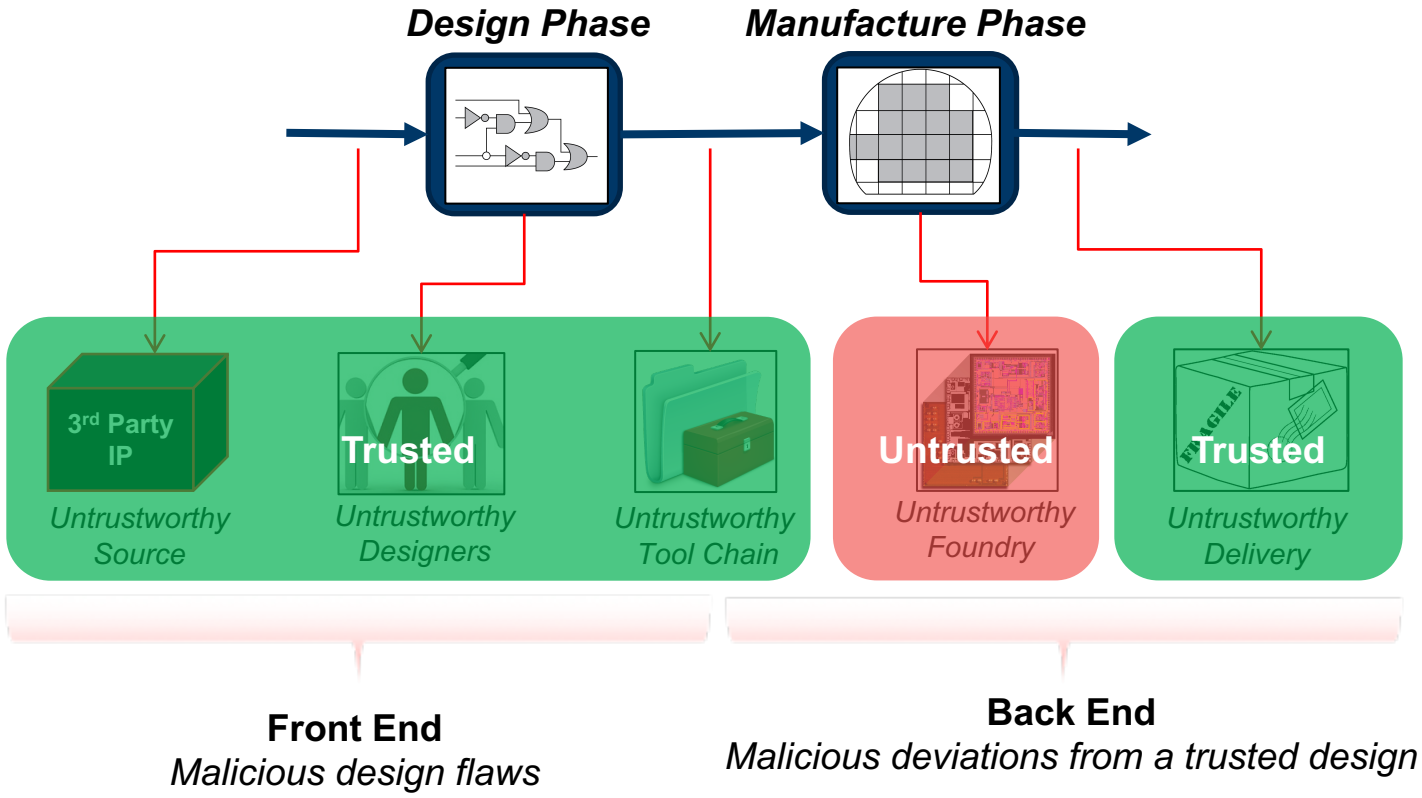
Motivation

Hardware Supply-chain Security



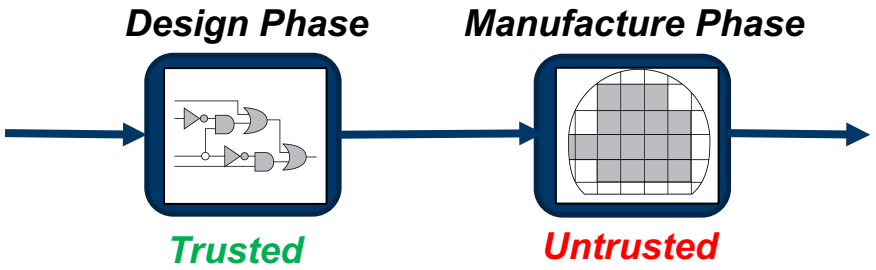
Motivation

Hardware Supply-chain Security



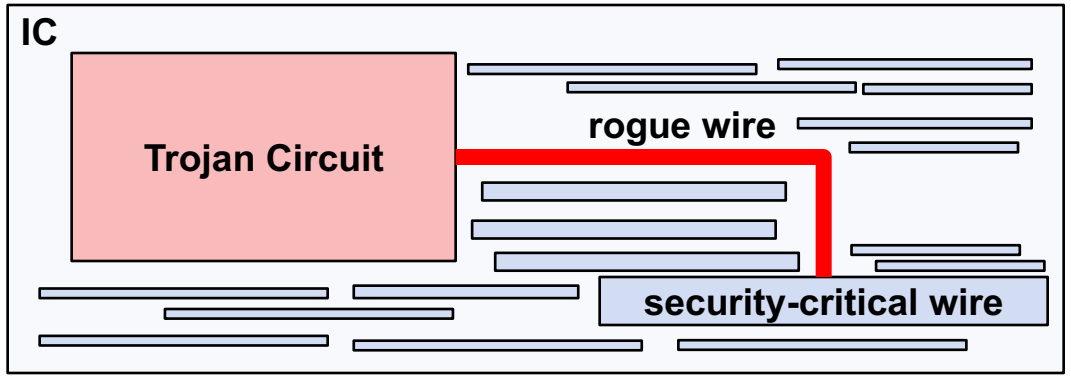
Motivation

- **Assuming one CAN create trustworthy IC designs, how can you guarantee the fabrication of these designs at untrusted foundries?**
- **Until 2016, hardware Trojans demonstrated in literature were easily detectable**
  - **Trigger circuits too sophisticated → easily detectable via physical inspection/side channels**
  - **Trigger circuits too simplistic → easily detectable via post-fab testing**
- **Oakland 2016 best paper (A2: Analog Malicious Hardware) demonstrated how a sophisticated trigger circuit could be crafted out of analog components at fabrication time making it stealthy enough to avoid detection.**



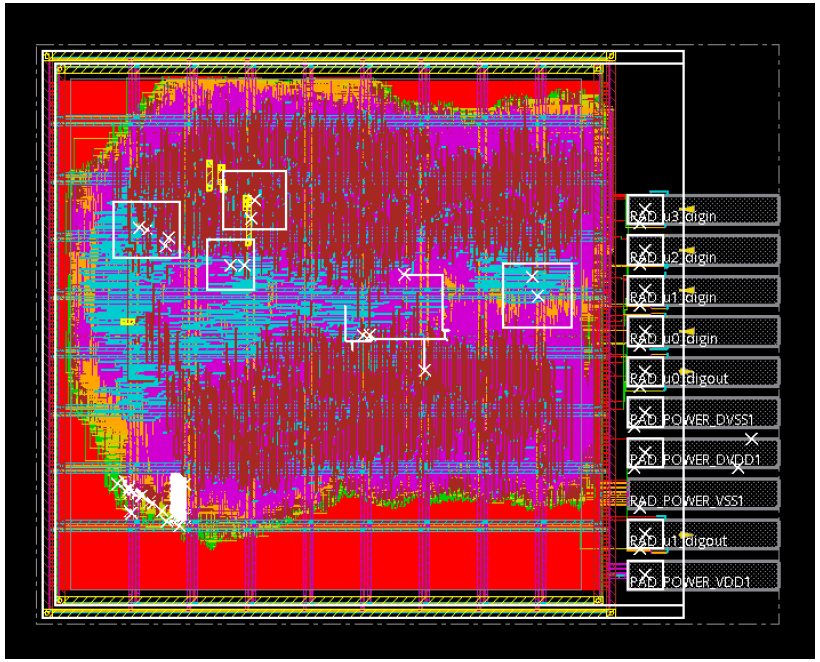
Problem

- How do we deal with stealthy fabrication-time attacks?
  - Detection
    - Stochastic Backside Imaging – image only areas of chip where Trojan is likely to be
  - Prevention
    - Defensive Layout – make it difficult for an attacker to insert the Trojan to begin with
- Need a framework to evaluate how difficult it is to insert Trojans at fabrication time
  1. How do we identify security critical nets in a design? (Nemo)
  2. How do we evaluate how difficult it is to attach a rogue wire to a critical wire in a design? (GDSII-Score)

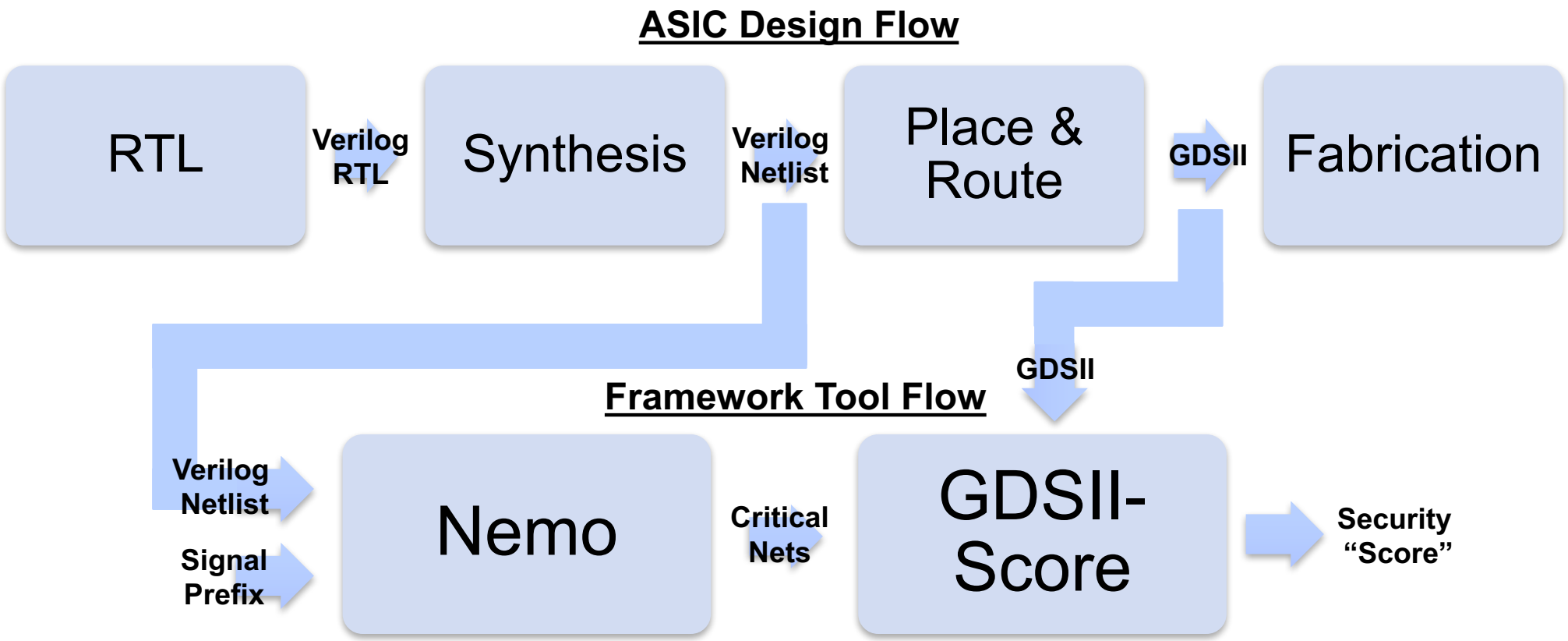


Preparation

- To study fabrication time attacks, must work with real IC design:
  - OR1200 Open Source Processor SoC (same as A2 paper)
  - Designed using 45nm ARM and IBM standard cell and I/O cell libraries
  - Synthesized with Cadence Genus v16.23
  - Placed and Routed with Cadence Innovus 17.1



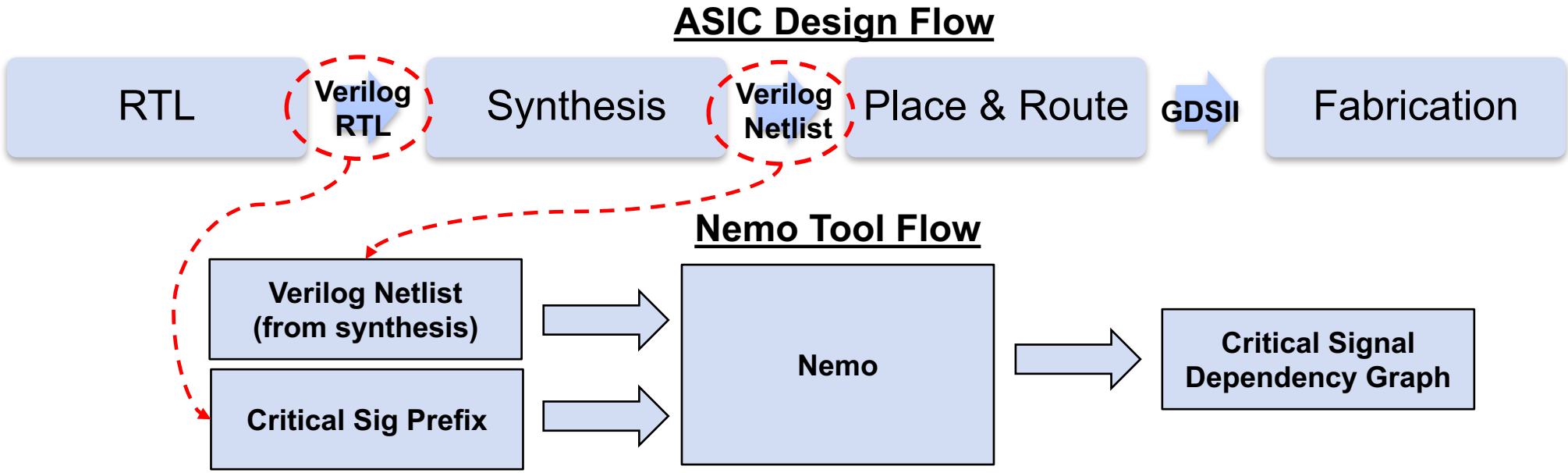
Approach


ASIC Design Flow
RTL
Verilog RTL
Synthesis
Verilog Netlist
Place & Route
GDSII
Fabrication
GDSII
Framework Tool Flow
Verilog Netlist
Signal Prefix
Nemo
Critical Nets
GDSII-Score
Security "Score"

Design

Identifying Security Critical Nets in a Design

- **Assume**: RTL designer has appended a specific prefix to "security critical" signal names in a design
- **Problem**: Given this, how can we identify the "fan-in" to these security critical signals?
- **Solution**: Nemo – a Verilog compiler back-end that outputs signal dependency graphs



Nemo has been open-sourced: https://github.com/mit-ll/nemo

Design

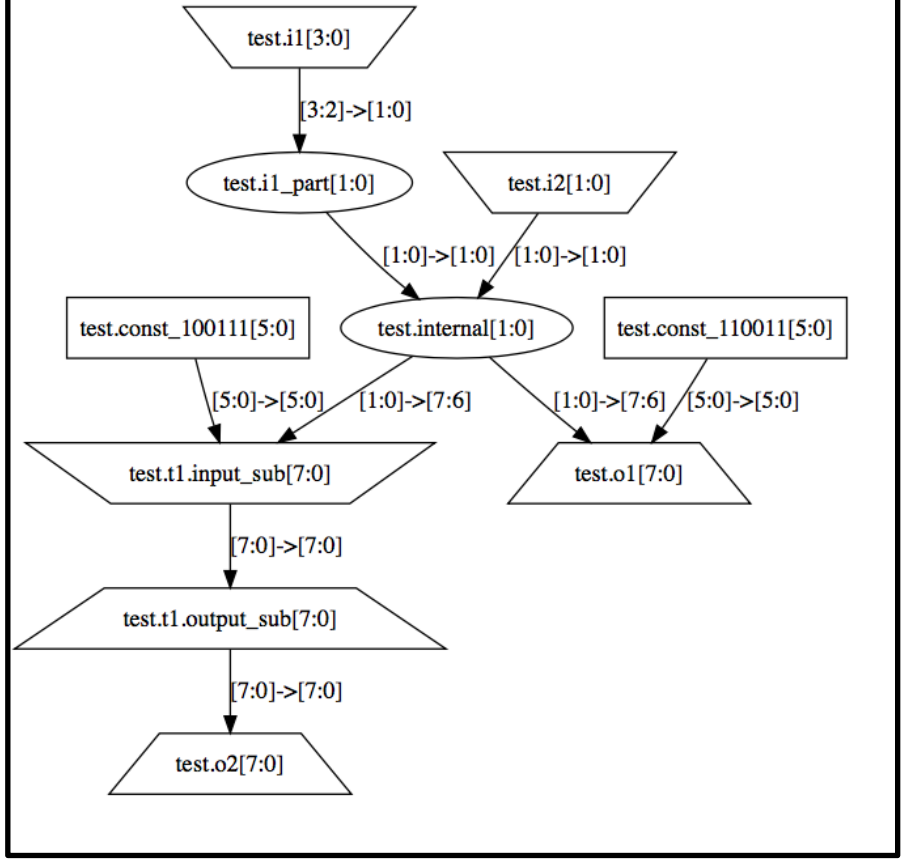
Nemo Example Usage

Nemo **INPUT**
(Synthesized Verilog Netlist)

```
module test(
    o1,
    o2,
    i1,
    i2,
);
//----------Output Ports----------
    output [7:0] o1;
    output [7:0] o2;

//----------Input Ports----------
    input[3:0] i1;
    input[1:0] i2;

    wire[1:0] internal;
    wire[1:0] i1_part;
//----------Code Starts Here----------
    assign i1_part  = i1[3:2];
    assign internal = i1_part & i2;
    assign o1 = {internal, 6'b110011};

    sub_test t1(.output_sub(o2), .input_sub({internal, 6'b100111}));

endmodule

`celldefine
module sub_test(
    output_sub,
    input_sub,
);

    output [7:0] output_sub;
    input  [7:0] input_sub;

endmodule
`endcelldefine
```

Nemo

Nemo **OUTPUT**
(Graphviz .dot file)



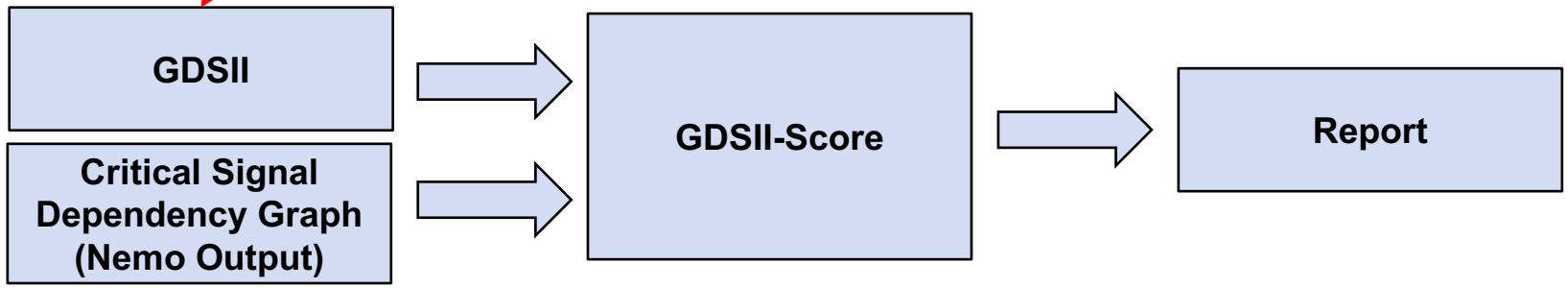
Design

Evaluating the Difficulty of Attaching Rogue Wires to Critical Wires

- **Problem**: Given a list of wires (output from Nemo), how can we identify how difficult it is to attach a rogue wire to one?
- **Solution**: GDSII-score – an automated framework for analyzing the layout of specific nets in a GDSII file

ASIC Design Flow

GDSII-Score Tool Flow



GDSII-Score will be open-sourced soon!

Design

How does GDSII-Score Work?

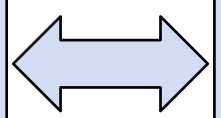
- **GDSII-Score is an *exstensible* Python framework for analyzing security-critical path objects (wires) in a GDSII file.**

class Layout():

LEF() – stores info about design rules

GDSII() – stores the GDSII data structures

CriticalNets() – Stores a list of security critical nets to examine (Nemo output)

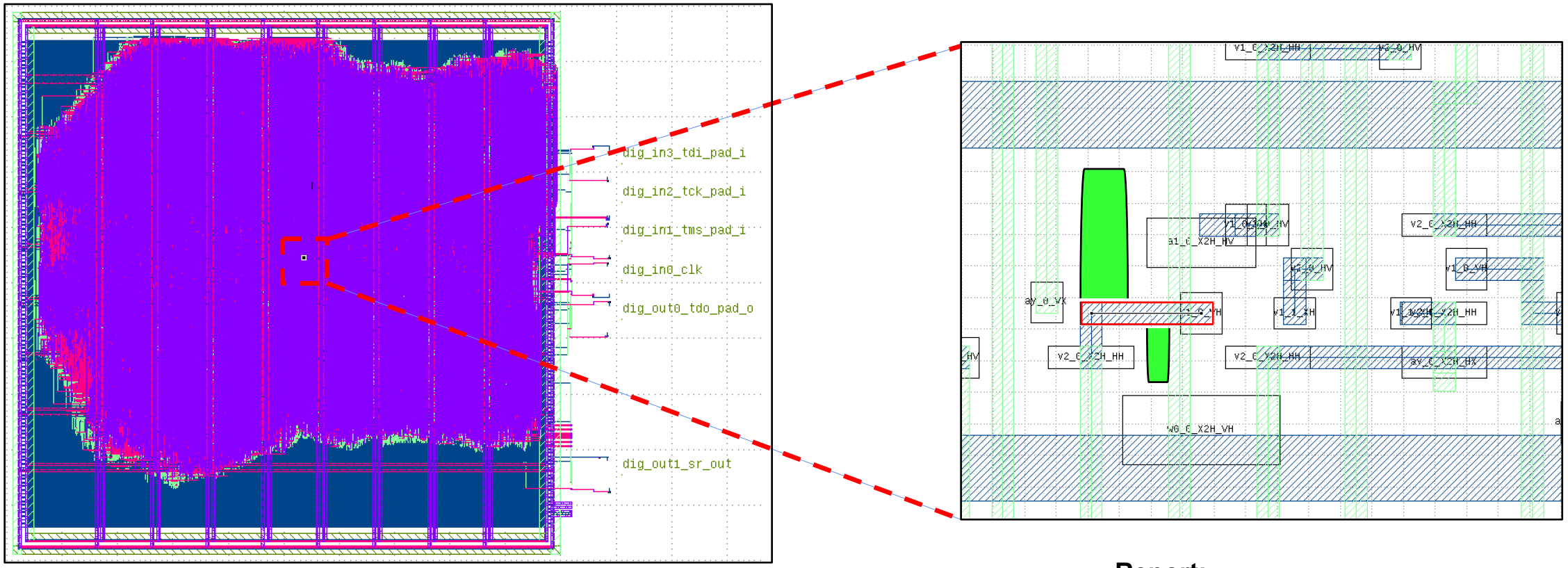


```
def analyze_critical_path_connection_points(layout):


    ############################
    # Write Scoring Algorithm HERE
    ############################
```

```
def main():
    INPUT_LEF_FILE_PATH       = 'XXX.lef'
    INPUT_LAYER_MAP_FILE_PATH = 'XXX.map'
    INPUT_GDSII_FILE_PATH     = 'XXX.gds'
    INPUT_DOT_FILE_PATH       = 'XXX.dot'

    # Load layout and critical nets
    layout = Layout(INPUT_LEF_FILE_PATH, \
        INPUT_LAYER_MAP_FILE_PATH, \
        INPUT_GDSII_FILE_PATH, \
        INPUT_DOT_FILE_PATH)

    # Analyze security critical paths in GDSII
    analyze_critical_path_connection_points(layout)
```

Design

Example GDSII-Score Metric: Calculating Wire Blockage

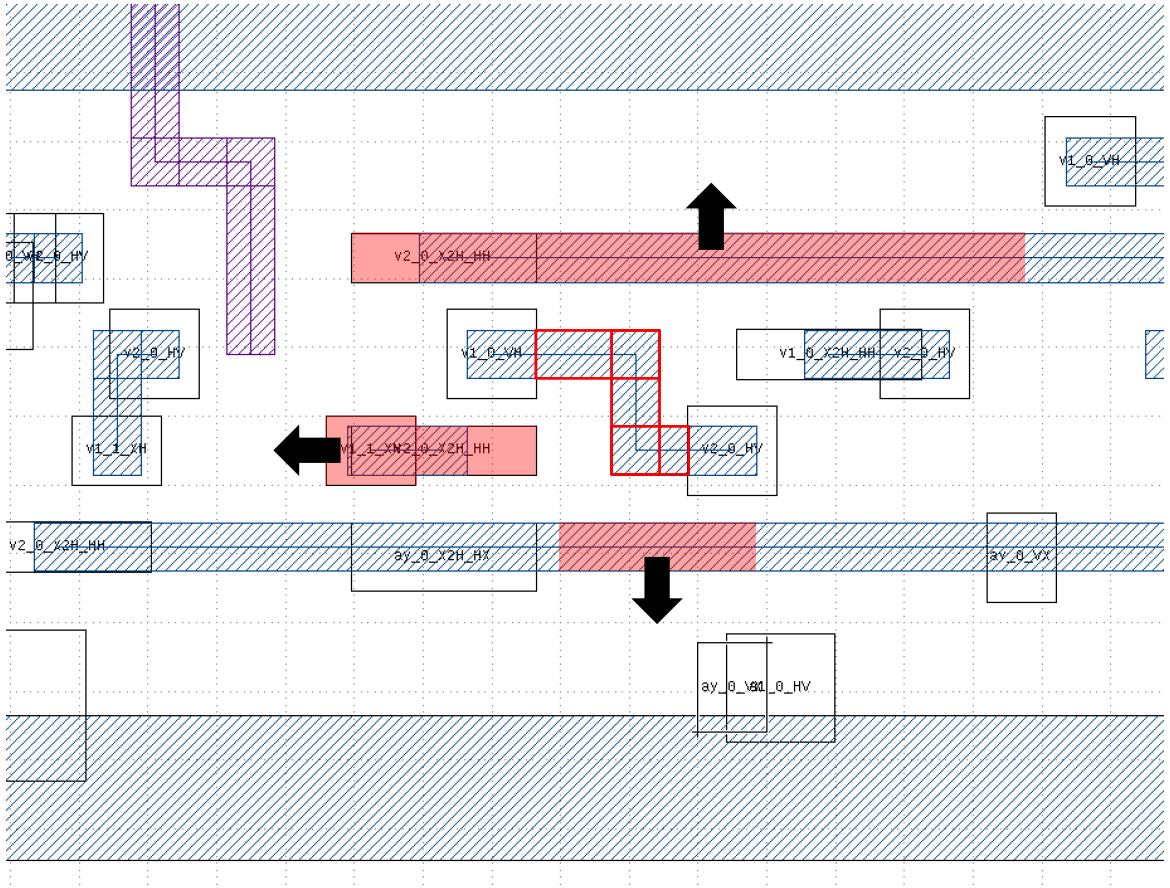


Report:
Critical Net 1 is 65% Blocked

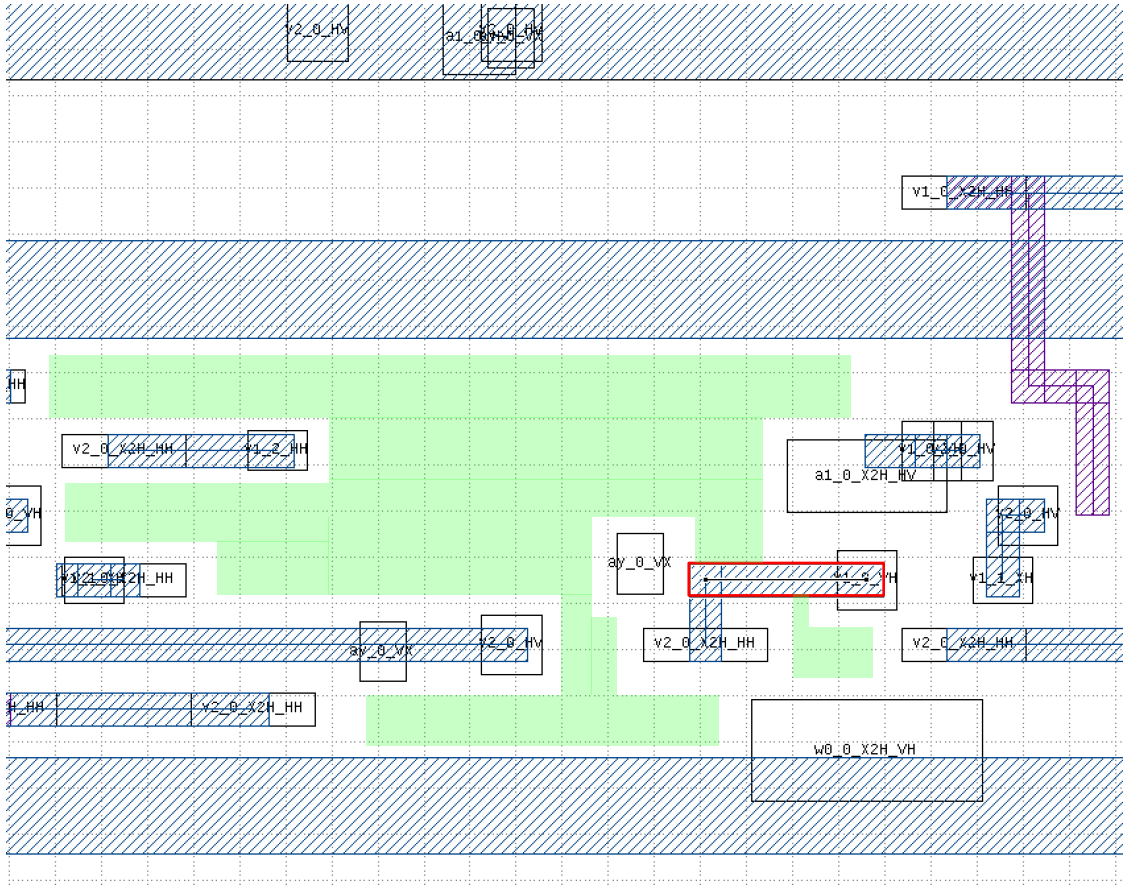
Design

Other GDSII-Score Metrics

- **Edit Distance**



- **Open Space for Implant**



Future Work

Nemo + GDSII-Score Framework

(Oakland 2018)

Stochastic Backside Imaging

(?)

Defensive Layout

(Usenix 2018)

Future Work

Open Questions/Tasks:

1. How to automate selection of security-critical nets?
2. Developing more advanced GDSII-Score metrics.

Use the framework to produce a sampling distribution of high-valued locations on an IC to image.

Use the framework to auto-generate place-&-route scripts that add “guard-wires” to sensitive nets

Acknowledgements

- **Mentors:**
  - **Matt Hicks**
  - **Kevin Bush**
- **Group 81:**
  - **Ted Lyszczarz**
  - **Brian Tyrrell**