POLITECNICO DI MILANO

Debugging and Automated Bug Detection for Hardware Generated with bambu

Tutorial @ FPT Conference 2017 – Melbourne– Australia

Pietro Fezzardi

Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
pietro.fezzardi@polimi.it

Outline

- Motivation and Goals
- Background: Hardware Debugging Methodologies
- Automated Bug-Detection with Discrepancy Analysis
- Remarks and Conclusion

Motivation

- What we had
  - Advanced compiler infrastructure
  - Different compiler optimizations and features to test
  - Extensive regression tests for multiple scenarios
  - Co-simulation workflow to identify bugs at the interfaces
- Shortcomings
  - Only find bugs at the interfaces
  - Cannot locate and isolate the first bug
  - Does not automatically backtrack to C code
  - Deep understanding of HLS and the exact optimizations is needed
  - Lots of time and trials-and-errors to do it manually

Ideas and Goals

- Turns out these shortcomings are a general issue
- All HLS frameworks face this challenges in debugging
- Design goals
  - Find bugs with single operation granularity inside the designs
  - Automatically backtrack to C
  - Automatically handle HLS compiler optimizations
  - Automatically handle HW/SW memory mapping
  - Avoid user interaction and manual operations (for regression tests)
  - Do not require to users deep knowledge of HLS engine internals
  - Automatically locate and isolate the first bug

Introduction to Discrepancy Analysis

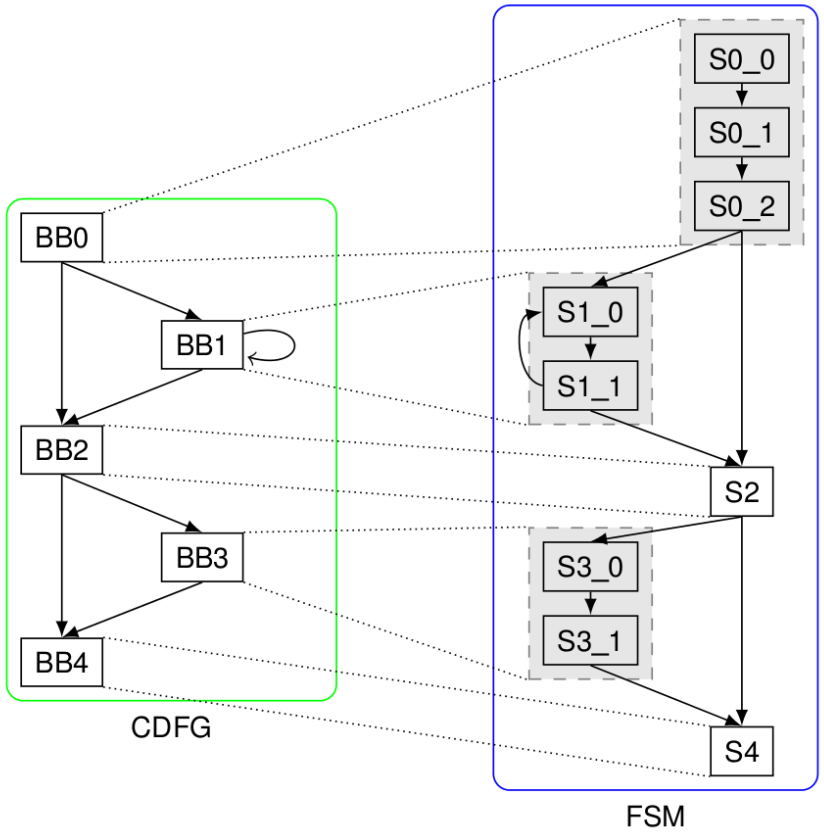
CDFG → behavior of a function

FSM → generated from CDFG with HLS

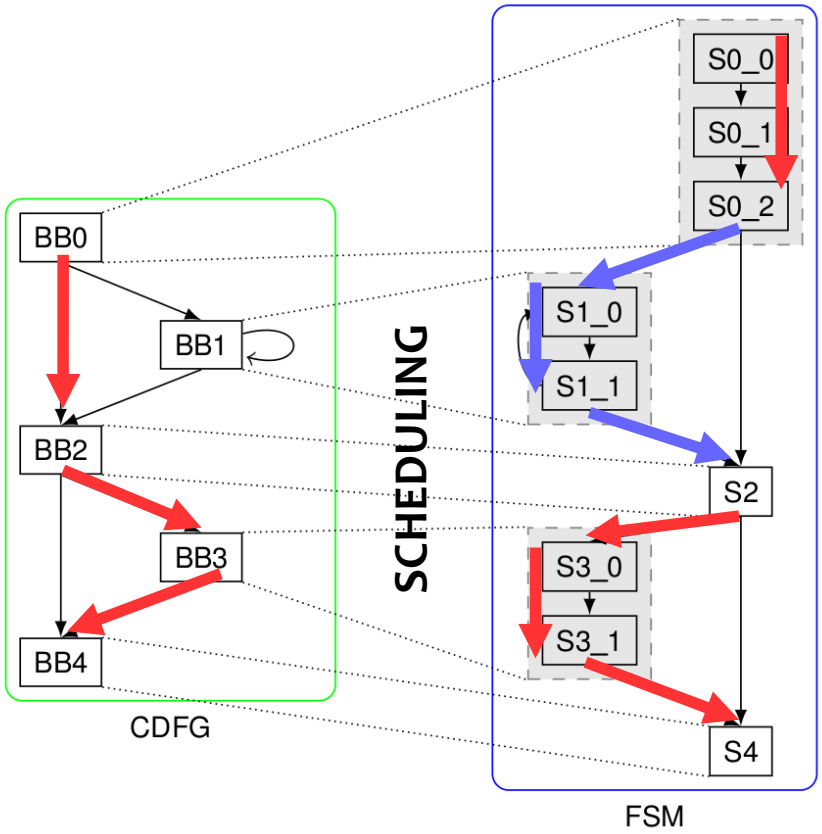
CDFG → describes SW execution

FSM → describes HW execution

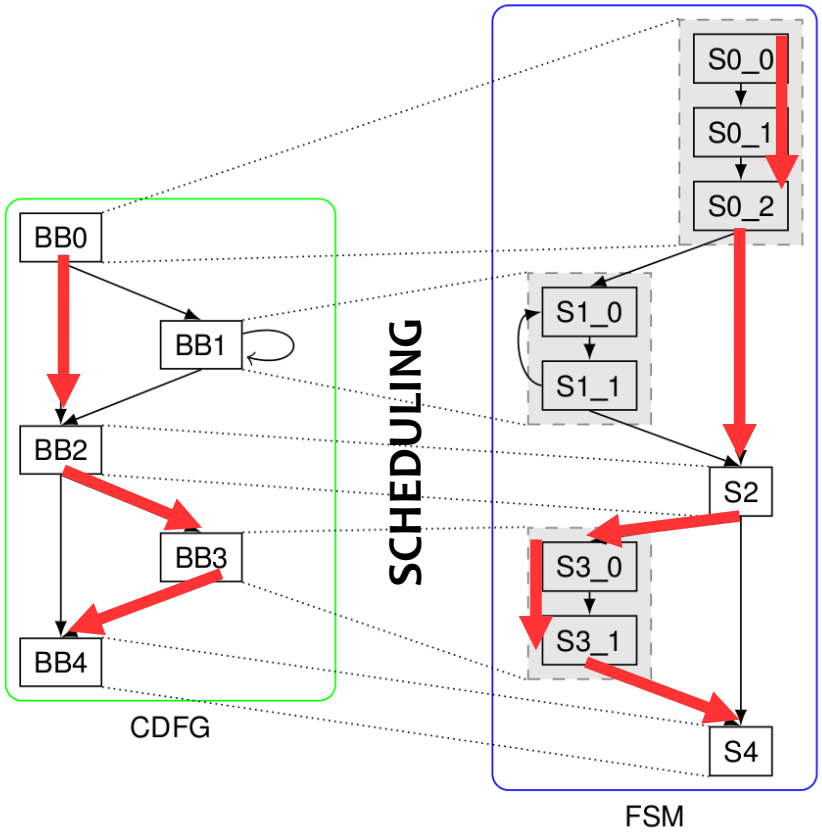
- They are executed with the same inputs
- We want to compare the two executions
- We want to be able to tell if they match
- We do it on 2 levels:
  - Control flow level
  - Operation level



Control Flow Traces (CFT)



NOT EQUIVALENT



EQUIVALENT

Comparing Control Flow Traces

- Collecting HCFT:
  - Select signals in the controller of every function's FSM
    - start_port
    - done_port
    - present_state
  - Dump the value changes during the simulation (VCD)
- Collecting SCFT:
  - Instrument the high-level source code
  - Dump the basic block number at the beginning of every basic block
- Comparing CFTs is straightforward using the scheduling map built by HLS

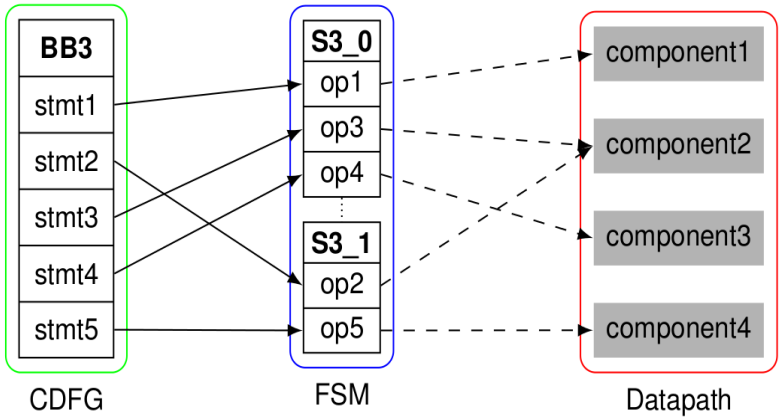
Control Flow Traces - Limitations

- Impossible to find bugs that do not change the control flow
  - wrong values not used for branching cannot be found
- Impossible to determine the moment of first discrepancy
  - the wrong value may alter the control flow long after its creation
- Impossible to isolate the failing component
  - the wrong value can be originated by a component it depends on
- We need a finer granularity → per-operation

OpTraces (OT)

- The Software OpTrace (SOT) of an operation is the list of values assigned to a variable during execution

- The Hardware OpTrace (HOT) of an operation is the list of values of the output signals of the component bounded to the operations, when the FSM is in a state where the operation is scheduled



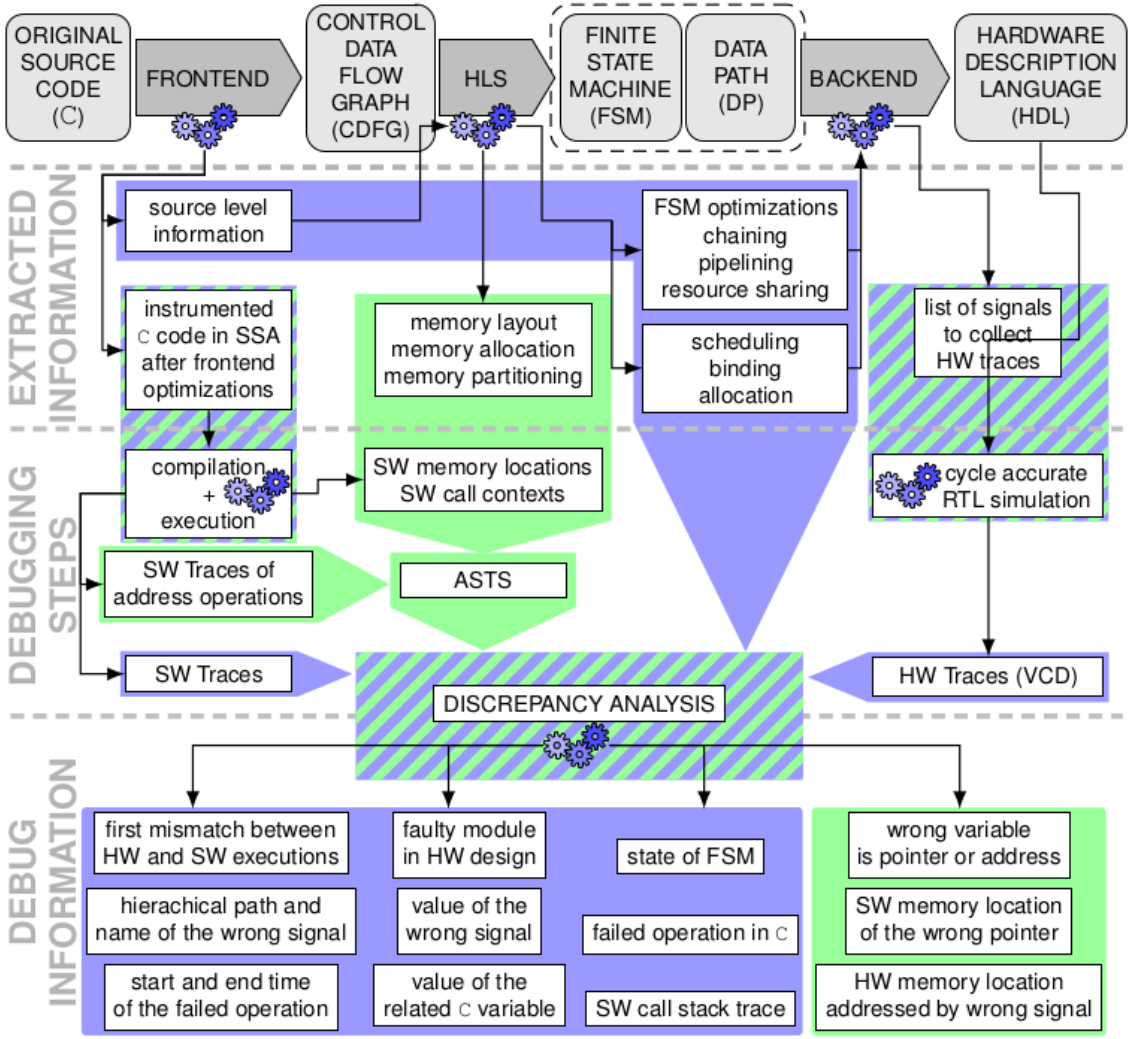
- A SOT and an HOT are equivalent if all the values of the SSAs assigned in the software operations are equal to the results of the associated operations in the corresponding states of the FSM, modulo some equivalence function.

- Example:
  - SSA assigned in stmt3 == out_signal of component2 when FSM is in state S3_0
  - SSA assigned in stmt2 == out_signal of component2 when FSM is in state S3_1

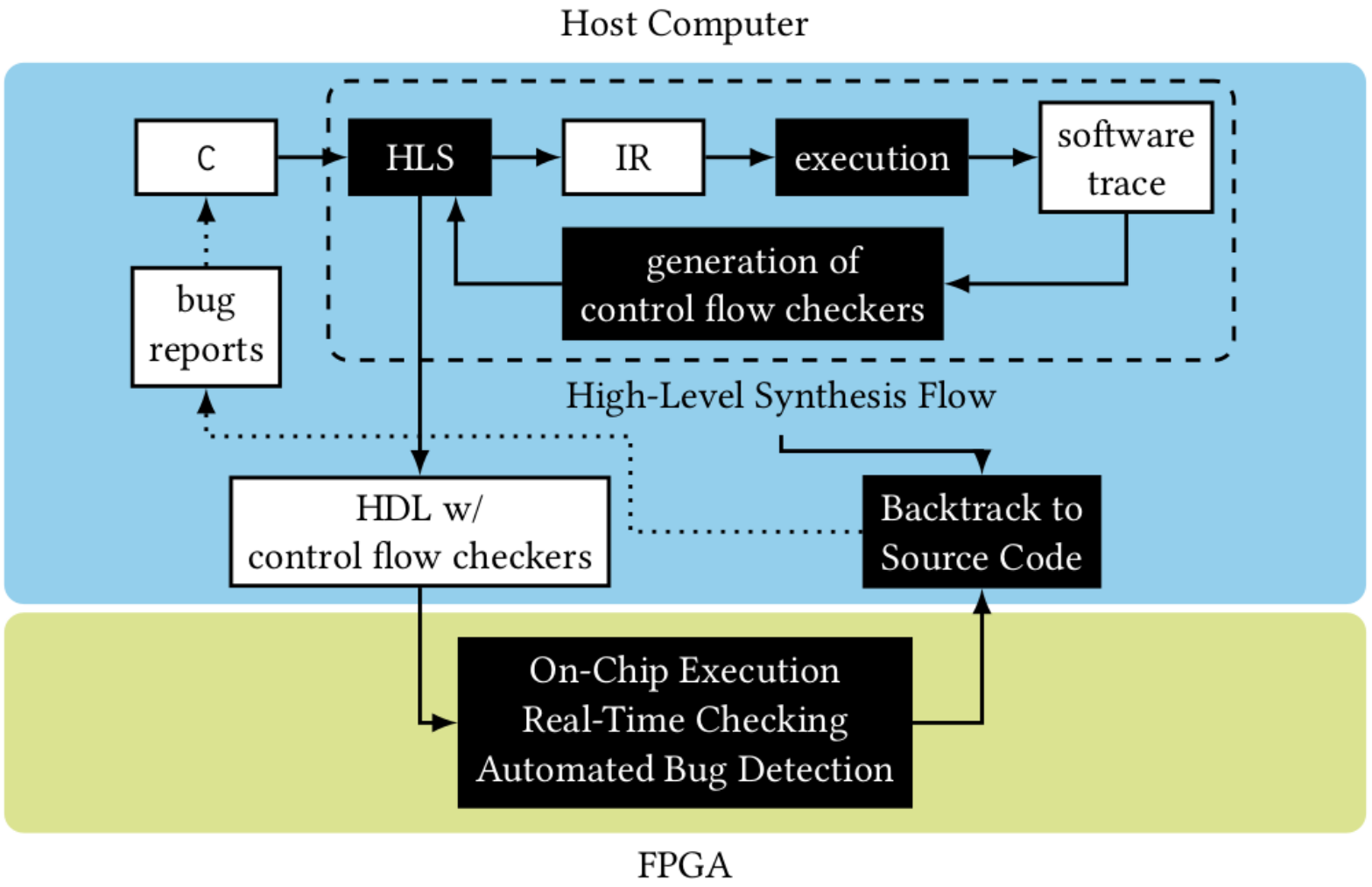
Comparing OpTraces

- Collecting HOTs:
  - Select the HW modules corresponding to operations in the FSM state
  - It is possible using allocation/binding
  - For the selected modules select the out_signal
  - Dump the signals during simulation (VCD)
- Collecting SOT:
  - Instrument the high-level source
  - Print values of every SSA variable after assignment
- Complexity of comparing the traces is linear with number of assignments
- The comparison may be something more complex than bitwise equality (floating points, pointers, custom data encoding)

Discrepancy Analysis Debug Flow



On-Chip Discrepancy Analysis



Advantages of Discrepancy Analysis

- Keep relationships between high-level code and HW
- Automatic selection of the signals necessary for debugging
- No limits on compiler and HLS optimizations
- No restriction on memory layouts
- HW/SW address translation to debug pointers
- Avoid user interaction user interaction
- Bug detection with fine-grained per-operation granularity
- Suitable for use in regression testing
- Saves lots of time automating error-prone task

bambu flags for discrepancy analysis

- `--discrepancy`

  activates the Discrepancy Analysis

- `--discrepancy-force-uninitialized`

  be more strict on discrepancies involving uninitialized values

- `--discrepancy-no-load-pointers`

  assumes pointers are never stored in memories

- `--discrepancy-permissive-ptrs`

  do not trigger hard errors on pointers and addresses

- `--discrepancy-only=comma,separated,list,of,functions`

  activates Discrepancy Analysis only inside the given functions

First example

- Three files:
  - main.c the C code that we want to translate to Verilog using High-Level Synthesis (HLS).
  - The top-level function is compute() and it uses a number of auxiliary functions that have to be mapped to the external verilog modules available in operations.v.
  - In operations.c it is possible to find the golden reference implementation in C for those very same modules.

- You are required to do the complete the following steps.

- 1) Do the necessary steps to integrate the verilog modules for `plus`, `minus`, `times`, `divide`, and `times2` (contained in operations.v) in a the HLS design synthesized with bambu. The top-level module of the final design must be the compute() function in main.c.

- 2) Write your own test cases for the whole design, passing them to the co-simulation workflow through an xml file with the proper format.

- Then run the bambu co-simulation flow, on the generated design, to check that behavior of the generated design is NOT compliant with the golden reference.

- 3) Use Discrepancy Analysis to localize the hardware module that is not respecting the specifications.