Source-Level Debugging for FPGA High-Level Synthesis

Nazanin Calagar, Stephen D. Brown, Jason H. Anderson

Presented by: Matthew Schlittler

The Problem

- Lack of debugging techniques
- If a bug is found, must use hardware techniques
- Can be difficult to debug large code bases

Past Work

- Debugging is relatively new research
- Sea Cucumber had multiple limitations
  - Doesn’t support Function calls or shared hardware resources
  - Never released

The Solution? - Inspect

- Gives a software view of the problem
- Features:
  - Step through C code
  - Set break points
  - Inspect variables
  - Show relationship between C and RTL
  - Discrepancy detection between simulation and hardware

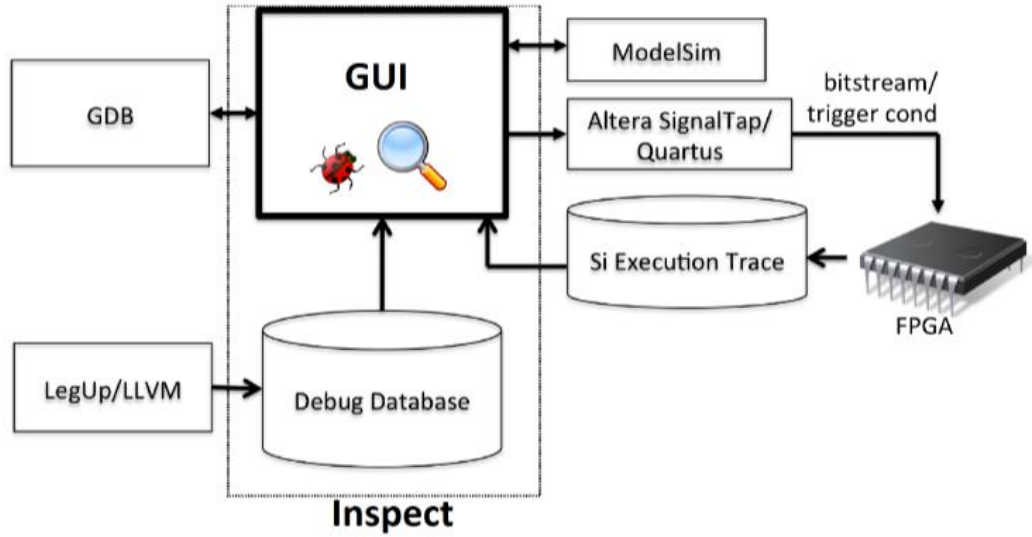
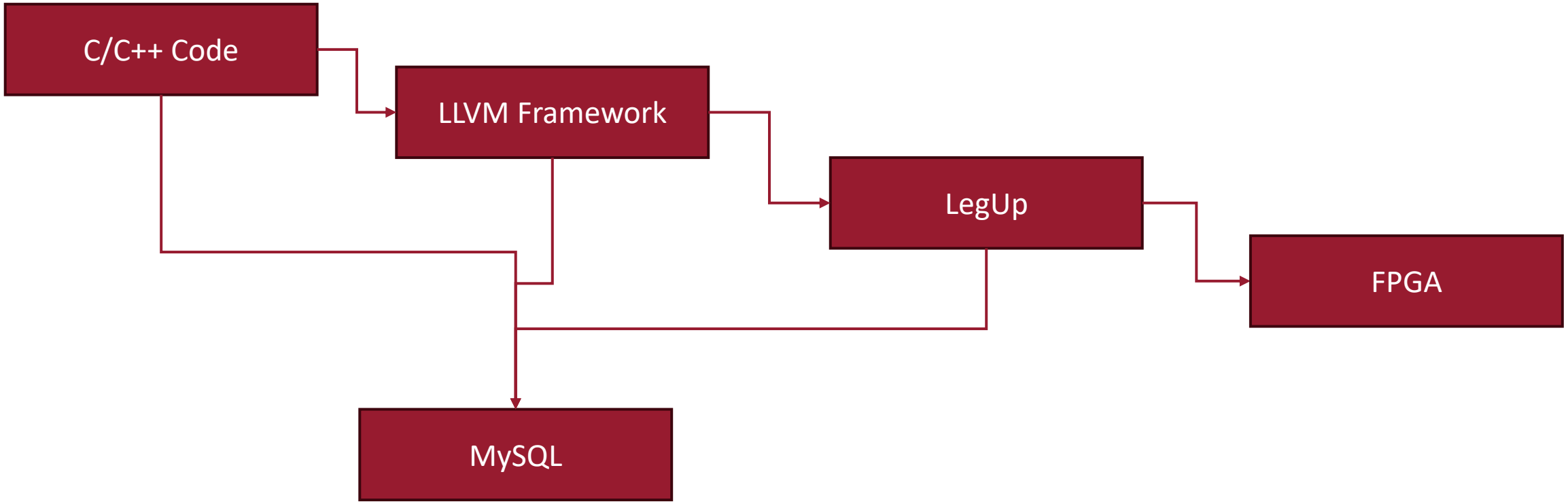


Fig. 1. Components of Inspect debug framework.

How it works



Debug Database

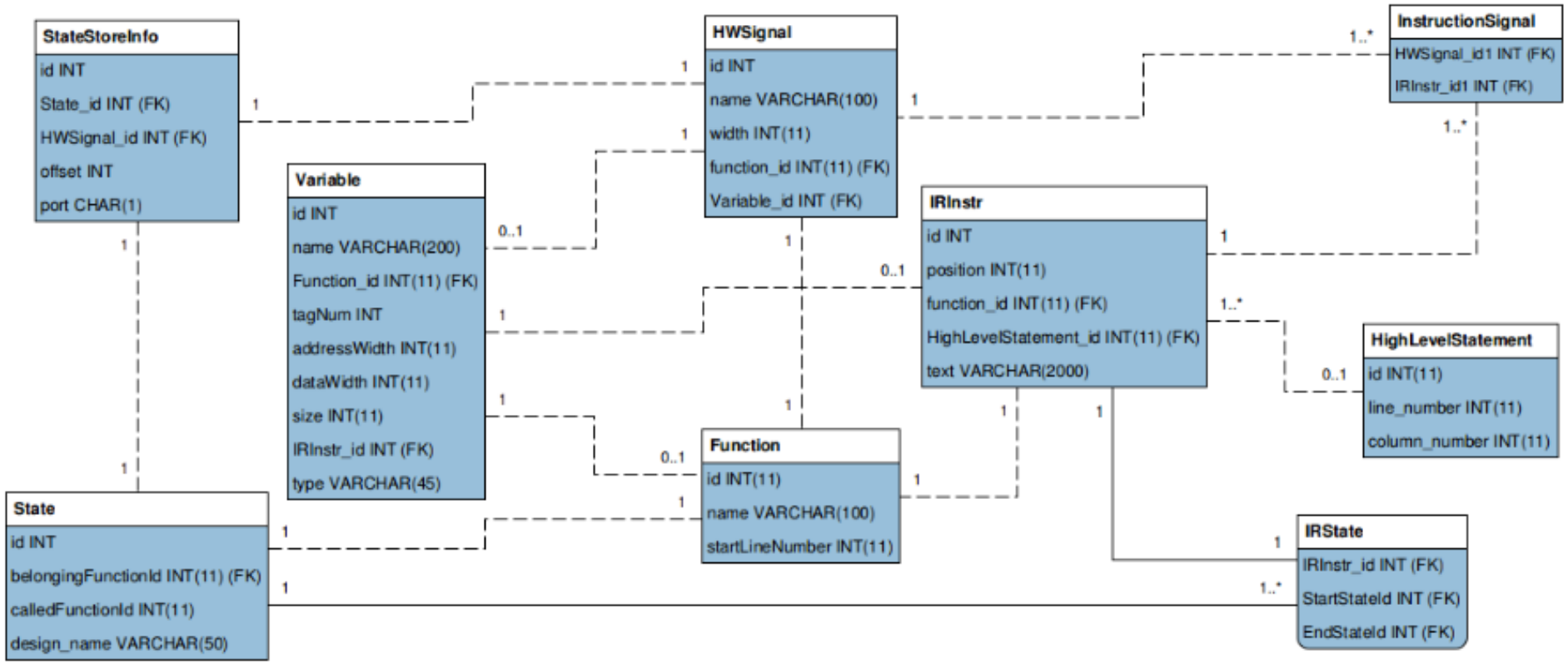
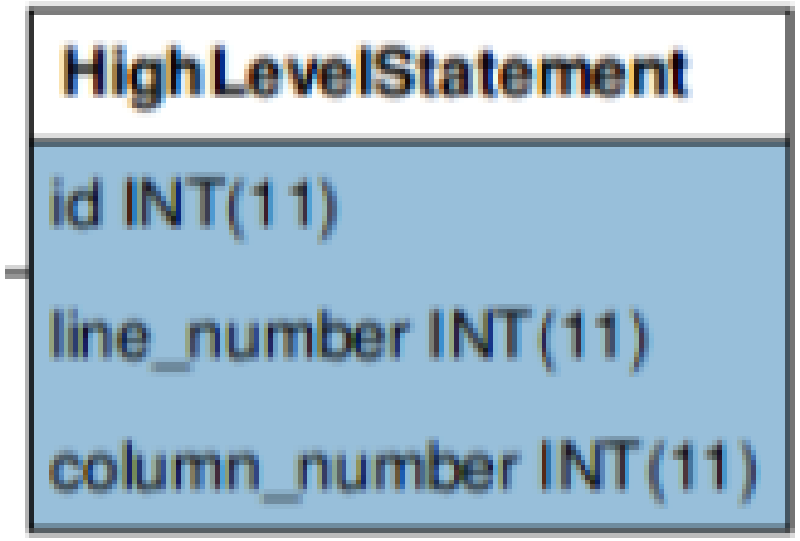


Fig. 3. Simplified Inspect debug database schema. Figure is auto-generated by MySQL.

High Level C Statements

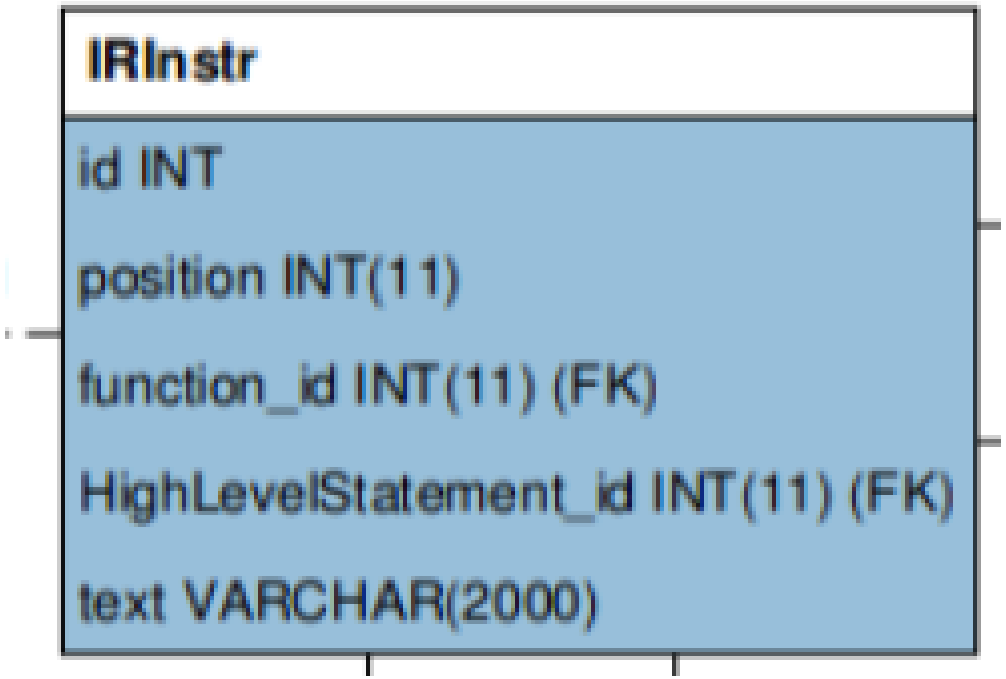
- Tracks
  - Line Number
  - Column Number
- Helpful when multiple statements in one line.

```
for (int i = 0; i < 100; i++)
{
    std::cout << (i + i - 1) << std::endl;
}
```



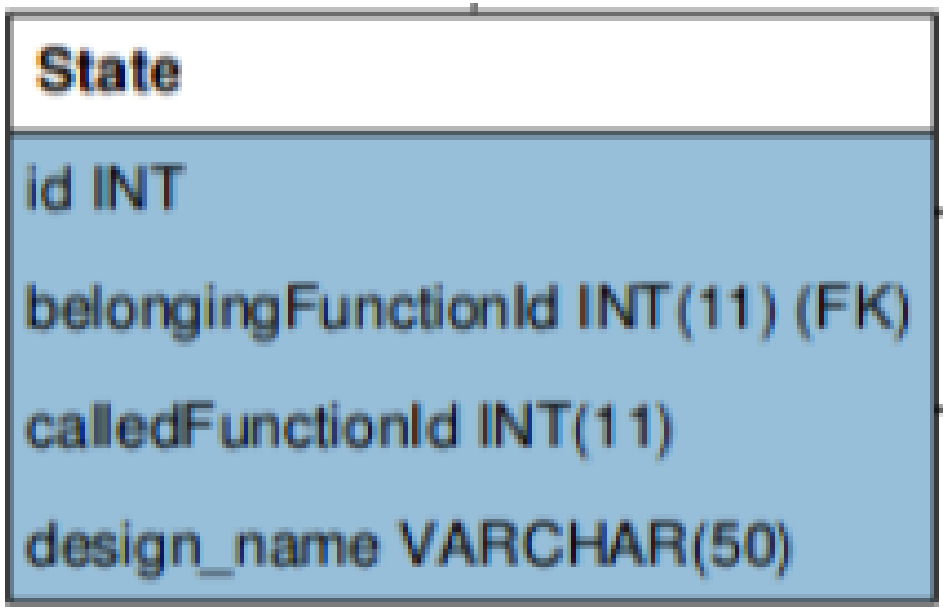
IR Instructions

- Tracks
  - Function
  - Instruction Text
  - Position in Function
  - High level statement



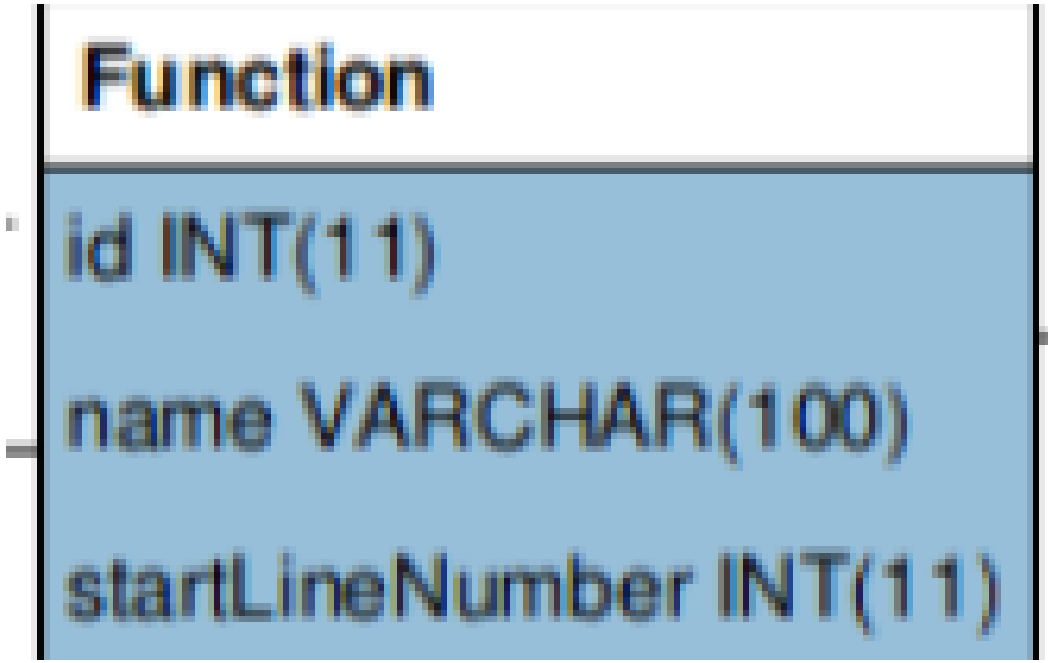
States

- Contains information about Finite State Machine (FSM)
- Tracks
  - Function it belongs to
  - Design name



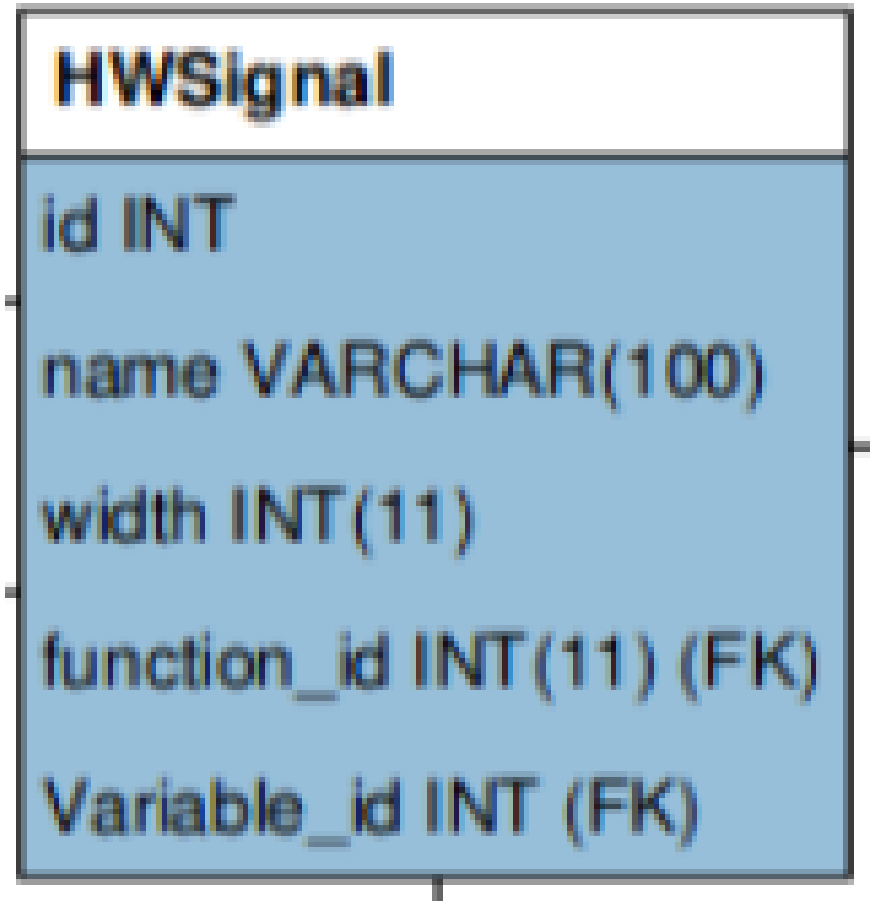
Functions

- Used to help link problem lines of code back to a specific function
- Tracks
  - Function name
  - Start of function



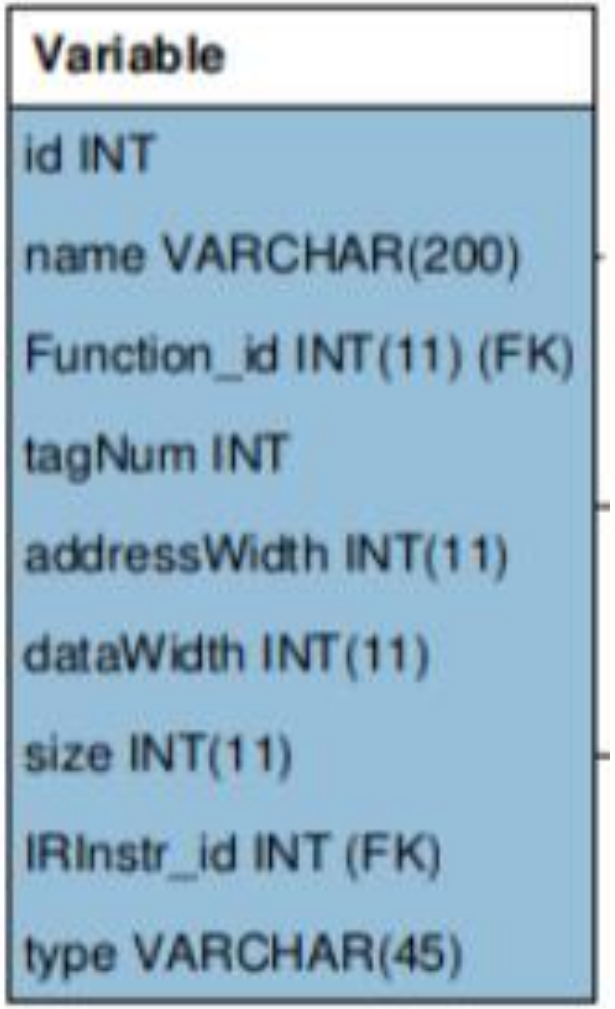
HW signals

- Tracks
  - Wires and Registers
  - Name
  - Bitwidth



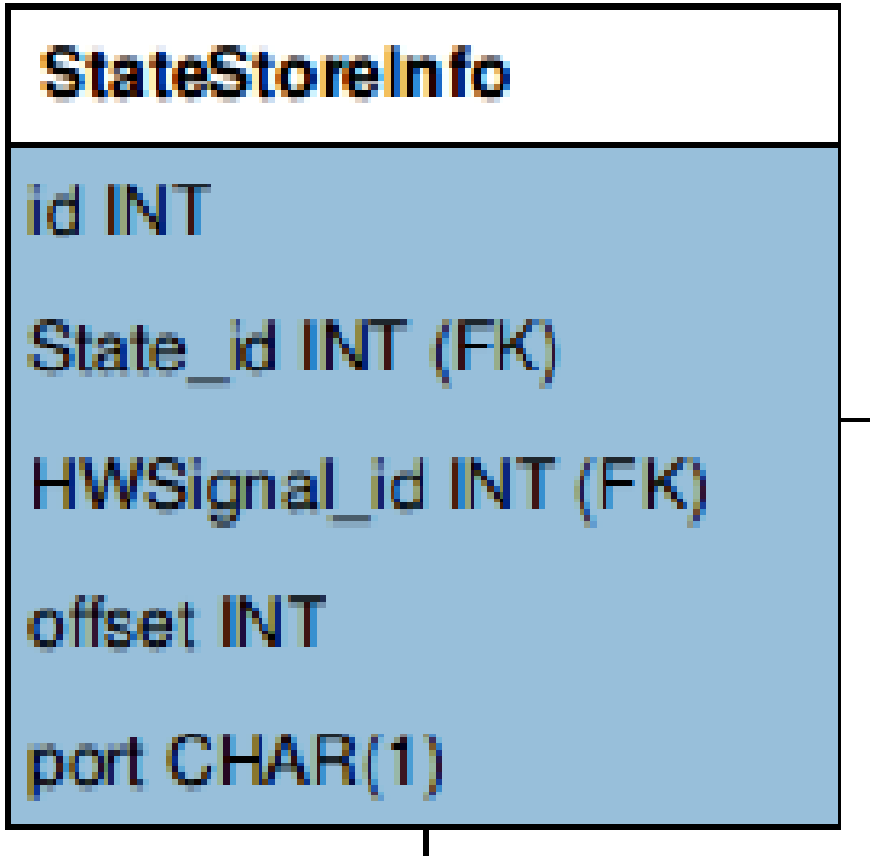
Variables

- Tracks
  - Name
  - Function
  - Address Size
  - Data Size
  - Number of elements



StateStoreInfo

- Represents the relationship between variable and hardware signal
- Tracks
  - State
  - H/W Signal
  - Memory Offset
  - Port of read/write



Case Study

- Gave example of the debug process
  - Simulation Mode
  - Silicon Mode

```
1 : #include <stdio.h>
2 : /**/
3 : int array[2][2][3] = {{{1,2,3},{4,5,6}}, ...
4 : int isPrime (int num)
5 : {
6 :   int i, test;
7 :   test = 1;
8 :   if (num % 2 == 0)
9 :     test = 0;
10:   else {
11:     for (i = 2 ; i < num; i++)
12:         if (num % i == 0)
13:           test = 0;
14:   }
15:   return test;
16: }
17: int main (void)
18: {
19:   int a, b, c, number, result;
20:   result = 0;
21:   for (a = 0; a < 2; a++) {
22:     for (b = 0; b < 2; b++) {
23:       for (c = 0; c < 3; c++) {
24:         number = array[a][b][c];
25:         result += isPrime(number);
26:       }
27:     }
28:   }
29:   ...
```

Fig. 4. C code for case study – finding the primes in a 3D array.

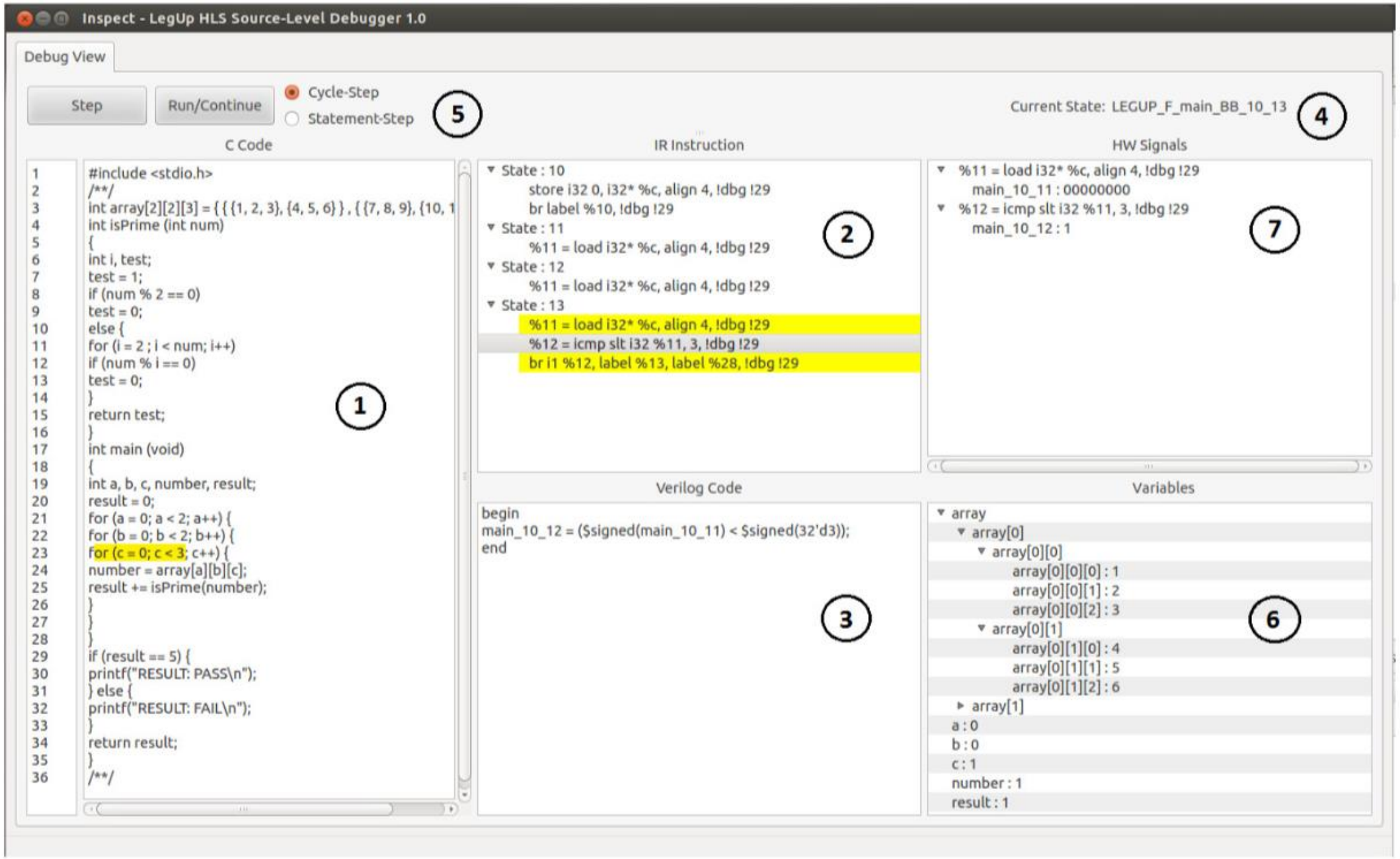
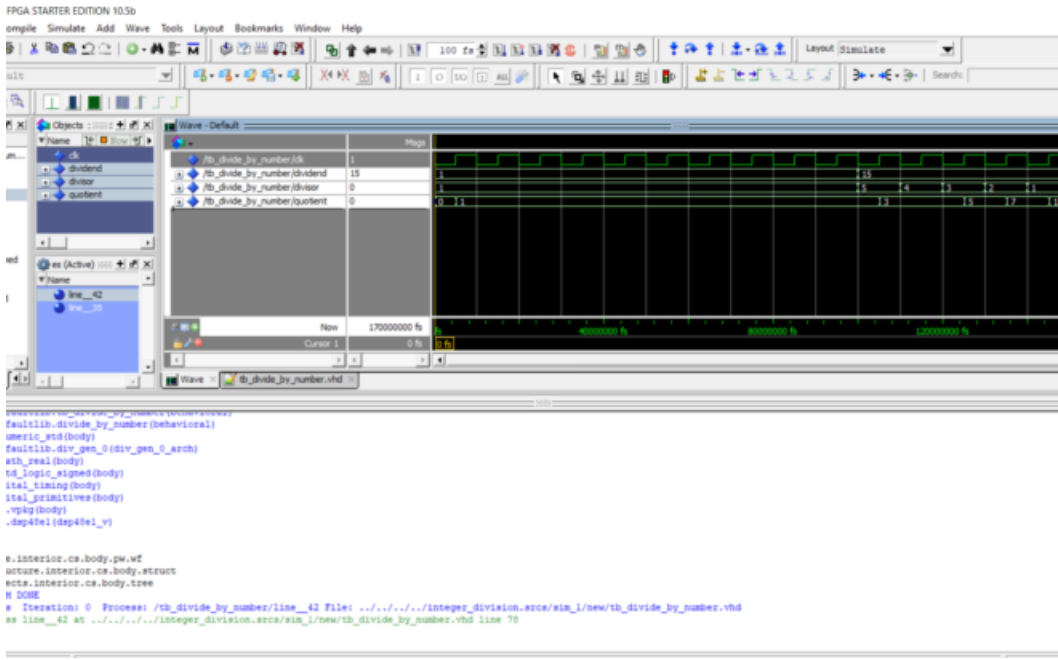


Fig. 5. Screenshot of Inspect during a debugging session.

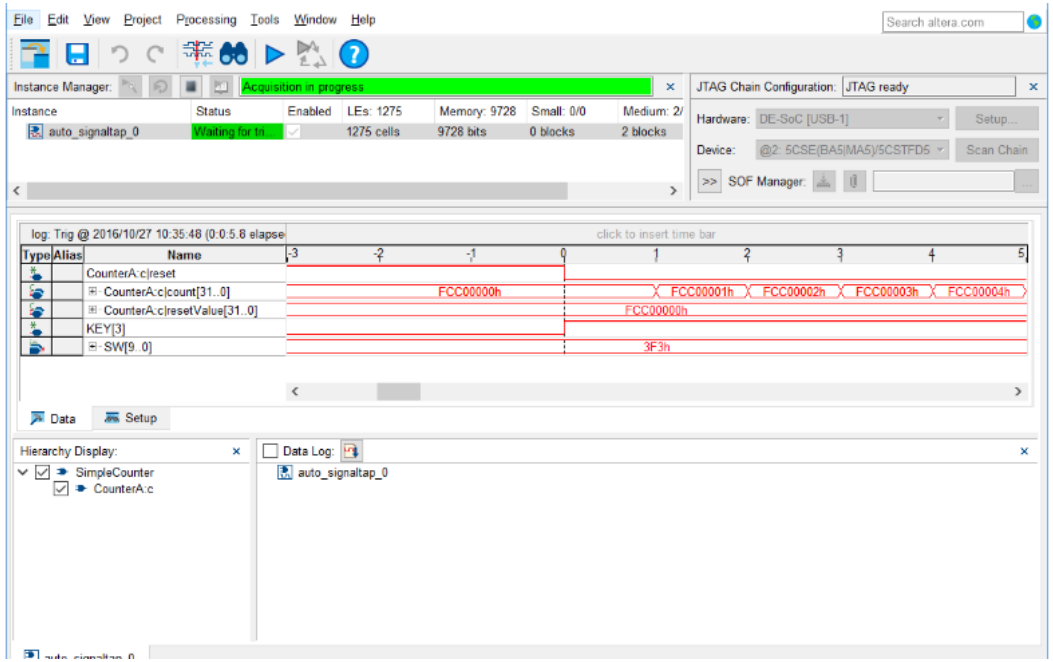
Simulation Mode

- Uses ModelSim
- Sends commands to ModelSim via TCP



Silicon Mode

- Utilizes Altera Signal Tap II
- User must choose signals to watch
- Inspect helps
  - Sliding window-based debug
  - Intelligent signal selection



Simulation vs Silicon

TABLE I. SIMULATION RUN-TIME COMPARISON.

Benchmark	RTL Sim (s)	Timing Sim (s)	Increase (×)
MIPS	0.9	76	84
MOTION	1	118	118
DFMULL	1.6	248	155
DFADD	0.8	37	46

Other functionality

- Execution management
  - Stepping
    - Statement step
    - Cycle step
  - Breakpoints
- Inspecting Signals
- RTL/Silicon Discrepancy Detection

Testing Inspect – S/W vs RTL

- Took CHStone benchmarks
- Inserted bugs in generated code
- Used Inspect discrepancy's functionality

Results

TABLE II. AUTOMATED HARDWARE/SOFTWARE DISCREPANCY DETECTION RESULTS.

Benchmark	Bug	# Bugs	1st Bug Cyc	Tot Cyc	Pass/Fail
MIPS	Bug #1	7579	503	16919	Fail
	Bug #2	16337	449	17582	Fail
	Bug #3	5497	1474	17684	Pass
GSM	Bug #1	13264	1785	14030	Fail
	Bug #2	16	27664	30424	Fail
	Bug #3	2817	6362	30642	Fail
MOTION	Bug #1	66	62	20212	Fail
	Bug #2	20	19425	19829	Fail
	Bug #3	88	18953	19816	Fail
DFMUL	Bug #1	25	138	3294	Fail
	Bug #2	3	479	3318	Fail
	Bug #3	21	588	3308	Fail
DFADD	Bug #1	51	168	8161	Fail
	Bug #2	735	68	10133	Fail
	Bug #3	2	400	8257	Pass
DFDIV	Bug #1	24	1851	6793	Fail
	Bug #2	112	1594	6769	Fail
	Bug #3	24	1808	6793	Fail
	GEOMEAN:	138.7	1046.4	10831.9	

Testing Inspect – RTL vs Silicon

- Took CHStone benchmarks
- Inserted bugs in generated code
- Introduced bugs by changing FSM

Results

TABLE III. SILICON DEBUG RESULTS.

Benchmark	Bug	1st Bug Cyc (# Configs)	Total Cyc (# Configs)	Pass/Fail
MIPS	Bug #1	412 (1)	6552 (4)	Pass
	Bug #2	677 (1)	827 (1)	Fail
	Bug #3	6594 (4)	6640 (4)	Fail
MOTION	Bug #1	8355 (5)	9347 (5)	Fail
	Bug #2	208 (1)	566 (1)	Fail
	Bug #3	8357 (5)	9204 (5)	Fail
DFMUL	Bug #1	486 (1)	1766 (2)	Fail
	Bug #2	208 (1)	∞ (∞)	Fail
	Bug #3	768 (1)	1948 (2)	Pass
DFADD	Bug #1	234 (1)	∞ (∞)	Fail
	Bug #2	208 (1)	4103 (3)	Fail
	Bug #3	983 (1)	4117 (3)	Fail

Future Work

- Support for watchpoints
- Better correlation between H/W and S/W
- “Wizard-like” functionality

Thoughts on Paper

- Contribution
  - Inspect was released
  - Multiple citations
- Strengths
  - Paper has been cited multiple times
  - Explained the functionality very well
- Weakness
  - Some stuff was not explained, i.e. acronyms

Sources

- https://ieeexplore.ieee.org/document/6927496 (Paper)
- https://www.llvm.org/
- https://www.mehmetburakaykenar.com/an-introductory-modelsim-tutorial-for-vivado-xilinx-users/116/
- http://staff.washington.edu/kd1uj/BEE271/Quartus/How%20to%20use%20SignalTap%20II.pdf