Jonas Fiala
Trinity Hall
jf613

Part II Project Proposal

Property-based Testing for Hardware

October 2019

Project Originators: Matthew Naylor and Jonas Fiala

Resources Required: See attached Project Resource Form

Project Supervisor: Matthew Naylor

Signature:

Directors of Studies: Simon Moore and Hatice Gunes

Signatures:

Overseers: Pietro Lió and Robert Mullins

Signatures:

Introduction and Description of the Work

Test benches are common when developing hardware and so many techniques have been developed for automatic testing and generation of test cases. Property-based testing requires the designer to specify a set of invariants for the module being tested, this allows automatic testing to check that these invariants always hold, reporting back if that is the case. This is a useful and now commonplace idea in automatic testing.

Property-based testing has become popular in the software community. Both **QuickCheck** [1] and **SmallCheck** [2] are existing testing libraries that encourage lightweight formal specification of software through the reward of automatic testing and small counter-examples. Recent work [3] has successfully applied the idea to hardware development in the Bluespec HDL.

Hardware verification tools are often commercial products and expensive, so the open-source community needs new approaches. This project will continue the work of **BlueCheck**, but explore some new avenues. Namely we will use an open-source Bluespec-like HDL [4] developed here in the CL, additionally we will use bounded exhaustive testing, which can can find minimal counterexamples without the need for shrink steps.

Here is an example of specifying and testing the firstHot function from the BlueCheck [3] paper. This function returns a bit-string in which only the least significant non-zero bit of the input bit-string is set:

```
// FirstHot implementation to test
function Bit#(n) firstHot(Bit#(n) x) = x & (~x+1);

// The defining properties that any First Hot function must meet:
// Exactly one output bit is set if nonzero, otherwise no bits set
function Bool prop_OneIsHot(Bit#(8) x) =
  countOnes(firstHot(x)) == (x == 0 ? 0 : 1);

// Hot bit in output is also hot in input
function Bool prop_HotBitCommon(Bit#(8) x) =
  (x & firstHot(x)) == firstHot(x);

// No less-significant hot bits in input than that set in output
function Bool prop_HotBitFirst(Bit#(8) x) =
    (x & (firstHot(x)-1)) == 0;
```

Starting Point

I have basic prior knowledge, from taking the **Computer Design** course in **Part IB** and the **Digital Electronics** course in **Part IA** including the practicals for both. Therefore an important part of the project will be learning to use the required tools.

Substance and Structure of the Project

The aim of the project is to build a Haskell library for automatic testing of modules written in Blarney. Primarily I will draw inspiration from SmallCheck [2] and apply it to a HDL. As such a method using bounded exhaustive testing will be implemented. This gives a clear demarcation between tested and untested cases, and also guarantees finding small counterexamples. The work consists of three main parts, described in the success criteria.

The project will likely be expanded to achieve some of the additional points mentioned in the next section, but which ones are chosen will depend on the effectiveness of the initial implementation. Further options may be discovered when researching this topic in depth and also implemented.

Success Criteria

The following should be achieved:

- Implement a Haskell library for automatic testing of combinatorial circuits written in Blarney
- Devise a suite of example properties and buggy circuits, to evaluate the effectiveness of the library
- Measure the proportion of bugs reported and the size of counter-examples found
- Synthesise a test bench to an FPGA and compare speeds against a simulated test bench

Finally there are many further possibilities beyond the success criteria for this project:

Sequential logic modules could be supported, testing a bounded sequence of inputs [3]

Random testing similar to QuickCheck and BlueCheck [1, 3]

Other verification methods (such as an SMT solver) could be created for Blarney and then compared to the bounded exhaustive testing method

References

[1] K. Claessen and J. Hughes. *QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs*, in Proceedings of ICFP'2000, (2000)

[2] C. Runciman, M. Naylor, F. Lindblad. *SmallCheck and Lazy SmallCheck: automatic exhaustive testing for small values*, in Haskell Symposium '08. pp. 37–48. ACM (2008)

[3] M. Naylor and S. W. Moore. *A Generic Synthesisable Test Bench*, in MEMOCODE 2015, pp. 128–137 (2015)

[4] *Blarney*, https://github.com/mn416/blarney

[5] K. Claessen. *Embedded Languages for Describing and Verifying Hardware*. PhD thesis, Chalmers University of Technology and Göteborg University (2001)

Timetable and Milestones

25th October to 21st November

Research property-based testing for software and also hardware, making sure to fully understand the related work. Setup and test the SmallCheck/QuickCheck libraries for Haskell, to better understand how they work and to improve skills with Haskell. Learn to use Blarney, building some example circuits as a learning exercise.

Milestones: Small Blarney modules of combinatorial logic implemented. Structure of testing framework and type-based generators designed.

22nd November to 5th December

Setup BlueCheck and Bluespec to test synthesizing test benches to an FPGA. Decide with the help of Matt on final design of the testing library, with further literature study if necessary. Create the enumerative generators.

Milestones: Structure necessary to meet the success criteria laid out and ready to work on my own over the Christmas vacation.

6th to 26th December (Christmas vacation)

Complete success criteria, namely: Implement a Haskell library for automatic testing of combinatorial circuits written in Blarney. If any problems arise can use second half of Christmas vacation to finish the work, at the cost of fewer additional features. Otherwise if finished early start work on Dissertation.

Milestones: Code satisfies the first point of the success criteria. Introduction chapter of Dissertation mostly complete.

27th December to 16th January (Christmas vacation)

Start building additional features as able, with priority for synthesizing test benches to an FPGA. Write Preparation chapter of Dissertation and start work on Implementation chapter.

Milestones: Synthesized test bench runs on FPGA. First 2 chapters of Dissertation mostly complete and structure of Implementation chapter laid out. Some of the proposed additional features complete.

17th January to 27th February

Create progress report and presentation. Run the library on the example buggy circuits in preparation for Evaluation chapter. Start of lent term is very intense so use the long time at low intensity to make small improvements and find any bugs.

Milestones: Progress report and presentation done. Test data for Evaluation generated.

28th February to 12th March

Complete all additional features and clean up code. Project should be stable at this point, with all changes being only minor after this point.

Milestones: Project is complete and stable to allow writing Dissertation.

13th March to 23rd April (Easter vacation)

Finish up the Introduction and Preparation chapters and write Implementation chapter. Make only small crucial changes to code (e.g. bug fixes)

Milestones: First 3 chapters of Dissertation complete. Codebase finalized, ready for submission.

13th March to 23rd April (Easter vacation)

Write Evaluation and Conclusions chapters, running any additional tests to gather evaluation data if necessary. I can use this time to do additional work on the project if Timetable has been moved back, otherwise for revision for final exams.

Milestones: Dissertation finished, ready for submission if necessary.

24th April to 7th May

Review whole project, check the Dissertation, and spend a final few days on whatever is in greatest need of attention.

Milestones: Dissertation is polished. Submission of Dissertation.

Resources Required

Haskell: ghc version 8.6.1 required for Blarney, but the default apt get ghc installs version 8.0.2 (this is what is on MCS) so must be downloaded manually.

CL account: can be used to access required version of ghc if necessary, also to use Bluespec/BlueCheck if necessary. Resource sponsor is **Simon Moore (swm11)**

Blarney: downloaded with git clone.

FPGA: for synthesizing testing benches. Resource sponsor is **Simon Moore (swm11)**

Machines in College: similar to MCS machines, only needed if personal hardware fails.

Personal laptop and computer: -

My laptop: 2.30 GHz CPU, 8 GB RAM, 118 GB Solid-state disk for OS and 447 GB Solid-state disk for data, Windows OS with Ubuntu VM

My computer: 3.50 GHz CPU, 16 GB RAM, 256 GB Solid-state disk for OS and 1 TB Hard disk for data, Windows OS with Ubuntu VM

Contingency plans: My contingency plans against data loss are that everything will be held under git on GitHub with daily checkpoints to my Google Drive and also weekly to USB Flash Drive kept only for that purpose. My contingency plans against hardware/software failure are that I can easily transition my work to the MCS/CL machines