Vericert OOPSLA 2021 Artifact

This artifact should support the claims made in the paper "Formal Verification of High-Level Synthesis". In the paper, our tool Vericert was referred to as using the temporary name "HLSCert". The claims that can be verified by the paper are the following:

- The mechanised proof of correctness of the translation from C into Verilog is provided and can be checked and rerun.
- All 27 PolyBench benchmarks can be recompiled using Vericert.
- The cycle counts of Vericert on the benchmarks can be checked and compared against LegUp 4.0.
- If Vivado is downloaded separately, then the whole performance section can be checked, including all the graphs that appear in the paper.

Claims that are not supported by the artifact

Unfortunately, we could not include our version of LegUp 4.0 in the artifact due to license restrictions. In addition to that, LegUp was recently bought by Microchip and renamed to SmartHLS[1], which means that it also cannot be freely downloaded anymore either, and the original open source version of LegUp 4.0 is not available anymore due to server issues in Toronto[2]. We have tried contacting the authors of LegUp in Toronto, but have not heard back yet on if our version of LegUp can be shared in the artifact.

Instead, we have included the net lists that LegUp generated from the benchmarks in the artifact, with all the optimisation levels that were tried, however, it does mean that these cannot be verified again and that other optimisation options cannot be tried.

In addition to that, the Vivado synthesis tool by Xilinx[3] is also commercial (but free to download), and therefore cannot be bundled into the artifact either. This synthesis tool was used to get accurate timing information about how the design would run on an FPGA, and also give the area that the design would take up on the FPGA. To be able to reproduce these results, it would therefore need Vivado to be set up so that the scripts can run.

[1] `https://www.microsemi.com/product-directory/fpga-design-tools/5590-hls#software-download`

[2] `https://legup.eecg.utoronto.ca`

[3] `https://www.xilinx.com/support/download.html`

Kick the tyres

First, the docker image needs to be downloaded and run, which contains the git repository:

```
docker pull ymherklotz/vericert
docker run -it ymherklotz/vericert sh
```

Then, one just has to go into the directory which contains the git repository and open a `nix-shell`, which will load a shell with all the correct dependencies loaded:

```
cd /vericert
nix-shell
```

Then, any commands can be run in this shell to run `vericert`, which has already been compiled and can be found in the `/vericert/bin` directory. For a quick test that it is working, a few very simple examples in the `/vericert/test` directory can be run by using the following inside of the `/vericert` directory:

```
make test
```

If this finishes without errors, it means that Vericert is working correctly.

Detailed Artifact Description

This section describes the detailed instructions to get the results for the different sections of the paper, first describing the structure of the proof and how to execute Vericert manually, to finally running Vericert on the benchmarks and get the cycle counts for the Vericert designs as well as the precompiled LegUp designs.

Directory structure of Vericert

The main directory structure of Vericert is the following:

`/src` Contains all the Coq and OCaml source files used for Vericert. The whole proof of correctness is therefore in this directory.

`/lib` This directory contains CompCert, on which Vericert is built upon. Vericert tries to separate CompCert and uses it only as a library, redefining a different top-level.

`/benchmarks` Contains the PolyBench/C benchmarks used as an evaluation in the paper, which are stored under `polybench-syn` for the benchmarks without dividers, and `polybench-syn-div` for the benchmarks with dividers.

`/docs` Contains a website and an `org-mode` file with some light documentation of the tool.

`/example` Contains some interesting observations that were made during the development, which are not directly relevant to Vericert.

`/include` Contains the divider implementation which can be imported and used in C files to get the better performance out of Vericert, instead of using native division.

`/ip` Contains hardware divider implementations which will be used in the future instead of the software implementation that is currently used in `/include`.

`/scripts` Contains some miscellaneous scripts and the `Dockerfile` which has been added for this artifcat.

`/test` Contains some very light test cases which are some minimal examples for working constructs.

Description of the proof

The proof is mostly located in `/src/hls`, which contains the proof of correctness of the 3AC to HTL transformation, as well as the transformation from HTL to Verilog. Any other files in the `/src/hls` directory that are not mentioned below are there for future optimisations such as scheduling, which are not used.

`/src/Compiler.v` The very top-level of the proof is located here and it contains the main proof of the compiler, which is the proof that the `transf_hls` function is correct, which takes C and outputs Verilog. The main proof of correctness is in the Theorem called `transf_c_program_correct`, which says that if the `transf_hls` function succeeded, that the backward simulation should hold between C and Verilog.

`/src/common` This directory contains some common library extensions and proofs that are used in other parts of Vericert. This includes the proof of correctness of Section 2.2.3, which is located in `/src/common/IntegerExtra.v` under the Theorem `shrx_shrx_alt_equiv`.

`/src/hls/Verilog.v` This file contains the whole Verilog semantics, together with the proof that the Verilog semantics are deterministic. This implements Section 3 from the paper.

`src/hls/Veriloggen.v` This file contains the generation of Verilog from HTL.

`src/hls/Veriloggenproof.v` This file contains the correctness proof of the generation of Verilog from HTL.

`/src/hls/HTL.v` This file contains the definition of the HTL intermediate language, together with its semantics.

`/src/hls/HTLgen.v` This file contains the generation of HTL from 3AC, which is the first step in the HLS transformation.

`/src/hls/HTLgenspec.v` This file contains the high-level specification of the translation from 3AC into HTL, together with a proof of correctness of the specification.

`/src/hls/HTLgenproof.v` This file contains the proof of correctness of the HTL generation from 3AC, where the main parts of the proof are the generation of Verilog operations, as well as the change in the memory model (load and store instructions).

`/src/hls/Memorygen.v` This file contains the definition and proof of the transformation which replaces naïve loads and stores into a proper RAM, which is described in Section 2.2.2.

`/src/hls/ValueInt.v` Contains our definition of values that are used in the Verilog semantics, and differ from the values used by CompCert, as they don't have a pointer type anymore.

`/src/hls/Array.v` Contains our definition of the memory model, which is a dependently typed array, which encodes its length. This is much more concrete than CompCert's abstract memory model, and closer to how it is actually modelled in hardware.

`/src/hls/AssocMap.v` Definition of association maps, which is the type that is used for Γ and Δ in Section 3.

How to manually compile using Vericert

To compile arbitrary C files, the following command can be used:

```
vericert main.c -o main.v
```

Which will generate a Verilog file with a corresponding test bench. The Verilog file can then be simulated by using the Icarus Verilog simulator:

```
iverilog main.v -o main
./main
```

This should print out the return value from the main function in addition to the number of cycles that it took to execute the hardware design.

Getting cycle counts for Vericert

There are two benchmark sets for which the results are given in the paper:

`/vericert/benchmarks/polybench-syn` Contains the PolyBench/C benchmark without any dividers, and instead the dividers are replaced by calls to `sdivider` and `smodulo` in `/vericert/include/hls.h`.

`/vericert/benchmarks/polybench-syn-div` Contains the PolyBench/C benchmark with dividers.

To get the cycle counts for Vericert from the benchmarks, the benchmarks can be compiled using the following:

```
cd /vericert/benchmarks/polybench-syn
```

or

```
cd /vericert/benchmarks/polybench-syn-div
```

depending on which benchmark should be run, and then running:

```
make
```

This will generate all the binaries for the simulation and execution of the C code. The cycle counts of the hardware can then be gotten by running:

```
./run-vericert.sh
```

This can take a while to complete, as simulation of hardware is quite slow. After around 30 minutes, there should be a `exec.csv` file which contains the cycle counts for each of the 27 benchmarks.

Getting the cycle counts for LegUp

Unfortunately, the benchmarks cannot be compiled from C to Verilog using LegUp, as it could not be included in the artifact, and does not seem to be freely available anymore.

However, our compiled Verilog designs from LegUp have been included for all the optimisation options that were tested for in the paper in Section 5.

Rebuilding the Docker image

The docker image can be completely rebuilt from scratch as well, by using the Dockerfile that is located in the Vericert repository at `/vericert/scripts/docker/Dockerfile`, which also contains this document.

To rebuild the docker image, one first needs to download the legup results for the benchmarks without divider[4] and with divider[5], and the tar files should be placed in the same directory as the `Dockerfile`. Then, in the `docker` directory, the following will build the docker image, which might take around 20 minutes:

```
docker build .
```

Then, using the hash it can be run in the same way as the docker container that was linked to this artifact:

```
docker run -it <hash> sh
```

[4] https://imperialcollegelondon.box.com/s/ril1utuk2n88fhoq3375oxiqcgw42b8a

[5] https://imperialcollegelondon.box.com/s/94clcbjowla3987opf3icjz087ozoi1o