-Standalone compilation using hclang and hclang++

Description of the standalone compilers hclang and hclang++

Introduction

The C++11/C++14 frontend of Hector is being shipped with two executables hclang and hclang++ that can be used to compile design files outside of Hector. They behave like regular compiler executables (like gcc/g++ or clang/clang++) with the only difference that they also produce formal information that can be used by Hector to generate the formal model of the design. Their command line options are compatible with clang/clang++, which are very similar to the gcc/g++ command line options. The hclang compiler is the default for compiling C code, the hclang++ compiler for C++ code. In the remainder of this document we only refer to hclang++ but everything mentioned here also applies to hclang as well.

The intention of having standalone executables is to be able to re-use existing Makefiles without modifications. The hclang++ compiler should be used for the following tasks:

- Compilation of source files into object files
- Linking of object files into an executable

Furthermore, object files produced by hclang++ can be archived into static libraries. These static libraries can then be used by hclang++ when linking an executable.

Note that linking an executable with dynamic libraries is currently not supported by hclang++. The link will succeed but the formal information in the dynamic library is lost in the linking process.

Note that all the source files, including libraries that are being linked into the executable must be compiled using hclang++. Otherwise, the formal information for source files not compiled using hclang++ won't be available to Hector.

Prerequisites

The PATH variable needs to be set up such that it points to the hclang++ executable. We assume that the user already has a HECTOR_HOME environment variable that points to the Hector installation:

```
export HECTOR_HOME=<path to Hector installation>
export PATH=$HECTOR_HOME/frontend/linux64/bin:$PATH
```

For accessing 32-bit executables, refer to directory linux instead of linux64. Running hclang++ does not require any licensing setup.

Recommended Steps

The hclang++ executable can be invoked just like g++ or clang++. Most command line options understood by gcc/g++ can also be used with hclang++. Some options, e.g. optimization levels `-O<n>` are accepted but ignored because hclang++ needs non-optimized debug information to generate the formal information.

Note that the executable should contain the top-level Hector function that has the calls to Hector::registerInput / Hector::registerOutput / Hector::beginCapture and Hector::endCapture in it. Currently, it is preferable that the top-level Hector function is named "main" because running "gdb" on the executable will automatically start in "main".

Example

In order to compile an executable from a C++ source file dut.cpp, use the following command:

```
hclang++ -o dut.exe dut.cpp
```

To first produce object files and then link them together into an executable, use this:

```
hclang++ -c -o dut1.o dut1.cpp
hclang++ -c -o dut2.o dut2.cpp
hclang++ -o dut.exe dut1.o dut2.o
```

To build a static library from object files and then use that library when linking the executable, use the following commands:

```
hclang++ -c -o lib1.o lib1.cpp
hclang++ -c -o lib2.o lib2.cpp
ar rs libmylib.a lib1.o lib2.o

hclang++ -o dut.exe dut.cc -lmylib
```

Importing the formal information into Hector

An executable that was generated by hclang++ contains all the formal related information that is necessary to build a formal model (DFG). The hclang++ flow requires enabling the C++11/C++14 frontend in Hector:

```
set_hector_comp_use_new_flow true
```

Then, the `cppan -import` command is used to import the formal information. For example, if the compiled executable is named spec.exe and the Hector top-level function is named “main”, use the following lines in the Hector script to produce the formal model:

```
create_design spec -top main
cppan -import spec.exe
compile_design
```