An Introduction to CHDL

Chad D. Kersey

Introduction

- CHDL is a C++ Hardware Design Library.
- Open source: GNU LGPL.
- Its name is a pun on a popular and well-established HDL, VHDL.
- Created to allow C++ template metaprogramming in hardware designs.
- This talk is a companion to the textual tutorials on `cdkersey.com`.

Prerequisites

The following are needed to get and use CHDL:

- C++ compiler supporting C++11 (GCC 4.7 or higher)
- Unix-like OS (Linux, Mac OS with Homebrew, Windows with Cygwin)
- Understand how libraries work on your platform of choice.
- Gnu Make
- Optional: Waveform viewer (GTKWave)

Exercise: Write a library in C++ that contains a single function `say_hi()` that prints "Hello, world!" to the screen. Compile and link it as a shared object and write a program, in a separate directory, that calls this function once and exits.

Getting the Source

The source can be obtained with the git client:

Downloading the Source

```
$ git clone https://github.com/cdkersey/chdl.git
```

Or from GitHub as a `.zip` archive:

https://github.com/cdkersey/chdl/archive/master.zip

Building CHDL

Once you have the code cloned or unpacked, you can build it using GNU Make.

Compiling the Source

```
$ cd chdl
$ make -j 8
$ sudo make install
```

The installation phase is optional and installs by default into /usr/local or whatever is set in the environment variable `PREFIX`. If this produces any unexpected errors, please report them to `cdkersey@gatech.edu`.

Running the Tests

Now you have CHDL built. You can verify it has built correctly using the test programs in the `test/` directory:

Running the Tests

```
chdl$ cd test
test$ make -j 4
test$ LD_LIBRARY_PATH=../ make run
```

This example is specific to Linux. On MacOS, use `DYLD_LIBRARY_PATH` instead of `LD_LIBRARY_PATH`. This can also be set to `$PREFIX/lib`.

Exercise: Build and install CHDL on your machine and run all of the tests.

Starting a Project

Creating the Project Files

```
$ mkdir 0_blinkenlights
$ cd 0_blinkenlights
$ touch Makefile; touch blinkenlights.cpp
```

Remember to use tab characters for indentation in makefiles.

Makefile

```
CXXFLAGS ?= -std=c++11
LDLIBS ?= -lchdl

blinkenlights: blinkenlights.cpp

clean:
	rm -f blinkenlights
```

Starting a Project

The most basic CHDL program. We could replace CHDL with any of thousands of other libraries and the structure would be the same. This is really just the most basic C++ program:

Boilerplate CHDL Code

```
#include <iostream>
#include <fstream>

#include <chdl/chdl.h>

using namespace std;
using namespace chdl;

int main() {
  return 0;
}
```

Starting a Project

Let's build our simple CHDL program:

Building our CHDL code:

```
0_blinkenlights$ make
0_blinkenlights$ ./blinkenlights
```

It does nothing, but it should do nothing without any error messages at least!

Building and Simulating a Simple Design

Now add the following code to your `main()` function just before the return statement:

Simple Example

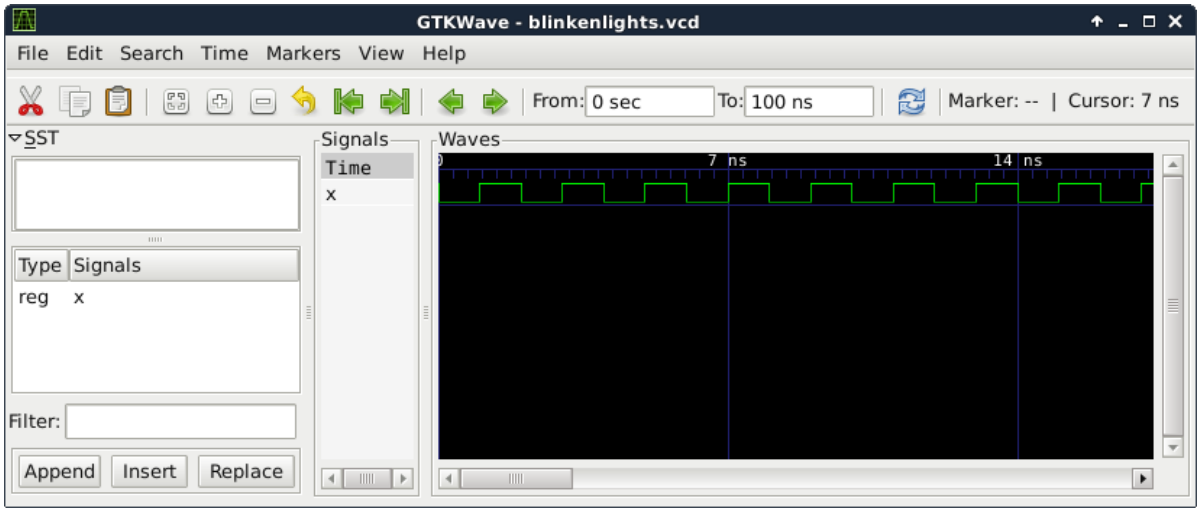
```
node x;
x = Reg(!x);
TAP(x);

ofstream vcd("blinkenlights.vcd");
run(vcd, 100);
```

Build and run your example again.

Building and Simulating a Simple Design

Viewing the Waveform

```
0_blinkenlights$ make
0_blinkenlights$ ./blinkenlights
0_blinkenlights$ gtkwave blinkenlights.vcd
```



Analysis of the Example

- CHDL program *generate*s hardware.
- `x` is a `node`; a digital signal.
- All `nodes` have one source and arbitrarily many sinks.
- A `node` can be considered synonymous with the gate (logic function) that provides its value.
- The `Reg` function creates a D flip-flop, shifting input in time by one clock period. This is the basic unit of storage.
- All cycles must pass through at least one `Reg`.

Exercise: Create a second node, `y`, that blinks half as fast as `x`.

Parallel Examples

Verilog, VHDL, and CHDL for comparison:

```
module blink(out);
  output out;
  reg phi, out;

  initial
    begin
       $dumpfile("dump.vcd");
       $dumpvars(2, blink);
       phi = 0;
       out = 0;
       #100
       $finish();
    end

   always
     begin
       #1 phi = !phi;
     end

   always @(posedge phi)
     begin
       out = !out;
     end
endmodule
```

```
library ieee;
use ieee.std_logic_1164.all;

entity blink is
end blink;

architecture rtl of blink is
  signal x, phi : bit;
begin
  process
  begin
    phi <= '0';
    wait for 5 ns;
    phi <= '1';
    wait for 5 ns;
  end process;

  process(phi)
  begin
    if (phi'event and phi='1') then
      x <= not x;
    end if;
  end process;
end rtl;
```

```
#include <fstream>
#include <chdl/chdl.h>

using namespace chdl;
using namespace std;

int main() {
  node x;

  x = Reg(Inv(x));

  TAP(x);

  ofstream vcd("blink.vcd");
  run(vcd, 10);

  return 0;
}
```

For simple examples, they are similar. Limitations become apparent as designs become complex.

4-bit Counter

4-bit Counter Function

```
bvec<4> Ctr() {
  bvec<4> c;
  c = Reg(c + Lit<4>(1));
  return c;
}
```

- Nodes can be combined into bvecs.
- Arithmetic and comparison operators are overloaded.
- Note function capitalization conventions: capital implies hardware.
- Note format for literal.
- bvec is just an alias for vec<node>.

Indexing bvecs

Indexing bvec

```
vec<256, bvec<8> > x;
for (unsigned i = 0; i < 256; ++i)
  x[i] = Lit<8>(i);

bvec<32> addr;
bvec<20> tag = addr[range<12,31>()];
bvec<8> idx = addr[range<4.11>()];
bvec<4> offset = addr[range<0,3>()];
```

- `range` is safe but values must be compile time constant.
- Indexing by integer can lead to run-time errors from out-of-range indices.
- There is no way to do variable ranges; these must be assigned with nested loops.

Multiplexers

As an example of a CHDL library function, consider the multiplexer:

Mux Example

```
vec<8, bvec<8> > matrix;

bvec<3> sel;
bvec<8> byte = Mux(sel, matrix);
```

Because of the way it is designed (using C++ templates), it can take a CHDL `vec` of any type, as long as it is ultimately comprised of `nodes`, and provide a way to index it.