An introduction to the AHIR-v2 tool-set

Madhav P. Desai

January 25, 2014

The purpose of the toolset

- Aid in the design and implementation of complex systems (in hardware).
- Describe the algorithm (behaviour) of the system using high-level languages (currently C), and migrate the algorithm to a hardware implementation.

For example

Design a circuit that takes two inputs, computes the maximum of the two and returns the maximum.

```
uint32_t maxOfTwo(uint32_t a, uint32_t b)
{
  uint32_t c = ((a > b) ? a : b);
}
```

Lets write a test-program to check this code

```
uint32_t maxOfTwo(uint32_t, uint32_t);
int main(int argc, char* argv[])
{
  if(argc < 3)
  {
     printf(stderr, "%s <uint32_t> <uint32_t>\n", argv[0]);
     return(1);
  }
  uint32_t c = maxOfTwo(atoi(argv[1]), atoi(argv[2]));
  printf(stdout,"Result = %d.\n", c);
  return(0);
}
```

We compile and run it

Go to ex1/ and check it out.

Conversion to hardware

The AHIR-v2 flow translates the code to hardware in four steps:

1. The source C code is compiled using the open-source compiler **clang** (we version 2.8), which produces byte-code in **llvm** format.
2. We take the **llvm** bytecode and translate it to our internal representation called **Aa**. This is done by a utility called **llvm2aa**.
3. Multiple **Aa** files can be linked using an **Aa** linker tool called **AaLinkExtMem**.
4. The linked **Aa** code is then converted to a virtual circuit using a **vC** representation. This is done using a utility called **aa2vc**.
5. Finally, the **vC** code is converted to VHDL which can then be simulated or synthesized. This conversion is done using a utility called **vc2vhdl**.

At each stage, various optimizations are carried out.

Lets see what happens

Go back to ex1/ and check it out.

Validating the VHDL

- In order to test the VHDL, we will use the same program that was used to test the original C code.
- Two processes are started: one executing the test-program and the other executing a VHDL simulator which simulates the generated VHDL. The two processes communicate through sockets.
- There is no need to write a new test-bench!

Lets take a look

Go back to examples/ex1 and check it out.

Lets make it a little more interesting

```
void maxDaemon()
{
  while(1)
  {
        uint32_t a = read_uint32("in_data");
        uint32_t b = read_uint32("in_data");
        c = ((a > b) ? a : b);
        write_uint32("out_data", c);
  }
}
```

The testbench for the Daemon

```
int main(int argc, char* argv[])
{
  uint32_t a, b, c;
  while(1)
  {
     scanf("%d", &a);
     scanf("%d", &b);
     write_uint32("in_data", a);
     write_uint32("in_data", b);
     c = read_uint32("out_data");
     printf(stdout,"Result = %d.\n", c);
  }
  return(0);
}
```

The testbench for the Daemon

Lets check out the software and the hardware generated from it.
Go to ex2/.

Installing AHIR-v2

- You will need to install llvm-2.8, clang-2.8, the BOOST graph library.
- You can download the source code from gitHub.
- Go to the v2/ folder in the git repository. You will need to source "build_bashrc", and then type "scons" to build the executables. You can then make a release by going back to the root of the git repository and then typing "make -f ReleaseMakeFile". This populates the Release directory with binaries, examples etc. which should be enough to get you going.