On verifying AhirV2 generated VHDL using software testbenches

Madhav Desai
Department of Electrical Engineering
Indian Institute of Technology
Mumbai 400076 India

September 2, 2014

The AhirV2 tool chain can be used to convert parts of a C program to VHDL (essentially, some of the functions in a program are mapped to VHDL). To verify the resulting VHDL, one would like to simulate it in a VHDL simulator (such as Modelsim from Mentor Graphics). The most natural way to do this is to use the original program itself as a testbench for this purpose.

- Stubs are created for the set of functions which are mapped to VHDL by the AhirV2 flow.
- The software testbench is compiled and linked with these stubs.
- Whenever a stub function is called, it tries to connect with a server created by the VHDL simulation process.
- The VHDL simulation process listens for calls from the stubs and exchanges data between the stubs and the actual VHDL being simulated.

1 An example

Consider the following program (lets say it is in file "prog.c"):

```
#include <stdlib.h>
#include <stdint.h>
#include <Pipes.h>
#include <stdio.h>


// note: initialized value..
uint32_t sum1 = 23;
uint32_t sum2 = 39;
```

```
// note: no problems with pointers :-)
uint32_t* tgt[2] = {&sum1, &sum2};

uint32_t get_sum(uint32_t idx)
{
  return(*(tgt[idx]));
}

void accumulate()
{
  int i = 0;
  while(1)
  {
     int nxt = read_uint32("in_data");
#ifdef SW
     printf("read %u\n", nxt);
#endif
     // ugly, but this is just a demo,
     // we are showing off.
     *(tgt[i])= ((*tgt[i]) + nxt);

     write_uint32("out_data",*(tgt[i]));
#ifdef SW
     printf("wrote %u\n", *(tgt[i]));
#endif
     i = 1 - i;
  }
}
```

This program describes a *system* which listens for data on a pipe "in_data", and sends data out on a pipe "out_data". The incoming data is accumulated into the variable *sum*, and there are two methods to set and get the value of *sum*.

Now to test this program, we can write a test-bench such as this one (lets call this file "testbench.c").

```
#include <pthread.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#include <Pipes.h>
#ifdef SW
#include <pipeHandler.h>
#include "prog.h"
```

```
#else
#include "vhdlCStubs.h"
#endif

void Exit(int sig)
{
fprintf(stderr, "## Break! ##\n");
exit(0);
}


// The model of the hardware accumulate thread
// is necessary when building the SW testbench.
#ifdef SW
DEFINE_THREAD(accumulate)
#endif

int main(int argc, char* argv[])
{
  signal(SIGINT,  Exit);
  signal(SIGTERM, Exit);

  uint32_t data_in[10], data_out[10];
  int i;

#ifdef SW
  init_pipe_handler();

  // register FIFO
  register_pipe("in_data",10,32,0);
  // register FIFO..
  register_pipe("out_data",10,32,0);

#endif

#ifndef SW
  // to set the initial value of sum.
  // in the hardware version, storage
  // variables are initialized by calling
  // this function (auto-generated by
  // the Aa linker AaLinkExtMem)
  global_storage_initializer_();
#endif


#ifdef SW
```

```
  // start the accumulate thread.
  // (this is necessary only in the SW model)
  PTHREAD_DECL(accumulate)
  PTHREAD_CREATE(accumulate)
#endif


  for(i = 0; i < 10; i++)
  {
    data_in[i] = i;
  }


  // write 10 things to in_data, it has enough room..
  write_uint32_n("in_data",(uint32_t*)data_in, 10);

  // read back 10 things from out_data..
  read_uint32_n("out_data",(uint32_t*)data_out, 10);


  fprintf(stdout,"from out_data, we read ");
  for(i=0; i < 10; i++)
    fprintf(stdout," %u ", data_out[i]);
  fprintf(stdout,"\n");


  fprintf(stdout,"Sum 0 is %d\n",get_sum(0));
  fprintf(stdout,"Sum 1 is %d\n",get_sum(1));

#ifdef SW
  close_pipe_handler();
#endif
}
```

In the SW case, the test-bench starts the accumulate thread, writes data to the hardware, and reads back stuff from the hardware. In the non-SW case, there is no need to start the accumulate thread, since it exists in the hardware model.

Obviously, we would prefer to use the same test-bench to verify that the VHDL system generated from "prog.c" functions correctly. The difference is that instead of using the pipeHandler, the test-bench now uses methods in SocketLib. Further, the VHDL is executed in a VHDL simulator; the simulator communicates with the testbench using sockets. The *ifdef's* in the test-bench and the system program indicate the difference between the pure software version of the system-test-bench combination and the hardware-software version.

The following Makefile builds a software-only testbench executable, and also converts the system described in prog.c to VHDL. The same testbench can be used to test the VHDL also.

```
# build software version of testbench (to check the "desired behaviour")
SOCKETLIB_INCLUDE=$(AHIR_RELEASE)/CtestBench/include
SOCKETLIB_LIB=$(AHIR_RELEASE)/CtestBench/lib
PIPEHANDLER_INCLUDE=$(AHIR_RELEASE)/pipeHandler/include
PIPEHANDLER_LIB=$(AHIR_RELEASE)/pipeHandler/lib
PTHREADUTILS_INCLUDE=$(AHIR_RELEASE)/pthreadUtils/include
VHDL_LIB=$(AHIR_RELEASE)/vhdl
VHDL_VHPI_LIB=$(AHIR_RELEASE)/CtestBench/vhdl
FUNCTIONLIB=$(AHIR_RELEASE)/functionLibrary/
SRC=./src
all: SW HW
TOAA:c2llvmbc llvmbc2aa  aalink
TOVC:c2llvmbc llvmbc2aa  aalink aa2vc
VC2VHDL: vc2vhdl  vhdlsim
AA2VHDLSIM: aa2vc  vc2vhdl  vhdlsim
TOVHDL:TOVC vc2vhdl

# llvm2aa opts: pipelined case, extract-do-while.
#LLVM2AAOPTS=--storageinit=true
LLVM2AAOPTS=-extract_do_while=true --storageinit=true -pipedepths=pipedepths.txt

PROGDEFS=

# the top-level modules
# -T specifies top-level daemon module
# -t specifies top-level slave module.
#
TOPMODULES=-T accumulate -t get_sum -t global_storage_initializer_


# compile with SW defined.
# note the use of IOLIB in building the testbench.
SW: $(SRC)/prog.c $(SRC)/prog.h $(SRC)/testbench.c
gcc -g -c -DSW $(PROGDEFS) -I$(PIPEHANDLER_INCLUDE)\
              -I$(FUNCTIONLIB)/include -I$(SRC) $(SRC)/prog.c
gcc -g -c -DSW $(PROGDEFS) -I$(PIPEHANDLER_INCLUDE)\
              -I$(PTHREADUTILS_INCLUDE) -I$(SRC) $(SRC)/testbench.c
gcc -g -o testbench_sw prog.o testbench.o\
              -L$(PIPEHANDLER_LIB) -lPipeHandler -lpthread

# five steps from C to vhdl simulator.
HW: c2llvmbc llvmbc2aa  aalink aa2vc  vc2vhdl  vhdlsim

AA2VHDL: aa2vc vc2vhdl vhdlsim
```

```
# C to llvm byte-code.. use clang.
c2llvmbc: $(SRC)/prog.c $(SRC)/prog.h
clang -O3 -std=gnu89 $(PROGDEFS)  -I$(SOCKETLIB_INCLUDE)\
              -I$(FUNCTIONLIB)/include -emit-llvm -c $(SRC)/prog.c
opt --indvars --loopsimplify prog.o -o prog.opt.o
llvm-dis prog.opt.o

# llvm byte-code to Aa..
llvmbc2aa:  prog.opt.o
llvm2aa $(LLVM2AAOPTS)  prog.opt.o | vcFormat >  prog.aa

# Aa to vC
aalink: prog.aa
AaLinkExtMem prog.aa | vcFormat > prog.linked.aa
AaOpt -B prog.linked.aa | vcFormat > prog.linked.opt.aa

aa2vc: prog.linked.opt.aa
Aa2VC -O -C prog.linked.opt.aa | vcFormat > prog.vc

# vC to VHDL
vc2vhdl: prog.vc
vc2vhdl -O -S 4 -I 2 -v -a -C -e ahir_system\
                 -w -s ghdl $(TOPMODULES) -f prog.vc
vhdlFormat < ahir_system_global_package.unformatted_vhdl\
                 > ahir_system_global_package.vhdl
vhdlFormat < ahir_system.unformatted_vhdl\
                 > ahir_system.vhdl
vhdlFormat < ahir_system_test_bench.unformatted_vhdl\
                 > ahir_system_test_bench.vhdl

# build testbench and ghdl executable
# note the use of SOCKETLIB in building the testbench.
vhdlsim: vhdlTb ghdlModel

vhdlTb: $(SRC)/testbench.c vhdlCStubs.h vhdlCStubs.c
gcc -c vhdlCStubs.c  -I$(SRC)\
                -I./ -I$(SOCKETLIB_INCLUDE)
gcc -c $(SRC)/testbench.c\
                -I$(PTHREADUTILS_INCLUDE) -I$(SRC)\
                -I./ -I$(SOCKETLIB_INCLUDE)
gcc -o testbench_hw testbench.o vhdlCStubs.o\
                -L$(SOCKETLIB_LIB) -lSocketLib -lpthread

ghdlModel: ahir_system.vhdl ahir_system_test_bench.vhdl ahir_system_global_package.vhdl
ghdl --clean
```

```
ghdl --remove
ghdl -i --work=GhdlLink  $(VHDL_LIB)/GhdlLink.vhdl
ghdl -i --work=ahir  $(VHDL_LIB)/ahir.vhdl
ghdl -i --work=aHiR_ieee_proposed\
                $(VHDL_LIB)/aHiR_ieee_proposed.vhdl
ghdl -i --work=work ahir_system_global_package.vhdl
ghdl -i --work=work ahir_system.vhdl
ghdl -i --work=work ahir_system_test_bench.vhdl
ghdl -m --work=work -Wl,-L$(SOCKETLIB_LIB) -Wl,-lVhpi ahir_system_test_bench

PHONY: all clean
```

To test the software, run testbench_sw. To verify the hardware (using the VHDL simulator GHDL), start testbench_sw in one shell, and then start ahir_system_test_bench in a different shell.