On verifying AhirV2 generated VHDL using software testbenches

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

March 4, 2012

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 <stdio.h>
// this file provides definitions
// of pipe access functions read/write.
#include <Pipes.h>

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

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


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!
*(tgt[i])= ((*tgt[i]) + nxt);

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


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

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>

#ifdef SW
// for the pipe read/write functions
```

```
#include <Pipes.h>
// for the pipeHandler daemon..
#include <pipeHandler.h>
#include "prog.h"
#else
#include "vhdlCStubs.h"
#endif

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

void *accumulate_(void* fargs)
{
   accumulate();
}

void *write_pipe_(void* a)
{
   write_uint32_n("in_data",(uint32_t*)a, 10);
}

void *read_pipe_(void* a)
{
   read_uint32_n("out_data",(uint32_t*)a, 10);
}

void* pipeHandler__(void* a)
{
pipeHandler();
}


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

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

#ifdef SW

// in the "software" case, we start a pipe-handler
// to manage the pipes..  Earlier, we were using named
// pipes, which were very flaky and difficult to debug.
// the pipeHandler generates a log file (pipeHandler.log)
// which is very useful for figuring out what happened.
// one can also use gdb to trace activity.
pthread_t phandler_t;
pthread_create(&phandler_t,NULL,&pipeHandler__,NULL);

usleep(100);

// register two FIFOs..
register_pipe("in_data",10,32,0);
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



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

pthread_t acc_t, wpipe_t, rpipe_t;

#ifdef SW

pthread_create(&acc_t,NULL,&accumulate_,NULL);
#endif
pthread_create(&wpipe_t,NULL,&write_pipe_,(void*)data_in);
pthread_create(&rpipe_t,NULL,&read_pipe_,(void*)data_out);


pthread_join(wpipe_t,NULL);
pthread_join(rpipe_t,NULL);
```

```
  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
pthread_cancel(acc_t);
killPipeHandler();
#endif
}
```

The test-bench starts three threads: one to write data to pipe in_data, one to read data from pipe out_data, and one to run the accumulate function. After the last data is read back from out_data, the test-bench gets the value of sum and prints it out.

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.

```
SOCKETLIB_INCLUDE=../../CtestBench/include
SOCKETLIB_LIB=../../CtestBench/lib
VHDL_LIB=../../vhdl
VHDL_VHPI_LIB=../../CtestBench/vhdl
all: SW HW

# compile with SW defined... this
# tests the SW.
SW: prog.c prog.h testbench.c
gcc -c -DSW -I$(SOCKETLIB_INCLUDE) prog.c
gcc -c -DSW -I$(SOCKETLIB_INCLUDE) testbench.c
gcc -o testbench_sw prog.o testbench.o -L$(SOCKETLIB_LIB) -lSocketLib -lpthread

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

# C to llvm byte-code.. use clang.
c2llvmbc: prog.c prog.h testbench.c
```

```
clang -std=gnu89 -I$(SOCKETLIB_INCLUDE) -emit-llvm -c prog.c
llvm-dis prog.o

# llvm byte-code to Aa..
llvmbc2aa:  prog.o
llvm2aa --storageinit=true prog.o | vcFormat >  prog.aa

# Aa to vC
aa2vc: prog.aa
AaLinkExtMem prog.aa | vcFormat > prog.linked.aa
Aa2VC -O -C prog.linked.aa | vcFormat > prog.vc

# vC to VHDL
vc2vhdl: prog.vc
vc2vhdl -C -e ahir_system -w -s ghdl -T accumulate\
               -t get_sum -t global_storage_initializer_ -f prog.vc
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: ahir_system.vhdl ahir_system_test_bench.vhdl testbench.c vhdlCStubs.h vhdlCStub
gcc -c vhdlCStubs.c -I./ -I$(SOCKETLIB_INCLUDE)
gcc -c testbench.c -I./ -I$(SOCKETLIB_INCLUDE)
gcc -o testbench_hw testbench.o vhdlCStubs.o  -L$(SOCKETLIB_LIB) -lSocketLib -lpthread
ghdl --clean
ghdl --remove
ghdl -i --work=ahir  $(VHDL_LIB)/ahir.vhdl
ghdl -i --work=ieee_proposed  $(VHDL_LIB)/ieee_proposed.vhdl
ghdl -i --work=work $(VHDL_VHPI_LIB)/Utility_Package.vhdl
ghdl -i --work=work $(VHDL_VHPI_LIB)/Vhpi_Package.vhdl
ghdl -i --work=work ahir_system.vhdl
ghdl -i --work=work ahir_system_test_bench.vhdl
ghdl -m --work=work -Wc,-g -Wl,-L$(SOCKETLIB_LIB) -Wl,-lVhpi ahir_system_test_bench

clean:
rm -f *.o* *.cf *.*vhdl vhdlCStubs.* *.vcd in_data* out_data* testbench_sw testbench_hw

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.