JVSIP Testing

Version 1.0

March 31, 2012

Introduction

This document describes the testing mechanism for the C VSIPL library included with the JVSIP distribution.

Note that these tests are user code and should be useable on any C VSIPL distribution as long as the distribution supports the same functionality as the JVSIP distribution (or at least the functionality of the particular test being run).

Note that any example VSIPL user program is, in a manner of speaking, a test. If one determines that a program functions as it should and gives the correct answer then this is a validation of some part of the VSIPL implementation used in the program. It is not a validation of all of the VSIPL implementation; but in the limit of lots of code being written that runs correctly then the probability of a correct VSIPL implementation is higher. However example programs are generally just that; an example of some VSIPL functionality; so we don't call them tests because it is not necessarily obvious if the example is working correctly or not.

The specific testing mechanism used here is to write many stand-alone VSIPL functions. Each function takes no arguments; checks results internal to the function against some known value, or a calculated value using a different method (Matlab for instance); and prints at least one and maybe many result checks of either *correct* (or *agree*), or *error* for whatever functionality is being tested.

To check the result of a test one searches for the word *error* in the output of the test which, if found, is an indicator that something is not working correctly. Note the "something not working" may be in the library functionality, the test itself, or in the truth value. Tracking down the cause of errors can be challenging.

JVSIP uses an automatic method (shell script) for creating a fully functional test so it is important to follow some simple guidance when writing a VSIPL test. This document gives this guidance and also tells how to compile and run the tests.

Build and Test Mechanism

There are two test scripts included inside the **c_VSIP_testing** directory.

The first script is called **gen_all.sh.** This script will create a test out of all files within the testing directory which have a **.h** extension. The script will create a file called **test_all.c** and compile the file creating an executable called **test_all**. The recommended method to run is

```
> ./gen_all.sh
> ./test_all >output
> grep error output
```

and then search the file output for error messages. You might also do

```
> ./test_all | grep error
```

if you don't think you might want to look at the output.

This test produces a lot of output so placing a filter on the test is recommended.

The second script is call **gen_atest.sh**. This will run a particular test. The script will create a file called **atest.c**, compile producing an executable called **atest** and then run **atest**. Note these files are still in the directory when the test completes, but are replaced if **gen_atest.sh** is run again.

An example run on matrix acos for float precision looks like

```
> ./gen_atest.sh macos_f.h

*******
TEST macos_f

test out of place, compact, user "a", vsipl "b"
macos_f(a,b)
 matrix a
[
1.000000  -1.000000 ;
0.707107  -0.707107 ;
0.000000  0.000000 ;
];
matrix b
[
0.000000  3.141593 ;
0.785398  2.356195 ;
1.570796  1.570796 ;
];
expected answer to 4 decimal digits
[
  0.0000    3.1416 ;
  0.7854    2.3562 ;
  1.5708    1.5708 ;
];
correct
check macos_f in place
correct
```

Methodology

The JVSIP test methodology for **test_all** uses the shell script along with a template file (called **testing.tmplt**) to create the test.   All files (tests, templates, shell scripts, etc.) are included in the directory **c_VSIP_testing.** There is no directory hierarchy. The individual test functions are written in header files which are included and called in the main test routine.

For the test **atest** no template is used and the shell script is stand-alone except for the argument which identifies the test.

For a quick-start, for the unix competent user, examination of the shell scripts and associated templates will be sufficient to understand my testing methodology. None of these files are very long or complicated.

For a quick start on writing testing routines I recommend reading some of the smaller routines such as `sin_d.h` or (for something a little more complicated, but still short) `cvmeansqval_d.h`.

The routine for main

A framework is needed for the tests. For **test_all** this is supplied by the **testing.tmplt** file and for **atest** it is generated by the shell script. As such it includes the VSIPL header file and other common files at the top, and calls the initialize and finalize routines in the proper spot so there is no need for the test routines to do this.

Printing results

Code for printing various VSIPL objects is included in header files. The appropriate header files (which contain the print function) are included in each test routine with guards (#ifndef GUARD #define GUARD) The print header file names have a suffix of *include* instead of *h*. This prevents the files from interfering with the shell scripts search for test files. They have a prefix of VU_ to indicate they are VSIP user functions.

Standard Test Function

Test functions are written in header files. The standard form is

```
static void a_test(void){
    /* some code */
}
```

The name of the file must be **a_test.h**. No files with extension ".h" should be in the testing directory unless it is a test. Since include files can have any naming convention this shouldn't be too severe a requirement. We also may not have two tests with the same name so a search before naming (`> ls | grep a_name`) is wise.

Examination of current tests and methodologies as well as generating **test_all.c** and **atest.c** files, and then examining the generated code, should be sufficient to allow anybody to write additional tests. Feel free to contact me if problems arise.

C code needed to run a_test

```
#include<stdio.h>
#include<string.h>
#include<vsip.h>

#define NDPTR_f ((vsip_scalar_f*)NULL)
#define NDPTR_d ((vsip_scalar_d*)NULL)
#define NDPTR_i ((vsip_scalar_i*)NULL)
#define NDPTR_si ((vsip_scalar_si*)NULL)

/* #include"func.h /
#include”a_test.h”
/ include as many tests as you want */
int main(){
  vsip_init((void*)0);
  /* func(); /
a_test();
/* call each test here */
  vsip_finalize((void*)0);
  return 0;
}
```