JVSIP Testing

Version 0.9

December 19, 2011

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 C VSIPL (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 mechanisim 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; and prints at least one and maybe many result checks of either *correct* (or *agree*), or *error* for whatever functionality is being tested. What is actually checked for is the printed error which is an indicator that something is not working correctly.

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

The JVSIP test methodology for the C library uses a shell script along with some template files to create a C program called `test_all.` All files (tests, templates, shell scripts, etc.) are included in the directory c_VSIP_testing. There is no directory hierachy. The individual test functions are written in header files which are included and called in the main test routine.

For a quick-start, for the unix competent user, examinination of the files
`Makefile`, `testing.tmplt`, and `gen_all.sh`
will be sufficent 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

The purpose of the `testing.tmplt` file is to provide a framework for the tests. As such it includes the vsip 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. Also included is a file I call VU_print.include. This is just some printing routines for use by the test routines. Note I don't use a '.h' here because the test generator script thinks all '.h' type files are test routines.

The file `testing.tmplt` is the structure of the `main()`program. Note that the comment lines that include the string `'func'` are important to the proper working of the shell script.

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.

C code needed to run a_test

```
#include<stdio.h>
#include<string.h>
#include<vsip.h>
#include"VU_print.include"
#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;
}
```