Technical Report, IDE0706, January 2007

AUTOMATIC TESTING OF STREAMBITS

Electrical Engineering

Erik Agrell, Tim Rosenkrantz

School of Information Science, Computer and Electrical Engineering, IDE

Halmstad University

Automatic testing of StreamBits

School of Information Science, Computer and Electrical Engineering, IDE
Halmstad University
Box 823, S-301 18 Halmstad, Sweden

December 2007

Preface

This project is the result of our master thesis at Halmstad University. The project has proved to be extremely challenging from time to time but it has certainly been the most rewarding part of this education.

We would like to thank Jerker Bengtsson for every "five" minutes sessions that has lasted for an hour. We would like to thank Jonathan Andersson for taking the time to answer all the stupid questions, for being a good friend throughout the project and for making this project more challenging with every update of the framework.

Last but not least we would like to thank our supervisor Veronica Gaspes for all the support, good input and motivation throughout this project.

Thank you!

Erik Agrell

erik.agrell@telia.com

Tim Rosenkrantz

tim_rosenkrantz@ipbolaget.com

Abstract

This thesis aims to develop an automatic testing tool for StreamBits, a programming language for parallel stream processing, currently being developed by Jerker Bengtsson at Halmstad University as part of his PhD project. StreamBits is an extension of StreamIT, developed at Massachusetts Institute of Technology(MIT), to include features that make it more suitable for 3G baseband applications.

The cost of verifying the functionality of software has lead to the development of several tools for automatizing the testing process. These tools are all language specific, therefore a tool for StreamBits needs to be developed. This is done by evaluating the techniques used in other test tools designed for other programming languages and use this information to create a test tool suitable for StreamBits. The goal is to make a user friendly tool with capability of performing both specification tests and verification of stream rates.

The results of our project are a well functioning specification based testing tool implemented as a package in the Java StreamBits framework. The tool can test properties of programs using specifications written as Java predicates and can verify stream rates for single threaded parts of StreamBit programs. The tool can also handle, and perform tests on StreamBit programs that cause the framework to stall. For each test performed a detailed log is generated including results from the specification test and stream rate test.

List of Figures

List of Tables

Listings

Contents

1 Introduction

This master thesis is one of three projects that will develop the tools to make StreamBits a useful programming language. The two other projects are concerned with the parser and evaluation of streambits. This thesis is concerned with the developoment of test tool for automatic testing of StreamBits.

1.1 Automatic testing

Since software development has become more and more advanced the labor of testing software has increased tremendously. At this point the cost of testing software often exceeds cost of the rest of the development. This fact has lead to the development of a number of techniques and tools for automating the testing process. Automatic testing has several advantages compared to manual testing and there exists several tools on the market today. These tools are all language specific and therefore a tool for StreamBits is needed. Automatic testing tools can perform tests, generate data and produce logs, helping the programmer to spot errors. This saves a lot of time during software development and as a result, it saves a lot of money. It also helps to detect errors so that they can be corrected at an early stage. These facts strongly motivate the need for an automated testing tool when a programming language is being developed.

1.2 StreamBits

StreamBits is a programming language for 3g baseband applications that is currently being developed by Jerker Bengtsson at Halmstad University[1]. StreamBits is built on the same technique of stream processing as StreamIT[2], which is a language created at Massachusetts institute of technology(MIT). Both StreamBits and StreamIT use filters and pipelines to form a network of filters(fig. 1.1). The goal for Streambits is to create a parallel programming language that can be mapped efficiently onto reconfigurable architectures. StreamBits is not a standalone language yet, it is today prototyped as a framework in Java. StreamBits is aimed to improve the areas that StreamIT lacks by giving the programmer a better way of expressing data-parallelism and bit-level computations, as well as providing for a configuration stream. These improvements are done by introducing new data types for bits and by adding functions that help the programmer to handle bits and parts of data at bit level. According to Jerker Bengtsson, bit level computatations on coarse-grained architectures can be improved so the performance can be compared to bit-level computations on fine-grained field programmable gate array (FPGA) architectures[1].

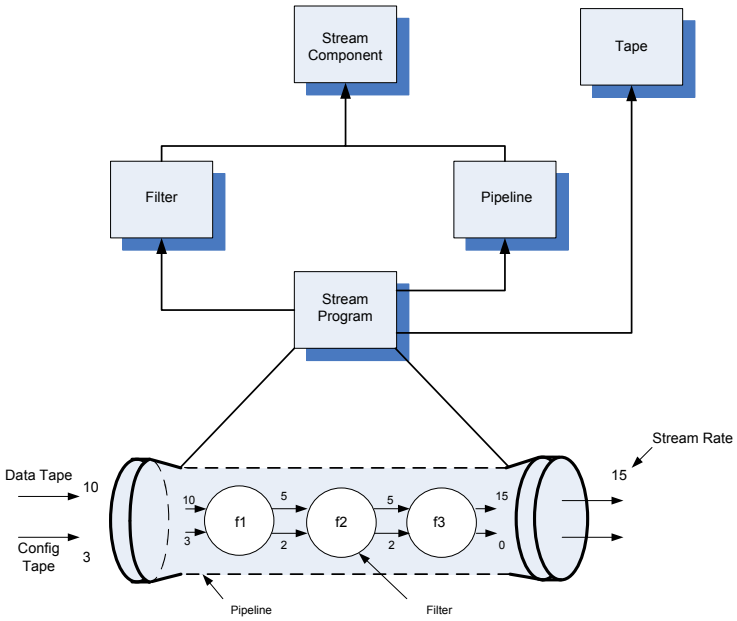

Figure 1.1: Framework abstraction

1.3 Project goals

The project aims to contribute to the implementation of tools that will support StreamBits programmers. The tool created in this project shall be used for automatic testing of programs written in StreamBits. The testing is divided into two main parts:

- Stream rate test
- Unit testing based on specifications

One problem that exists today is the problem of maintaining the specified stream rate. Each component in a StreamBits program has a defined streamrate for configuration stream and data stream. Today it is up to the programmer to maintain these stream rates. Our tool should be able to spot stream rate errors by checking the actual stream rate in runtime according to the component specification. The second major part is the unit test which should allow the programmer to express properties of different components in StreamBits and perform testing according to these specifications.

2 Literature review

A common method of testing used in object oriented programming is unit testing which is to test program parts before testing the complete program. This facilitates testing by enable the programmer to perform small tests and locate errors at an early stage in the software development phase. The classic approach to software development is the waterfall model where the complete program is implemented before any testing is performed. This guarantees that the program is properly integrated as the complete program is tested. However it can result in errors which affect every part of the program and therefore is hard to locate and correct. A more modern approach to software development is the iterative development model. This is to perform testing throughout the development phase. Unit testing facilitates and encourages modifications during development and therefore supports iterative development techniques such as extreme programming.

A limitation with unit testing is that the integration of the different parts are not tested. Therefore the complete program must also be tested after finishing unit testing to verify the integration.

There are two major methods upon which unit testing is done. The first is specification based testing [2.1] and the second is value checking based testing [2.2]. Value checking based testing is essentially comparing the output of a unit with the known correct output. Specification based testing is based upon the programmer making a thorough specification expressing the properties of the program targeted for testing. The tool then performs tests according to this specification [3]. Given a specification this technique can automate every part of the testing process. The disadvantage with value checking based testing compared to specification based testing is that it requires manually written test data.

2.1 Specification based testing

Specification based testing is the commonly used method for automated testing in object oriented programming. The technique is that the programmer writes a specification for each unit in the program. The testing tool then tests the unit by generating data automatically and on the output control different properties given in the specification. The specification is traditionally written in an informal (natural) language as a comment for each unit, that describes the expected functionality. To ensure that the program is exhaustive tested the specification must be thorough and complete. [4] [5]

2.1.1 Formal Specification languages

To enable automatic testing the specification must be written in a formal language. The formal language is especially designed for describing the behavior of a program and it can be interpreted by a testing tool. The syntax of a formal language that can be implemented by a computer program is given by a context free grammar consisting of terminals and rules how the terminals can be put together.

The example in Fig.2.1 shows the structure of a formal language with the terminals a and b and rules how these can be put together to strings consisting of a number of a´s and an equally large number of b´s such as ab or aaabbb.

Listing 2.1: Formal specification language example

```
1 tokens <int> 'a' 'b'  //terminals
2
3 S : 'a' 'b'       /* a should be followed by b */
4   | 'a' S 'b'     /* arbitrary number of instances of ab */
5   |               /* empty */
6   ;
```

Formal languages for specifications are often used as parts of annotations. Annotations are embedded in comments, starting with for example @ followed by a keyword, but instead of being of no other use than documentation, these annotations can be read by an outside program such as a testing tool. The compiler still treats the annotations as comments which means that adding annotations have no impact on the semantics of the programming language. The benefits of annotations are that a programmer can write specifications within functions. An outside program can read the annotations and do something useful with the information, in this case generate test code suitable for each function. [6]

One such tool is apt(annotations processing tool) which is embedded in Java 1.5. Apt is a command line based program that takes a Java file as input together with a set of commands. There are 4 libraries included in Java 1.5 which apt uses to define annotations [6].

- com.sun.mirror.apt: interfaces to interact with the tool
- com.sun.mirror.declaration: interfaces to model the source code declarations of fields, methods, classes, etc.
- com.sun.mirror.type: interfaces to model types found in the source code
- com.sun.mirror.util: various utilities for processing types and declarations, including visitors

The program reads the annotations present in the source file and sends them to the associated annotations factory which is set to produce the testcode for each annotation.

2.1.2 Design by contract

Design by contract is a software development technique for object oriented programming. This technique was originally developed by Bertrand Meyer [7] for the programming language Eiffel. The programmer writes the specifications as a contract in the interface of the program. The contract is described in a formal language to enable a testing tool such as iContract [3.2.1] to verify the performance of the method according to the contract. The contract is made up by three parts, *preconditions* , *postconditions* and *invariants*. [8]

Preconditions in the contract specify the conditions on states that must be fulfilled before methods can be called.

Postconditions define the conditions that should hold after methods complete execution. This is the main part of the contract specifying that the method performs as expected.

Invariants defines conditions that are not supposed to change when public methods are executed.

2.2 Value checking based testing

This method differs from specification based testing in the way that properties are not expressed, instead the programmer writes tables of input values and the expected output value. The tool tests the unit for each given input. The test passes if the input matches the given output. Two tools that uses this way of testing is JUnit [9] and Roast[10]. This method was mostly used in the beginning of automatic testing since there was no other effective way of doing verification and validation. The downside of this method is that it is impossible to automate the data generation, the tables have to be written manually which makes the use of value checking based testing very limited. The development of specification based testing and automatic data generation made this method obsolete and it is rarely used today.

3 Background

There exist several language specific tools for automatic testing on the market today. In this chapter some of these tools are presented. This chapter will also provide a brief introduction to programming in StreamBits.

3.1 Value checking based Testing tools

Value checking based testing tools perform tests by checking a known correct output for a specific input. These tools can perform automatic tests but lack the capability of generating test data. The programmer defines the test data in tables of inputs and corresponding outputs. To be able to determine the correct output the programmer performing the test must have full knowledge of the program that is tested. In some cases this includes access to the source code to determine every possible path [11].

The automatic process in this type of tool is the verification part, once the tables have been written the program will verify all output values and typically terminates the program once an incorrect value has been detected. For a test to pass all inputs must match the correct output.

This type of tool can be time demanding and lack flexibility as it requires manually written test data before any automatic testing can be done. Also, the number of tests that can be performed is limited.

3.1.1 JUnit

JUnit[9] is a tool for writing and running tests that was developed by Erich Gamma and Kent Beck. To perform a test the programmer must first define a test class that imports the following Java libraries.

- org.junit.* - The Junit framework
- static org.junit.Assert.* -A set of assert methods
- java.util.*

The test class should express the properties of the object under test, this is done by using assertions. An assertion is method which takes as an argument an expression that must be true for the program to work properly [12].

Junit uses assert functions with the purpose to verify that the object under test is in a valid state. For example if x should be greater than y then the assertion would be assert($x > y$). If y is greater than x the program throws an exception. The programmer should provide an assert function for each method that should be tested. The Assert library mentioned

above contains a number of common assertions such as assert - equals and not equal. As JUnit is a value checking based testing tool the data generation is not automated, but JUnit provides a framework for writing test cases and put these cases together into test suits. By testing a suit of test cases, JUnit can make a test more efficient and less time demanding. Listing 3.1 shows how to write a test that verifies that an arraylist is empty.

Listing 3.1: A single Junit test

```
1 @Test
2     public void testEmptyCollection(){
3         Collection collection = new ArrayList();
4         assertTrue(collection.isEmpty());
5     }
```

Listing 3.2 is an extension of test class above that can perform two tests, first verify that the arraylist is empty and secondly that an object was successfully added to the arraylist by validating that the size of the arraylist equals 1.[13] When using this method for defining two or more tests a setUp method must be used to initiate the variables used in the tests.

Listing 3.2: Multiple Junit tests

```
1  package junitfaq;
2      import org.junit.*;
3      import static org.junit.Assert.*;
4      import java.util.*;
5
6      public class SimpleTest {
7          private Collection<Object> collection;
8
9          @Before
10         public void setUp() {
11             collection = new ArrayList<Object>();
12         }
13
14         @Test
15         public void testEmptyCollection() {
16             assertTrue(collection.isEmpty());
17         }
18
19         @Test
20         public void testOneItemCollection() {
21             collection.add("itemA");
22             assertEquals(1, collection.size());
23         }
24     }
```

3.2 Specification based testing tools

Today testing of software for commercial use is often performed by a third part [14], because the developer is considered to know too much about the program to perform objective testing. This requires the use of specification based testing as the programmer performing the test does not have access to the source code. Many agile software development techniques such as extreme programming, which has become a commonly used

technique in object oriented programming, is also based on specification based testing [15].

Because of this specification based testing has become a popular testing technique and has lead to the development of many different specification based testing tools. This chapter will explain some of these tools.

3.2.1 iContract

iContract is a widely used tool for automated testing of Java programs. The tool was developed by Reto Kramer[16] and is built upon the design by contract technique. The tool consists of two major parts, a formal language to write the contracts and a tool that generates code for enforcing the contract.

iContract - The formal language

The formal language iContract is made up by standard *design by contract techniques* with preconditions, postconditions and invariants. The contracts are written as comments in the Java file using annotations. The conditions is declares as *@pre* , *@post* and *@invariant*. By writing the contracts as comments the file is not affected and can be compiled with any regular Java compiler which gives good compatibility.

iContract - The tool

The tool iContract is implemented as a preprocessor in Java that reads the contract and produces a decorated[1] version of the Java program. The decorated version includes code for checking and enforcing the conditions at runtime and code for throwing appropriate exceptions. Since these tests are done during runtime, there is no need to generate test data.

Example 1 (listing 3.3) shows a small program with a contract written in the iContract formal language. The program calls the Calc method which the contract is implemented on. The contract includes one precondition and one postcondition. The precondition (line 5) declares that the input (Var1) must be greater than zero. The postcondition (line 6) declares that the calculated return value should be greater than Var1.

Listing 3.3: iContract example 1

```
1  FILE: Pow.Java
2  // Program for calculating the power of 2 of a positive integer
3    interface Pow{
4    /**
5    * @pre Var1 > 0
6    * @post return > Var1
7    */
8    int calc(int Var1};
9   }
10 FILE: Calc.Java
11   public int Calc(int Var1){
12
13   return (Var1 * Var1);
14   }
```

[1] file with added code

From the example seen in listing. 3.3 the preprocessor produces the decorated version of the file Calc.Java seen in listing. 3.4. This code ensures that the program will throw an exception and generate an error report if the contract is broken.

Listing 3.4: iContract example 2

```
1  FILE: Calc.Java
2  public int Calc(int Var1){
3  //#*#-----------------------------------------------------------------
4  boolean __pre_passed_1 = false; // true if pre-cond_1 passed.
5  //checking Calc(int Var1)
6  if(!__pre_passed_1) {
7    if( Var1 > 0 ) __pre_passed_1 = true; //Calc(int Var1)
8  }
9  if(!__pre_passed_1) {
10   throw new RuntimeException("Calc.Java:1:␣error:␣precondition␣"
11                              +"violated␣(Calc(int␣Var1)):␣"      +
12                              "(/*Calc(int␣Var1)*/␣(Var1␣>␣0))"
13   );}
14 int __return_value_holder_;
15 /*return (Var1 * Var1); */
16 __return_value_holder = (Var1 * Var1);
17 if(!(__return_value_holder_ < (Var1))) throw new RuntimeException("
     Calc.Java:1:error:postcondition" +"violated␣(Calc(int␣Var1)):␣"
       + "(/*return*/(Var1*Var1)␣␣<␣Var1))" );} //
     #*#-----------------------------------------------------------------
18 return __return_value_holder_;
19 }
```

The decorated file is only used in the testing phase and it is not visible for the programmer nor a part of the final product.

3.2.2 Korat

Korat is a tool for generating test cases for automated testing in Java. The tool uses specification based testing and generates test cases from the preconditions in the specification. The specification can be written in any formal language as long as it can be translated into Java predicates[2]. However the developers of Korat have only implemented the tool using the formal language JML which uses Java syntax and semantics. This enables the programmer to work with Korat without having to learn another programming language. Another advantage with JML is that the specifications can be written with the full expressiveness and power of Java.

Given a specification in a formal language Korat uses this to generate a predicate method (usually known as a repOk or checkRep method) based upon preconditions in the specification. This predicate is then used to generate a number of test cases within certain boundaries. Korat runs the method that is targeted for testing on each generated test case and verifies the correctness of the case compared to the postcondition in the specification. [17]

[2]i.e. a Java method that returns a boolean value

3.2.3 QuickCheck

QuickCheck [18] is an automatic specification based testing tool developed by Koen Claessen and John Hughes. The goal when developing this tool was to make it light-weight i.e. the tool is meant to support agile development technique [19]. Quickcheck is limited for Haskell programs and the final tool is only 300 lines of code.

A specification in QuickCheck is implemented as a predicate called property. The programmer defines one or more properties for each program part that is to be be tested. The program is then tested for a large number of randomly generated data inputs and the property is used to verify the program. If every condition in the property holds the test has passed.

QuickCheck uses random data generation but to ensure complete and thorough testing the programmer can limit the amount and type of data generated, making the test more limited and accurate. QuickCheck encourages unit testing by defining a property for each part of the program but it can also test complete program as a separate unit. The results of the test is normally reported simply as passed or failed but it also has the possibility to collect data to produce a histogram. Listing 3.5 shows a property and also how the call to QuickCheck is done. The property describes a rule defining that if two vectors are concatenated and then reversed the result should be the same as if the second vector was reversed and then concatenated with the reverse of vector 1.

Listing 3.5: QuickCheck property

```
1
2 prop_RevApp xs ys =
3         reverse (xs ++ ys) == reverse ys ++ reverse xs
4                 where types = (xs ::[Int], ys ::[Int])
5
6 Test> QuickCheck prop_RevApp
7 OK, passed 100 tests.
```

Example

Given two vetors: xs =[10,4,7,1,9] and ys =[5,2,3,6,8]

Each vector reversed separately
reverse(xs)=[9,1,7,4,10]
reverse(ys)=[8,6,3,2,5]

concatenating the vector reverse(ys) with reverse(xs)
reverse(ys) ++ reverse(xs) = [8,6,3,2,5] ++ [9,1,7,4,10] = **[8,6,3,2,5,9,1,7,4,10]**

concatenating the vector xs with ys, then reverse the result
reverse([10,4,7,1,9]++[5,2,3,6,8]) = **[8,6,3,2,5,9,1,7,4,10]**

3.3 Programming in Streambits

This section describes the different parts and the structure of StreamBits from the programmers point of view. StreamBits is designed for executing multiple tasks on multiple data streams concurrently, this is done by using the standard components pipelines and filters.

3.3.1 Components

There are three kinds of components from which a StreamBit program is built of: pipelines, filters and tapes(fig.3.1)

Filters is the component designed to perform the work on a given stream of input data. All filters are added to a pipeline which is the main part of a program. By adding filters in a different order the pipeline can be designed to perform many different tasks. Filters can be executed concurrently within a pipeline forming a multi threaded program.

Every connection inside the pipeline is done by tapes which are implemented as FIFO array blocking queues that stores the streams that comes out of each filter and passes on the resulting stream to the next filter. Each filter has two tapes, one config tape for the configuration and one data tape for the data stream. The tapes have two functions called *pop* to retrieve values from the input tape and *push* to add values to the output tape.

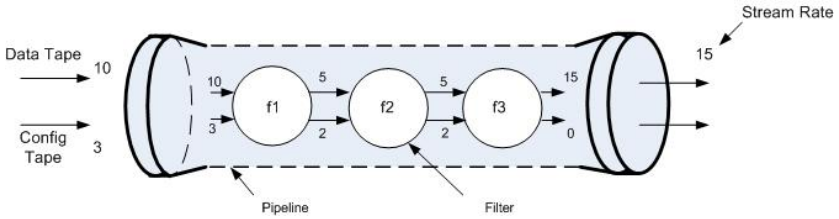

Figure 3.1: Pipeline

3.3.2 Data types

There are six data types implemented in StreamBits. Four of them are basic types that can be found in most existing programming languages.

1. intST - Integer in stream form
2. floatST - Float in stream form
3. byteST - Byte in stream form
4. voidST - Null type

The two remaining types are vecST and bitvecST. VecSt is an array type which can hold any one of the five other types. This type allows fine-grained data parallell operations to

be expressed within a filter [1]. BitvecST is implemented to improve the capabilities of bit level computations in StreamBits.

Unlike the standard data types found in other languages each of the data types in StreamBits implements an interface. These interfaces consists of a number of help functions related to the different data types, example there is an interface called LogicalType that contains the standard operations for logical operations, such as And, Or, Left shift and Right shift.

Table 3.1 and table 3.2[1] show how StreamBits has improved the bit level computations compared to C with functions using bitvecST and vecST. In each case it shows that StreamBits can perform complex tasks with a single function call. These functions is also machine independent and can be mapped onto any architecture, unlike the corresponding expressions in C.

Table 3.1: StreamBits compared with C

bitvecST oper.	Corresponding C expr.
bitslice(m : n)	(t & $w_{m:0}$)
bitsliceL(m : n)	(t & $w_{m:0}$) << (w - m)
bitsliceR(m : n)	(t & $w_{m:0}$) >> n
bitslicePack(m : n)	N/A

lmerge(k : l m : n)	if l <= (m - n):
	((t & $w_{k:l}$) << C_1) | ((s & $w_{m:n}$) >> n)
	if l>(m - n):
	((t & $w_{k:l}$) >> C_2)|((s&$w_{m:n}$) >>n)

Table 3.2: Operator comparison vecST

StreamBits oper.	Corresponding C expr.
vecslice(m : n)	for i = 0 to 4{t[e_i] & $w_{m:n}$}
vecsliceL(m : n)	for i = 0 to 4{t[e_i] & $w_{m:n}$ << $(w-m)$}
vecsliceR(m : n)	for i = 0 to 4{t[e_i] & $w_{m:n}$ >> n}
lmerge(k : l,m : n)	for i = 0 to 4{ if l <= (m - n) (t[e_i]c & $w_{k:l}$) << C_1 | (s[e_i] & $w_{m:0}$) >> n if l > (m - n) t[e_i] & $w_{k:l}$) >> C_2 | (s[e_i] & $w_{m:0}$) >> n }

3.3.3 Init- Configure- Work

Each filter can work in three different modes: init, config and work. The init mode is used for initiation of the filter. The config is used for reconfiguration of the filter during runtime. The work mode is the part of the filter that process the data stream. This is done by reading the data stream from the in tape and perform different operations on the data before passing it on to the next filter.

Fire is a command used to execute the pipeline. Once the fire command has been given it will keep all filters working by calling each filters Configure and Work mode continuously in that order. During one configure session the filter reads the config stream and sets

different variables inside the filter which describes how the filter should behave in the next work session.

The fact that configure is called continuously means that the entire pipeline can be reconfigured during runtime. Listing 3.6 shows a filter which purpose is dependent on the config stream. The variable mode determines if the filter should add or subtract the incoming values, since Configure is called continuously and before Work, mode can toggle from one work session to another. This means that the current configuration will be applied to each value that is popped from the data stream during the following work mode.

Listing 3.6: Reconfiguration during runtime

```
1  public class FilterExample extends Filter<intST, intST, intST, intST
       >{
2
3  int mode;
4
5  public FilterExample() {
6          super.setDataRate(new intST(2),new intST(1));
7          super.setConfRate(new intST(1),new intST(0));
8      }
9
10 public void init(){
11    mode=1;
12  }
13
14 public void work(){
15
16    if(mode==1)
17    {
18        pushD(popD().getVal+popD.getVal);
19    }
20    else
21    {
22        pushD(popD().getVal-popD.getVal);
23    }
24  }
25
26 public void configure(){
27
28     /* set mode according to config tape during runtime*/
29     mode=popC().getVal;
30  }
31
32 }
```

3.3.4 Main function

Main creates a new StreamProgram (listing 3.7) which makes a function call to streamProgram (Note the difference, no capital s) where the programmer builds up the pipeline. Once the function streamProgram terminates and returns to StreamProgram the pipe initiates each of the added components and fires the pipeline automatically.

Listing 3.7: StreamProgram

```
1  public StreamProgram(){
2
3                  /* Build pipeline structure */
4                  streamProgram();
5
6                  /* Initiate and start all components */
7                  for(int i = 0; i < nFirings; i++){
8                          ErrorHandler.setRunning(true);
9                          initComponent();
10
11                         ErrorHandler.setRunning(true);
12                         fire();
13                 }
14         }
```

3.3.5 Threaded Framework and streamrate

The entire framework is built for operating on streams of data. This structure is especially used when performing tasks in parallel. Every filter in StreamBits can be view as a thread that is executed the first time the fire command is given. Several threads can be running at the same time even though some of the threads are dependent on other threads. This dependency can sometimes cause the pipeline to freeze i.e. every filter is waiting for data from the previous filter.

The stream rate describes the number of *pops* and *pushes* made on each tape during one work session. Example 3.8 shows a work session with defined stream rate. The specified stream rate can be seen om line 2, in this case it is 2:1 meaning that each time the work mode is executed two values will be *popped* from the incoming data tape and one value will be *pushed* on the outgoing data tape. It is up to the programmer to maintain the stream rate. This is done by making sure that the filter in work mode always performs the defined number of push and pops regardless of the configure mode and conditional push and pops. Too many push can result in faulty processing of the data streams. Too few push/pops or too many pops results in a severe stream rate error [3].

Listing 3.8: Streamrate example

```
1
2 super.setDataRate(new intST(2),new intST(1));
3 {
4                     intST value1 = popD();
5
6                     intST value2 = popD();
7
8                     pushD(value1 - value2);
9 }
```

[3] i.e stream rate errors causing the component to stall

4 Implementation

This chapter describes the details of the test program created in for this master thesis. The program is a specification based testing tool with specifications written as Java *predicates*, similar to QuickCheck. The program is implemented as a package in the StreamBits framework written by Jerker Bengtsson[1] and Jonathan Andersson.

The tool developed in this project is implemented as a test framework that simulates runtime. This means that the testing of a program is done before the program is moved to the actual application, and the specific parts used for the testing tool will not affect the performance during runtime.

The program can be divided into two parts: the test program which is hidden from the programmer and the Main and Property which works as an interface toward the programmer. Figure 4.1 shows an overview of the complete test program and the programmer interface. The left side of the figure is the interface visible to the programmer and the right part is the package containing the test program.

The visible part of the test is built up by the test components, a property class, the main program and the test log created by the test program. The test components are the filters and pipelines targeted for testing. For each test the programmer also writes a property class containing a predicate. This is the specification for which the test is performed. The test is then initiated by the programmers main method. This is done by calling the test program´s main part, Autotest.

The test program which is the hidden part consists of the class Autotest and a package of standard data generators. Autotest is the main program which builds up and runs the test; the result is then reported in the log file. The data generators are used for generating suitable test data; the programmer can choose between random test data, data in certain intervals and test data with unit step. The programmer can also define new datagenerators.

4.1 The test tool

In this section the test tool is explained in detail. This part of the program is made up of the tool for unit testing, an interface for generating test data and a part for validating the stream rate of the test components. These three parts are tied together in the class Autotest which handles the structure of the test and also the communication with respect to the programmer. Unit testing is carried out by a part called datacollector, the stream rate testing is implemented in the framework and handled by Autotest and the final part, test data generation is made up of an interface which handles 14 different kinds of data generators.

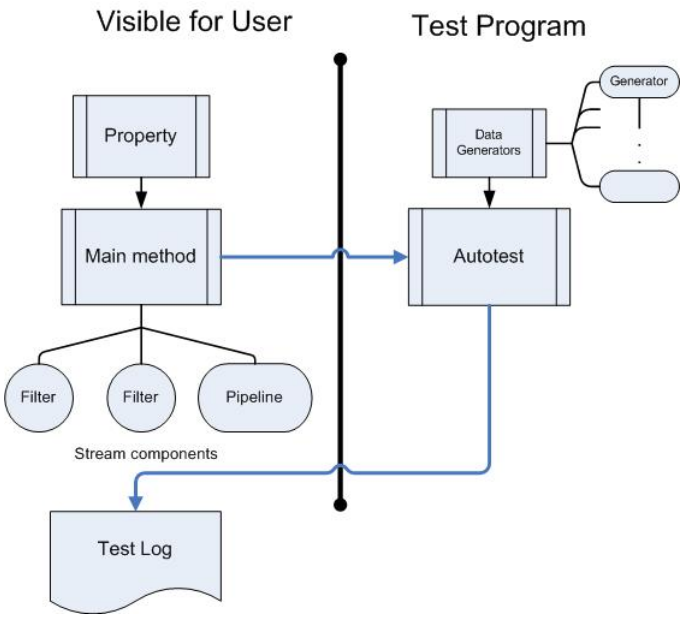

Figure 4.1: Test program overview

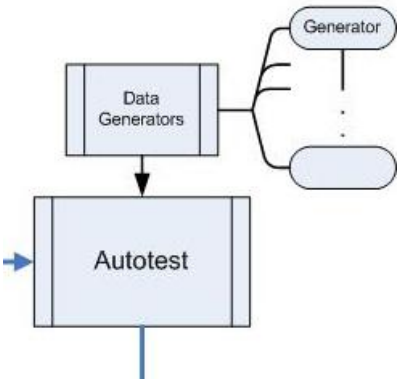

Figure 4.2: Hidden part of test tool

4.1.1 Autotest

This is the main part of the tool where the test is initiated and performed (fig. 4.3). The program starts by creating a pipeline that is used only for testing. To this pipeline the tool adds a test data generator and a config filter. There are generators for random data, intervals, steps and ramps available. Which generator to use in each test is specified by the programmer in the class test. The config filter provides the test pipe with a configuration stream which is provided by the programmer in test. When a suitable configuration stream and test data stream exists, the program adds the different components to be tested. At the end two filters called the data collector and the LogFilt is added. The data collector is the part of the program performing unit testing. This is done by calling the defined property class which is the specification written for the test. The LogFilt handles the log file that is printed after each test. Each part of the test program is instantiated using generics so the program can handle any data type available.

The complete test pipeline is then initiated and started by the commands initComponent and fire. After the completion of the test Autotest produces a test report in a txt file called log.

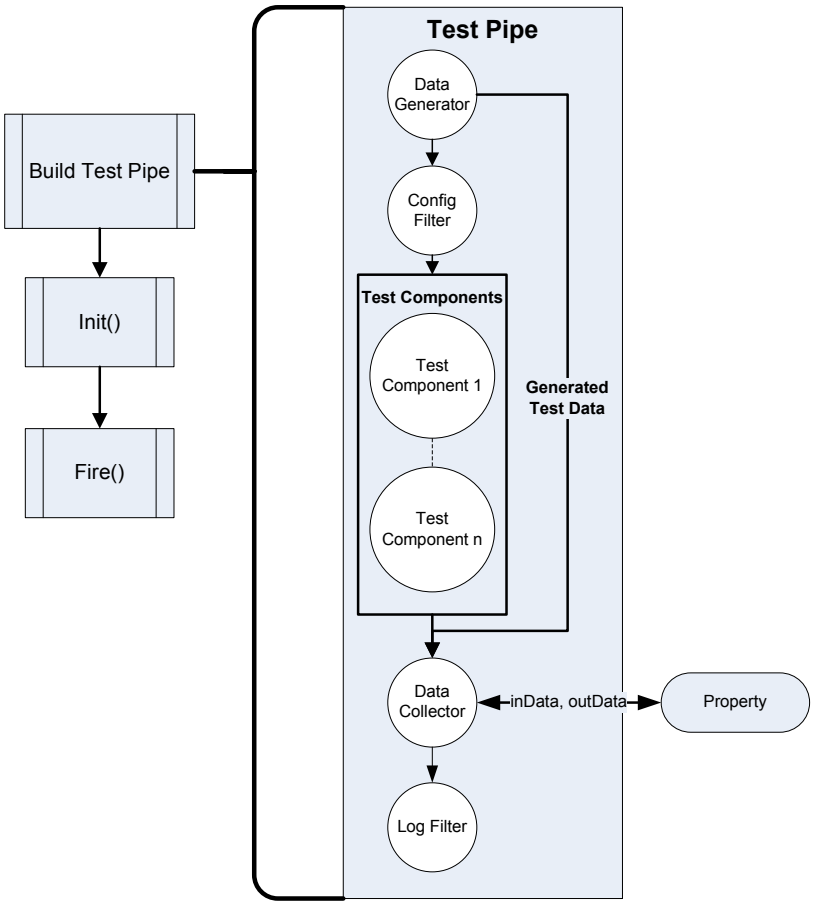

Figure 4.3: Flowgraph for Autotest

4.1.2 Data generators

The purpose of the data generators it is to generate suitable test data for the unit test. There are three types of generators, random and interval and step. The different generators take different arguments needed to generate the data according to the programmers specification. These arguments are the interval which is defined from *LowBound* to *UpperBound*. The *step* argument describes the step of an increasing or decreasing data sequence. Some generators also need an argument that defines a length. The random generators produce random test data within an interval given by the arguments passed along to the data generator. The interval generators generate test data as a ramp which begins with LowBound and make steps according to the *Step* variable. The step generators produce a unit step from LowBound to UpperBound. The step can be delayed a number of elements using the delay argument.

StreamBits has 5 different data types and each of this data types has every type of the generator that could be useful. There is also implemented an voidST generator for testing

programs which requires no data input.

Each generator produces a stream of test data according to the generator´s out rate. In the building and initiation of the test pipeline in Autotest the generators out rate is matched with respect to the first test components inrate. In this way the generators always generates test streams of proper length. The available data generators are shown in table 4.1

Table 4.1: Availible data generators

	intST	floatST	vecST	byteST	bitvecST	voidST
Random	•	•	•	•	•	
Ramp	•	•	•	•		
Step	•	•	•	•	•	

The different possible arguments required for each generator can be seen in table 4.2. To define which type of elements used in vecST the generator requires a string representation of that type and vecST is also initiated with one generic.

Table 4.2: Arguments for each generator

	intST	floatST	vecST	byteST	bitvecST
Random	L,H	L,H	L,H,Len,T	H,L	Len
Ramp	L,S	L,S	L,S,Len,T	L,S	n/a
Step	L,H,D	L,H,D	L,H,Len,T,D	L,H,D	Len,D

L = Lower boundary limit, H = Upper boundary limit, S = Step, Len = Length T = Type D = Delay

The interface that the data generator is built upon also allows the programmer to define specific data generators in a simple way. Each data generator is implemented as a filter which should provide the test filters with data during work mode. All generators share the variables and functions shown in listing 4.1. The only difference between the generators is the constructor and the init sequence. This enables the user to write a specific generator implementing this interface. The programmer only writes the init which generates the test data and a constructor for defining input variables. An example of a specific generator used in a matrix multiplication test (5.2.2) can be seen in appendix C.

Listing 4.1: Source class

```
1
2  public abstract class Source <Exp1,Exp2,Exp3,Exp4> extends Filter<Exp1,
       Exp2,Exp3,Exp4>{
3
4      public Source(){
5      }
6
7      /* Define the tapes in the correct way */
8      /* Must be called by the programmer in the beginning of init */
9      public void DefTapes(){
10     ...
11     }
12
13     /* Set variables to generate correct amount and type of data */
14     public void set_test(Test t){
```

```
15      ...
16      }
17
18      /* Funtion to handle timing */
19      public void setWait(boolean temp){
20      ...
21      }
22
23      /* No configure needed, the generator is the first filter */
24      public void configure(){
25
26      }
27
28      /* Function that returns the generated data */
29      public Tape[] RetTape(){
30      ...
31      }
32
33      /* Pushes the generated data */
34      public void work(){
35      ...
36
37      }
38  }
```

4.1.3 Config

The purpose of this filter is to set the input configuration that the programmer has defined. The configuration stream is loaded into a buffer in initiation mode and then pushed on the out config tape in work mode. Since this filter is added after the data generator the data stream from the input has to pass through this filter without any modifications this is done by pop from the in tape and push to data out tape in work mode.

The configuration used is specified by the programmer in the class test sent to Autotest. In this class there is an array with generic tapes, the programmer defines one tape for each test performed.

4.1.4 Data collector

Data collector is a filter added at the end of the test pipeline. This is the part performing the unit test. Before calling the property a complete set of resulting data must be obtained. In some applications (ex matrix multiplication 5.2) the test components are executed several times before a complete test result is obtained, therefore the iteration variable in test class exists. The iteration variable is set by the programmer and it defines how many times the last test component will be executed to obtain a complete result. The data collector waits until it has enough elements before it proceeds with unit testing.

The data collector also handles severe stream rate errors. In the data collector a array of timers is implemented, one for each test to be performed. Java 1.5 can not handle multiple timers, therefore two classes developed by David Flanagan [20] are used. These classes

which are developed to solve this specific problem are Timer and TimerTask (Appendix F) which implements the corresponding classes in Java 1.5.

In the beginning of each test the corresponding timer is executed, the test components then have a limited time pushing enough values to the data collector. The deadline for timeout is set by the programmer, depending on what she knows about expected execution times of her pipeline.

If the data collector does not receive enough data a timeout occurs. When this happens the test is aborted and the currently obtained data and configuration are sent to a method in the log filter which handles timeout errors and produces a log.

If a timeout error does not occur unit testing is performed. This is done by calling the property method (4.2.2) with the generated test stream and the, from the test pipeline, resulting data stream. The property method returns a boolean value to represent the outcome of the test.

4.1.5 The log filter

The log filter is the last part of the test pipeline. The purpose of this filter is to produce a test log which is done by collecting data from the data collector, generator and configuration filter. Results from the property test are obtained from the datacollector and the resulting data and config stream are given on the in tapes to logFilt. The generators provide the current input data stream for the test and the config filter provides the configuration input stream.

The log filter produces a report as a txt file called log followed by the date and time it was created. This report contains results of the unit test and also detailed results for the stream rate test for each component.

To simplify debugging a class called LogPrint is available to the programmer, which can be used to print error messages directly to the log file. To make the program user friendly the class LogPrint is implemented to mimic the System.out.print* class in Java and uses the same syntax with print() and println().

4.1.6 Stream rate test

The stream rate test is one of the major parts of this thesis. Every filter and pipeline has a defined stream rate for the data and the config tape. It is up to the programmer to maintain these rates by making the right number of push and pop from the tapes. This can be a problem when the filter contains several conditional pushes and pops. Since the tapes are implemented as array blocking queues they make the StreamBit program stall if a filter is trying to push to a queue that is full, or pop from an empty queue. To solve this problem there are counters implemented in every tape in the framework, counting the number of pushes and pops done during runtime.

The stream rates are defined but as StreamBits is multi threaded a component can be executed several times during one tests. The iteration variable in test class is used to calculate the stream rates for each component during one complete test. After completion of the test the log filter reads the counters and report the results to the log file.

4.2 Test program from programmers point of view

This section will cover the functionality of the test program from the programmers point of view (fig. 4.2). The goal has always been to produce a user friendly tool needing a minimum background information to use it.

There are three main parts visible for the programmer. It is the specification which is implemented as a Java predicate, a test class whose only purpose is to hold the different variables needed for the test and a main method. The benefit from implementing the specification as a predicate is that it can be written in the same language as the part of the program that is tested. This makes the program much more user-friendly by eliminating the need for another formal language.

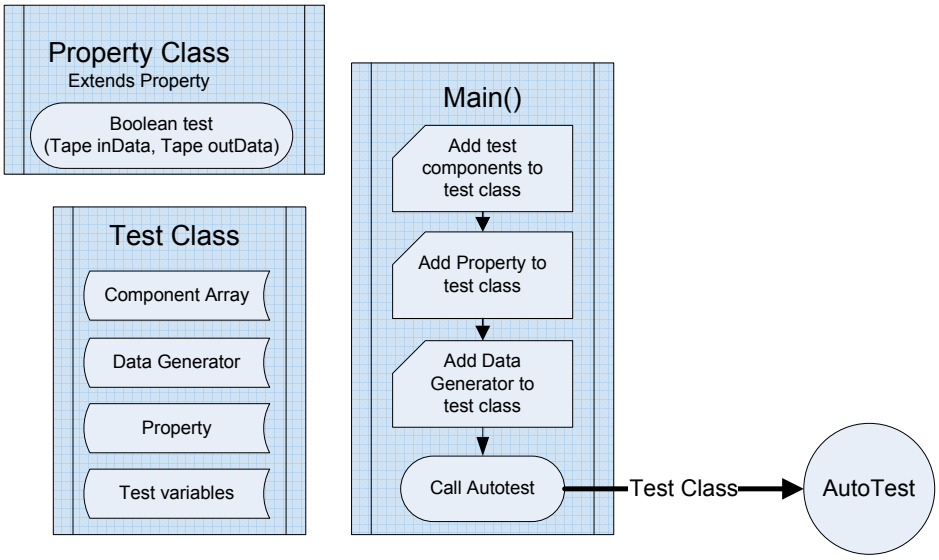

Figure 4.4: Visible part of test tool

4.2.1 Main

The initiation of the test from the programmer point of view is done in the main file. This is done by initiating a new Autotest object and a new test object. The constructor for Autotest requires a test class as parameter which holds the different variables defining the test (4.2.3). The programmer initiates this test class and configures the test by setting the different variables before initiating Autotest.

4.2.2 Property

Property is an interface (listing 4.2) containing a boolean function called test, which requires two tapes as arguments. These tapes consist of the output from the data generator (i.e. the input for the test pipeline), and the output from the entire test pipeline. In the interface the types of the tapes are implemented as generics[21] which allows the programmer to define the type in the property class declaration.

Listing 4.2: Property Interface

```
1 public interface Property <Exp1,Exp2>
2 {
3         public boolean test(Tape<Exp1> indata,Tape<Exp2> outdata);
4 }
```

The test method (listing 4.2) is the specification used in the test. Test is written as a Java predicate where the programmer writes code using the in- and out-data to verify that the test components behave as expected. In the property the programmer can also write a error code using LogPrint.

Listing 4.3: Example: programmers test method

```
1 public class userProperty implements Property<intST,intST>{
2
3     public boolean test(Tape <intST> indata,Tape <intST> outdata) {
4
5         intST temp = indata.pop();
6
7         return(temp == 5);
8
9     }
```

4.2.3 Class Test

The test class is intended to provide a clear view of the test parameters required in a test. By defining each parameter in a viewable test class the programmer can easily get an overview and set the required variables. The different parameters in the test class can be seen in fig. 4.5.

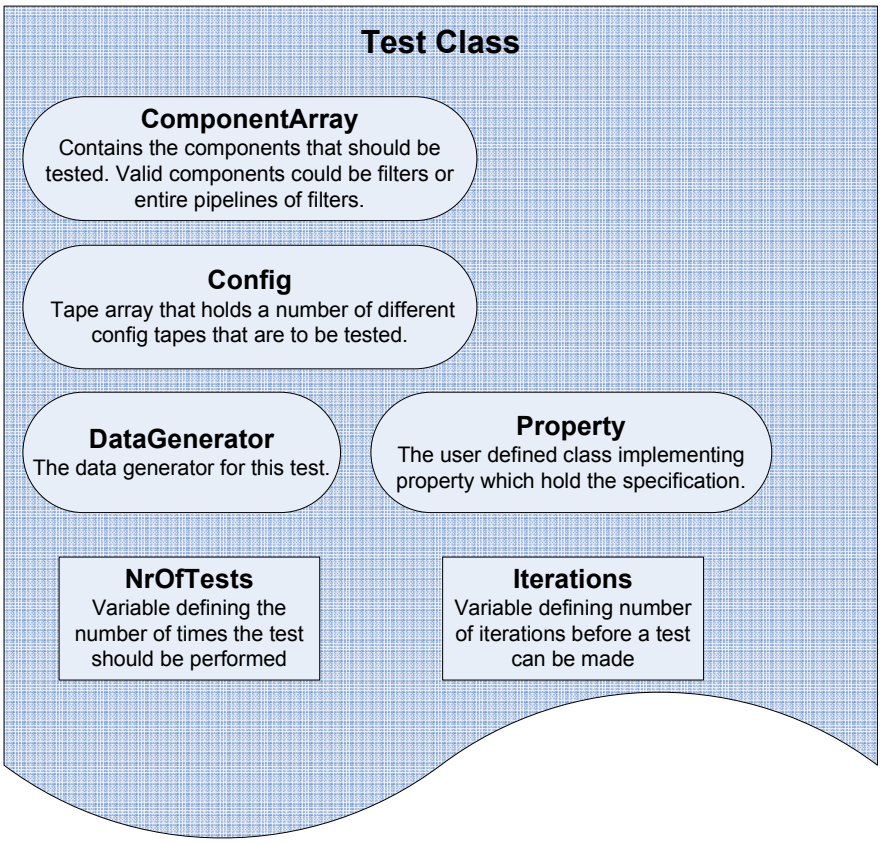

Figure 4.5: The test class

The ComponentArray holds the filters and pipelines targeted for testing. If a multi threaded network is used then the multi threaded part has to be put into a pipeline for the test tool to be able to handle the component and perform stream rate test. Configuration is an array of generic tapes, one tape for each test. The programmer puts the requested configuration stream here and the configuration filter will provide the config stream to the test components. DataGenerator is the generator defined by the programmer to use for the test. Property is the specification to be tested. NrOfTests is the number of times the specification should be tested. This is used to simulate a StreamBits program in runtime executed over and over again. Iteration is the iterations required for the test components to produce a complete set of resulting data. The final variable is timeout which sets the timeout limit before the test should abort due to pipeline stall. This variable is set to 10 seconds by default but it is supposed to be set by the preogrammer depending on her knowledge of the proogram being tested.

4.3 Execution of test tool

This section is intended to give a overview of the complete test, it will describe the work flow and communication of different parts of the test tool during a testrun.

Autotest handles the initiation of the test, this is done by building a test pipeline and by sending the required variables to each part.

The test pipeline (fig. 4.6) always begins with a data generator. Each iteration the data generator sends out a set of generated data enough for one test. After its completion a wait variable in the data generator is set true causing the generator to hold until the current test's completeness. The data is sent to the config filter which push the current tests configuration tape onto the tape into the test components.

The data collector collects the output data and configuration streams from the test objects. A timer is used to handle severe stream rate errors. In case of a complete set of data received, the data collector calls the generator to obtain a copy of the indata used for the current test. Data collector then performs the specification test by calling the property with a copy of the indata and the received outdata. This must be done before pushing the data forward to the log filter as the log would be produced before the property has finished otherwise.

The log filter receives the resulting data and configuration stream from the data collector. It calls the generator to obtain a copy of the indata and the config filter to get the current config stream. The stream rate counters are read for each test component in the pipeline and the result is reported to the log. The stream rate counters are reset and the generator is called setting the flag wait to false which triggers the next test.

This ”stop and wait” technique in the generator prevents the test objects from continuing to work on the next test thus continuing counting the push and pop resulting in faulty stream rate error reports.

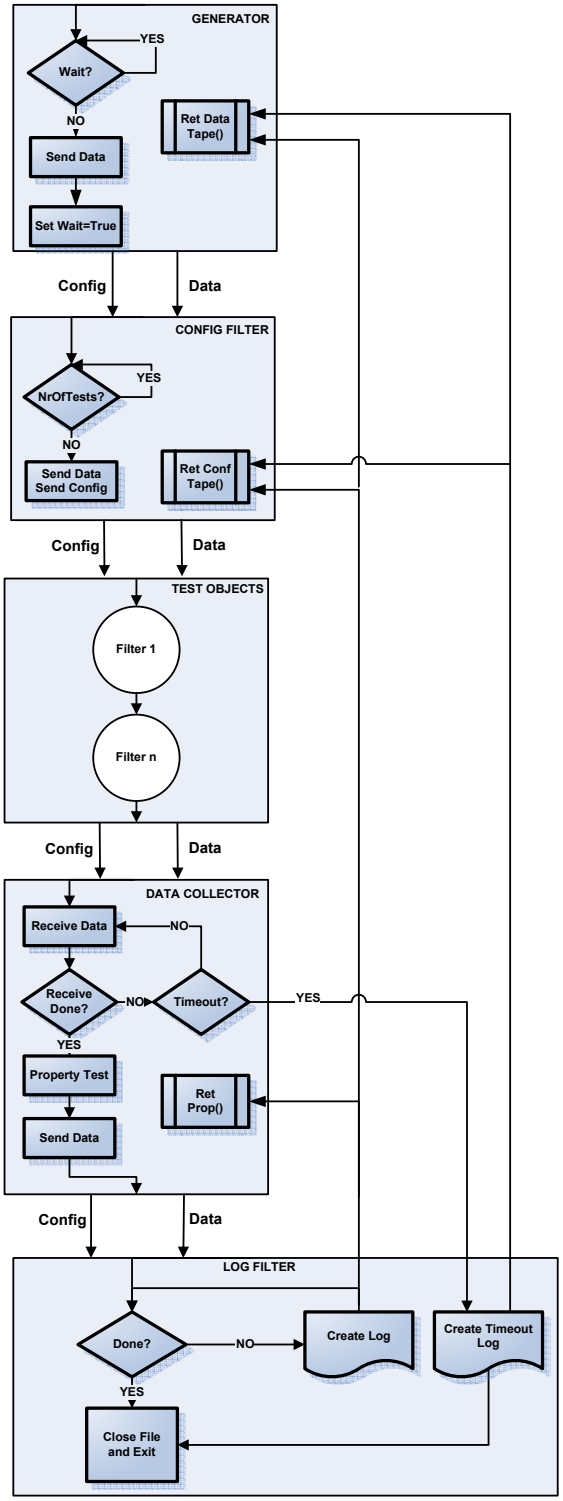

Figure 4.6: Execution of the test tool

5 Results

This chapter demonstrates the complete test tool created in this thesis. Two different tests on different components are shown. The first test is a filter network which adds incoming elements together. The second test is a multi threaded matrix multiplication pipeline. For each test the programmer defines the test in a main class and writes a specification as a property.

5.1 Adder

This is a test of a simple streamprogram which adds incoming values together. This program is made up by three connected adder filters (fig. 5.1). The test intends to show how the test tool handles multiple components. Streamrate is tested for every component and a unit test is performed for the entire network.

5.1.1 Functionality

Each filter is designed to *pop* two values from the input tape and add the values together and then push the result. The filter does not use any configuration.

5.1.2 Test

The test is defined in main and a specification for the unit test is written as a property.

In the main class (listing 5.1) the test is defined and initiated. The programmer defines a new class of type test called TestClass which holds all the variables required for the test.

In TestClass the filters to be tested are added to a streamcomponent array and streamrates for each component is set. The first adder will take eight intST elements and the last adder will produce one resulting intST element. A data generator is specified, in this case the generator is an intSTrand which produces random intST. As an argument to the generator the value of the produced elements is set between 0 and 1000. Finally the test program is called with the given test parameters.

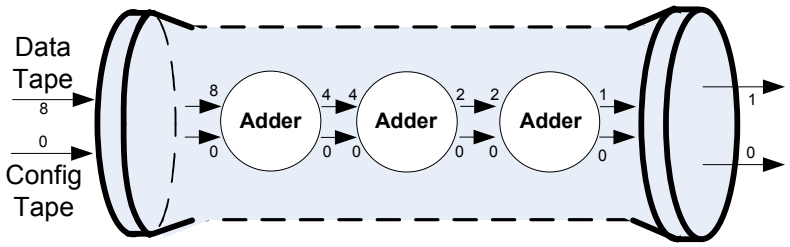

Figure 5.1: Adder pipeline

Listing 5.1: Main program for adder

```
1  /*New Class test is created*/
2  Test TestClass = new Test();
3
4  /* Component Array containing components targeted for testing */
5  TestClass.ComponentArray = new StreamComponent[3];
6  TestClass.ComponentArray[0]= new Adder();
7  TestClass.ComponentArray[1]= new Adder();
8  TestClass.ComponentArray[2]= new Adder();
9
10 /* Set stream rates for each component */
11 TestClass.ComponentArray[0].setDataRate(new intST(8),new intST(4));
12 TestClass.ComponentArray[1].setDataRate(new intST(4),new intST(2));
13 TestClass.ComponentArray[2].setDataRate(new intST(2),new intST(1));
14
15 /* Set desired data generator */
16 TestClass.DataGenerator = new intSTrand(0,1000);
17
18 /* Initiate test */
19 AutoTest a = new AutoTest(TestClass);
```

5.1.2.1 Property

To test the filter a simple property (listing 5.2) is used. The property reads every element on the provided tape indata and sum these together. The sum is then controlled to be the same as the single value on out data. If this is the case the property will return true indicating that unit test has passed.

Listing 5.2: Adder property

```
1  public class AdderProperty implements Property<intST,intST>{
2
3     public boolean test(Tape<intST> indata,Tape<intST> outdata){
4
5           boolean passed=true;
6
7           outlength=outdata.length();
8           inlength=indata.length();
9           int sum=0;
10
11          for(int i = 0; i < outlength; i++)
12          {
13                          sum=0;
14              /*Add two values from the intape*/
15              sum  = indata.pop().getVal();
16              sum += indata.pop().getVal();
17
18              /*Check that the sum matches the output */
19              if(sum!=outdata.pop())
20              {
21                  passed=false;
22              }
23          }
24          return passed;
25      }
26 }
```

5.1.3 Results

In this section two tests are demonstrated. The first test intends to demonstrate the log file. The second test demonstrates a programming error causing property to fail and the way of locating the error. To perform a thorough test a program should also be runned several times simulating the runtime work of a streamprogram. The result of such test can be found in appendix A.2.

5.1.3.1 Testrun 1

Upon completion of the test a log file is created (listing. 5.3). The log always begin with a line stating the date and time it was created. The next part is the streamrate report for each component. The *inRate* and *outRate* describes the configured input/output streamrate, *Pop* and *Push* describe the actual streamrate. For the streamrate test to pass, the actual *push/pop* must match the requested *push/pop*.

Listing 5.3: Adder testrun 1, log part 1

```
1  Log created : Sat Dec 09 16:07:15 CET 2006
2
3   Test Nr: 1 / 1
4  ********************************************************************
5  Testing data streamrate   , component: streambits.userprogram.
       Adder@122cdb6
6  Data inRate =    8 ,Data outRate=    4, nr pop done =8 ,nr push done =4
7  Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
8  Streamrate test Passed
9
10 Testing data streamrate   , component: streambits.userprogram.
       Adder@1ef9157
11 Data inRate =    4 ,Data outRate=    2, nr pop done =4 ,nr push done =2
12 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
13 Streamrate test Passed
14
15 Testing data streamrate   , component: streambits.userprogram.
       Adder@12f0999
16 Data inRate =    2 ,Data outRate=    1, nr pop done =2 ,nr push done =1
17 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
18 Streamrate test Passed
```

After all streamrate tests has completed the property described in 5.1.2.1 is tested. The entire test result is also displayed. If any streamrate or the property test fails, the entire test fails.

Listing 5.4: Adder testrun 1, log part 2

```
1 Testing Property: streambits.userprogram.AdderProp@a31e1b
2
3 Property test passed
4
5 ***************
6 * Test PASSED *
7 ***************
```

For debugging purposes all the generated in data and the resulting outdata is printed together with configuration if such is used (listing 5.5).

Listing 5.5: Adder testrun 1, log part 3

```
1
2  Data input: 60  62  0  36  22  37  17  23
3
4  Data output: 257
5
6 ************************************************************************
```

5.1.3.2 Testrun 2

The second test simulates a property test failure and the way of locating the error. The test fails and the testlog indicates a property test fail(listing 5.6).

Listing 5.6: Adder testrun 2, log 1

```
1 Testing Property: streambits.userprogram.AdderProp@16a55fa
2
3 Property test failed
4
5 ***************
6 * Test FAILED *
7 ***************
8
9  Data input: 93  14  2  67  94  83  77  90
10
11  Data output: 744
```

The data input and output are printed in the log but the error is not obvious. The programmer choose to test only one component at a time with more values and also adds a LogPrint function in the property printing the correct sum for each element(listing 5.7).

Listing 5.7: Updated property adder test

```
1 ...
2 int result = outdata.pop().getVal();
3
4 LogPrint.LogPrintln("Calculated result: " + result  + " correct
    result " + sum);
5
6 /*Check that the sum matches the output */
7 if(sum!=result)
8 ...
```

In the resulting log the error is more clear. With more resulting elements it clearly shows that the error occurs every time and with more thorough examination of the output values they appear to be two times the first input value given.

Listing 5.8: Adder testrun 2, log 2

```
1  ...
2  Testing Property: streambits.userprogram.AdderProp@13e205f
3
4  Property test failed
5
6  User Log:
7
8  Calculated result: 104 correct result 86
9  Calculated result: 144 correct result 107
10
11 ...
12 Calculated result: 52 correct result 77
13 Calculated result: 52 correct result 97
14
15 ***************
16 * Test FAILED *
17 ***************
18
19  Data input: 52  34  72  35  ...  85  28  81  87  26  51  26  71
20
21  Data output: 104  144  ...  170  162  52  52
22
23 ********************************************************************
24 ...
```

The error in this case occured when calculating the sum in the adder filter (listing 5.9) where a variable d was sent instead of variable e. The complete log can be seen in appendix A.1.

Listing 5.9: Error in filter adder

```
1 ...
2 intST d = popD();
3
4 intST e = popD();
5
6 intST sum = d.add(d);
7 ..
```

5.2 Matrix multiplication

This is a test of a pipeline which performs multiplication between two matrices A and B of dimension $n \times n$. The pipeline receives two vectors of length n and produces n number of resulting intST elements out.

5.2.1 Functionality

The pipeline (fig. 5.2]) designed to perform the multiplication is built up by two switches and several threads, each thread contains one VecMul and one VecAdd filter. The number of threads in the pipeline is determined by the dimension of the matrixes.

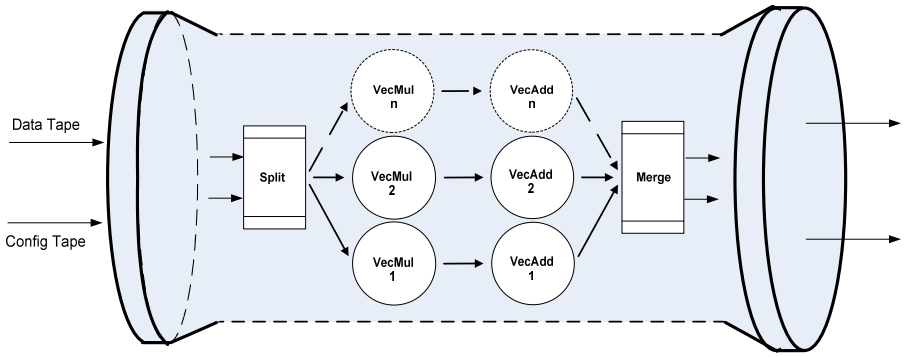

Figure 5.2: Matrix Multiplication pipeline

A matrix multiplication is defined as $C_{ij} = \sum_{k=1}^{n} A_{ik} * B_{kj}$ (fig. 5.3 [22]).

If A is a matrix with dimension m-by-n and B a matrix with dimension n-by-k , the resulting matrix will be of dimension m-by-k.

if

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots \\ a_{2,1} & a_{2,2} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} b_{1,1} & b_{1,2} & \dots \\ b_{2,1} & b_{2,2} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

then

$$\mathbf{AB} = \begin{bmatrix} a_{1,1}\begin{bmatrix} b_{1,1} & b_{1,2} & \dots \end{bmatrix} + a_{1,2}\begin{bmatrix} b_{2,1} & b_{2,2} & \dots \end{bmatrix} + \cdots \\ a_{2,1}\begin{bmatrix} b_{1,1} & b_{1,2} & \dots \end{bmatrix} + a_{2,2}\begin{bmatrix} b_{2,1} & b_{2,2} & \dots \end{bmatrix} + \cdots \\ \vdots \end{bmatrix}$$

Figure 5.3: Matrix Multiplication

In the matrix multiplication pipeline this operation is done with a split which distributes the different vectors from the intape. The split filter gives each thread it´s own row from the first matrix and all the columns from the second matrix. A configuration element is used to separate the static row from matrix A from the columns provided from matrix

B. Configuration 0001 means a row from matrix A and config 0000 means a column from matrix B.

Once each thread have recieved one row and one column, the VecMul filter performs multiplication between each element in the vectors and passes the result onto VecAdd where the products are added together. In the end of the pipeline a merge binds the different threads together and produces a single tape out. Each iteration of the pipeline will produce n elements in the final matrix, i.e. it will take n iterations to perform a $n \times n$ matrix multiplication.

5.2.2 Test

To test the matrix multiplication two different types of tests are performed. The tests are based on two simple properties for matrix multiplication, $A \times 0 = 0$ and $A \times I = A$.

The programmer defines the tests in the main class (listing. 5.10). There are three arguments given: the dimension of the matrices, the number of iterations required to get a complete test result and the nr of tests to be performed.

An instance of class test is created and the pipeline is loaded as the only element in the ComponentArray in TestClass. The streamrate for the component is set to $2 \rightarrow n$ both for data and config.

To produce a suitable configuration tape the programmer adds for each test n times the configuration 0001 and n times the configuration 0000, to the array with config tapes in TestClass. The property(5.2.2.1) written by the programmer is also included in TestClass. The matrix multiplication is tested with different tests using two different properties (5.2.2.1).

To produce a complete result the pipeline has to be iterated a number of times equal to the dimension of the matrices, therefore the iteration variable is set to n.

The two tests performed uses different data generators. The first test, $A \times 0 = 0$ uses a vecSTstep generator which first produces n number of vecST for the A matrix followed by n number of vecST containing zero for the second 0 matrix.

The second test $A \times I = A$ uses a special generator (appendix C) written by the programmer, based on the vecSTrand generator. The special generator first produces n number of random vecST for the A matrix followed by n number of specific vecST for the I matrix.

To initiate the test the class Autotest is called with TestClass as an argument.

Listing 5.10: Defining test matrix multiplication in main

```
1  public class Main extends StreamProgram<voidST,voidST,voidST,voidST>{
2
3   public void streamProgram(String args[]){
4
5      /*New Class test is created*/
6      Test TestClass = new Test();
7
8      /*set dimension */
9
10     int dim  = args[0].toInteger();
11     /* Component Array containing components targeted for testing */
```

```
12      TestClass.ComponentArray = new StreamComponent[1];
13
14      TestClass.ComponentArray[1]= matrixMulTest(dim);
15
16      /* Set stream rates for each component */
17      TestClass.ComponentArray[0].setDataRate(new intST(2),new intST(
            dim));
18      TestClass.ComponentArray[0].setConfRate(new intST(2),new intST(
            dim));
19
20      /* Set data generator:
21      vecSTrand(int vecLength, int lowbound, int highbound);*/
22      TestClass.DataGenerator = new vecSTrand(dim,0,10);
23
24      /* Set Property written by programmer */
25      TestClass.property = new MatrixMult();
26
27      /* Set number of iterations required to obtain a complete test
            result */
28      TestClass.iterations = args[1].toInteger();
29
30      /* Define number of tests */
31      TestClass.NrOfTests = args[2].toInteger();
32
33      /* Initiate test */
34      AutoTest a = new AutoTest(TestClass);
35
36  }
37  public Pipeline matrixMulTest(int Dimension){
38  ...
39  }
40 }
```

5.2.2.1 Property

To verify the result using the test tool, a property is written by the programmer. To test the matrix multiplication two tests are performed using two different properties.

The first property (listing 5.11) tests that $A \times 0 = 0$. This done by testing that every output value is zero.

Listing 5.11: Property 1 for matrix multiplication

```
1  public class MatrixAx0 implements Property<vecST,intST>{
2
3      public boolean test(Tape<vecST> indata,Tape<intST> outdata){
4
5          boolean passed=true;
6
7          /*Delete Dummy data */
8          ...
9
10         int outlength=outdata.length - dim;
11
12         LogPrint.LogPrintln("Testing␣property␣A␣x␣0␣=␣0");
13
```

```
14          //check property, output = 0
15          for(int i = 0; i < outlength; i++)
16          {
17              if(outdata.pop().getVal() != 0)
18              {
19                  passed=false;
20              }
21          }
22
23          return passed;
24      }
25  }
```

The second property (listing 5.12) tests that $A \times I = A$. This is done by comparing the resulting output with the input generated by the datagenerator.

Listing 5.12: Property 2 for matrix multiplication

```
1  public class Unit_MatrixProp implements Property<vecST,intST>{
2
3      public boolean test(Tape<vecST> indata,Tape<intST> outdata){
4
5          boolean passed=true;
6
7          int outlength=outdata.length;
8          int dimension=(int)Math.sqrt(outlength);
9
10           /*Transform vecST indata to intST to be able to compare to
                 outdata*/
11         ...
12
13         /*check property*/
14         for(int i = 0; i < outlength; i++){
15             if(intST_DataIn[i] != outdata.pop())
16             {
17                 passed=false;
18             }
19         }
20         return passed;
21     }
22 }
```

5.2.3 Results

This section describes two different tests on matrix multiplication. Both these test uses matrices of dimension four. The complete logs can be found in appendix D. Additional test logs for other dimensions can be found in appendixE.

5.2.3.1 Testrun 1

The first testrun is done using the first property testing $A \times 0 = 0$. The dimension is set to 4 , and number of tests to 1. The test passes and the result is reported in both the compiler output and in the log file.

The streamrate part of the log file shows the resulting streamrates for the matrix multiplication pipeline. The streamrates for one test is 8 elements in and 16 elements out on both the data and config tape. This is verified by the test program and reported in the test log (listing 5.13).

Listing 5.13: Matrix testrun 1, log part 1

```
1  Test Nr: 1 / 1
2  ************************************************************************
3  Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@6ca1c
4  Data inRate    = 8 ,Data outRate    = 16, nr pop done =8 ,nr push done =16
5  Config inRate = 8 ,Config outRate = 16, nr Pop done =8 ,nr push done =16
6  Streamrate test Passed
```

The second part of the test is the test using the property. Here the result of the property test is reported and the user log is printed to the file (listing 5.14).

Listing 5.14: Matrix testrun 1, log part 2

```
1
2  Testing Property: streambits.userprogram.MatrixAx0@665753
3
4  Property test passed
5
6  User Log:
7
8  Testing property A x 0 = 0
9
10 ...
```

The final part of the log reports the results for the complete test and prints the data generated for the test, data produced from the test and the corresponding configuration data (listing 5.15). Both the property test and the streamrate test for each component passed and therefore the complete test passes. The complete test log for this test can be seen in appendix D.1.

Listing 5.15: Matrix testrun 1, log part 3

```
1
2  ***************
3  * Test PASSED *
4  ***************
5
6   Data input: [3][3][3][3]  [3][3][3][3]  [3][3][3][3]  [3][3][3][3]
       [0][0][0][0]  ...
7
8   Data output: 0  0  0  0  0 ...
9
10  Config input:0001 0001 0001 0001 0000 ...
11
12  Config output: 0001  0001  0001  0001  0000  ...
```

5.2.3.2 Testrun 2

A second test i performed using the second property $A \times I = A$. This test is performed one time also using matrices with dimension four. To generate test data a specific data generator is used. Both the streamrate and the property test pass and the complete test is passed (listing 5.16).

Listing 5.16: Matrix testrun 2, log part 1

```
1  ...
2
3  Streamrate test Passed
4
5
6  Testing Property: streambits.userprogram.Unit_MatrixProp@4a65e0
7
8  Property test passed
9
10 ***************
11 * Test PASSED *
12 ***************
13
14 Data input: [0][3][0][1]  [2][7][7][8]  [9][9][2][7]  [0][2][2][2]
      [1][0][0][0]  [0][1][0][0]  ...
15
16 ...
```

5.2.3.3 Testrun 3

In testrun three the programmer uses the same property as in test one and instead the matrix multiplications ability as a stream program meant to be executed over and over again with different matrices is tested. This is done by setting the number of tests argument, performing several tests after each other.

The first test runs and produces a test log identical to testrun one. In the second test a stream rate error occurs causing the pipeline to stall. The timeout function in the test tool aborts the test and produces a test log.

In case of a timeout and aborted test a report of the generator and config filter are written to the log. This is for debugging purposes and enables the programmer to see if the generator and config filter is able to push elements onto the tapes or if these are blocked by the first component. In the log for this test(listing 5.17) both the generator and config filter were able to push all the requested elements to the matrix multiplication pipeline.

Listing 5.17: Matrix testrun 3, log part 1

```
1  Log created : Fri Jan 19 13:47:19 CET 2007
2
3   Test Nr: 1 / 3
4   ...
5
6   Test Nr: 2 / 3
7  **********************************************************************
8
9  Timeout occured due to streamrate error
10
11 Generator info:
12 Number of data elements requested by data generator:      8, number of
       elements possible to push: 8
13 Number of config elements requested by config generator: 8, number of
       elements possible to push: 8
14 ...
```

The next part (listing 5.18) shows the streamrate states at the time the test was aborted. This shows that no values were produced on the tape out and only six values were removed from both the config tape and the data tape. This indicates an error in the pipeline which stalls and therefore does not complete pop of the provided elements.

Listing 5.18: Matrix testrun 3, log part 2

```
1
2  Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@12ac982
3  Data inRate =    8 ,Data outRate=    16, nr pop done =6 ,nr push done =0
4  Config inRate = 8 ,Config outRate= 16, nr Pop done =6 ,nr push done=0
5  ************************
6  *Streamrate test Failed*
7  ************************
```

When a stream rate error that causes a timeout occurs the property test is not performed. This is because a complete set of resulting data is not produced. The test terminates with result; test failed. Complete log can be found in appendix D.3.

5.2.3.4 The stream rate error

The stream rate error in testrun three is caused by a conditional push in the VecMul filter in the matrix multiplication pipeline. When reconfiguring the pipeline and setting new static rows for matrix A there are no data elements pushed forward to the next filter VecAdd, the configuration stream is however still pushed forward. This causes a severe stream rate error as VecAdd waits for data elements from VecMul and VecMul instead waits to push further configuration elements onto the tape to VecAdd which already is

full. This is a typical conditional stream rate error which causes the entire pipeline to stall.

The solution is to always comply with the specified streamrates. Therefore code is added in VecMul which pushes unnecessary data forward in reconfiguration mode i.e dummy data. In this way the filter complies to the defined streamrates also in reconfiguration mode. The dummy data is not a part of the final result and is meant to be discarded at a later time.

5.2.3.5 Testrun 4

The matrix multiplication is again tested as a stream program. The test is built up in the same way as in testrun three with the stream rate error fixed. As the matrix multiplication pipeline now produces n number of dummy data elements which is not part of the result, the property is modified to discard these elements.

This time every part of the test passes (listing 5.19), the complete log can be seen in appendix D.4 .

Listing 5.19: Matrix testrun 4

```
1
2  Log created : Fri Jan 19 13:39:11 CET 2007
3
4   Test Nr: 1 / 3
5
6  ***************
7  * Test PASSED *
8  ***************
9  ...
10  Test Nr: 2 / 3
11
12 ************************************************************************
13 Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@12ac982
14 Data inRate    = 8 ,Data outRate  = 20, nr pop done =8 ,nr push done =20
15 Config inRate = 8 ,Config outRate= 20, nr Pop done =8 ,nr push done =20
16 Streamrate test Passed
17
18 Testing Property: streambits.userprogram.MatrixAx0@14693c7
19
20 Property test passed
21
22 User Log:
23
24 Testing property A x 0 = 0
25
26 ***************
27 * Test PASSED *
28 ***************
29 ...
30
31  Test Nr: 3 / 3
32
33 ***************
```

```
34 * Test PASSED *
35 ***************
36 ...
```

5.2.3.6 Testrun 5

The final test is done using the second property, $A \times I = A$, testing the matrix multiplication as a stream program executed three times. The updated matrix multiplication pipeline is used and the property is modified to discard the dummy data produced.

The test is successful and both the stream rate and the property test pass (listing 5.20). The complete log can be seen in appendix D.5.

Listing 5.20: Matrix testrun 5

```
1
2  Log created : Fri Jan 19 13:42:18 CET 2007
3
4  Test Nr: 1 / 3
5
6  ***************
7  * Test PASSED *
8  ***************
9  ...
10 Test Nr: 2 / 3
11 ************************************************************************
12 Testing data streamrate  , component: framework.streamcomponent.
       Pipeline@12ac982
13 Data inRate =   8 ,Data outRate=   20, nr pop done =8 ,nr push done =20
14 Config inRate = 8 ,Config outRate= 20, nr Pop done =8 ,nr push done=20
15 Streamrate test Passed
16
17
18 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
19
20 Property test passed
21
22 User Log:
23
24 Testing property A x I = A
25
26 ***************
27 * Test PASSED *
28 ***************
29 ...
30  Test Nr: 3 / 3
31
32 ***************
33 * Test PASSED *
34 ***************
35 ...
```

6 Conclusion

The final results of this thesis is a useful and well functioning testing tool for StreamBits which could find previously undetected errors on the first test run with StreamBits programs. The initial goals to develop a tool capable of both specification testing and stream rate testing was achieved.

The tool is developed as a specification based testing tool which motivates extreme programming. The specifications are implemented as predicates which results in a user friendly tool as the programmer does not require the need for a new formal language.

In the beginning of this project, StreamBit was implemented as a single threaded sequential framework implemented in Java. Parallel to this project a new multi threaded version of the framework has been developed by Jerker Bengtsson and Jonathan Andersson. It has been a challenge in this project to keep up with the changes in the framework and to develop a method to test stream rates in multi threaded StreamBit programs. The result is a program which can test stream rates on isolated StreamBit components, sequential stream programs and separate threads. To enable stream rate testing of multi threaded program parts these have to be put together and tested as a single component. The stream test also have the ability to handle stream rate errors that would cause the framework to freeze.

6.1 Future work

The most important future work of the resulting test program is to enhance its capabilities for testing parts of a multi threaded stream program. This is partially implemented with counters in each tape, in the multi threaded parts. The main problem with stream rate test in multi threaded StreamBit programs is to predict the correct stream rate values, as components can be executed different number of times , during one test, even inside a single pipeline.

In this implementation the log file is implemented as a .txt file, future work would be to typeset this as an html or latex document thus simplifying debugging.

There is a risk of using the same formal language for the test objects and the property, the property can be written in the same way as the test object and will therefore have similar weak points. To eliminate this risk the property interface could be provided with additional predefined methods and the possibility of using a second formal language.

We are very satisfied with our achievements and the result of this project. The tool works as intended and we hope that the results of this thesis will aid the future development of a testing tool when StreamBits has become a stand alone programming language.

Bibliography

[1] Jerker Bengtsson, "Efficient implementation of stream applications on processor arrays / Jerker Bengtsson," Licentiate, Chalmers tekniska högskola, 412 96 Göteborg, Feb. 2006.

[2] K. Kuo, "The streamit development tool: A programming environment for streamit," M.Eng. Thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2004. [Online], Available: http://cag.lcs.mit.edu/commit/papers/04/kkuo-meng-thesis.pdf

[3] L. C. Zhenyu Huang, Ed., *Automated Solutions: Improving the efficiency of software testing.* IACIS, 2003. [Online], Available: http://www.iacis.org/iis/2003_iis/PDFfiles/HuangCarter.pdf

[4] B. Liskov and J. Guttag, *Program Development in Java.* Cambridge, Mass.: The MIT Press, 2001.

[5] Jon Edvardsson, "Contributions to Program- and Specification-based Test Data Generation," Licentiate, Department of Computer and Information Science, Linköpings universitet, SE-581 83 Linköping, Sweden, Dec. 2002.

[6] Sun Microsystems, Annotations, Sun Microsystems, Inc., 2004 [cited 2006-06-02]. [Online], Available: http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html

[7] B. Meyer, *Object-Oriented Software Construction*, 2nd ed., ser. The Object-Oriented Series. Englewood Cliffs (NJ), USA: Prentice-Hall, 1997.

[8] Wikimedia Foundation, Design by contract, Wikimedia Foundation, Inc, May 2006 [cited 2006-05-21]. [Online], Available: http://en.wikipedia.org/wiki/Design_by_contract_

[9] K. Beck, *JUnit Pocket Guide*, 1st ed. 10 Fawcett Street Cambridge, MA 02138, USA: O'Reilly Media, 2004.

[10] N. Daley, D. Hoffman, and P. Strooper, "A framework for table-driven testing of Java classes," *softpe*, vol. 32, no. 5, pp. 465–493, Apr. 2002.

[11] Wikimedia Foundation, White Box Testing, Wikimedia Foundation, Inc, Feb. 2006 [cited 2006-12-01]. [Online], Available: http://en.wikipedia.org/wiki/White_box_testing

[12] Wikimedia Foundation, Assertion - computing, Wikimedia Foundation, Inc, 2006 [cited 2006-12-08]. [Online], Available: http://en.wikipedia.org/wiki/Assertion_%28computing%29

[13] J. Nielsen, Junit example, OSTG Open Source Technology Group, 46939 Bayside Parkway, Fremont, CA 94538, Feb. 2006 [cited 2006-12-06]. [Online], Available: http://junit.sourceforge.net/doc/faq/faq.htm

[14] Nilesh Parekh, "Software Testing - Black Box Testing Strategy," *Buzzle.com*, Apr. 2005. [Online], Available: http://www.buzzle.com/editorials/4-10-2005-68349.asp

[15] K. Beck, *Extreme Programming Explained: Embracing Change*. 75 Arlington Street, Suite 300, Boston, MA 02116: Pearson Education, Addison-Wesley Professional., May 1999.

[16] R. Kramer, "iContract – the Java design by contract tool," in *TOOLS 26: Technology of Object-Oriented Languages and Systems*. IEEE Computer Society Press, Aug. 1998, pp. 295–307.

[17] C. Boyapati, S. Khurshid, and D. Marinov, "Korat: Automated testing based on Java predicates," in *ISSTA'02*. Roma, Italy: ACM, 2002, pp. 123–133.

[18] K. Claessen and J. Hughes, "Quickcheck: a lightweight tool for random testing of haskell programs," in *ICFP*, 2000, pp. 268–279. [Online], Available: http://doi.acm.org/10.1145/351240.351266

[19] Wikimedia Foundation, Agile software development, Wikimedia Foundation, Inc, 2006 [cited 2006-12-21]. [Online], Available: http://en.wikipedia.org/wiki/Agile_software_development

[20] David Flanagan, *Java Examples in a Nutshell*, 1st ed. 10 Fawcett Street Cambridge, MA 02138, USA: O'Reilly Media, 2000.

[21] Qusay H. Mahmoud, Using and Programming Generics in J2SE 5.0, Sun Microsystems, Inc., Oct. 2004 [cited 2006-10-23]. [Online], Available: http://java.sun.com/developer/technicalArticles/J2SE/generics/index.html

[22] Wikimedia Foundation, Matrix multiplication, Wikimedia Foundation, Inc, 2005 [cited 2006-12-08]. [Online], Available: http://en.wikipedia.org/wiki/Matrix_multiplication

A Log adder test

This appendix contains complete results logs from tests performed on the adder component.

A.1 Test 1

This test is performed on a single adder filter adding each two elements in to one element out.

Listing A.1: Adder test log 1

```
1  Log created : Sun Dec 10 12:57:49 CET 2006
2
3
4   Test Nr: 1 / 1
5  ************************************************************************
6  Testing data streamrate   , component: streambits.userprogram.
       Adder@156ee8e
7  Data inRate =    20 ,Data outRate=    10, nr pop done =20 ,nr push done
       =10
8  Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
9  Streamrate test Passed
10
11
12 Testing Property: streambits.userprogram.AdderProp@13e205f
13
14 Property test failed
15
16 User Log:
17
18 Calculated result: 104 correct result 86
19
20 Calculated result: 144 correct result 107
21
22 Calculated result: 36 correct result 82
23
24 Calculated result: 92 correct result 91
25
26 Calculated result: 198 correct result 196
27
28 Calculated result: 80 correct result 104
29
30 Calculated result: 170 correct result 113
31
32 Calculated result: 162 correct result 168
33
34 Calculated result: 52 correct result 77
35
36 Calculated result: 52 correct result 97
37
```

```
38 ***************
39 * Test FAILED *
40 ***************
41
42  Data input: 52  34  72  35  18  64  46  45  99  97  40  64  85  28  81
        87  26  51  26  71
43
44  Data output: 104  144  36  92  198  80  170  162  52  52
45
46 ************************************************************************
```

A.2 Test 2

Listing A.2: Adder test log 2

```
1  Log created : Sun Dec 10 11:20:07 CET 2006
2
3
4   Test Nr: 1 / 4
5  ************************************************************************
6  Testing data streamrate  , component: streambits.userprogram.
       Adder@156ee8e
7  Data inRate =   8 ,Data outRate=   4, nr pop done =8 ,nr push done =4
8  Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
9  Streamrate test Passed
10
11 Testing data streamrate  , component: streambits.userprogram.
       Adder@47b480
12 Data inRate =   4 ,Data outRate=   2, nr pop done =4 ,nr push done =2
13 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
14 Streamrate test Passed
15
16 Testing data streamrate  , component: streambits.userprogram.
       Adder@19b49e6
17 Data inRate =   2 ,Data outRate=   1, nr pop done =2 ,nr push done =1
18 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
19 Streamrate test Passed
20
21
22 Testing Property: streambits.userprogram.AdderProp@141d683
23
24 Property test passed
25
26 ***************
27 * Test PASSED *
28 ***************
29
30  Data input: 39  69  29  35  56  36  50  92
31
32  Data output: 406
33
34 ************************************************************************
35
36
37  Test Nr: 2 / 4
```

```
38 ********************************************************************
39 Testing data streamrate  , component: streambits.userprogram.
       Adder@156ee8e
40 Data inRate =   8 ,Data outRate=   4, nr pop done =8 ,nr push done =4
41 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
42 Streamrate test Passed
43
44 Testing data streamrate  , component: streambits.userprogram.
       Adder@47b480
45 Data inRate =   4 ,Data outRate=   2, nr pop done =4 ,nr push done =2
46 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
47 Streamrate test Passed
48
49 Testing data streamrate  , component: streambits.userprogram.
       Adder@19b49e6
50 Data inRate =   2 ,Data outRate=   1, nr pop done =2 ,nr push done =1
51 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
52 Streamrate test Passed
53
54
55 Testing Property: streambits.userprogram.AdderProp@141d683
56
57 Property test passed
58
59 ***************
60 * Test PASSED *
61 ***************
62
63  Data input: 83  13  91  26  11  45  84  97
64
65  Data output: 450
66
67 ********************************************************************
68
69
70  Test Nr: 3 / 4
71 ********************************************************************
72 Testing data streamrate  , component: streambits.userprogram.
       Adder@156ee8e
73 Data inRate =   8 ,Data outRate=   4, nr pop done =8 ,nr push done =4
74 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
75 Streamrate test Passed
76
77 Testing data streamrate  , component: streambits.userprogram.
       Adder@47b480
78 Data inRate =   4 ,Data outRate=   2, nr pop done =4 ,nr push done =2
79 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
80 Streamrate test Passed
81
82 Testing data streamrate  , component: streambits.userprogram.
       Adder@19b49e6
83 Data inRate =   2 ,Data outRate=   1, nr pop done =2 ,nr push done =1
84 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
85 Streamrate test Passed
86
87
88 Testing Property: streambits.userprogram.AdderProp@141d683
```

```
89
90  Property test passed
91
92  ***************
93  * Test PASSED *
94  ***************
95
96   Data input: 44  46  68  46  43  89  68  50
97
98   Data output: 454
99
100 ************************************************************************
101
102
103  Test Nr: 4 / 4
104 ************************************************************************
105 Testing data streamrate  , component: streambits.userprogram.
       Adder@156ee8e
106 Data inRate =   8 ,Data outRate=   4, nr pop done =8 ,nr push done =4
107 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
108 Streamrate test Passed
109
110 Testing data streamrate  , component: streambits.userprogram.
       Adder@47b480
111 Data inRate =   4 ,Data outRate=   2, nr pop done =4 ,nr push done =2
112 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
113 Streamrate test Passed
114
115 Testing data streamrate  , component: streambits.userprogram.
       Adder@19b49e6
116 Data inRate =   2 ,Data outRate=   1, nr pop done =2 ,nr push done =1
117 Config inRate = 0 ,Config outRate= 0, nr Pop done =0 ,nr push done=0
118 Streamrate test Passed
119
120
121 Testing Property: streambits.userprogram.AdderProp@141d683
122
123 Property test passed
124
125 ***************
126 * Test PASSED *
127 ***************
128
129  Data input: 87  36  75  22  13  92  49  37
130
131  Data output: 411
132
133 ************************************************************************
```

B Property - matrix multiplication test

B.1 Property $A \times 0 = 0$

Listing B.1: Property 1 for matrix multiplication, $A \times 0 = 0$

```
1  /*
2   * MatrixAx0.java
3   *
4   * Created on den 18 januari 2007, 10:23
5   *
6   * To change this template, choose Tools | Template Manager
7   * and open the template in the editor.
8   */
9
10 package streambits.userprogram;
11 import framework.streamcomponent.*;
12 import framework.types.*;
13 import java.util.*;
14 import streambits.framework.TestFiles.*;
15
16
17 public class MatrixAx0 implements Property<vecST,intST>{
18
19     int dim;
20
21     public void SetDim(int tmp)
22     {
23         dim = tmp;
24     }
25
26
27     public boolean test(Tape<vecST> indata,Tape<intST> outdata){
28
29         boolean passed=true;
30
31         /*Delete Dummy data */
32
33         for(int i = 0; i < dim; i++)
34         {
35             outdata.pop();
36         }
37
38         int outlength=outdata.length - dim;
39
40         LogPrint.LogPrintln("Testing property A x 0 = 0");
41
42         //check property, output = 0
43         for(int i = 0; i < outlength; i++)
44         {
45             if(outdata.pop().getVal() != 0)
46             {
```

```
47                  passed=false;
48              }
49          }
50
51          return passed;
52      }
53  }
```

B.2 Property $A \times I = A$

Listing B.2: Property 2 for matrix multiplication, $A \times I = A$

```
1  /*
2   * Unit_MatrixProp.java
3   *
4   * Created on den 17 januari 2007, 10:40
5   *
6   * To change this template, choose Tools | Template Manager
7   * and open the template in the editor.
8   */
9
10 package streambits.userprogram;
11 import framework.streamcomponent.*;
12 import framework.types.*;
13 import java.util.*;
14 import streambits.framework.TestFiles.*;
15
16
17 public class Unit_MatrixProp implements Property<vecST,intST>{
18
19     int dim;
20
21     public void SetDim(int tmp)
22     {
23         dim = tmp;
24     }
25     public boolean test(Tape<vecST> indata,Tape<intST> outdata){
26
27         boolean passed=true;
28
29         LogPrint.LogPrintln("Testing property A x I = A");
30
31         /*Delete Dummy data */
32
33         for(int i = 0; i < dim; i++)
34         {
35             outdata.pop();
36         }
37
38         int inlength=indata.length;
39         int outlength=outdata.length - dim;
40         int dimension=dim;
41
42         vecST DataIn[]= new vecST[inlength];
43         intST intST_DataIn[]= new intST[outlength];
44
45
```

```
46          int position=0;
47          for(int i=0; i<inlength;i++)
48          {
49              DataIn[i]=indata.pop();
50          }
51          //Add elements colomn by colomn
52          for(int i=0; i<dimension;i++) {
53              for(int j=0; j<dimension;j++) {
54                  intST_DataIn[position]=(intST)DataIn[j].getElement(i)
                        ;
55                  position++;
56              }
57
58          }
59          //check property
60          for(int i = 0; i < outlength; i++)
61          {
62              if(intST_DataIn[i].getVal()!=outdata.pop().getVal())
63              {
64                  passed=false;
65              }
66          }
67
68          return passed;
69      }
70  }
```

C DataGenerator - matrix multiplication test

Listing C.1: Data generator for Matrix multiplication test

```
1  public class Unit_MatrixFilt <Exp1 extends VecType> extends Source<
       voidST,vecST,voidST,voidST>{
2
3      public void init(){
4
5            /* Define tapes */
6          DefTapes();
7
8            /* Initiation of variables */
9            ...
10
11         for(int j=0;j<test.NrOfTests;j++){
12
13             /* Generate A */
14             for(int i = 0; i < dimension;i++){
15
16                 vecST<Exp1> temp = new vecST<Exp1>();
17
18                 for(int h = 0; h < dimension;h++){
19                         temp.addElement((Exp1)(new intST((int)Low+r.
                               nextInt(((int)High)-(int)Low))));
20                 }
21                 buffer[j].push(temp);
22                 buffer2[j].push(temp);
23             }
24
25             /* Generate I */
26
27             for(int l = 0; l < dimension;l++){
28
29                 vecST<Exp1> temp = new vecST<Exp1>();
30
31                 for(int k = 0; k < dimension;k++){
32
33                     if(l==k){
34                                 temp.addElement((Exp1)(new intST(1)))
                                       ;
35                     }
36                         else{
37                     temp.addElement((Exp1)(new intST(0)));
38                         }
39
40                         buffer[j].push(temp);
41                     buffer2[j].push(temp);
42                 }
43                 }
44             }
45     }
46 }
```

D Log matrix multiplication test

This appendix contains the complete resulting logs from five different tests of a matrix multiplication pipeline.

D.1 Test 1

Listing D.1: Matrix test log 1

```
1  Log created : Fri Jan 19 13:45:50 CET 2007
2
3
4   Test Nr: 1 / 1
5  ************************************************************************
6  Testing data streamrate  , component: framework.streamcomponent.
       Pipeline@12ac982
7  Data inRate =   8 ,Data outRate=   16, nr pop done =8 ,nr push done =16
8  Config inRate = 8 ,Config outRate= 16, nr Pop done =8 ,nr push done=16
9  Streamrate test Passed
10
11
12 Testing Property: streambits.userprogram.MatrixAx0@665753
13
14 Property test passed
15
16 User Log:
17
18 Testing property A x 0 = 0
19
20 ***************
21 * Test PASSED *
22 ***************
23
24  Data input: [3][3][3][3]  [3][3][3][3]  [3][3][3][3]  [3][3][3][3]
       [0][0][0][0]  [0][0][0][0]  [0][0][0][0]  [0][0][0][0]
25
26  Data output: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
27
28  Config input:0001 0001 0001 0001 0000 0000 0000 0000
29
30  Config output: 0001  0001  0001  0001  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000
31
32 ************************************************************************
```

D.2 Test 2

Listing D.2: Matrix test log 2

```
1  Log created : Fri Jan 19 13:49:02 CET 2007
2
3
4   Test Nr: 1 / 1
5  ************************************************************************
6  Testing data streamrate  , component: framework.streamcomponent.
       Pipeline@12ac982
7  Data inRate =   8 ,Data outRate=   16, nr pop done =8 ,nr push done =16
8  Config inRate = 8 ,Config outRate= 16, nr Pop done =8 ,nr push done=16
9  Streamrate test Passed
10
11
12 Testing Property: streambits.userprogram.Unit_MatrixProp@4a65e0
13
14 Property test passed
15
16 ***************
17 * Test PASSED *
18 ***************
19
20  Data input: [0][3][0][1]  [2][7][7][8]  [9][9][2][7]  [0][2][2][2]
       [1][0][0][0]  [0][1][0][0]  [0][0][1][0]  [0][0][0][1]
21
22  Data output: 0  2  9  0  3  7  9  2  0  7  2  2  1  8  7  2
23
24  Config input:0001 0001 0001 0001 0000 0000 0000 0000
25
26  Config output: 0001  0001  0001  0001  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000
27
28 ************************************************************************
```

D.3 Test 3

Listing D.3: Matrix test log 3

```
1  Log created : Fri Jan 19 13:47:19 CET 2007
2
3
4   Test Nr: 1 / 3
5  ************************************************************************
6  Testing data streamrate  , component: framework.streamcomponent.
       Pipeline@12ac982
7  Data inRate =   8 ,Data outRate=   16, nr pop done =8 ,nr push done =16
8  Config inRate = 8 ,Config outRate= 16, nr Pop done =8 ,nr push done=16
9  Streamrate test Passed
10
11
12 Testing Property: streambits.userprogram.MatrixAx0@14693c7
13
14 Property test passed
15
16 User Log:
```

```
17
18 Testing property A x 0 = 0
19
20 ***************
21 * Test PASSED *
22 ***************
23
24  Data input: [3][3][3][3]  [3][3][3][3]  [3][3][3][3]  [3][3][3][3]
       [0][0][0][0]  [0][0][0][0]  [0][0][0][0]  [0][0][0][0]
25
26  Data output: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
27
28  Config input:0001 0001 0001 0001 0000 0000 0000 0000
29
30  Config output: 0001  0001  0001  0001  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000
31
32 ************************************************************************
33
34
35  Test Nr: 2 / 3
36 ************************************************************************
37
38 Timeout occured due to streamrate error
39
40
41 Generator info:
42 Number of data elements requested by data generator:     8, number of
       elements possible to push: 8
43 Number of config elements requested by config generator: 8, number of
       elements possible to push: 8
44
45
46 Testing data streamrate  , component: framework.streamcomponent.
       Pipeline@12ac982
47 Data inRate =   8 ,Data outRate=   16, nr pop done =6 ,nr push done =0
48 Config inRate = 8 ,Config outRate= 16, nr Pop done =6 ,nr push done=0
49 ***********************
50 *Streamrate test Failed*
51 ***********************
52
53
54 Property: streambits.userprogram.MatrixAx0@14693c7 not tested due to
       streamrate error
55
56 ***************
57 * Test FAILED *
58 ***************
59
60  Data input: [3][3][3][3]  [3][3][3][3]  [3][3][3][3]  [3][3][3][3]
       [0][0][0][0]  [0][0][0][0]  [0][0][0][0]  [0][0][0][0]
61
62  Data output:
63
64  Config input:0001 0001 0001 0001 0000 0000 0000 0000
65
66  Config output:
```

```
67 |
68 |**********************************************************************
```

D.4 Test 4

Listing D.4: Matrix test log 4

```
1  |Log created : Fri Jan 19 13:39:11 CET 2007
2  |
3  |
4  | Test Nr: 1 / 3
5  |**********************************************************************
6  |Testing data streamrate   , component: framework.streamcomponent.
   |   Pipeline@12ac982
7  |Data inRate =    8 ,Data outRate=    20, nr pop done =8 ,nr push done =20
8  |Config inRate = 8 ,Config outRate= 20, nr Pop done =8 ,nr push done=20
9  |Streamrate test Passed
10 |
11 |
12 |Testing Property: streambits.userprogram.MatrixAx0@14693c7
13 |
14 |Property test passed
15 |
16 |User Log:
17 |
18 |Testing property A x 0 = 0
19 |
20 |***************
21 |* Test PASSED *
22 |***************
23 |
24 | Data input: [3][3][3][3]   [3][3][3][3]   [3][3][3][3]   [3][3][3][3]
   |     [0][0][0][0]   [0][0][0][0]   [0][0][0][0]   [0][0][0][0]
25 |
26 | Data output: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
27 |
28 | Config input:0001 0001 0001 0001 0000 0000 0000 0000
29 |
30 | Config output: 0001   0001   0001   0001   0000   0000   0000   0000   0000
   |     0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000
31 |
32 |**********************************************************************
33 |
34 |
35 | Test Nr: 2 / 3
36 |**********************************************************************
37 |Testing data streamrate   , component: framework.streamcomponent.
   |   Pipeline@12ac982
38 |Data inRate =    8 ,Data outRate=    20, nr pop done =8 ,nr push done =20
39 |Config inRate = 8 ,Config outRate= 20, nr Pop done =8 ,nr push done=20
40 |Streamrate test Passed
41 |
42 |
43 |Testing Property: streambits.userprogram.MatrixAx0@14693c7
44 |
45 |Property test passed
46 |
```

```
User Log:

Testing property A x 0 = 0

***************
* Test PASSED *
***************

 Data input: [3][3][3][3]  [3][3][3][3]  [3][3][3][3]  [3][3][3][3]
     [0][0][0][0]  [0][0][0][0]  [0][0][0][0]  [0][0][0][0]

 Data output: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 Config input:0001 0001 0001 0001 0000 0000 0000 0000

 Config output: 0001  0001  0001  0001  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000

*********************************************************************


 Test Nr: 3 / 3
*********************************************************************
Testing data streamrate  , component: framework.streamcomponent.
    Pipeline@12ac982
Data inRate =   8 ,Data outRate=   20, nr pop done =8 ,nr push done =20
Config inRate = 8 ,Config outRate= 20, nr Pop done =8 ,nr push done=20
Streamrate test Passed


Testing Property: streambits.userprogram.MatrixAx0@14693c7

Property test passed

User Log:

Testing property A x 0 = 0

***************
* Test PASSED *
***************

 Data input: [3][3][3][3]  [3][3][3][3]  [3][3][3][3]  [3][3][3][3]
     [0][0][0][0]  [0][0][0][0]  [0][0][0][0]  [0][0][0][0]

 Data output: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

 Config input:0001 0001 0001 0001 0000 0000 0000 0000

 Config output: 0001  0001  0001  0001  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000

*********************************************************************
```

D.5 Test 5

Listing D.5: Matrix test log 5

```
1  Log created : Fri Jan 19 13:42:18 CET 2007
2
3
4   Test Nr: 1 / 3
5  ************************************************************************
6  Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@12ac982
7  Data inRate =    8 ,Data outRate=    20, nr pop done =8 ,nr push done =20
8  Config inRate = 8 ,Config outRate= 20, nr Pop done =8 ,nr push done=20
9  Streamrate test Passed
10
11
12 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
13
14 Property test passed
15
16 User Log:
17
18 Testing property A x I = A
19
20 ***************
21 * Test PASSED *
22 ***************
23
24  Data input: [6][6][7][0]   [8][6][2][6]   [8][0][6][9]   [0][9][6][0]
       [1][0][0][0]   [0][1][0][0]   [0][0][1][0]   [0][0][0][1]
25
26  Data output: 0  0  0  0  6  8  8  0  6  6  0  9  7  2  6  6  0  6  9  0
27
28  Config input:0001 0001 0001 0001 0000 0000 0000 0000
29
30  Config output: 0001   0001   0001   0001   0000   0000   0000   0000   0000
       0000   0000   0000   0000   0000   0000   0000   0000   0000   0000   0000
31
32 ************************************************************************
33
34
35  Test Nr: 2 / 3
36 ************************************************************************
37 Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@12ac982
38 Data inRate =    8 ,Data outRate=    20, nr pop done =8 ,nr push done =20
39 Config inRate = 8 ,Config outRate= 20, nr Pop done =8 ,nr push done=20
40 Streamrate test Passed
41
42
43 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
44
45 Property test passed
46
47 User Log:
48
49 Testing property A x I = A
```

```
50
51 ***************
52 * Test PASSED *
53 ***************
54
55  Data input: [6][1][4][6]  [8][5][2][5]  [4][5][4][3]  [6][9][4][0]
        [1][0][0][0]  [0][1][0][0]  [0][0][1][0]  [0][0][0][1]
56
57  Data output: 0  0  0  0  6  8  4  6  1  5  5  9  4  2  4  4  6  5  3  0
58
59  Config input:0001 0001 0001 0001 0000 0000 0000 0000
60
61  Config output: 0001  0001  0001  0001  0000  0000  0000  0000  0000
        0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
62
63 ********************************************************************
64
65
66  Test Nr: 3 / 3
67 ********************************************************************
68 Testing data streamrate  , component: framework.streamcomponent.
       Pipeline@12ac982
69 Data inRate =  8 ,Data outRate=  20, nr pop done =8 ,nr push done =20
70 Config inRate = 8 ,Config outRate= 20, nr Pop done =8 ,nr push done=20
71 Streamrate test Passed
72
73
74 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
75
76 Property test passed
77
78 User Log:
79
80 Testing property A x I = A
81
82 ***************
83 * Test PASSED *
84 ***************
85
86  Data input: [4][7][2][9]  [0][3][8][9]  [5][2][7][2]  [0][9][3][6]
        [1][0][0][0]  [0][1][0][0]  [0][0][1][0]  [0][0][0][1]
87
88  Data output: 0  0  0  0  4  0  5  0  7  3  2  9  2  8  7  3  9  9  2  6
89
90  Config input:0001 0001 0001 0001 0000 0000 0000 0000
91
92  Config output: 0001  0001  0001  0001  0000  0000  0000  0000  0000
        0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
93
94 ********************************************************************
```

E Additional test logs - matrix multiplication test

E.1 Test 1

Test 1 test property $A \times 0 = 0$ five times using matrices of dimension 2.

Listing E.1: Matrix test log 1

```
1  Log created : Tue Jan 23 17:16:20 CET 2007
2
3
4   Test Nr: 1 / 5
5  ************************************************************************
6  Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@12ac982
7  Data inRate =    4 ,Data outRate=    6, nr pop done =4 ,nr push done =6
8  Config inRate = 4 ,Config outRate= 6, nr Pop done =4 ,nr push done=6
9  Streamrate test Passed
10
11
12 Testing Property: streambits.userprogram.MatrixAx0@14693c7
13
14 Property test passed
15
16 User Log:
17
18 Testing property A x 0 = 0
19
20 ***************
21 * Test PASSED *
22 ***************
23
24  Data input: [8][8]   [8][8]   [0][0]   [0][0]
25
26  Data output: 0   0   0   0   0   0
27
28  Config input:0001 0001 0000 0000
29
30  Config output: 0001   0001   0000   0000   0000   0000
31
32 ************************************************************************
33
34
35  Test Nr: 2 / 5
36 ************************************************************************
37 Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@12ac982
38 Data inRate =    4 ,Data outRate=    6, nr pop done =4 ,nr push done =6
39 Config inRate = 4 ,Config outRate= 6, nr Pop done =4 ,nr push done=6
40 Streamrate test Passed
41
42
```

```
43 Testing Property: streambits.userprogram.MatrixAx0@14693c7
44
45 Property test passed
46
47 User Log:
48
49 Testing property A x 0 = 0
50
51 ***************
52 * Test PASSED *
53 ***************
54
55  Data input: [8][8]  [8][8]  [0][0]  [0][0]
56
57  Data output: 0  0  0  0  0  0
58
59  Config input:0001 0001 0000 0000
60
61  Config output: 0001  0001  0000  0000  0000  0000
62
63 ************************************************************************
64
65
66  Test Nr: 3 / 5
67 ************************************************************************
68 Testing data streamrate  , component: framework.streamcomponent.
      Pipeline@12ac982
69 Data inRate =   4 ,Data outRate=   6, nr pop done =4 ,nr push done =6
70 Config inRate = 4 ,Config outRate= 6, nr Pop done =4 ,nr push done=6
71 Streamrate test Passed
72
73
74 Testing Property: streambits.userprogram.MatrixAx0@14693c7
75
76 Property test passed
77
78 User Log:
79
80 Testing property A x 0 = 0
81
82 ***************
83 * Test PASSED *
84 ***************
85
86  Data input: [8][8]  [8][8]  [0][0]  [0][0]
87
88  Data output: 0  0  0  0  0  0
89
90  Config input:0001 0001 0000 0000
91
92  Config output: 0001  0001  0000  0000  0000  0000
93
94 ************************************************************************
95
96
97  Test Nr: 4 / 5
98 ************************************************************************
```

```
99  Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@12ac982
100 Data inRate =    4 ,Data outRate=    6, nr pop done =4 ,nr push done =6
101 Config inRate = 4 ,Config outRate= 6, nr Pop done =4 ,nr push done=6
102 Streamrate test Passed
103
104
105 Testing Property: streambits.userprogram.MatrixAx0@14693c7
106
107 Property test passed
108
109 User Log:
110
111 Testing property A x 0 = 0
112
113 ***************
114 * Test PASSED *
115 ***************
116
117  Data input: [8][8]   [8][8]   [0][0]   [0][0]
118
119  Data output: 0   0   0   0   0   0
120
121  Config input:0001 0001 0000 0000
122
123  Config output: 0001   0001   0000   0000   0000   0000
124
125 ************************************************************************
126
127
128  Test Nr: 5 / 5
129 ************************************************************************
130 Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@12ac982
131 Data inRate =    4 ,Data outRate=    6, nr pop done =4 ,nr push done =6
132 Config inRate = 4 ,Config outRate= 6, nr Pop done =4 ,nr push done=6
133 Streamrate test Passed
134
135
136 Testing Property: streambits.userprogram.MatrixAx0@14693c7
137
138 Property test passed
139
140 User Log:
141
142 Testing property A x 0 = 0
143
144 ***************
145 * Test PASSED *
146 ***************
147
148  Data input: [8][8]   [8][8]   [0][0]   [0][0]
149
150  Data output: 0   0   0   0   0   0
151
152  Config input:0001 0001 0000 0000
153
```

```
154  Config output: 0001  0001  0000  0000  0000  0000
155
156 ************************************************************************
```

E.2 Test 2

Test 2 test property $A \times I = A$ five times using matrices of dimension 2.

Listing E.2: Matrix test log 2

```
1  Log created : Tue Jan 23 16:44:51 CET 2007
2
3
4   Test Nr: 1 / 5
5  ************************************************************************
6  Testing data streamrate  , component: framework.streamcomponent.
       Pipeline@12ac982
7  Data inRate =   4 ,Data outRate=   6, nr pop done =4 ,nr push done =6
8  Config inRate = 4 ,Config outRate= 6, nr Pop done =4 ,nr push done=6
9  Streamrate test Passed
10
11
12 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
13
14 Property test passed
15
16 User Log:
17
18 Testing property A x I = A
19
20 ***************
21 * Test PASSED *
22 ***************
23
24  Data input: [9][3]  [0][9]  [1][0]  [0][1]
25
26  Data output: 0  0  9  0  3  9
27
28  Config input:0001 0001 0000 0000
29
30  Config output: 0001  0001  0000  0000  0000  0000
31
32 ************************************************************************
33
34
35  Test Nr: 2 / 5
36 ************************************************************************
37 Testing data streamrate  , component: framework.streamcomponent.
       Pipeline@12ac982
38 Data inRate =   4 ,Data outRate=   6, nr pop done =4 ,nr push done =6
39 Config inRate = 4 ,Config outRate= 6, nr Pop done =4 ,nr push done=6
40 Streamrate test Passed
41
42
43 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
```

```
44
45 Property test passed
46
47 User Log:
48
49 Testing property A x I = A
50
51 ***************
52 * Test PASSED *
53 ***************
54
55  Data input: [4][6]  [9][9]  [1][0]  [0][1]
56
57  Data output: 0  0  4  9  6  9
58
59  Config input:0001 0001 0000 0000
60
61  Config output: 0001  0001  0000  0000  0000  0000
62
63 ************************************************************************
64
65
66  Test Nr: 3 / 5
67 ************************************************************************
68 Testing data streamrate  , component: framework.streamcomponent.
       Pipeline@12ac982
69 Data inRate =   4 ,Data outRate=   6, nr pop done =4 ,nr push done =6
70 Config inRate = 4 ,Config outRate= 6, nr Pop done =4 ,nr push done=6
71 Streamrate test Passed
72
73
74 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
75
76 Property test passed
77
78 User Log:
79
80 Testing property A x I = A
81
82 ***************
83 * Test PASSED *
84 ***************
85
86  Data input: [7][9]  [9][7]  [1][0]  [0][1]
87
88  Data output: 0  0  7  9  9  7
89
90  Config input:0001 0001 0000 0000
91
92  Config output: 0001  0001  0000  0000  0000  0000
93
94 ************************************************************************
95
96
97  Test Nr: 4 / 5
98 ************************************************************************
```

```
99  Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@12ac982
100 Data inRate =    4 ,Data outRate=    6, nr pop done =4 ,nr push done =6
101 Config inRate = 4 ,Config outRate= 6, nr Pop done =4 ,nr push done=6
102 Streamrate test Passed
103
104
105 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
106
107 Property test passed
108
109 User Log:
110
111 Testing property A x I = A
112
113 ***************
114 * Test PASSED *
115 ***************
116
117  Data input: [3][7]   [0][7]   [1][0]   [0][1]
118
119  Data output: 0   0   3   0   7   7
120
121  Config input:0001 0001 0000 0000
122
123  Config output: 0001   0001   0000   0000   0000   0000
124
125 ************************************************************************
126
127
128  Test Nr: 5 / 5
129 ************************************************************************
130 Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@12ac982
131 Data inRate =    4 ,Data outRate=    6, nr pop done =4 ,nr push done =6
132 Config inRate = 4 ,Config outRate= 6, nr Pop done =4 ,nr push done=6
133 Streamrate test Passed
134
135
136 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
137
138 Property test passed
139
140 User Log:
141
142 Testing property A x I = A
143
144 ***************
145 * Test PASSED *
146 ***************
147
148  Data input: [1][5]   [8][8]   [1][0]   [0][1]
149
150  Data output: 0   0   1   8   5   8
151
152  Config input:0001 0001 0000 0000
153
```

```
154  Config output: 0001  0001  0000  0000  0000  0000
155
156 ************************************************************************
```

E.3 Test 3

Test 3 test property $A \times 0 = 0$ five times using matrices of dimension 8.

Listing E.3: Matrix test log 3

```
1  Log created : Tue Jan 23 17:14:29 CET 2007
2
3
4   Test Nr: 1 / 5
5  ************************************************************************
6  Testing data streamrate  , component: framework.streamcomponent.
       Pipeline@12ac982
7  Data inRate =   16 ,Data outRate=   72, nr pop done =16 ,nr push done
       =72
8  Config inRate = 16 ,Config outRate= 72, nr Pop done =16 ,nr push done=72
9  Streamrate test Passed
10
11
12 Testing Property: streambits.userprogram.MatrixAx0@14693c7
13
14 Property test passed
15
16 User Log:
17
18 Testing property A x 0 = 0
19
20 ***************
21 * Test PASSED *
22 ***************
23
24  Data input: [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
        [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
        [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
        [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
        [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
        [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
        [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
        [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
25
26  Data output: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
          0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0
27
28  Config input:0001 0001 0001 0001 0001 0001 0001 0001 0000 0000 0000
       0000 0000 0000 0000 0000
29
30  Config output: 0001  0001  0001  0001  0001  0001  0001  0001  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
```

```
    0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
    0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
    0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
    0000  0000  0000  0000  0000  0000  0000  0000
31
32 ********************************************************************
33
34
35  Test Nr: 2 / 5
36 ********************************************************************
37 Testing data streamrate  , component: framework.streamcomponent.
      Pipeline@12ac982
38 Data inRate =   16 ,Data outRate=   72, nr pop done =16 ,nr push done
      =72
39 Config inRate = 16 ,Config outRate= 72, nr Pop done =16 ,nr push done=72
40 Streamrate test Passed
41
42
43 Testing Property: streambits.userprogram.MatrixAx0@14693c7
44
45 Property test passed
46
47 User Log:
48
49 Testing property A x 0 = 0
50
51 ***************
52 * Test PASSED *
53 ***************
54
55  Data input: [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
      [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
      [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
      [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
      [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
      [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
      [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
      [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
56
57  Data output: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0  0
58
59  Config input:0001 0001 0001 0001 0001 0001 0001 0001 0000 0000 0000
      0000 0000 0000 0000 0000
60
61  Config output: 0001  0001  0001  0001  0001  0001  0001  0001  0000
      0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
      0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
      0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
      0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
      0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
      0000  0000  0000  0000  0000  0000  0000  0000
62
63 ********************************************************************
64
```

```
65
66  Test Nr: 3 / 5
67 ************************************************************************
68 Testing data streamrate  , component: framework.streamcomponent.
      Pipeline@12ac982
69 Data inRate =   16 ,Data outRate=   72, nr pop done =16 ,nr push done
      =72
70 Config inRate = 16 ,Config outRate= 72, nr Pop done =16 ,nr push done=72
71 Streamrate test Passed
72
73
74 Testing Property: streambits.userprogram.MatrixAx0@14693c7
75
76 Property test passed
77
78 User Log:
79
80 Testing property A x 0 = 0
81
82 ***************
83 * Test PASSED *
84 ***************
85
86  Data input: [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
        [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
        [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
        [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
        [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
        [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
        [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
        [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
87
88  Data output: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
          0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
         0  0  0  0  0  0  0
89
90  Config input:0001 0001 0001 0001 0001 0001 0001 0001 0000 0000 0000
       0000 0000 0000 0000 0000
91
92  Config output: 0001  0001  0001  0001  0001  0001  0001  0001  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000
93
94 ************************************************************************
95
96
97  Test Nr: 4 / 5
98 ************************************************************************
99 Testing data streamrate  , component: framework.streamcomponent.
      Pipeline@12ac982
100 Data inRate =   16 ,Data outRate=   72, nr pop done =16 ,nr push done
      =72
```

```
101 Config inRate = 16 ,Config outRate= 72, nr Pop done =16 ,nr push done=72
102 Streamrate test Passed
103
104
105 Testing Property: streambits.userprogram.MatrixAx0@14693c7
106
107 Property test passed
108
109 User Log:
110
111 Testing property A x 0 = 0
112
113 ***************
114 * Test PASSED *
115 ***************
116
117  Data input: [3][3][3][3][3][3][3][3]   [3][3][3][3][3][3][3][3]
        [3][3][3][3][3][3][3][3]   [3][3][3][3][3][3][3][3]
        [3][3][3][3][3][3][3][3]   [3][3][3][3][3][3][3][3]
        [3][3][3][3][3][3][3][3]   [3][3][3][3][3][3][3][3]
        [0][0][0][0][0][0][0][0]   [0][0][0][0][0][0][0][0]
        [0][0][0][0][0][0][0][0]   [0][0][0][0][0][0][0][0]
        [0][0][0][0][0][0][0][0]   [0][0][0][0][0][0][0][0]
        [0][0][0][0][0][0][0][0]   [0][0][0][0][0][0][0][0]
118
119  Data output: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
          0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
        0  0  0  0  0  0  0
120
121  Config input:0001 0001 0001 0001 0001 0001 0001 0001 0000 0000 0000
        0000 0000 0000 0000 0000
122
123  Config output: 0001  0001  0001  0001  0001  0001  0001  0001  0000
        0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
        0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
        0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
        0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
        0000  0000  0000  0000  0000  0000  0000  0000
124
125 ************************************************************************
126
127
128  Test Nr: 5 / 5
129 ************************************************************************
130 Testing data streamrate  , component: framework.streamcomponent.
       Pipeline@12ac982
131 Data inRate =  16 ,Data outRate=  72, nr pop done =16 ,nr push done
       =72
132 Config inRate = 16 ,Config outRate= 72, nr Pop done =16 ,nr push done=72
133 Streamrate test Passed
134
135
136 Testing Property: streambits.userprogram.MatrixAx0@14693c7
137
138 Property test passed
```

```

User Log:

Testing property A x 0 = 0

***************
* Test PASSED *
***************

 Data input: [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
     [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
     [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
     [3][3][3][3][3][3][3][3]  [3][3][3][3][3][3][3][3]
     [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
     [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
     [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]
     [0][0][0][0][0][0][0][0]  [0][0][0][0][0][0][0][0]

 Data output: 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
       0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
      0  0  0  0  0  0  0

 Config input:0001 0001 0001 0001 0001 0001 0001 0001 0000 0000 0000
     0000 0000 0000 0000 0000

 Config output: 0001  0001  0001  0001  0001  0001  0001  0001  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000

*************************************************************************
```

E.4 Test 4

Test 4 test property $A \times I = A$ five times using matrices of dimension 8.

Listing E.4: Matrix test log 4

```
Log created : Tue Jan 23 17:10:44 CET 2007


 Test Nr: 1 / 5
*************************************************************************
Testing data streamrate  , component: framework.streamcomponent.
    Pipeline@12ac982
Data inRate =   16 ,Data outRate=   72, nr pop done =16 ,nr push done
    =72
Config inRate = 16 ,Config outRate= 72, nr Pop done =16 ,nr push done=72
Streamrate test Passed


```

```
12 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
13
14 Property test passed
15
16 User Log:
17
18 Testing property A x I = A
19
20 ***************
21 * Test PASSED *
22 ***************
23
24  Data input: [9][7][1][1][0][1][3][1]   [6][4][1][1][7][6][6][0]
        [6][3][0][3][4][9][6][7]   [1][2][0][2][3][1][2][1]
        [1][7][0][8][0][8][1][9]   [3][4][4][2][3][5][7][8]
        [4][5][9][1][7][4][9][5]   [7][2][2][7][6][1][4][3]
        [1][0][0][0][0][0][0][0]   [0][1][0][0][0][0][0][0]
        [0][0][1][0][0][0][0][0]   [0][0][0][1][0][0][0][0]
        [0][0][0][0][1][0][0][0]   [0][0][0][0][0][1][0][0]
        [0][0][0][0][0][0][1][0]   [0][0][0][0][0][0][0][1]
25
26  Data output: 0  0  0  0  0  0  0  0  9  6  6  1  1  3  4  7  7  4  3  2
          7  4  5  2  1  1  0  0  0  4  9  2  1  1  3  2  8  2  1  7  0  7
       4  3  0  3  7  6  1  6  9  1  8  5  4  1  3  6  6  2  1  7  9  4  1
        0  7  1  9  8  5  3
27
28  Config input:0001 0001 0001 0001 0001 0001 0001 0001 0000 0000 0000
       0000 0000 0000 0000 0000
29
30  Config output: 0001  0001  0001  0001  0001  0001  0001  0001  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000
31
32 ********************************************************************
33
34
35  Test Nr: 2 / 5
36 ********************************************************************
37 Testing data streamrate  , component: framework.streamcomponent.
      Pipeline@12ac982
38 Data inRate =   16 ,Data outRate=   72, nr pop done =16 ,nr push done
      =72
39 Config inRate = 16 ,Config outRate= 72, nr Pop done =16 ,nr push done=72
40 Streamrate test Passed
41
42
43 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
44
45 Property test passed
46
47 User Log:
48
49 Testing property A x I = A
```

```
50
51 ***************
52 * Test PASSED *
53 ***************
54
55  Data input: [2][3][0][3][6][8][5][7]   [1][9][6][2][0][1][9][3]
        [6][6][9][2][1][3][9][3]   [9][8][7][1][2][1][4][6]
        [8][7][0][3][9][2][5][1]   [7][1][9][3][0][7][4][7]
        [4][7][1][3][2][3][6][3]   [6][0][4][7][6][2][4][0]
        [1][0][0][0][0][0][0][0]   [0][1][0][0][0][0][0][0]
        [0][0][1][0][0][0][0][0]   [0][0][0][1][0][0][0][0]
        [0][0][0][0][1][0][0][0]   [0][0][0][0][0][1][0][0]
        [0][0][0][0][0][0][1][0]   [0][0][0][0][0][0][0][1]
56
57  Data output: 0  0  0  0  0  0  0  0  2  1  6  9  8  7  4  6  3  9  6  8
         7  1  7  0  0  6  9  7  0  9  1  4  3  2  2  1  3  3  3  7  6  0
       1  2  9  0  2  6  8  1  3  1  2  7  3  2  5  9  9  4  5  4  6  4  7
        3  3  6  1  7  3  0
58
59  Config input:0001 0001 0001 0001 0001 0001 0001 0001 0000 0000 0000
       0000 0000 0000 0000 0000
60
61  Config output: 0001  0001  0001  0001  0001  0001  0001  0001  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000
62
63 ********************************************************************************
64
65
66  Test Nr: 3 / 5
67 ********************************************************************************
68 Testing data streamrate   , component: framework.streamcomponent.
      Pipeline@12ac982
69 Data inRate =   16 ,Data outRate=   72, nr pop done =16 ,nr push done
      =72
70 Config inRate = 16 ,Config outRate= 72, nr Pop done =16 ,nr push done=72
71 Streamrate test Passed
72
73
74 Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
75
76 Property test passed
77
78 User Log:
79
80 Testing property A x I = A
81
82 ***************
83 * Test PASSED *
84 ***************
85
86  Data input: [5][2][1][5][0][8][8][3]   [7][1][6][9][1][3][6][4]
       [2][0][5][7][7][5][1][2]   [4][4][9][3][8][3][8][5]
```

```
    [4][0][8][1][1][5][5][8]   [3][0][5][9][9][2][9][6]
    [5][6][4][1][5][4][2][2]   [5][8][2][9][9][5][0][7]
    [1][0][0][0][0][0][0][0]   [0][1][0][0][0][0][0][0]
    [0][0][1][0][0][0][0][0]   [0][0][0][1][0][0][0][0]
    [0][0][0][0][1][0][0][0]   [0][0][0][0][0][1][0][0]
    [0][0][0][0][0][0][1][0]   [0][0][0][0][0][0][0][1]

 Data output: 0  0  0  0  0  0  0  0  5  7  2  4  4  3  5  5  2  1  0  4
       0  0  6  8  1  6  5  9  8  5  4  2  5  9  7  3  1  9  1  9  0  1
     7  8  1  9  5  9  8  3  5  3  5  2  4  5  8  6  1  8  5  9  2  0  3
      4  2  5  8  6  2  7

 Config input:0001 0001 0001 0001 0001 0001 0001 0001 0000 0000 0000
     0000 0000 0000 0000 0000

 Config output: 0001  0001  0001  0001  0001  0001  0001  0001  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000

**************************************************************************


 Test Nr: 4 / 5
**************************************************************************
Testing data streamrate  , component: framework.streamcomponent.
    Pipeline@12ac982
Data inRate =   16 ,Data outRate=   72, nr pop done =16 ,nr push done
    =72
Config inRate = 16 ,Config outRate= 72, nr Pop done =16 ,nr push done=72
Streamrate test Passed


Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7

Property test passed

User Log:

Testing property A x I = A

***************
* Test PASSED *
***************

 Data input: [7][2][9][1][5][5][2][1]   [7][4][6][8][0][1][1][5]
     [2][3][7][3][6][3][5][1]   [0][0][0][0][2][7][5][2]
     [1][2][9][0][2][7][7][5]   [6][8][7][5][4][1][7][9]
     [9][3][2][3][3][1][1][8]   [3][7][9][9][0][4][3][2]
     [1][0][0][0][0][0][0][0]   [0][1][0][0][0][0][0][0]
     [0][0][1][0][0][0][0][0]   [0][0][0][1][0][0][0][0]
     [0][0][0][0][1][0][0][0]   [0][0][0][0][0][1][0][0]
     [0][0][0][0][0][0][1][0]   [0][0][0][0][0][0][0][1]

```

```
119  Data output: 0  0  0  0  0  0  0  0  7  7  2  0  1  6  9  3  2  4  3  0
          2  8  3  7  9  6  7  0  9  7  2  9  1  8  3  0  0  5  3  9  5  0
       6  2  2  4  3  0  5  1  3  7  7  1  1  4  2  1  5  5  7  7  1  3  1
        5  1  2  5  9  8  2
120
121  Config input:0001 0001 0001 0001 0001 0001 0001 0001 0000 0000 0000
       0000 0000 0000 0000 0000
122
123  Config output: 0001  0001  0001  0001  0001  0001  0001  0001  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000
       0000  0000  0000  0000  0000  0000  0000  0000
124
125  ************************************************************************
126
127
128  Test Nr: 5 / 5
129  ************************************************************************
130  Testing data streamrate   , component: framework.streamcomponent.
       Pipeline@12ac982
131  Data inRate =   16 ,Data outRate=   72, nr pop done =16 ,nr push done
       =72
132  Config inRate = 16 ,Config outRate= 72, nr Pop done =16 ,nr push done=72
133  Streamrate test Passed
134
135
136  Testing Property: streambits.userprogram.Unit_MatrixProp@14693c7
137
138  Property test passed
139
140  User Log:
141
142  Testing property A x I = A
143
144  ***************
145  * Test PASSED *
146  ***************
147
148  Data input: [4][7][9][3][1][7][9][5]  [9][2][0][4][3][0][1][1]
       [8][8][8][4][5][1][1][2]  [0][5][8][5][7][5][2][9]
       [8][9][8][2][8][8][6][6]  [1][9][4][9][7][1][5][5]
       [3][2][1][8][8][4][3][5]  [8][3][8][6][7][5][8][9]
       [1][0][0][0][0][0][0][0]  [0][1][0][0][0][0][0][0]
       [0][0][1][0][0][0][0][0]  [0][0][0][1][0][0][0][0]
       [0][0][0][0][1][0][0][0]  [0][0][0][0][0][1][0][0]
       [0][0][0][0][0][0][1][0]  [0][0][0][0][0][0][0][1]
149
150  Data output: 0  0  0  0  0  0  0  0  4  9  8  0  8  1  3  8  7  2  8  5
          9  9  2  3  9  0  8  8  8  4  1  8  3  4  4  5  2  9  8  6  1  3
       5  7  8  7  8  7  7  0  1  5  8  1  4  5  9  1  1  2  6  5  3  8  5
        1  2  9  6  5  5  9
151
152  Config input:0001 0001 0001 0001 0001 0001 0001 0001 0000 0000 0000
       0000 0000 0000 0000 0000
```

```
153
154  Config output: 0001 0001 0001 0001 0001 0001 0001 0001 0000
         0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
         0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
         0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
         0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
         0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
         0000 0000 0000 0000 0000 0000 0000 0000
155
156  ********************************************************************************
```

F Software appendix

The complete testing tool developed in this project is implemented as a package in the StreamBits Java framework. Figure F.1 shows an overview of the different files in this package.

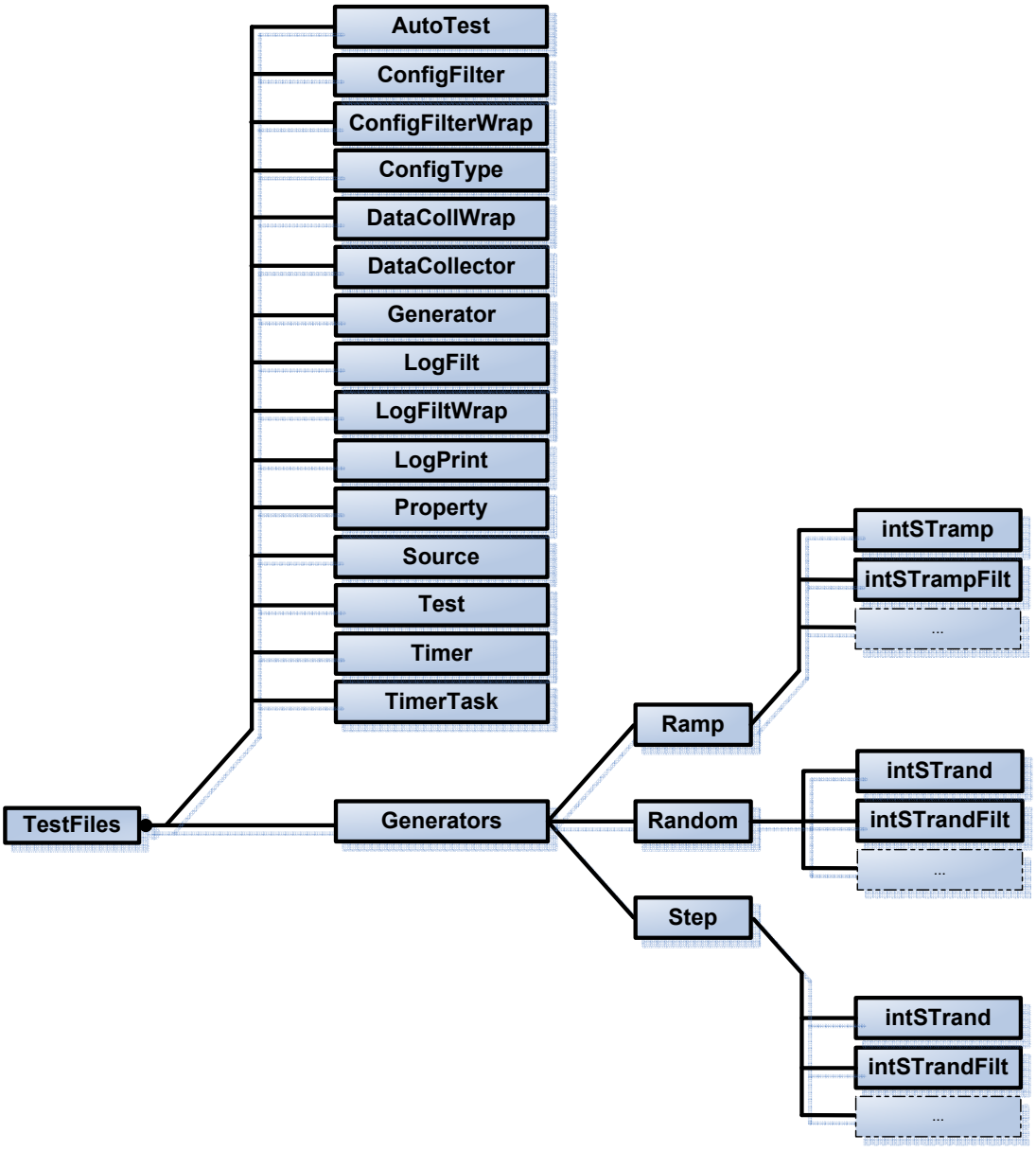

Figure F.1: Testing tool package

- Autotest - Main engine of testing tool
- ConfigFilter - Filter for providing test configuration streams
- ConfigFilterWrap - Wrapper class for ConfigFilter
- ConfigType - Specific type for handling configuration tapes
- DataCollWrap - Wrapper class for DataCollector
- DataCollector - Filter for collecting test data
- Generator - Interface for data generators
- LogFilt - Filter for creating test logs
- LogFiltWrap - Wrapper class for LogFilt
- LogPrint - Class for handling user log
- Property - Interface for writing properties
- Source - Interface for handling data generators
- Test - Class defining TestClass variable
- Timer - Class for handling multiple timers
- TimerTask - Class for handling multiple timer task
- Generators - Package of predefined data generators

  Ramp - Package containing ramp generators

  Random - Package containing random generators

  Step - Package containing step generators