Design and Optimization of the Algorithms and the Data Structures used in the Chip Analysis Tools

By

Harikrushna G. Vanpariya

08MCE019

NIRMA UNIVERSITY
INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2010

Design and Optimization of the Algorithms and the Data Structures used in the Chip Analysis Tools

Major Project

Submitted in fulfillment of the requirements

For the degree of

M.Tech. Computer Science and Engineering

By

Harikrushna G. Vanpariya

08MCE019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2010

Declaration

This is to certify that

i) The thesis comprises my original work towards the degree of Master of Technology in Computer Science and Engineering at Nirma University and has not been submitted elsewhere for a degree.

ii) Due acknowledgement has been made in the text to all other material used.

Harikrushna G. Vanpariya

Certificate

This is to certify that the Major Project entitled ”Design and Optimization of the Algorithms and the Data Structures used in the Chip Analysis Tools” submitted by Harikrushna G. Vanpariya (08MCE019), towards the fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. S.N. Pradhan	Mr. Ashu Talwar
Guide, Professor,	Project Manager
Department Computer Engineering,	ST Microelectronics
Institute of Technology,	Greater NOIDA
Nirma University, Ahmedabad	

Prof. D. J. Patel	Dr. K Kotecha
Professor and Head,	Director,
Department Computer Engineering,	Institute of Technology,
Institute of Technology,	Nirma University,
Nirma University, Ahmedabad	Ahmedabad

Abstract

This project aims to minimize the time and space complexity required in the chip analysis tool. To minimizing the complexity various algorithms has to be implemented and the various tactics are applied at the various level of implementations. These algorithms and tactics are described in the thesis.

Following chip analysis tools has been optimized at various levels.

- Component Descriptor Language Utility
- Timing Power Product tool
- CDL Verilog comparison tool
- Pattern Search engine

A new data structure is designed for Component descriptor language utility to minimize the space complexity of the tool.

Level of parsing and collaboration level has been updated to reduce time complexity of the Timing Power Product tool.

Modifying of the programming language for the optimizing the execution timing for the comparisons in the CDL Verilog comparison tool.

Change in the file structure and the change in the pattern of include file in main file to minimize the time complexity of the pattern search engine.

Acknowledgements

I am deeply indebted to my thesis supervisor for his constant guidance and motivation. He has devoted significant amount of his valuable time to plan and discuss the thesis work. Without his experience and insights, it would have been very difficult to do quality work.

I would also like to extend my gratitude to Mr. Ashu Talwar and Mr. Najaf Zaidi for fruitful discussions during Design and Optimization of the Algorithms and the Data Structures used in the Chip Analysis Tools meetings and for their encouragement.

Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to partial fulfill the requirements For the degree program successfully.

- Harikrushna G. Vanpariya
08MCE019

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Real life tools are designed without the complexity measurements.Effective output can be generated in minimum time by efficiently working tools.

Following Tools get studied and optimized for effectively working.

- Component Descriptor Language Utility
- Timing Power Product tool
- CDL Verilog comparison tool optimization
- Pattern Search engine

1.1 Component Descriptor Language Utility

1.1.1 Description

For any component (chip) created by the developer there is a file created in the back end. This file describes the detail of each pin of the component and also describe that the pin is connected with which pin of internal component. Each component has some internal components inside, those internal components are also described in that file and the connection between various components is also described. SO, for each component there is a bulky file which describe the complete structure of the chip. [5]

1.2 Timing Power Product tool

1.2.1 Description

According to various CMOS technology the modeling libraries are defined. These libraries can be used for the dynamic analysis of the device behavior. Various CMOS technology has level of certification being defined. This level of certification contains the libraries in specific format. By using this certification level the functionality should be fetched from the libraries. These libraries containing the data related to the leakage, timing and power of the device. Equate these data and create a summarized and readable file from it. After creating the complete abstract format from the library, search the minimum and the maximum parameters (timing, power, leakage, current ,VDD, VDDS, VDDI, load, etc.) so that one can easily analysis the behavior of the CMOS technology. Using parameter and the minimum and maximum value of it plot the graph representing the behavior of the device.

1.3 CDL Verilog comparison tool optimization

1.3.1 Description

Minimizing the execution time for the comparison of the CDL (Component Description Language) file and Verilog file. CDL file contains the description of the whole component with the internal pin connection structure.
Verilog file describe the component structure. Verilog HDL is a hardware description language used to design and document electronic systems. Verilog HDL allows designers to design at various levels of abstraction.

1.4 Pattern Search engine

1.4.1 Description

A new language introduce for the user who do not know the shell programming then also able to fulfill his requirements using tool using simple English like language.
This tool has to design from the scratch and fulfill all the requirement with minimum complexity.
The tool has to be dynamically designed so that it can be reusable and modifiable.[3]

1.5 Thesis Organization

The rest of the thesis is organized as follows. Notice how chapters are referred by means of *slash*refhandoff command. Also see in handoff chapter how handoff is labeled.

Chapter 2, *Related terminologies*, describes the various terms like, garbage collector,working of vector in java, Memory allocation by Array List,working of the grep function in the Linux,Verilog file format and the Timing power Product structure overview.

Chapter 3, *Problem Definition and Existing Methodologies*, presents the problem definition and describes the existing methodologies used to execute tools.

In **chapter 4**, *The Proposed Algorithm*, a new algorithm for minimizing the time or space complexity of the tools.

Chapter 5, *Implementation*, describes in brief the implementation performed according to the proposed algorithms and the results gained by making updates. The implementation results along with the performance analysis of the proposed algorithm are presented.

Finally, in **chapter 6** concluding remarks and scope for future work is presented.

Chapter 2

Related Terminologies

Various terminologies have been studied for the various tools relate to the working and requirements of the tool.

2.1 Component Descriptor Language Utility

2.1.1 Garbage collector

The Java virtual machine's heap stores all objects created by a running Java application. Objects are created by the new, newarray, anewarray, and multianewarray instructions, but never freed explicitly by the code.Garbage collection is the process of automatically freeing objects that are no longer referenced by the program. This chapter does not describe an official Java garbage-collected heap, because none exists. As mentioned in earlier chapters, the Java virtual machine specification does not require any particular garbage collection technique. It doesn't even require garbage collection at all. But until infinite memory is invented, most Java virtual machine implementations will likely come with garbage-collected heaps. This chapter describes various garbage collection techniques and explains how garbage collection works in Java virtual machines. Accompanying this chapter on the CD-ROM is an applet that interactively illustrates the material presented in the chapter. The applet, named

Heap of Fish, simulates a garbage-collected heap in a Java virtual machine. The simulation–which demonstrates a compacting, mark-and-sweep collector–allows you to interact with the heap as if you were a Java program: you can allocate objects and assign references to variables. The simulation also allows you to interact with the heap as if you were the Java virtual machine: you can drive the processes of garbage collection and heap compaction. At the end of this chapter, you will find a description of this applet and instructions on how to use it.

Importance of Garbage collector

The name "garbage collection" implies that objects no longer needed by the program are "garbage" and can be thrown away. A more accurate and up-to-date metaphor might be "memory recycling." When an object is no longer referenced by the program, the heap space it occupies can be recycled so that the space is made available for subsequent new objects. The garbage collector must somehow determine which objects are no longer referenced by the program and make available the heap space occupied by such unreferenced objects. In the process of freeing unreferenced objects, the garbage collector must run any finalizers of objects being freed.

In addition to freeing unreferenced objects, a garbage collector may also combat heap fragmentation. Heap fragmentation occurs through the course of normal program execution. New objects are allocated, and unreferenced objects are freed such that free portions of heap memory are left in between portions occupied by live objects. Requests to allocate new objects may have to be filled by extending the size of the heap even though there is enough total unused space in the existing heap. This will happen if there is not enough contiguous free heap space available into which the new object will fit. On a virtual memory system, the extra paging (or swapping) required to service an ever growing heap can degrade the performance of the executing program. On an embedded system with low memory, fragmentation could cause the virtual machine to "run out of memory" unnecessarily.

Garbage collection relieves you from the burden of freeing allocated memory. Knowing when to explicitly free allocated memory can be very tricky. Giving this job to the Java virtual machine has several advantages. First, it can make you more productive. When programming in non-garbage-collected languages you can spend many late hours (or days or weeks) chasing down an elusive memory problem. When programming in Java you can use that time more advantageously by getting ahead of schedule or simply going home to have a life.

A second advantage of garbage collection is that it helps ensure program integrity. Garbage collection is an important part of Java's security strategy. Java programmers are unable to accidentally (or purposely) crash the Java virtual machine by incorrectly freeing memory.

A potential disadvantage of a garbage-collected heap is that it adds an overhead that can affect program performance. The Java virtual machine has to keep track of which objects are being referenced by the executing program, and finalize and free unreferenced objects on the fly. This activity will likely require more CPU time than would have been required if the program explicitly freed unnecessary memory. In addition, programmers in a garbage-collected environment have less control over the scheduling of CPU time devoted to freeing objects that are no longer needed.

Garbage Collection Algorithms

Any garbage collection algorithm must do two basic things. First, it must detect garbage objects. Second, it must reclaim the heap space used by the garbage objects and make the space available again to the program.

Garbage detection is ordinarily accomplished by defining a set of roots and determining reachability from the roots. An object is reachable if there is some path of references from the roots by which the executing program can access the object. The roots are always accessible to the program. Any objects that are reachable from the roots are considered "live." Objects that are not reachable are considered garbage, because they can no longer affect the future course of program execution.

The root set in a Java virtual machine is implementation dependent, but would always include any object references in the local variables and operand stack of any stack frame and any object references in any class variables. Another source of roots are any object references, such as strings, in the constant pool of loaded classes. The constant pool of a loaded class may refer to strings stored on the heap, such as the class name, superclass name, superinterface names, field names, field signatures, method names, and method signatures. Another source of roots may be any object references that were passed to native methods that either haven't been "released" by the native method. (Depending upon the native method interface, a native method may be able to release references by simply returning, by explicitly invoking a call back that releases passed references, or some combination of both.) Another potential source of roots is any part of the Java virtual machine's runtime data areas that are allocated from the garbage-collected heap. For example, the class data in the method area itself could be placed on the garbage-collected heap in some implementations, allowing the same garbage collection algorithm that frees objects to detect and unload unreferenced classes.

Any object referred to by a root is reachable and is therefore a live object. Additionally, any objects referred to by a live object are also reachable. The program is able to access any reachable objects, so these objects must remain on the heap. Any objects that are not reachable can be garbage collected because there is no way for the program to access them.

The Java virtual machine can be implemented such that the garbage collector knows the difference between a genuine object reference and a primitive type (for example, an int) that appears to be a valid object reference. (One example is an int that, if it were interpreted as a native pointer, would point to an object on the heap.) Some garbage collectors, however, may choose not to distinguish between genuine object references and look-alikes. Such garbage collectors are called conservative because they may not always free every unreferenced object. Sometimes a garbage object will be wrongly considered to be live by a conservative collector, because an object reference look-alike referred to it. Conservative collectors trade off an increase in garbage collection speed for occasionally not freeing some actual garbage.

Two basic approaches to distinguishing live objects from garbage are reference counting and tracing. Reference counting garbage collectors distinguish live objects from garbage objects by keeping a count for each object on the heap. The count keeps track of the number of references to that object. Tracing garbage collectors actually trace out the graph of references starting with the root nodes. Objects that are encountered during the trace are marked in some way. After the trace is complete, unmarked objects are known to be unreachable and can be garbage collected. [11]

2.1.2 Array List in java

public class ArrayList
extends AbstractList
implements List, RandomAccess, Cloneable, Serializable
Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

The size, isEmpty, get, set, iterator, and listIterator operations run in constant time. The add operation runs in amortized constant time, that is, adding n elements requires O(n) time. All of the other operations run in linear time (roughly speaking). The constant factor is low compared to that for the LinkedList implementation.
Each ArrayList instance has a capacity.The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost.
An application can increase the capacity of an ArrayList instance before adding a large number of elements using the ensureCapacity operation. This may reduce the amount of incremental reallocation.
Note that this implementation is not synchronized. If multiple threads access an ArrayList instance concurrently, and at least one of the threads modifies the list structurally, it must be synchronized externally. (A structural modification is any operation that adds or deletes one or more elements, or explicitly resizes the backing array; merely setting the value of an element is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the list. If no such object exists, the list should be "wrapped" using the Collections.synchronizedList method. This is best done at creation time, to prevent accidental unsynchronized access to the list.[4]

2.1.3 vector in Java

public class Vector

extends AbstractList

implements List, RandomAccess, Cloneable, Serializable

The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.

Each vector tries to optimize storage management by maintaining a capacity and a capacityIncrement. The capacity is always at least as large as the vector size; it is usually larger because as components are added to the vector, the vector's storage increases in chunks the size of capacityIncrement. An application can increase the capacity of a vector before inserting a large number of components; this reduces the amount of incremental reallocation.

As of the Java 2 platform v1.2, this class has been retrofitted to implement List, so that it becomes a part of Java's collection framework. Unlike the new collection implementations, Vector is synchronized.

The Iterators returned by Vector's iterator and listIterator methods are fail-fast: if the Vector is structurally modified at any time after the Iterator is created, in any way except through the Iterator's own remove or add methods, the Iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the Iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future. The Enumerations returned by Vector's elements method are not fail-fast.[4]

2.2 Timing Power Product tool

2.2.1 Certification level 1: Timing Power, Leakage Tool

Each component has three libraries to represent its behavior. Formats of the libraries for certification level -1 are as follow :

In this version each library (Power2.2, Timing2.1 or Leakage2.3) contains the various cell. Each cell has certain pins. Each pin has specific behavior in terms of Power, timing and leakage. Each library has approx. size 300MB.

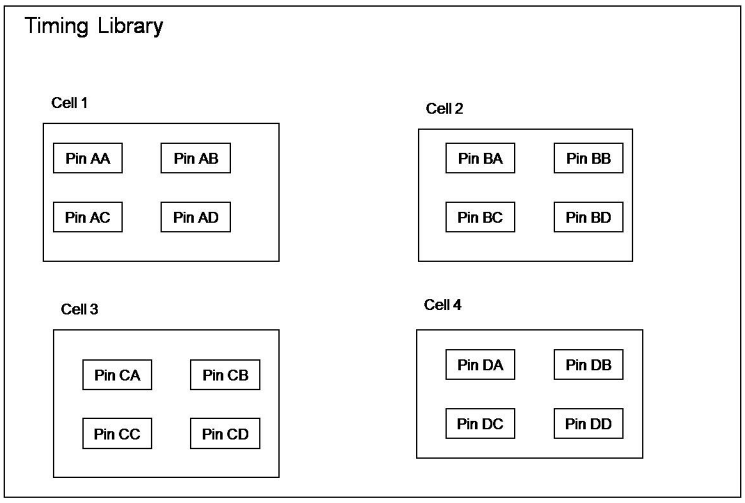


Figure 2.1: Timing Library Structure

2.2.2 Certification level 2: Timing Power, Leakage Tool

For any component single library contains the all parameters. Format of the library for certification level 2 is as follow.

Each pin has specific behavior in terms of Power, timing and leakage as mentioned in 2.4. Each library has approx. size 900MB.

2.3 CDL Verilog comparison tool optimization

2.3.1 Verilog File Structure

Verilog differs from regular programming languages (C, Pascal, ...) in 3 main aspects: (1) simulation time concept, (2) multiple threads, and (3) some basic circuit concepts like network connections and primitive gates. If you know how to program in C and you understand basic digital design then learning Verilog will be easy.

Modules In Verilog, circuit components are designed inside a module. Modules can contain both structural and behavioral statements. Structural statements represent circuit components like logic gates, counters, and microprocessors. Behavioral

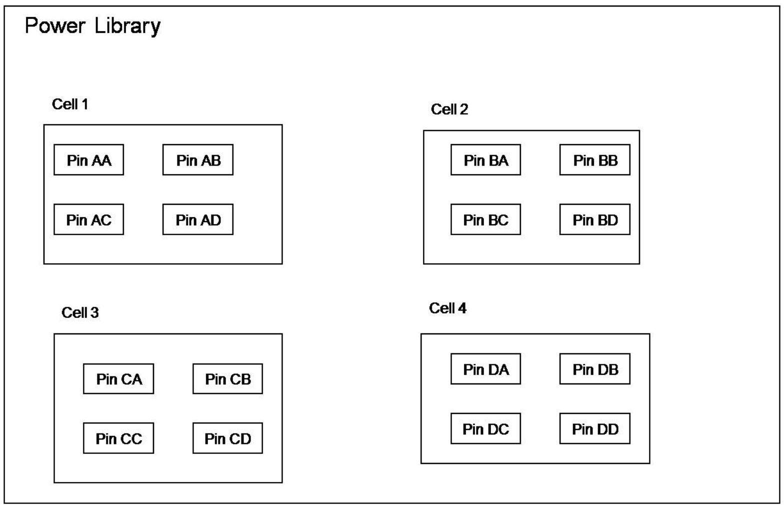


Figure 2.2: Power Library Structure

level statements are programming statements that have no direct mapping to circuit components like loops, if-then statements, and stimulus vectors which are used to exercise a circuit. Figure 1 shows an example of a circuit and a test bench module. A module starts with the keyword module followed by an optional module name and an optional port list. The key word endmodule ends a module. [10]

```
‘timescale 1ns / 1ps
//create a NAND gate out of an AND and an Invertor
module some_logic_component (c, a, b);
       // declare port signals
  output c;
  input a, b;
      // declare internal wire
  wire d;
      //instantiate structural logic gates
  and a1(d, a, b); //d is output, a and b are inputs
  not n1(c, d);    //c is output, d is input
```

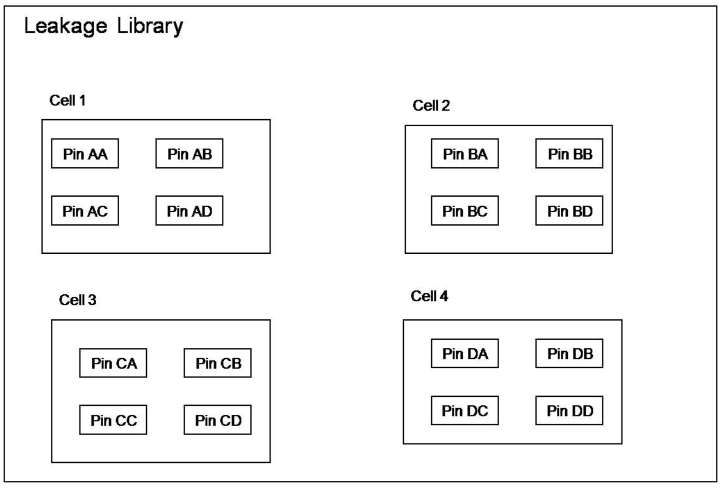


Figure 2.3: Leakage Library Structure

```
endmodule


//test the NAND gate
module test_bench; //module with no ports
  reg A, B;
  wire C;

  //instantiate your circuit
  some_logic_component S1(C, A, B);

  //Behavioral code block generates stimulus to test circuit
  initial
    begin
      A = 1'b0; B = 1'b0;
      #50 $display("A = %b, B = %b, Nand output C = %b \n", A, B, C);
      A = 1'b0; B = 1'b1;
      #50 $display("A = %b, B = %b, Nand output C = %b \n", A, B, C);
```

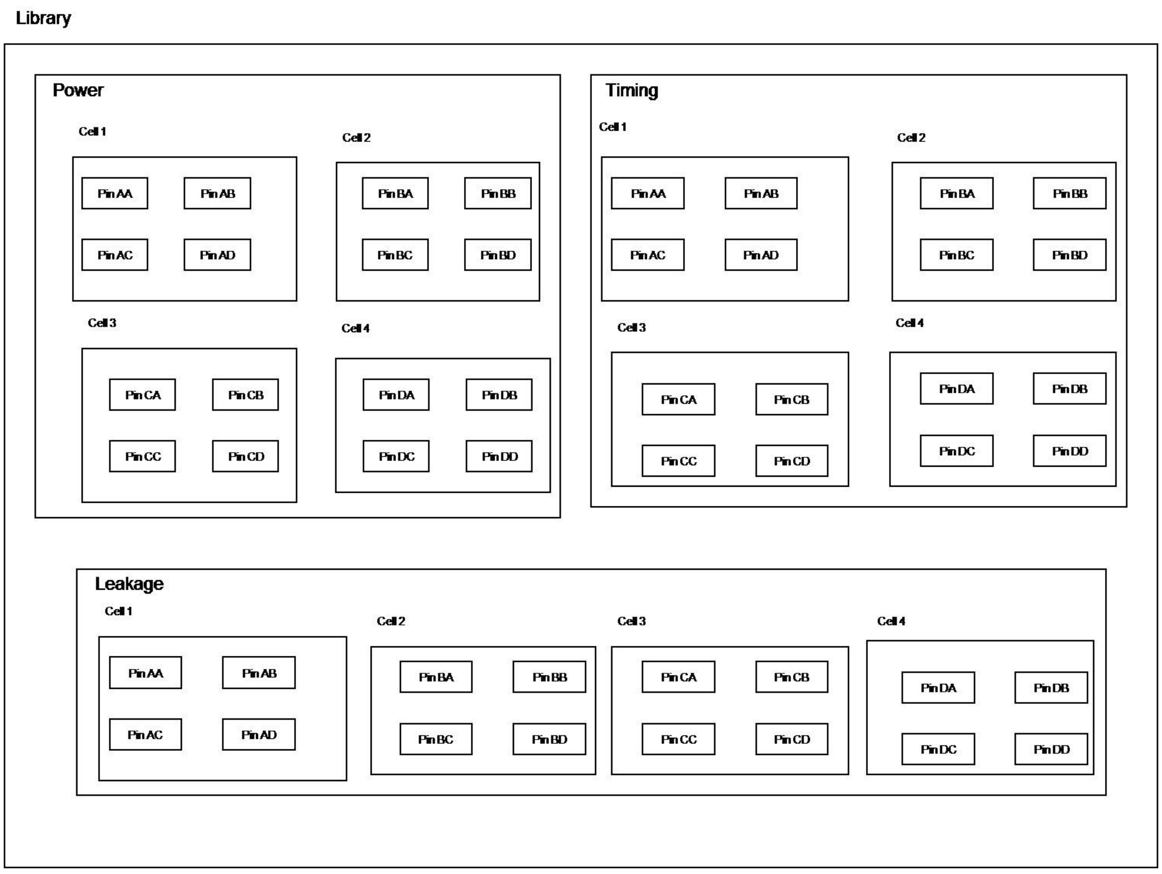


Figure 2.4: All Three Library Structure

```
        A = 1'b1; B = 1'b0;
        #50 $display("A = %b, B = %b, Nand output C = %b \n", A, B, C);
        A = 1'b1; B = 1'b1;
        #50 $display("A = %b, B = %b, Nand output C = %b \n", A, B, C);
    end
endmodule
```

2.4 Pattern Search Engine

2.4.1 Grep function complexity and working

Grep Uses "Boyer-Moore algorithm" for searching pattern

Boyer-Moore algorithm

The algorithm of Boyer and Moore compares the pattern with the text from right to left. If the text symbol that is compared with the rightmost pattern symbol does not occur in the pattern at all, then the pattern can be shifted by m positions behind this text symbol. The following example illustrates this situation

Example:

0	1	2	3	4	5	6	7	8	9	...
a	b	b	a	d	a	b	a	c	b	a
b	a	b	a	c						
					b	a	b	a	c	

The first comparison d-c at position 4 produces a mismatch. The text symbol d does not occur in the pattern. Therefore, the pattern cannot match at any of the positions 0, ..., 4, since all corresponding windows contain a d. The pattern can be shifted to position 5. The best case for the Boyer-Moore algorithm is attained if at each attempt the first compared text symbol does not occur in the pattern. Then the algorithm requires only O(n/m) comparisons

Bad character heuristics

This method is called bad character heuristics. It can also be applied if the bad character, i.e. the text symbol that causes a mismatch, occurs somewhere else in the pattern. Then the pattern can be shifted so that it is aligned to this text symbol. The next example illustrates this situation Example:I

Comparison b-c causes a mismatch. Text symbol b occurs in the pattern at

Table I: Bad character heuristics

0	1	2	3	4	5	6	7	8	9	...
a	b	b	a	b	a	b	a	c	b	a
b	a	b	a	c						
		b	a	b	a	c				

Table II: Good suffix heuristics

0	1	2	3	4	5	6	7	8	9	...
a	b	a	a	b	a	b	a	c	b	a
c	a	b	a	b						
		c	a	b	a	b				

positions 0 and 2. The pattern can be shifted so that the rightmost b in the pattern is aligned to text symbol b.

Good suffix heuristics

Sometimes the bad character heuristics fails. In the following situation the comparison a-b causes a mismatch. An alignment of the rightmost occurrence of the pattern symbol a with the text symbol a would produce a negative shift. Instead, a shift by 1 would be possible. However, in this case it is better to derive the maximum possible shift distance from the structure of the pattern. This method is called good suffix heuristics Example:II

The suffix ab has matched. The pattern can be shifted until the next occurrence of ab in the pattern is aligned to the text symbols ab, i.e. to position 2. In the following situation the suffix ab has matched. There is no other occurrence of ab in the pattern.Therefore, the pattern can be shifted behind ab, i.e. to position 5.III

Example:

In the following situation the suffix bab has matched. There is no other occurrence of bab in the pattern. But in this case the pattern cannot be shifted to position 5 as before, but only to position 3, since a prefix of the pattern (ab) matches the end of bab. We refer to this situation as case 2 of the good suffix heuristics.

Table III: Good suffix heuristics example

0	1	2	3	4	5	6	7	8	9	...
a	b	c	a	b	a	b	a	c	b	a
c	b	a	a	b						
					c	b	a	a	b	

Table IV: Good Suffix example case - II

0	1	2	3	4	5	6	7	8	9	...
a	a	b	a	b	a	b	a	c	b	a
a	b	b	a	b						
			a	b	b	a	b			

Example:IV

The pattern is shifted by the longest of the two distances that are given by the bad character and the good suffix heuristics.

Searching algorithm

The searching algorithm compares the symbols of the pattern from right to left with the text. After a complete match the pattern is shifted according to how much its widest border allows. After a mismatch the pattern is shifted by the maximum of the values given by the good-suffix and the bad-character heuristics

Boyer-Moore searching algorithm

```
void bmSearch() {
    int i=0, j;
    while (i<=n-m)
    {
        j=m-1;
        while (j>=0 && p[j]==t[i+j]) j--;
        if (j<0)
        {
            report(i);
```

```
                i+=s[0];
            }
            else
                i+=Math.max(s[j+1], j-occ[t[i+j]]);
        }
    }
```

Analysis

If there are only a constant number of matches of the pattern in the text, the Boyer-Moore algorithm performs O(n) comparisons in the worst case. The proof of this is rather difficult. In general O(nm) comparisons are necessary, e.g. if the pattern is am and the text an. By a slight modification of the algorithm the number of comparisons can be bounded to O(n) even in the general case. If the alphabet is large compared to the length of the pattern, the algorithm performs O(n/m) comparisons on the average. This is because often a shift by m is possible due to the bad character heuristics. [3]

Chapter 3

Problem Definition and Existing Methodologies

3.1 Component Descriptor Language (CDL) Utility

3.1.1 Existing methodology

- Vectors are used to store the structure of the component.
- No garbage collector is used to re collect the free space
- Format used to store the structure of the component was not optimum.

3.1.2 Problem statement

- The file more that 500MB cannot be stored in the structure defined.
- It required more memory at the run time to store the complete structure of the component.
- The execution time required to the complete file is very high.

3.2 Timing Power Product tool

3.2.1 Existing work

Algorithm

- Step 1. Parse the input value and store in the corresponding variable
- Step 2. Validate the input value accordingly [Complexity : O(maximum input value possible)]
  - File given as argument does exist
  - Validate the cell name in the libraries
- Step 3. Get the all PVT names that has to be parsed
- Step 4. Go through each PVT and parse Power Library
- Step 5. Go through each PVT and parse Timing Library
- Step 6. Go through each PVT and parse Leakage Library
- Step 7. Merge the libraries by cell name and pin name O (n*m*p) n=number of cells in timing libraries , m=number of cells in power libraries , p= number of cells in Leakage libraries
- Step 8. Parse Timing Library [O (n*m), n= total cell in the file , m=total pin in the selected cell]
  - Search for the cell name O (n) , n=total cell in the file
  - Get the pin from the particular cell
  - Search for the timing definition defined in the pin detail
  - If the timing details found for the pin then get the related Pin details.
  - Get the unique comment to uniquely identify the timing element of the particular pin

  - Get the condition for falling and rising details for the pin

- Step 9. Parse Power Library [O (n*m), n= total cell in the file , m=total pin in the selected cell]
  - Search for the cell name
  - Get the pin from the particular cell
  - Search for the power definition defined in the pin detail
  - If the power details found for the pin then get the related Pin details.
  - Get the unique comment to uniquely identify the timing element of the particular pin
  - Get the condition for falling and rising details for the pin
- Step 10. Parse the value from the Leakage library
  - Get the supply value defined in the leakage library
  - Compare the value of the supply with the minimum and maximum value of the supply defined by user

3.2.2 Problem statement

Certification level 1: Timing Power, Leakage Tool

- Fetch any parameter (timing ,Power, leakage or supply) for any cell
- Parameter can be fetched for any specific pin or all pins of a particular cell.
- Limit the value of the any parameter
- All parameter for whole library can be extracted
- Store the output in the readable file format (.csv)
- Plot graph for the extracted parameters

- Time Complexity must be optimum
- Optimizing execution time

Certification level 2: Timing Power, Leakage Tool

- Fetch any parameter (timing ,Power, leakage or supply) for any cell
- Fetch any parameter (timing ,Power, leakage or supply) for any cell
- Parameter can be fetched for any specific pin or all pins of a particular cell.
- Limit the value of the any parameter.
- All parameter for whole library can be extracted.
- Store the output in the readable file format (.csv)
- Plot graph for the extracted parameters.
- Time Complexity must be optimum.
- Optimizing execution time
- Compatible with the older version of certification level.

3.3 CDL Verilog comparison tool optimization

3.3.1 Existing Methodology

Existing Parsing Algorithm

```
Store value assigned to a variable\\
Parse the string and store minimum and maximum value\\
For mimumvalue to maximum value\\
    Print pinaname with pinnumber and value\\
End For\\
```

3.3.2 Problem Statement

Minimize the execution time complexity of the CDL verilog comparison tool.

3.4 Pattern Search engine

3.4.1 Problem Definition

- pattern can be reused many times
- for same file more than one pattern can be defined
- for any file dependency of the pattern can be specified
- instance of pattern in particular file can be mentioned
- to search the whole directory files can be omitted from searching
- specific rules can be defined for specific file in same directory

3.4.2 Utile file Optimization

Utile file contains the details of the total number on a specific cell. It also contains the detail of the details of the strobe in terms of timing and strobe in terms of value details. It has the details that which pin should be strobe at which time and the expected value at the specific time at that pin. The specific check of the pattern search engine tool get missing or extra pins which are not strobe for timing or value from the utile file. The execution time of the finding missing/extra pin has to be optimized.[9]

3.5 Summary

This chapter presented work already done to solve the problem described in section 3.2.2. After studying these terminologies the way to optimize the tool is found. [8]

Chapter 4

The Proposed Algorithm

Describe your approach in this chapter. It may include algorithms, equations, figures or any other model to support your method.

4.1 Timing Power Product tool

4.1.1 Proposed Algorithm

- Step 1. Parse the input value and store in the corresponding variable
- Step 2. Validate the input value accordingly [Complexity : O(maximum input value possible)]
  - File given as argument does exist
  - Validate the cell name in the libraries
- Step 3. Get the all PVT names that has to be parsed
- Step 4. Go through each PVT and parse Power Library
- Step 5. Go through each PVT and parse Timing Library
- Step 6. Go through each PVT and parse Leakage Library

- Step 7. Parse Timing Library [O (n*m), n= total cell in the file , m=total pin in the selected cell]
  - Optimization : If user do not want to search all cell and all pins the complexity reduces to O(n*m) , n=required cells , m=selected pins)
  - Search for the cell name [O(n) , n=total cell in the file]
  - Get the pin from the particular cell [O(n) , n= total number of pin in selected cell]
  - Search for the timing definition defined in the pin detail
  - If the timing details found for the pin then get the related Pin details.
  - Get the unique comment to uniquely identify the timing element of the particular pin
  - Get the condition for falling and rising details for the pin
- Step 8. Parse Power Library [O (n*m), n= total cell in the file , m=total pin in the selected cell]
  - Optimization : If user do not want to search all cell and all pins the complexity reduces to O(n*m) , n= required cells , m=selected pins)
  - Search for the cell name [O(n) , n=total cell in the file]
  - Get the pin from the particular cell [O(n) , n= total number of pin in selected cell]
  - Search for the power definition defined in the pin detail
  - If the power details found for the pin then get the related Pin details.
  - Get the unique comment to uniquely identify the timing element of the particular pin
  - Get the condition for falling and rising details for the pin
- Step 9. Combine the out put of the timing and the power library

- Step 10. Do following process to the combination of the timing and power library
  - Uniquely identify the each element of the timing and power
  - To collaborate the value of the timing and the power library value.
  - Rearrange the value to get the data in specific format (.CSV)
- Step 11. Parse the value from the Leakage library
  - Get the supply value defined in the leakage library
  - Compare the value of the supply with the minimum and maximum value of the supply defined by user

4.2 CDL Verilog comparison tool optimization

Generate another file (AWK) for looping and printing the value.

```
maxCBusValue=15
for (minimumvalue to maximumvalue) {
    print pinname with pinnumber and value
}
```

4.3 Pattern Search engine

4.3.1 Proposed Design Structure for Pattern Search Engine

Design for the pattern search engine is described in the following table.

Table I: Design for Pattern Search Engine

Name	**Description**	**Example**
Pattern Name(M):	Name of the pattern	frstPatrn
Pattern(M):	pattern syntax	$^{A} * [0-9] * [A-Z]$
Expression Name(M):	Name of the expression(unique)	frstExpr
Expression(M):	syntax of the expression	frstPatrn [joiner] secPatrn [joiner] thrdPatrn
Delimiter(O):	delimiter to which field should be separated	":"
field number(O):	after applying delimiter which field have to be fetched from the file	4
Depends on Expression(O)	Name of the expression on which this expression depends	thrdExpr
Range Name (M):	Name of range	RngNm
start pattern / line number (M if end pattern is not defined)	line number or the regular expression from which the data has to start fetching	frstPatrn (or 5)
end pattern / line number(M)	last line number or the regular expression till which the data has to be fetched	lstPatrn (or 100)
Delimiter(O):	delimiter to which field should be separated	":"
field number(O):	after applying delimiter which field have to be fetched from the file	4

Joiner:	eq,AND, OR,!(can be used instead of joiner)	
Pattern List(M): (Sequence is important here):	Name of pattern List	patrnLst
Pattern(M/O):	Name of the already defined pattern	frstPatrn
Pattern(M/O):	Name of the already defined pattern	thrdPatrn
Script Name(M/O):	Name of the script which want to execute	scrptNm
Range Name(M/O):	name of the range already defined in the file	RngNm
Expression Name(M/O):	Name of the expression already defined	thrdExpr
Collection(M):	Name of collection	frstCol
Filename(M):	complete path of the file with the name	subdir1/subsubdir2/file30 file3
Patterns Name /Expression Name /Range Name	pattern name or expression name or patten list separated by space which wanted to apply to the file is written here	frstPatrn thrdExpr patrnLst Rngnm

Patterns/Expression /pattern List(O):	repeat as above	
Place to search(O):	repeat as above	
Instance of pattern(O):	repeat as above	
Directory File(O):	true (if directory for file is specified)	true/false(not req. to def.)
Collection(M):	collection for a directory	colYYY
Directory Name(M):	name of the directory in which the pattern has to be searched	dir1
Files to exclude(O):	pattern not to search from some file then mention its name in this field	subdir1/subsubdir2/file30
Patterns Name/Expression /Range name Pattern List(M)	pattern name or expression name or patten list separated by space which wanted to apply to the file is written here	frstPtrn thrdExpr ptrnLst RngNm
Place to search(O):	particular place to search the pattern EOF (End of File), EOL (End of Line),BOF(Begin of File),BOL(Begin of Line)	EOF (End of File)
Instances of pattern(O):	number of instances wanted to search from the file more than this will result in negative output	5
Special File(O):	if special pattern has to apply to some files of the directory	file1
Output(O):	to reuse the intermediate search result	outFrstCol
Patterns/Expression /pattern List(O):	repeat as above	
Place to search(O):	repeat as above	
Instance of pattern(O):	repeat as above	

Output Status

successfully and all	0	If search
	Non-zero:	log will be generated
		for the failure file
		or directory

Design Diagram for Pattern Search Engine

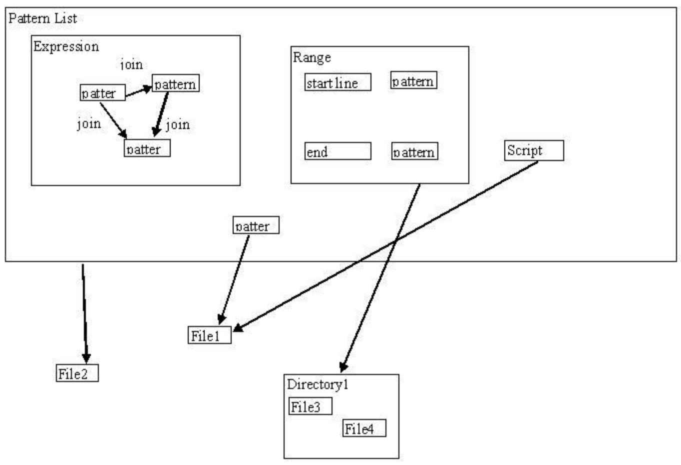


Figure 4.1: Design Architecture of Pattern Search Engine

4.3.2 Utile file Optimization

Utile file format is as follow :

```
filename_extention.utile
```

```
Program  "fileName" "extention"; #include
"relativePath/includeFile1"
```

```
All required pin details.

#include "relativePath/includeFile2"

For i:=0 to 100
    Set pin value for i

Strobe for timing to set value for various cycle . . .

Strobe for value to set the value at each pin . . .
```

Current implementation

Include the file in the main file with the file name above it. If the extra pin found then it has to backtrack the complete included file to get the extra pin location. Depicted in the figure 4.2

4.4 Summary

- Timing Power Product tool
  By using this algorithm the time complexity and the space complexity can be very with the number of the cells required in the output. [2] [1] [7]

- CDL Verilog comparison tool optimization
  The algorithm defined for the tool will optimize the time complexity required to generate the pin list from the verilog file which is the maximum time consuming task of the tool

- Pattern Search engine
  The Design for the tool is the used friendly and reusable with the optimum

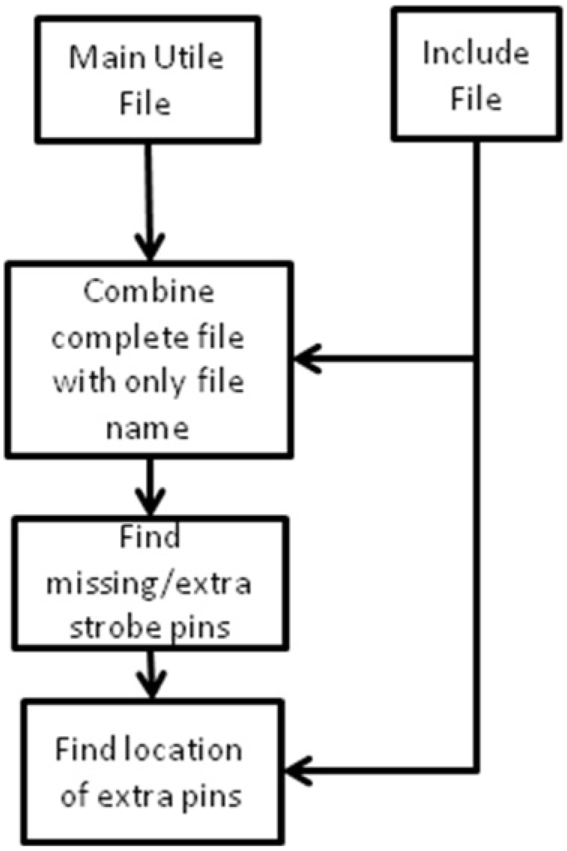

Main Utile File
Include File
Combine complete file with only file name
Find missing/extra strobe pins
Find location of extra pins

Figure 4.2: Execution flow to get missing pin before optimization.

options. [6] [5]

Chapter 5

Implementation

5.1 Component Descriptor Language Utility

- Use of vector
  - Internally, both the ArrayList and Vector hold onto their contents using an Array. When you insert an element into an ArrayList or a Vector, the object will need to expand its internal array if it runs out of room. A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.[4]
  - Due to use of vector the memory was getting out of room at the time of execution[4]
  - Thus data structure used in the tool was updated to ArrayList instead of Vector[4]
- Garbage Collector
  - Garbage collector execute during the execution of the java code at regular interval of time. When the garbage collector called it frees the memory of variable which is not in used, but it cannot be able to free the memory of

any data structure used. So increase the number of variables instead of data structure. [11]

5.2 Timing Power Product tool

Timing Power Product Tool operates in steps described in the algorithm. The diagram describes the complete working before and after the optimizing the tool.

Basically the tool work in following steps

- Get path of various libraries like timing, power and leakage
- Merge the libraries fetch the required parameter from the external parameters applied to the tool
- Generate the .csv file as output
- By using Macros the graph can be plotted from the .csv(output) file.

Above steps are depicted in following diagram 5.1

The merging of the library was performed before the fetching the useful data in the previous version of the tool.5.2

To minimize the time complexity the merging of the file is performed only after the useful data has been fetched from the file. 5.3

5.2.1 results

After the optimization following optimization has been achieve in the tool execution time.I

5.3 CDL Verilog comparison tool optimization

CDL Verilog comparison tool works on the Algorithm described in the Algorithm section. CDL Verilog comparison tool generate the pin list from both CDL and

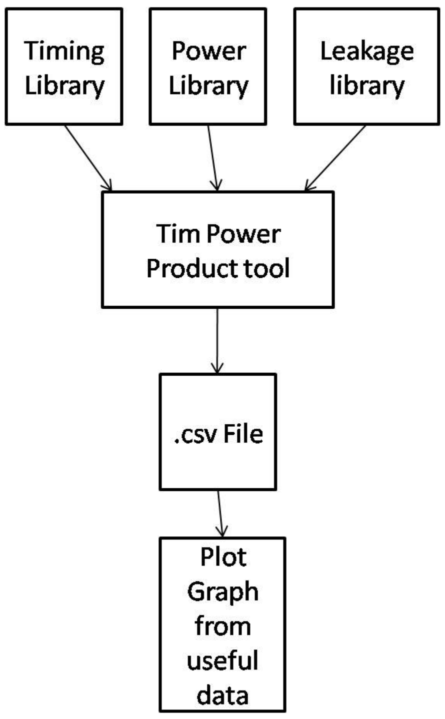


Figure 5.1: Flow of the Timing Power Product tool

Verilog file and compare it. Time consumed to the generate pin list from the Verilog file has to be reduced to optimize the tool's time complexity. 5.4

5.4 Pattern Search engine

Pattern search engine is the tool which is used to create custom collection of the pattern through which particular result get fetched from the files or the directories.It's working is described in the following steps:

- Checks are defined with the list of the patterns and the global patterns
- Function file is defined by the developer and time by time new function can be updated to this file to add more functionality to the tool.
- By using check file and the function admin can generate different kind of checks

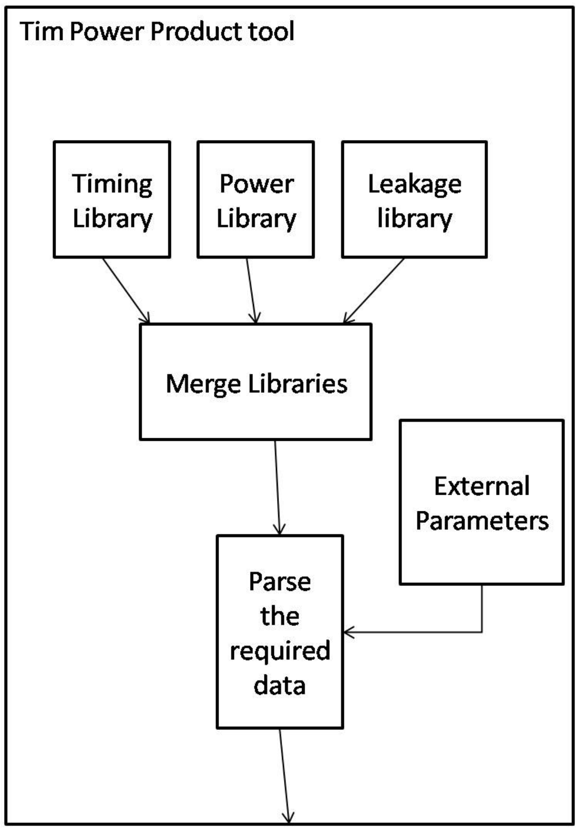


Figure 5.2: Flow of the Timing Power Product tool before optimization

according to requirements.

- By using the check files and the function files Pattern search engine tool generate three type of data.
  - Template for AQA tool. Which contains the path of the check list in shell format and function file path
  - Checks in Shell programming
  - All function files in collaborate format.
- AQA tool collaborate all the checks and with the functions using AQA template file. The tool generate single executable product out of it.

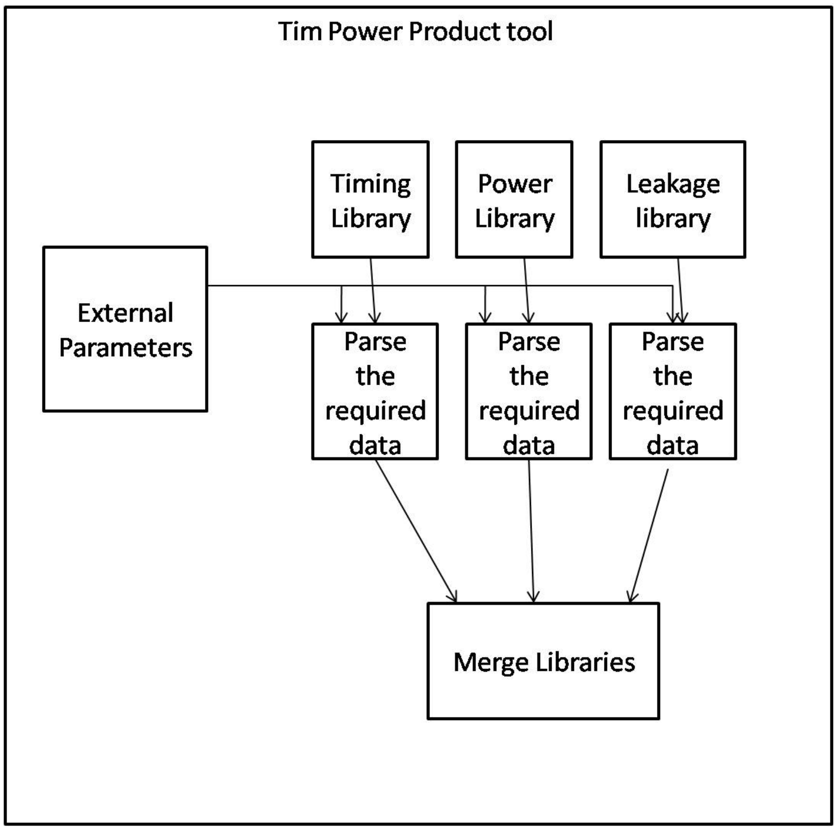


Figure 5.3: Flow of the Timing Power Product tool after optimization

- Some dynamic parameters are set to the final product which can be changed at the run time.
- Final executable single product can be executed by changing the dynamic variables in it.

Work flow diagram

Workflow of the Pattern search engine is describe in the following diagram. 5.5

Implemented components

Function Files These files are created by the programmer(s). It contains shell programming functions. These functions get the arguments as inputs and generate

Table I: Timing,Power and Leakage complexity

Previous Timing Power Leakage algorithm	O(n*m*p), n= total number of cell in timing library, m= total number of cell in leakage library, p= total number of cell in Leakage Library
Optimized Timing Power Leakage Algorithm	O(n*m*p), n= selected number of cell in timing library, m= selected number of cell in leakage library, p= selected number of cell in Leakage Library

two parameters.

a. Output of searched pattern in a log file.

b. Status of executed commands.

Check Files These files created by the Admin. Checks are created using functions defined in the function files. Each check can individually save and integrated with other check. Checks define the sequence of results and the pattern/files to generate these results.

Parser Parser parses the saved function and check files. Parser generates suitable data structure according to the file given as the input. It supports following two kind of data structure:

- Data Structure for functions

```
Function Data Structure
    Name of function
    Argument Data Structure
        Name of argument
```

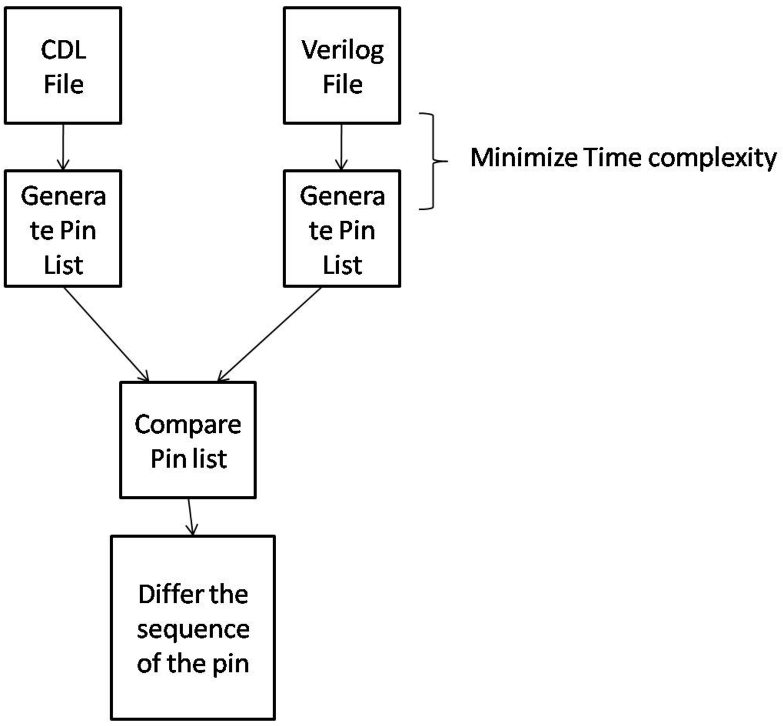


Figure 5.4: Flow of the CDL Verilog comparison tool

```
        Valid Types of Argument (List)
    Function description
```

- Data Structure for checks

```
Check Data Structure
    Check name
    Variable Data Structure
        Name of Variable
        Type of Variable
        Value of Variable
    Result Data Structure
```

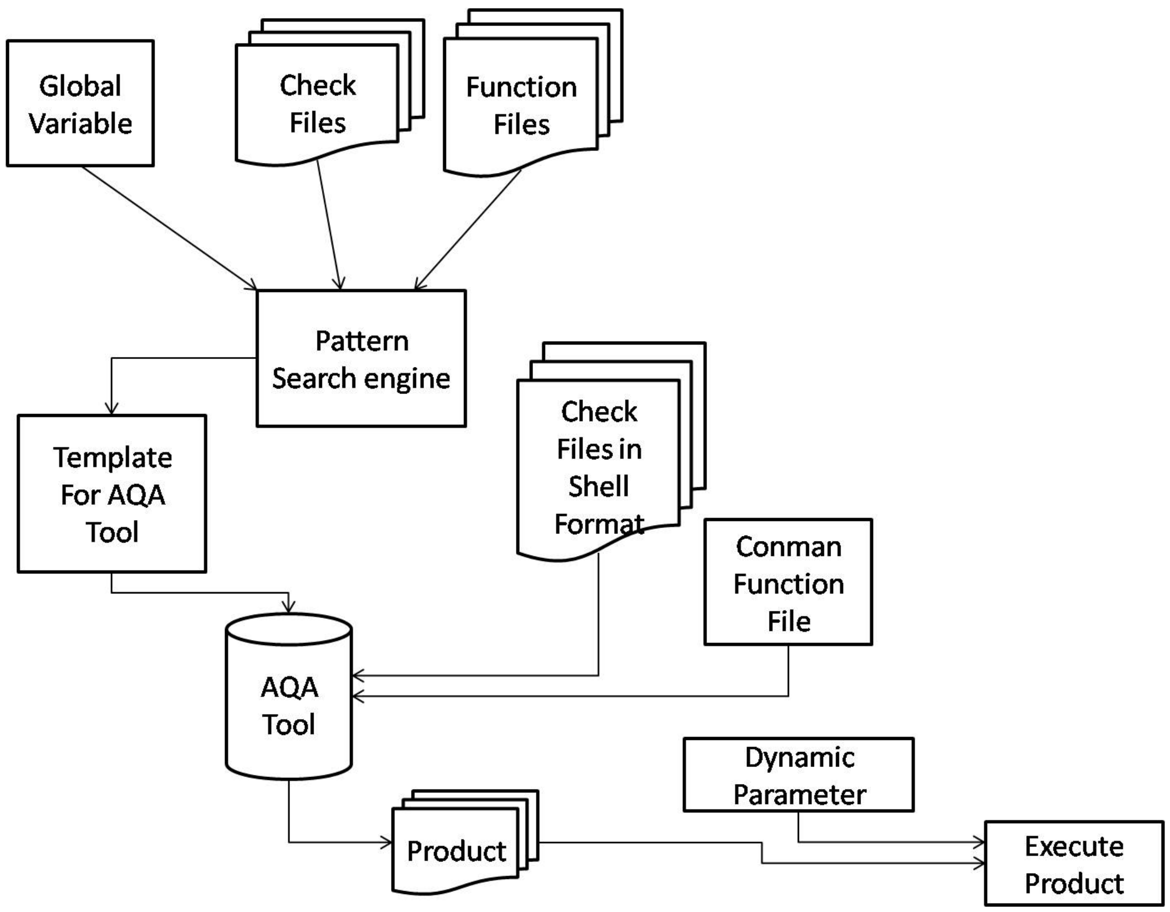


Figure 5.5: Workflow of the search engine tool.

```
          Name of Result
          Type of Result
          Values of Result
      Final Result
```

AQA Template file
- These file is used to configure AQA tool
- It also contains the relative paths of the shell files and supporting function files

Shell files
- These files contain the script to execute and generate the output (log) file and status (Success/Fail) of the execution.
- This file contains the relative path of Function files

Supporting function files
- Supporting function file contains the function with the description of the each function defined in it

5.4.1 Class Description

5.4.2 Class Hierarchy

Class hierarchy for the default package is defined in the figure 5.6 Class hierarchy for the Data structure package is defined in the figure 5.7

5.4.3 Snapshot

5.4.4 Utile file Optimized implementation

Suffix the special kind of the line number at the time of including the file inside the main file(as depicted below).

`Eg. filename_extention.utile`

Table II: Default package class details - 1

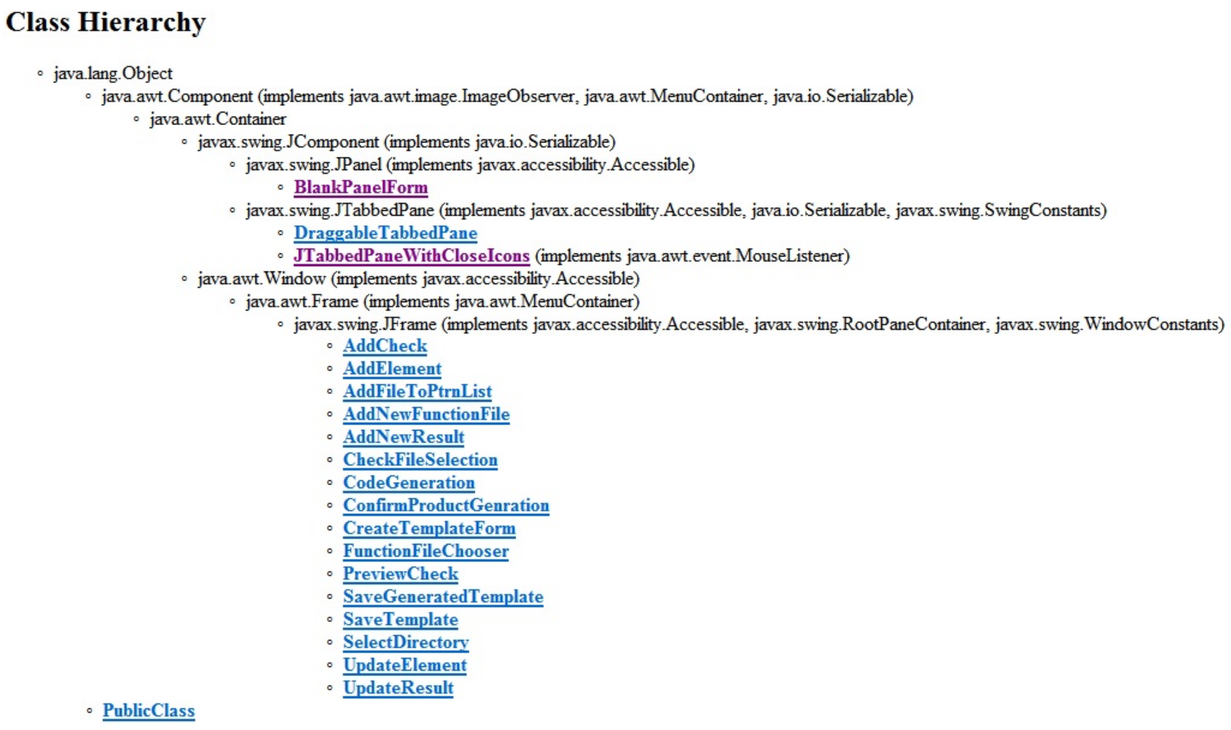
Package Default	
Class Summary	
AddCheck	Add check to the form.
AddElement	Add Elements to the Form and update UI.
AddFileToPtrnList	Add file type variable to the variable space in the
	application.
AddNewFunctionFile	User define function can be added to the tool at
	the run time by this class.
AddNewResult	New result can be added to the result area of
	the tool.
BlankPanelForm	This form internally consist the variable space
	and the result space in which the variables and
	results can be defined.
CheckFileSelection	Valid check file can be imported to reuse the
	saved checks.
CodeGeneration	Generated Code will be displayed in this Form.
ConfirmProductGenration	Generated product path will be given to the user
	after successfully creation of the product.
CreateTemplateForm	CreateTemplateForm contains the blank
	template in which various checks can be added
	accordig to user requirements.
DraggableTabbedPane	Dragable property will be set to all the tabbed
	pane.
FunctionFileChooser	valid function file will be selected using this file
	chooser.
PreviewCheck	Preview of the created check can be displayed at
	the runtime.

Table III: Default package class details - 2

Package Default	
Class Summary	
PublicClass	This calss contains the global variables that can
	be accessed across various classes.
SaveGeneratedTemplate	Save generated template with proper format for
	reusing it in future.
SaveTemplate	Save generated template with proper format for
	reusing it in future.
SelectDirectory	Form to select directory from the hard drive
UpdateElement	Update the Element values in the Form.
UpdateResult	Update the values of the already added result in
	the result field.

Table IV: DataStructure package class details

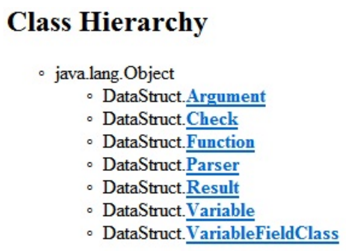
Package DataStructure	
Class Summary	
Argument	Parser to store arguments of the function
	file or the script file.
Check	Bean to store check details.
Function	Bean to store Function details.
Parser	This class perform various parsing
	operations required to get the data from file.
Result	Bean to store the Result Details from the
	check file of from Form.
Variable	Bean for Variable of the Check
VariableFieldClass	This java bean contains the getter and setter



Class Hierarchy

- java.lang.Object
  - java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - java.awt.Container
      - javax.swing.JComponent (implements java.io.Serializable)
        - javax.swing.JPanel (implements javax.accessibility.Accessible)
          - **BlankPanelForm**
        - javax.swing.JTabbedPane (implements javax.accessibility.Accessible, java.io.Serializable, javax.swing.SwingConstants)
          - **DraggableTabbedPane**
          - **JTabbedPaneWithCloseIcons** (implements java.awt.event.MouseListener)
      - java.awt.Window (implements javax.accessibility.Accessible)
        - java.awt.Frame (implements java.awt.MenuContainer)
          - javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
            - **AddCheck**
            - **AddElement**
            - **AddFileToPtrnList**
            - **AddNewFunctionFile**
            - **AddNewResult**
            - **CheckFileSelection**
            - **CodeGeneration**
            - **ConfirmProductGenration**
            - **CreateTemplateForm**
            - **FunctionFileChooser**
            - **PreviewCheck**
            - **SaveGeneratedTemplate**
            - **SaveTemplate**
            - **SelectDirectory**
            - **UpdateElement**
            - **UpdateResult**
  - **PublicClass**

Figure 5.6: Hierarchy of the default package.



Class Hierarchy

- java.lang.Object
  - DataStruct.**Argument**
  - DataStruct.**Check**
  - DataStruct.**Function**
  - DataStruct.**Parser**
  - DataStruct.**Result**
  - DataStruct.**Variable**
  - DataStruct.**VariableFieldClass**

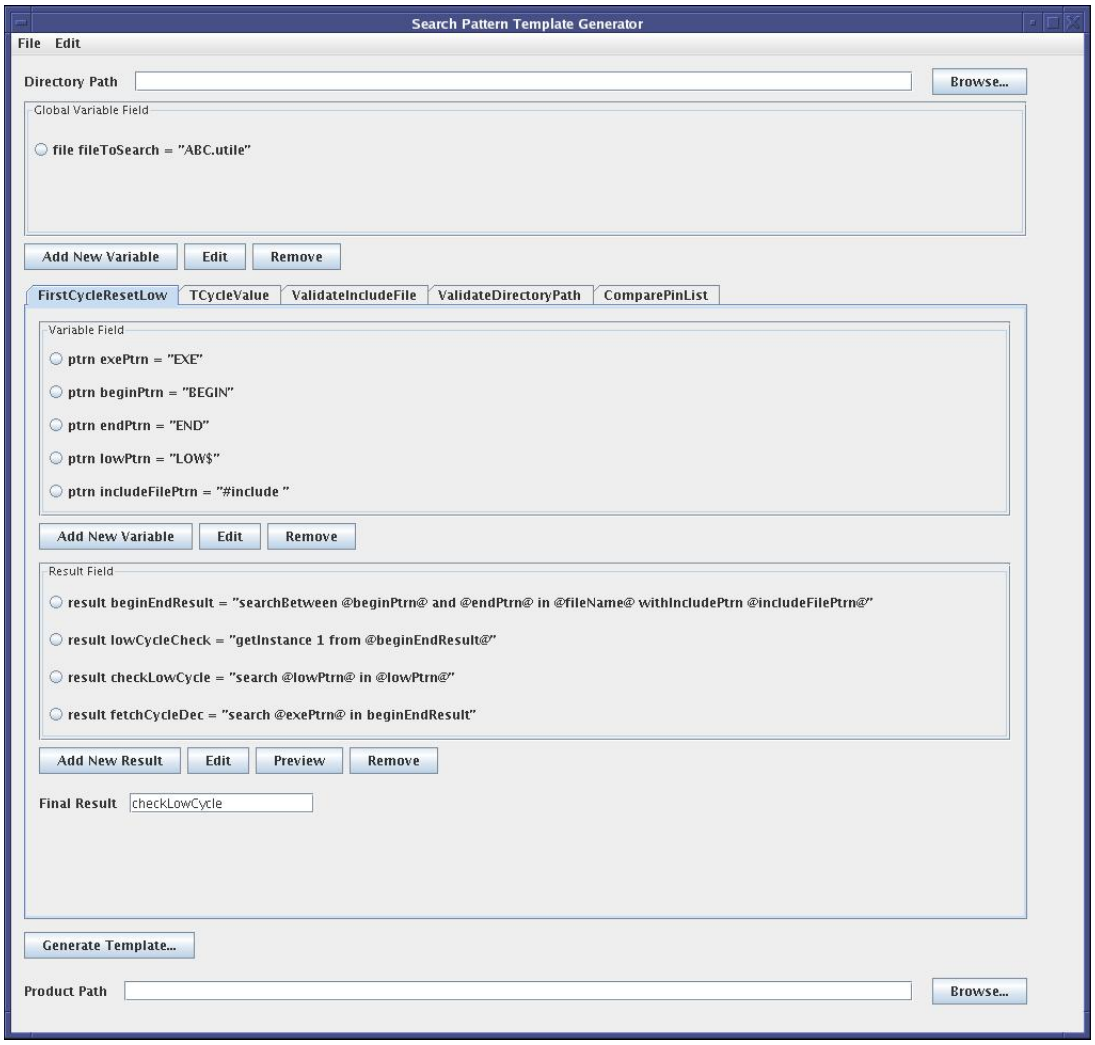
Figure 5.7: Hierarchy of the data Structure package.

```
1 Program  "fileName" "extension";

2_0 #include "relativePath/includeFile1" 2_1 content of file
"relativePath/includeFile1"

3 All required pin details.

4_0 #include "relativePath/includeFile2" 4_1 content of file
"relativePath/includeFile2"

5 For i:=0 to 100 6   Set pin value for i

7 Strobe for timing to set value for various cycle . . .

81 Strobe for value to set the value at each pin . . .
```

Include the file in the main file with the file name and the special line number allocation according to the include file name. If the extra pin found then it does not have to backtrack the complete included file to get the extra pin location. Depicted in the figure 5.11

5.5 Summary

Above implementation is performed according to the proposed algorithms and studying the related terminologies for the tool.



Directory Path

Global Variable Field

file fileToSearch = "ABC.utile"

Variable Field

ptrn exePtrn = "EXE"

ptrn beginPtrn = "BEGIN"

ptrn endPtrn = "END"

ptrn lowPtrn = "LOW$"

ptrn includeFilePtrn = "#include "

Result Field

result beginEndResult = "searchBetween @beginPtrn@ and @endPtrn@ in @fileName@ withIncludePtrn @includeFilePtrn@"

result lowCycleCheck = "getInstance 1 from @beginEndResult@"

result checkLowCycle = "search @lowPtrn@ in @lowPtrn@"

result fetchCycleDec = "search @exePtrn@ in beginEndResult"

Final Result: checkLowCycle

Product Path

Figure 5.8: Pattern Search engine tool GUI - with predefine checks

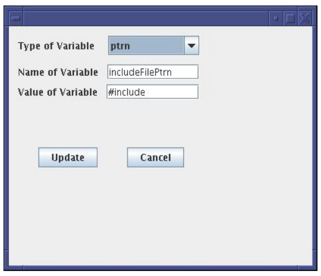
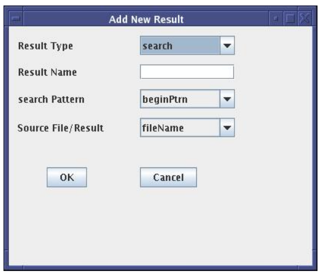

Type of Variable: ptrn
Name of Variable: includeFilePtrn
Value of Variable: #include

Figure 5.9: Pattern Search engine tool GUI - add new pattern


Add New Result

Result Type: search
Result Name:
search Pattern: beginPtrn
Source File/Result: fileName

Figure 5.10: Pattern Search engine tool GUI - add new result

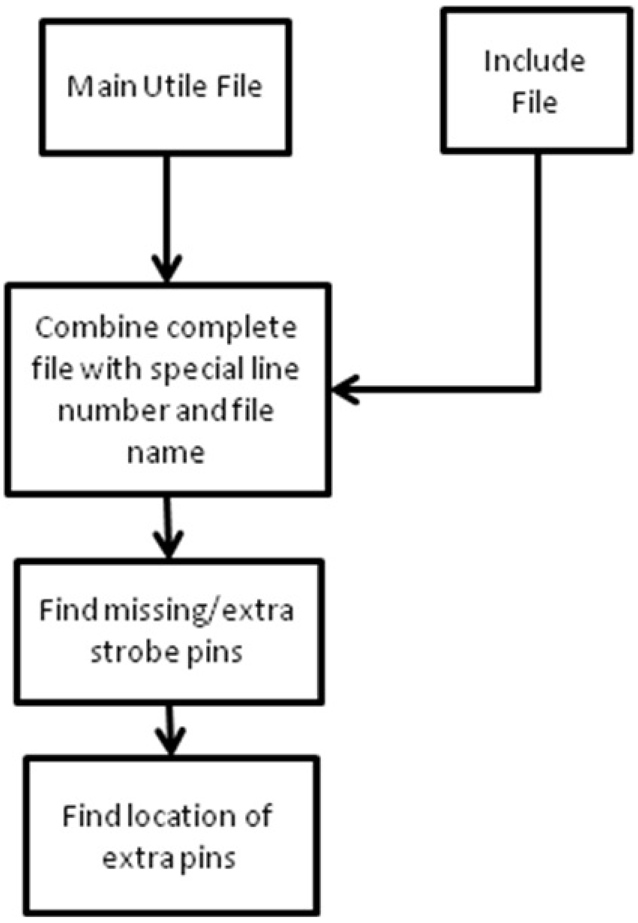

Main Utile File
Include File
Combine complete file with special line number and file name
Find missing/extra strobe pins
Find location of extra pins

Figure 5.11: Execution flow to get missing pin after optimization.

Chapter 6

Conclusion and Future Scope

6.1 Result Analysis

6.1.1 Execution time analysis of Utile file optimization

The execution time analysis is performed in java profiler tool. As shown in the figure 6.1 the class createTemplateForm requires the maximum time for execution. The second highest time required by the DataStructure.Parser class. To optimize this parser class various parsing techniques have been used to optimize the execution time. Structure of the output template has been modified to optimize the parser class.

6.1.2 Backtracking of the calling class

Backtracking of the calling class (6.2) helps to find out the all calling class of the particular class and the time taken by the class during the calling that class.

6.1.3 Heap allocation Analysis

Heap allocation analysis 6.3 depicts the heap size required by the application and the available heap size in JVM for that application. Due to use of the ArrayList instead of the Vector, when the heap will be full it will be expanded to the half of the actual size of the heap. If the vector is used for storing data then the heap size will be doubled when heap will be full. The figure-7 depicts the heap allocation during the execution of the application.

6.1.4 Analysis

The garbage collection analysis 6.4 shows the invocation on the garbage collection during the execution of the application.

6.1.5 Thread Analysis

The thread analysis result 6.5 depicts the number of threads at the some instance of the time and the number of the classes at that instance of the time.

6.1.6 Optimized results

Following diagram (6.6) depicts that due to making change in the classes taking more time for execution the execution time will be reduces.

Optimized parser class

As shown in the 6.7, Parser class requires lesser number of the bytes for the execution and the less objects will be created at the time of the execution which reduces the time to execute.

6.1.7 Execution time analysis in Utile File

The time required for the backtracking will be reduces which decrease the time to required the get the missing pins from the utile file.The optimized results are depicted in the table I II [LSF is used to find the execution time]

Table I: Analysis for 5 include file in the main file

(for 5 include file in the main file)	execution time
Time before the optimization	515 ms
Time after the optimization	468 ms

Table II: Analysis for 7 include file in the main file

(for 7 include file in the main file)	execution time
Time before the optimization	749 ms
Time after the optimization	612 ms

6.2 Conclusion

In this dissertation, we proposed to optimized the following tool by analyzing and make proper degradation to it. Also design Pattern Search engine from scratch with minimum time complexity and with user friendly environment.

- Component Descriptor Language Utility

- Timing Power Product tool

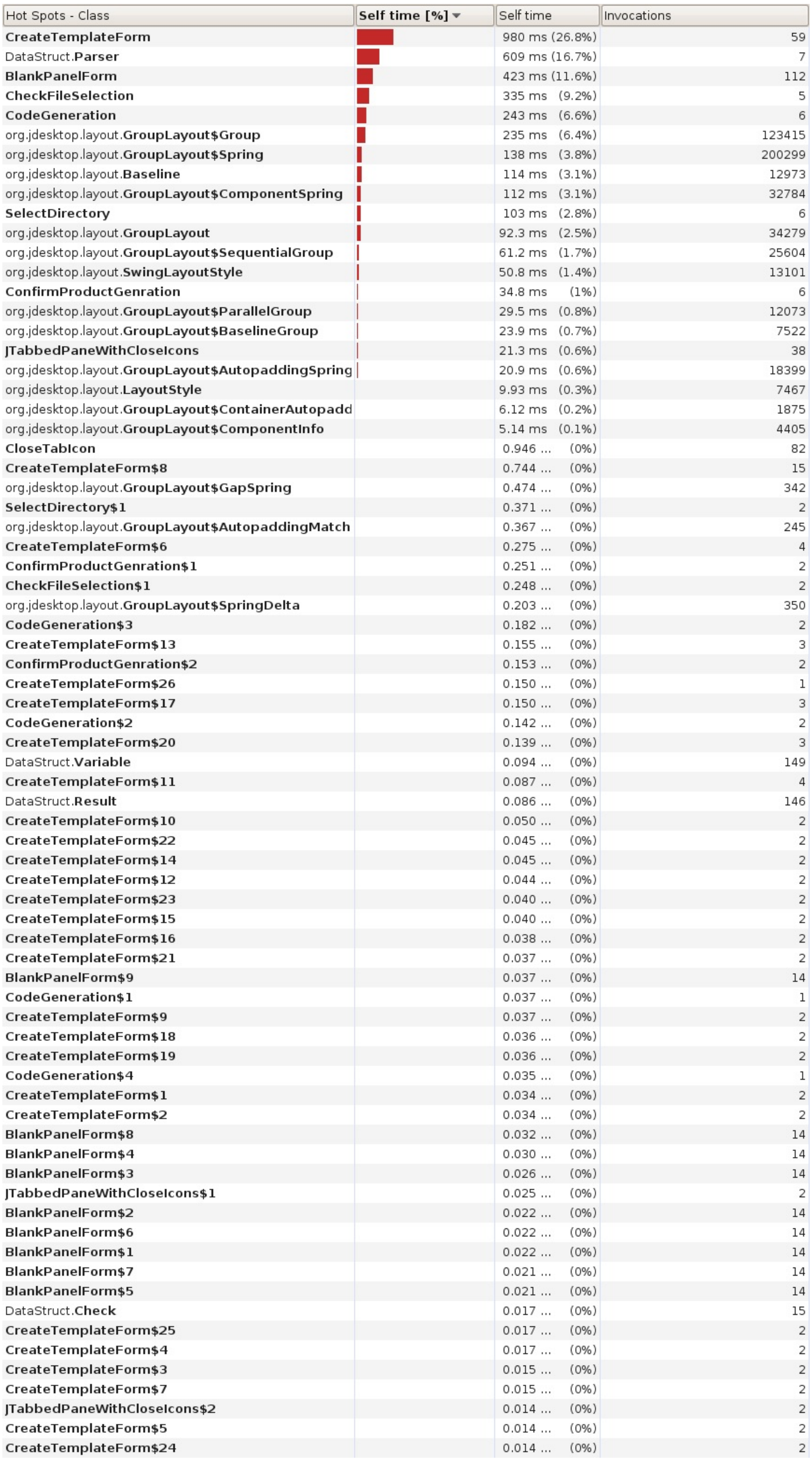
- CDL Verilog comparison tool optimization

- Pattern Search engine

From the results we can see that the tools get optimized with time complexity and Component Descriptor Language Utility had drastically decrease it's space complexity also.Pattern Search engine tool is designed with user friendly and dynamic environment for further improvements.

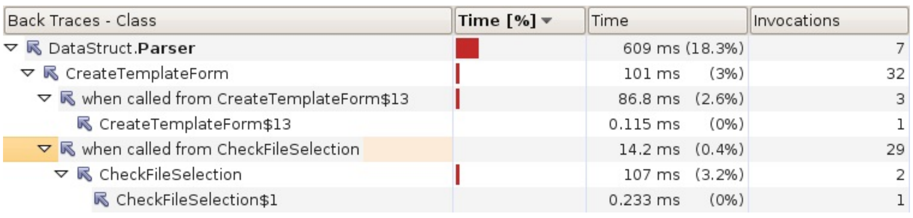
6.3 Future Scope

- Component Descriptor Language Utility
  - The tool can further optimized using the changing the structure of the data structured.
  - Parsing of the file can be performed by the Shell programming decrease the time complexity of the tool.
- Timing Power Product tool
  - This tool can further optimized by parallel processing the timing, power and leakage libraries to get the detailed output terms.
- CDL Verilog comparison tool optimization
  - Optimize the time to generate pin list from the CDL can further optimize the time complexity, because it is the next higher time consuming process after the verilog to pin generation.
- Pattern Search engine
  - Pattern search engine is dynamic tool which can be used for any new required search requirements.
  - It can further implemented for any languages instead of the shell programming only.



Hot Spots - Class	Self time [%]	Self time	Invocations
CreateTemplateForm		980 ms (26.8%)	59
DataStruct.**Parser**		609 ms (16.7%)	7
BlankPanelForm		423 ms (11.6%)	112
CheckFileSelection		335 ms (9.2%)	5
CodeGeneration		243 ms (6.6%)	6
org.jdesktop.layout.**GroupLayout$Group**		235 ms (6.4%)	123415
org.jdesktop.layout.**GroupLayout$Spring**		138 ms (3.8%)	200299
org.jdesktop.layout.**Baseline**		114 ms (3.1%)	12973
org.jdesktop.layout.**GroupLayout$ComponentSpring**		112 ms (3.1%)	32784
SelectDirectory		103 ms (2.8%)	6
org.jdesktop.layout.**GroupLayout**		92.3 ms (2.5%)	34279
org.jdesktop.layout.**GroupLayout$SequentialGroup**		61.2 ms (1.7%)	25604
org.jdesktop.layout.**SwingLayoutStyle**		50.8 ms (1.4%)	13101
ConfirmProductGenration		34.8 ms (1%)	6
org.jdesktop.layout.**GroupLayout$ParallelGroup**		29.5 ms (0.8%)	12073
org.jdesktop.layout.**GroupLayout$BaselineGroup**		23.9 ms (0.7%)	7522
JTabbedPaneWithCloseIcons		21.3 ms (0.6%)	38
org.jdesktop.layout.**GroupLayout$AutopaddingSpring**		20.9 ms (0.6%)	18399
org.jdesktop.layout.**LayoutStyle**		9.93 ms (0.3%)	7467
org.jdesktop.layout.**GroupLayout$ContainerAutopadd**		6.12 ms (0.2%)	1875
org.jdesktop.layout.**GroupLayout$ComponentInfo**		5.14 ms (0.1%)	4405
CloseTabIcon		0.946 ... (0%)	82
CreateTemplateForm$8		0.744 ... (0%)	15
org.jdesktop.layout.**GroupLayout$GapSpring**		0.474 ... (0%)	342
SelectDirectory$1		0.371 ... (0%)	2
org.jdesktop.layout.**GroupLayout$AutopaddingMatch**		0.367 ... (0%)	245
CreateTemplateForm$6		0.275 ... (0%)	4
ConfirmProductGenration$1		0.251 ... (0%)	2
CheckFileSelection$1		0.248 ... (0%)	2
org.jdesktop.layout.**GroupLayout$SpringDelta**		0.203 ... (0%)	350
CodeGeneration$3		0.182 ... (0%)	2
CreateTemplateForm$13		0.155 ... (0%)	3
ConfirmProductGenration$2		0.153 ... (0%)	2
CreateTemplateForm$26		0.150 ... (0%)	1
CreateTemplateForm$17		0.150 ... (0%)	3
CodeGeneration$2		0.142 ... (0%)	2
CreateTemplateForm$20		0.139 ... (0%)	3
DataStruct.**Variable**		0.094 ... (0%)	149
CreateTemplateForm$11		0.087 ... (0%)	4
DataStruct.**Result**		0.086 ... (0%)	146
CreateTemplateForm$10		0.050 ... (0%)	2
CreateTemplateForm$22		0.045 ... (0%)	2
CreateTemplateForm$14		0.045 ... (0%)	2
CreateTemplateForm$12		0.044 ... (0%)	2
CreateTemplateForm$23		0.040 ... (0%)	2
CreateTemplateForm$15		0.040 ... (0%)	2
CreateTemplateForm$16		0.038 ... (0%)	2
CreateTemplateForm$21		0.037 ... (0%)	2
BlankPanelForm$9		0.037 ... (0%)	14
CodeGeneration$1		0.037 ... (0%)	1
CreateTemplateForm$9		0.037 ... (0%)	2
CreateTemplateForm$18		0.036 ... (0%)	2
CreateTemplateForm$19		0.036 ... (0%)	2
CodeGeneration$4		0.035 ... (0%)	1
CreateTemplateForm$1		0.034 ... (0%)	2
CreateTemplateForm$2		0.034 ... (0%)	2
BlankPanelForm$8		0.032 ... (0%)	14
BlankPanelForm$4		0.030 ... (0%)	14
BlankPanelForm$3		0.026 ... (0%)	14
JTabbedPaneWithCloseIcons$1		0.025 ... (0%)	2
BlankPanelForm$2		0.022 ... (0%)	14
BlankPanelForm$6		0.022 ... (0%)	14
BlankPanelForm$1		0.022 ... (0%)	14
BlankPanelForm$7		0.021 ... (0%)	14
BlankPanelForm$5		0.021 ... (0%)	14
DataStruct.**Check**		0.017 ... (0%)	15
CreateTemplateForm$25		0.017 ... (0%)	2
CreateTemplateForm$4		0.017 ... (0%)	2
CreateTemplateForm$3		0.015 ... (0%)	2
CreateTemplateForm$7		0.015 ... (0%)	2
JTabbedPaneWithCloseIcons$2		0.014 ... (0%)	2
CreateTemplateForm$5		0.014 ... (0%)	2
CreateTemplateForm$24		0.014 ... (0%)	2

Figure 6.1: Execution time of the complete application



Back Traces - Class	Time [%]	Time	Invocations
DataStruct.**Parser**		609 ms (18.3%)	7
CreateTemplateForm		101 ms (3%)	32
when called from CreateTemplateForm$13		86.8 ms (2.6%)	3
CreateTemplateForm$13		0.115 ms (0%)	1
when called from CheckFileSelection		14.2 ms (0.4%)	29
CheckFileSelection		107 ms (3.2%)	2
CheckFileSelection$1		0.233 ms (0%)	1

Figure 6.2: Backtracking of the calling class

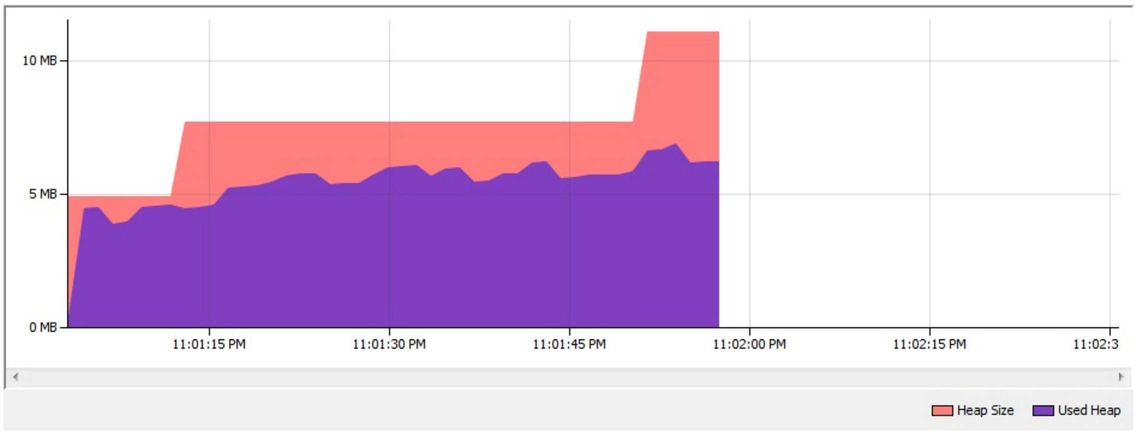


Figure 6.3: Heap allocation Analysis

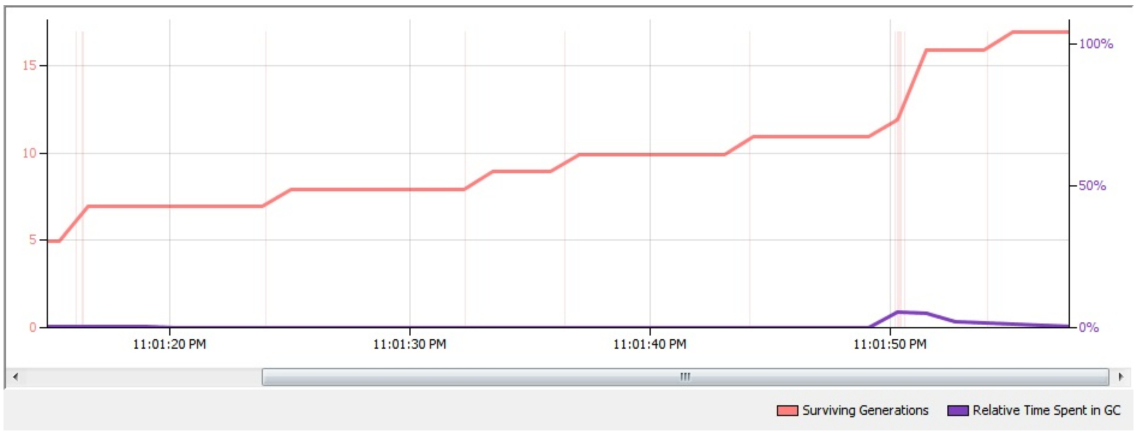


Figure 6.4: Garbage collection Analysis

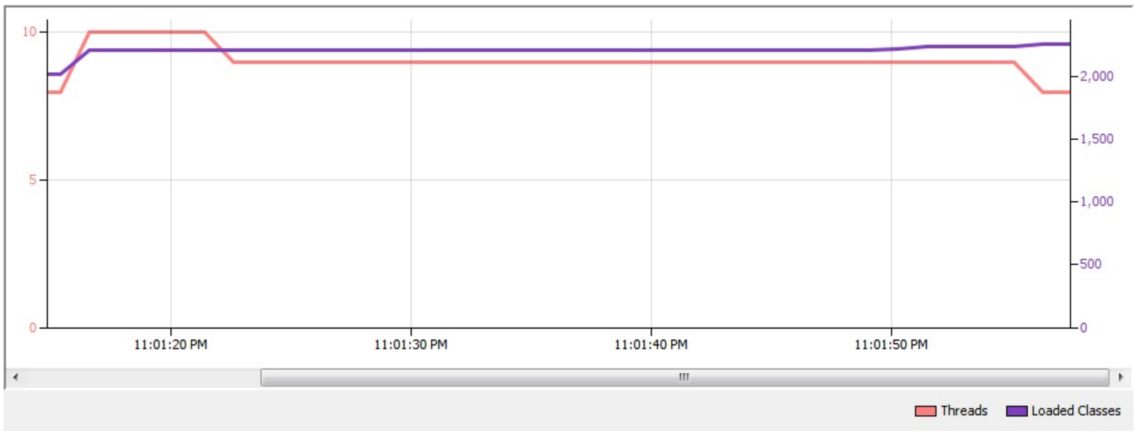
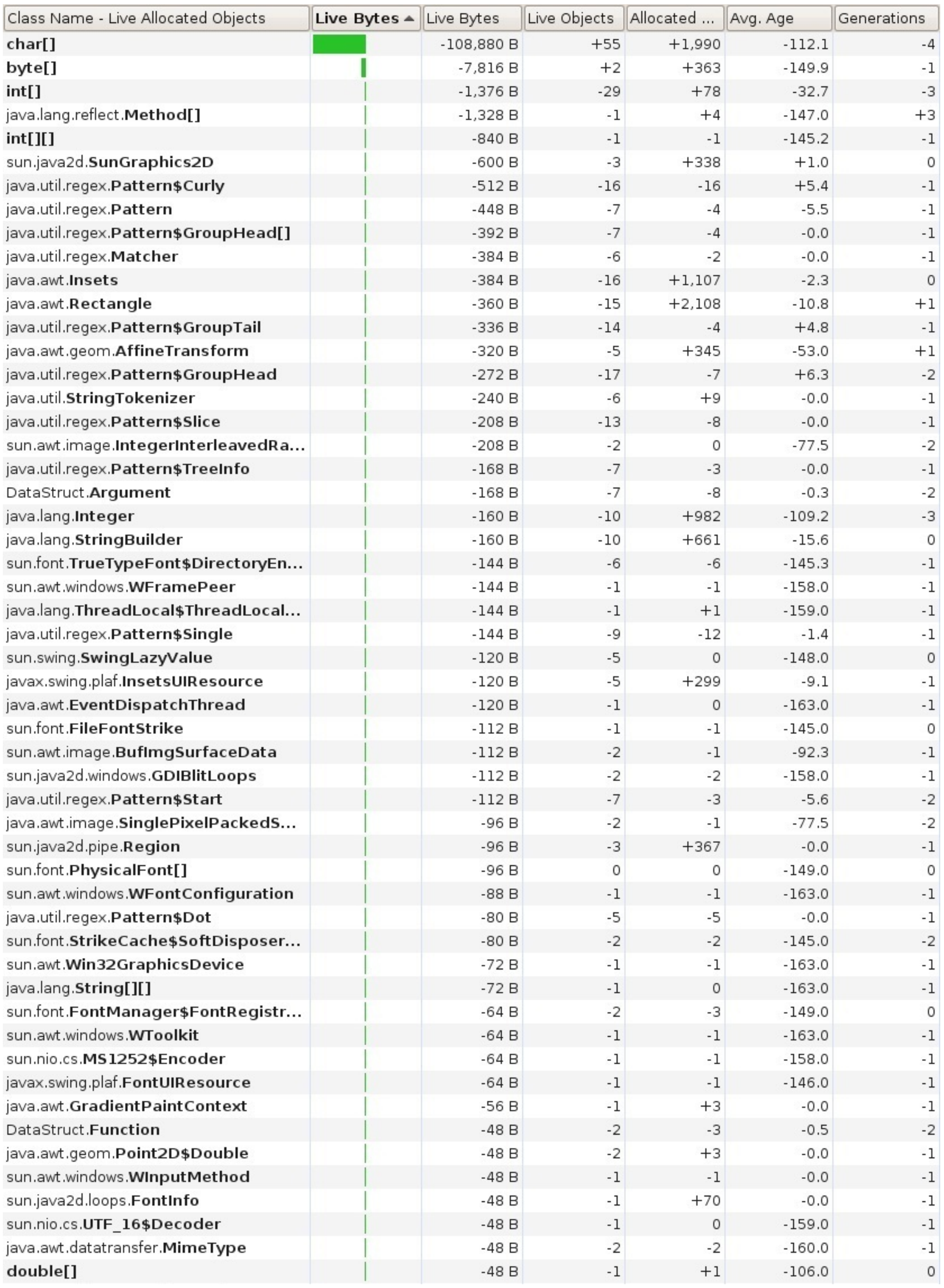
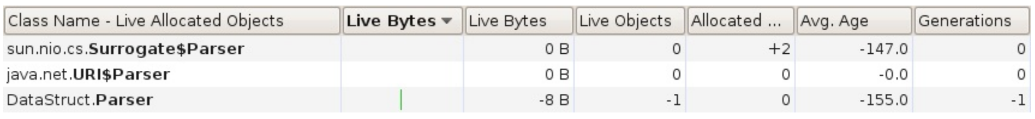


Figure 6.5: Thread Analysis



Class Name - Live Allocated Objects	Live Bytes ▲	Live Bytes	Live Objects	Allocated ...	Avg. Age	Generations
char[]		-108,880 B	+55	+1,990	-112.1	-4
byte[]		-7,816 B	+2	+363	-149.9	-1
int[]		-1,376 B	-29	+78	-32.7	-3
java.lang.reflect.Method[]		-1,328 B	-1	+4	-147.0	+3
int[][]		-840 B	-1	-1	-145.2	-1
sun.java2d.SunGraphics2D		-600 B	-3	+338	+1.0	0
java.util.regex.Pattern$Curly		-512 B	-16	-16	+5.4	-1
java.util.regex.Pattern		-448 B	-7	-4	-5.5	-1
java.util.regex.Pattern$GroupHead[]		-392 B	-7	-4	-0.0	-1
java.util.regex.Matcher		-384 B	-6	-2	-0.0	-1
java.awt.Insets		-384 B	-16	+1,107	-2.3	0
java.awt.Rectangle		-360 B	-15	+2,108	-10.8	+1
java.util.regex.Pattern$GroupTail		-336 B	-14	-4	+4.8	-1
java.awt.geom.AffineTransform		-320 B	-5	+345	-53.0	+1
java.util.regex.Pattern$GroupHead		-272 B	-17	-7	+6.3	-2
java.util.StringTokenizer		-240 B	-6	+9	-0.0	-1
java.util.regex.Pattern$Slice		-208 B	-13	-8	-0.0	-1
sun.awt.image.IntegerInterleavedRa...		-208 B	-2	0	-77.5	-2
java.util.regex.Pattern$TreeInfo		-168 B	-7	-3	-0.0	-1
DataStruct.Argument		-168 B	-7	-8	-0.3	-2
java.lang.Integer		-160 B	-10	+982	-109.2	-3
java.lang.StringBuilder		-160 B	-10	+661	-15.6	0
sun.font.TrueTypeFont$DirectoryEn...		-144 B	-6	-6	-145.3	-1
sun.awt.windows.WFramePeer		-144 B	-1	-1	-158.0	-1
java.lang.ThreadLocal$ThreadLocal...		-144 B	-1	+1	-159.0	-1
java.util.regex.Pattern$Single		-144 B	-9	-12	-1.4	-1
sun.swing.SwingLazyValue		-120 B	-5	0	-148.0	0
javax.swing.plaf.InsetsUIResource		-120 B	-5	+299	-9.1	-1
java.awt.EventDispatchThread		-120 B	-1	0	-163.0	-1
sun.font.FileFontStrike		-112 B	-1	-1	-145.0	0
sun.awt.image.BufImgSurfaceData		-112 B	-2	-1	-92.3	-1
sun.java2d.windows.GDIBlitLoops		-112 B	-2	-2	-158.0	-1
java.util.regex.Pattern$Start		-112 B	-7	-3	-5.6	-2
java.awt.image.SinglePixelPackedS...		-96 B	-2	-1	-77.5	-2
sun.java2d.pipe.Region		-96 B	-3	+367	-0.0	-1
sun.font.PhysicalFont[]		-96 B	0	0	-149.0	0
sun.awt.windows.WFontConfiguration		-88 B	-1	-1	-163.0	-1
java.util.regex.Pattern$Dot		-80 B	-5	-5	-0.0	-1
sun.font.StrikeCache$SoftDisposer...		-80 B	-2	-2	-145.0	-2
sun.awt.Win32GraphicsDevice		-72 B	-1	-1	-163.0	-1
java.lang.String[][]		-72 B	-1	0	-163.0	-1
sun.font.FontManager$FontRegistr...		-64 B	-2	-3	-149.0	0
sun.awt.windows.WToolkit		-64 B	-1	-1	-163.0	-1
sun.nio.cs.MS1252$Encoder		-64 B	-1	-1	-158.0	-1
javax.swing.plaf.FontUIResource		-64 B	-1	-1	-146.0	-1
java.awt.GradientPaintContext		-56 B	-1	+3	-0.0	-1
DataStruct.Function		-48 B	-2	-3	-0.5	-2
java.awt.geom.Point2D$Double		-48 B	-2	+3	-0.0	-1
sun.awt.windows.WInputMethod		-48 B	-1	-1	-0.0	-1
sun.java2d.loops.FontInfo		-48 B	-1	+70	-0.0	-1
sun.nio.cs.UTF_16$Decoder		-48 B	-1	0	-159.0	-1
java.awt.datatransfer.MimeType		-48 B	-2	-2	-160.0	-1
double[]		-48 B	-1	+1	-106.0	0

Figure 6.6: Difference with the previous results



Class Name - Live Allocated Objects	Live Bytes ▼	Live Bytes	Live Objects	Allocated ...	Avg. Age	Generations
sun.nio.cs.Surrogate$Parser		0 B	0	+2	-147.0	0
java.net.URI$Parser		0 B	0	0	-0.0	0
DataStruct.Parser		-8 B	-1	0	-155.0	-1

Figure 6.7: Parser class result analysis

References

[1] Anders P. Ravn Hans Sndergaard, Bent Thomsen. A ravenscar-java profile implementation. Paris, France, October 2006. Proceedings of the 4th international workshop on Java technologies for real-time and embedded systems.

[2] Steve King Jagun Kwon, Andy Wellings. Predictable memory utilization in the ravenscar-java profile. Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03), 2003.

[3] J.S. MOORE R.S. BOYER. A fast string searching algorithm. pages 762–772. Communications of the ACM, 1977.

[4] Tony Sintes. Find out the difference between vector and arraylist. http://www.javaworld.com/javaworld/javaqa/2001-06/03-qa-0622-vector.html., June 2001.

[5] ST Microelectronics, Greater NOIDA. *The component descriptor language documentation.* Internal document.

[6] ST Microelectronics, Greater NOIDA. *Optimizing Tactics.* Internal document.

[7] ST Microelectronics, Greater NOIDA. *Profiling in Java.* Internal document.

[8] ST Microelectronics, Greater NOIDA. *Usage of LSF to measure the execution time.* Internal document.

[9] ST Microelectronics, Greater NOIDA. *Utile File Structure.* Internal document.

[10] ST Microelectronics, Greater NOIDA. *Verilog file Format.* Internal document.

[11] Bill Venners. Javas garbage-collected heap. http://www.javaworld.com/javaworld/jw-08-1996/jw-08-gc.html, Aug 1996.

Index