Mini Project Report

Integrated Framework for Analysis and Visualization of Embedded Platforms

Submitted by
Ashish Gupta (98131)
Manan Sanghi (98140)

Under Supervision of:
Prof. M. Balakrishnan
Prof. Anshul Kumar

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY DELHI

November 2000

Acknowledgements

We are pleased to acknowledge Prof. M. Balakrishnan and Prof. Anshul Kumar for their invaluable guidance during the course of this project work.

We extend our sincere thanks to Mr. Vishal Bhatt who continuously helped us throughout the project and without his guidance, this project would have been an uphill task.

We are also grateful to other members of the ASSET team who co-operated with us regarding some issues.

We would also like to thank 'Software Farm' (www.swfm.com) for writing the very useful Mica Graphics Framework Toolkit for Java under the Open Source banner which greatly helped us in writing the visualization part.

Last but not the least, Mr. Ashish Shah supervisor of FPGA Lab also co-operated with us nicely for the smooth development of this project.

November 2000

Ashish Gupta (98131)

Manan Sanghi (98140)

Contents

Chapter

1

Introduction

Overview

This report discusses the result of the work done in development of "Integrated Framework for Analysis and Visualization for Embedded Systems" on Java Platform. It is a part of the ASSET (Automated SynthesiS of Embedded sysTems) project going in Computer Science Department, IIT Delhi and aims at the development of an application framework for providing a common platform for facilitating the use of methodological approach developed by the ASSET team and integration of various tools developed during the execution of the project.

Background and Motivation

Embedded Systems can be found in a large variety of applications today like image processing, networking and wireless communication. They essentially comprise of a processor and some hardware built around it. The software is used for achieving fast turn around times while the hardware is used to speedup critical portions of the system. Till now, the design of Embedded Systems was largely carried out in an ad-hoc manner. With dramatically decreasing silicon costs, it is now possible to implement very complex systems on a single chip. With over 100 million transistors per IC expected by the turn of the century, the expected complexity of such systems will require a rigorous design

methodology with the development of supporting design tools. This is precisely the focus of ASSET project.

The ASSET project aims at the development of a design methodology for embedded systems for vision/image processing applications. The idea is that given a system specification, by following the methodology and with the help of the tools developed to support it, the user will be able to synthesize a system that meets his constraints.

A tool was required to integrate all the design tools discussed above along with the capability to perform the same functions manually. Analysis and Visualization of the target platform was also required to know its performance. This project deals with the development of such a tool which will assist in the implementation of the above methodology.

Objective

The final goal of the project was twofold.

1. An Integrated Framework was required for interaction with the various tools (like Software/Hardware Estimation, Partitioning, Synthesis tools etc.) with the platform specification being done in the application itself.

2. Based on the final platform configuration and bindings, an Analysis and Visualization framework was required for getting performance metrics of the system and for visualization of the analysis results and the target platform.

Along with above main goals , capability to design the target platform manually was also desired.

Methodology

To implement the above goals , the following methodology needs to be followed :

1. Specifying the Application and various components of the Architecture.

2. Specifying the bindings between the tasks and the resources either manually or by the design tools.

3. Specifying the port interconnections between the resources.

4. Analysis : Extracting the data required for analysis and the doing the analysis.

5. Synthesis using the synthesis tools developed by the ASSET team.

Chapter
2

Tool Description

User Interface

The tool is very user friendly and intuitive and uses a GUI interface implemented in JAVA to communicate wih the user. Various features are self – explanatory.

Forms are easy to fill in and components can be added , removed and updated very easily through a single dialog box. The application includes tool-tip hints to give a brief description of the particular input field.

List boxes are used to display all the components at once so that user can see all the components of a particular type at once. One can just select the component and modify and remove the component.

Features

1. Intuitive interface

2. Clean separation of various components to facilitate easy modification and revision.

3. All the configuration data is maintained in a separate file to facilitate easy modification

If the tool needs to be upgraded to include more features, for instance if it is desired to include more elaborate specification of FPGAs then the separation of the data file containing all the data of the

specification will prove to be extremely useful. Also maintaining a separate file for the purpose helps in centralisation of the data for easy understanding of the source code and the implementation methodology.

4. Analysis Component is kept modular to facilitate multiple analysis models.

Analysis models may need upgradation from time to time depending upon the varying nature of the systems the tool may be used for. To facilitate easy upgradation of analysis model great care has been undertaken

- All the data required for analysis is kept in a separate file.

- The data is collected by a '*data collator'* which collects data from the various data sources (application specification, target platform specification, SUIF annotations, user etc.). As the data is generated by the other tools and stored in the SUIF annotations, **only this data collator needs to be changed without disturbing the rest of the analysis.**

- The '*analyzer*' i.e. the actual analysis model is clearly separated from the other analysis components (like the data collator, the visualizer etc.). So more sophisticated analysis models only needs the modification of the ‘analyzer’ without being concerned with the rest of the analysis.

- Visualization of the analysis result is also made modular. The *'visualizer'* reads the analysis result kept in a separate file and generate the desired visualizations of these results (currently it generates pie charts).

5. Quick and easy saving and loading of System configuration.

Since the specification of the Application and the Target Platform can be very intricate, a option for saving the current configuration is a very much desired. All the configuration data (including the binding

and the interconnection information) could be easily stored in different files. So one can work on multiple configurations simultaneously. In fact, it is made as convenient as saving, loading and editing a text file from a standard text-editor.

6. Option of 2D or 3D pie chart for viewing analysis results.

7. Visualizer features preset layout and draggable components to provide flexibility to the user for choosing between different layouts or designing his own.

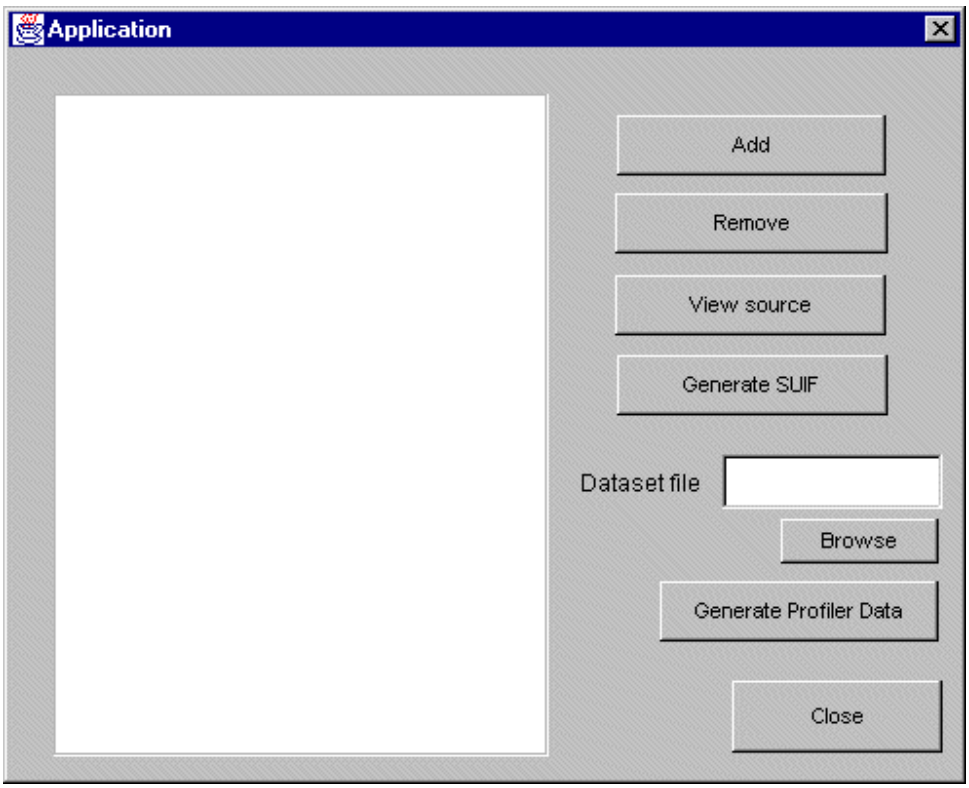
8. Includes an internal Text Editor for easy viewing and editing of application(C files), Processor description file etc. from within the tool.

9. All the Data Structures for storing configuration data is maintained in a separate file to facilitate easy modification.

Specification

First of all the specification of the target platform must be specified completely. It consists of :

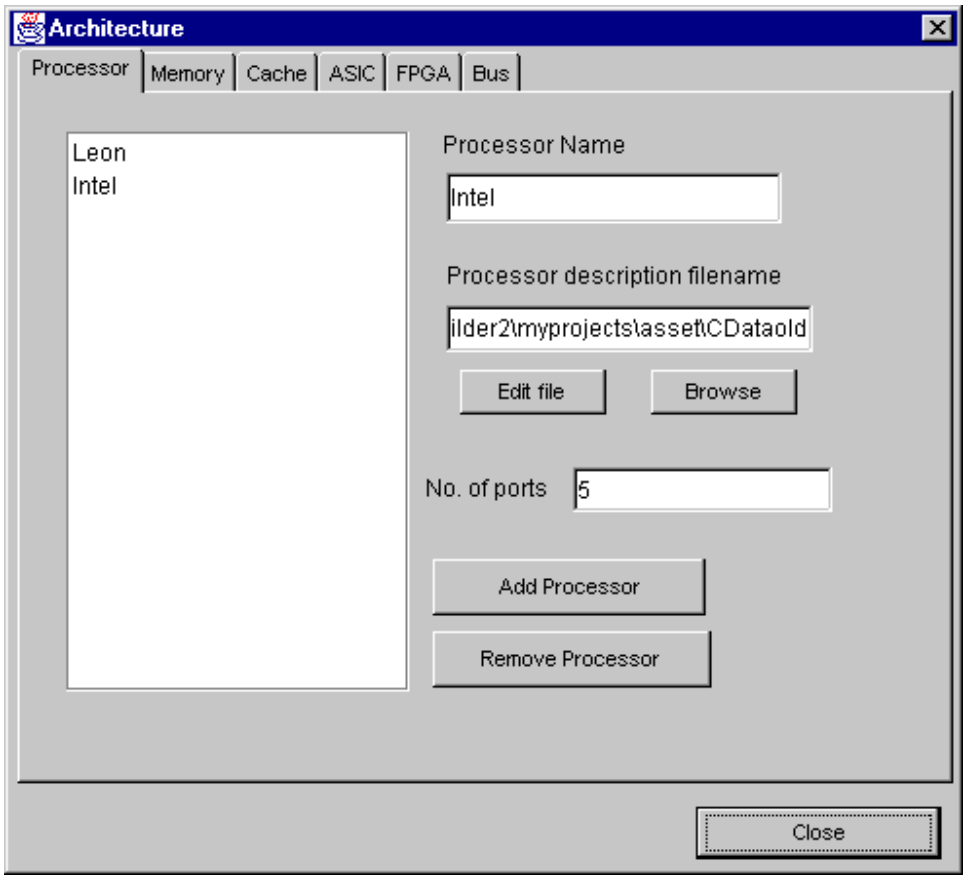
1. Application Specification



In the application specification , the C files which constitute the target application are specified. After specifying the C files, one can :

- View the source with the internal text editor
- Generate SUIF files for the corresponding C file using the ctosuif tool.
- Generate profiler data for each of the C files which may be required by estimation and partitioning tools etc. A Dataset for each C file can be specified which is needed by the profiler.

2. Architecture Components Specification

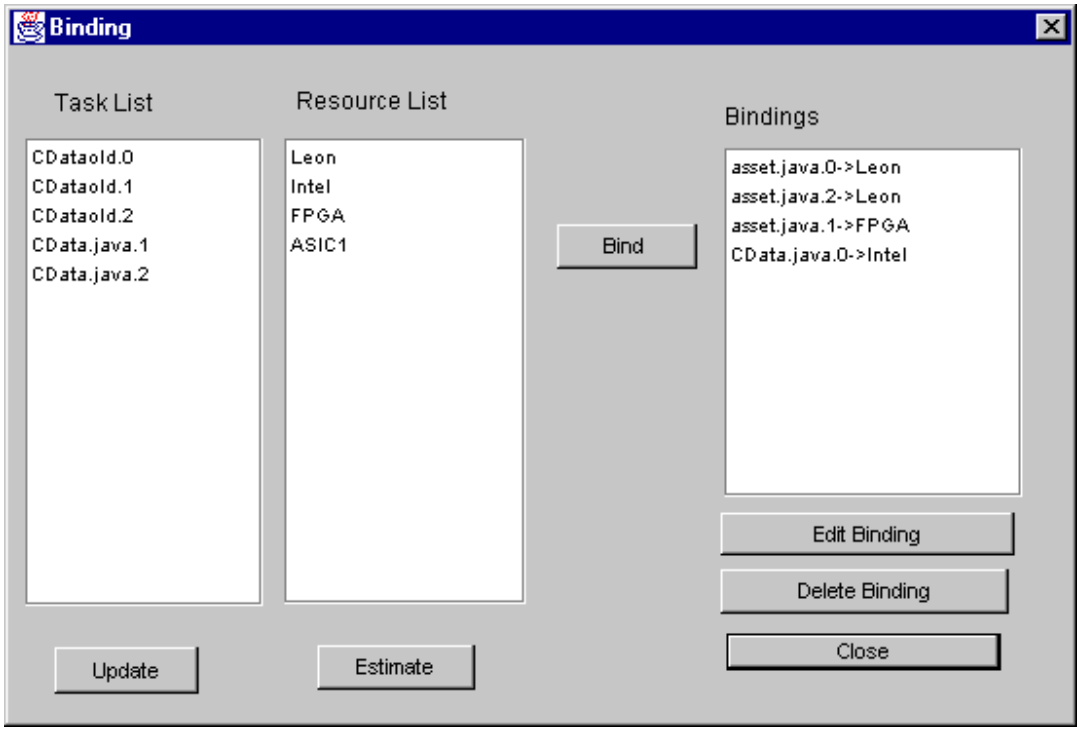


All the components in the target platform are specified in the above dialog box. The various types of components are :

- Processor
- FPGA
- ASIC
- Memory
- Cache
- Bus

See the Component Specification table for the parameters required for each component.

3. Binding Specification

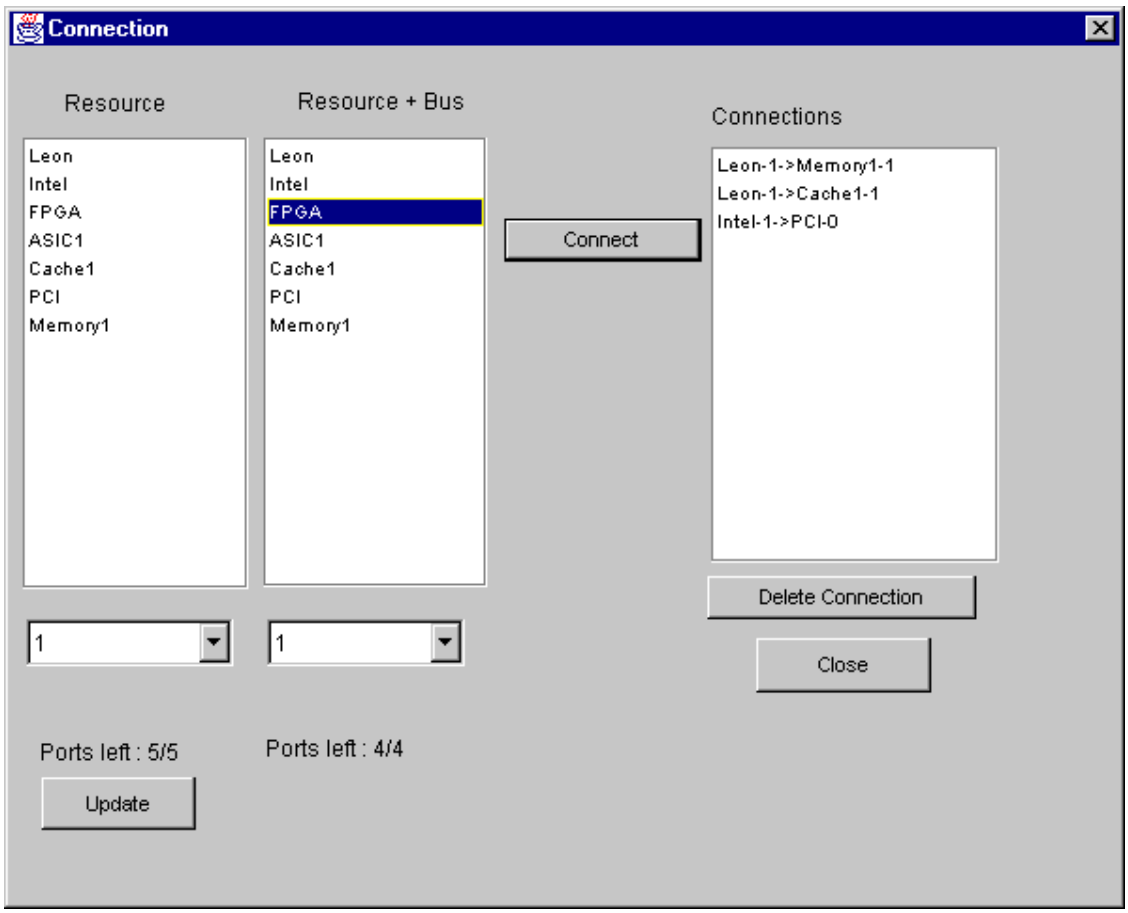


Here, the bindings between the various tasks and resources are specified manually. The interface is intuitive to facilitate manual binding. The two list boxes on the left display the tasks and resources respectively. To bind a task, select a task and corresponding resource from the list boxes and click Bind. The bound pairs are shown in the list box on the right. After pressing Bind , the *channel – port binding Dialog* pops up to specify the bindings between the various channels of the task and the ports of the resource.

From the above dialog, provision is also there to directly estimate the performance of the task on the chosen resource. Tool automatically calls the Software or Hardware Estimator depending on the chosen resource.

After specifying a binding , one can re-edit the channel – port bindings or delete the binding.

4. Interconnection Specification



The interconnections between the various resources can be specified here. Here the port numbers for each connection can also be specified.

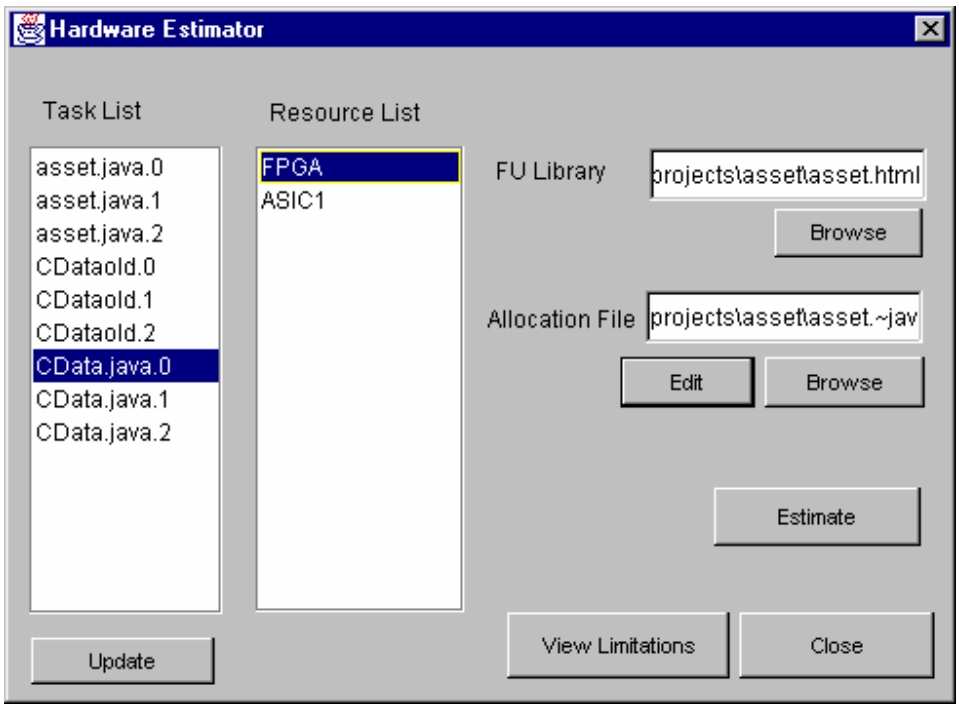
The interface and procedure to connect two resources is similar to Binding discussed previously. Whenever the user clicks on a resource in one of the resource lists, information regarding its total and remaining number of ports is also displayed at the bottom.

Calling the Design tools

After specifying the various components one can also use the tools to automate the task of binding etc. The tools currently supported are:

Note : Currently , the actual tools are not called since some tools are not ready. However the tool calling interface ad paramter passing was tested with the help of a dummy tool which was written Visual C++.

1. Hardware Estimator

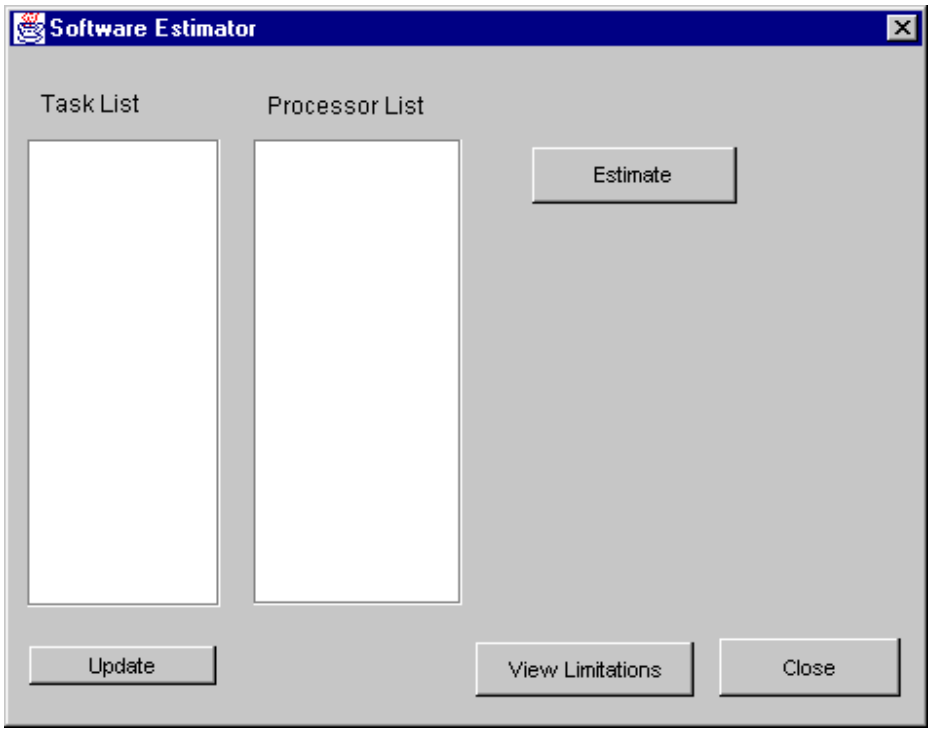


Here, user specifies the tasks and the resource for the estimator tool. Only FPGAs and ASICs are displayed in the resource list. Functional Unit library and Allocation File library also need to be specified before calling the HW Estimator.

One can also view the current limitations of the HW Estimator by clicking on the 'View Limitations' button.

On clicking the 'Estimate' button , the HW Estimator tool is called.

2. Software Estimator



This option is similar to the HW Estimator option except that only processors are displayed in the resource list.

3. Partitioner

On choosing this option, the partitioner tool is called.

Analysis

Before doing the Analysis, the following steps are to be followed.

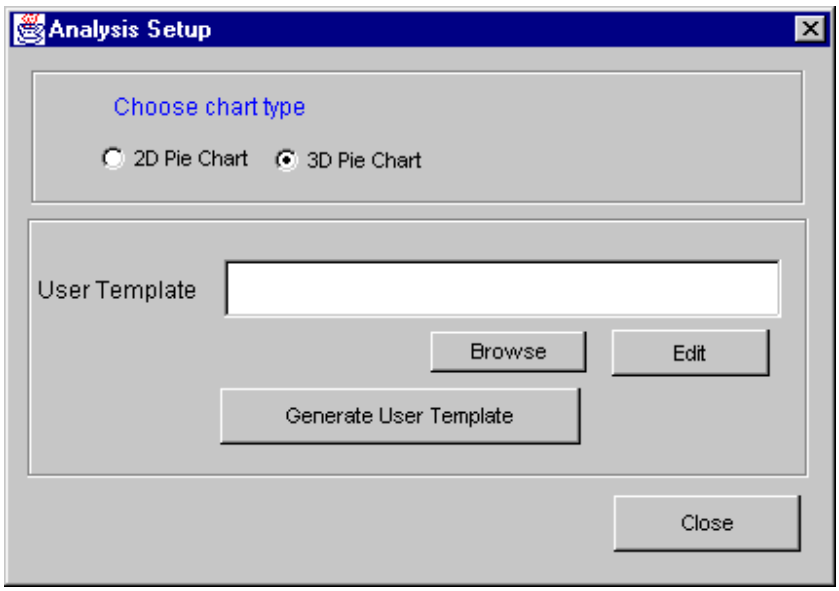
1. Extract Data

Before proceeding with the analysis, the data needed by the Analyzer must be extracted from the various sources as required. The exact procedure of extracting the data is specified in the FillData() function in the Class CAnalysis_Data. One only needs to modify this function and recompile this file to

change the data extraction process. After clicking this option, if the data is extracted successfully, the tool is ready for analysis.

Note: Some data may also come directly from the user. To facilitate this, the concept of User Template File is introduced. See Appendix B for more details.

2. Analysis Setup



Here one can choose the option of displaying 2D or 3D pie chart for the analysis results.

User Template Generation

Here, the user template file for the data given by the user manually is specified. One can generate a blank template by clicking on 'Generate User Template' button.

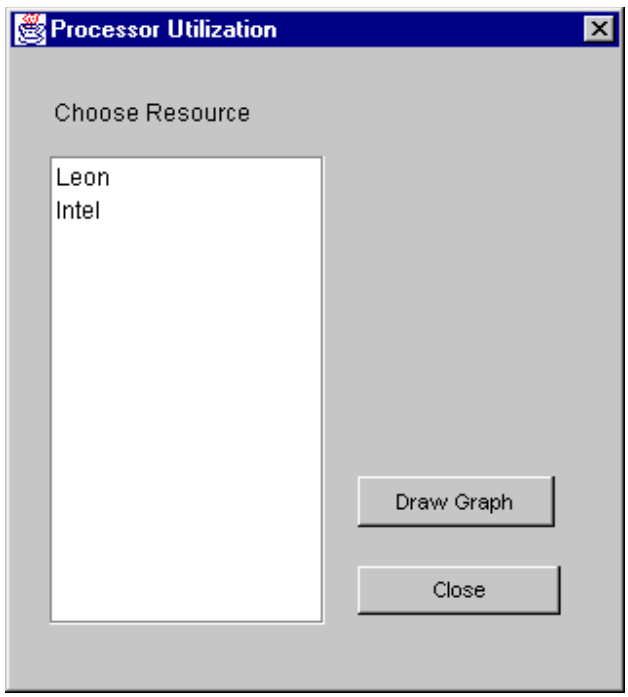
Then the file is chosen by clicking 'Browse' and the required data can be entered by the user by clicking 'Edit', which calls the internal text editor for editing the file. See Appendix B for more details.

3. Analyzer

According to the currently implemented analysis model, four types of analysis is provided:

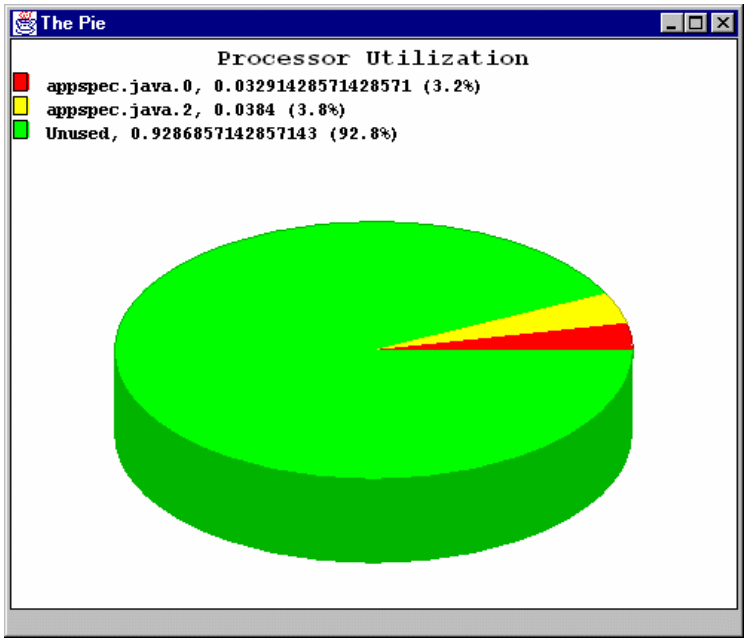
- Memory Space Utilization
- Memory Bandwidth Utilization
- Processor Utilization
- Bus Bandwidth Utilization

> The current analysis model implemented in this tool is same as the model used in the previous project (See References). However, since the analysis part is modular, it can be easily changed according to the needs.



By clicking the desired option, a dialog option is displayed for choosing the resource for which the analysis is to be done (see below)

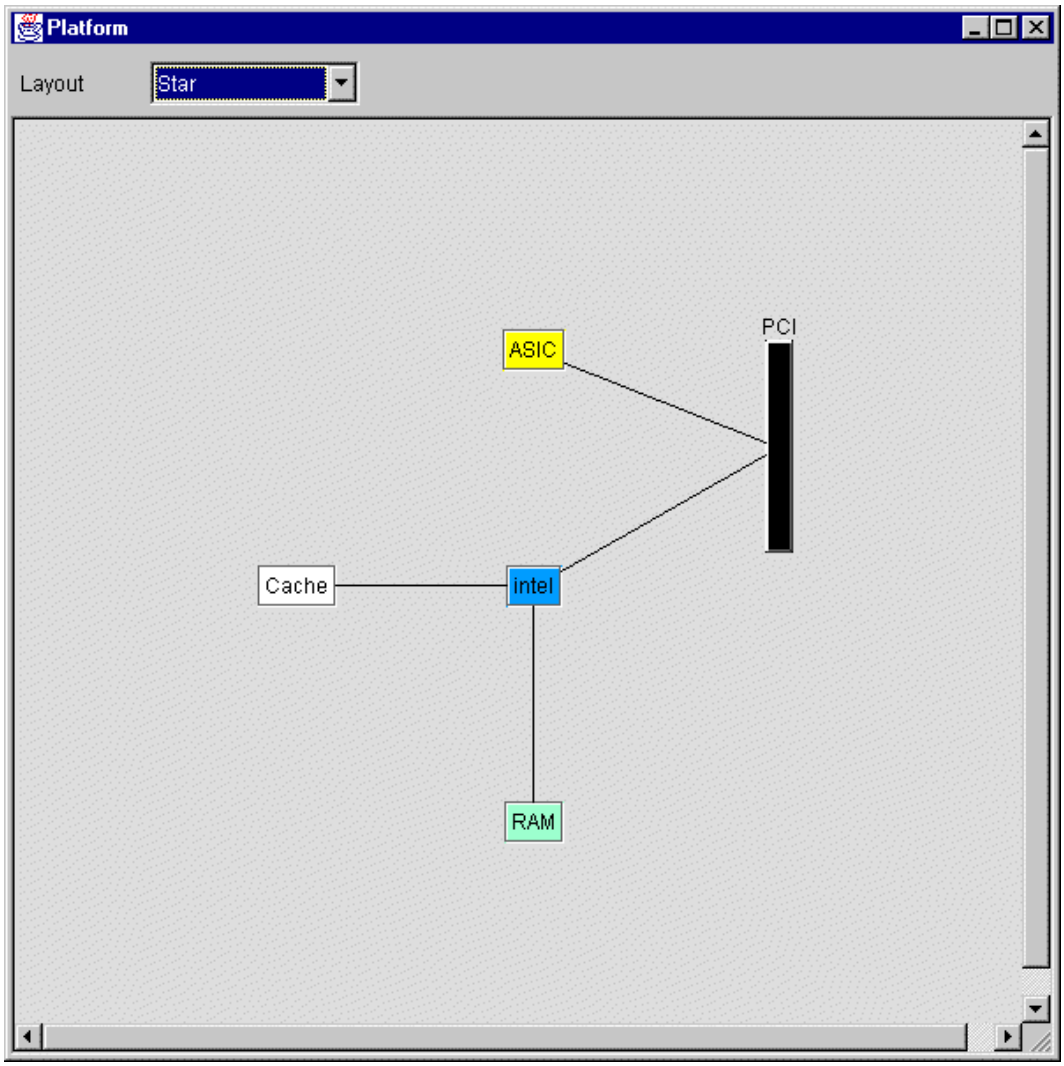
To perform analysis for a particular resource, click on the resource and click ‘Draw Graph’ to display the analysis results graphically with the help of pie charts.



Visualization

In Platform Visualization, two types of Visualization schemes are supported :

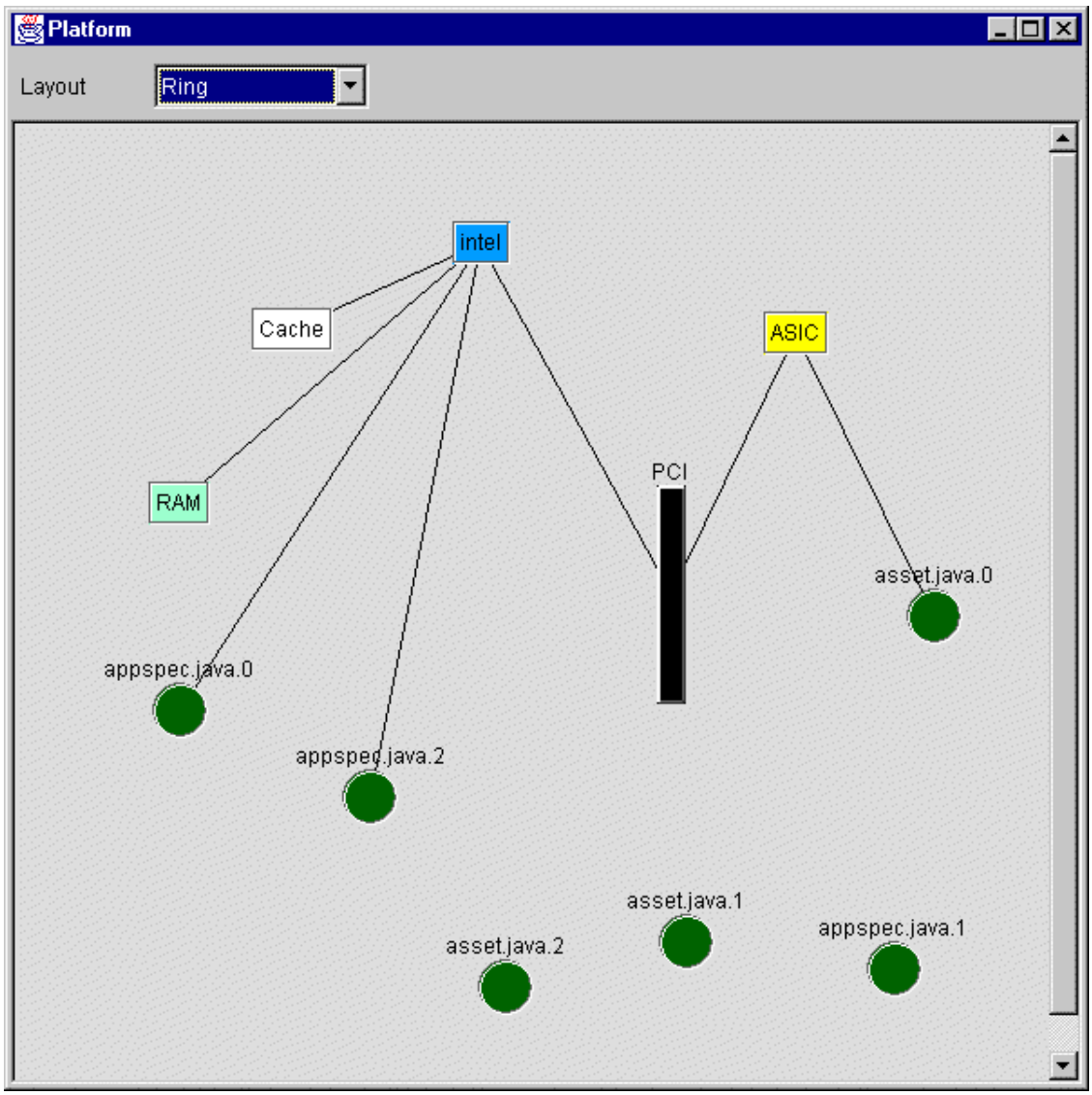
1. Platform Visualization



As shown above, all the components along with their interconnections are displayed. The various components are colour coded according to their type for easy identification. The Visualizer also supports the following options:

- Preset layouts are included like Star Layout, Ring Layout, which arrange the components according to fixed style.
- The components displayed above are can be dragged with the mouse to provide flexibility to the user to choose a custom layout.

2. Binding Visualization



In the binding visualization, the tasks bound to resources are also shown along with the platform. The tasks are shown as dark green circles, which are connected to the appropriate resource. Tasks, which are not currently bound, are also shown.

Synthesis

The synthesis tools currently supported are:

- Hardware (VHDL)
- Software (RC)
- Interface
- Kernel
- ASIP

Currently the above tools are not called. Dummy functions (Event handlers for above options) are included where one just has to fill in the code for calling the above tools appropriately.

Chapter

3

Modularity of Analysis and Visualization

The main highlight of the analysis and visualization part is that it is modular for easy modification and upgradation to more advanced analytic models.

Overview

Analysis

The Analysis part consists of two components :

1. Data Collator
2. Analyzer

The responsibility of the Data Collator is to collect all the data required for analysis from various sources and fill the data in the data structures required by the analyzer. The analysis part is not concerned with the source of the data and just produces the results.

Visualization

The output coming from the Analyzer is used by the Visualizer to display the results graphically.

Implementation

The Data Collator and the data structures required by the analyzer are present in the Class CAnalysis_Data.

The FillData() function present in the above class fills all the data structures from various sources.

The analysis is done in the Class CAnalysis.
For each analysis , separate functions are present. Each of these functions can be changed independently to implement a different analysis model. The functions assume that the data structures required for the analysis have already been filled up the Data collator.

Currently the functions present are :

public void MemorySpaceUtilization(String memname);

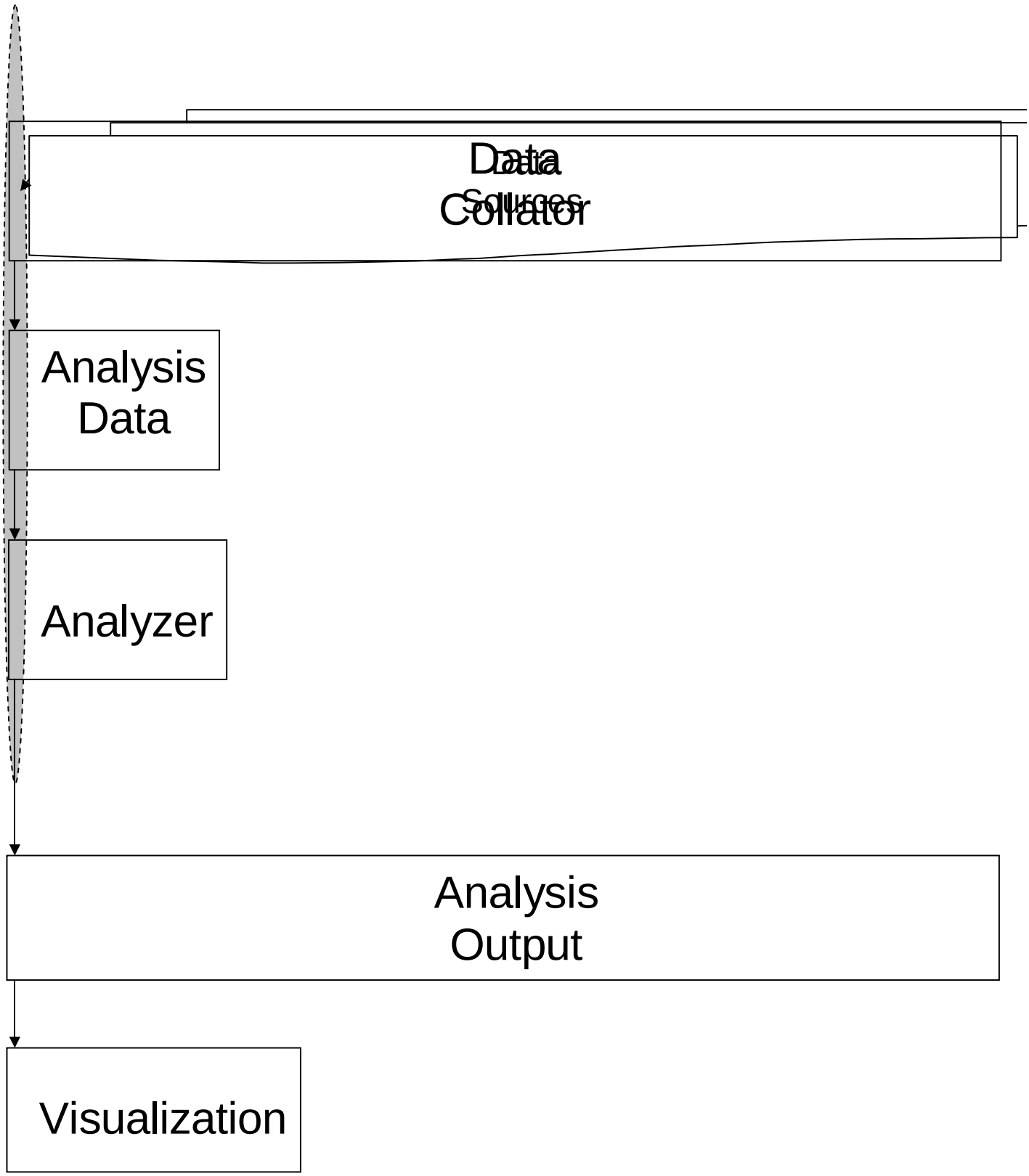
public void MemoryBandwidthUtilization(String memname);

public void ProcessorUtilization(String processorname);

public void BusBandwidthUtilization(String busname);

Each of these functions changes the *Global.graphdata* data structure which is used by the Visualizer to display the results.

Figure : Graphical overview of Analysis and Visualization



Chapter

Future Work

- **Data input from SUIF**

  SUIF interaction needs to be implemented as soon as appropriate SUIF interface is available. To implement this one needs to change the Data Collator part so that the data required by the Analyzer can be picked up from SUIF annotations.

- **More Sophisticated Analysis**

  Since the Analysis, part is modular, more sophisticated analysis models can be implemented. In fact, one can also have multiple analysis models in different files which can be used according to the requirement.

- **Option of component Library**

  A Library feature can be added to the program from which pre-specified components can included in the platform without giving their specifications again. Code for this will be minimal as the code for saving and opening the data structures is already implemented.

- **Saving of Visualization to image files**

  Option can be included to save the visualization to a graphic file format for persistent storage, which will be useful for presentation purposes.

  Note: On Windows 9x/NT Platform , the screenshot can be taken by pressing Alt+PrintScrn which can then be pasted in a Graphics application and saved to a file.

- Connections between the bus and the resources can be made more intuitive in the Visualizer

- Consistency check for ensuring proper channel – port communications can be made more informative. If interconnections are not consistent , then it can inform where exactly is the problem.

Appendix A

Specifications of Analysis Data

Input Item	Format	From (present)	From (finally)	Comments
1. Application				
(for all) Tasks				
Task name	String	Spec	Spec	
Space occupied	Int	User	Profiler	
Periodicity	Int	User		
Load-store count	Int	User	Profiler	
Dynamic instruction count	Int	User	Profiler	
ITC				
(for all) ITCs				
Task name	String	Spec	Spec	
Data	Int	User		
Mode	Int	User		
Bandwidth	Int	User		
2. Target platform				
(for all) Processors				
Processor name	String	Spec	Spec	
Speed	Int	User	Spec	To be extracted from processor description file
Bytes/instruction	Int	User	Description File	
(for all) Memory				
Memory name	String	Spec	Spec	
Access time	Int	Spec	Spec	
Bandwidth	Double	Spec	Spec	
Size	int	Spec	Spec	
(for all) Cache				
Cache name	String	Spec	Spec	
Hitratio	Double	Spec	Spec	
Access time	int	Spec	Spec	
(for all) FPGA				
FPGA name	String	Spec	Spec	
(for all) ASIC				
ASIC name	String	Spec	Spec	
(for all) Bus				
Bus name	String	Spec	Spec	
Bandwidth	Int	Spec	Spec	

3. Binding Information		Spec	Partitioner	
4. Interconnection		Spec	NA	

Spec : Specification (this data is obtained from Application Specification and the Target Platform Specification from within the tool)

User : this data comes from the user currently but will be eventually generated by the different tools

---

Appendix B

User Template

To make it convenient for the user to feed in the data manually the tool generates a **user template file.** This user template marks all the fields required for analysis that need to be fed in by the user. So all the user has to do is fill in the required slots.

Note that the template file will be different for different application and platform specification. A sample user template file is shown below.

```
 #Data regarding Tasks
<CAnalysis.~jav.0>
spaceoccupied =
2000
periodicity = 100
load store count = 2000
dynamic instruction count = 10000
data = 100
mode = 2
bandwidth = 10

<CAnalysis.~jav.1>
spaceoccupied = 1000
periodicity = 200
load store count = 3000
dynamic instruction count = 20000
data = 200
mode = 1
bandwidth = 20

<CAnalysis.~jav.2>
spaceoccupied = 3000
periodicity = 300
load store count = 2000
dynamic instruction count = 10000
data = 100
mode = 2
bandwidth = 15


#Data regarding Processors
<pentium>
speed = 600
Bytes/Instruction = 2

<trimedia>
speed = 400
Bytes/Instruction = 3
```

References

1. Analysis and Visualization of Platforms , M.Tech Thesis Report , CH.Sheshagiri

2. www.swfm.com – Mica Graphics Framework.