J2EE client-server platform delivering integrated IP and EDA solutions for power and performance critical digital systems

Robert Graham

Thesis submitted for the degree of Master of Science
by Research 2005
(part-time)

The University of Edinburgh

Abstract

The commercial world of digital circuit design is a highly competitive environment. Speak to the CEO of any company in this field, or in fact almost any industry for that matter, and they will openly validate that the critical keys to success are minimising costs while ensuring fast-time-to-market. If the overhead costs associated with developing a product are in excess of the revenues generated from that product, a business will not survive. However, going overboard on cost-cutting will have a negative impact on time-to-market due to lack of resources, hence a strategic balance between these must be sought.

This thesis explores the potential of a novel means of providing remote access to both intellectual property (IP) macro-components and EDA tool suites. To this end a web enabled J2EE client-server based EDA and IP portal type platform has been designed and developed. Through a dedicated integrated development environment incorporating schematic capture technology, engineers can download and plug together virtual instances of state-of-the-art digital IP cores. After creating a virtual design it can be functionally verified and synthesised using packages from a range of server-side EDA tools.

This approach has several key benefits. Firstly development timescales can be radically shortened as the framework is architected around libraries of highly optimised macro-components that can be rapidly plugged together to form complex digital systems. SoC's derived from these macro-components will be inherently low-power, reducing the need for designers to have in-depth experience in algorithmic low power techniques. Secondly the architecture of this framework allows seamless and transparent design capabilities incorporating these IPs, followed by RTL level verification without users ever having direct access to the RTL. This maintains the security of the intellectual property invested in the macro-component libraries without compromising design performance. Finally client users of the framework have access to EDA tool suites without directly incurring their acquisitioning and licensing costs. Accessing these tools as a web service will help to mitigate some of the overhead costs of digital system design. Ideally this will lead to a low cost of ownership solution well suited to SME's, allowing them to operate and succeed in the highly competitive SoC market place.

Declaration of originality

I declare that this thesis and all work described herein is my own work, with the following exceptions:

Figure 1-3 is based upon a figure in [14].

Student Signature:

Student Name:	Robert Ian Graham
Principal supervisor:	Prof. Tughrul Arslan
Second supervisor:	Dr. Alister Hamilton

Acknowledgements

As anyone engaging in state-of-the-art technology research projects will only be too aware, the quest for innovative and novel solutions is a daunting task for any one person to undertake single-handed. Hence there are several people that I would like to pay homage to for their contributions of time and effort in helping me along the way.

Firstly I'd like to show my appreciation to my supervisor, Tughrul Arslan.

Secondly I would also like to voice my appreciation to others that contributed towards, supported and guided this project. A great deal of hardware development support was provided by Ahmet Erdogan, a true expert in the field of digital system design. Much of the research in this thesis is focused towards the needs of small businesses operating with limited budgets. A great insight into the business and working practices of these small enterprises was provided by Andrew Sijan.

There are several other people I would like to acknowledge for their indirect support and guidance. These are Matthew Lawrenson, Gordon Stuart and David Walton, the directors of Spiral Gateway for showing me that life has uncertainties – but that these can progress to better opportunities. They have given me invaluable experience for which I am grateful.

I would like to give huge thanks to Ruth Moore. She is a very special person who was able to provide inspiration, emotional guidance and support at the times when I really needed it. Her love and caring meant everything to me, helping me to focus and ultimately become who I am now.

Finally I would like to take this opportunity to give a special mention to Yutian Zhao. She has provided fantastic support and comfort throughout the time I have known her. Her amazing character and personality always lifts my spirits when I'm feeling down. Hopefully she will continue to provide this through a life-long bond and partnership.

Key words

Low power, macro-component, IP, IP Libraries, IP Cores, SIP, virtual components,, schematic capture, java, J2EE, EJB, CORBA, design patterns, internet, design services, EDA, SoC, ASIC, algorithmic optimisation, SME.

Table of contents

Table of figures

Table of tables

Abbreviations

The following is a dictionary of the acronyms and terminology that may be used throughout this document.

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
BES	Borland Enterprise Server
BMP	Bean Managed Persistence
CMOS	Complementary Metal Oxide Semiconductor
CMP	Container Managed Persistence
CORBA	Common Object Request Broker Architecture
DLL	Dynamic Link Library
EDA	Electronic Design Automation
EJB	Enterprise Java Bean
FIR	Finite Impulse Response
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
I/O	Input/Output
IDE	Integrated Development Environment
IP	Intellectual Property
J2EE	Java 2 Enterprise Edition
JAR	Java Archive
JDBC	Java Database Connectivity
LAN	Local Area Network
LSI	Large Scale Integration
MAC	Multiply Accumulator
MVC	Model-View-Controller
OO	Object Oriented
PDF	Portable Document Format
RAM	Random Access Memory
ROM	Read Only Memory
SIP	Silicon Intellectual Property
SME	Small to Medium sized Enterprises
SoC	System on Chip
SQL	Structured Query Language
VLSI	Very Large Scale Integration
VSIA	Virtual Standards Interface Alliance
XML	eXtensible Markup Language

Legal issues

Patents

Some of the work outlined and detailed in this thesis is protected by a published patent: GB0301993.2 [28].

Copyright and trademark information

The work implemented in this project requires the use of and interaction with several commercial EDA applications. The products or tool names and the company names are protected by copyrights. General copyrights and trademarks are acknowledged below and apply throughout this document.

Copyrights © and trademarks ™used are

Cadence

Synopsys

BuildGates

DesignWare

Verilog-XL

NC-Verilog

MySQL

Third party software licenses

The software developed during this project required the use of several 3rd party applications and libraries. The licenses covering the usage and terms and conditions of these products can be found in Appendix D. The majority of these 3rd party products require the distribution of their full license with any application using them.

Chapter 1
Introduction

In this modern era of rapid evolution in the electronics industry, circuit designers are always eager to be abreast of the latest electronic design automation (EDA) solutions and technologies. However, due to the inordinate complexity of leading edge design tools, the costs attributed with them discourage a significant proportion of design companies from their use. This unfortunately relegates financially constrained companies to using dated and/or inferior EDA systems in an attempt to develop what should still be classed as state of the art hardware products for their customers. In an attempt to curb costs and enhance production efficiency, a design facility may decide to upgrade their development tool-kits to maintain compliance with the latest technologies. Not only is this an expensive venture but also potentially time consuming. After acquiring the appropriate software packages, they need to be installed on site. Considering the heavily bloated distributions provided by the majority of big named vendors, significant resources might need to be allocated to the new set-up. Computer processing and storage requirements may need to be reviewed. Assuming that computational resources suffice, a significant proportion of time will then need to be dedicated to configuring a stable and reliable installation. Once up and running, optimum efficiency can only be achieved via constantly tweaking and patching these overtly complex CAD environments. As dedicated support staff are an expensive proposition, especially to small to medium sized enterprises (SME) this chore is all to often allocated to the design engineers themselves. This arrangement significantly detracts the small company from its core competency – that of designing integrated circuits.

Remarkably, since published in 1965, Moore's Law [1] has prevailed to be the guiding trend in defining the semiconductor roadmap. Now forty years later, respected industry figures including Walden Rhines, Chief Executive of Mentor Graphics [2], continue to commend this empirical observation that the performance and component densities of integrated circuits doubles approximately every eighteen months [3]. Even as device geometries shrink into the ultra-deep-submicron, 0.13µm and below, novel solutions continue to emerge [4] to overcome the growing challenges faced – ensuring Gordon Moore's predictions will hold valid well into the future.

Considering these ever-escalating complexities in modern integrated circuits, designers are increasingly turning to technology libraries of 'Intellectual Properties (IP)' or 'virtual component blocks'. These are silicon verified hardware core modules designed primarily for reuse as functional blocks in larger architectures. Through the plugging together of these cores, larger designs and even full systems can be realised. The critical advantage of this approach is that complex applications can be rapidly assembled by linking together the desired cores, providing of course their interface protocols are compatible. This means that designs are no longer constructed from the ground up but rather from the pedestals of these core building blocks. Additionally, the level of expertises required to implement advanced designs is greatly diminished. Typically pre-designed IP modules are fully optimised in terms of power consumption and/or silicon area, both of which are traits precluded to the realms of highly experienced design engineers. Reinforcing the physical benefits of the IP approach is the fact that each core is guaranteed to synthesis error free to a particular silicon technology. Therefore design engineers need only verify and debug the interfacing between the cores they have chosen to embed in their designs. Hence the amalgamation of the benefits provided by the IP approach helps to score the primary business goal – that of minimising time to market.

However the concept of reusable IP cores is not without its problems. For small design companies the acquisition of suitable libraries can be an expensive reality. Currently the world's leading vendor in IP core libraries is Synopsys with its DesignWare [7] products for which there are twenty-five thousand customers. This library has a starting price of $17,000 and offers a range of foundation level cores. As well as containing cores essential to most designs like adders and multipliers, it also contains building blocks that may never be of use to particular customers. Design managers must then be able to justify the cost of the entire library against the proportion of the library they realistically intend to use. In a highly competitive industry costs must always be minimised leading to the classic “make-or-buy” decision [8]. Design managers may see it as cheaper for their company to make their own IP blocks rather than purchase acquiring from 3rd parties. But doing so diverts resources from the company’s goal of developing the products key to its expertise meaning longer development cycles and increasing its time to market. Missing early market opportunities will, in the long run, cost the company.

There are also potential problems for authors of reusable IP cores intending to sell them, either as stand-alone blocks or as part of a library. The most obvious issue is with security. Once an IP has been sold, the author looses direct control of their intellectual property. The provider is then left to trust that any new proprietor will respect and obey the terms and conditions of any license accompanying the virtual circuit blocks. However there are several steps that the author can take to enhance the security and protect their IP.

- <u>Pre-compliation</u>

  There are two types of synthesis and verification tools, interpreters and compilers. Interpreters take the source hardware description language (HDL) code and interprets it at run-time. An example of an interpreter is Verilog-XL from Cadence. For these tools to operate, the source code needs to be available meaning that the security is potentially compromised. Compilers, such as NC-Verilog – also from Cadence, are tools that initially convert the source code into a binary representation that can be more rapidly simulated. IP vendors can choose to distribute their cores in a pre-compiled form without the HDL source leading to a more secure solution. This is the security concept behind the Synopsys DesignWare libraries. Although secure there is an element of vendor-lock-in meaning the IP can only be used with a specific range of EDA tools.

- <u>Encryption</u>

  As security is of critical concern in the IP industry there has been significant interest in encryption. To date the barrier to encryption technologies has been the need for all tool vendors to adopt the same standard. Recently the Virtual

Standards Interface Alliance (VSIA) has begun leading the initiative to define an industry wide encryption standard [9] that will benefit both EDA and IP vendors.

- Obfuscation
  There are some simpler techniques providing a reasonable degree of IP protection. The most popular of these is obfuscation [10, 11]. In lay terms obfuscation takes an intelligible source code and makes various manipulations to reduce its readability to the point whereby it is impractical to understand or reverse engineer it. These are typically automated refactoring [12] operations that will change the names of all variables, methods and functions into seemingly random assortments of characters. Key to the success of this technique is the fact that all the new names are very similar so that a human reading it will quickly become confused and unable to follow its structure. Additional manipulations include stripping out comments and whitespace to further reduce its intelligibility. This security through obscurity technique only impinges a human's ability to decipher the code. The obfuscated transform is functionally equivalent to the original source code.

1.1 Inspiration

Prior to starting of this project, market research by The University of Edinburgh identified a need for a low cost solution that could help small to medium scale enterprises (SME's) compete in the fiercely competitive consumer electronics industry. Many SME's are sitting on potential technology that could make a real impact in the market place, but are inhibited by high-cost EDA tools and lack of high-performance, low-power design skills. In response this thesis investigates and sees the realisation of a platform addressing these inhibitors. The proposed platform has since been patented – "*System and method for rapid prototyping of ASIC systems*" [28].

1.2 Objectives

Having identified the issues highlighted in the previous sections, this thesis will proceed to describe research and development conducted to investigate a potential solution. The key objectives of the research are:

1. Remote access to intellectual property libraries
2. Security of the IP invested in these libraries
3. Remote access to EDA tool suites
4. Encompass the above objectives within an effective design platform
5. Efficiency, scalability and reconfigurability of this design platform

All of these objectives will be integrated into a single platform using internet technologies as a communication medium. This effectively constitutes a client-server architecture whereby the server securely hosts IP libraries and EDA tool suites which can be accessed anywhere in the world using a specifically designed client development environment. Based upon IP libraries originally developed at the University of Edinburgh, the resultant framework should facilitate the secure rapid deployment of silicon solutions for low-power, high-performance applications.

1.3 Intellectual property library

As previously indicated the primary objective of this research is the deployment of intellectual property cores through a web-based EDA service. Hence a suitable range of IP cores must be made available. For many years The University of Edinburgh has been researching and developing low-power, high-performance techniques for digital systems. Through time, the university has collated a large assortment of cores aimed at many different applications. One MSc research project by Eric Zwyssig [13] attempted to build a catalogue of IPs, called the macro-component library. Macro-components are essentially IP core building blocks that can be used to form a solid foundation upon which advanced, high-performance, low power digital systems can be constructed. The framework described in this thesis builds upon Eric's work as well as incorporating many additional IPs developed in other research projects since. The following subsections will highlight some of the key characteristics of these macro-components justifying the thesis title which claims suitability for power and performance critical digital systems.

Throughout this thesis, the terms “macro-component”, “core” and “IP” may be used interchangeably in reference to macro-components.

1.3.1 Reusability

By far the most important characteristic of macro-components is their reusability. If a future vision of large-scale system-on-chip (SoC) applications is to be a reality, an extensive re-use methodology must be implemented [5]. This being recognized throughout the industry, two of the major EDA vendors have collaborated to publish the book “Reuse Methodology Manual” [6] as a set of recommended guidelines to be followed by digital hardware design engineers.

During the design of the macro-components, careful consideration is given to their interface configuration and degree of internal flexibility. Designed in a generic fashion with common interfaces across similar category types, these blocks can be “plugged together” like jig-saw pieces to form advanced systems of arbitrary size and complexity. An example of this can be seen in Figure 1-4. Design trade-offs can then be considered and optimisations introduced by simply substituting key blocks in the design with alternatives of a similar category. For example a datapath’s performance may be dominated by its multiplier. Performance trade-offs in terms of power, area and speed can then be made by “plugging in” the appropriate alternative multiplier from the library into the design.

1.3.2 Parameterisability

The issue of internal flexibility is dealt with through parameterisation. Parameterisation is the process of configuring the characteristics of the core at design compilation time. For example the bit-width of the module inputs may be defined using a parameter called `input_width`. Variations of that component can then be derived by simply over-riding the `input_width` parameter value at design time. Verilog HDL supports the `parameter` keyword to identify the available parameters in the module. VHDL uses the term *generics* for the same purpose, but as the macro-component library is purely defined using Verilog, we will no longer refer to generics in the thesis.

1.3.3 Inherent low power characteristics

One of the key objectives when designing the macro-components was “low power”. With the increasing demand for portable products, power consumption has become one of the key issues in SoC design. All portable consumer products are dependent upon their batteries. Unfortunately power demands for new applications is growing faster than the rate of technological advances in battery technology. To compensate for this, application designers must attempt to make circuitry that consumes less power. The power consumption for CMOS based digital circuits can be calculated as shown in Figure 1-1.

$$\text{Power} = \underbrace{S \times C_L \times V_{dd}^2 \times f_{clk}}_{\textit{switching}} + \underbrace{I_{sc} \times V_{dd}}_{\textit{short circuit}} + \underbrace{I_L \times V_{dd}}_{\textit{leakage}}$$

Figure 1-1
power dissipation in a CMOS digital circuit

Figure 1-1 shows that the power consumed in a CMOS circuit can be attributed to three distinct characteristics, these being switching capacitance, short circuit current and leakage current. The dominant form of power consumption can generally be attributed to gate switching. The principle of gate switching is demonstrated using the NOT gate of Figure 1-2. Each gate has an associated parasitic capacitance C_L that must be charged and discharged each time that gate switches. The larger the number of gates in a design, the greater it’s switching capacitance and hence the more power it consumes.

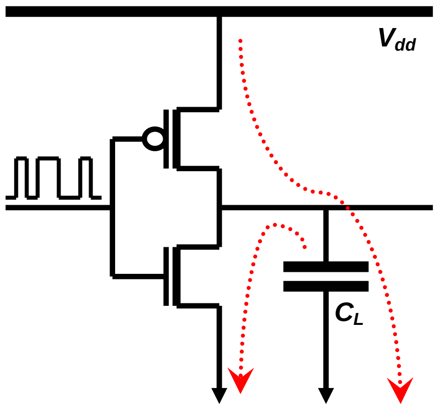
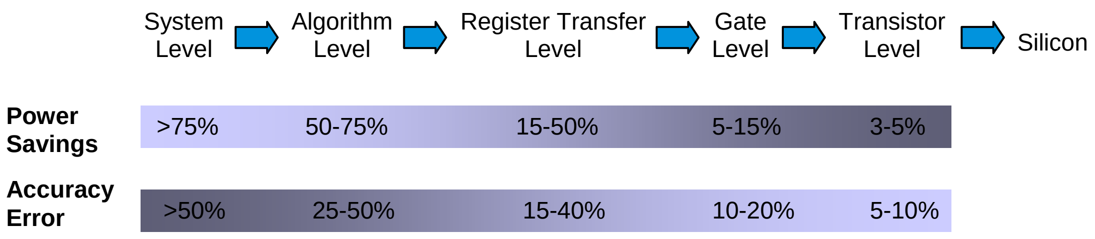


Figure 1-2
Switching power dissipation

All the macro-components incorporate various *algorithmic-level* optimisations that lead to significant power savings, which in some cases can be as high as 75% – see Figure 1-3 [14]. As these cores are available as system-level building blocks, any design incorporating them will naturally inherit these power savings. Hence the macro-component libraries offer the opportunity for engineers to design and develop low-power systems with little or no experience in low-power methodologies.



	System Level	Algorithm Level	Register Transfer Level	Gate Level	Transistor Level	Silicon
Power Savings	>75%	50-75%	15-50%	5-15%	3-5%	
Accuracy Error	>50%	25-50%	15-40%	10-20%	5-10%	

Figure 1-3
Available power savings at different levels of design abstraction [14]

1.3.4 Technology independent

All macro-cores are written in pure synthesisable Verilog RTL and are totally independent of any underlying technology cell libraries. This gives enormous flexibility and greatly extends the range of end-use applications for the cores. Design engineers typically work to a specification that will define the cell library vendor and the device geometry size. Hence a customer using UMC's 0.18µm cell libraries will have a comparable power/performance advantage to another customer building on TSMC’s 0.15µm silicon process when using these cores.

1.3.5 Tool flow independent

The characteristics that make the macro-components technology independent also make them tool flow independent. All HDL design files are written in pure Verilog RTL and only describe the system architecture. Necessary synthesis directives and optimisation constraints are included in separate tool-flow dependent script files as opposed to pseudocomments in the Verilog source code. Different scripts can be developed for each desired tool flow.

1.3.6 Constructing an IP from macro-components

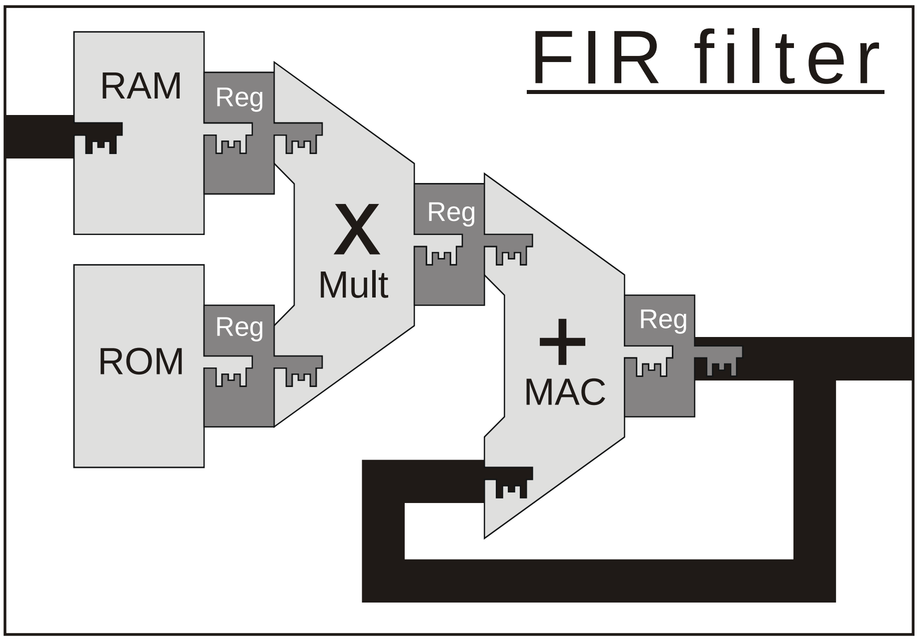


Figure 1-4
FIR filter constructed from macro-cores

Figure 1-4 shows how a Finite Impulse Response (FIR) filter can be rapidly assembled using macro-components from the library. As can be seen the basic architecture comprises a RAM, ROM, multiplier and accumulator [15]. The RAM stores the incoming data to be processed. Coefficients defining the transfer response of the filter are stored in the ROM. On each clock cycle, one of the coefficients is multiplied by the in input signal in the RAM and the results accumulated by the multiply accumulate unit (MAC). Studies of this architecture have shown that the majority of the power is consumed by the multiplier [16]. Using the macro-component concept, the multiplier can be simply removed from the design and replaced with another one that leads to better power/area/speed characteristics.

Referring to Figure 1-4 the multiply unit is one of several multiplier macros. All these macros have a similar interface, meaning they have the same inputs and outputs, making it relatively straightforward to swap between them. The figure shows these inputs and outputs (I/O's) using a *key* symbol. This is intended to reflect the parameterisability of the core. When it is instantiated in a design, it should be parameterised such that its ports *fit* with the modules around it. Most the macro-components have extensive parameterisations allowing them to be adapted for a variety of situations.

1.3.7 Macro-Libraries

As a result of several years of dedicated research at the University of Edinburgh, a vast resource of highly optimised macro-component IP cores have been designed and developed. These IP cores range in complexity from simple adders and multipliers to higher-level blocks including FIR filters, receivers and various codecs. Based upon work originally conducted by Eric Zwyssig [13], a single macro-component library was developed. During this research period the macro-component library was radically extended in terms of available cores and their end use applications. To capitalise upon the potential extensibility of the IP delivery platform, the single macro-component library was partitioned into the arrangement shown in Figure 1-5.

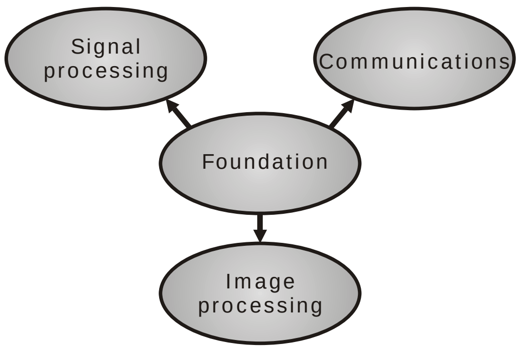


Figure 1-5
Macro-component IP core libraries

1.3.7.1 Foundation

The foundation library is the base library from which all other libraries are built. It contains the majority of the low-level cores that are essential dependencies of the higher-level libraries. As such the cores found in this library have minimal scope for low power optimisation. Rather it is the way in which these components are used in the larger designs that lead to substantial algorithmic power savings. Examples of the types of macros found in this library are adders, subtracters, registers and multipliers.

1.3.7.2 Signal processing

The signal processing library complements the foundation library to include various DSP functions such as Finite Impulse Response (FIR) filters and Fast Fourier Transforms (FFTs) [17].

1.3.7.3 Communications

All macros specifically relating to communications based applications are to be found in this library. Macros found here include high-throughput FIR filters [18] and Code Division Multiple Access (CDMA) receivers [19] [20].

1.3.7.4 Image processing

All image processing macros can be found in this library. Example applications found here include Discrete Cosine Transforms (DCTs) [21] and wavelets [22].

1.3.7.5 Cache libraries

Each of the main libraries has one or more corresponding shadow libraries, referred to as cache libraries. Cache libraries are used to house all dependency modules which, in their own right of limited value, but critical for their parent macros. An example cache module is one of the Wallace tree stages of the Wallace multiplier. The Wallace tree state module is of little or no value to any application other than the Wallace based multiplier. Placing these modules into the cache libraries keeps the main libraries clutter free. Therefore every module found in one of the main libraries should be a useful core in its own right. This arrangement is essential for the platform deployment as only the non-cache libraries need be included in the server-side databases.

1.3.8 Design Flow

Figure 1-6 depicts a typical design flow that most application specific integrated circuit (ASIC) engineers will be familiar with. Each stage in the design flow represents one of the steps necessary to progress a design from high-level RTL to production and silicon ready GDSII representation. However with constantly shrinking process technology geometries, there are great challenges to meeting timing closure. This can require further optimisations to the design flows [23].

The first stage of the design process is to describe the system being developed using a hardware description language (HDL). This particular design flow uses Verilog as the HDL but VHDL can also be used. A testbench will then be developed to verify the functionality of the HDL description. Assuming that the HDL model is described in register-transfer-level (RTL) notation, it can then be synthesised into a technology specific netlist. Again this should be verified, preferably using the same testbench as that used for the RTL to ensure that the synthesis process completed correctly and that the resulting netlist is functionally equivalent. Finally a floorplanning process is employed to lay out the design on silicon. A final verification stage should then be employed to ensure that the design still operates as expected and timing closure is met.

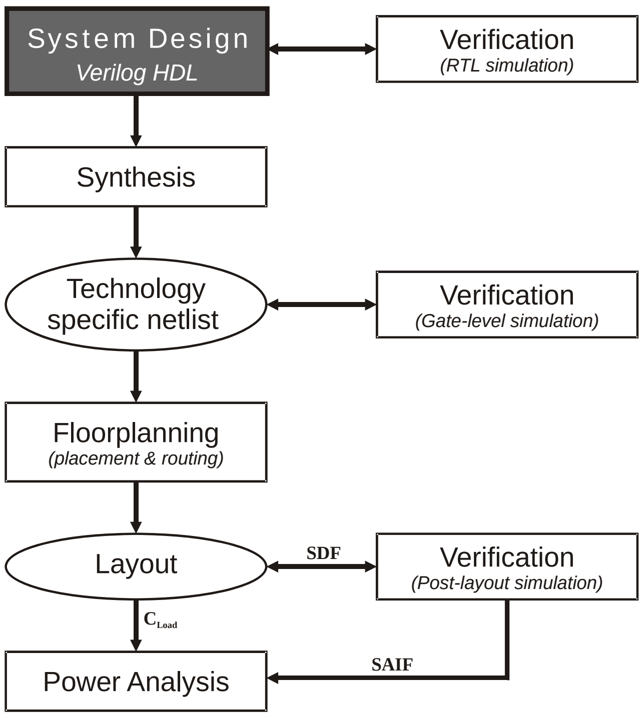


Figure 1-6
Typically used ASIC design flow

This particular design flow has been adopted for this platform due to its obvious hierarchy and the ease with which this can be automated. This hierarchy has clearly defined steps such as RTL design, verification, synthesis, physical design, and physical verification. However, if closely inspected it is possible to identify potential concurrency in the flow through partitioning. An example of this is the hierarchical concurrent flow graph (HCFG) which captures the iterative, hierarchical, and concurrent nature of ASIC design flows [24].

Each of the steps in the design flow is performed by a particular software package tool. To a certain extent the design flow is fixed in terms of the tools implementing each stage, as those from each vendor will employ different types of optimizations. Recent studies have shown that it is possible to exploit the potential of modern programming languages to simultaneously optimise design flows to support multiple EDA tools [25].

1.4 Electronic design automation – *EDA*

Since the introduction of hardware description languages (HDL's) in the 1960's, a great deal of time, money and research effort has been dedicated to developing more effective and efficient ways of designing electronic circuits. The majority of this work has been in finding new ways of raising the level of abstraction with which designs are described. Originally engineers described circuits in terms of transistors, later progressing to gates. In the late 1980's rudimentary logic synthesis tools emerged further raising design abstraction to register transfer level (RTL). Today the industry is heading towards behavioural level abstraction with hardware compiled directly from C/C++ based languages. As chips become ever larger, incorporating more complexity, there is a corresponding demand for new means of simplifying their design process.

In 2004 the global semiconductor industry had an estimated value in excess of $200 billion [26]. Propelling this industry's growth and maintaining the world's insatiable appetite for new technological applications is a supporting plethora of electronic design automation tools. It is this unfathomable diversity of EDA products and tools that drives the bleeding-edge of the current technological revolution. Indeed Moore's Law can also be considered to apply to the past, current and future roadmap of the EDA sector [27].

Referring to Figure 1-6 a typical digital design flow has many distinct stages. Each of these stages will generally require one or more dedicated EDA tools. The first stage in the flow is designing the 'system' and presenting it as an HDL description. This in itself could be further divided into many sub-sections. For example the system could first be mathematically modelled using a high-level tool such as MATLAB to verify the algorithms. Performance analysis tools may be used to optimise the hardware/software partitioning. Code generation tools could be used to automatically derive the HDL from suitable templates. In fact the list of system design and optimisation tools is almost endless. Following system design is the verification stage, which ensures the HDL description is functionally accurate and/or equivalent to the system model. The RTL files are then synthesised into a technology specific and once again verified to ensure compliance with the system specification model. All of these stages are considered the "front-end design". A further suite of EDA tools is then necessary for the back-end design stages, which are concerned with optimising the silicon layout.

1.5 Remote IP and EDA services

This section will take a look at current practices in the semi-conductor industry and then introduce the potential of deploying these as remote web-based services.

1.5.1 Remote IP access

The typical industry model is to purchase intellectual property libraries such as those available from DesignWare [7] and install them on their local area networks (LAN). For SME's in particular this can be an expensive proposition. These libraries come with a high price tag and can contain more blocks than the customer ever realistically expects to make use of. Additionally the user may be locked into using a particular EDA tool flow for the given library, such as Synopsys' DesignCompiler.

Intellectual property libraries really only apply to foundation level cores. Larger cores are usually purchased as stand-alone blocks. Worldwide there a hundreds of vendors selling thousands of IP cores. This can make finding the right core for a particular application a daunting task. Recognising this, The VCX [29] and Design and Reuse [30] set up web services acting as central repositories for IP cores from different vendors. Customers registered with these companies could purchase 3rd party cores via their services. Although expanding the intellectual property market, these services were not without their problems. Customers had limited assurances that the cores would fulfil their needs in a satisfactory way until after purchase and run through a simulation environment. Some vendors selling via these services were worried about the security of their IP and thus only released technology specific netlist implementations of their cores, or in some case hard IP blocks. These may not suit customer's design flows and would take much longer to verify in simulation environments.

The solution described in this thesis attempts to circumvent some of these problems. All IP cores available through this platform are stored in RTL form on a secure server. The IP modules available on the server can be browsed using either the schematic capture development environment described in Chapter 1 or potentially via HTML web pages. This platform comprises a server-side database cataloguing the interfacing characteristics of all IPs in the libraries. Virtual ghost images of these cores can be downloaded and instances of them pluged together within the design environment, all while under the impression that the cores are locally installed on their workstation. As only IP block interfacing information is downloaded from the server, the service is fast enough to operate in real time.

This arrangement also helps maintain the security of the intellectual property within the macro-components as the RTL never leaves the server. However and the design is still constructed at the RTL level verification performance levels are still high.

1.5.2 Remote EDA access

Since the dotcom boom in the early 2000's there has been significant interest in attempts to provide EDA services via the internet. Most of these are based on portal technology that emerged around the same time. The term portal means gateway and refers to web sites that attempt to offer a diverse range of services under a unified interface. A classic example of a well known portal is the Yahoo web site. This site is a portal to search engines, email, online shopping and chat rooms to name but a few. In the context of semiconductor design, a web portal could offer IP search engines, component catalogues, vendor information and even access to EDA tools. With this in mind, the term portal could also be used to cover the services of The VCX and Design & Reuse, mentioned in section 1.5.1.

As the technology progressed, enterprise web portals emerged and were seen as the new principle in software engineering of the time [31]. Possibly one of the most comprehensive descriptions of the potential services available under an EDA type enterprise web portal environment is captured in a patent belonging to Cadence [32]. This was a very broad patent covering many technologies in a very vague manner – possibly making it difficult to enforce. This was later refined, targeting only chip design (dropping electronic circuit design from the title) [33] and was to become the bases for an enterprise portal web site known as SpinCircuit. However this service never really took off and the web site was only active for a short period, although Cadence still owns the *spincircuit.com* domain name. A potential reason for the initial failure of SpinCircuit is the inordinate complexity involved in constructing such a service, integrating IP database, verification, synthesis and even chip fabrication using nothing more than a web browser. Others have adopted simpler approaches, for example using portal technology to search remote vendor IP databases, download the relevant information and us a 'neural dynamic hub' to simplify transporting design information between local design tools [34].

1.5.3 IP & EDA development framework

There has been much research involved in designing and developing remote IP services and remote EDA services. However there is little evidence of a solution that truly integrates both of these services in a seamless manner. Cadence seems to have tried by offering a portal interconnecting design engineers with services and IP libraries from disparate vendors. However this is problematic as IP from one vendor may not necessarily smoothly interact with those from another vendor. Additionally, IP optimised for tools from one EDA flow may have poor characteristics if the designer decides to access a different tool set through the portal.

This thesis outlines the framework for a system that attempts to provide both IP and EDA remote services as a tightly integrated solution. Within this framework, designers have 'virtual access' to extensive libraries of high-performance, low power IP cores. From within a custom developed schematic capture application designers can plug together virtual instances of macro-components, gradually building comprehensive digital and DSP systems. This framework is architected such that the users perceive that they are using IP libraries and EDA tools within their own LAN, as they are working from a locally installed executable application that interacts with the server as opposed to a set of HTML pages viewed using a web browser. This gives the design environment the same responsive feel as traditional tools without the latency associated with HTML based solutions and their notoriously slow page refreshing. Seeing only the schematic capture client application, the concept of an underlying portal supported by an enterprise application server is abstracted away.

Additionally, this framework has been designed such that it can operate independently of the internet portal technology. Making extensive use of the façade design pattern in conjunction with the model-view-controller design pattern, the IP catalogues can be defined within an SQL database (of the same structure back-ending the application server) or within XML files. Similarly the simulation and verification tools can be accessed via the portal or on the local area network. This novel enhancement extends the functionality and extensibility beyond other solutions currently available.

1.6 Summary

This chapter introduced some of the problems currently facing the digital design community and in particular those operating under tight financial constraints. These are predominantly access to high-quality design building blocks, protection of intellectual property and the expense of state-of-the-art EDA tool facilities.

An IP/EDA framework was proposed that would attempt to mitigate some or all of these issues. This framework is based upon extensible libraries of intellectual property macro-components and a supporting EDA tool flow wrapped together and deployed as a remote service. The service would see IP securely protected on the server with a client side application accessing virtual images of the macro-cores. This will protect the intellectual property, while providing users all the performance benefits of RTL level design. Access to EDA tools will allow resultant designs to be verified and synthesised.

Chapter 2

Client-server architecture

To implement a system with capabilities defined in the introduction chapter requires a client-server software architecture. There are currently two technology platforms for implementing such services, Java 2 Enterprise Edition (J2EE) and Microsoft .NET. If optimised and performance tuned, .NET would seem to have a slight advantage [35] however it is tied to the Microsoft platform making it unsuitable in this application as it must integrate EDA tools running on Sun Solaris. J2EE solutions are highly-complicated multi-tier platforms with minimal constituent parts being an enterprise scale application server and database servers hosted on a powerful workstation, or even clusters of workstations. The configuration and architecture of the software and hardware in J2EE environments and the trade-offs between them has a significant effect upon the performance of these servers [36].

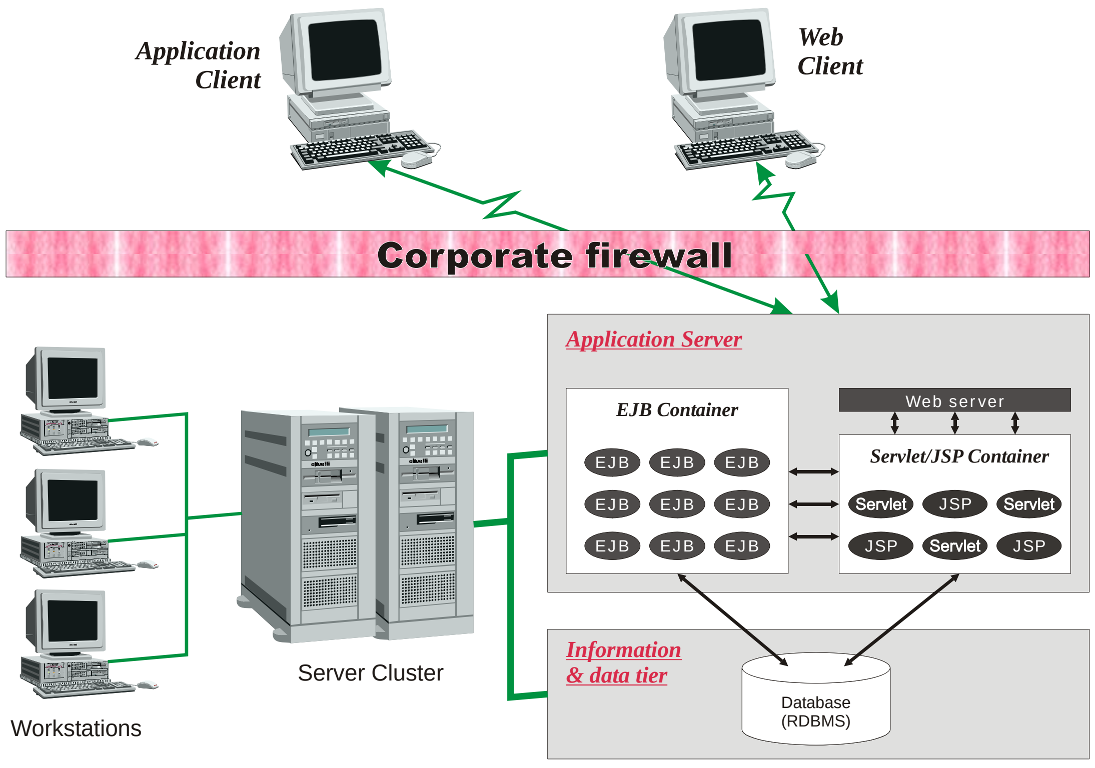


Figure 2-1
Client-server architecture

Figure 2-1 presents a generalisation of the J2EE client-server architecture adopted for the IP and EDA solutions provider. This is a multi-tier solution where the first tier is the client's application environment. In theory the clients can be any application with network/internet access and supporting an appropriate communication protocol such as CORBA. CORBA, standing for common object request broker architecture, is a standard that enables objects from different vendors and from different platforms to communicate and interoperate within enterprise environments. In this thesis the client platform is a schematic capture environment that will be described in Chapter 5. Separating the client and server tiers is a corporate firewall hosted on the server network that ensures secure isolation of the server-side data and services. Behind the firewall is a series of applications supporting the business logic, sub-divided into further tiers, all running on a local area network (LAN). These tiers include web servers, applications servers, database servers, network file servers and optional load balancers for clustering. This project extends this conventional architecture to include workstation resources to host the EDA tool suites required for IP verification and synthesis.

2.1 Programming language

The goal of this research is to develop a framework for a system that would allow access to remote libraries of intellectual property cores across networks or even the internet. Although essentially web-based services, the solution is not provided as a web site. It is a stand-alone executable application that simply uses web protocols to facilitate communication between the client and the server. One of the key objectives is that it is to host a graphical user interface (GUI). To a certain extent this limits the available choice of development languages for the client-side to Visual Basic, TCL-TK, Java or C/C++ and a suitable widget set such as GTK.

UNIX has always been the platform of choice for many ASIC designers. However cost considerations now mean that Linux is rapidly gaining popularity as the prefered operating system. In the FPGA world, the majority of development tools are only available for Microsoft Windows. To fulfil the research objective of platform scalality, the ideal solution will operate on all of these operating systems.

With this in mind, Java was chosen as the most suitable development language due to its portability. Other influential factors include its excellent networking capabilities (which include CORBA), vast resource and API libraries, extensive documentation, its ease of use and of course direct source code compatibility with that used for the J2EE server. This will later prove beneficial as it will allow for a reconfigurable application which, through integrating the server code, can operate independently of the server.

Although generally not renowned for its execution speed, Java has been successfully used in other EDA type applications with acceptable levels of performance [37, 38]. However speed is less of a concern in this project as the main data intensive processing will occur on natively compiled EDA tools installed on the server workstations. Hence extensive networkability is more important than raw processing speed. Again Java is the winning choice due to the architecture of its underlying virtual machine that offers greater scalability, flexibility and functionality than other distributed computing protocols such as remote procedure calls (RPC) [39].

2.2 Application server

As can be seen in Figure 2-1 the application server forms the backbone of the services hosted behind the firewall. An application server is a platform implementing the business logic, information and data processing tiers in typical enterprise scale environments and large web-based projects [40]. These software systems combine distributed computing technologies and object-orientated (OO) techniques with multi-user resource pooling and secure transactions to provide scaleable and portable business systems. To ensure a degree of integration between hosted services, all application servers comply with the Sun’s published specifications – the latest release of which is version 1.4 [41].

All services necessary for the deployment of IP macro-cores across the web were developed as Enterprise Java Beans (EJBs) that were then installed in the application server. A comprehensive guide to the EJB 2.0 framework can be found in the book Enterprise JavaBeans [42] although there is a brief summary in section 3.3.

There are a number of application servers that could be used to service this application such as BEA Web Logic [43], Borland’s Enterprise Application Server (BES) [46] and a freeware offering known as JBOSS [44]. The work documented in this thesis is built around BES, due to its integration with Borland JBuilder Enterprise Edition [45]. BES is a multi-partition application server, meaning that it can serve multiple applications, each of which are referred to as a partition. The work described in this thesis only needs to use a single partition and all relevant EJB packages need to be installed on that partition. Table 2-1 shows a list of all packages that were developed for and installed on the enterprise server.

JAR Name	Size
`Library_Management_EJB.jar`	*662Kb*
`Source_Generator_EJB.jar`	*82Kb*
`Math.jar`	*77Kb*
`File_Management.jar`	*27Kb*

Table 2-1
EJB JARs installed on application server

In addition to those listed in Table 2-1, a number of third party packages are also required. Table 2-2 lists the necessary 3rd party packages that have been used and provides a general description of each of them. As these packages are generic in nature and can potentially be applied to any application they are installed into a different location on the application server. Packages in this location are automatically included in every partition’s classloader.

JAR Name	Purpose
`jakarta-oro-2.0.8.jar`	*Regular expression parsing*
`jdom.jar`	*High-level XML data structures*
`sax.jar`	*Low-level XML reader and writer*
`xerces.jar`	*XML parser*
`mysql-connector-java.jar`	*MySQL database driver for java*
`commons-dbcp-1.1.jar`	*Database connection pooling*

Table 2-2
Required third party JARs

Enterprise scale application servers are extremely complicated pieces of software managing a vast array of processes and services. In order to extract the maximum potential from these tools, they must be carefully optimised and performance tuned. This typically requires dedicated support staff highly experienced with the particular server. They are generally assisted using various J2EE application performance management systems, a number of which are reviewed here [47].

2.3 Backend database

An application server provides a framework for remote clients to access and manipulate information and data on a server, but it does not itself provide any data storage mechanism. A separate persistence layer must be installed and then integrated with the application server. The persistence layer can take many forms, ranging from simple text files to extensible mark-up language (XML) [48] scripts, but generally takes the form of a 3rd party database server. Unlike a typical stand-alone database application such as Microsoft Access, a database server allows multiple users to simultaneously read and write data – complementing the needs of the application server which is designed to service a large number of concurrent users. Communication with database servers is generally by way of the widely adopted structured query language (SQL) [49]. SQL is a script based language that allows applications and users to rapidly access and manipulate the necessary information.

In the domain freeware of database servers there are essentially two dominant players, Postgres [50] and MySQL [51]. Postgres offers superior features and functionality but unfortunately its Java interfacing is poor making it difficult to integrate with the application server. Hence the back-end database used in this research is MySQL.

Both the database server and the application server run as two independent services and can in fact operate on different workstations on the LAN. To enable them to talk to each other, a Java database connectivity (JDBC) driver [53, 54] must be installed. Fortunately an excellent JDBC driver is available for MySQL called ConnectorJ [52] and it integrates well with the application server. Unfortunately though, Borland's Enterprise Server does not support container managed persistence (CMP) with any freeware database server and so all EJBs had to use bean managed persistence (BMP). This makes little difference in terms of functionality but required more effort to develop each EJB as all the database integration logic has to be manually developed.

A brief summary of BMP and CMP EJB’s can be found in section 3.3 with a more comprehensive description of CMP and BMP in [42].

Opening a connection to a database is a computational and resource expensive operation. Each and every query and update issued to the database requires its own dedicated connection. This is a well known bottleneck that has a drastic effect upon performance. To compensate, a database connection pool is generally used. This is another layer that sits between the JDBC driver and the database server. The connection pool maintains a number of open connections to the database. When the JDBC driver requires a new connection it simply asks the pool manager to grant it one. As the connection is always open, a significant amount of time and resource is saved leading to much improved performance. The connection pool adopted for this project is DBCP [55].

2.4 CAD tools

The application server and database serve merely as a framework allowing transaction based integration between clients and server applications. One of the fundamental goals is to allow digital designs to be functionally verified and synthesised on the server. This means that the appropriate EDA tools must be installed on the server and integrated with the application server. For verification a fast and efficient simulation engine called Icarus Verilog [56] was used, however alternatives such as Verilog-XL and NC-Verilog from Cadence can also be used. Synthesis was generally performed using either Synopsys’ DesignCompiler or Cadence’s BuildGates.

2.5 Deployment

To deploy this service the server must be suitably configured, a process known as hosting, and client executable applications must be compiled. These processes are described in the following sections.

2.5.1 Hosting

For the server to be accessible to client applications it must be suitably hosted. The host can be a single workstation, but considering the nature of this project, a hosting cluster is more appropriate. A hosting cluster is a LAN of workstations dedicated to the purpose. In such an arrangement at least one machine is dedicated to each of the server tasks. One machine would host the application server with another for the database server. Several additional machines would also be needed to farm out design verification and synthesis jobs. If deemed necessary slave servers could also be installed providing a redundant backup and facilitating comprehensive fail-over support. If 24/7 uptime must be guaranteed, having redundant servers eliminates issues associated with single-point-of-failure. Should one of the main servers overload or crash, a slave mirror will automatically take over maintaining seamless interaction with the client application. The majority of application servers and database servers have built-in support for clustering, slave servers and automatic failover [57].

Configuration and maintenance of a cluster is considered outside the scope of this research and therefore all above mentioned applications have been installed on a single workstation. However this arrangement is only sufficient to support a very small number of concurrent clients, perhaps 3 at most.

2.5.2 Client application

Deploying the service to clients simply involves providing them with a suitably packaged executable application. Having been written in Java, the client application must be packaged into a series of JAR files. JAR files are java archives containing all the byte-code compiled classes associated with the project. A list of the JARs developed for the client applications is given in Table 2-3. A full list of all java source files used in both the client and server can be found in appendix A.

JAR name	Size
category_stubs.jar	*213Kb*
category_manager.jar	*149Kb*
corecompositor_stubs.jar	*241Kb*
filemanagement.jar	*27Kb*
icon_images.jar	*131Kb*
ip_shell_client.jar	*605Kb*
librarymanagement.jar	*609Kb*
librarymanagement_stubs.jar	*219Kb*
librarymanagementinterfaces.jar	*245Kb*
math.jar	*77Kb*
sourcegeneratorinterface.jar	*377Kb*

Table 2-3
Client JARs

Along with the JARs developed as part of this project, some 3rd party JARs also need to be provided to clients. The necessary 3rd party JARs for the client side are the same as those listed in Table 2-2 with the exception of the MySQL database driver package.

All of the above mentioned packages are necessary for the client application to perform its required functionality. However none of these provide any support for client – server communication and integration through CORBA. For this several additional JARs need to be packaged along with the client application. These JARs are specific to the particular application server being employed. For BES a list of the necessary JARs can be found in Table 2-4. These packages provide support for EJB container interfacing, implementations of communications protocols, secure data transmission (encryption) etc.

Borland Enterprise Server JARs for client support
asrt.jar
beandt.jar
dx.jar
lm.jar
vbejb.jar
vbjorb.jar
vbsec.jar
xmlrt.jar

Table 2-4
List of BES client-server integration JARs

2.6 Summary

This chapter presented an overview of the client-server architecture and deployable services based upon the J2EE standard. This standard uses CORBA as the communication protocol and so also has to be supported by the client application. To simplify overall development Java was adopted as the programming language for client application as some code can be shared with that of the server. Enterprise scale systems can be defined as a collection of server-side services. An application server is the main gateway for the client to access these services and co-ordinates all of the business logic. This particular application requires an SQL database and EDA tool suite as constituent parts of the enterprise layers. Choices and justifications for the products implementing these constituents was presented. The same architecture can be realised using different products, possibly leading to variations in performance and scalability. Quite probably the largest variation would be seen using a combination of application server and database that allows for container managed persistence meaning more scope for optimisation.

Chapter 3
Prerequisites

This chapter presents a general overview of some of the practices and principles of software design that have been applied in this research. These concepts have been described in adequate detail here and will be referred back to in later chapters.

3.1 Design patterns

At the heart of any software system is the data that it manipulates. In fact it can be argued that the sole purpose of computer software is invariably data processing. Hence it is vital to try and model the data in such a way that it leads to the optimal performance of the surrounding software system. It was this need is what spurred the creation and rapid adoption of object-orientated (OO) design software platforms like Smalltalk, C++ and Java. As OO software methodologies became widely adopted, several reoccurring design problems became obvious. In 1995 the book "*Design Patterns: Elements of Reusable Object-Orientated Software*" [58] published 23 patterns that allowed for the creation of more flexible, elegant and ultimately reusable designs. Independent studies have since shown that in general it is beneficial to use these design patterns over simpler methods even when the actual design problem is simpler than that solved by the pattern [59, 60]. Many of these patterns have been successfully employed in this application, with a description of the most significant ones given in the following sub-sections.

3.1.1 Façade design pattern

One of the key novelties of the system developed in this project stems from employing the Façade design pattern. In simple terms the façade pattern is a single class that masquerades the functionality of several sub-systems through a unified interface. Example sub-systems could be persistence layers. While not being processed, the data samples need to be stored to disk. However there are many forms in which the data can be persisted. For a web-based system backed by an application server, the persistence layer in the eyes of the client is by way of Enterprise Java

Beans. However the EJBs themselves are not data persistence objects but merely gateways to underlying databases. For stand-alone applications, the persistence layer will be the local or network file system where data can be stored in either ASCII or binary forms. Hence the point being emphasised here is that the same data can be and may need to be persisted in several different forms. In fact all data models in this project can be stored in 4 forms, namely serialized straight to disk, in proprietary XML, to an SQL complaint database or by way of EJBs – which may use either SQL or XML at their backend. In the façade pattern, an interface is created defining the functionality of the persistence layer in terms of method calls. Hence rather than referring to the actual object responsible for reading and writing to the appropriate persistence layer, the application will converse with the interface. The interface will then reference an instance of the object responsible for the correct persistence type. The actual persistence object will generally extend an abstract class that will implement the functionality common to all persistence methods.

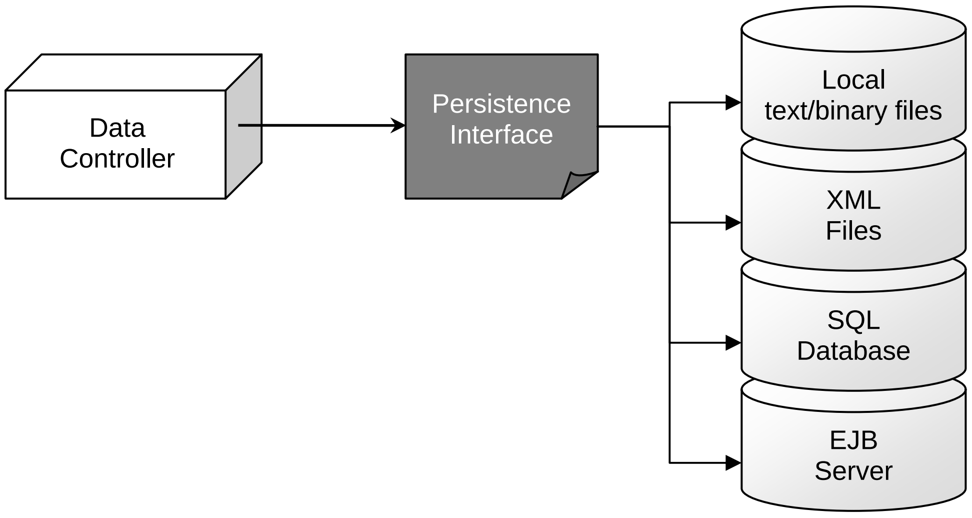


Figure 3-1
Façade design pattern applied to data persistence

Figure 3-1 shows how the Façade design pattern has been applied to the task of data storage. The client application may need to store data in one of many locations depending upon its current run-time condition. While downloading data from the server, it will need to interface with the EJB container. But to conserve memory on the client workstation, it may be necessary to temporarily flush the data to local storage such as its hard disk. To save and reload the data the persistence mechanism must change, for example to an XML description. As this project makes extensive use of the MVC design pattern, all data has a corresponding controller. One of the tasks of the controller is to interact with the appropriate data storage mechanism. The controller will maintain a single instance of an object that implements the storage interface. From Figure 3-1 it can be seen that four objects implement this interface, one for each type of data storage. Adapting the controller for a new type of data persistence layer is simply a matter of updating the storage object instance variable.

3.1.1.1 Stand-alone client application

Through extensive and intelligent use of the façade design pattern, the client application has been made dynamically reconfigurable to operate in one of several environments. Its primary mode of operation is in client-server configuration where the façade effectively hides the server from the client. In another mode, the client can operate independently of the enterprise application server interacting directly with EDA tools available on the LAN and obtaining its IP catalogue information straight from the database via JDBC. This was easily possible as the database integration logic was readily available as the EJB persistence model uses BMP as opposed to CMP. In its 3rd mode, the IP catalogue information can be sourced from locally available XML files.

3.1.2 Observer design pattern and model-view-controller

A popular and well known data interaction paradigm in object-orientated software is the model-view-controller (MVC) architecture [61]. Originally developed for the Smalltalk-80 language, the MVC architecture ensures that the data (model), the data manipulators (controller) and data visualisations (view) are all modularised into separate entities. In simple software applications these three functions are typically combined inside a single object. For example, a GUI panel used to set a series of values will maintain corresponding instance variables for those values. Interactive manipulation of the widgets on the panel will then directly control these values and update the view in the panel. However this approach has several drawbacks. Debugging the application is considerably more difficult because the code maintaining data integrity is mixed with that necessary for visualisation and control. Additionally the versatility, adaptability and scalability of the application will be severely restricted.

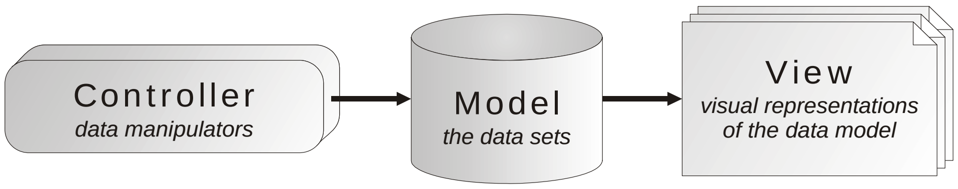


Figure 3-2
Model-View-Controller architecture

Figure 3-2 above depicts the relationships between the model, view and controller. In the MVC implementation, the model refers to the actual data to be manipulated by the software. Generally there is a single model class in the MVC architecture, although multiple domain models have been implemented with varying degrees of success [62]. A custom class will be created with the appropriate instance variables and corresponding get/set methods for the particular data in question. Each object instance of this class will then represent one piece of data. These data objects are made *observable* (by extending the ***`java.util.Observable`*** class), meaning that they will notify all *observers* (objects that implement the ***`java.util.Observer`*** interface) of any changes that are made to them, in accordance with the observer design pattern. An observer is any 'listener' whose purpose is to give a visual representation of the data and hence implements the 'view' aspect of the MVC. This can for example be a panel giving a text description of the data, or maybe a pie chart or any other means of data presentation. Each model can have an unlimited number of views listening to it, meaning that several GUI panels can be updated immediately any changes to the data take place. This loose coupling mechanism defined by the observable design pattern ultimately allows for optimal flexibility as additional views can be added as and when needed. If this architecture is strictly adhered to, the only means of modifying the data is by way of the controller. The controller is typically a singleton object and implements all the necessary logic to process the user input.

3.1.3 Singleton design pattern

A singleton object is the sole instance of a particular class. There is no public constructor and the only means of obtaining an instance of it is via a public static method, usually called `getInstance()`. Calls to the get instance method will return the single instance object of that class, creating it if necessary using a private constructor.

3.1.4 Factory design pattern

The factory design pattern enables the type of object sub-class to be determined at application run time. This has been extensively used for component modelling and symbol schematic generation. For example if the user wishes to insert a FIFO into their design, the factory will create a FIFO symbol object enabling it to be correctly rendered on the drawing area.

3.2 Serialization

Due to the web-based nature of the software that has been developed, all data model objects had to be designed in such a way that they can be sent across networks. In java this means that all data models must have the ability to be directly translated into a sequential data stream. This is a process known as serialization and can be applied to any objects that implement the ***`java.io.Serializable`*** interface. However for an object to be truly serializable, all other objects stored or referenced in instance variables within that object must also be serializable. Unfortunately this relegates all GUI classes in the ***`javax.swing`*** package, as none of them implement the serializable interface. You may think that this will not be a problem as long as we don't store references to any swing components. But remember from section 3.1.2 that all these models adhere to the MVC architecture and all view components registered with a data model are typically swing components. Hence attempting to directly serialize these data models to a stream will almost always fail. One potential, but not very elegant, solution to this problem would be to delete the references to all observers before serialization and subsequently them reinstate them.

The chosen solution for all data models that need to be serialized is to maintain two independent instances of them. The main instance would integrate with the MVC environment while the second would only maintain a copy of the original's data excluding references to non-serializable objects. On occasions when information has to be uploaded to the server, the data values in the clone would be updated to match those of the master copy and it would then be serialized and transmitted. If the information is modified by the server, the data values in the client's master copy would then be updated after it is downloaded again.

3.3 Enterprise Java Beans (EJB)

Enterprise Java Beans, first introduced in 1997, is a means of abstracting data and transactions which are based on distributed object technologies. At the time this technology saw the convergence of transaction processing monitors and distributed object services including CORBA and RMI. Now EJB has established itself as one of the most important enterprise technologies, becoming a de facto standard with the release of version 1.0 in 1998. The application described in this thesis complies with latest specification – that being version 2.0. EJBs are server-side component models that operate within the confines of a dedicated container, generally called an application server. This application server is the runtime environment for EJBs and is responsible for providing life-cycle management, accounting for transactions, persistence, concurrency and security. There are essentially two types of EJB – session and entity.

3.3.1 Entity EJB

An entity EJB is essentially an interface to a set of data that defines a concept representing the information tier in an enterprise system. On the server there exists an entity EJB for each and every object of information – generally meaning a record in a database. Within the context of this project an entity could be a single IP macro-component. There are two persistence models for entity EJB – container managed and bean managed.

3.3.1.1 Container managed persistence (CMP)

CMP EJBs rely upon the container to implement the data-access calls to their persistent data sources. Information about the persistence layer is supplied to the container when the beans are deployed. This greatly simplifies the EJB development process and extends its portability. If for example the back-end persistence layer changes, perhaps from one database vendor to another, only the container needs to be updated while the EJBs remain the same.

3.3.1.2 Bean managed persistence (BMP)

With BMP the bean developer must write the persistence layer interaction code into the bean itself. This is generally by way of JDBC calls to the underlying database and requires more effort from the developer's point of view.

3.3.2 Session EJB

Session EJBs are generally used to provide access to server-side resources including entity EJBs and workflows. A workflow is a way of describing a task to be performed in the server. An example workflow could be performing verification or synthesis on behalf of a client. There are essentially two forms of session bean – stateful and stateless.

3.3.2.1 Stateless

A stateless session bean maintains no state between its method calls. It executes a single method and returns it result without affecting the state of the object – as it has no state. These beans also tend to be general purpose and are not dedicated to any one particular client.

3.3.2.2 Stateful

Stateful beans are dedicate to a particular client and maintain conversational state – meaning they are objects that can hold instance variable that can be set by the client. However these variables only have the lifespan of the bean itself, meaning they are not written to a persistence layer when the bean is no longer needed.

3.3.3 EJB container

Clients interact with EJBs by way of the container. When attempting to access an EJB, the container will return a remote reference to that EJB. The client can then invoke methods in that remote reference, which the container then delegates to the EJB implementation.

3.4 Transactions

As will be demonstrated in Chapter 4 and Chapter 6, the server-side data is stored in the database across multiple tables. This presents a potential opportunity for data corruption as a write operation may update fields in one table and then fail before the corresponding fields in another table are revised. To circumvent this, the use of transactions has been integrated at the database level. Transactions wrap multi-table data modification operations into a single atomic operation. Within a transaction based environment, individual updates and inserts are sent to the database server and then ‘committed’ if all constraints are met, or ‘rolled-back’ if a failure occurs. A typical failure would be an invalid foreign key constraint. For a database to correctly support transactions, it must be ACID compliant – Atomicity, Consistency, Isolation and Durability [63]. ACID transactions are supported in MySQL if InnoDB table types [64] are used.

Within typical J2EE environments, atomicity and transaction support is implemented at the application server container level using container managed persistence (CMP). Unfortunately Borland's Enterprise Server does not support CMP with non-commercial databases, such as MySQL, and transaction management had to be manually designed in using bean managed persistence (BMP).

3.5 Regular expressions

Several parts of this project necessitate the parsing of data from text files. An example of this is extracting module structure information from Verilog source files. The approach adopted was to use regular expressions [65]. Regular expressions are a type of formula for matching strings that follow some sort of pattern. The regular expression will use a series of normal characters and metacharacters to describe the pattern to be captured. Normal characters are numerical digits and upper and lower case letters. Metacharacters are characters and sequences of characters that have special meanings. A full description of regular expressions is beyond the scope of this text, but a example will be provided as follows. To find a floating point number the following regular expression could be used:

```
"([-]?[0-9]+[.][0-9]+)"
```

This statement means that the minus sign is optional, as denoted by the question mark. A floating point number is then made of one or more characters from 0 – 9 followed by a period and then a second group of numerical characters.

This research has been developed using the Java 1.3.1 platform, which has no direct support for regular expressions. Hence regular expression functionality was incorporated using a third party package called Jakarta ORO [66]♣. As of Java version 1.4.1, regular expressions have become a natively supported feature.

♣ The Jakarta license can be found in Appendix D.

3.6 Summary

This chapter presented some of the general concepts and techniques applied throughout the thesis. Rather than repeatedly describe them throughout the body of the text, they have been described here and back references made where necessary.

One of the strategies leading to a highly flexible and scaleable platform stems from the use of design patterns. From the 23 published patterns, 4 have been extensively applied, these being the façade, MVC, singleton and factory design patterns.

All services on the enterprise application server are constructed from EJBs. A very general overview of the main EJB technology incorporated in this work was presented. CMP is the preferred means of entity management as it simplifies bean development. BMP involves manually crafting the EJBs database transaction logic in the java source files. Using BMP means that the database integration classes can also be made directly accessible to the client, allowing it to be reconfigured in a stand-alone mode through design pattern implementation.

Chapter 4
Server-side IP-libraries

Section 1.3.7 highlighted the fact that the current range of macro-components has been divided into four IP libraries. To integrate these libraries into an EDA platform their contents must be carefully structured and catalogued.

4.1 Library file structure

An IP library is a location on the file system containing the home directories of all macro-components belonging to it. The structure of a library and its macro-components must be strictly defined to ensure seamless integration with the appropriate server beans, which have been architected around this topology.

4.1.1 Macro-component directory structure

Figure 4-1 shows the directory structure naming conventions imposed upon the macro-components. Each macro-component resides within its "module home" directory under which all its supporting files are located. The structural hierarchy of a module home is depicted in Figure 4-1. Nested inside a module home are a further four directories.

1. RTL

   The RTL directory houses the Verilog source file and optional dependency link file. To be compliant with the QIP standard defined by the VSIA organization [9] the Verilog source file must have the same name as the top module defined inside it. If the module instantiates dependency sub-modules, a link file called `verilog_dependency.lst`, must also be provided in the RTL directory. These link files will be described in section 4.2.

2. Simulation

   The simulation directory contains the necessary routines to functionally verify the macro. This will include a Verilog testbench and optional input data files. A golden output reference file can also be provided against which the simulation results can be compared.

3. Synthesis

   The syn directory maintains all files related to the synthesis procedure.

4. Documentation

   The documentation directory, called doc, contains any documentation associated with the core in PDF and XML forms.

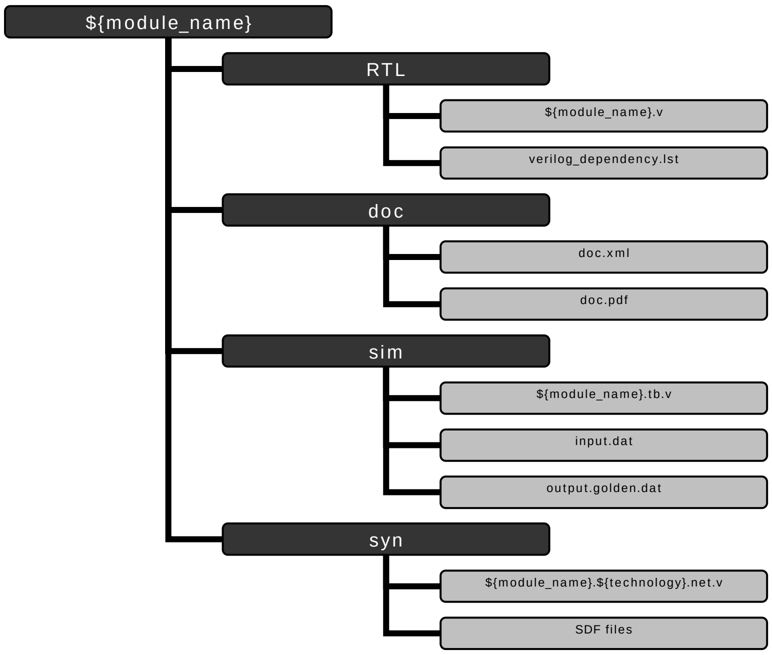


Figure 4-1
Directory and file structure of macro-component

4.1.2 Library manager

During runtime operation the server needs to know the location of the IP libraries and the macro-component file system topology defined in section 4.1.1. This information can be hard-coded into the appropriate EJBs, but doing so would limit the flexibility, scalability and portability of the application. Hence a system based upon a dynamic library manager utility was investigated. The library manager is built upon the singleton design pattern described in section 3.1.3, making it an independent entity available at all points in the application without the need to propagate references to it. In fact this arrangement also makes it independent of the enterprise server itself. It can therefore also be called directly from the schematic capture environment allowing the IP library path and structure information to be available without the need to query the application server. This is one of the key characteristics of the software enabling standalone use of the design environment independent of the server. Figure 4-2 shows how the client circuit development environment makes use of the singleton design pattern in conjunction with the façade design pattern, introduced in section 3.1.1, to facilitate both client-server and standalone configurations.♣

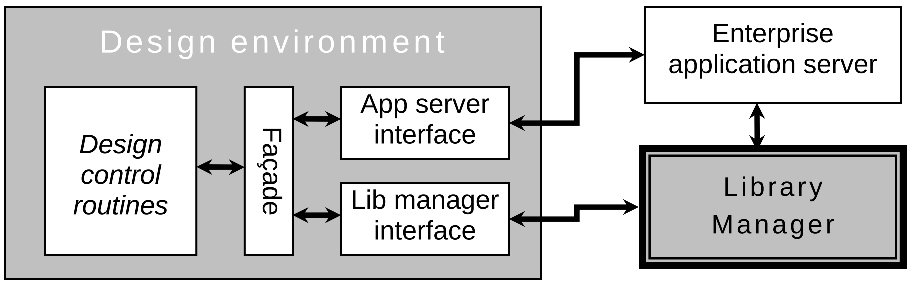


Figure 4-2
Library manager interaction using the façade design pattern

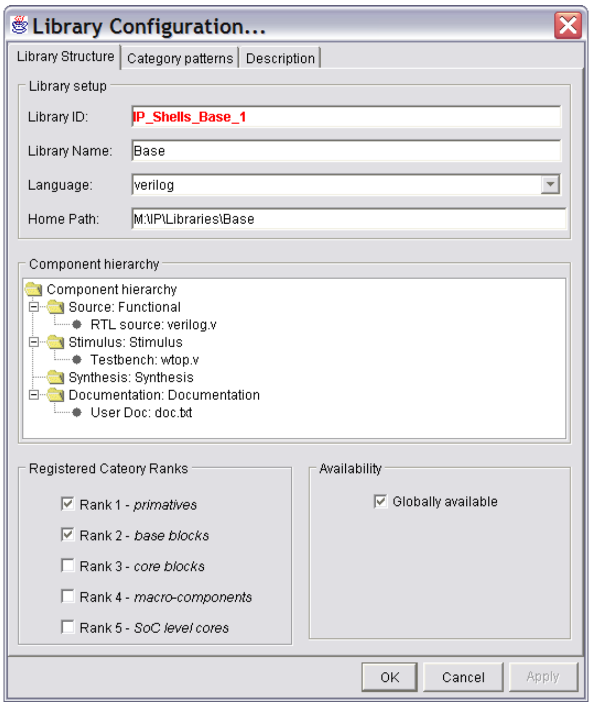
The task of the library manager is to cache the paths to all installed IP libraries and the topology of the macros in each library. Figure 4-3 shows the database schema used to cache the appropriate library information.

```
+------------------+--------------------+------+-----+---------+-------+
| Field            | Type               | Null | Key | Default | Extra |
+------------------+--------------------+------+-----+---------+-------+
| Library_ID       | varchar(50) binary |      | PRI |         |       |
| Library_Name     | varchar(50) binary |      | UNI |         |       |
| Available        | tinyint(1)         |      |     | 0       |       |
| HDL              | varchar(20)        | YES  |     | verilog |       |
| Library_Path     | varchar(255) binary| YES  |     | NULL    |       |
| Source_Path      | varchar(255)       | YES  |     | NULL    |       |
| Source_File      | varchar(255)       | YES  |     | NULL    |       |
| Sub_Modules_File | varchar(255)       | YES  |     | NULL    |       |
| Stimulus_Path    | varchar(255)       | YES  |     | NULL    |       |
| Stimulus_File    | varchar(255)       | YES  |     | NULL    |       |
| Test_Bench_Path  | varchar(255)       | YES  |     | NULL    |       |
| Test_Bench_File  | varchar(255)       | YES  |     | NULL    |       |
| Doc_Path         | varchar(255)       | YES  |     | NULL    |       |
| Doc_File         | varchar(255)       | YES  |     | NULL    |       |
| Description      | text               | YES  |     | NULL    |       |
+------------------+--------------------+------+-----+---------+-------+
```

Figure 4-3
Database schema for IP library information

♣ Note that the design environment can only be used in one configuration at a time. It must be launched in the appropriate manner to be used in either client-server or standalone configurations.

A screen shot of an application developed to configure an IP library can be seen in Figure 4-4. This is part of the IP cataloguing application that will be described in section 4.5.6



Library setup

Library ID: IP_Shells_Base_1

Library Name: Base

Language: verilog

Home Path: M:\IP\Libraries\Base

Component hierarchy

- Component hierarchy
  - Source: Functional
    - RTL source: verilog.v
  - Stimulus: Stimulus
    - Testbench: wtop.v
  - Synthesis: Synthesis
  - Documentation: Documentation
    - User Doc: doc.txt

Registered Cateory Ranks

- [✓] Rank 1 - *primatives*
- [✓] Rank 2 - *base blocks*
- [ ] Rank 3 - *core blocks*
- [ ] Rank 4 - *macro-components*
- [ ] Rank 5 - *SoC level cores*

Availability

- [✓] Globally available

Figure 4-4
Library manager configuration window

To maximize portability the paths to all macro-components are defined relative to the library to which they belong. When the path to a macro is needed, as will be the case for verification and synthesis, a request will be sent to the library manager application. The library manager caches the paths to all libraries using a set of keys to identify each library in a look-up table. The library manager start-up configuration file, for the libraries described in section 1.3.7, can be seen in Figure 4-5.

```
#------------------------------------------------------------------
# Library paths configuration file
#

sg_foundation_1                = M:/Libraries/foundation
sg_foundation_cache_1          = M:/Libraries/foundation_cache
sg_signal_processing_1         = M:/Libraries/signal_processing
sg_signal_processing_cache_1   = M:/Libraries/signal_processing_cache
sg_communications_1            = M:/Libraries/communications
sg_communications_cache_1      = M:/Libraries/communications_cache
sg_image_processing_1          = M:/Libraries/image_processing
sg_image_processing_cache_1    = M:/Libraries/image_processing_cache
```

Figure 4-5
Library path configuration file

4.2 Module dependency handling linker

A fundamental characteristic of the macro-component libraries is the inherent hierarchical structure. There is a great deal of interdependency between the various macro-modules in each library and the corresponding caches, and also between the high-level libraries and the underlying foundation library. It is however this hierarchical relationship that gives these libraries their flexibility and low-power characteristics. Smaller macros are used as the building blocks for the larger cores. Low power optimizations in sub-blocks are inherited by the large blocks incorporating them.

The architecture of the IP server must be able to cope with this hierarchical library structure. If a user builds a design on their client application and decides to functionally verify it, the server must be able to locate and link together all the necessary dependency modules. Commercially available EDA integrated development environments (IDE) use their own internal proprietary linking mechanisms. HDL developers relying on the command-line for compilation and synthesis typically use various shell scripting languages such as Perl, Make, TSCH, BASH or TCL. Although very capable in their own right, they are not directly suitable for this project as they are not directly supported by the application server.

```
sg_signal_processing_cache_fft_1    sg_commutator1_fft64p_t_4
sg_signal_processing_cache_fft_1    sg_butterfly_fft_tpo_radix4
sg_signal_processing_cache_fft_1    sg_modcmultiplier_fft_booth_wallace_16
sg_signal_processing_cache_fft_1    sg_commutator2_fft64p_t_4
sg_signal_processing_cache_fft_1    sg_multless_fft64p_po_16
sg_signal_processing_cache_fft_1    sg_commutator3_fft64p_a_4
sg_foundation_1                     sg_flipflop_d
```

Figure 4-6
Sub-module dependency list file for an FFT

A custom linking application has been developed for this project based on very simple dependency script files like the one in Figure 4-6. These scripts contain a single row entry for each *direct* dependency of the macro-component. Each entry defines the name of the dependency module and the library ID of the library it belongs to. The linker application reads these files and queries the library manager to build the corresponding dependency tree. Entries in the dependency file represent branches in the tree. After resolving the path to a dependency, dependencies of that dependency are also resolved and added as sub-branches to the tree in a recursive manner.

Figure 4-7 shows the dependency hierarchy for the FFT whose dependency file is shown in Figure 4-6. As can be seen in the figure, all direct dependencies are in the first row of the graph. Subsequent rows in the graph represent the rest of the hierarchy for the entire FFT architecture.

This hierarchical linking mechanism makes this tool much more powerful and scaleable than traditional script-based design practices. Referring to the FFT example in Figure 4-7, the performance of the architecture can be radically changed by substituting the multiplier with an alternative from the library. This change can be implemented by simply replacing the `sg_modcmultiplier_fft_booth_wallace_16` with the new module and updating the single entry one dependency file, a process which only takes a few minutes. The design can then be re-verified and synthesised. If the FFT happens to be part of a larger system, typical applications including OFDM and CDMA, the change will be automatically adopted. Obviously this will have implications for version control and regression testing, but this is outside the scope of this thesis.

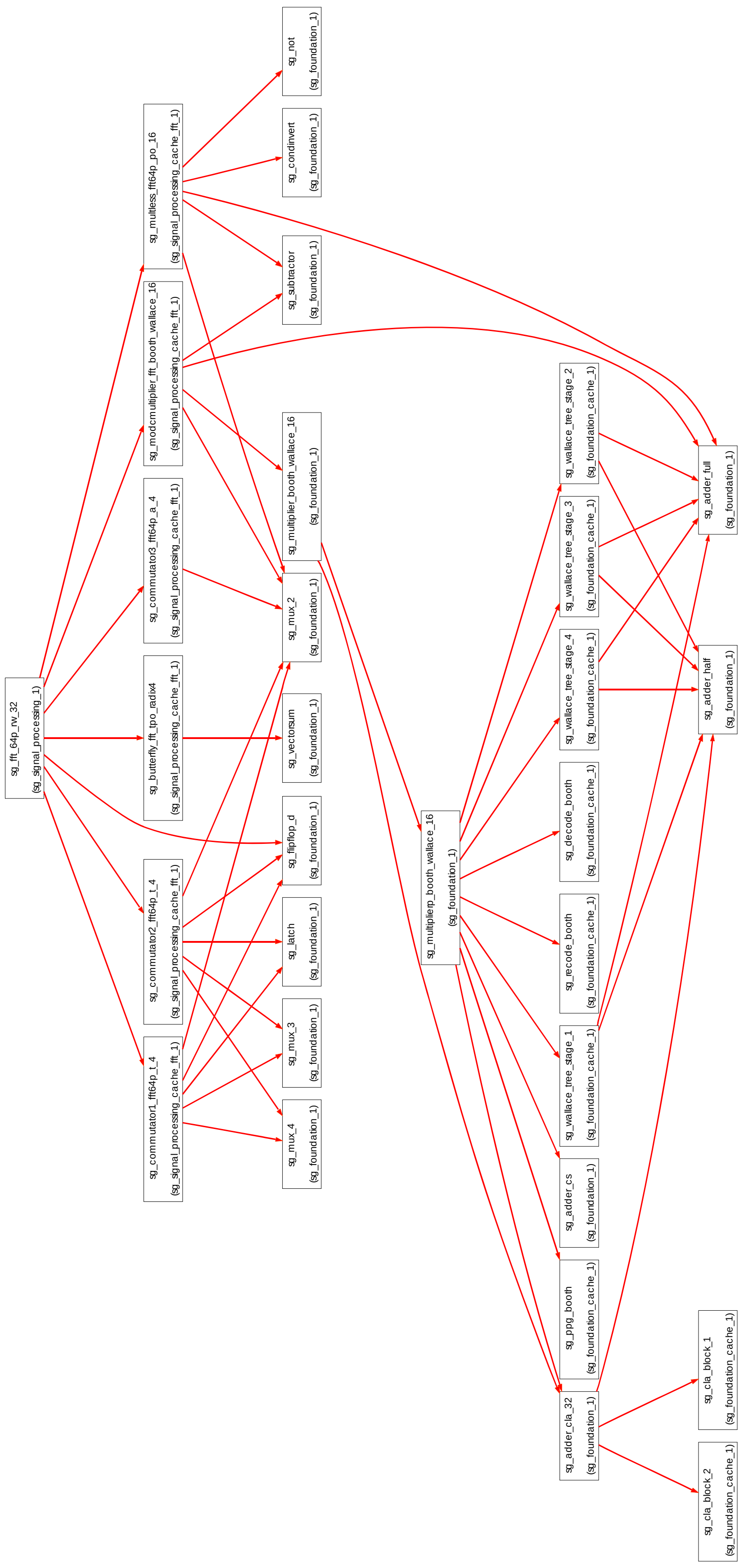


Figure 4-7
Dependency hierarchy for an FFT

4.3 Component cataloguing

As clients use the service they will want to know what macros are available and what their interfaces look like. It is possible to build this information from the file system as requests are made, but it would waste significant resources to do so. The optimal strategy is to catalogue all relevant information into a database which will then be interrogated, via the application server, by client requests. The following sections describe the information that has been catalogued.

4.4 Component categories

Typically when an IP vendor releases a library, it will simply be a large collection of component cores, possibly catalogued in a manner similar to the library structure defined in section 4.1. One of the problems with this arrangement is that it can be difficult to determine the nature of each library component without continually referring to the accompanying documentation. For example, if the system designer wishes to instantiate a multiplier complying with a particular specification they need to be able to identify and discriminate between all available multipliers in the source library. Sometimes depending upon the naming convention adopted by the vendor, it can be possible to guess the type of component based upon its name, for example, all multipliers may have a name starting with “mult”. Some of the currently available EDA tools attempt to pick up on this and make a rudimentary attempt to classify components based upon their names.

One of the novel aspects of this EDA solution is the inherent ability to allow for the classification of components into distinct categories. A category is essentially a template defining the interfacing characteristics of a type of component. Given an IP library, it may for example contain several multipliers. A multiplier has a well-defined purpose – take two values and multiply them together to produce a resultant product. However, architecturally a multiplier can be realised using a number of different algorithms, such as Booth, Wallace array etc., to achieve certain goals such as speed and performance, reduce power consumption or minimise the silicon area footprint while retaining the functional specification. Neglecting the internal architecture, the functionality and interface to the component will remain constant, that is it will have two inputs, a carry in and an output product. This being the case, it is theoretically possible to simply replace a multiplier in a design with any other multiplier in the library, providing of course that the design specification can still be met. With this in mind, a multiplier *category* can be defined to represent all multipliers in the available libraries. The same principle holds true for other types of components such as adders, counters, FFTs and receivers to name a few.

4.4.1 Category modelling

Categories are globally defined entities meaning that they are not specific to individual libraries. Being general-purpose component type descriptions, it makes sense to define them only once and have them apply to any and all available libraries. Hence several libraries may contain components conforming to the same category or conversely a given category type may only reside in a single library.

Within the EDA environment, a category has been modelled using a category model object. Figure 4-8 shows a simplified UML diagram depicting the adopted models for a category and its associated ports. Every category has a name by which it is referenced, for example, a suitable name might be "`multiplier`". As the name acts as the primary key in a category search, it is important that all category names are unique. In addition to the name, the category model also houses an array of objects defining the characteristics of all available ports. The characteristics of the ports will be described in the following sections.

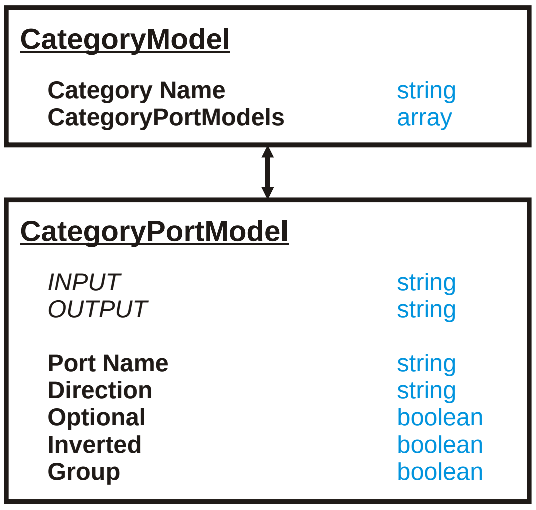
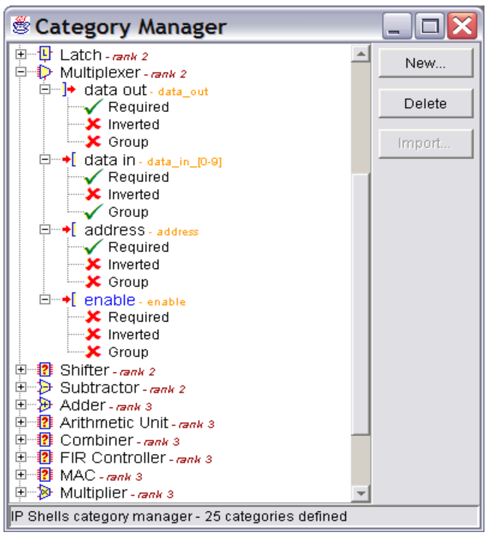


Figure 4-8
Simplified UML diagram of a CategoryModel and CategoryPortModel objects



```
Latch - rank 2
Multiplexer - rank 2
    data out - data_out
        ✓ Required
        ✗ Inverted
        ✗ Group
    data in - data_in_[0-9]
        ✓ Required
        ✗ Inverted
        ✓ Group
    address - address
        ✓ Required
        ✗ Inverted
        ✗ Group
    enable - enable
        ✗ Required
        ✗ Inverted
        ✗ Group
Shifter - rank 2
Subtractor - rank 2
Adder - rank 3
Arithmetic Unit - rank 3
Combiner - rank 3
FIR Controller - rank 3
MAC - rank 3
Multiplier - rank 3
IP Shells category manager - 25 categories defined
```

Figure 4-9
Screen grab of category manager application

4.4.2 Category port modelling

The category port model is used to describe the characteristics of each port on a component conforming to a particular category. The model has five variables that describe a port and any constraints to which it must comply. These are described as follows.

- **Name**

  Every port in a category model is identified by its name and as such each port defined in a category must have a unique name attribute. Although there is no strict guideline on naming conventions, it is recommended that the chosen port name reflects its intended purpose. For example a port acting as the carry out of a category of component should preferably be called "`carry out`".

- **Direction**

  A port is basically a interconnect interface that allows for the propagation of signals in a pre-determined direction. Permissible port directions are *input* and *output*. Provision has also been made for *inout* ports, however these are unlikely to ever be used as they cannot always be directly synthesized.

- **Optional**

  To allow for a degree of flexibility in the definition of categories, it is possible to specify some ports as being optional. An example of an optional port is the enable signal on a multiplexer. Rather than having to define separate categories for multiplexers with and without enables, it is more convenient to define a single category which has one or more optional ports.

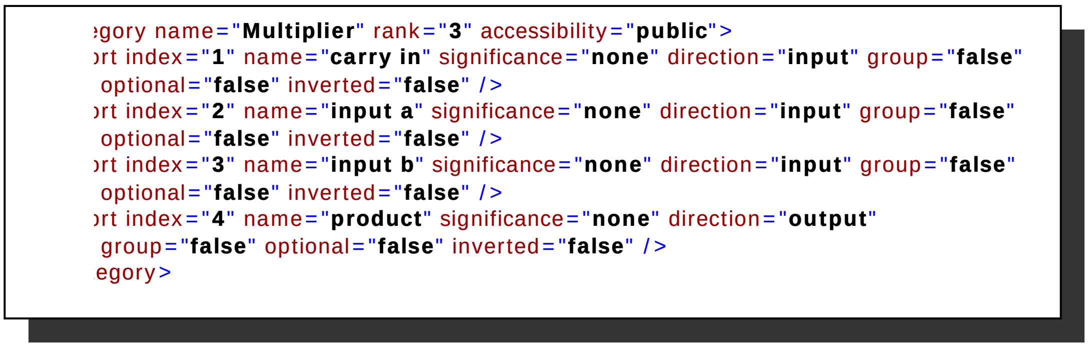
- **Inverted**

  Inverted ports are normally denoted on schematics through the use of a circle on the port symbol. A classic example of an inverted port is the output from a NAND gate

- **Group**

  A category does not strictly define the number of ports a component can have. An example of such a component with an unknown number of ports is a multiplexer. The number of data input ports on a multiplexer can theoretically range from 2 to any positive integer. Rather than define separate categories for each multiplexer size, it is better to specify a single port model representing all the data inputs and assign a *group* attribute.

Figure 4-9 shows a snapshot of a GUI application that allows categories to be created and visualized. This application builds upon the MVC architecture and can import and export the category information to a local XML file, directly into the database or through the EJB container using the façade design pattern. Figure 4-10 shows the XML representation of the category model for a multiplier. There will be a separate category entity for each type of component.



```
ɔgory name="Multiplier" rank="3" accessibility="public">
ɔrt index="1" name="carry in" significance="none" direction="input" group="false"
 optional="false" inverted="false" />
ɔrt index="2" name="input a" significance="none" direction="input" group="false"
 optional="false" inverted="false" />
ɔrt index="3" name="input b" significance="none" direction="input" group="false"
 optional="false" inverted="false" />
ɔrt index="4" name="product" significance="none" direction="output"
 group="false" optional="false" inverted="false" />
egory>
```

Figure 4-10
XML model of a multiplier category model

For persistence using either the EJB container or directly to and from the database, the database schemas of Figure 4-11 and Figure 4-12 have been implemented.

```
+-----------------+-------------------------------+------+-----+---------+----------------+
| Field           | Type                          | Null | Key | Default | Extra          |
+-----------------+-------------------------------+------+-----+---------+----------------+
| Category_ID     | int(11)                       |      | PRI | NULL    | auto_increment |
| Category_Name   | char(50)                      |      | UNI |         |                |
+-----------------+-------------------------------+------+-----+---------+----------------+
```

Figure 4-11
Database schema for category model

```
+-----------------+-------------------------------+------+-----+---------+----------------+
| Field           | Type                          | Null | Key | Default | Extra          |
+-----------------+-------------------------------+------+-----+---------+----------------+
| Category_Port_ID | int(11)                      |      | PRI | NULL    | auto_increment |
| Port_Type       | char(25)                      |      |     |         |                |
| Port_Direction  | enum('input','output','inout') | YES |     | NULL    |                |
| Collection      | tinyint(1)                    |      |     | 0       |                |
| Optional        | tinyint(1)                    |      |     | 0       |                |
| Inverted        | tinyint(1)                    |      |     | 0       |                |
| Category_ID     | int(11)                       |      | MUL | 0       |                |
+-----------------+-------------------------------+------+-----+---------+----------------+
```

Figure 4-12
Database schema for a category port model

4.5 Component model

To capture the characteristics and attributes of the individual macros, a component model has been developed. This component model maintains all information about a module that will enable it to be used as a black-box object in a client design. This means that all input/output (I/O) ports and parameterisation values are included in the model.

Figure 4-13 shows the various interactions of the component model, which is designed in accordance with the MVC architecture. The supported persistence layers are XML, EJB via the application server and direct database integration. The original component model is derived using an application that parses the Verilog RTL source file describing the module. When a client wishes to incorporate a module into their design, they do so by download the appropriate component model from the server. A detailed UML diagram showing the component model class and all its interactions can be found in Figure 4-22.

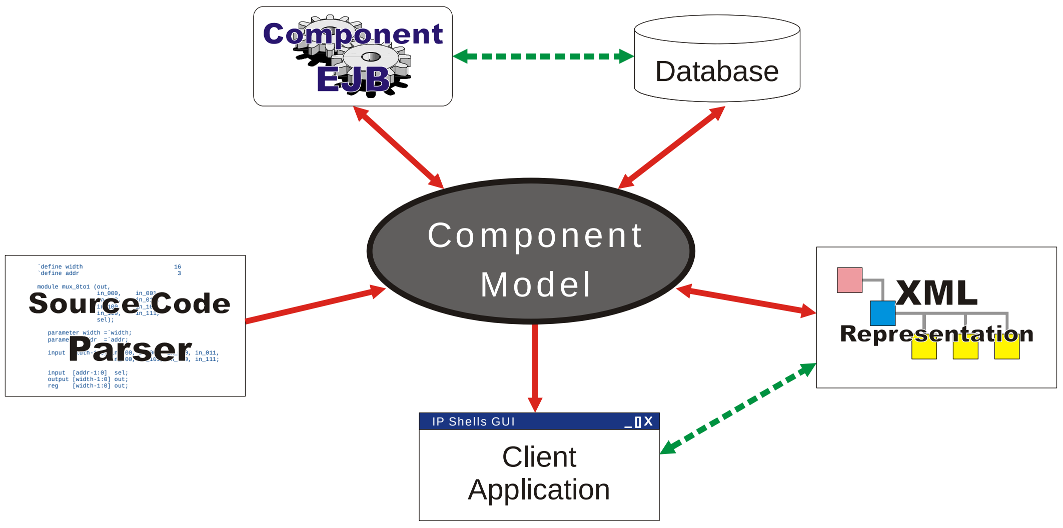


Figure 4-13
Component model and its integration with the environment

Figure 4-14 shows an XML persistence model for a 16-bit carry-look-ahead adder from the foundation library. The corresponding database schema is shown in Figure 4-15. The database schema shown is only for the component model and not its associated parameters and ports. The schemas for these models will be shown and described in sections 4.5.1 and 4.5.2 respectively.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<components>
  <component category="Adder" name="adder_cla_16bit">
    <library ID="sg_foundation_1" />
    <parameters number="1">
      <parameter name="width">
        <attributes type="INTEGER" signed="false" size="32" base="10" />
        <decimal_value default="16" current="16" />
        <argument index="1" />
      </parameter>
    </parameters>
    <ports number="4">
      <port name="output_sum" direction="output">
        <category type="sum" />
        <width expression="width" />
        <inverted negate="false" />
        <argument index="1" />
      </port>
      <port name="input_a" direction="input">
        <category type="input a" />
        <width expression="width" />
        <inverted negate="false" />
        <argument index="2" />
      </port>
      <port name="input_b" direction="input">
        <category type="input b" />
        <width expression="width" />
        <inverted negate="false" />
        <argument index="3" />
      </port>
      <port name="input_carry" direction="input">
        <category type="carry in" />
        <width expression="1" />
        <inverted negate="false" />
        <argument index="4" />
      </port>
    </ports>
  </component>
</components>
```

Figure 4-14
XML example of a component model

```
+-------------------+-----------------+------+-----+---------+----------------+
| Field             | Type            | Null | Key | Default | Extra          |
+-------------------+-----------------+------+-----+---------+----------------+
| Component_ID      | int(11)         |      | PRI | NULL    | auto_increment |
| Component_Name    | char(50) binary |      | MUL |         |                |
| Category_ID       | int(11)         |      | MUL | 0       |                |
| Available         | tinyint(1)      |      |     | 0       |                |
| Last_Modified     | timestamp(14)   | YES  |     | NULL    |                |
| Doc_ID            | int(11)         | YES  |     | NULL    |                |
| Doc_Last_Modified | timestamp(14)   | YES  |     | NULL    |                |
| Library_ID        | char(50) binary |      |     |         |                |
+-------------------+-----------------+------+-----+---------+----------------+
```

Figure 4-15
Component model database schema

A component model always refers to a single named module in a Verilog source file. Hence for this model to work there must be a means of identifying the correct RTL code from any given component model. Remembering from section 4.1 that all modules are sub-directories inside a library directory, it is possible to uniquely identify a macro-component from its module name and library ID. To retrieve the source code for a particular macro, the library manager can be queried using module name and its library ID. As shown in section 4.1.2 the library manager can resolve the full path to a library from a library identifier string.

4.5.1 Parameter modelling

As stated in section 1.3.2 many of the macro-components are parameterisable. To enable the parameterization in this platform, the characteristics of each parameter must be catalogued. Figure 4-16 describes the database schema employed for the persistence of module parameters. Section 3.11 of the IEEE Verilog Standard [69] defines the full specification of Verilog parameters but this application at this time only implements a sub-set of this specification. However sufficient functionality is implemented to cover all parameter usage in the macro-component libraries.

```
+-----------------+-----------------------+------+-----+---------+----------------+
| Field           | Type                  | Null | Key | Default | Extra          |
+-----------------+-----------------------+------+-----+---------+----------------+
| Parameter_ID    | int(11)               |      | PRI | NULL    | auto_increment |
| Parameter_Name  | char(20) binary       |      |     |         |                |
| Value_Type      | enum('INTEGER','REAL') |     |     | INTEGER |                |
| Signed          | tinyint(1)            |      |     | 0       |                |
| Value_Size      | int(11)               |      |     | 32      |                |
| Default_Value   | char(30)              |      |     |         |                |
| Base            | int(11)               |      |     | 10      |                |
| Parameter_Index | int(11)               |      |     | 0       |                |
| Doc_ID          | int(11)               | YES  |     | NULL    |                |
| Component_ID    | int(11)               |      | MUL | 0       |                |
+-----------------+-----------------------+------+-----+---------+----------------+
```

Figure 4-16
Parameter model database schema

Each parameter is uniquely identifiable within a module by its name and will either represent an integer or floating point number. Various number bases can be used, including binary, octal decimal and hexadecimal, but the actual default value will always be stored in its decimal form with an additional field recording the originally used base. Parameter values can be specified in one of two ways in Verilog. The first is to use the `defparam` statement. The second method is to pass in the appropriate values during the module instantiation. In this method the positional indexes of the values in the instantiation string defines which values are applied to which parameters, assuming that there is more than one. Hence the persistence layer also records the parameter index value.

4.5.2 Port modelling

Interfacing macro-components in a design is by way of their ports. Hence all information defining the characteristics of each port must be stored. Figure 4-17 shows the database schema for the port definitions.

```
+------------------+--------------------------------+------+-----+---------+----------------+
| Field            | Type                           | Null | Key | Default | Extra          |
+------------------+--------------------------------+------+-----+---------+----------------+
| Port_ID          | int(11)                        |      | PRI | NULL    | auto_increment |
| Port_Name        | char(30) binary                |      |     |         |                |
| Category_Port_ID | int(11)                        | YES  |     | NULL    |                |
| Argument_Index   | int(11)                        |      |     | 0       |                |
| Direction        | enum('input','output','inout') |      |     | input   |                |
| Width_Expression | char(200)                      |      |     |         |                |
| Inverted         | tinyint(1)                     |      |     | 0       |                |
| Doc_ID           | int(11)                        | YES  |     | NULL    |                |
| Component_ID     | int(11)                        |      | MUL | 0       |                |
+------------------+--------------------------------+------+-----+---------+----------------+
```

Figure 4-17
Port model database schema

All ports are uniquely identifiable within a module using their names and can be either inputs, outputs or inouts. An enumeration field is used to store the port direction. Like parameters, ports can be linked using their positional indexes in the module instantiation and so an argument index field is provided. However this is not recommended practice and ports should instead be identified using only their names. All ports on a module have a pre-determined width. This width can be specified either using integer values or me be compile time configured using parameters. To model this flexibility, the width of the port is defined using width-expression string, which can later be decoded into a numeric value.

4.5.3 Port width expression

If a module port is non-parameterisable its width will usually be specified using a fixed range expression such as `[15:0]` implying a width of 16 bits. Parameterisable ports will substitute one or more of the numeric values for parameter names, for example `[parameter_1:0]`. In many occasions the port widths will involve complicated expressions of both numeric values and parameter names. Figure 4-18 shows how the port widths of a fully parameterisable multiplier have been specified. In this example, the width of `product` is twice the value of parameter `word_size`, as defined using the expression `[2*word_size-1:0]`.

```
module multiplier(input_a, input_b, carry_in, product);

   parameter word_size = 16;

   input [word_size - 1:0] input_a;
   input [word_size - 1:0] input_b;
   input                   carry_in;

   output [2 * word_size - 1:0] product;
```

Figure 4-18
Parameter expressions used to define port widths

Rather than storing a static numeric value, each port model has a *width expression* object. The width expression object maintains a list of all parameters that have an influence upon it and a custom built calculator. This calculator, which is part of a mathematics package JAR, is able to dynamically calculate the width of a port from this expression.

Legal expressions can use any of the operators from the set {`^ / * + -`} and braces to specifically define precedence where needed. Observe this set and you will notice that the operators do not all have the same level of precedence. Division and multiplication have a higher precedence that addition and subtraction while the power operator (^) has the highest priority. But in the Verilog description, all mathematical expressions are defined using an infix notation. A direct interpretation of these expressions will not always lead to the correct result. For example directly equating `2*16-1` will give a value of 31 whereas `16-1*2` will yield the result 30 when it should actually be 14. To account for operator precedence, an implementation of a postfix calculator was developed.

A postfix calculator first takes the operators and operands of an infix expression and compiles them into in a new order. An example of the reordering is shown below.

Infix expression: `16 – 1 * 2`

Postfix equivalent stack: `16 1 2 * -`

This postfix equivalent expression is then evaluated using a second stack. Numeric value elements are popped off the postfix stack and pushed onto the evaluation stack. In the above example, the values 16, 1 and 2 will be popped from the postfix stack and pushed onto the evaluation stack. When the process comes across the operator (`*`) the top values are popped from the evaluation stack, in this case 2 and 1, and the operator applied to them – resulting in `2 * 1 = 2`. The value `2` is then pushed back onto the evaluation stack before the next iteration of reading the postfix stack begins. In the next iteration, the operator (`-`) is popped from the postfix stack. Again as this is an operator as opposed to a numeric value, the top two values are popped from the evaluation stack, these being `16` and `2`, and equated using the operator to give `16 - 2 = 14` which is then pushed back onto the evaluation stack. As the postfix stack is now empty, the final answer will be the single numeric value left on the evaluation stack, this being `14`.

4.5.4 Verilog parser

In order to initially construct the component model database, the necessary metrics must be compiled from the Verilog source files. This is performed in an automated stage using a custom made parsing application. The parser reads the Verilog files and extracts the necessary information.

The first stage of the parser is to read the file into memory and remove any comments from it. It is important to remove the comments as this will minimise later confusion. For example, some sections of code may have been rendered inoperable by being within comment blocks rather than having been physically removed. This is common practice during design debugging stages. Rather than developing a more intelligent parser, a simpler implementation can be realised if all comments are first removed. A potential drawback to this is that all pseudo-comments will also be removed. Pseudo-comments are special directives included within the source code intended for particular EDA tools [70]. Embedded within comment blocks, these directives are invisible to all tools except those specifically looking for them.

```
`define dwidth   32
`define width    16
`define depth1   16
`define depth2    8
`define cwidth    5
`define swidth    4

module fft_lp_64p (clk,in,Xfro,Xfio,reset);

   parameter dwidth  = `dwidth;
   parameter depth1  = `depth1;
   parameter depth2  = `depth2;
   parameter swidth  = `swidth;
   parameter cwidth  = `cwidth;
   parameter width   = `width;
```

→

```
module fft_lp_64p (clk,in,Xfro,Xfio,reset);

   parameter dwidth  =  32;
   parameter depth1  =  16;
   parameter depth2  =  16;
   parameter swidth  =   8;
   parameter cwidth  =   5;
   parameter width   =   4;
```

Figure 4-19
Example of definition expansion during pre-processing

The second stage performs pre-processing routines on the Verilog code. Pre-processing is the task of identifying all macros defined in the source header information and expanding this into the source body. An example of this expansion process is demonstrated in Figure 4-19. As well as macro expansion, pre-processors can generate blocks of code. Facilities for this are built into the VHDL language in the form of *generate* statements. This capability is now also finding its way into some of the high-end Verilog tools such as NC-Verilog from Cadence. Other generation capabilities include expansion of arrays of sub-module instances [71].

The third stage reads and caches all modules in a given Verilog source file. Although not a recommended design practice, it is still possible to have several modules all within a single source file. Referring to the directory structure defined in section 4.1.1 it can be seen that module name must be the same as the name of the home directory of the macro-component. Hence this module will be selected from the file by default, although other modules can be extracted if necessary.

The fourth stage accepts the source for an individual module and parses it to extract the information necessary to create a component model object. This is typically performed using source code parsers which build syntactic level models. However a lexical parsing approach is simpler to implement and often yields results comparable to those of a traditional parser [72]. This application performs lexical parsing using regular expressions to extract the necessary information. A list of some of the regular expression patterns used to extract the necessary information is given below.

Pre-process pattern
```
 `define[ \t]+([^, \t]+)[ \t]+([^\r\n]+)
```

Parameters pattern
```
 parameter[ \t]+([^, \t]+)[ \t]*=[ \t]*([^;]+)
```

Module pattern
```
 module[ \t]+([^ \t\\(]+)[ \t]*(\\([^\\)]+\\))*[ \t]*;
```

I/O port pattern
```
  (input|output)([ \t\n]*\\[[^\\]]+\\][ \t\n]*|[ \t\n]+)([^\"\\(\\);]+);
```

Width pattern
```
  \[([^:]+):([^\]]+)\]
```

Integer pattern
```
  ([-]?[0-9]+)
```

Floating point number pattern
```
  ([-]?[0-9]+[.][0-9]+)
```

Equation pattern
```
  [ \t]*([\+-])[ \t([-]?[0-9]+)
```

4.5.5 Assigning category to component model

Having created a component model through parsing the Verilog source code as defined in section 4.5.4 only the structural characteristics of the module is known. A subsequent stage to bind that component model to a category group is needed. Category binding involves trying to match a component model to a category model and then assigning that category model to the component model.

The category model described in section 4.4.1 defines the characteristics of a category, but makes no attempt to correlate between components and categories. Mapping components to categories is performed using regular expression pattern matching. A set of regular expression patterns is maintained for each category in a separate database table. The schema for this category pattern model is shown in Figure 4-20.

Note that a foreign key index to a library ID is provided. This means that a separate set of category patterns can be stored for each library.

```
+-------------------------+-----------------+------+-----+---------+----------------+
| Field                   | Type            | Null | Key | Default | Extra          |
+-------------------------+-----------------+------+-----+---------+----------------+
| Category_Pattern_ID     | int(11)         |      | PRI | NULL    | auto_increment |
| Category_Pattern_Regexp | char(250)       |      |     |         |                |
| Library_ID              | char(50) binary |      | MUL |         |                |
| Category_ID             | int(11)         |      | MUL | 0       |                |
+-------------------------+-----------------+------+-----+---------+----------------+
```

Figure 4-20
Category pattern model database schema

The mapping stage involves attempting to match the macro-component name against each of the category name patterns (each belonging to the same library). An example of a regular expression pattern for FIR filters is shown below.

```
fir_(block|comb|conv|seg)_[0-9]+tap_[0-9]+b_(gen|lp)_(booth|mult)
```

This expression expects the filter name to start with "`fir_`" followed by one of four words describing the architecture. Next the pattern covers the number of taps and word-size bit-width. The final sections of the pattern identify whether it is generic or low power and the multiplier used. Assuming that the component name matches this pattern, it *may* be an FIR filter. Final confirmation is by checking that the port names also match a set of port name patterns. Figure 4-21 shows the database schema for the table used to store the category port name patterns.

```
+--------------------------+-----------------+------+-----+---------+----------------+
| Field                    | Type            | Null | Key | Default | Extra          |
+--------------------------+-----------------+------+-----+---------+----------------+
| Category_Port_Pattern_ID | int(11)         |      | PRI | NULL    | auto_increment |
| Port_Pattern_Regexp      | char(250)       |      |     |         |                |
| Category_Port_ID         | int(11)         |      | MUL | 0       |                |
| Library_ID               | char(50) binary |      | MUL |         |                |
+--------------------------+-----------------+------+-----+---------+----------------+
```

Figure 4-21
Category port pattern model database schema

If the macro-name matches a category name pattern but the number of ports is inconsistent or do not match the port name patterns, the process will continue to iterate through the remaining categories until it finds a compete match.

Assuming that a matching category is found, the name of that category is assigned to the component model. In a similar fashion, the port models are bound to the correct port categories. This will later allow schematic capture symbols to be mapped over the top of and bound to component models.

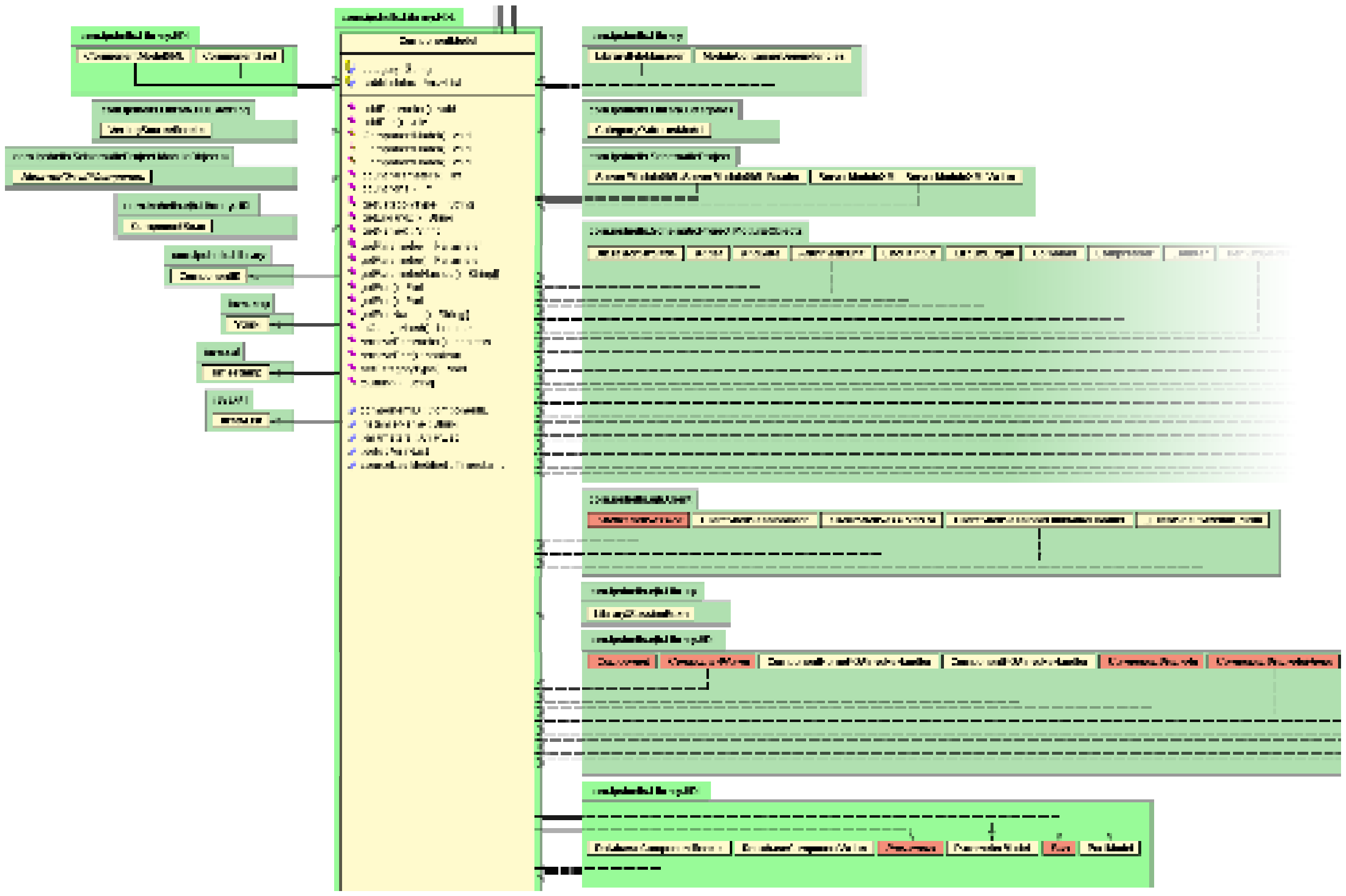
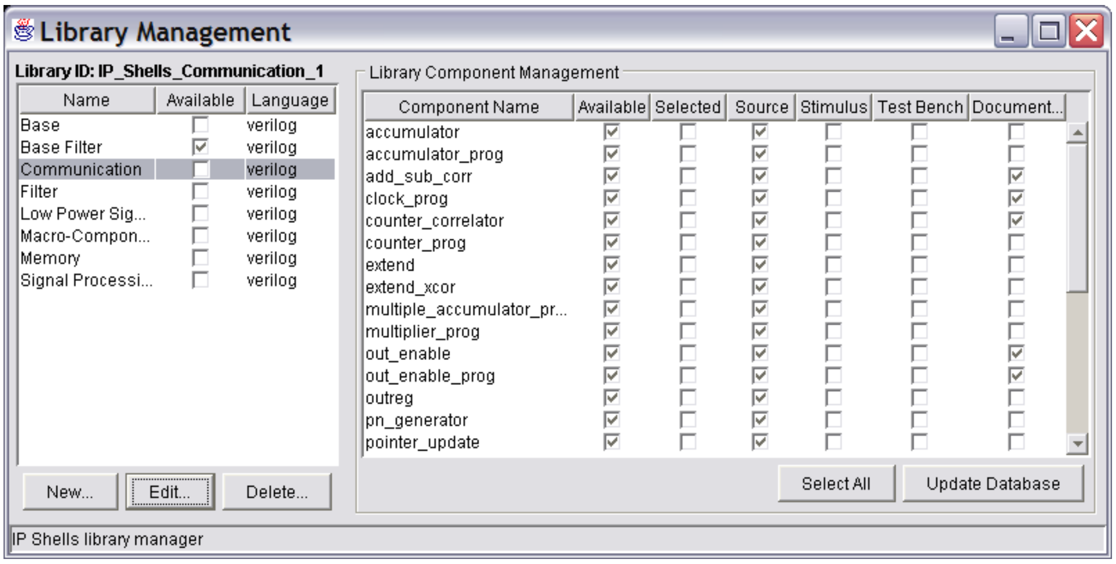


Figure 4-22
UML representation of a component model

4.5.6 Cataloguing application

Cataloguing the component models into the appropriate persistence layer is performed using a GUI application specifically developed for the task. A screen-shot of this GUI application, built around the MVC design pattern, can be seen in Figure 4-23.



Library ID: IP_Shells_Communication_1

Name	Available	Language
Base	☐	verilog
Base Filter	☑	verilog
Communication	☐	verilog
Filter	☐	verilog
Low Power Sig...	☐	verilog
Macro-Compon...	☐	verilog
Memory	☐	verilog
Signal Processi...	☐	verilog

Library Component Management

Component Name	Available	Selected	Source	Stimulus	Test Bench	Document...
accumulator	☑	☐	☑	☐	☐	☐
accumulator_prog	☑	☐	☑	☐	☐	☐
add_sub_corr	☑	☐	☑	☐	☐	☑
clock_prog	☑	☐	☑	☐	☐	☑
counter_correlator	☑	☐	☑	☐	☐	☑
counter_prog	☑	☐	☑	☐	☐	☐
extend	☑	☐	☑	☐	☐	☐
extend_xcor	☑	☐	☑	☐	☐	☐
multiple_accumulator_pr...	☑	☐	☑	☐	☐	☐
multiplier_prog	☑	☐	☑	☐	☐	☐
out_enable	☑	☐	☑	☐	☐	☑
out_enable_prog	☑	☐	☑	☐	☐	☑
outreg	☑	☐	☑	☐	☐	☐
pn_generator	☑	☐	☑	☐	☐	☐
pointer_update	☑	☐	☑	☐	☐	☐

IP Shells library manager

Figure 4-23
Library catalogue manager GUI

This tool has two panels. The panel on the left lists the libraries that have been registered with the library manager. These can be edited using the panel presented in Figure 4-4. The right-hand panel lists all macro-components belonging to the selected library.

To update the database, the program will iterate through each of the selected macros and insert them into the database if they are not already there, or update existing entries if the source file's last modified timestamp is not equal to that recorded in the database for that component. Inserting new components into the database is relatively straightforward, but updating is somewhat more complicated. For example, a component which has already been stored in the database has now been extensively modified. These modifications mean that the component now has less parameters and less ports. All database entries for ports and parameters that have been removed from the design also have to be located and removed from the database.

4.6 Component documentation

Even the most advanced IP cores are of limited use if they are incorrectly interfaced or not exploited to their full potential. Hence for end users to extract maximum potential from their investment they will be expecting full supporting documentation accompanying each and ever macro component. Typically, as in the case of the DesignWare library from Synopsys, each high-level component is comprehensively described in the form of an extensive PDF document. Each of the cores in the macro-component library has supporting documentation, also in PDF files. Although not implemented in this project, due to time restrictions, it should be possible to deliver these PDF documents to the client application via the application server.

4.6.1 Entity based documentation model

The concept of an entity based documentation model is an attempt to assist design engineers to rapidly use the macro-components. In this model each *entity* of a component is independently documented. The top level entity of a component is the component itself, for which there will be a supporting document. Ports and parameters are then considered as entities of that component and each independently documented. When an engineer needs to know the influences that a particular parameter has upon a component for example, they can rapidly access it rather than searching through an overly comprehensive PDF file.

To facilitate rapid access to an entity's documentation, it is all cached in the server-side database. From Figure 4-15, Figure 4-16 and Figure 4-17 it can be seen that the tables for components, parameters and ports respectively all have provision for supporting documentation. Entity documentation is read into the database during the cataloguing operation described in section 4.5.6. The original documentation can be written in either individual text files (one per entity) or compiled in a single XML file.

4.7 Macro-component database

The previous sections described the individual tables in the database necessary to model category and component attributes and characteristics. Figure 4-24 depicts how all of these tables link together. The SQL script to create this table arrangement can be found in appendix B.

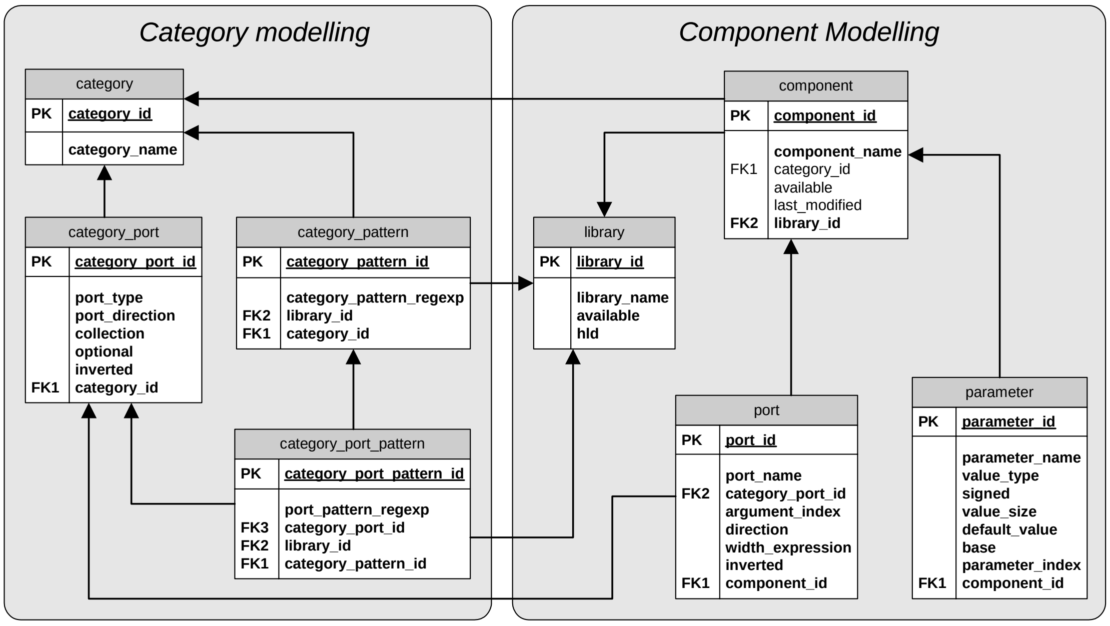


Figure 4-24
Component and category modelling database

4.8 IP library EJB's

Within the enterprise environment, each entity is modelled by an appropriate EJB. Section 4.7 shows the structure of the information cached in a database for the IP libraries. Each database table is an entity and thus bean managed persistence EJB's wrapping each table were used.

4.9 Summary

This chapter described the structure of the server-side backend IP libraries being delivered as a service through this EDA framework. The physical structure of the underlying file system and the module linking mechanisms were described and demonstrated using a particular FFT as an example. This example highlighted the potential to rapidly modify the architecture of advanced digital systems and re-verify in a matter of minutes. This chapter also presented a means of categorising and cataloguing IP metrics into a variety of persistence layers. Effective use of the façade design pattern was demonstrated allowing part of the design environment to be used in multiple configurations, client-server based and as a stand-alone tool.

Chapter 5
Client-side schematic capture tool

The previous chapter described the IP libraries to be deployed as a service within an EDA platform. This chapter describes a one potential EDA environment that allows end users to access these server-side IP libraries through a graphical user interface (GUI). Considering the client-server architecture of the platform, there were two possible means of developing the interface.

1. **Web browser interface**

   The majority of distributed applications are developed around traditional web browsers such as Microsoft's Internet Explorer and Mozilla. The solution could be based on standard HTML and dynamically generated graphics, web scripting languages including JavaScript or perhaps as a MacroMedia Flash application. Cadence, and what was their SpinCircuit EDA portal, was built around web browsers. One of the drawbacks of browser-based solutions is their slow responsiveness, especially if the network connection is slow. Additionally, in applications such as this page timeout becomes an issue. Large digital systems take considerable amounts of time to verify and synthesis, making controlling the process via a browser quite difficult.

2. **Executable application**

   The alternative is to build an executable application that makes us of the internet as a communication protocol to the server. Such an application would need to be installed on the end-user's computer.

The client side environment described in this thesis is based upon an executable application developed in Java. It should also be possible to use any other programming language that supports the CORBA communication protocol, such as C++ combined with a suitable widget set like GTK to act as the GUI. Having used Java, portions of the server code can be directly integrated into the client by way of the façade design pattern, meaning that it can also be operated in a stand-alone mode, completely independent of the server.

5.1 Schematic capture environment

The interfacing environment for the client-side EDA platform is based on traditional schematic capture. There are many schematic capture packages for electronic design, both commercially and freely available. Typically these wrap around proprietary libraries of IP building blocks and have static links to a predefined set of verification and synthesis tools. The key differentiator of this solution is neither the IP libraries nor back-end EDA tools are locally available. Instead the schematic capture environment wraps around network services available on the enterprise application server. A screen-shot of this rudimentary GUI can be seen in Figure 5-1. As can be seen, it resembles other well known schematic capture tools, but uniquely masquerades the underlying server in a transparent manner.

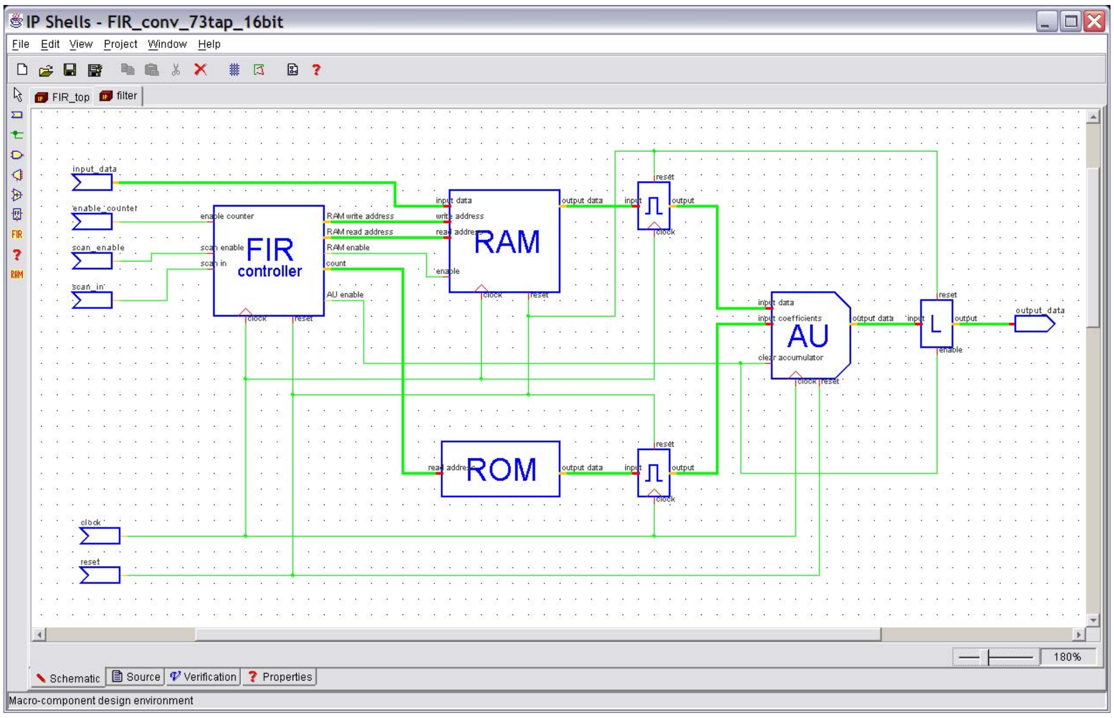


Figure 5-1
Screen shot of conventional FIR in the schematic capture environment

5.1.1 Accessing macro-components

Before a design can be created, the list of all available macro-components must be made available to the client application. Not all macros on the server will be available to every client. Using server-side constructs to be described in section 6.1, restrictions can be imposed upon available modules based upon client user details. When requested the server generates the list of available cores from its database and transmits it to the client. The returned list contains the component ID and corresponding library ID of every available macro-component. Additionally the list is hierarchically structured and ordered into groups by category before being serialized and returned by the server. This model ensures that the volume of data involved in the transaction is minimised and ensures a fast start-up for the client which initiates this request when it launches.

Within the client the list of macros is available as a series of menus on the left-hand side of the application as seen in Figure 5-1. These menus are dynamically constructed to match the hierarchal structure defined in the returned list. A close-up of one of these menus expanded is shown in Figure 5-2.

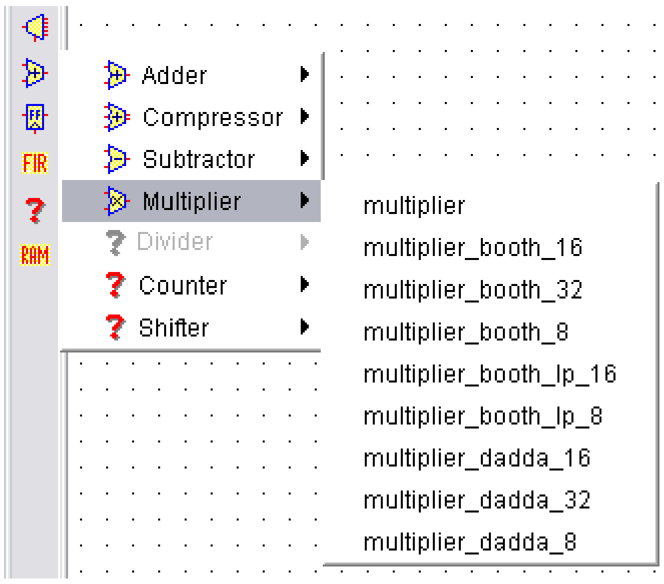


Figure 5-2
Menu of multipliers available to the client

The management system constructing the menus is built around the façade design pattern, meaning that can be initialised from the J2EE server, SQL database or from XML descriptions facilitating stand-alone operation of the application. Response times of all three approaches are comparable.

5.1.2 Downloading ‘virtual’ macro-components

Selecting one of the components from the menu prepares the application to draw it on the screen. The first stage of the process is to source the appropriate component model. To ensure optimal performance, the environment operates a caching mechanism for virtual components ensure that they are only downloaded once. If not in the cache, the module and library ID’s of the requested component are sent to the server which responds be returning the appropriate component model.

5.1.2.1 Response performance

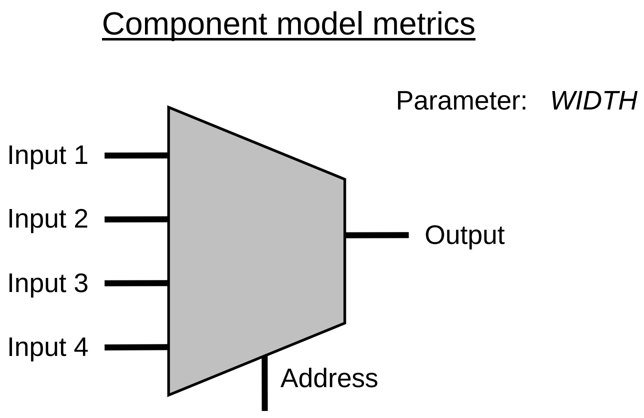


Figure 5-3
Database queries for 4-input multiplexer

The response time of the component model acquisition is proportional to the number of entities associated with the component model. In this case the entities are ports and parameters. When constructing the component model the server component model EJB needs to query the database about each individual port and parameter, a potentially time consuming process. With the 4-input multiplexer shown in Figure 5-3 the server must issue a total of 8 queries, one for the component model metrics, one for the parameter and six for the ports. The total client-server transaction time to download the component model for the 4-input multiplexer took 0.3 seconds.

However the same test for a 512-input multiplexer took 1.2 seconds due to the 516 database queries, a noticeable response time while using the tool. Independent research suggests that implementing an EJB caching architecture to reduce stress on the database server could greatly improve response times [73] though.

5.1.3 Creating a design

Once a component model has been downloaded to the client, it can be represented on the design area through a dynamically generated symbol. Component symbols can then be interconnected by net symbols as can be seen in Figure 5-1 which shows a conventional FIR filter.

The server-side libraries contain a number of components that allow different configurations of the same design to be created. For example the FIR filter of Figure 5-1 can be easily adapted to use a more advanced architecture by simply replacing some of the components in the schematic. This can be upgraded in a matter of minutes to a block processing algorithm [74] by simply substituting the controller and arithmetic unit (AU) components and adding an additional control net between them, as shown in Figure 5-4. The end result will then be a functionally equivalent filter but algorithmically optimised to consume less power. Being functionally equivalent, the same testbench can be applied to both implementations.

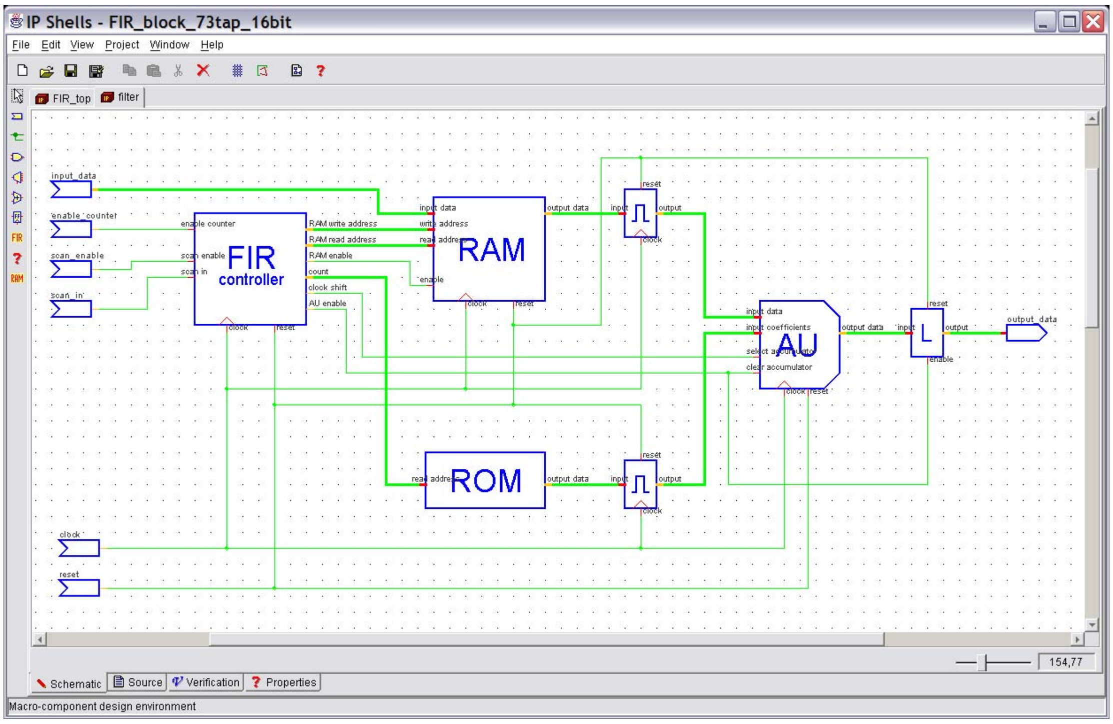
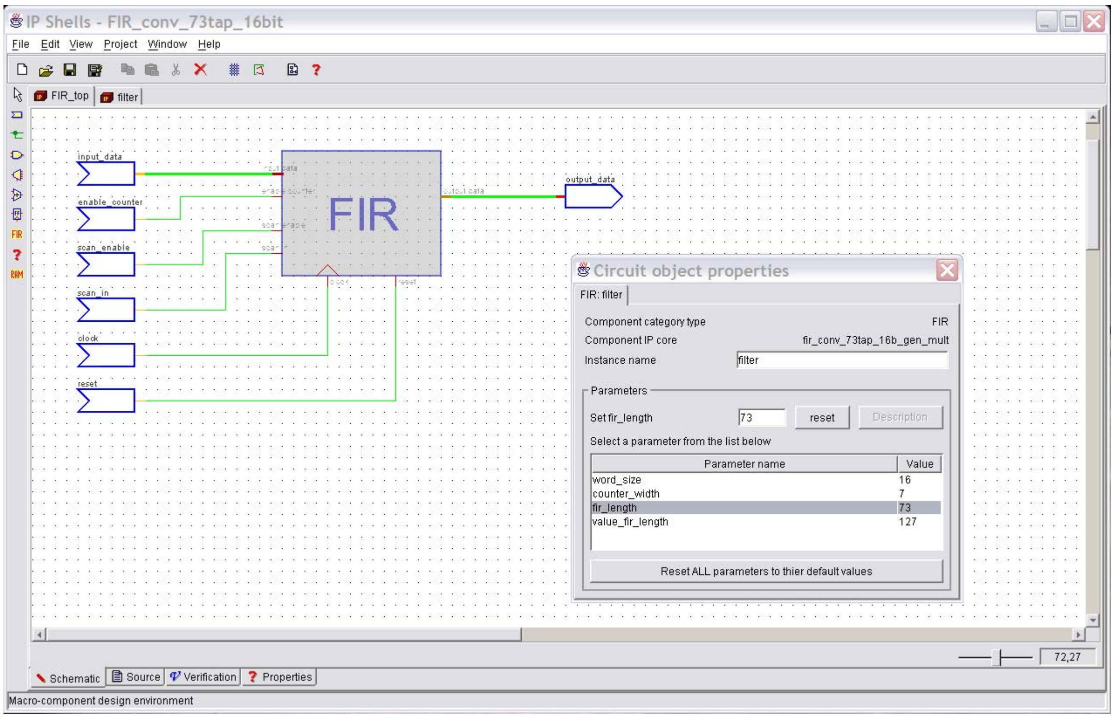


Figure 5-4
FIR block processing algorithm

5.1.4 Parameterising components

One of the fundamental concepts of macro-components is that they are parameterisable, as stated in section 1.3.2. As parameter information is encoded into the component model, the characteristics of the virtual images of the component models can be readily adapted. Figure 5-5 shows the parameters associated with the top level of an FIR filter and how they can be modified.



Circuit object properties

FIR: filter

Component category type	FIR
Component IP core	fir_conv_73tap_16b_gen_mult
Instance name	filter

Parameters

Set fir_length: 73 — reset — Description

Select a parameter from the list below

Parameter name	Value
word_size	16
counter_width	7
fir_length	73
value_fir_length	127

Reset ALL parameters to thier default values

Figure 5-5
Configuring macro-component parameters

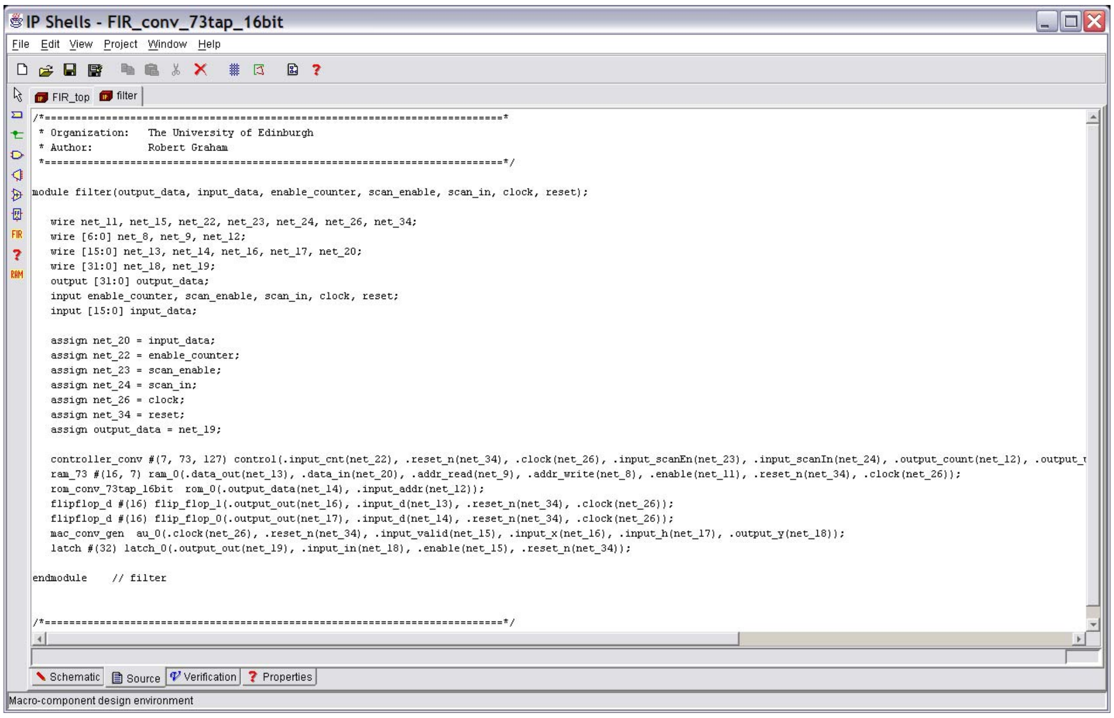
Section 4.6.1 of this thesis introduced the concept of entity based documentation. With reference to Figure 5-5, any supporting documentation for a parameter can be visualised via the description button, making the design process more intuitive and accelerating development.

5.1.5 Hierarchial design

Good design methodology dictates that digital systems are developed making extensive use of hierarchical modelling concepts [6, 67]. The two basic types of design methodology are top-down and bottom-up. In each of these models the design is recursively partitioned and fragmented into sub-blocks, each of specific functionality. This environment allows designs to be partitioned into multiple modules, each described by its own schematic. A design project can consist of several user-design modules plugged together.

5.1.6 Simulation and verification

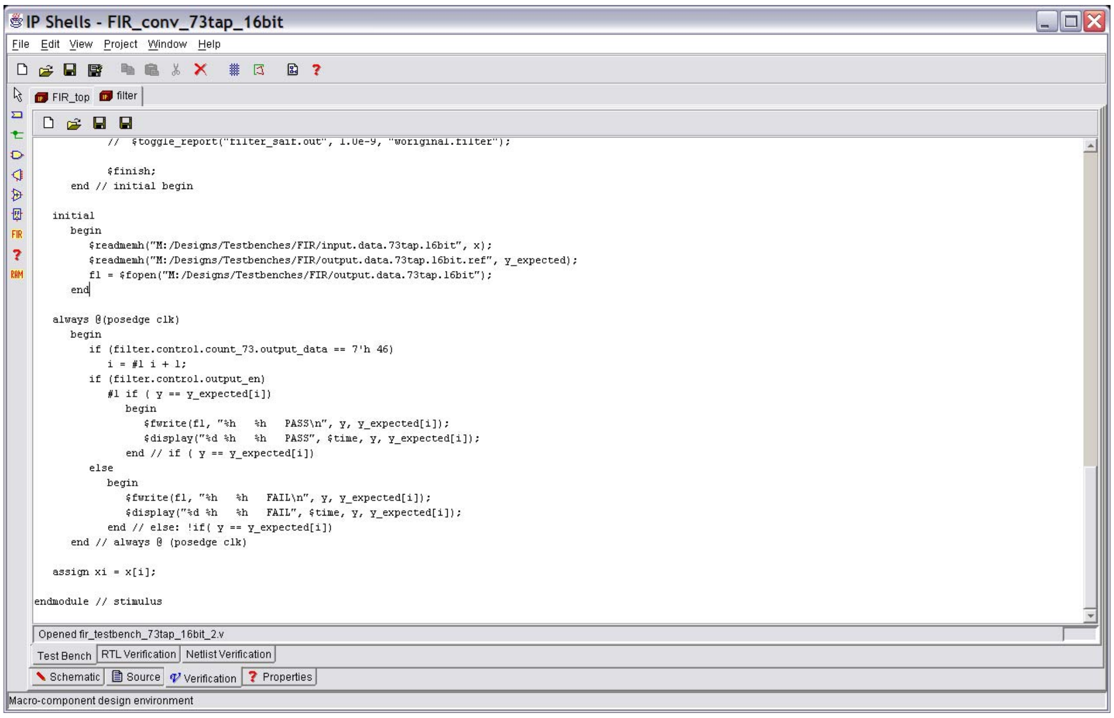
Once a design has been created, it should be functionally verified. Verification is the process of ensuring that a design functions in a manner correct to its intended application. The verification procedure begins by linking together the RTL source code for all modules in the design. In the client-server environment the verification stage must take place on the server. When a design is to undergo simulation, the project information is serialised and streamed to the server where it is then extracted into the user's file space. After synchronization, which will be described in the next chapter, the top-level Verilog RTL is generated using a stateless EJB. This is performed in the server as opposed to the client for reasons of security and design data integrity. This can be streamed back to the client if necessary as is shown in Figure 5-6



```
/*===========================================================================*
 * Organization:   The University of Edinburgh
 * Author:         Robert Graham
 *===========================================================================*/

module filter(output_data, input_data, enable_counter, scan_enable, scan_in, clock, reset);

   wire net_11, net_15, net_22, net_23, net_24, net_26, net_34;
   wire [6:0] net_8, net_9, net_12;
   wire [15:0] net_13, net_14, net_16, net_17, net_20;
   wire [31:0] net_18, net_19;
   output [31:0] output_data;
   input enable_counter, scan_enable, scan_in, clock, reset;
   input [15:0] input_data;

   assign net_20 = input_data;
   assign net_22 = enable_counter;
   assign net_23 = scan_enable;
   assign net_24 = scan_in;
   assign net_26 = clock;
   assign net_34 = reset;
   assign output_data = net_19;

   controller_conv #(7, 73, 127) control(.input_cnt(net_22), .reset_n(net_34), .clock(net_26), .input_scanEn(net_23), .input_scanIn(net_24), .output_count(net_12), .output_
   ram_73 #(16, 7) ram_0(.data_out(net_13), .data_in(net_20), .addr_read(net_9), .addr_write(net_8), .enable(net_11), .reset_n(net_34), .clock(net_26));
   rom_conv_73tap_16bit  rom_0(.output_data(net_14), .input_addr(net_12));
   flipflop_d #(16) flip_flop_1(.output_out(net_16), .input_d(net_13), .reset_n(net_34), .clock(net_26));
   flipflop_d #(16) flip_flop_0(.output_out(net_17), .input_d(net_14), .reset_n(net_34), .clock(net_26));
   mac_conv_gen  au_0(.clock(net_26), .reset_n(net_34), .input_valid(net_15), .input_x(net_16), .input_h(net_17), .output_y(net_18));
   latch #(32) latch_0(.output_out(net_19), .input_in(net_18), .enable(net_15), .reset_n(net_34));

endmodule    // filter


/*===========================================================================*/
```

Figure 5-6
Generated RTL Verilog from a schematic

Testbenches are the traditional means of performing functional verification. Written in behavioural level Verilog, a testbench forms the (non-synthesisable) top level module in the design. It then stimulates the design by applying input stimuli while comparing the output against reference sets of data. Figure 5-7 shows how a testbench can be added to the project. A comprehensive guide to verification strategies and testbenches can be found in the book Writing Testbenches [68].

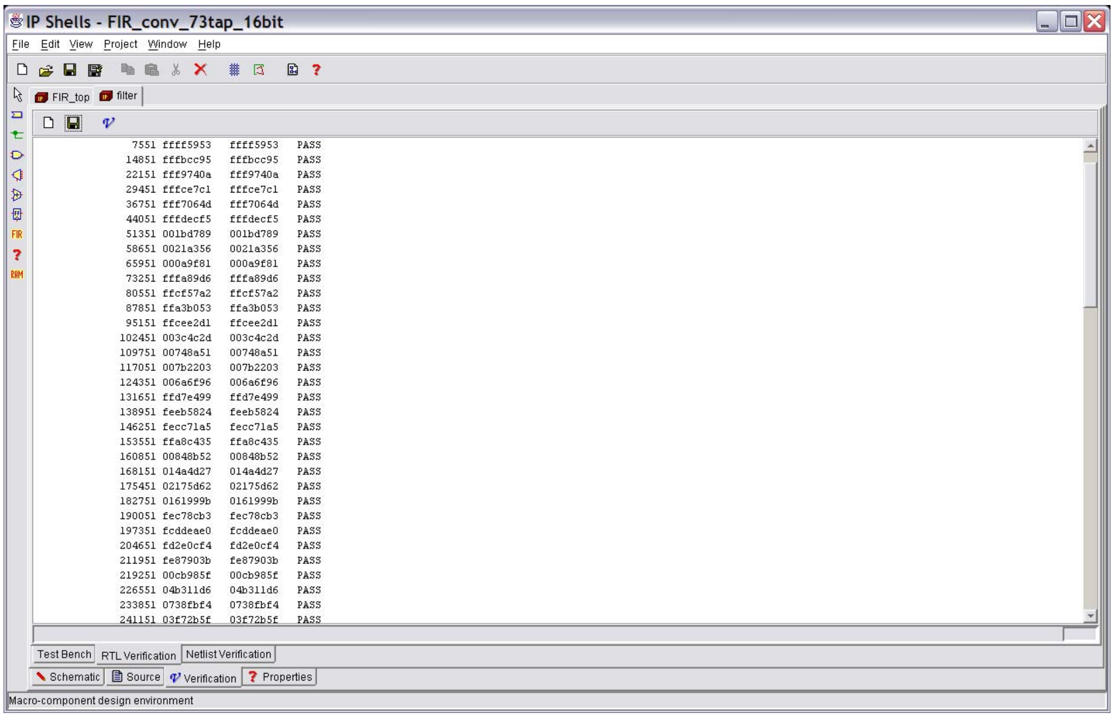

IP Shells - FIR_conv_73tap_16bit
File Edit View Project Window Help
FIR_top
filter
Test Bench
RTL Verification
Netlist Verification
Schematic
Source
Verification
Properties
Macro-component design environment

```
         //  $toggle_report("filter_saif.out", 1.0e-9, "woriginal.filter");

         $finish;
      end // initial begin

   initial
      begin
         $readmemh("M:/Designs/Testbenches/FIR/input.data.73tap.16bit", x);
         $readmemh("M:/Designs/Testbenches/FIR/output.data.73tap.16bit.ref", y_expected);
         fl = $fopen("M:/Designs/Testbenches/FIR/output.data.73tap.16bit");
      end

   always @(posedge clk)
      begin
         if (filter.control.count_73.output_data == 7'h 46)
            i = #1 i + 1;
         if (filter.control.output_en)
            #1 if ( y == y_expected[i])
               begin
                  $fwrite(fl, "%h   %h   PASS\n", y, y_expected[i]);
                  $display("%d %h   %h   PASS", $time, y, y_expected[i]);
               end // if ( y == y_expected[i])
         else
            begin
               $fwrite(fl, "%h   %h   FAIL\n", y, y_expected[i]);
               $display("%d %h   %h   FAIL", $time, y, y_expected[i]);
            end // else: !if( y == y_expected[i])
      end // always @ (posedge clk)

   assign xi = x[i];

endmodule // stimulus
```

Opened fir_testbench_73tap_16bit_2.v

Figure 5-7
Testbench for functional verification of a module

After synchronizing the client project with the server and building the necessary RTL files the verification procedure can commence. The server will link together all necessary RTL modules using the linker mechanism described in section 4.2 and then invoke the appropriate verification tools in a process to be described in section 6.4.3. Once the module has been verified on the server, the results will be streamed back to the client and as shown in Figure 5-8.


IP Shells - FIR_conv_73tap_16bit
File Edit View Project Window Help
FIR_top
filter
Test Bench
RTL Verification
Netlist Verification
Schematic
Source
Verification
Properties
Macro-component design environment

```
  7551 ffff5953   ffff5953   PASS
 14851 fffbcc95   fffbcc95   PASS
 22151 fff9740a   fff9740a   PASS
 29451 fffce7c1   fffce7c1   PASS
 36751 fff7064d   fff7064d   PASS
 44051 fffdecf5   fffdecf5   PASS
 51351 001bd789   001bd789   PASS
 58651 0021a356   0021a356   PASS
 65951 000a9f81   000a9f81   PASS
 73251 fffa89d6   fffa89d6   PASS
 80551 ffcf57a2   ffcf57a2   PASS
 87851 ffa3b053   ffa3b053   PASS
 95151 ffcee2d1   ffcee2d1   PASS
102451 003c4c2d   003c4c2d   PASS
109751 00748a51   00748a51   PASS
117051 007b2203   007b2203   PASS
124351 006a6f96   006a6f96   PASS
131651 ffd7e499   ffd7e499   PASS
138951 feeb5824   feeb5824   PASS
146251 fecc71a5   fecc71a5   PASS
153551 ffa8c435   ffa8c435   PASS
160851 00848b52   00848b52   PASS
168151 014a4d27   014a4d27   PASS
175451 02175d62   02175d62   PASS
182751 0161999b   0161999b   PASS
190051 fec78cb3   fec78cb3   PASS
197351 fcddeae0   fcddeae0   PASS
204651 fd2e0cf4   fd2e0cf4   PASS
211951 fe87903b   fe87903b   PASS
219251 00cb985f   00cb985f   PASS
226551 04b311d6   04b311d6   PASS
233851 0738fbf4   0738fbf4   PASS
241151 03f72b5f   03f72b5f   PASS
```

Figure 5-8
Results returned from server-side RTL simulation

5.1.7 Simulation time

During simulation the enterprise application server spawns the verification tool as an independent process on the host server. Hence it is fair to say that the time taken to verify a design is directly comparable to that of traditional local host simulation times.

5.2 Drawing schematic symbols

Each macro-component is represented on the client application as a schematic symbol. A symbol must be available for each type of component category that is to be displayed. Symbols are defined using dedicated Java objects that implement the `java.awt.Shape` interface. The shape interface defines a framework allowing geometric shapes to be displayed and manipulated within a graphical environment.

5.2.1 Symbol geometry

Symbols have been designed with inherent dynamic geometric scaling, making it easily possible to zoom in and out on the schematic. This has been achieved by defining the geometry of each symbol as a multiple of X and Y grid co-ordinates. Both X and Y scaling values can be independently specified, but this is generally unnecessary and for convenience they have been locked together using the sliding zoom facility at the bottom right-hand corner of the schematic screen. Figure 5-9 shows how the shape of the adder category has been defined. The variable ‘`g`’ is used to represent the X-Y grid scaling.

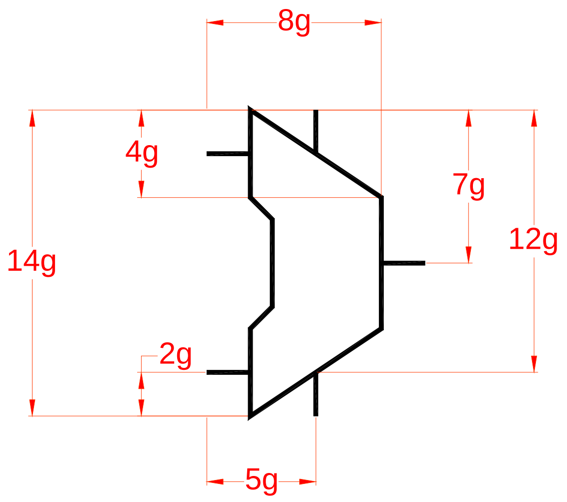


Figure 5-9
Anatomy of an adder symbol

Certain category symbols have been defined in such a way to allow them to be scalable in terms of number of inputs and/or outputs. Example of such component categories are multiplexers, de-multiplexers and gates. Rather than create new symbols for each component size, an additional variable is added to the geometry definition. Figure 5-10 and Figure 5-11 demonstrate how this variable ‘i’ can be used to draw an AND gate and OR gate with an arbitrary number of inputs.

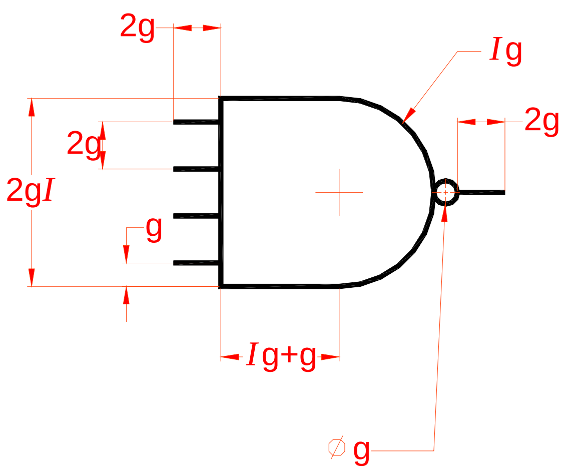


Figure 5-10
Constructing an AND Gate symbol

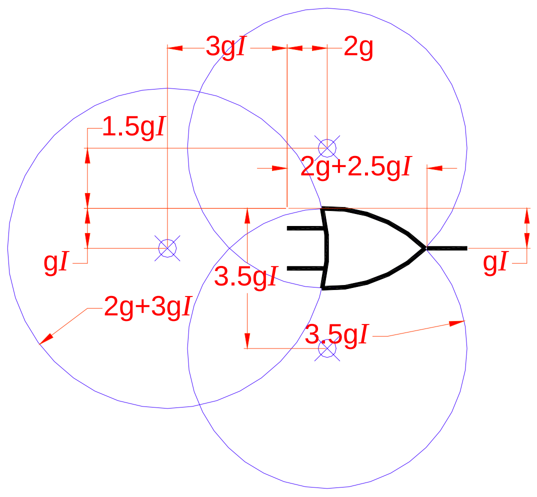


Figure 5-11
Constructing an OR Gate symbol

5.2.2 Symbol to macro-component binding

To minimize the need to create huge libraries of symbols, each symbol has been designed to cover a particular category range. Hence a means of mapping the component models within a particular category to the corresponding symbol is needed. Each symbol is designed taking into account the information compiled in the category model database. This means that the port names of both the category and symbol must match. The symbol also accounts for optional ports and does not display them if they are not present in the component model. Figure 5-12 shows the binding process for a carry save adder to an adder symbol.

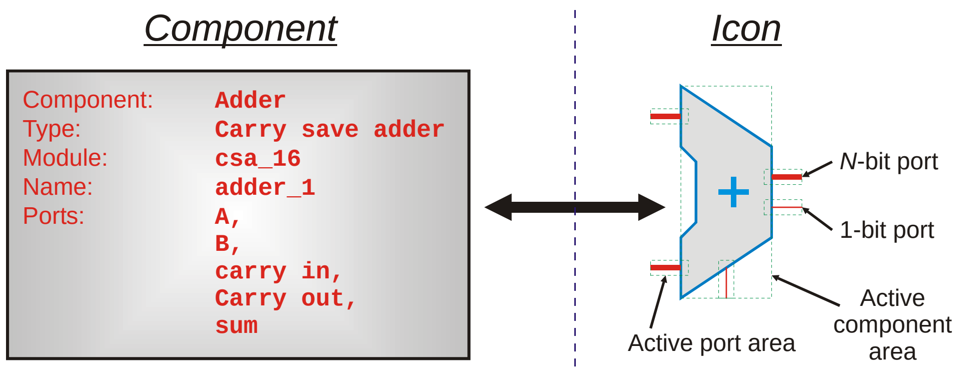


Figure 5-12
Binding a symbol to a macro-component

5.2.3 Symbol bounding regions

To allow the user to interact with the schematic, a set of bounding regions for each symbol are defined. The bounding regions for an adder symbol are shown in the dotted green outline of Figure 5-12. Firstly the symbol body is placed within a rectangular bounding box. Mouse clicks detected within this bounding box are used to highlight the component. Smaller bounding boxes surround all of the ports. These bounding boxes are used when the user attempts to connect a net to the port. On the screen, 1-bit ports are one pixel wide, but it is impractical to expect the user to be pixel perfect in clicking on the port. Hence the bounding box defines a region around the port, ±½ a grid spacing, allowing for a degree of mouse control error.

5.2.4 Nets

Component symbols are interconnected using nets. Like components, nets are schematically represented using symbols, in this case a `NetSymbol` object. Net symbols are a collection of line objects defining the path of the net. Nets support branching abilities allowing fanout to be implemented.

5.2.5 Drawing controllers

The most complicated aspect of developing any schematic capture environment is that of interpreting mouse events. The GUI environment supports a large number of features, but these must all be accessible using nothing more than the two buttons found on a typical mouse. Through these two buttons, the user must be able to create new components, relocate these components, edit their characteristics, draw nets and obtain design information to name but a few. To simplify supporting all of these features a range of application specific mouse controllers have been developed. The first is a default controller that allows symbols to be moved and manipulated and their properties edited. For the instantiation and placement of new components, a component symbol controller extends the functionality of the default controller. A final controller has been implemented for the manipulation of nets. Polymorphic switching between these controllers occurs depending upon the user's current activities.

5.3 License management

Considering the client-server configuration of this project, a means of protecting the valuable intellectual property on the server is necessary. As highlighted in section 1.2 of the introduction IP security is one of the key objectives of the research. Rather than trusting the customers to obey the conditions of a written license agreement [75], a simple but effective license management scheme was developed. This was considered important for the reasons highlighted in the following paragraphs.

The first and most obvious reason is to ensure the product can only be operated by those who have been granted the specific right. Software piracy is now a massive industry which is very damaging to legitimate software authors and vendors [76]. Steps must be taken to ensure that software applications can only be distributed and used within tightly controlled circumstances. Without any form of protection, a software application can be installed and run from any compatible computer. Typical license management practices will attempt to restrict the number of instances and locations from which the application will run. A license server will usually be installed with the application that will serve licenses to clients on the local network. One of the most widely used license servers is FlexLM [77].

The second and potentially less obvious reason for license management is controlling the lifetimes of products. This is especially true for alpha and beta releases, as issuing short term licenses will ensure that products expire after a fixed time period. Operation and support costs will be reduced if vendors know that there are no customers using out of date software. When a customer's license expires, a renewed license can be granted providing they download the relevant patches, updates or latest versions.

The client application developed here can be made compatible with commercial license management solutions such as FlexLM, but this option was not available during the project development. Hence a simple but relatively comprehensive licensing procedure based on simple text files was investigated. An example of a license file can be seen in Figure 5-13. When the client application is invoked, the license file is read and the constraints within it interpreted. Failure to comply with any of its directives will prevent the application from loading. To prevent this license file from being copied and given to multiple users, the application server has been configured to only grant a session to a single instance of a named user.

```
###############################################################
# Product license file

User_Name           = rig
Password            = ********
Host                = 129.215.[0-9]+.[0-9]+
Products            = tool_1@1.6:tool_2@[1-2]\.[0-9]+:
Start_Date          = 07-04-2005 AD at 07:01:09 BST
Expiry_Date         = 31-12-2005 AD at 23:59:59 GMT

Signature           = 2A2566180AF1CE1EB40FBF93AC8AA330

#-------------------------------------------------------------#
```

Figure 5-13
Example license file

As can be seen in Figure 5-13 a license file has a format similar to a shell properties file containing several `property = value` entries. The range of properties are defined as follows:

- *User Name*

  The client’s username as defined in the server-side database. Each subscriber will have a unique username allowing them server login access.

- *Password*

  Combined with the username, each client must also have a password allowing them to log into the server. Defining the password in the license file is optional as it could lead to potential security breaches. This is optional and if excluded it will be prompted for as the application loads.

- *Host*

  The host property defines the IP network addresses of the workstations that can be used to access the server. The adopted IP addressing scheme is the V4 IP protocol which has the form [byte.byte.byte.byte] with the classic example being `127.0.0.1` – the IP address of localhost. The host value is actually a regular expression of the host address matching pattern. It is therefore possible to match any host address using the expression `[0-9]+.[0-9]+.[0-9]+.[0-9]+.`

- *Products*

  The products property defines the range of products covered by the license. Several products can then be covered by a single license. All products must be listed here separated by the colon [`:`] character. A product value has two constituent parts, a name and a version, these being separated by the @ character. Figure 5-13 shows that 2 products are covered by the license, `tool_1` and `tool_2`. As can be seen `tool_1` version 1.6 is supported. Like the hosts property, regular expressions can be used.

- *Start Date*

  The start date specifies the time from when the license will active.

- *Expiry Date*

  The expiry date specifies the time after which the product will no longer operate.

- *Signature*

  The signature is a hash code protecting the contents of the file from unauthorized modification. This signature is derived using a highly lossey encoding of parts of the rest of the contents of the license. An MD5 hash of the resulting encoding is then used as the final signature and message authentication code minimizing the chances of reverse engineering its encryption [78, 79].

5.4 Summary

This chapter presented an effective means of facilitating transparent end-user access to the server-side IP libraries protected using authentication mechanisms. The main interface was developed as a schematic capture package allowing virtual ghost images of server-side macro-components to be graphically stitched together, manipulated and customized. Once a schematic representation of a design is complete, it can be verified on the server. This methodology has the unique ability to allow designers to construct low-power, high-performance digital designs using highly optimized libraries while protecting the intellectual property invested in those libraries. The server-side RTL cannot be read nor copied without specific access being granted. GUI elements for macro-components are based upon lightweight symbols which are independent of the module that they form wrappers for, unlike heavyweight solutions in which the symbols and RTL code are intertwined.

Chapter 6
Client-server integration

This chapter is intended to describe the interaction between the client and server and the processes operating on the server to support the client.

6.1 Client profile record

Before a person can access the services provided by this framework they must be registered on the server. Registration simply means having an account and corresponding profile on the server. In its simplest form, an account is nothing more than a username and password. To log onto the system, a client user must provide a valid username and password which should match an existing database entry. An account can thus be expired by simply removing a particular client's record from the database. If no matching username is found, the client application will not load, even if they have a valid license file.

```
+------------+--------------------+------+-----+---------+-------+
| Field      | Type               | Null | Key | Default | Extra |
+------------+--------------------+------+-----+---------+-------+
| User_Name  | varchar(16) binary |      | PRI |         |       |
| Password   | varchar(16)        |      |     |         |       |
| File_Space | varchar(255)       |      |     |         |       |
| First_Name | varchar(20)        |      |     |         |       |
| Last_Name  | varchar(20)        |      |     |         |       |
| Job_Title  | varchar(50)        | YES  |     | NULL    |       |
| Email      | varchar(50)        | YES  |     | NULL    |       |
| Public_Key | blob               | YES  |     | NULL    |       |
| Company_ID | int(11)            |      | MUL | 0       |       |
+------------+--------------------+------+-----+---------+-------+
```

Figure 6-1
Client database table schema

Figure 6-1 shows the database schema for the table maintaining client profile information. Each user can be uniquely identified by their username. A password field is also provided for use in authentication. The 'file space' field defines the path to the user's home directory on the server. Due to the nature of this framework, simulating and synthesizing designs requires the appropriate source files to physically exist on the server's file system. Each time a new client project is created, all files associated with that project will be stored on this location on the server.

The client table on its own is sufficient for individual users to access the services. However as noted in this thesis introduction, such a framework may be of particular interest to SME'. Hence provision for company information is by way of the table described in Figure 6-2. The client table supports a foreign key linking clients to their associated company.

```
+--------------------+--------------------+------+-----+---------+----------------+
| Field              | Type               | Null | Key | Default | Extra          |
+--------------------+--------------------+------+-----+---------+----------------+
| Company_ID         | int(11)            |      | PRI | NULL    | auto_increment |
| Company_Name       | varchar(50)        |      |     |         |                |
| Company_Password   | varchar(16) binary |      |     |         |                |
| Department         | varchar(100)       | YES  |     | NULL    |                |
| Street             | varchar(100)       | YES  |     | NULL    |                |
| City               | varchar(20)        | YES  |     | NULL    |                |
| State_County       | varchar(20)        | YES  |     | NULL    |                |
| Country            | varchar(20)        | YES  |     | NULL    |                |
| Zip_Post_Code      | varchar(20)        | YES  |     | NULL    |                |
| Company_Email      | varchar(30)        | YES  |     | NULL    |                |
| Company_Public_Key | blob               | YES  |     | NULL    |                |
| Company_Phone      | varchar(25)        | YES  |     | NULL    |                |
| Company_Fax        | varchar(25)        | YES  |     | NULL    |                |
| Company_Web        | varchar(50)        | YES  |     | NULL    |                |
+--------------------+--------------------+------+-----+---------+----------------+
```

Figure 6-2
Company database table schema

A novel feature of this framework is the ability to restrict the libraries to which individual clients have access to. Figure 6-3 shows the schema for a table that defines the relationships between clients and libraries. Adding or removing entries in this table enables or disables client's visibility to individual libraries respectively. Changes here will define the contents of the list of available macro-components described in section 5.1.1.

```
+-------------------+------------------+------+-----+---------+----------------+
| Field             | Type             | Null | Key | Default | Extra          |
+-------------------+------------------+------+-----+---------+----------------+
| Client_Library_ID | int(11)          |      | PRI | NULL    | auto_increment |
| User_Name         | char(16) binary  |      | MUL |         |                |
| Library_ID        | char(50) binary  |      | MUL |         |                |
+-------------------+------------------+------+-----+---------+----------------+
```

Figure 6-3
Client-library database table schema

6.2 Client-server session

After authentication, using the license management described in section 5.3, a server session is opened for a particular client instance. On the server side the session is maintained using a stateful session EJB. Note that a stateful bean was chosen rather than the more traditional route of using stateless beans of client sessions. This decision was taken as a result of the lengthy periods of time required for verification and synthesis of certain designs – which can be many hours and simplifies timeout issues which can results in stale handles held by the client.

This implementation should provide the framework for future enhancements to the platform that will allow the server session to be maintained tracking the progress of long synthesis and verification jobs. The client will be able to obtain a handle to the stateful session and store that on its local file system. Handles are a means of allowing EJB clients to re-establish a connection to a previously accessed bean instance. The client will then be free to disconnect from the server, in essence, log-off. They can then later log back on and using the handle, reconnect to their previous session to monitor the progress of the verification and/or synthesis processes.

In its present state, the client must maintain an open connection to the server while process jobs are running. Figure 6-4 provides a UML diagram of the client shell session EJB supporting this functionality.

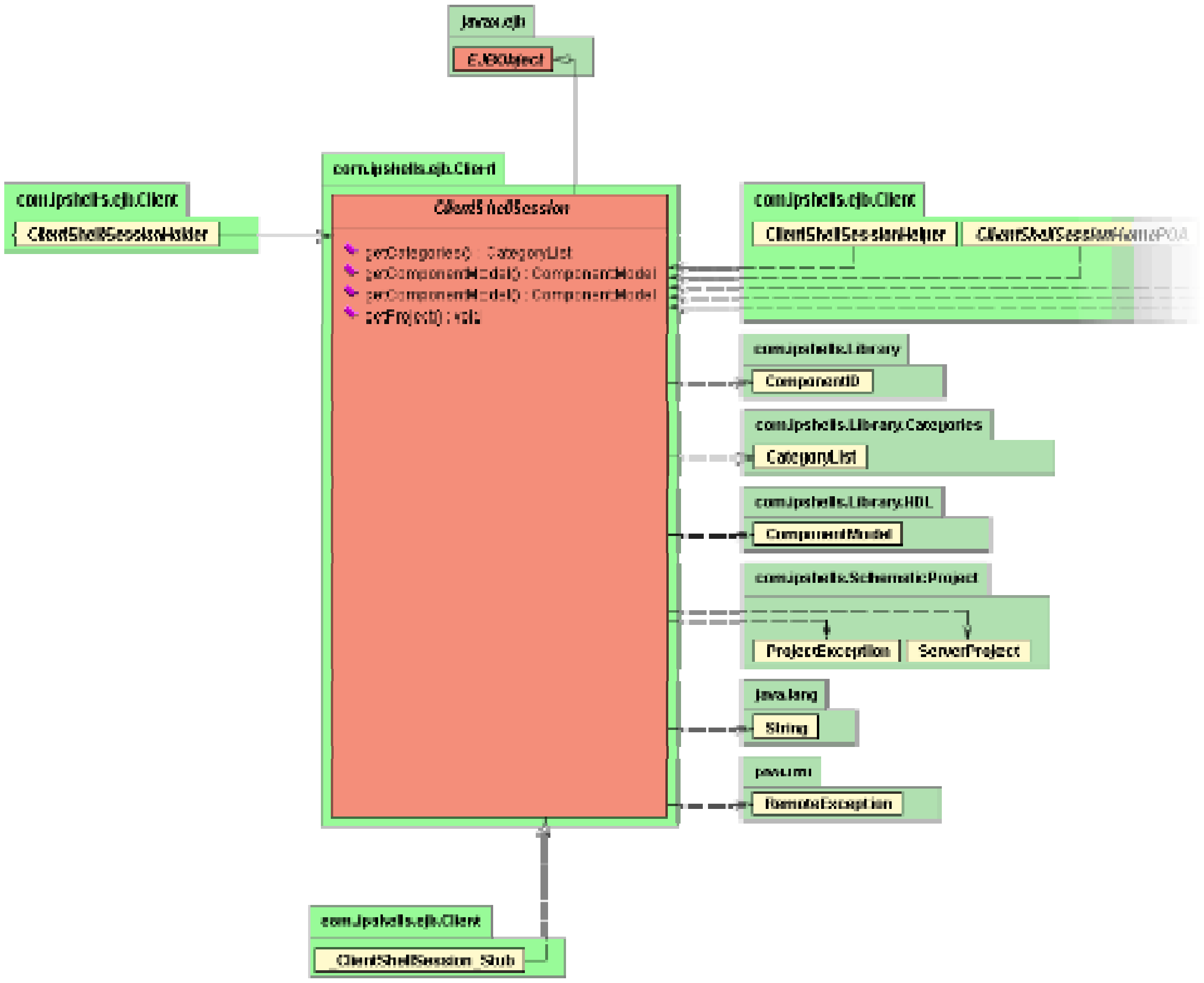


Figure 6-4
UML diagram of client shell session EJB

6.3 Project session

Each design in the client schematic capture tool is referred to as a project. In fact the client is able to have several projects open simultaneously and the user can switch back and forward between them. To do this, each client-side project must maintain a handle to a server-side project session EJB for which a UML description can be seen in Figure 6-5. This is actually an interface to real EJB object which is described in Figure 6-6.

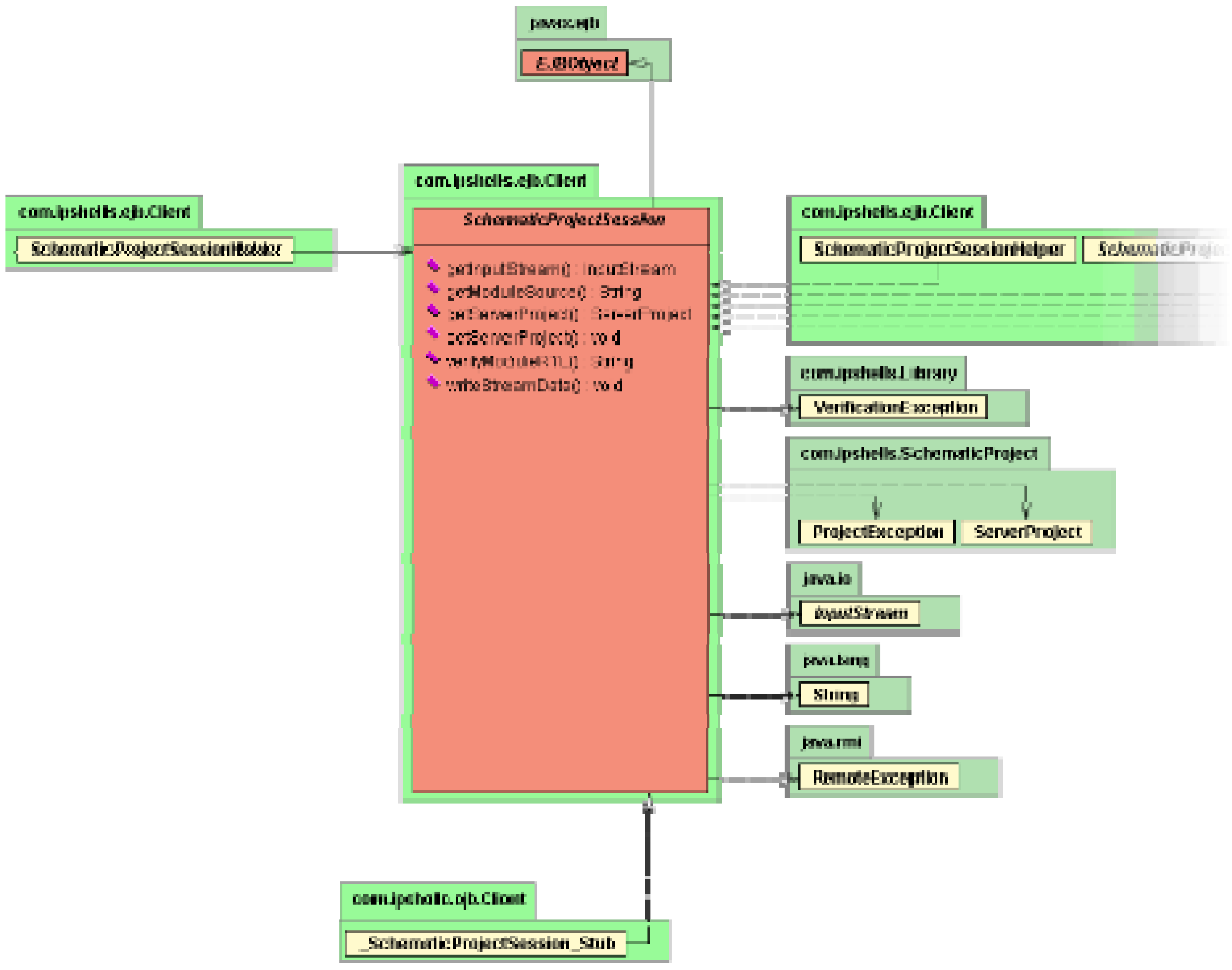


Figure 6-5
UML diagram of schematic project session EJB interface

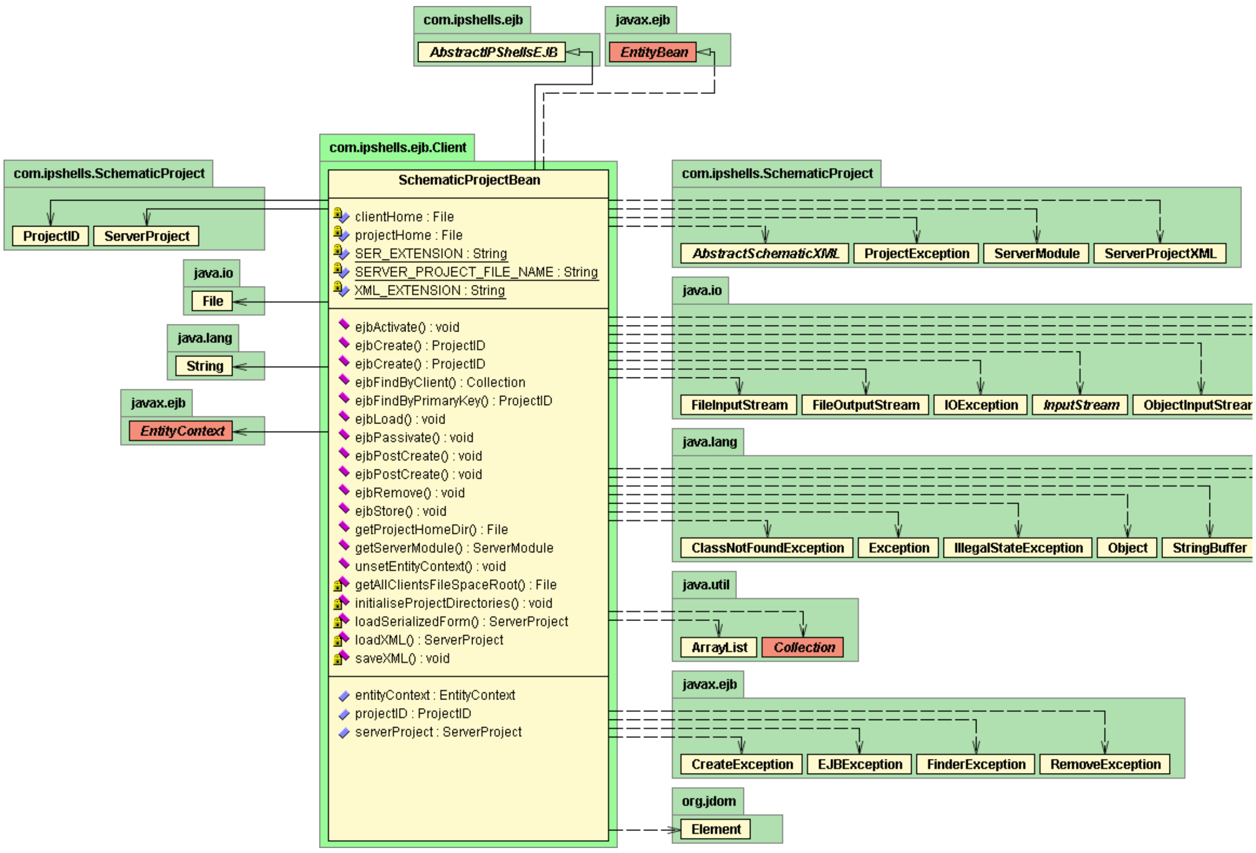


Figure 6-6
UML diagram of schematic project bean EJB

6.3.1 Server project persistence

Each project belonging to a client is, in the eyes of the server, an actual entity. Hence it is modelled using the entity EJB shown in Figure 6-6. Entity EJBs always have their data written into a persistence layer when not active. Typically this will be a SQL database or an XML file. However considering the nature of the data in this project, the chosen persistence layer is a physical directory on the server's file system. Section 6.1 stated that each client has a home directory on the server. All projects that a client has will be stored as directories inside their home directory. Projects within the home directory are identified using a unique ID code assigned to them when they are first created.

While building their design, the full project description is held locally in a designated directory on the client's workstation. When the client user needs access to EDA resources on the server, such as a verification tool, the locally held project description will be synchronized with the server. This happens by serializing the project objects and streaming them across the network/internet. Once on the server, project details are written to the appropriate server-side project directory as an XML file. Any RTL files that need to be dynamically generated are also written to this directory.

6.4 EDA tool-flow integration

One of the key features of this framework is that it allows client-side users to access server-side verification and synthesis tools. A novel aspect is that verification and synthesis can only be performed on the server – maintaining the security of the IP cores used as the design building blocks. Figure 5-6 shows that it is possible for the user to see the RTL code corresponding to their schematic design. It is even possible for them to save this code to a file on their local network where it could be accessed by their own EDA tool flows. Closer inspection of the RTL in Figure 5-6 reveals that it instantiates many other component, which only exist in the server-side macro-component libraries rendering the raw schematic translation RTL effectively useless to the client user.

6.4.1 Verification

When the client sends a request for a module to be verified, the synchronization process described in section 6.3.1 is first performed. The module to be verified is then extracted from the project description, translated into Verilog RTL and written as a file into the server project home directory. A similar process is then used to extract the testbench, if one has been included, from the project description. Before the module and testbench can be compiled, a list of all dependency modules is constructed using the processes defined in section 4.2. The appropriate verification tool is then invoked using the techniques to be described in section 6.4.3 – Running native applications in an enterprise environment. When the verification process is complete the results generated are streamed back to the client where they can be displayed, see Figure 5-8.

6.4.2 Module synthesis

Once the user is satisfied that their design is complete and functionally accurate it can be synthesized to a technology specific netlist representation. From the client's perspective, the synthesis process is very similar to that of verification. Again the project is synchronized with the server and the appropriate files extracted into the user's server-side disk space. A synthesis script corresponding to the module is then dynamically generated and written to the same directory. Appendix C shows an example synthesis script for the BuildGates design flow. This script is then executed and the resulting netlist returned to the user.

6.4.3 Running native applications in an enterprise environment

To provide key services, the application server must integrate itself with the EDA tool flow environment. This integration must, as a minimum, allow it to launch native applications such as verification and synthesis tools and feed back results to the client user. Two ways of doing this are described in the following sub-sections.

6.4.3.1 Java Native Interface (JNI)

The preferred way of controlling native applications within a Java environment is through the Java Native Interface (JNI). JNI is a framework specification and corresponding API developed by Sun Microsystems [80]. Through JNI methods and functions written in native languages such as C, C++ and assembly can be called as though they were locally available on the Java platform. To access functions via JNI, they must be available in the form of a platform specific libraries – meaning dynamic link libraries (DLL) on Microsoft Windows or shared objects (SO) on Unix/Linux.

JNI has been successfully used in used in many automated applications [81] and distributed processing environments [82], though this uses RMI as opposed to an application server. Other works have attempted similar things, but using CORBA as the communication protocol as opposed to RMI, allowing processes to be monitored and controlled across the internet [83].

Although JNI is the optimal solution for controlling native processes, it was not possible to be used in the framework at this time. To make use of JNI, the API (meaning the function and method names, arguments and return types) of the native libraries must be known. As the DLLs and SOs are proprietary to the vendor companies Cadence and Synopsys, the API information was unavailable.

6.4.3.2 Executing system commands

The alternative to JNI is to start and control the native application as a process from the Java environment. Java provides a series of **`Runtime.exec(`*`command`*`)`** methods which can be used to control native executables. As the native application tools will be operated within the server environment, they are controlled using a custom EJB stateless session bean. When a client requires access to an underlying EDA product the EJB will need to invoke and control the appropriate tool. However due to the restrictions imposed by the J2EE specifications [41], EJB’s are not allowed to manage threads. This is rather unfortunate as two threads are necessary to process to the data from the executable, one for the standard output stream and the other for the standard error stream. When an application such as a Verilog compiler is running, it is not uncommon for it to dump information detailing its current state and any incurred errors to the terminal from which it was started. More correctly, it will be printing data to two independent streams known as the standard output and the standard error. All general data will be sent to the standard output stream while any error messages will be written to the standard error stream. The effects of this will be noticed when attempting to redirect the output from a command to a file. The following is an example of using the UNIX command **`ls`**, which list directory contents.

```
linux % ls -lh existing.file non_existent.file
ls: non_existent.file: No such file or directory
-rw-rw-rw-    1 rig      None     1.4M Apr 17 21:10 existing.file
```

The command **`ls –lh`** is attempting to list the attributes of both an existing and a non-existing file. As the command succeeds for the existing file, the attributes of it are written to the standard output stream. However the command failed or the nonexistent file and an error message was generated and sent to the standard error stream. Note that the error message appeared first as the standard error has a higher priority than standard out.

The solution implemented to circumvent this limitation is to create a separate executable that controls the EDA tool, recording its output using two threads. The contents of these two threads are then merged together and sent only to the standard output. The EJB can then operate within the J2EE specification as it is now guaranteed that all information will be presented on a single stream. To allow the session bean to distinguish between the standard messages and error messages, a prefix is added to each line as shown below.

```
STD_ERR:>> ls: non_existent.file: No such file or directory
STD_OUT:>> -rw-rw-rw-  1 rig None 1.4M  Apr 17 21:10 existing.file
```

Interpreting the prefix of each line allows the server to once again differentiate between standard and error type messages. Standard data and error information can then be provided to the client through two independent feeds allowing for better tool interaction. For example all standard data, which may be a Verilog netlist, will appear within the client editor. If an error message needs to be displayed, it is generally more appropriate to do this using a pop-up dialog box.

6.4.4 Job farming

Deploying a system such as this within the EDA community would demand considerable resources on the server. A single server machine would suffice for running the enterprise application server, database server and user home directory file server. However simultaneously running verification and synthesis jobs will quickly saturate the server. Hence for these tasks, some form of job farming would be necessary. Most enterprise servers can be clustered, meaning that several servers can operate collectively as a single entity. However this would be uneconomical as an enterprise server would need to be installed on each workstation in the farm. Potential solutions in the literature include JavaSplit [84, 85] which allows for the transparent execution of multithreaded Java applications across a network of workstations.

6.5 Server-side EJBs

This section presents in diagrammatic form all of the server-side EJBs developed in the project. Figure 6-7 shows a UML representation of all the EJBs associated with maintaining and serving information regarding the macro-component libraries. Figure 6-8 depicts the EJBs maintaining the client metrics and project processing status.

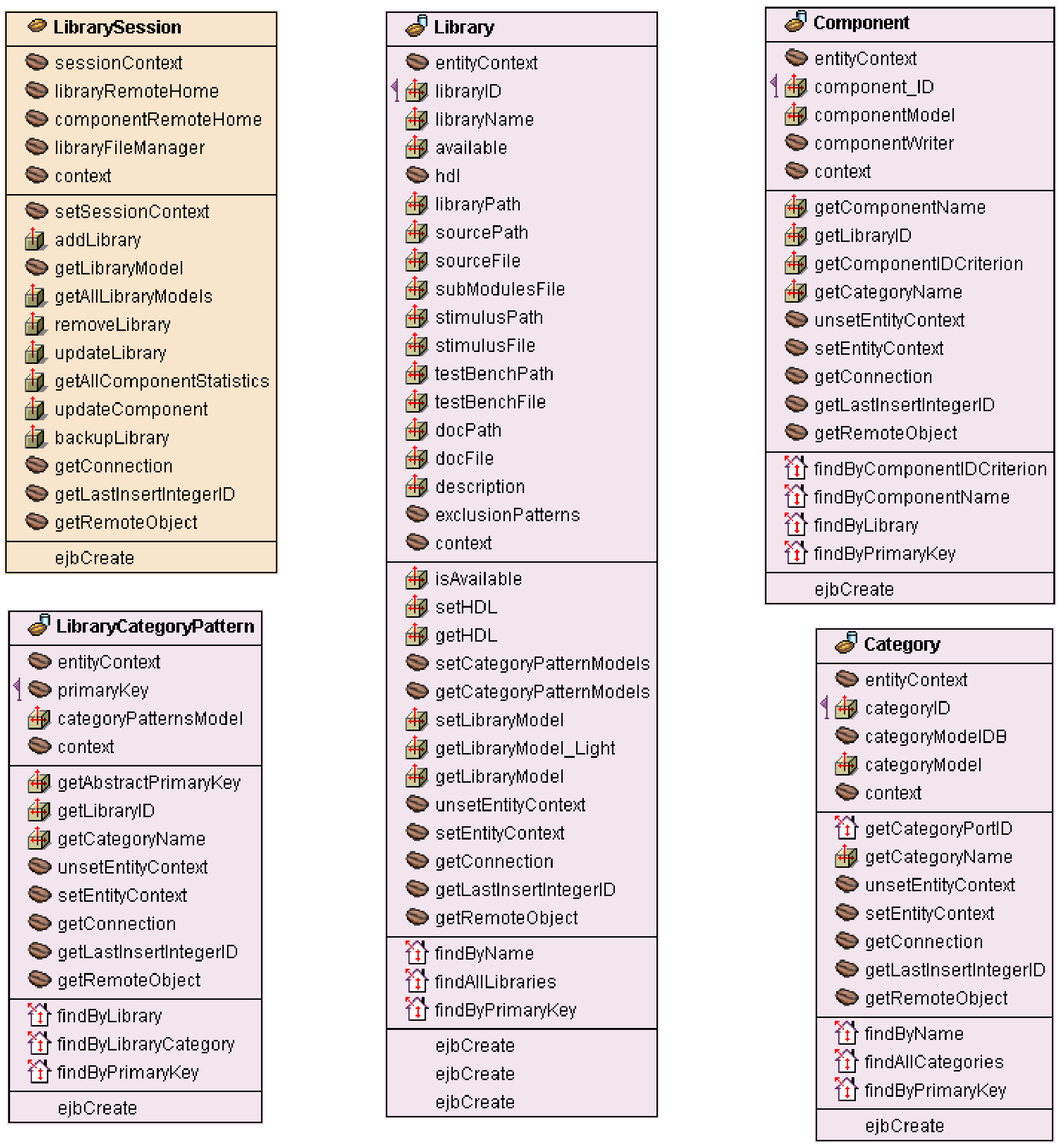


Figure 6-7
UML diagram of the EJBs used IP library modelling

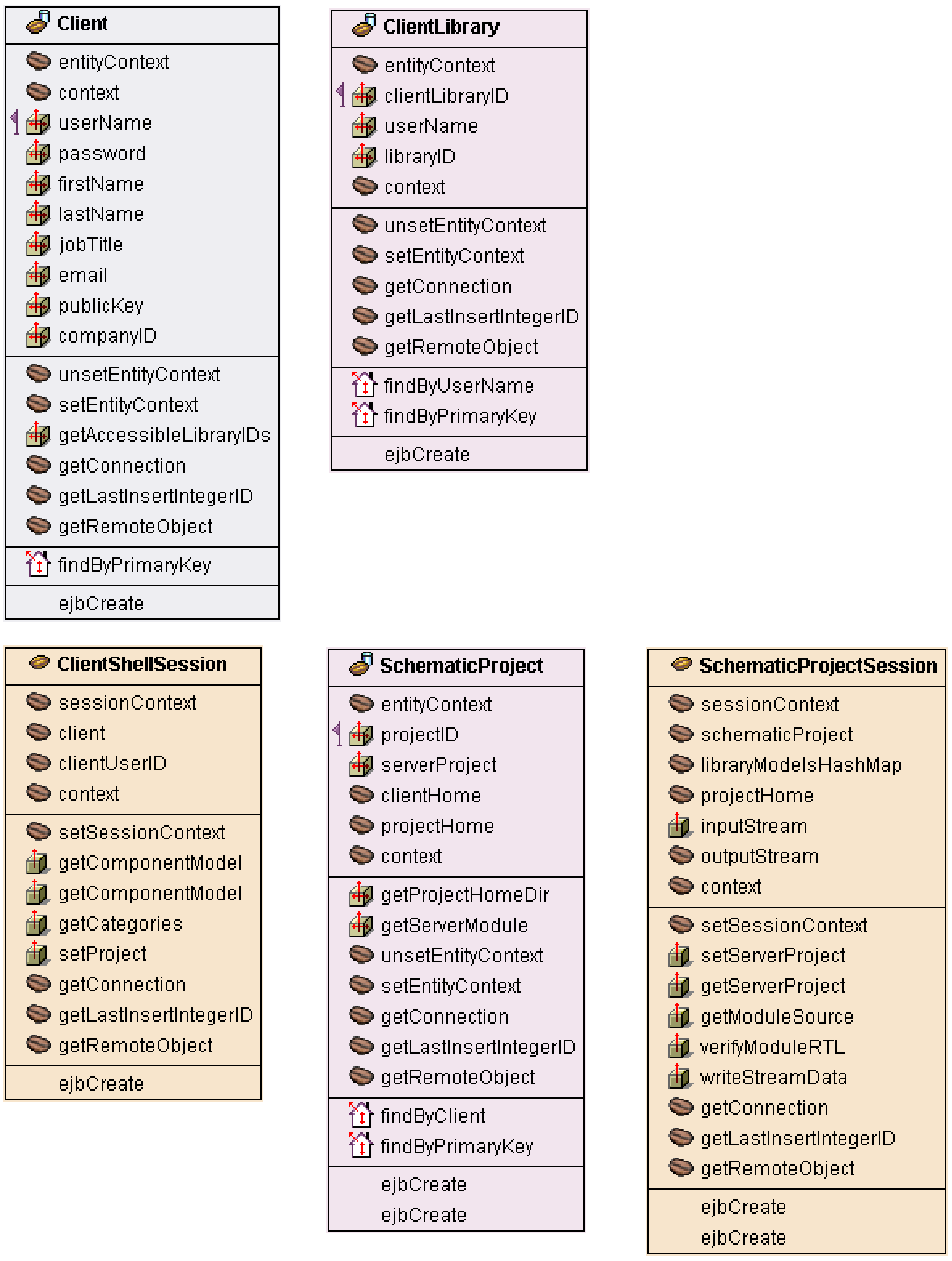


Figure 6-8
UML diagram of the client information and project interaction EJBs

6.6 Summary

This chapter described the major server-side services and how they have been made available to the client application. This began by describing a user account and the information necessary for a client session login. Following this was description of a project session and server-side persistence of project data.

J2EE servers are typically designed to deliver services that run within the enterprise environment. This particular application is different in that part of the service involves EDA tools that are completely independent and not directly linkable to the enterprise server without knowledge of their proprietary APIs. A solution has been developed that allows threading restrictions in the J2EE specification to be overcome. However this solution is not elegant and adds an extra layer and potential point of failure.

Chapter 7
Conclusions and future work

Today’s microelectronics industry is evolving at an extraordinary pace as can clearly be seen by the diversity of highly sophisticated, yet relatively cheap portable consumer electronic devices available from high street retailers. To survive in such a competitive market environment, vendors are compelled to rapidly diversify their products to provide new benefits to attract potential customers. For example, mobile phone manufacturers are extending their feature lists (GPRS and Bluetooth to name but a few), reducing their size (greater use of SoC technologies) and enhancing battery life (reducing power consumption). This requires access to both advanced system-level building blocks and state-of-the-art EDA tools.

The aims of this thesis were to research and develop a potential solution tackling some of the issues facing cost constrained digital system designers. This meant investigating and realising a platform that can provide access to both IP and EDA within a secure integrated environment. A suitable framework catering for this has been described in the previous six chapters and has since been patented by The University of Edinburgh.

This framework is built upon the J2EE standard giving rise to client-server based architecture. The server hosts extensive libraries of intellectual property macro-components and a suite of commercial EDA tools. These can be securely accessed by a specifically developed client-side schematic capture application. Within the schematic capture environment, users can construct digital systems using virtual instances of the macro-components and remotely verify and synthesis them.

The first objective of this research was to provide remote access to a set of intellectual property libraries. Through the developed architecture it is clearly possible to manipulate server-side IP modules from within the remotely installed schematic capture application. Intelligent database structuring also means that all server-side IP modules are categorised making them easy to search and identify.

The second objective was to protect the intellectual property invested in the macro-component libraries. The architecture of this platform means that the source code for the IP remains permanently on the server and completely inaccessible to users of the schematic capture environment. Indeed full RTL level verification and synthesis can even be performed without direct access to the RTL. This novel arrangement eliminates some of the issues associated with other IP protection methods such as pre-compilation, encryption and obfuscation. However at some point the client-side users of this platform will need to have access to the design in some form so that they can proceed towards tape-out. A solution to this problem has not been addressed in this thesis and could probably be investigated in a future study. One possible way to proceed would be to allow the client to only download the netlist of their final design. This should minimise the reverse-engineerability of the IP, especially for the complex algorithmic level optimisations employed in the macro-components as described in section 1.3.3.

The third objective was to facilitate access to remote EDA tool suites. This level of services has been incorporated into the platform allowing comprehensive RTL verification and very rudimentary synthesis. The client development environment provides the functionality to allow Verilog testbenches to be associated with each design schematic. During the verification process the project data is serialized and uploaded to the server and the appropriate tools invoked. The results produced by the verification tool are then streamed back to the client and displayed. This should allow the user to identify potential design issues and correct them. Synthesis can be performed in a similar manner but is currently feature limited. The process simply involves dynamically generating and executing a generic synthesis script, an example of which can be seen in appendix C. Future research projects based upon this platform may wish to investigate means of adding more options for synthesis such as types of optimisations. Both verification and synthesis are performed in this platform by invoking server-side shell commands from an EJB. This is a less than ideal approach and a method not directly supported by EJB containers due to threading issues. However a solution has been developed and implemented that mitigates the threading issues but still suffers from the unreliability associated with executing shell commands. Future work may wish to investigate the Java Native Interface (JNI) as a more suitable and stable means of controlling 3rd party applications.

The fourth objective was to integrate the first three objectives into a single integrated environment. This has been successfully achieved as the schematic capture application can effectively be used without the designer realising that they are working within a distributed environment. The client environment provides transparent access to server-side resources giving the impression that the IP libraries and EDA tools are locally available. This effectively abstracts away much of the complexity of the platform architecture and provides an interface with which digital design engineers will be immediately familiar.

The final objective was to ensure efficiency, scalability and reconfigurability of the platform. Overall the platform operates in a highly efficient manner and its real time speed is sufficient to compete with other commercial stand-alone products. Building upon the J2EE standard ensures a high degree of scalability. If the platform is to be deployed to a high volume of users the clustering facilities of the enterprise application server can be exploited to ensure an acceptable level of performance. In addition the developed architecture allows the IP libraries to easily expanded to accommodate more cores covering a greater diversity of algorithms and end-use applications. Undoubtedly the most processor intense aspect of this platform is from the EDA products used for verification and synthesis. To ensure maximum scalability future research may wish to investigate integrating an efficient job farming policy and load balancing arrangement. This should mean that the responsiveness of the platform is not compromised.

Reconfigurable aspects of the platform stem from extensive use of design patterns, most notably the façade design pattern. Intelligent use of these patterns enables the platform to be operated in one of several configurations. The main configuration is client-server mode, but it is also possible to bypass the server and interact with the database directly. This gives the same functionality but will not have the same degree of scalability. Alternatively the schematic capture application can operated in a conventional stand-alone mode writing its data into local XML files. The performance difference between these configurations is negligible, with a slight latency associated with remotely accessing the J2EE server or database server. The most noticeable difference occurs during launch of the client application which has to execute authentication procedures when accessing the J2EE server.

7.1 Future work

The conclusions mentioned above highlighted some key areas that could be explored in future research projects. However this section will identify some additional work that could be undertaken to further extend functionality.

One of the attempts of this research was to discover a means of facilitating efficient design methodologies based on macro-components. This means that tradeoffs can be made in designs by swapping sub-modules for alternatives of functional equivalency. This was demonstrated in section 5.1.3 where the schematic for a conventional FIR filter was converted to a more power efficient block-processing scheme. Research publications by the original authors of these filters prove that the principle of the macro-components is valid. However users of this platform with no prior experience of the macro-libraries will not know which modules should be embedded to optimise towards their particular constraints. Future research should therefore be targeted towards integrating a metrics database into the platform. This database should catalogue key characteristics such as power, area and speed – both delay and latency. If such a database is accessible to the schematic capture environment designers will have a better understanding of the tradeoffs available enabling them to create more efficient designs tailored towards their particular needs in a shorter period of time.

This platform is restricted to using modules available on the server-side IP libraries. However it is quite likely that many designers will wish to incorporated their own modules into the schematic. Some means of integrating both client and server side IPs into the platform would be a novel extension.

References

1. G. Moore, "*Cramming more components onto integrated circuits*" Electronics, vol. 38, no. 8, 1965. Available: ftp://download.intel.com/research/silicon/moorespaper.pdf
2. Mentor Graphics. Available: http://www.mentor.com/
3. W. C. Rhines, "*Moore's law is unconstitutional*", VLSI Design, 2005. 18th International Conference on 3-7 Jan. 2005, pp 31 – 32.
4. Shekhar Borkar, "*Obeying Moore's law beyond 0.18 micron*", ASIC/SOC Conference, 2000. Proceedings. 13th Annual IEEE International 13-16 Sept. 2000, pp 26 – 31.
5. Pierre J. Bricaud, "*IP reuse creation for system-on-a-chip design*", Custom Integrated Circuits, 1999. Proceedings of the IEEE 1999, 16-19 May 1999 pp 395 – 401.
6. Michael Keating & Pierre Bricaud, "*Reuse Methodology Manual for System-On-Chip Designs – Second Edition*", Kluwer Academic Publishers © 1999.
7. Synposys Products and solutions, DesignWare® Intellectual Property. Available: http://www.synopsys.com/products/designware/designware.html
8. David Besanko, David Dranove, Mark Shanley, "*Economics of Strategy, Second, Edition*", John Wiley & Sons © 2000, pp 109–135.
9. VSIA Alliance. Available: http://www.vsia.org
10. Christian S. Collberg & Clark Thomborson, "*Watermarking, tamper-proofing, and obfuscation - tools for software protection*", Software Engineering, IEEE Transactions on Volume 28, Issue 8, Aug. 2002 pp 735 – 746.
11. Diomidis Spinellis, "*Global analysis and transformations in preprocessed languages*", Software Engineering, IEEE Transactions on Volume 29, Issue 11, Nov. 2003, pp 1019 – 1030.
12. Tom Mens and Tom Tourwe, "*A survey of software refactoring*", Software Engineering, IEEE Transactions on Volume 30, Issue 2, Feb 2004, pp 126 – 139.
13. Eric Zwyssig, "*Low Power Digital Filter Design for Hearing Aid Applications*", MSc thesis: The University of Edinburgh, 9 October 2000.
14. Synoysys White Paper, "*Managing Power in Ultra Deep Submicron ASIC/IC Design*", May 2002.

15. A.T. Erdogan, E. Zwyssig, T. Arslan, “*Architectural Trade-offs in the Design of Low Power FIR Filtering Cores*”, IEE Proceedings - Circuits, Devices and Systems, Vol. 151, No. 1, 5 Feb. 2004, pp. 10–17.

16. T. Arslan and A. Erdogan, “Low power multiplication scheme for FIR filter implementation on single multiplier CMOS DSP processor”; IEEE Electronics Letters Vol. 32, No. 21 October 1996.

17. M. Hasan and T. Arslan, “*Implementation of low power FFT processor cores using a novel order-based processing scheme*”, IEE Proceedings - Circuits, Devices and Systems, Vol. 150, No. 3, 6 June 2003, pp. 149–154.

18. C.H. Wang, A.T. Erdogan, and T. Arslan, “*High Throughput and Low Power FIR Filtering IP Cores”*, IEEE International SOC Conference (SOCC 2004), pp. 127-130, Santa Clara, California, September 12-15, 2004.

19. Paper Hasan, M., Arslan, T., Thompson, J.S., “*A novel low power pipelined architecture for a MC-CDMA receiver*”, IEEE 3rd Int. Symposium on Image and Signal Processing and Analysis (ISPA 2003), Vol. 2, pp. 1048-1053, Rome, 18–20 Sep. 2003.

20. Aydin, N.; Arslan, T.; Cumming, D.R.S., “*Direct sequence CDMA based wireless interface for an integrated sensor microsystem*”, 4th Int. IEEE EMBS Special Topic Conference on Information Technology Applications in Biomedicine, Birmingham, UK, 24-26 April 2003, pp. 370–373 S.

21. S. Masupe, T. Arslan, “*Low power order based DCT processing algorithm*”, The 2001 IEEE Int. Symposium on Circuits and Systems (ISCAS'2001), Sydney, Australia, 6-9 May 2001, Vol. 2, pp. 5–8.

22. K.C.B. Tan and T. Arslan, “*An Embedded Extension Algorithm for the Lifting based Discrete Wavelet Transform in JPEG2000”*, 2002 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2002), 13-17 May 2002, Volume: 4, pp. 3513-3516.

23. Olivier Coudert, “*Timing and design closure in physical design flows*”, Quality Electronic Design, 2002. Proceedings. International Symposium on 2002, pp 511 – 516.

24. Vineet Sahula, C.P.Ravikumar, D.Nagchoudhuri, “*Improvement of ASIC design processes*” Design Automation Conference, 2002. Proceedings of ASP-DAC 2002. 7th Asia and South Pacific and the 15th International Conference on VLSI Design. Proceedings. 7-11 Jan. 2002, pp 105 – 110.

25. Lionel Bening & Harry Foster, “*Optimizing multiple EDA tools within the ASIC design flow*”, Design & Test of Computers, IEEE, Volume 18, Issue 4, July-Aug. 2001, pp 46 – 55.

26. Sumit DasGupta, "*Looking back, looking around*", Design & Test of Computers, IEEE, Volume 21, Issue 4, July-Aug. 2004, pp 271 – 273.

27. Ahmed Hemani, "*Charting the EDA roadmap*", Circuits and Devices Magazine, IEEE, Volume 20, Issue 6, Nov.-Dec. 2004, pp 5 – 10.

28. Tughrul Arslan, Robert Graham, Robert Thomson, 2004, "*System and Method for Rapid Prototyping of ASIC Systems*", The University of Edinburgh, International patent number: WO2004068535, 2004-08-12.

29. The Virtual Component Exchange. Available: http://www.thevcx.com/

30. Design and Reuse. Available: http://www.us.design-reuse.com/

31. Tushar K. Hazra, "*Building enterprise portals: principles to practice*", Software Engineering, 2002. ICSE 2002. Proceedings of the 24th International Conference on 2002, pp 623 – 633.

32. William H. Robertson, James M. Plymale, 2000, "*Method and system for facilitating electronic circuit and chip design using remotely located resources*", Cadence Design Systems, Inc, International patent number: W0 01/65422 A2.

33. William H. Robertson, James M. Plymale, 2002, "*Method and system for chip design using remotely located resources*", Cadence Design Systems, Inc, United State Patent Application: 20020188910, 2003-07-15.

34. Don Brown, 2001, "*Electronic product design system*", State Patent Application: 20020156757.

35. John Grundy, Zhong Wei, Radu Nicolescu & Yuhong Cai, "*An environment for automated performance evaluation of J2EE and ASP.NET thin-client architectures*", Software Engineering Conference, 2004. Proceedings. 2004 pp 300 – 308.

36. Yan Liu & Ian Gorton, "*An Empirical Evaluation of Architectural Alternatives for J2EE and Web Services*", Software Engineering Conference, 2004. 11th Asia-Pacific, 30-03 Nov. 2004, pp 10 – 17.

37. Jochen Mades, Thomas Schneider, Manfred Glesner, Andre Windisch & Wolfgang Ecker, "*A JAVA-based mixed-signal design environment*", Integrated Circuits and Systems Design, 2000. Proceedings. 13th Symposium on 18-24 Sept. 2000, pp 301 – 306.

38. João M. P. Cardoso & Horácio Neto, "*Compilation for FPGA-based reconfigurable hardware*", Design & Test of Computers, IEEE, Volume 20, Issue 2, March-April 2003. pp 65 – 75.

39. Ann Wollrath, Jim Waldo & Roger Riggs, "*Java-centric distributed computing*", IEEE Micro, Volume 17, Issue 3, May-June 1997, pp 44 – 53.

40. Eric Altendorf, Moses Hohman & Roman Zabicki, "*Using J2EE on a large, Web-based project*", Software, IEEE, Volume: 19 Issue: 2 Mar/Apr 2002, pp 81 – 89.

41. Sun Microsystems, "*Java™ 2 Platform Enterprise Edition Specification, v1.4*", Available: http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf

42. Richard Monson-Haefle, "Enterprise JavaBeans – 3rd Edition", O'Reilly & Associates © 2001.

43. BEA Systems. Available: http://www.bea.com

44. JBOSS. Available: http://www.jboss.org

45. Borland, JBuilder. Available: http://www.borland.co.uk/jbuilder

46. Borland, Borland Enterprise Server. Available: http://www.borland.co.uk/bes

47. Jiang Guo, Yuehong Liao & Behzad Parviz, "*A survey of J2EE application performance management systems*", Web Services, 2004. Proceedings. IEEE International Conference on 6-9 July 2004, pp 724 – 731.

48. Elliotte Rusty Harold & W. Scott Means, "*XML in a Nutshell*", O'Reilly & Associates ©2001

49. Kevin E. Kline & Daniel Kline, "*SQL in a Nutshell - Second Edition*", O'Reilly & Associates ©2004

50. PostgreSQL. Available: http://www.postgres.org

51. MySQL. Available: http://www.mysql.com

52. MySQL, Connector/J: http://www.mysql.com/products/connector/j/

53. George Reese, "Database *Programming with JDBC and Java – Second Edition*", O'Reilly & Associates © 2000.

54. Huwei Guan, Ip, H.H.S. & Yanchun Zhang, "*Java-based approaches for accessing databases on the Internet and a JDBC-ODBC implementation*", Computing & Control Engineering Journal Volume 9, Issue 2, April 1998 pp 71 – 78.

55. The Jakarta Project, The DBCP Component. Available: http://jakarta.apache.org/commons/dbcp/

56. Icarus Verilog. Available: http://www.icarus.com/eda/verilog/

57. Ana-Maria Cretu, Voicu Groza, Abdul Al-Dhaher & Rami Abielmona, "*Performance evaluation of a software cluster*", Instrumentation and Measurement Technology Conference, 2002. IMTC/2002. Proceedings of the 19th IEEE, Volume 2, 21-23 May 2002, pp 1543 – 1548.

58. Erich Gamma, Richard Helm, Ralph Johnson & John Vissides "*Design Patterns: elements of reusable object-oriented software*", Addison-Wesley ©1995.

59. Lutz Prechelt, Barbra Unger, Walter F. Tichy, Peter Brössler & Lawrence G. Votta, "*A controlled experiment in maintenance: comparing design patterns to simpler solutions*", Software Engineering, IEEE Transactions on Volume 27, Issue 12, Dec. 2001, pp 1134 – 1144.

60. Stuart A. Yeates & Michel de Champlain, "*Design patterns in garbage collection*" Technology of Object-Oriented Languages and Systems, 1997. TOOLS 25, Proceedings, 24-28 Nov. 1997, pp 80 – 98.

61. H.M.Deitel, P.J.Deitle & S.E.Santry, "*Advanced Java 2 Platform*", Prentice Hall ©2002, Page 86.

62. Michael J. Mahemoff & Lorraine J. Johnston, "*Handling multiple domain objects with Model-View-Controller*", Technology of Object-Oriented Languages and Systems, 1999. TOOLS 32. Proceedings 22-25 Nov. 1999, pp 28 – 39.

63. Yuetang Deng, Phyllis Frankl & Zhongqiang Chen, "*Testing database transaction concurrency*", Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on 6-10 Oct. 2003, pp 184 – 193.

64. InnoDB – Transaction, row level locking, hot backup and foreign keys of MySQL, Available: http://www.innodb.com

65. Jeffrey E. F. Friedl, "*Mastering Regular Expressions*", O'Reilly; 2 edition ©July 2002

66. The Apache Jakarta Project, Jakarta ORO. Available: http://jakarta.apache.org/oro/

67. Samir Palnitkar, "Verilog *HDL – A Guide to Digital Design and Synthesis*", SunSoft Press A Prentice Hall Title © 1996.

68. Janick Bergeron, "*Writing Testbenches Functional Verification of HDL Models*", Kluwer Academic Publishers © 2000.

69. IEEE Standard Hardware Description Language, IEEE Std 1364-2001.

70. Synopsys DesignWare Developers Guide, March 2004. Available: http://www.synopsys.com/products/designware/docs/doc/dwf/manuals/dwdg.pdf

71. Swapnajit Mittra, "*VIP: a Verilog Interpreter for Preprocessing*", Verilog HDL Conference, 1996. Proceedings., 1996 IEEE International 26-28 Feb. 1996, pp 34 – 38.

72. Anthony Cox & Charles Clarke, "*A comparative evaluation of techniques for syntactic level source code analysis*", Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific, 5-8 Dec. 2000, pp 282 – 289.

73. Avraham Leff & James T. Rayfield, "*Improving application throughput with enterprise JavaBeans Caching*", Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on 19-22 May 2003, pp 244 – 251.

74. Ahmet T. Erdogan and Tughrul Arslan, "*Low Power Block-based FIR Filtering Cores*", 2003 IEEE International Symposium on Circuits and Systems (ISCAS 2003), MAY 25-28, 2003, vol. 5, pp. 341 – 344.

75. Irah I. Donner, "*Don't judge a software license by its cover*", Computer, Volume 29, Issue 10, Oct. 1996, pp 114 – 115.

76. Moez Limayem, Mohamed Khalifa & Wynne W. Chin, "*Factors motivating software piracy: a longitudinal study*", Engineering Management, IEEE Transactions on Volume 51, Issue 4, Nov. 2004, pp 414 – 425.

77. Macrovision, FLEXlm. Available: http://www.macrovision.com/products/legacy_products/flexlm/index.shtml

78. Peter Gutmann, David Naccache and Charles C. Palmer, "*When hashes collide [applied cryptography]*", Security & Privacy Magazine, IEEE, Volume 3, Issue 3, May-June 2005, pp: 68 – 71.

79. Bart Preneel and Paul C. van Oorschot, "*On the security of iterated message authentication codes*", Information Theory, IEEE Transactions on Volume 45, Issue 1, Jan. 1999, pp:188 – 199.

80. Sun Microsystems, "*Java Native Interface Specification*" Available: http://java.sun.com/j2se/1.4.2/docs/guide/jni/spec/jniTOC.html

81. Grzegorz Czajkowski, Laurent Daynès & Mario Wolczko, "*Automated and portable native code isolation*", Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on 27-30 Nov. 2001, pp 298 – 307.

82. Masanobu Koga & Yuli Kikukawa, "*Integration of platform-dependent simulators using distributed object and native language interface*", SICE 2002. Proceedings of the 41st SICE Annual Conference Volume 1, 5-7 Aug. 2002, pp 374 – 379.

83. Lian Chen & Armin Eberlein, "*A framework of a Web-based distributed control system*", Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on Volume 2, 4-7 May 2003, pp 1351 – 1354.

84. Michael Factor, Assaf Schuster and Konstantin Shagin, "*JavaSplit: a runtime for execution of monolithic Java programs on heterogenous collections of commodity*

workstations”, Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on 2003, pp: 110 – 117.

85. Michael Factor, Assaf Schuster and Konstantin Shagin, *“A distributed runtime for Java: yesterday and today”*, Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International 26-30 April 2004, pp:159.

Appendix A
Java source files

```
 1 com/ipshells/AbstractGUIMain.java
 2 com/ipshells/AbstractIPShellAction.java
 3 com/ipshells/AbstractMain.java
 4 com/ipshells/DatabaseConnection.java
 5 com/ipshells/EditorPanel.java
 6 com/ipshells/ExecMain.java
 7 com/ipshells/Library/AbstractVerifySynthesize.java
 8 com/ipshells/Library/Categories/CategoriesTestClient.java
 9 com/ipshells/Library/Categories/Category.java
10 com/ipshells/Library/Categories/CategoryBean.java
11 com/ipshells/Library/Categories/CategoryComponents.java
12 com/ipshells/Library/Categories/CategoryException.java
13 com/ipshells/Library/Categories/CategoryGroup.java
14 com/ipshells/Library/Categories/CategoryHome.java
15 com/ipshells/Library/Categories/CategoryList.java
16 com/ipshells/Library/Categories/CategoryManagerMain.java
17 com/ipshells/Library/Categories/CategoryModel.java
18 com/ipshells/Library/Categories/CategoryModelDB.java
19 com/ipshells/Library/Categories/CategoryModelEJB.java
20 com/ipshells/Library/Categories/CategoryModelPanel.java
21 com/ipshells/Library/Categories/CategoryModelPersistence.java
22 com/ipshells/Library/Categories/CategoryModelXML.java
23 com/ipshells/Library/Categories/CategoryPatternsModel.java
24 com/ipshells/Library/Categories/CategoryPersistenceManager.java
25 com/ipshells/Library/Categories/CategoryPortModel.java
26 com/ipshells/Library/Categories/CategoryPortModelPanel.java
27 com/ipshells/Library/Categories/CategoryPortPatternsModel.java
28 com/ipshells/Library/Categories/CategoryRemote.java
29 com/ipshells/Library/Categories/CategoryRemoteHome.java
30 com/ipshells/Library/Categories/CategoryTreePanel.java
31 com/ipshells/Library/ComponentID.java
32 com/ipshells/Library/ComponentStatistics.java
33 com/ipshells/Library/ComponentUpdatePanel.java
34 com/ipshells/Library/ExclusionPatterns.java
35 com/ipshells/Library/HDL/ComponentDocumentationModel.java
36 com/ipshells/Library/HDL/ComponentModel.java
37 com/ipshells/Library/HDL/ComponentModelXML.java
38 com/ipshells/Library/HDL/ComponentTest.java
39 com/ipshells/Library/HDL/DatabaseComponentReader.java
40 com/ipshells/Library/HDL/DatabaseComponentWriter.java
41 com/ipshells/Library/HDL/HDLParsingException.java
42 com/ipshells/Library/HDL/HDLSourceReader.java
43 com/ipshells/Library/HDL/Parameter.java
44 com/ipshells/Library/HDL/ParameterExpression.java
45 com/ipshells/Library/HDL/ParameterModel.java
46 com/ipshells/Library/HDL/Port.java
47 com/ipshells/Library/HDL/PortModel.java
48 com/ipshells/Library/HDL/PortParameterContainer.java
49 com/ipshells/Library/HDL/Verilog/Generators/AbstractHDLGenerator.java
50 com/ipshells/Library/HDL/Verilog/Generators/AbstractVerilogGenerator.java
51 com/ipshells/Library/HDL/Verilog/Generators/Definition.java
52 com/ipshells/Library/HDL/Verilog/Generators/DemuxGenerator.java
53 com/ipshells/Library/HDL/Verilog/Generators/GUI/EditorPanel.java
54 com/ipshells/Library/HDL/Verilog/Generators/GUI/FileMenu.java
55 com/ipshells/Library/HDL/Verilog/Generators/GUI/GeneratorActions.java
56 com/ipshells/Library/HDL/Verilog/Generators/GUI/GeneratorFrame.java
57 com/ipshells/Library/HDL/Verilog/Generators/GUI/HDLGeneratorMain.java
58 com/ipshells/Library/HDL/Verilog/Generators/GUI/HelpMenu.java
59 com/ipshells/Library/HDL/Verilog/Generators/GUI/MainMenuBar.java
60 com/ipshells/Library/HDL/Verilog/Generators/GUI/ModuleConfigPanel.java
61 com/ipshells/Library/HDL/Verilog/Generators/GUI/ModuleControlPanel.java
62 com/ipshells/Library/HDL/Verilog/Generators/GUI/SubModulePanel.java
63 com/ipshells/Library/HDL/Verilog/Generators/GUI/VariablesPanel.java
64 com/ipshells/Library/HDL/Verilog/Generators/GeneratorSkeleton.java
65 com/ipshells/Library/HDL/Verilog/Generators/HDLGenerator.java
66 com/ipshells/Library/HDL/Verilog/Generators/ModuleVariablesModel.java
67 com/ipshells/Library/HDL/Verilog/Generators/MuxGenerator.java
68 com/ipshells/Library/HDL/Verilog/Generators/Parameter.java
```

```
 69  com/ipshells/Library/HDL/Verilog/Generators/RAMCorrelatorGenerator.java
 70  com/ipshells/Library/HDL/Verilog/Generators/RAMEnableGenerator.java
 71  com/ipshells/Library/HDL/Verilog/Generators/RAMGenerator.java
 72  com/ipshells/Library/HDL/Verilog/Generators/ShiftLeftGenerator.java
 73  com/ipshells/Library/HDL/Verilog/Generators/SubModule.java
 74  com/ipshells/Library/HDL/Verilog/Generators/Variable.java
 75  com/ipshells/Library/HDL/Verilog/VerilogInteger.java
 76  com/ipshells/Library/HDL/Verilog/VerilogObjectNameTextField.java
 77  com/ipshells/Library/HDL/Verilog/VerilogParsingException.java
 78  com/ipshells/Library/HDL/Verilog/VerilogSourceReader.java
 79  com/ipshells/Library/HDL/Verilog/VerilogSyntax.java
 80  com/ipshells/Library/LibraryCategoryPatternPanel.java
 81  com/ipshells/Library/LibraryConfigDialog.java
 82  com/ipshells/Library/LibraryConfigPanel.java
 83  com/ipshells/Library/LibraryException.java
 84  com/ipshells/Library/LibraryFileManager.java
 85  com/ipshells/Library/LibraryManager.java
 86  com/ipshells/Library/LibraryManagerMain.java
 87  com/ipshells/Library/LibraryManagerPanel.java
 88  com/ipshells/Library/LibraryModel.java
 89  com/ipshells/Library/LibraryModelPersistence.java
 90  com/ipshells/Library/LibraryPathResolver.java
 91  com/ipshells/Library/ModuleContainerDependencies.java
 92  com/ipshells/Library/ModuleDependencies.java
 93  com/ipshells/Library/ModuleDependenciesMain.java
 94  com/ipshells/Library/VerificationException.java
 95  com/ipshells/Library/Verify.java
 96  com/ipshells/Library/VerifyMain.java
 97  com/ipshells/Library/Verify_Icarus.java
 98  com/ipshells/License.java
 99  com/ipshells/LicenseSignature.java
100  com/ipshells/LicenseWriter.java
101  com/ipshells/Math/BinaryOperators.java
102  com/ipshells/Math/PostfixCalculator.java
103  com/ipshells/Math/PostfixCompiler.java
104  com/ipshells/Math/PostfixObjectCompiler.java
105  com/ipshells/Math/UnaryOperators.java
106  com/ipshells/MemoryStatsPanel.java
107  com/ipshells/PersistenceModel.java
108  com/ipshells/ProcessExecMain.java
109  com/ipshells/SchematicProject/AbstractSchematicXML.java
110  com/ipshells/SchematicProject/ComponentPropertiesPanel.java
111  com/ipshells/SchematicProject/ModuleCanvas.java
112  com/ipshells/SchematicProject/ModuleException.java
113  com/ipshells/SchematicProject/ModuleObjects/AbstractCircuitComponent.java
114  com/ipshells/SchematicProject/ModuleObjects/AbstractCircuitIO.java
115  com/ipshells/SchematicProject/ModuleObjects/AbstractCircuitObject.java
116  com/ipshells/SchematicProject/ModuleObjects/Adder.java
117  com/ipshells/SchematicProject/ModuleObjects/AndGate.java
118  com/ipshells/SchematicProject/ModuleObjects/ArithmeticUnit.java
119  com/ipshells/SchematicProject/ModuleObjects/Bus.java
120  com/ipshells/SchematicProject/ModuleObjects/CircuitInput.java
121  com/ipshells/SchematicProject/ModuleObjects/CircuitOutput.java
122  com/ipshells/SchematicProject/ModuleObjects/Combiner.java
123  com/ipshells/SchematicProject/ModuleObjects/Compressor.java
124  com/ipshells/SchematicProject/ModuleObjects/Counter.java
125  com/ipshells/SchematicProject/ModuleObjects/Demultiplexer.java
126  com/ipshells/SchematicProject/ModuleObjects/Divider.java
127  com/ipshells/SchematicProject/ModuleObjects/FFT.java
128  com/ipshells/SchematicProject/ModuleObjects/FIRController.java
129  com/ipshells/SchematicProject/ModuleObjects/Filter.java
130  com/ipshells/SchematicProject/ModuleObjects/FlipFlop.java
131  com/ipshells/SchematicProject/ModuleObjects/IncompatibleConnectionException.java
132  com/ipshells/SchematicProject/ModuleObjects/InputPort.java
133  com/ipshells/SchematicProject/ModuleObjects/InstantiateComponent.java
134  com/ipshells/SchematicProject/ModuleObjects/Inverter.java
135  com/ipshells/SchematicProject/ModuleObjects/Latch.java
136  com/ipshells/SchematicProject/ModuleObjects/MAC.java
137  com/ipshells/SchematicProject/ModuleObjects/ModuleContainer.java
138  com/ipshells/SchematicProject/ModuleObjects/ModuleParameter.java
139  com/ipshells/SchematicProject/ModuleObjects/ModulePort.java
140  com/ipshells/SchematicProject/ModuleObjects/ModuleVerilogGenerator.java
141  com/ipshells/SchematicProject/ModuleObjects/MultipleDriverException.java
142  com/ipshells/SchematicProject/ModuleObjects/Multiplexer.java
143  com/ipshells/SchematicProject/ModuleObjects/Multiplier.java
144  com/ipshells/SchematicProject/ModuleObjects/NandGate.java
145  com/ipshells/SchematicProject/ModuleObjects/NorGate.java
```

```
146  com/ipshells/SchematicProject/ModuleObjects/OrGate.java
147  com/ipshells/SchematicProject/ModuleObjects/OutputPort.java
148  com/ipshells/SchematicProject/ModuleObjects/RAM.java
149  com/ipshells/SchematicProject/ModuleObjects/ROM.java
150  com/ipshells/SchematicProject/ModuleObjects/Receiver.java
151  com/ipshells/SchematicProject/ModuleObjects/Shifter.java
152  com/ipshells/SchematicProject/ModuleObjects/Subtractor.java
153  com/ipshells/SchematicProject/ModuleObjects/Symbols/AbstractComponentSymbol.java
154  com/ipshells/SchematicProject/ModuleObjects/Symbols/AbstractSymbol.java
155  com/ipshells/SchematicProject/ModuleObjects/Symbols/AdderSymbol.java
156  com/ipshells/SchematicProject/ModuleObjects/Symbols/AndGateSymbol.java
157  com/ipshells/SchematicProject/ModuleObjects/Symbols/ArithmeticUnitSymbol.java
158  com/ipshells/SchematicProject/ModuleObjects/Symbols/BusSymbol.java
159  com/ipshells/SchematicProject/ModuleObjects/Symbols/BusSymbolLine.java
160  com/ipshells/SchematicProject/ModuleObjects/Symbols/CircuitInputSymbol.java
161  com/ipshells/SchematicProject/ModuleObjects/Symbols/CircuitObjectSymbol.java
162  com/ipshells/SchematicProject/ModuleObjects/Symbols/CircuitOutputSymbol.java
163  com/ipshells/SchematicProject/ModuleObjects/Symbols/CombinerSymbol.java
164  com/ipshells/SchematicProject/ModuleObjects/Symbols/ComponentSymbol.java
165  com/ipshells/SchematicProject/ModuleObjects/Symbols/CompressorSymbol.java
166  com/ipshells/SchematicProject/ModuleObjects/Symbols/DeMultiplexerSymbol.java
167  com/ipshells/SchematicProject/ModuleObjects/Symbols/FFTSymbol.java
168  com/ipshells/SchematicProject/ModuleObjects/Symbols/FIRControllerSymbol.java
169  com/ipshells/SchematicProject/ModuleObjects/Symbols/FIRSymbol.java
170  com/ipshells/SchematicProject/ModuleObjects/Symbols/FlipFlopSymbol.java
171  com/ipshells/SchematicProject/ModuleObjects/Symbols/InstantiateComponentSymbol.java
172  com/ipshells/SchematicProject/ModuleObjects/Symbols/InverterSymbol.java
173  com/ipshells/SchematicProject/ModuleObjects/Symbols/LatchSymbol.java
174  com/ipshells/SchematicProject/ModuleObjects/Symbols/ModuleSymbols.java
175  com/ipshells/SchematicProject/ModuleObjects/Symbols/MultiplexerSymbol.java
176  com/ipshells/SchematicProject/ModuleObjects/Symbols/MultiplierSymbol.java
177  com/ipshells/SchematicProject/ModuleObjects/Symbols/NandGateSymbol.java
178  com/ipshells/SchematicProject/ModuleObjects/Symbols/NorGateSymbol.java
179  com/ipshells/SchematicProject/ModuleObjects/Symbols/OrGateSymbol.java
180  com/ipshells/SchematicProject/ModuleObjects/Symbols/PortSymbol.java
181  com/ipshells/SchematicProject/ModuleObjects/Symbols/RAMSymbol.java
182  com/ipshells/SchematicProject/ModuleObjects/Symbols/ROMSymbol.java
183  com/ipshells/SchematicProject/ModuleObjects/Symbols/ReceiverSymbol.java
184  com/ipshells/SchematicProject/ModuleObjects/Symbols/SubtractorSymbol.java
185  com/ipshells/SchematicProject/ModuleObjects/Symbols/SymbolException.java
186  com/ipshells/SchematicProject/ModuleObjects/Symbols/XnorGateSymbol.java
187  com/ipshells/SchematicProject/ModuleObjects/Symbols/XorGateSymbol.java
188  com/ipshells/SchematicProject/ModuleObjects/Wire.java
189  com/ipshells/SchematicProject/ModuleObjects/XnorGate.java
190  com/ipshells/SchematicProject/ModuleObjects/XorGate.java
191  com/ipshells/SchematicProject/ModulePanel.java
192  com/ipshells/SchematicProject/ModulePropertiesPanel.java
193  com/ipshells/SchematicProject/ModuleXML.java
194  com/ipshells/SchematicProject/NewProjectPanel.java
195  com/ipshells/SchematicProject/ParameterPropertiesPanel.java
196  com/ipshells/SchematicProject/Project.java
197  com/ipshells/SchematicProject/ProjectActions.java
198  com/ipshells/SchematicProject/ProjectException.java
199  com/ipshells/SchematicProject/ProjectID.java
200  com/ipshells/SchematicProject/ProjectID_XML.java
201  com/ipshells/SchematicProject/ProjectManager.java
202  com/ipshells/SchematicProject/ProjectPanel.java
203  com/ipshells/SchematicProject/ProjectReadWrite.java
204  com/ipshells/SchematicProject/ProjectXML.java
205  com/ipshells/SchematicProject/ServerModule.java
206  com/ipshells/SchematicProject/ServerModuleXML.java
207  com/ipshells/SchematicProject/ServerProject.java
208  com/ipshells/SchematicProject/ServerProjectXML.java
209  com/ipshells/SchematicProject/SymbolPropertiesTabbedPane.java
210  com/ipshells/SchematicProject/TestBench.java
211  com/ipshells/SchematicProject/TestBenchPanel.java
212  com/ipshells/SchematicProject/VerificationPanel.java
213  com/ipshells/SchematicProject/VerifyRTLPanel.java
214  com/ipshells/ServerConnection.java
215  com/ipshells/ShellClient/ComponentSelection/AbstractSymbolController.java
216  com/ipshells/ShellClient/ComponentSelection/BusPopupMenu.java
217  com/ipshells/ShellClient/ComponentSelection/BusSymbolController.java
218  com/ipshells/ShellClient/ComponentSelection/CategoryComponentsMenu.java
219  com/ipshells/ShellClient/ComponentSelection/CategoryGroupPopupMenu.java
220  com/ipshells/ShellClient/ComponentSelection/CircuitIOPopupMenu.java
221  com/ipshells/ShellClient/ComponentSelection/CircuitObjectSelector.java
222  com/ipshells/ShellClient/ComponentSelection/CircuitObjectSelectorToolBar.java
```

```
223 com/ipshells/ShellClient/ComponentSelection/ComponentSymbolController.java
224 com/ipshells/ShellClient/ComponentSelection/DefaultSymbolController.java
225 com/ipshells/ShellClient/ComponentSelection/SymbolControllerManager.java
226 com/ipshells/ShellClient/DragAndDropHandler.java
227 com/ipshells/ShellClient/EditMenu.java
228 com/ipshells/ShellClient/EnvironmentActions.java
229 com/ipshells/ShellClient/FileMenu.java
230 com/ipshells/ShellClient/HelpMenu.java
231 com/ipshells/ShellClient/MainMenuBar.java
232 com/ipshells/ShellClient/MainToolbar.java
233 com/ipshells/ShellClient/ProjectMenu.java
234 com/ipshells/ShellClient/SessionManager.java
235 com/ipshells/ShellClient/ShellClientAbout.java
236 com/ipshells/ShellClient/ShellClientFrame.java
237 com/ipshells/ShellClient/ShellClientMain.java
238 com/ipshells/ShellClient/ViewMenu.java
239 com/ipshells/ShellClient/WindowMenu.java
240 com/ipshells/XML/Schematic.java
241 com/ipshells/XML/SchematicReader.java
242 com/ipshells/XML/SchematicScript.java
243 com/ipshells/XML/WriteSchematic.java
244 com/ipshells/ejb/AbstractIPShellsEJB.java
245 com/ipshells/ejb/Client/Client.java
246 com/ipshells/ejb/Client/ClientBean.java
247 com/ipshells/ejb/Client/ClientHome.java
248 com/ipshells/ejb/Client/ClientLibrary.java
249 com/ipshells/ejb/Client/ClientLibraryBean.java
250 com/ipshells/ejb/Client/ClientLibraryHome.java
251 com/ipshells/ejb/Client/ClientLibraryRemote.java
252 com/ipshells/ejb/Client/ClientLibraryRemoteHome.java
253 com/ipshells/ejb/Client/ClientRemote.java
254 com/ipshells/ejb/Client/ClientRemoteHome.java
255 com/ipshells/ejb/Client/ClientShellSession.java
256 com/ipshells/ejb/Client/ClientShellSessionBean.java
257 com/ipshells/ejb/Client/ClientShellSessionHome.java
258 com/ipshells/ejb/Client/SchematicProject.java
259 com/ipshells/ejb/Client/SchematicProjectBean.java
260 com/ipshells/ejb/Client/SchematicProjectHome.java
261 com/ipshells/ejb/Client/SchematicProjectRemote.java
262 com/ipshells/ejb/Client/SchematicProjectRemoteHome.java
263 com/ipshells/ejb/Client/SchematicProjectSession.java
264 com/ipshells/ejb/Client/SchematicProjectSessionBean.java
265 com/ipshells/ejb/Client/SchematicProjectSessionHome.java
266 com/ipshells/ejb/DBStatistics.java
267 com/ipshells/ejb/Library/HDL/Component.java
268 com/ipshells/ejb/Library/HDL/ComponentBean.java
269 com/ipshells/ejb/Library/HDL/ComponentHome.java
270 com/ipshells/ejb/Library/HDL/ComponentRemote.java
271 com/ipshells/ejb/Library/HDL/ComponentRemoteHome.java
272 com/ipshells/ejb/Library/HDL/Verilog/Generators/HDLSourceGenerator.java
273 com/ipshells/ejb/Library/HDL/Verilog/Generators/HDLSourceGeneratorBean.java
274 com/ipshells/ejb/Library/HDL/Verilog/Generators/HDLSourceGeneratorHome.java
275 com/ipshells/ejb/Library/HDL/Verilog/Generators/HDLSourceGeneratorLocal.java
276 com/ipshells/ejb/Library/HDL/Verilog/Generators/HDLSourceGeneratorLocalHome.java
277 com/ipshells/ejb/Library/HDL/Verilog/Generators/HDLSourceGeneratorTestClient1.java
278 com/ipshells/ejb/Library/Library.java
279 com/ipshells/ejb/Library/LibraryBean.java
280 com/ipshells/ejb/Library/LibraryCategoryPattern.java
281 com/ipshells/ejb/Library/LibraryCategoryPatternBean.java
282 com/ipshells/ejb/Library/LibraryCategoryPatternHome.java
283 com/ipshells/ejb/Library/LibraryCategoryPatternRemote.java
284 com/ipshells/ejb/Library/LibraryCategoryPatternRemoteHome.java
285 com/ipshells/ejb/Library/LibraryHome.java
286 com/ipshells/ejb/Library/LibraryRemote.java
287 com/ipshells/ejb/Library/LibraryRemoteHome.java
288 com/ipshells/ejb/Library/LibrarySession.java
289 com/ipshells/ejb/Library/LibrarySessionBean.java
290 com/ipshells/ejb/Library/LibrarySessionHome.java
291 com/ipshells/ejb/io/Process.java
292 com/ipshells/ejb/io/ProcessBean.java
293 com/ipshells/ejb/io/ProcessHome.java
294 com/ipshells/hdl/AbstractHDLSyntax.java
295 com/ipshells/hdl/HDLException.java
296 com/ipshells/hdl/HDLObjectNameTextField.java
297 com/ipshells/hdl/HDLParsingException.java
298 com/ipshells/hdl/HDLSourceReader.java
299 com/ipshells/hdl/verilog/VerilogSyntax.java
```

```
300 com/ipshells/hdl/vhdl/VHDLSyntax.java
301 com/ipshells/library/AbstractLibraryModelCache.java
302 com/ipshells/library/LibraryComponents.java
303 com/ipshells/library/LibraryConfigDialog.java
304 com/ipshells/library/LibraryException.java
305 com/ipshells/library/LibraryManager.java
306 com/ipshells/library/LibraryManagerButtonPanel.java
307 com/ipshells/library/LibraryManagerMain.java
308 com/ipshells/library/LibraryModel.java
309 com/ipshells/library/LibraryModelPanel.java
310 com/ipshells/library/LibraryModelPersistence.java
311 com/ipshells/library/LibraryModelXML.java
312 com/ipshells/library/LibraryTable.java
313 com/ipshells/library/categories/AbstractLibraryCategoryPatterns.java
314 com/ipshells/library/categories/CategoryException.java
315 com/ipshells/library/categories/CategoryManager.java
316 com/ipshells/library/categories/CategoryManagerMain.java
317 com/ipshells/library/categories/CategoryModel.java
318 com/ipshells/library/categories/CategoryModelDB.java
319 com/ipshells/library/categories/CategoryModelEJB.java
320 com/ipshells/library/categories/CategoryModelPanel.java
321 com/ipshells/library/categories/CategoryModelPersistence.java
322 com/ipshells/library/categories/CategoryModelXML.java
323 com/ipshells/library/categories/CategoryPatternException.java
324 com/ipshells/library/categories/CategoryPatternTreePanel.java
325 com/ipshells/library/categories/CategoryPatternsModel.java
326 com/ipshells/library/categories/CategoryPortModel.java
327 com/ipshells/library/categories/CategoryPortModelPanel.java
328 com/ipshells/library/categories/CategoryPortPatternsModel.java
329 com/ipshells/library/categories/CategoryRemote.java
330 com/ipshells/library/categories/CategoryRemoteHome.java
331 com/ipshells/library/categories/CategoryTreePanel.java
332 com/ipshells/library/categories/LibraryCategoryPatterns.java
333 com/ipshells/library/categories/LibraryCategoryPatternsXML.java
334 com/ipshells/library/components/Component.java
335 com/ipshells/library/components/ComponentException.java
336 com/ipshells/library/components/ComponentID.java
337 com/ipshells/library/components/ComponentInterfaceModel.java
338 com/ipshells/library/components/Parameter.java
339 com/ipshells/library/components/Port.java
340 com/ipshells/swing/NumberTextField.java
```

Appendix B
MySQL database creation script

```
DROP DATABASE IF EXISTS IP_Shells;
CREATE DATABASE IP_Shells;
USE IP_Shells;

DROP TABLE IF EXISTS Libraries;
CREATE TABLE IF NOT EXISTS Libraries(
    Library_ID          CHAR(50)  BINARY NOT NULL PRIMARY KEY,
    Library_Name        CHAR(50)  BINARY NOT NULL UNIQUE,
    Available           BIT       DEFAULT 0 NOT NULL,
    HDL                 CHAR(20)  DEFAULT "verilog",
    Library_Path        CHAR(255) BINARY,
    Source_Path         CHAR(255),
    Source_File         CHAR(255),
    Sub_Modules_File    CHAR(255),
    Stimulus_Path       CHAR(255),
    Stimulus_File       CHAR(255),
    Test_Bench_Path     CHAR(255),
    Test_Bench_File     CHAR(255),
    Doc_Path            CHAR(255),
    Doc_File            CHAR(255),
    Description         TEXT)
    TYPE = InnoDB;


DROP TABLE IF EXISTS Categories;
CREATE TABLE IF NOT EXISTS Categories(
    Category_ID                 INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    Category_Name               CHAR(50) UNIQUE NOT NULL)
    TYPE = InnoDB;


DROP TABLE IF EXISTS Category_Patterns;
CREATE TABLE IF NOT EXISTS Category_Patterns(
    Category_Pattern_ID         INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    Category_Pattern_Regexp     CHAR(250) NOT NULL,
    Library_ID                  CHAR(50) BINARY NOT NULL,
    INDEX library_index (Library_ID),
    FOREIGN KEY (Library_ID) REFERENCES Libraries(Library_ID)
        ON DELETE CASCADE,
    Category_ID                 INT NOT NULL,
    INDEX category_index (Category_ID),
    FOREIGN KEY (Category_ID) REFERENCES Categories(Category_ID)
        ON DELETE CASCADE)
    TYPE = InnoDB;


DROP TABLE IF EXISTS Category_Ports;
CREATE TABLE IF NOT EXISTS Category_Ports(
    Category_Port_ID            INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    Port_Type                   CHAR(25) NOT NULL,
    Port_Direction              ENUM('input', 'output', 'inout') DEFAULT NULL,
    Collection                  BIT DEFAULT 0 NOT NULL,
    Optional                    BIT DEFAULT 0 NOT NULL,
    Inverted                    BIT DEFAULT 0 NOT NULL,
    Category_ID                 INT NOT NULL,
    INDEX category_index (Category_ID),
    FOREIGN KEY (Category_ID) REFERENCES Categories(Category_ID)
        ON DELETE CASCADE)
    TYPE = InnoDB;


DROP TABLE IF EXISTS Category_Port_Patterns;
CREATE TABLE IF NOT EXISTS Category_Port_Patterns(
    Category_Port_Pattern_ID    INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    Port_Pattern_Regexp         CHAR(250) NOT NULL,
    Category_Port_ID            INT NOT NULL,
    INDEX category_port_index (Category_Port_ID),
    FOREIGN KEY (Category_Port_ID) REFERENCES Category_Ports(Category_Port_ID)
        ON DELETE CASCADE,
    Library_ID                  CHAR(50) BINARY NOT NULL,
    INDEX library_index (Library_ID),
    FOREIGN KEY (Library_ID) REFERENCES Libraries(Library_ID)
        ON DELETE CASCADE)
    TYPE = InnoDB;


DROP TABLE IF EXISTS Exclusions;
CREATE TABLE IF NOT EXISTS Exclusions(
    Exclusion_ID                INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    Exclusion_Pattern_Regexp    CHAR(100) NOT NULL,
    Library_ID                  CHAR(50) BINARY NOT NULL,
    INDEX library_index (Library_ID),
    FOREIGN KEY (Library_ID) REFERENCES Libraries(Library_ID)
        ON DELETE CASCADE)
    TYPE = InnoDB;

/*                                                              */
/* Create the "component related" tables for this database...   */
/*                                                              */
DROP TABLE IF EXISTS Components;
CREATE TABLE IF NOT EXISTS Components(
        Component_ID        INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
        Component_Name      CHAR(50) BINARY NOT NULL,
        Category_ID         INT NOT NULL,
        Available           BIT DEFAULT 0 NOT NULL,
        Last_Modified       TIMESTAMP(14) NOT NULL,
        Doc_ID              INT,
        Doc_Last_Modified   TIMESTAMP(14),
        Library_ID          CHAR(50) BINARY NOT NULL)
        TYPE = InnoDB;

DROP TABLE IF EXISTS Ports;
CREATE TABLE IF NOT EXISTS Ports(
        Port_ID             INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
        Port_Name           CHAR(30) BINARY NOT NULL,
        Category_Port_ID    INT,
        Argument_Index      INT NOT NULL,
        Direction           ENUM('input', 'output', 'inout') NOT NULL,
        Width_Expression    CHAR(200) NOT NULL,
        Inverted            BIT DEFAULT 0 NOT NULL,
        Doc_ID              INT,
        Component_ID        INT NOT NULL,
        INDEX component_index (Component_ID),
        FOREIGN KEY (Component_ID) REFERENCES Components(Component_ID)
               ON DELETE CASCADE)
        TYPE = InnoDB;

DROP TABLE IF EXISTS Parameters;
CREATE TABLE IF NOT EXISTS Parameters(
        Parameter_ID            INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
        Parameter_Name          CHAR(20) BINARY NOT NULL,
        Value_Type              ENUM('INTEGER', 'REAL') NOT NULL,
        Signed                  BIT NOT NULL DEFAULT 0,
        Value_Size              INT NOT NULL DEFAULT 32,
        Default_Value           CHAR(30) NOT NULL,
        Base                    INT NOT NULL DEFAULT 10,
        Parameter_Index         INT NOT NULL,
        Doc_ID                  INT,
        Component_ID            INT NOT NULL,
        INDEX component_index (Component_ID),
        FOREIGN KEY (Component_ID) REFERENCES Components(Component_ID)
               ON DELETE CASCADE)
        TYPE = InnoDB;


/*
 * Add all the relevant indexes to the appropriate columns...
 */
ALTER TABLE Components      ADD INDEX  name_index               (Component_Name);
ALTER TABLE Components      ADD INDEX  category_index           (Category_ID);
ALTER TABLE Ports           ADD INDEX  port_component_index     (Component_ID);
ALTER TABLE Parameters      ADD INDEX  parameter_component_index(Component_ID);
```

```
/*
 *  Create client information tables…
 */
DROP TABLE IF EXISTS Company;
CREATE TABLE IF NOT EXISTS Company(
    Company_ID          INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    Company_Name        CHAR(50) NOT NULL,
    Company_Password    CHAR(16) BINARY NOT NULL,
    Department          CHAR(100),
    Street              CHAR(100),
    City                CHAR(20),
    State_County        CHAR(20),
    Country             CHAR(20),
    Zip_Post_Code       CHAR(20),
    Company_Email       CHAR(30),
    Company_Public_Key  BLOB,
    Company_Phone       CHAR(25),
    Company_Fax         CHAR(25),
    Company_Web         CHAR(50))
    TYPE = InnoDB;


DROP TABLE IF EXISTS Client;
CREATE TABLE IF NOT EXISTS Client(
    User_Name           CHAR(16)  BINARY NOT NULL PRIMARY KEY,
    Password            CHAR(16)  NOT NULL DEFAULT "",
    File_Space          CHAR(255) NOT NULL DEFAULT "",
    First_Name          CHAR(20)  NOT NULL,
    Last_Name           CHAR(20)  NOT NULL,
    Job_Title           CHAR(50),
    Email               CHAR(50),
    Public_Key          BLOB,
    Company_ID          INT NOT NULL,
    INDEX Company_Index (Company_ID),
    FOREIGN KEY (Company_ID) REFERENCES Company(Company_ID)
        ON DELETE CASCADE)
    TYPE = InnoDB;


DROP TABLE IF EXISTS Client_Library;
CREATE TABLE IF NOT EXISTS Client_Library(
    Client_Library_ID   INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    User_Name           CHAR(16) BINARY NOT NULL,
    INDEX USER_NAME_INDEX (User_Name),
    FOREIGN KEY (User_Name) REFERENCES Client(User_Name)
        ON DELETE CASCADE,
    Library_ID          CHAR(50) BINARY NOT NULL,
    INDEX Library_Index (Library_ID),
    FOREIGN KEY (Library_ID) REFERENCES Libraries(Library_ID)
        ON DELETE CASCADE)
    TYPE = InnoDB;
```

Appendix C
Example synthesis script

Shown below is an example synthesis script for Cadence's BuildGates. This script, written in the TCL language, is automatically generated and executed by the application server when a client schematic is to be synthesized.

```
set TOP_MODULE      sg_dct_idct_po_4x4
set RTL_FILE        sg_dct_idct_po_4x4.v
set SRC_DIR         .
set LIB_DIR
       /cadence/libraries/TSMC/TSMCHOME/digital/Front_End/timing_power/tcb013
lphp_211a
set RTL_DIR         $SRC_DIR
set RPT_DIR         $SRC_DIR
set SYN_DIR         $SRC_DIR

set_global message_verbosity_level             8
set_global echo_commands                       true
set_global report_precision                    5
set_global fix_multiport_nets                  true
set_global sdc_write_unambiguous_names         false
set_global line_length                         1000


read_tlf ${LIB_DIR}/tcb013lphptc.tlf

set_global target_technology  tcb013lphptc

read_verilog  ${RTL_DIR}/${RTL_FILE}

do_build_generic -all

set_top_timing_module  ${TOP_MODULE}

# Setting up Hierarchical and Timing Context"
issue_message -type info "--> Setting up Hierarchical & Timing Context ..."

# Setting Ideal Clocks"
issue_message -type info "--> Setting Ideal Clocks ..."
set_clock ideal_clock -period 50 -waveform {0 25}

set_clock_arrival_time -rise 0 -fall 25 -clock ideal_clock clk
set_data_arrival_time 0 -clock ideal_clock [find -port -input *]
set_data_required_time 40 -clock ideal_clock [find -port -output *]


#Setting Design Rules
```

```
issue_message -type info "--> Setting Desgin Rules ..."
set_slew_time_limit 2.3 [ find -ports -noclocks * ]
set_global fanout_load_limit 15

#Setting wire load models
issue_message -type info "--> Setting wire load..."

set_operating_conditions      NCCOM
set_wire_load                 TSMC16K_Fsg_Conservative
set_wire_load_mode            enclosed

do_optimize

# write netlist
write_verilog -hier ${TOP_MODULE}.net.v

# binary netlist
write_adb ${TOP_MODULE}.adb

# timing file
write_sdf ${TOP_MODULE}.bg.sdf

# has cct achieved timing constraints
report_timing > ${TOP_MODULE}.report.time
report_area -summary -hierarchical -cells > ${TOP_MODULE}.report.area

quit
```

Appendix D
3rd party software licenses

The following text is the license file for projects covered by the Apache Software license, which includes the packages:

• **Jakarta ORO**	– the regular expression package
• **Xerces**	– the XML parser.
• **DBCP**	– the database connection pool

The following text it’s the license file for **SAX**, the XML API used in this project.

The following text is the license file for the **JDOM** XML parser used in this project.

The following text is the GNU license which covers usage of the **MySQL** database server and its **JDBC** driver.

  Also, for each distributor's protection, we want to make certain
that everyone understands that there is no warranty for this free
library. If the library is modified by someone else and passed on, we
want its recipients to know that what they have is not the original
version, so that any problems introduced by others will not reflect on
the original authors' reputations.

  Finally, any free program is threatened constantly by software
patents. We wish to avoid the danger that companies distributing free
software will individually obtain patent licenses, thus in effect
transforming the program into proprietary software. To prevent this,
we have made it clear that any patent must be licensed for everyone's
free use or not licensed at all.

  Most GNU software, including some libraries, is covered by the ordinary
GNU General Public License, which was designed for utility programs. This
license, the GNU Library General Public License, applies to certain
designated libraries. This license is quite different from the ordinary
one; be sure to read it in full, and don't assume that anything in it is
the same as in the ordinary license.

  The reason we have a separate public license for some libraries is that
they blur the distinction we usually make between modifying or adding to a
program and simply using it. Linking a program with a library, without
changing the library, is in some sense simply using the library, and is
analogous to running a utility program or application program. However, in
a textual and legal sense, the linked executable is a combined work, a
derivative of the original library, and the ordinary General Public License
treats it as such.

  Because of this blurred distinction, using the ordinary General
Public License for libraries did not effectively promote software
sharing, because most developers did not use the libraries. We
concluded that weaker conditions might promote sharing better.

  However, unrestricted linking of non-free programs would deprive the
users of those programs of all benefit from the free status of the
libraries themselves. This Library General Public License is intended to
permit developers of non-free programs to use free libraries, while
preserving your freedom as a user of such programs to change the free
libraries that are incorporated in them. (We have not seen how to achieve
this as regards changes in header files, but we have achieved it as regards
changes in the actual functions of the Library.) The hope is that this
will lead to faster development of free libraries.

  The precise terms and conditions for copying, distribution and
modification follow. Pay close attention to the difference between a
"work based on the library" and a "work that uses the library". The
former contains code derived from the library, while the latter only
works together with the library.

  Note that it is possible for a library to be covered by the ordinary
General Public License rather than by this special one.

engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise

```
permitted, and provided that you do these two things:

    a) Accompany the combined library with a copy of the same work
    based on the Library, uncombined with any other library
    facilities.  This must be distributed under the terms of the
    Sections above.

    b) Give prominent notice with the combined library of the fact
    that part of it is a work based on the Library, and explaining
    where to find the accompanying uncombined form of the same work.

  8. You may not copy, modify, sublicense, link with, or distribute
the Library except as expressly provided under this License.  Any
attempt otherwise to copy, modify, sublicense, link with, or
distribute the Library is void, and will automatically terminate your
rights under this License.  However, parties who have received copies,
or rights, from you under this License will not have their licenses
terminated so long as such parties remain in full compliance.

  9. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Library or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Library (or any work based on the
Library), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Library or works based on it.

  10. Each time you redistribute the Library (or any work based on the
Library), the recipient automatically receives a license from the
original licensor to copy, distribute, link with or modify the Library
subject to these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  11. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Library at all.  For example, if a patent
license would not permit royalty-free redistribution of the Library by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any
particular circumstance, the balance of the section is intended to apply,
and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any
```

The following text is the license file for the Borland Enterprise Server (BES) used in this project.

BORLAND SOFTWARE CORPORATION
LICENSE TERMS

BORLAND ENTERPRISE SERVER

NOTICE: THIS BORLAND SOFTWARE PRODUCT (TOGETHER WITH ITS ACCOMPANYING DOCUMENTATION, THE "PRODUCT") IS THE PROPERTY OF BORLAND SOFTWARE CORPORATION ("BORLAND"). THE PRODUCT IS MADE AVAILABLE TO YOU, THE ORIGINAL PURCHASER, SUBJECT TO THE FOLLOWING LICENSE AGREEMENT ("LICENSE"). PLEASE READ THIS LICENSE CAREFULLY BEFORE INSTALLING OR USING THE PRODUCT. A COPY OF THIS LICENSE IS AVAILABLE FOR YOUR FUTURE REFERENCE IN THE "LICENSE.TXT" FILE PROVIDED WITH THE PRODUCT.

YOU MAY ACCEPT THIS LICENSE BY PLACING A CHECK IN THE "I ACCEPT THE TERMS OF THE LICENSE AGREEMENT" BOX BELOW. YOU MAY REJECT THIS LICENSE, AND TERMINATE THIS INSTALLATION PROCESS, BY PLACING A CHECK IN THE "I DO NOT ACCEPT THE TERMS OF THE LICENSE AGREEMENT" BOX BELOW. IF YOU DO NOT ACCEPT THIS LICENSE, THEN YOU MAY NOT INSTALL OR USE THE PRODUCT. IN THAT CASE, YOU MAY, WITHIN TEN (10) DAYS AFTER YOU FIRST RECEIVED THE PRODUCT, RETURN IT TO BORLAND OR YOUR BORLAND AUTHORIZED RESELLER, ALONG WITH ITS ORIGINAL PACKAGING AND PROOF-OF-PURCHASE, FOR A FULL REFUND. ANY USE BY YOU OF THIS PRODUCT ALSO CONSTITUTES YOUR ACCEPTANCE OF THESE TERMS.

Borland is only willing to grant you this License if you obtained the Product from Borland or a Borland authorized reseller. If you obtained the Product from any other source you may not install or use the Product.

1. OWNERSHIP. The Product is proprietary to Borland. The Product is licensed, not sold, to you notwithstanding any reference herein to "purchases." You acknowledge and agree that: (a) the Product is protected under U.S. copyright and other laws; (b) Borland and its licensors retain all copyrights and other intellectual property rights in the Product; (c) there are no implied licenses under this License, and any rights not expressly granted to you hereunder are reserved by Borland; (d) you acquire no ownership or other interest (other than your license rights) in or to the Product; and (e) Borland owns all copies of the Product, however made. You agree that you will not, at any time, contest anywhere in the world Borland's ownership of the Product, nor will you challenge the validity of Borland's rights in the Product. You have no rights hereunder to use any trademark or service mark belonging to Borland.

2. GRANT OF LICENSE AND SCOPE OF USE.

2.1 Licenses and License Keys. For each license which you acquire to use the Product, the edition of the Product (i.e., Web Edition, Visibroker Edition or AppServer Edition) which you are granted a license to use and the type of license (i.e., development or deployment) which you are granted are set forth on a license key certificate (a "License Key Certificate") which is provided to you either with the Product media or separately by Borland.

In order to activate and use the Product pursuant to each license which you acquire, you must enter the appropriate license key ("License Key") when prompted by the Product. For each license to use the Product which you acquire, the License Key will be included on the corresponding License Key Certificate.

For more information on which Licenses, License Key Certificates and/or License Keys, if any, should be included with the Product media you should contact Borland or your Borland authorized reseller.

2.2   Development License. The terms of this Section 2.2 are applicable to you only if you have purchased a Development License for the Product directly from Borland or a Borland authorized reseller.

Subject to the terms and conditions of this Agreement, Borland grants to you a personal, nonexclusive, nontransferable and limited license to install and execute the Product on one (1) computer for use by you or, if you are an entity, one (1) person in your organization ("Named User") solely to develop, compile (including byte code compile), and test, in source or object code form, your own application programs and other works ("Works") designed to work with the Product ("Development License"). You may also make one (1) backup copy of the Product solely to replace the original copy provided to you if the original copy is damaged or destroyed, provided that you may not deploy any such backup copy on a backup server without first obtaining the requisite licenses from Borland. Unless you have purchased a Deployment License from Borland (defined below) or a Borland authorized reseller, you may not deploy or use the Product on a production basis.

The Product contains certain time-out features designed to prevent the Product from being deployed in a runtime environment for longer than a specified period without Deployment License keys. Deployment License keys will be provided to you at such time as you purchase the requisite Deployment Licenses.

WORKS THAT YOU CREATE USING THE PRODUCT MAY REQUIRE THE PRODUCT IN ORDER TO RUN. WITHOUT THE REQUISITE DEPLOYMENT LICENSE KEYS, THE PRODUCT WILL BE DISABLED AND THOSE WORKS MAY NO LONGER RUN. YOU SHOULD THEREFORE TAKE PRECAUTIONS TO AVOID ANY LOSS OF DATA THAT MIGHT RESULT.

You may purchase from Borland the right to make multiple copies of the Product (each, a "Licensed Copy") for development use. Such rights will be effective only when granted in writing by Borland and are conditioned upon your payment of the applicable fees. If you purchase such additional rights, you are granted, for each Licensed Copy a single Development License as set forth in Section 2.1, subject to all the terms and conditions of this Agreement. You agree that you will not install or execute the Product on more computers than the number of Licensed Copies that Borland has expressly permitted you to make and you will not allow more than one (1) Named User to use each Licensed Copy.

All rights not specifically granted to you herein are retained by Borland.

2.3. Deployment License. The terms of this Section 2.3 are applicable to you only if you have purchased a Deployment License for the Product directly from Borland.

Deployment Licenses are granted subject to certain limitations and usage parameters specific to the type of Deployment License you have acquired from Borland. Your invoice will show the number of Deployment Licenses, by license type, for the Product you have acquired from Borland and will list the usage parameters for each Deployment License. Borland offers the types of Deployment Licenses for the Product listed below:

(a) CONCURRENT USER: For each Concurrent User Deployment License you purchase, if any, Borland grants you the personal, nonexclusive, nontransferable and limited license to install and execute the Product in object code form on a production basis on one (1) computer and to permit the Product as so installed to be accessed at any one time by no more than number of servers or users (i.e. connections) indicated on your invoice ("Concurrent Users"). In exercising your rights under a Concurrent User Deployment License, you must provide reasonable security or administrative system to ensure that the number of Concurrent Users never exceeds the number indicated on your invoice.

(b) SERVER/CPU: For each Server/CPU Deployment License you purchase, if any, Borland grants you the personal, nonexclusive, nontransferable and limited license to install and execute the Product in object code form on a production basis on one (1) server hardware unit that is operating the platform designated on your invoice and that has no more than the number of CPUs designated on your invoice and to permit the Product as so installed to be accessed at any one time by an unlimited number of concurrent users. You may not install or execute the Product on any type of device that has a greater number of CPUs and you may not, under a given CPU Deployment License, install or execute the Product on more than one (1) computer.

The total number of Product copies used by you and the manner in which such copies are used may not exceed or deviate from the type and number of licenses acquired by you and the usage parameters indicated on your invoice (i.e, type of platform and number of CPUs, or number of Concurrent Users).

If you would like to install and/or deploy a greater number of copies of the Product than the number of Deployment Licenses you have acquired from Borland or if you would like to alter the type or usage parameters of any Deployment License you have acquired from Borland, you must first contact Borland to obtain written approval and pricing for any such modifications to your license.

All rights not specifically granted to you herein are retained by Borland.

2.4 Standby Server License. The terms of this Section 2.4 are applicable to you only if you have purchased a Standby Server License for the Product from Borland or a Borland authorized reseller.

Borland grants to you a personal, nonexclusive, nontransferable and limited license to install the Product on a single standby server and to use the Product in object code form solely in a standby, non-production mode for use in the event the primary server fails ("Standby Server License"). You may not use the Product on a production basis under a Standby Server License except during periods when one or more copies of the Product for which you have purchased a Deployment License are nonfunctional, and, in such event, only in accordance with the terms of the Deployment License.

2.5 Backup Server License. The terms of this Section 2.5 are applicable to you only if you have purchased a Backup Server License for the Product from Borland or a Borland authorized reseller.

Borland grants to you a personal, nonexclusive, nontransferable and limited license to install the Product on a single backup server and to use the Product in object code form solely for backup or disaster recovery purposes ("Backup Server License"). You may not use the Product on a production basis except during periods when one or more copies of the Product for which you have purchased a Deployment License are nonfunctional, and, in such event, only in accordance with the terms of the Deployment License.

2.6 Competitive Product Restrictions. You may not include the Product or any component thereof, including any Redistributables (as defined in Section 3) in any general-purpose software development tool, library, component, or any other product that is generally competitive with or a substitute for the Product or any other Borland product offerings; nor may you use the Product to create a product or operate a service that is generally competitive with the Product or any other Borland product offerings, including any general-purpose software development tool. The foregoing restriction does not apply, however, to your use of the Product to develop "plug-ins" (i.e., integrations created using the Product's open tools API) so long as such plug-ins are designed to provide supplementary functionality to the Product.

2.7 No Sublicensing, Resale or Distribution; Certain Restrictions. You are not permitted to reproduce the Product for sublicensing, resale, lending, leasing, deployment or distribution to any party, including without limitation, distributing the Product as part of a VAR, OEM, distributor or reseller arrangement. If you integrate the Product into a Work and intend to distribute or resell the resulting integrated Work, you must contact Borland to obtain the appropriate distribution license. All rights not specifically granted to you herein are retained by Borland.

3. GENERAL TERMS THAT APPLY TO COMPILED PROGRAMS AND REDISTRIBUTABLES.

3.1 Redistributables. The Product may include certain files, libraries and/or source code specifically designated as "redistributables" by Borland in the accompanying printed or on-line documentation and that are necessary to use Works created using the Product ("Redistributables"). From time to time, Borland may designate other files as Redistributables. You should refer to the documentation, including any "readme" or "deploy" files provided with the Product, for additional information regarding Redistributables. Subject to the terms and conditions of this License, you may freely redistribute source code or compiled code that is entirely your own and does not contain any Redistributables.

3.2 Licensing of Redistributables. Subject to the terms and conditions of this License including the restrictions of Section 3.3, Borland grants you the personal, non-exclusive, non-transferable and limited license to: (a) make exact copies of the Redistributables and distribute those copies solely as components of your Works and solely as required for permitting end users of the Works ("End Users") to install and execute the Works; (b) install and execute Redistributables, without modification, on computers that you own or possess solely for your own internal use; and (c) sublicense to your End Users the personal, non-exclusive, non-transferable right to install and execute Redistributables, without modification, solely as components of Works and solely for such End Users' own internal use, subject to End Users' compliance with the restrictions in Section 5 as to Redistributables. The rights granted to you under this Section 3.2 may not be exercised by others, including co-developers, regardless of how you might compile, link, or package your Works. These rights apply only to Redistributables and to no other file, library, source code or other component or derivative work of the Product. They may be exercised only with respect to Works created by you using a duly licensed, properly registered copy of the Product.

3.3 Certain Restrictions. Regardless of any modifications that you make and regardless of how you might compile, link, or package your Works: (a) you may not permit your End Users to modify or further distribute Redistributables or use Redistributables in any program that they create; (b) you may not use Borland's or any of its suppliers' names, logos, or trademarks to market your Works, except to state descriptively that your Work was written using the Product; (c) you may not incorporate or distribute a Redistributable with a Work or other program that is capable of functioning as a general purpose development tool, library, or component, or is otherwise generally competitive with or a substitute for the Product or the Redistributable itself; (d) all copies of the Works you create must bear a valid copyright notice, either your own or the Borland copyright notice that appears on the Product, and you may not remove or alter any Borland copyright, trademark or other proprietary rights notice contained in any portion of the Redistributables; and (e) you may only distribute Redistributables with Works that add primary and substantial functionality to the Redistributables and are not merely a set or subset of any of the Redistributables, and that are created in accordance with the terms of this License.

3.4 Relationship with End Users. Except as set forth in Section 4, there are no third party beneficiaries to this Agreement. Consequently, Borland provides no warranty at all to any person, other than the limited warranty provided to you the original purchaser of the Product, as set forth herein, and you will be solely responsible to your End Users (or anyone else who uses or acquires Works) for support, service, upgrades, or technical or other assistance (including with respect to any Redistributables included therein), and such persons will have no right to contact Borland for any services or assistance. You will indemnify, defend and hold Borland, its licensors, its suppliers and each of their respective employees, officers, directors and affiliates, harmless from and against any claims or liabilities arising out of or related to the use, procurement, reproduction or distribution of your Works by third parties.

3.5 Third Party Software. The Product, including Redistributables, may include source code, redistributable files, and/or other files provided by a third party vendor ("Third Party Product"). Since use of Third Party

Product might be subject to license restrictions imposed by the third party vendor, you should refer to the on-line documentation (if any) provided with Third Party Product for any license restrictions imposed by the third party vendor. In any event, any license restrictions imposed by a third party vendor are in addition to, not in lieu of, the terms and conditions of this License.

3.6 Other Rights. Contact Borland for the applicable royalties due and other licensing terms for all other uses or distribution of the Redistributables.

4. SPECIAL TERMS.
The following terms and conditions ("Special Terms") are specific to certain editions, versions and components of the Product and are in addition to the provisions of Sections 2 and 3. If any provision of the Special Terms applicable to the Product conflicts with any other provision of this License, then the provision of the Special Terms will supercede and control.

4.1 SPECIAL TERMS THAT APPLY TO BORLAND ENTERPRISE SERVER APPSERVER EDITION, VISIBROKER EDITION AND WEB EDITION

ADDITIONAL LICENSE TERMS FOR API DECOMPILER
Notwithstanding the prohibition hereunder against decompiling the Product, Borland grants to you as the licensed user of the Product the limited right to use that portion of the Product identified as "API Decompiler" for inspecting the public application programming interface (API) of the JAVA BEANS COMPONENT LIBRARY, DATAEXPRESS, DBSWING and OPENTOOLS API modules.

ADDITIONAL LICENSE TERMS FOR JDATASTORE
The portion of the Product identified as JDATASTORE (if any) is not a Redistributable. Notwithstanding any provision herein to the contrary, you may only use JDATASTORE under this License to develop, compile and test Works and for no other purpose. You must purchase a separate JDATASTORE deployment license from Borland or an authorized reseller for each copy of any Work that you deploy if the Work incorporates, or requires use of, JDATASTORE or includes the file "jdslicense.jar." You are not permitted to reproduce JDATASTORE for sublicensing, resale, lending, leasing, deployment or distribution to any party, including without limitation, distributing JDATASTORE as part of a VAR, OEM, distributor or reseller arrangement. If you integrate JDATASTORE or the file "jdslicense.jar" into a Work and intend to distribute or resell the resulting integrated Work, you must contact Borland to obtain the appropriate distribution license.

The following two types of JDATASTORE deployment licenses are available from Borland:

(a) "Local Server License." When you purchase a Local Server License with respect to a particular licensed Work, Borland grants you a personal, nonexclusive, nontransferable and limited license to install and execute JDATASTORE on one (1) computer ("Local Server") solely for your own internal data processing purposes to provide a limited number of local support connections. A "local support connection" is a communication session between JDATASTORE, as resident on the Local Server, and a copy of the licensed Work that is also executing on the Local Server and that is used solely for your internal data processing needs. For more information on the number of local support connections accessible under a Local Server License you should contact Borland directly.

(b) "Unlimited Server License." When you purchase an Unlimited Server License with respect to a particular Work, Borland grants you a personal, nonexclusive, nontransferable and limited license to install and execute JDATASTORE on one (1) computer ("Unlimited Server") solely for your own internal data processing purposes to provide an unlimited number of support connections. A "support connection" is a communication session between JDATASTORE, as resident on the Unlimited Server, and a copy of the licensed Work that is executing either on the Unlimited Server or on a remote machine. You may not, under an Unlimited Server License, install or execute JDATASTORE on more than one (1) computer. An Unlimited Server License for JDATASTORE is also granted with each Deployment License purchased for Borland Enterprise Server Web Edition.

4.2   SPECIAL TERMS THAT APPLY TO APPSERVER EDITION AND VISIBROKER EDITION ONLY

ADDITIONAL LICENSE TERMS FOR VISIBROKER
The portion of the Product identified as VISIBROKER is not a Redistributable and is licensed for development purposes only. Notwithstanding any provision herein to the contrary, you may only use VISIBROKER under this License for developing, compiling and testing Works and for no other purpose. You must purchase a separate VISIBROKER deployment license from Borland or an authorized reseller before deploying any Work that incorporates, or requires use of, VISIBROKER. You are not permitted to reproduce VISIBROKER for sublicensing, resale, lending, leasing, deployment or distribution to any party, including without limitation, distributing VISIBROKER as part of a VAR, OEM, distributor or reseller arrangement. If you integrate VISIBROKER into a Work and intend to distribute or resell the resulting integrated Work, you must contact Borland to obtain the appropriate distribution license.

ADDITIONAL LICENSE TERMS FOR BORLAND VISIBROKER EDITION SECURITY SERVICE
The portion of the Product identified as BORLAND VISIBROKER EDITION SECURITY SERVICE is not a Redistributable and is included on the Product media for your convenience only. This License does not permit you to execute, or otherwise use the BORLAND VISIBROKER EDITION SECURITY SERVICE portion of this Product for any purpose. You must purchase a separate Development and/or Deployment License for BORLAND VISIBROKER EDITION SECURITY SERVICE directly from Borland before developing or deploying any Work that uses such portion of the Product. You are not permitted to reproduce BORLAND VISIBROKER EDITION SECURITY SERVICE for sublicensing, resale, lending, leasing, deployment or distribution to any party, including without limitation, distributing BORLAND VISIBROKER EDITION SECURITY SERVICE as part of a VAR, OEM, distributor or reseller arrangement. If you integrate BORLAND VISIBROKER EDITION SECURITY SERVICE into a Work and intend to distribute or resell the resulting integrated Work, you must contact Borland to obtain the appropriate distribution license.

4.3   SPECIAL TERMS THAT APPLY TO APPSERVER EDITION ONLY

ADDITIONAL LICENSE TERMS FOR BORLAND APPSERVER

The portion of the Product identified as BORLAND APPSERVER is not a Redistributable and is licensed for development purposes only. Notwithstanding any provision herein to the contrary, you may only use BORLAND APPSERVER under this License for developing, compiling and testing Works and for no other purpose. You must purchase a separate BORLAND APPSERVER deployment license from Borland or an authorized reseller before deploying any Work that incorporates, or requires use of, BORLAND APPSERVER. You are not permitted to reproduce BORLAND APPSERVER for sublicensing, resale, lending, leasing, deployment or distribution to any party, including without limitation, distributing BORLAND APPSERVER as part of a VAR, OEM, distributor or reseller arrangement. If you integrate BORLAND APPSERVER into a Work and intend to distribute or resell the resulting integrated Work, you must contact Borland to obtain the appropriate distribution license.

4.4 ADDITIONAL LICENSE TERMS APPLICABLE TO EVALUATION SOFTWARE

If you have received an evaluation or trial version of the Product, you may exercise your rights under this license to use the Product and to create Works for the sole purpose of evaluating or demonstrating the Product. Your license is for a term of sixty (60) days from the date you obtain the serial number and license key for the Product or otherwise begin using the Product ("Evaluation Period"). Unless you have purchased a Deployment License from Borland (defined below) or a Borland authorized reseller, you may not deploy or use the Product on a production basis.

You may not use the Product for any commercial, business, governmental or institutional purpose of any kind. At the end of the Evaluation Period, further use of the Product by you is prohibited without the purchase of a commercial license. If you do not purchase a license for the Product at the end of the Evaluation Period, you hereby agree to permanently remove or delete the Product from all computer systems on which it was installed and destroy any software and documentation received, and not to reinstall a new copy of the Product. If you desire to continue to use the Product following the Evaluation Period, you should contact Borland or a Borland authorized reseller to order commercial licenses to use the Product.

THE PRODUCT CONTAINS A TIME-OUT FEATURE THAT DISABLES ITS OPERATION AFTER THE EXPIRATION OF THE EVALUATION PERIOD. WORKS THAT YOU CREATE DURING THE EVALUATION PERIOD MAY REQUIRE THE PRODUCT IN ORDER TO RUN. UPON EXPIRATION OF THE EVALUATION PERIOD, THOSE WORKS MAY NO LONGER RUN. YOU SHOULD THEREFORE TAKE PRECAUTIONS TO AVOID ANY LOSS OF DATA THAT MIGHT RESULT.

4.5 ADDITIONAL LICENSE TERMS APPLICABLE TO THIRD PARTY SOFTWARE

Certain components of the Product use or incorporate third-party software programs and/or libraries ("Third-party Software") which are loaded (in both object and source code form) on the Product media. You agree that Borland's third-party licensors and suppliers are intended third party beneficiaries of all terms and conditions of this License intended to protect intellectual property rights in the Product (including the Third-party Software) and limit certain uses thereof.

ADDITIONAL LICENSE TERMS FOR SONICMQ - JAVA TECHNOLOGY RESTRICTIONS

The following Java Technology Restrictions are applicable to the component of the Product identified as SonicMQ: You may not modify the Java Platform Interface ("JPI", identified as classes contained within the "java" package or any subpackages of the "java" package), by creating additional classes within the JPI or otherwise causing the addition to or modification of the classes in the JPI. In the event that you otherwise create an additional class and associated API(s) which (i) extends the functionality of a Java platform, and (ii) is exposed to third party software developers for the purpose of developing additional software which invokes such additional API, you must promptly publish broadly an accurate specification for such API for free use by all developers. You may not create, or authorize your licensees to create additional classes, interfaces, or subpackages that are in any way identified as "java", "javax", "sun" or similar convention as specified by Sun in any class file naming convention.

This product includes code licensed from RSA Data Security.

ADDITIONAL LICENSE TERMS FOR APACHE SOFTWARE
The Product or third party products or components included with or in the Product may include software developed by the Apache Software Foundation (http://www.apache.org/). Copyright (c) 1999-2000 The Apache Software Foundation. All rights reserved. The names "Xerces", "Tomcat", "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from the software developed by the Apache Software Foundation without prior written permission. Products derived from the software developed by the Apache Software Foundation may not be called "Apache", nor may "Apache" appear in their name, without prior written permission. For written permission, please contact apache@apache.org.

ADDITIONAL LICENSE TERMS FOR GNU COMPONENT(S)
No GNU public license or similar open source licenses contained within the Product or any third party products or components included with or in the Product shall be removed or modified and no proprietary rights notices or agreements accompanying the Product or any third party products or components included with or in the Product shall be deleted.

5.   LIMITATIONS. You may not: (a) modify, adapt, alter, translate, or create derivative works of the Product or merge the Product with other software other than as described in the Product's accompanying documentation or as approved of in writing by Borland; (b) lease, rent or loan the Product to any third party; (c) sublicense, distribute or otherwise transfer the Product or any component thereof to any third party except as expressly authorized under Section 3; (d) reverse engineer, decompile, disassemble, or otherwise attempt to derive the source code of the Product; (e) remove, alter, or obscure any confidentiality or proprietary notices (including copyright and trademark notices) of Borland or its suppliers on the Product; (f) allow third parties to access or use the Product such as in a time-sharing arrangement or operate the Product as part of a service bureau or, except as expressly authorized under Sections 2, 3 or 4, otherwise for the use or benefit of third parties; (g) reproduce or use the Product except as expressly authorized under Sections 2, 3 or 4; or (h) disclose or publish performance benchmark results for the Product. The rights granted under this License apply only to this Product. You must procure a separate license to use other Borland software. Furthermore, you may not permit your

End Users to conduct the restricted activities limited by items (a) through (e), (g) and (h) above insofar as they apply to Redistributables, and such End User's sublicense rights to the Redistributables are conditioned upon compliance with such limitations. The limitations in this Section 5 apply equally to your use of the Product, in whole or in part, including any component or Redistributable.

6. LIMITED WARRANTY AND DISCLAIMER. Borland warrants to you, the original purchaser and to no other party, that any physical media included with the Product, as and when provided to you, will be free of physical defects in materials and workmanship for a period of ninety (90) days after the date that you initially acquire the Product. Your exclusive remedy and Borland's sole liability for breach of this warranty is that Borland will replace any defective media returned to Borland within the ninety (90) day warranty period. This warranty does not apply to damages resulting from misuse, abuse or neglect. Any replacement media will be warranted as above for the remainder of the original warranty period or twenty (20) days from the date we ship it to you, whichever is longer. EXCEPT FOR THIS EXPRESS LIMITED WARRANTY, THE PRODUCT IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND. BORLAND HEREBY EXCLUDES AND DISCLAIMS ALL IMPLIED OR STATUTORY WARRANTIES, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, QUALITY, NON-INFRINGMENT, TITLE, RESULTS, EFFORTS OR QUIET ENJOYMENT. THERE IS NO WARRANTY THAT THE PRODUCT WILL BE ERROR-FREE OR WILL FUNCTION WITHOUT INTERRUPTION. YOU ASSUME THE ENTIRE RISK FOR THE RESULTS OBTAINED USING THE PRODUCT. TO THE EXTENT THAT BORLAND MAY NOT DISCLAIM ANY WARRANTY AS A MATTER OF APPLICABLE LAW, THE SCOPE AND DURATION OF SUCH WARRANTY WILL BE THE MINIMUM PERMITTED UNDER SUCH LAW.

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER LEGAL RIGHTS, WHICH VARY FROM STATE TO STATE.

7. SERVICES; UPDATES; PRODUCT CHANGES. Borland is not required under this License to provide any installation, training or other services to you. Such services, if available, must be purchased separately. If, pursuant to a separate support agreement or otherwise, Borland provides you with a new release, error correction, update, upgrade or other modification to the Product, such modification will be deemed part of the Product, and subject to the terms of this License, unless the modification is expressly provided subject to a separate license agreement. Borland reserves the right at any time not to release or to discontinue release of any Product and to alter prices, features, specifications, capabilities, functions, licensing terms, release dates, general availability or other characteristics of any future releases of the Product.

8. REGISTRATION. You must register the Product with Borland as a condition to your rights to use the Product. You will be prompted to register the Product at the time of your installation or first use of the Product, at which time you will be notified (or directed to online resources explaining) how registration information provided by you may be used and you will be afforded the opportunity to opt out of certain uses of such information.

9.   CONFIDENTIALITY. The Product (including its underlying source code) and the terms of this License contain confidential information of Borland. You agree to hold this information in confidence, not disclose it to any person (other than your employees and individual contractors who use the Product and who have agreed to keep confidential any of Borland's confidential information that you provide to them), and not use it for any purpose other than the use and operation of the Product as permitted under this License. These restrictions do not apply to any information which is or becomes (through no fault of yours) publicly available. If you are required by law or order of a court or other government authority to disclose Borland's confidential information, then you will immediately notify Borland as soon as possible, but in any event prior to the disclosure, and will cooperate with Borland, at its expense and request, in any lawful action to contest or limit the scope of such required disclosure.

10.   LIMITATION OF LIABILITY. IN NO EVENT WILL BORLAND BE LIABLE TO ANY PARTY FOR ANY INDIRECT, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY, SPECIAL OR PUNITIVE DAMAGES, INCLUDING ANY LOSS OF PROFIT, REVENUE, BUSINESS OPPORTUNITY OR DATA, ARISING FROM OR RELATING TO THIS LICENSE OR THE PRODUCT, WHETHER IN CONTRACT, IN TORT OR OTHERWISE, EVEN IF BORLAND KNEW, SHOULD HAVE KNOWN OR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BORLAND'S TOTAL CUMULATIVE LIABILITY ARISING FROM OR RELATED TO THIS LICENSE OR THE PRODUCT, WHETHER IN CONTRACT, IN TORT OR OTHERWISE, WILL NOT EXCEED THE FEES ACTUALLY PAID BY YOU UNDER THIS LICENSE. THIS SECTION 10 WILL APPLY EVEN IF AN EXCLUSIVE REMEDY HEREUNDER HAS FAILED OF ITS ESSENTIAL PURPOSE.

SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

11.   THIRD PARTY CLAIMS. Borland will defend and settle any suit brought against you by a third party (not your affiliate) based on a claim that the Product infringes upon any U.S. copyright and Borland will pay those costs and damages finally awarded against you in such suit that are specifically attributable to such claims or those amounts payable by you under a settlement of such suit. The foregoing obligations are conditioned on you: (a) notifying Borland promptly in writing of such action; (b) giving Borland sole control of the defense thereof and any related settlement negotiations; and (c) cooperating and, at Borland's request and expense, assisting in such defense. If the Product becomes, or in Borland's opinion is likely to become, the subject of an infringement claim that Borland is required to defend, then Borland may (at its option and expense) either: (a) procure for you the right to continue using the Product; (b) replace or modify the Product so that it becomes non-infringing; or (c) terminate this License and your rights hereunder to use the Product and refund a pro rata portion of any license fee you paid under this License, based on a three (3) year product life. Notwithstanding the foregoing, Borland will have no obligation under this Section 11 or otherwise with respect to any infringement claim based upon: (a) any use of the Product not in accordance with this License or the products accompanying documentation: (b) any use of the Product in combination with other products, equipment, software, or data not provided by Borland; (c) any use of any version of the Product other than the most current version made available to you; or (d) any modification of the Product by any person other than Borland or its authorized agents ("Excluded

Claims"). You will indemnify Borland against all liability, damages and costs (including reasonable attorneys' fees) resulting from or related to an Excluded Claim. This section 11 states Borland's entire liability and your sole and exclusive remedy for third-party claims relating to the Product.

12. AUDIT. During the term of this License and for one (1) year thereafter, upon reasonable notice and during normal business hours, Borland or its outside auditors will have the right to enter your premises and access your records and computer systems to verify that you have paid to Borland the correct amounts owed under this License and determine whether the Products are being used in accordance with the terms of this License. You will provide reasonable assistance to Borland in connection with this provision. You agree to pay the cost of the audit if any underpayments during the period covered by the audit amount to more than five percent (5%) of the fees actually owed for that period.

13. TERM AND TERMINATION.

13.1 Term. The term of this License will begin as of the date that you receive the Product and will remain in effect perpetually unless terminated under this Section 13.

13.2 Termination for Convenience. You may terminate this License for any reason, or for no reason, by giving Borland five (5) days' written notice.

13.3 Termination for Cause. Borland may terminate this License if you breach your obligations hereunder. Borland will effect such termination by giving you notice of termination, specifying therein the alleged breach. If your breach is curable, you will have a grace period of thirty (30) days after such notice is served to cure the breach described therein. If the breach is cured within the thirty (30) day grace period, then this License will remain in effect; otherwise, this License will automatically terminate upon the conclusion of the thirty (30) day grace period.

13.4 Effect of Termination. Upon the termination of this License for any reason the following terms shall apply: (a) all rights granted under this License will immediately terminate and you must stop all use of the Product and any Redistributables; (b) you must immediately pay to Borland all fees and expenses accrued prior to the effective date of termination; (c) you must return to Borland or destroy all copies of the Product provided to or made by you, and will, within ten (10) days after the effective date of termination, provide Borland with written certification that all such copies have been returned or destroyed; (d) you must return to Borland all of its other confidential information that you have received during the term of this License; and (e) all provisions of this Agreement with the exception of the licenses granted in Sections 2, 3 and 4, will survive termination of this License for any reason.

14. GENERAL PROVISIONS.

14.1 Canadian Transactions. If you are subject to Canadian law, you agree to the following:

The parties hereto have expressly required that the present License and its Exhibits be drawn up in the English language. / Les parties aux presentes ont expressement exige que la presente Convention et ses Annexes soient redigees en langue anglaise.

14.2 Hazardous Uses. The Product is not intended for use, and you may not use or allow others to use the Product, in connection with any application requiring fail-safe performance such as the operation of nuclear power facilities, air traffic control or navigation systems, weapons control systems, life support systems, or any other system whose failure could lead to injury, death, environmental damage or mass destruction. You agree that Borland will have no liability of any nature, and you are solely responsible, for any expense, loss, injury or damage incurred as a result of such use of the Product.

14.3 Governing Law; Venue and Jurisdiction. This License will be governed by and construed in accordance with the laws of the United States and the State of California, without giving effect to any conflicts or choice of laws principles that would require the application of the laws of a different jurisdiction. The parties expressly exclude the application of the 1980 United Nations Convention on the International Sale of Goods (if applicable). Subject only to the provisions of Section 14.6, any legal action, suit or proceeding arising out of or relating to this License must be instituted exclusively in a court of competent jurisdiction, federal or state, located within the State of California, and in no other venue. Each party further irrevocably consents to personal jurisdiction and venue in, and agrees to service of process issued or authorized by, any such court.

14.4 No Jury Trials; No Joinder. Each party hereby irrevocably waives its right to a jury trial in any legal action, suit or proceeding between the parties arising out of or relating to this License. A copy this License may be filed with the court as written consent by both parties to a bench trial. You agree that any dispute you may have against Borland cannot be joined with any dispute of any other person or entity in a lawsuit, arbitration or any other proceeding, or resolved on a classwide basis.

14.5 Severability. If any provision of this Agreement is held to be illegal, invalid or unenforceable for any reason, then such provision will be enforced to the maximum extent permissible and the remainder of the provisions of this Agreement will remain in full force and effect.

14.6 Equitable Relief. The parties acknowledge and agree that it is impossible to measure in money the damages that will accrue to Borland by reason of your breach of this Agreement and that such a breach will cause irreparable harm to Borland. In addition to any other right or remedy available at law or in equity Borland will be entitled to specific performance or injunctive relief to enforce or prevent any breach of Sections 2, 3, 4, 5 or 9 without posting a bond or other security, and may apply to any court of competent jurisdiction for such relief notwithstanding the provisions of Section 14.3.

14.7 Attorneys' Fees. In any action, suit or proceeding arising out of or relating to this License, Borland, if it is the prevailing party, will be entitled to recover from you its reasonable attorneys' fees and expenses in addition to any other relief that may be awarded.

14.8 Notices. A notice that is required or permitted under this Agreement must be in writing and may be delivered by facsimile, electronic mail, hand delivery, courier or mail. Notice will be deemed effective upon the earlier of actual receipt by the intended recipient or five (5) days after deposit with the U.S. Postal Service as certified or registered mail (postage prepaid and return receipt requested) addressed to recipient at its address set forth on the first page above or as amended by notice pursuant to this section.

14.9 Assignment. You may not assign this License or assign any of your rights or delegate any of your obligations under this License, by operation of law or otherwise (including by merger, sale of assets or consolidation), without Borland's prior written consent, which may be granted, conditioned or withheld in Borland's sole discretion. Any attempted assignment you in violation of this Section 14.10 will be void and will constitute a material breach of this Agreement. Notwithstanding the foregoing, Borland may assign this License at any time in its sole discretion. Subject to the foregoing, this Agreement will be binding upon and will inure to the benefit of the parties and their respective successors and permitted assigns.

14.10 Export Control. You may not directly or indirectly transfer the Product, including its documentation, to any country if such transfer would be prohibited by applicable law, including the U.S. Export Administration Act and the regulations issued thereunder. You agree to the foregoing and you are representing and warranting that you are not located in, under the control of, or a national or resident of any such country. You will be solely responsible for identifying and complying with all laws of any jurisdiction outside of the United States regarding the import, export or use of Products and technical data supplied by Borland. You will obtain at your own expense all licenses, permits or approvals required by any government to use the Product.

14.11 U.S. Government Rights. The Product is a "commercial item" as that term is defined at 48 C.F.R. 2.101, consisting of "commercial computer software" and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212. Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4, all U.S. Government end users acquire the Product with only those rights set forth in this License.

14.12 Waiver and Modifications. All waivers must be in writing. Any waiver or failure to enforce a provision of this License on one occasion shall not be deemed a waiver of any other provision or such provision on any other occasion. This License may only be amended by a written document signed by both parties.

14.13 Entire License. This License constitutes the entire, final and exclusive agreement between you and Borland regarding the specific license transaction described herein. No prior agreements, understandings, statements, proposals or representations, written or oral, apply. No written or oral statement, advertisement or product description not expressly contained in this License can be used to alter or supplement its