5-2002

# On a Grid-Based Interface to a Special-Purpose Hardware Cluster

Jeanne Marie Lehrter
*University of Tennessee - Knoxville*

To the Graduate Council:

I am submitting herewith a thesis written by Jeanne Marie Lehrter entitled "On a Grid-Based Interface to a Special-Purpose Hardware Cluster." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael Langston, Major Professor

We have read this thesis and recommend its acceptance:

Stacy Prowell, Greg Peterson

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Jeanne Marie Lehrter entitled "On a Grid-Based Interface to a Special-Purpose Hardware Cluster". I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Michael Langston

Major Professor

We have read this thesis
and recommend its acceptance:

Stacy Prowell

Greg Peterson

Accepted for the Council:

Dr. Anne Mayhew

Vice Provost and Dean of Graduate Studies

(Original signatures are on file in the Graduate Student Services Office.)

# On a Grid-Based Interface to a Special-Purpose Hardware Cluster

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Jeanne Marie Lehrter

May 2002

## Acknowledgments

This thesis work is part of the SInRG project which is funded by the National Science Foundation.

I'd like to thank Dr. Michael Langston for being a wonderful teacher and advisor throughout my time at the University of Tennessee. And I also thank Dr. Langston, Dr. Stacy Prowell, and Dr. Greg Peterson for serving on my thesis committee.

Thanks to all the people at the Innovative Computing Laboratory who helped me to understand the NetSolve software and implement it for this project.

## Abstract

Grid computing is an important and useful tool that enables resource sharing within a computational community. In the grid computing model, each node of the grid is a computational resource that may be used by applications residing on other nodes of the grid or on machines outside of the grid. To make grid computing more accessible to all users, the Innovative Computing Lab on campus at the University of Tennessee has produced a software package called NetSolve. NetSolve is a client-server application that defines interfaces for accessing and utilizing the software and hardware resources of the nodes in the grid community. A computational grid is currently being built on the University of Tennessee, Knoxville campus and is named the Scalable Intracampus Research Grid, or SInRG. This grid will have nodes located in different departments on campus, and each node will offer a special software or hardware resource. The SInRG node in the Electrical and Computer Engineering Department will offer special-purpose hardware in the form of reconfigurable computing boards. In this project, several functions were selected to be added to the SInRG node in the ECE department that can benefit from hardware acceleration and that would be useful to users. Then an interface was developed between the NetSolve grid software and each of these selected functions. Through these interfaces, users may access the special-purpose hardware that resides in this node and run their applications on this hardware in order to benefit from the hardware acceleration available.

# Contents

# Chapter 1

# Introduction

## 1.1 Previous Work

As the size and complexity of mathemetical computations have grown, the need for a different computing model has emerged. In many cases, computations now need faster processors, specialized software and hardware libraries, and more resources in general to achieve acceptable performance levels. However, often these resources are not locally available to users due to budget constraints or lack of expertise. The grid computing model emerged as a way of making resources available to users not just on the local level, but rather on the global level. With grid computing, users without adequate local resources can find and access resources at remote sites and therefore achieve better performance.

The University of Tennessee in Knoxville is currently in the process of building a computational grid for research. This project is called the Scalable Intracampus

Research Grid or SInRG. A computational grid consists of different resources connected by a network and the software that enables resource sharing. This software within the context of a grid is called middleware. SInRG relies on the middleware NetSolve, but there are other middleware packages available. Some of these other packages are discussed briefly below.

The Condor system [Con] was developed at the University of Wisconsin, Madison and is a high-throughput computing system. Condor is implemented within a collection of machines networked together called a Condor pool. When a computer is idle, the CPU cycles on that machine are wasted. The Condor system works by monitoring the activity of the machines within the Condor pool. If a machine is idle, Condor can send a job to that machine for execution and use CPU cycles that would have otherwise been wasted. By efficiently monitoring the machines within the pool and using the available idle resources, Condor can achieve a high-throughput system. The Condor system is similar to NetSolve in its goals and implementation, and therefore an interface has been developed between Condor and NetSolve. Through this interface, a Condor pool is seen as a single resource available to NetSolve users. Users within the Condor pool have priority, while users from NetSolve have a lower priority [CD99].

Ninf [Nin], developed by the Electrotechnical Laboratory in Tsukuba, Japan, is another middleware package similar to NetSolve in its goals and implementation. However, Ninf and NetSolve implement different networked computing paradigms. Ninf uses the proxy computing paradigm in which the user's code and data are both sent to the

server which runs the given user code on the data and returns the result. NetSolve on the other hand, uses the remote computing paradigm. In this paradigm the code and software libraries reside on the server. The user's data is sent to the server where the appropriate code is run with the given user data, and the result is then returned to the client. As with Condor, an interface has been developed between Ninf and NetSolve, but due to the different computing paradigms discussed above, an extra step was needed to achieve a functional interface. A piece of software called a NetSolve-Ninf adapter was written so that now NetSolve clients can access Ninf resources and Ninf clients can access NetSolve resources. This adapter performs the necessary "protocol translation, interface translation, and data transfer" needed to achieve seamless resource sharing between the two systems [CD99].

Globus [FK97] and Legion [GW96] [GWF+94] are two other middleware packages available today. These packages while still similar to NetSolve, are less like it than the Condor and Ninf systems mentioned above. Globus and Legion are based on a networked virtual computer paradigm, while the systems mentioned above and NetSolve are based on a loosely-coupled client-server-agent (CSA) model [SIn]. This difference results in Legion and Globus needing to build their own directory and communication services which in turn requires more effort from users in order to implement their applications. Using the CSA paradigm allows the creation of a higher level interface, as in NetSolve, and therefore the group of possible users is much larger since less effort and expertise is needed to run their code.

## 1.2 Thesis Goals

The goals of this thesis correspond to the goals of the Scalable Intracampus Research Grid (SInRG) project, of which this thesis is a part. The goals of SInRG are to provide computational resources from within the grid to user applications through an efficient and easy-to-use interface. While this thesis maintains these SInRG goals, its specific goals pertain to the special-purpose hardware resources available through the grid [SIn].

The specific goals of this thesis project are to provide an efficient and easy-to-use NetSolve interface to the special-purpose hardware available in the Electrical and Computer Engineering (ECE) department's node of the grid. This special-purpose hardware offered by the ECE node is PCs with Field Programmable Gate Arrays (FPGAs) attached to them. FPGAs can offer great speedup to some an applications. However, programming an FPGA with an application can be time consuming and requires hardware expertise on the part of the programmer. This results in FPGAs only being used by those with this expertise, while those without cannot achieve the acceleration offered by the FPGAs. This explains why an easy-to-use interface is particularly important with respect to the ECE node and its hardware.

The SInRG middleware, NetSolve, allows the creation of this easy-to-use interface to the special-purpose hardware. The NetSolve interface allows a user to easily access these FPGAs without being concerned about or even aware of all the technical aspects of running code on the FPGAs. The user simply supplies the input and output variables to the NetSolve interface and all of the processing and implementation details are invisible

to the user. Freeing users from having to understand the details of programming the FPGAs allows many users to achieve the speedup that they offer.

# Chapter 2

# The Case for Special Purpose Hardware

## 2.1 FPGAs

In the past, two types of processors were available for computations. These two types are general purpose hardware and application specific hardware. General purpose hardware is capable of processing any application, while application specific hardware is programmed to perform one type of application only and is not optimized or may be unable to perform other types of applications. While application specific hardware is constrained in the number of applications it can perform efficiently, its advantage is that it can run its specific application much more quickly than general purpose hardware. Since general purpose hardware must be able to handle all types of applications, it is

not optimized to run any one type of application. As a result, applications that run on general purpose hardware run at less than optimal speed. Application specific hardware on the other hand, is optimized for its specific application and therefore can run the application more quickly than the general purpose hardware.

Application specific hardware is implemented in custom designed hardware circuits. These circuits are called Application Specific Integrated Circuits (ASICs). As stated above, ASICs are designed to achieve optimal performance for one type of application. General purpose computers often include ASIC cards to perform commonly used functions such as graphics calculations. And while these common functions are accelerated by the ASIC cards, the other less commonly used applications run on the general purpose processor and therefore run at less than optimal speed.

The goal would be to have all applications run on ASICs rather than general purpose hardware in order to achieve better performance overall, but there are several factors that prevent this approach. First, designing an ASIC is very time-consuming. It can take hardware designers months or longer to develop an optimized hardware design for an application. This long development time can be very expensive to companies and therefore to consumers. Second, the cost and time it takes to manufacture these ASICs can be high. Unless a large number of ASICs will be needed, the cost per ASIC will be too high to be practical. Third, once the ASIC is designed and manufactured, any changes to the ASIC will still be very expensive. If a bug is found after the ASICs have been manufactured, some portions of the design process will have to be repeated,

and again this may take a long time. After several years of use, an application may need to be upgraded. Even if the changes are minor, a new design must be created for the application and then new ASICs must be manufactured. Implementing even the smallest changes to a design can be very costly and time-consuming. As a result of these factors very few applications merit the expense of an application specific implementation in hardware.

The development of FPGAs changed all of that. FPGAs are Field Programmable Gate Arrays. FPGAs are reconfigurable hardware that can be reprogrammed to implement a series of applications. When an FPGA is produced it is not designed to perform any specific task. It's like a blank processor on which you can program any application. FPGAs offer an application specific hardware implementation without the high cost or long wait involved with developing an ASIC. So, you can program the FPGA to have optimal performance for one application, and then you can reprogram it to have optimal performance for a different application. The time it takes to reprogram an FPGA is on the order of microseconds. The reprogrammable feature of FPGAs removes the time and cost overhead involved with ASIC design and implementation. So now if you find bugs in your design after it has been programmed onto the FPGA, the bugs can cheaply be eliminated. Or if you want to upgrade your software, again the change is much less expensive than when using an ASIC.

With the development of FPGAs, the future view of computing changed. Rather than having a computer with perhaps a few ASICs for acceleration of commonly used

functions, instead a computer could contain one or more FPGAs. With this configuration, when an application requires fast graphics calculations, the FPGA will be reconfigured to handle those calculations. The FPGA will be able to run these operations much more quickly than the general purpose hardware since the FPGA will be specifically configured to handle those operations. When an application requires an encryption algorithm or image processing or many other possibilities, again the FPGA will be reconfigured to handle those operations and they will run much more quickly on the FPGA than on the original general purpose hardware. With a series of FPGA reconfigurations, a user can attain much better overall performance since the hardware will continually reconfigure itself to suit the current application.

Reprogrammability on the FPGAs is achieved through the use of hardware components called configurable logic blocks (CLBs). A CLB implements the functionality of logic gates. Based on the input into the CLB, different gates or logic designs can be implemented by a CLB. One FPGA can contain thousands of these CLBs, and these CLBs are interconnected through a variety of configurable data paths with appropriate topologies on the chip. When these CLBs are used together, they can implement any logic design. The currently configured design can be reprogrammed by sending signals to the FPGA which are then passed on to the CLBs. The signals received cause the CLBs to change their current functionality and as a result the whole FPGA logic design changes functionality and a new logic design is then available on the FPGA.

One of the areas in which FPGAs have been most useful is in ASIC prototyping.

As stated above, the process of designing an ASIC can be long and expensive. And the cost of correcting bugs found after the ASICs have been manufactured can be very high. This is one instance in which FPGAs can be helpful. An FPGA can be programmed with the proposed ASIC design, and then serve as a prototype for testing this design. Having a prototype allows designers to perform more intensive testing more quickly. Before FPGA prototyping, designers had to rely on software tools to test the design. These software tools are not capable of testing some aspects of an ASIC design such as timing constraints. Without this testing capability, designers were forced to have a few ASICs manufactured specifically for testing, knowing that they would not be the final design. Manufacturing these test ASICs added to the overall production cost. Another benefit of the FPGA prototyping is the ability to connect the FPGA prototype to the other components in the system. This allows designers to test the interfacing from the ASIC design perspective and from the perspective of the other components in the system. This testing allows ASICs designers and the other component designers to discover possible interfacing problems before the ASIC has been manufactured. In the end, FPGA prototyping results in a cheaper and more fully tested ASIC product since the design can be fully tested and corrected on the FPGA prototype.

## 2.2 Access Issues

While the FPGA technology offers new possibilities and capabilities to users, efficiently programming an FPGA can be so difficult as to turn away most users. When you have

one FPGA, programming an application onto it can be relatively easy. When you have a problem that is too large to fit onto one FPGA though, the problem of partitioning arises. Partitioning divides an application into sections that are then programmed onto different FPGAs. In order to divide an application efficiently, many constraints need to be considered in the partitioning algorithm. Accounting for all of these constraints can be very difficult and time-consuming.

Some of the constraints to be considered when partitioning an application have to do with the physical aspects of the FPGAs being used. Different FPGAs can hold different amounts of logic. Also, some FPGAs have on-chip memory in the form of RAM. The size of the RAM and where on the chip it is located are important factors in the design process. Similar designs can have large differences in their performance based on how they access the available on-chip memory. The interconnection networks and topologies are different among different FPGAs as well and smart designers must take this network into account. While these constraints are certainly factors in programming one FPGA, they become much more important and complex as the size of the problem and the number of FPGAs involved increase.

As stated above, efficiently partitioning an application onto multiple FPGAs is a hard task that requires technical expertise. In order to make FPGAs accessible to more users, software tools are needed to automate the partitioning process. Some tools have already been developed that automate parts of this process and more tools are currently being developed, but there is still a strong need for more comprehesive

software [OKS+00] [OKS+01] [Cha]. In order for all users to benefit from the hardware acceleration offered by FPGAs, new software tools are needed. The NetSolve interface to the FPGAs developed in this thesis will provide an easy-to-use interface to the hardware acceleration, but more work and more implementations on the FPGAs are still needed.

# Chapter 3

# Grid-Based Software

## 3.1 A NetSolve Overview

NetSolve is an example of middleware that unifies the software, hardware, and network components of a computational grid. Through the use of NetSolve, resources can be shared by users within the grid. NetSolve is based on a client-server-agent computing model. In this model, the client is the user attempting to access a resource available in the computational grid. The server owns the resource that the client wants to access. The agent is a "resource broker" in the sense that it determines which available server would best respond to the client request. Each of these three components will be discussed more fully below. Currently, NetSolve can run on Linux and all popular variants of the UNIX operating system, and the client code can be run on Linux, UNIX, Windows NT, and Windows 2000 operating systems. Additionally, you do not need root or superuser priviledges to run the agent, server, or client components of NetSolve.

The client-server-agent model used in NetSolve is a loosely-coupled heterogeneous environment. Loosely-coupled means that while different machines running NetSolve may be on the same local or global network, each machine may be owned or managed by different people or organizations. If a machine is running an agent, that machine's owner can decide when to make the agent available to users. If the machine is running a server, again the owner decides whether or not to make the server available, and also which functions that server will offer to users. The fact that the NetSolve environment is heterogeneous means that machines with different architectures, operating systems, and internal data representations can interact and share their resources. Sharing resources between different systems with incompatible data formats is enabled by the use of the common XDR protocol when transferring data. If two hosts do use the same data format, their common format is used rather than the standard XDR protocol in an effort to decrease transmission times of large amounts of data [AAB+].

## 3.2 The NetSolve Agent

As stated above, the agent is the "resource broker" in the NetSolve system. Each agent keeps a database of the different NetSolve servers along with each server's capabilities, usage statistics, and failure rates. When a user makes a NetSolve call, the request is first sent to the local NetSolve agent. The agent checks the client request to see if the function name and passed parameters are valid. If they are valid, the agent then checks its database to determine which servers can process the requested service. Based on the

usage statistics kept on each server in the database, the agent returns an ordered list of available servers that can handle the request to the client in order of lowest activity to highest activity.

Sending a list of potential servers to the client rather than one potential server name is part of the NetSolve fault tolerance scheme. If the first server on the list is unavailable to the client, the client can then try the next server name on the list. If all of the listed servers are unavailable, the client can then contact the agent once again for a new list of potential servers. This scheme allows for server failure without disrupting the rest of the NetSolve system and without increasing the network traffic between the client and the agent.

## 3.3 The NetSolve Client

The client in the NetSolve system is the user attempting to access remote resources. The client must request these resources through defined interfaces which can be divided into two groups: interactive and programming.

Interactive interfaces are useful in that the user does not have to write any code in order to access remote resources. Instead the user can gain access through much simpler interfaces. There are currently two types of interactive interfaces to NetSolve. The first is a tool interface which is available in MATLAB and Mathematica. Within these two software tools users can make NetSolve requests to remote servers from the interactive shells provided by the software. Both of these software tools also allow users to make

blocking or non-blocking NetSolve calls which allows some user-level parallelism. The other interactive interface is the shell interface that is available within a UNIX shell. This interface is functionally similar to the tool interfaces except that it doesn't do the same parameter checking as is done with MATLAB or Mathematica function calls. Here users must be more careful when passing parameters to the NetSolve functions [CD00].

Currently, programming interfaces are available in C and Fortran. When the appropriate NetSolve libraries are included in the C or Fortran code, users can make function calls using the libraries' function interfaces to request resources from remote servers. Java and Excel programming interfaces are currently in design. As with MATLAB, there are blocking and non-blocking NetSolve calls available in both C and Fortran that allow some parallelism.

Each function that is available through a NetSolve server has a calling sequence associated with it. When the client makes a NetSolve request through any of the interfaces mentioned above, the request and the function's parameters must match the calling sequence of the desired problem. A description of available functions along with their C and Fortran calling sequences is available and should be checked by users before trying to run their code. These descriptions can be found on the web at the NetSolve homepage (http://icl.cs.utk.edu/netsolve) or accessed through NetSolve management tools. More information on these tools can be found in Appendix A of this document.

## 3.4 The NetSolve Server

The servers in the NetSolve system are computational servers and are seen as resources by the agent. Different servers can service different sets of functions based on which libraries and functions have been installed on the server. When you download the NetSolve server code from the NetSolve website, some functions are included in the distribution download. The server administrator can then add more functionality to the server through the process described below.

To add a new function to a server's capabilities, you must first write a problem description file or PDF for that function. A function's PDF can be considered a "wrapper" for that function since it defines the interface between NetSolve and the software libraries installed on the server machine. The format of the PDF is highly structured and relies on the use of keywords to define different aspects of the function. The information about the function included in the PDF enables the server to respond to and service requests for that function.

The PDF can be divided logically into different sections based on the function characteristics being defined in that section. In the first section you must define the function name and the names and locations of the header files and libraries needed to compile the function on the server as well as the language in which the underlying library is written. Right now, NetSolve can only deal with libraries written in C or Fortran. Next, you must define the input and output parameters of the function. In NetSolve, all function parameters are defined as high-level data structures that are called Net-

Solve objects. Again, the definition of the input and output is highly structured and all function parameters must correspond to some NetSolve object. Once the inputs and outputs are defined, you must then define the calling sequence that specifies the order in which parameters need to be passed in the client code function call. The last section of the PDF is the pseudo-code section. This section contains any necessary preparation of the input parameters, the function call(s) to the underlying libraries, and any necessary updates to the output before it is passed back to the client. The calling sequence and the pseudo-code of the PDF are written in a special mnemonics syntax developed for NetSolve. This syntax is necessary so that the high-level NetSolve objects that describe the input and output of a function can relate to the low-level input and output descriptions in the library code. Additional optional information may be added to the PDF as well such as keywords that define the complexity of the function or specify that the function is parallel and uses MPI [AAB+].

## 3.5 NetSolve Load Balancing

Part of the SInRG goal is to provide users access to remote resources in an efficient manner. In order to achieve this goal, load balancing on the server machines in the NetSolve system must be considered and this task is performed by the NetSolve agent. Since there can be any number of available servers within a system and each server's workload may change often, the task of achieving a good load balance among the servers can be difficult. When a client makes a NetSolve request, the agent receives the request

first. It is the job of the agent to first determine which servers are capable of processing the request and then, of that set, which server would be the "best" server to process the request. The agent then returns a list of possible servers to the client in order of best choice server to worst choice server. Defining which server is "best" for a given problem can be a complicated task itself.

The classification of a server as good or bad is based on the execution time for a given problem. This execution time must be estimated by the agent for each server and each estimate can be divided into two parts: the time to transmit the input and output over the network and the time to perform the function on the server. The time to transmit the data can be further broken into the following pieces: the network latency and bandwidth between the client and the server and the sizes of the input and output to be sent. The time to perform the function can also be broken up, and it's subdivisions are: the size of the problem, the complexity of the function's algorithm(s), and the performance of the server. The performance of the server is further divided into the workload and the raw performance of the server and its calculation is described in more detail below.

To calculate a performance estimate for each server, NetSolve uses a simple theoretical model that takes the workload and raw performance of the server as inputs and returns the performance estimate. This model is defined as:

$$p = (P * 100 * n)/(100 * n + \max(w - 100 * (n - 1), 0))$$

where $p$ is the performance estimate, $P$ is the raw performance, $n$ is the number of

processors on the machine, and $w$ is the workload. The raw performance, $P$, is defined as the performance on the server when only the benchmarking software is run on the machine. NetSolve uses the LINPACK benchmark to get a performance rating for each server and this benchmark is run on the server when it is first started.

In the calculation of the estimated execution time, the parameters used can be broken into three different categories, client-dependent parameters, static server-dependent parameters, dynamic server-dependent parameters, as shown below.

The Client-dependent parameters are

- The size of the data to be sent to the server
- The size of the result to be received by the client
- The size of the problem

The client-dependent parameters are given to the agent when the client makes its function request.

The Static server-dependent parameters are

- The network characteristics between the server and the client
- The complexity of the algorithm to be run on the server
- The raw performance of the server

The static server-dependent parameters are known to the agent after each server begins to run. The complexity of the algorithm is known to the agent once the server is started

since this information is either included in the PDF of the function or a default value is used if this information is not given in the function's PDF. As stated above, each server determines its raw performance at startup and sends that information to the agent when the benchmarking is complete. The network characteristics are calculated several times and used to create an average value for the latency and the bandwidth. Even though these averages may change somewhat, they are still considered to be static since they are not expected to change significantly. These values are stored in the agent's database and are known when the client request is received.

There is one Dynamic server-dependent parameter:

- The workload on the server

The workload of the server is the only dynamic parameter in the calculation of the estimated execution time on the server. This value is periodically calculated by the server and sent to the agent. The agent stores this value in a cache and uses the currently cached value in its performance calculations when it receives a client request. In order to maintain an accurate workload value in the agent's cache, the workload value must be sent to the agent often but not so often as to burden the network with these transmissions or to burden the server with its workload calculations. Two parameters are used to determine how often the workload should be transmitted. The first is the time slice parameter which defines the minimum amount of time between workload transmissions. There is a minimum length of time because often the workload levels do not change dramatically and so they do not need to be sent after each time slice.

Instead, the second parameter is used to determine when the workload has changed enough to warrant retransmission. This parameter is called the confidence interval. If the workload changes enough to go beyond the confidence interval, then at the end of the current time slice the new workload level will be transmitted to the agent [CD00].

With server workload values cached in the agent, the agent is able to calculate the performance estimate for each server. This value allows the agent to then determine the estimated execution time for each server and ultimately determine which server is the "best" server to handle a client's request. Since this calculation is based on the changing workload of the servers, the network characteristics, and the problem function characteristics the agent is able to effectively balance the load of requests across all of the available servers.

## 3.6 NetSolve Fault Tolerance

Fault tolerance is an important part of the NetSolve implementation since the NetSolve system is loosely coupled. The servers within the system are owned by different people or groups and therefore servers may become unavailable at the server administrator's will. Also since the servers may be spread over a large network, a variety of different network problems may arise that prevent a client from accessing certain servers. Since these different types of failures are possible, the NetSolve system was designed to deal with these failures effectively without losing much performance.

Most system failures are detected by the client when trying to contact the server. If

a client does detect a failure, the client notifies the agent and then attempts to contact the next server on its list of possible servers sent to it by the agent when the function request was first sent. When the agent receives the failure report from the client, the agent marks the event in its server database. Within this database, the agent records if a machine is reachable or unreachable and also records if the server on that machine is running, stopped, or failed. If a failing server is restarted within a certain amount of time, the server will not be removed from the agent's list of servers. If the server is not restarted however, it will be removed from the agent's list and that server will not be sent future client requests. If a server is unreachable or stopped for more than a day, the server is removed from the agent's list of servers. The agent also records the number of failures a server experiences. If a server's number of failures exceeds the defined threshold, it will also be removed from the agent's list of servers.

The list of possible servers returned to the client from the agent after the client's NetSolve call is part of the NetSolve fault tolerance scheme. If a client is attempting to contact a server or if the function has already started to run on the server and then an error arises, the client can then try to use the next server on its list. Having the list prevents the client from having to contact the agent for a new server name after each failure. This can reduce the amount of network traffic and agent processing. If all of the servers in the list are having failures, the client can then contact the agent for a new list. Since each failure detected by the client is reported to the agent, the agent will return a new list to the client the second time around. So even if there are multiple

server failures, the system can continue to perform and the client will eventually receive a result from some functioning server [CD00].

# Chapter 4

# Achievements

## 4.1 The Cluster for Advanced Machine Design

Within the SInRG project, different clusters will be formed and each cluster will offer some special software or hardware resource. These clusters are called grid service clusters or GSCs. Each GSC will be managed by a different group. These GSCs are then networked together and can share their respective resources with users through the use of NetSolve. This thesis is based on work on the GSC built in the Electrical and Computer Engineering department at UT which is named the Cluster for Advanced Machine Design. This cluster contains reconfigurable hardware that provides hardware acceleration for some applications.

The Cluster for Advanced Machine Design is made up of many different pieces of hardware in order to provide the best performance and service. First there are eleven SUN 220R Dual 450MHz UltraSPARC II processors that run the Solaris operating

system. There are also 8 Pentium III PCs each with a Xilinx Virtex-1000 reconfigurable chip inside. The PCs run the Linux operating system. An A1000 RAID data storage unit is also in the GSC in order to provide enough space for the NetSolve function parameters of both local and remote user requests. A 1 Gbit/s Foundry data switch with 24 ports is used to connect these components to each other. This high speed switch provides advanced quality of service to the cluster [SIn]. The cluster is then linked to the rest of the SInRG grid through the 100 Mbit/s UT network line.

The Xilinx FPGAs attached to each of the PCs are implemented in a new reconfigurable computing environment called Pilchard. Pilchard was developed at the Chinese University of Hong Kong. In this implementation, the FPGA is plugged into the PC's standard 133 MHz synchronous dynamic RAM Dual In-line Memory Modules slot or DIMM slot. Previously, FPGAs were plugged into a PCI slot in the PC and used the PCI peripheral bus to communicate with the CPU and other devices. As FPGA technology has increased, the speed of standard bus' has not increased at a comparable rate and therefore data transfer bottlenecks occur. In an effort to avoid this bottleneck, the Pilchard system takes advantage of the DIMM slots available in order to obtain the higher bandwidth and lower latency offered by memory [LLC$^+$]. Ultimately, attaching the FPGAs to a DIMM slot rather than a PCI slot results in faster execution time of a function.

## 4.2 Software Requirements

In order for the Cluster for Advanced Machine Design to participate in the NetSolve system and make its resources available to remote users, two additional pieces of software are required on the server machine beyond the NetSolve server software and all software components required by the NetSolve server to run, eg. a C code compiler. First there must be a software library containing the code of the function or functions that will be made available to users. Second a PDF, described in section 3.1, is needed to define the interface between NetSolve and the function code. For some of the functions offered by this GSC, a software version and a hardware accelerated version will be available to users. Offering both forms of the function will allow users to benchmark the functions and determine how much speedup is achieved with the special-purpose hardware. So in addition to the previously stated software requirements, in order to run on an FPGA, a VHDL or some other hardware description language version of the function is needed. VHDL is the traditionally used language in this field. The VHDL code then goes through a synthesis process in which the software application is mapped onto the FPGA hardware. At the end of the synthesis you get a configuration file, which is simply a bit stream, for that specific application for the specific FPGA being used. When the configuration file is loaded onto that FPGA, the bit stream defines for the FPGA how its CLBs should be set in order for the desired logic design to be implemented in the hardware. So the software version of the function will run in the CPU and the hardware version of the function will load the configuration file onto the

FPGA and the application will be run there. A PDF must be created for both the software and hardware version of the function.

### 4.2.1 Code

In this project we focused on one application as our prototype for future implementations. This prototype is based on the fast-fourier transformation problem or FFT. The software version of FFT and the PDF for the software version of FFT are provided below with a description as an example. The PDF for the hardware version of the FFT function appears in appendix B of this document. The VHDL code for the FFT function appears in appendix D.

**C Code**

```
#include <stdio.h>
#include <math.h>

#define DATATYPE16 short
#define DATATYPE32 long
#define SCALE 32767

DATATYPE16 *real, *imag, *cosine, *sine;
int FFTSIZE;

/*****************************************************************/

read_input(char *in_file)
{
  FILE *fd;
  int i, r, im;

  fd = fopen(in_file, "r");
```

```
  for(i=0; i< FFTSIZE; i++)
  {
        fscanf(fd, "%d %d\n", &r, &im);
        real[i] = (DATATYPE16)r;
        imag[i] = (DATATYPE16)im;
  }
  fclose(fd);

}

/*****************************************************************/

print_result(char *out_file)
{
  FILE *fd;
  int i;

  fd = fopen(out_file, "w");

  for(i=0; i<FFTSIZE; i++)
        fprintf(fd, "%d %d\n", real[i], imag[i]);

  fclose(fd);
}
/*****************************************************************/

init_coefs()
{
  int i;
  double angle, angleinc;

  angle=0;
  angleinc=2.0*M_PI/FFTSIZE;

  for(i=0;i<FFTSIZE;i++)
  {
    cosine[i] = (DATATYPE16)( SCALE * cos(angle));
    sine[i]   = (DATATYPE16)(-SCALE * sin(angle));
    angle += angleinc;
  }
}
```

```
/*****************************************************************/

perform_FFT()
{
  int index1, index2, index3, aindex=0, inc=1;
  int step = FFTSIZE/2;
  DATATYPE32 real32, imag32;

  inc = 1;
  for (index1=FFTSIZE; index1 > 1; index1/=2 )
  {
    for (index2=0; index2 < step; index2++)
    {
      for (index3=index2; index3 < FFTSIZE; index3 += (step<<1))
      {
        real32   = (real[index3] - real[index3+step]) * cosine[aindex];
        real32  -= (imag[index3] - imag[index3+step]) * sine[aindex];
        imag32   = (imag[index3] - imag[index3+step]) * cosine[aindex];
        imag32  += (real[index3] - real[index3+step]) * sine[aindex];

        real[index3]   = real[index3] + real[index3+step];
        imag[index3]   = imag[index3] + imag[index3+step];

        real[index3+step]   = real32/32768;
        imag[index3+step]   = imag32/32768;
      }
      aindex = aindex + inc;
    }
    aindex = 0;
    inc += inc;
    step= step>>1;
  }
 }
}

/*****************************************************************/

void fft(char* inputfile, char* outputfile, int size)
{

  FFTSIZE = size;
```

```
  real = (DATATYPE16*)malloc(FFTSIZE*sizeof(DATATYPE16));
  imag = (DATATYPE16*)malloc(FFTSIZE*sizeof(DATATYPE16));
  cosine = (DATATYPE16*)malloc(FFTSIZE*sizeof(DATATYPE16));
  sine = (DATATYPE16*)malloc(FFTSIZE*sizeof(DATATYPE16));

  read_input(inputfile);
  init_coefs();
  perform_FFT();
  print_result(outputfile);

  free(real);
  free(imag);
  free(sine);
  free(cosine);

}
```

### 4.2.2 PDF's

Below is the FFT PDF that defines the NetSolve interface to the FFT C code shown above.

```
@PROBLEM fft
@INCLUDE </avocado/homes/lehrter/my_netfuncs/libfpga.h>
@LIB /avocado/homes/lehrter/my_netfuncs/libfpga.a
@LANGUAGE C
@MAJOR ROW
@PATH /fpga/
@DESCRIPTION
This function performs an in-place decimation-in-frequency operation
(Sande-Tukey FFT), leaving the output shuffled. Bit reversal
is not performed.
@INPUT 2
@OBJECT FILE CHAR Input1
input file
@OBJECT SCALAR I  Input2
size
@OUTPUT 1
```

```
@OBJECT FILE CHAR Output1
output file
@COMPLEXITY 1,1
@CALLINGSEQUENCE
@ARG I0
@ARG O0
@ARG I1
@CODE
fft(@I0@, @O0@,*@I1@);
@END_CODE
```

## 4.3 Implementations

The Cluster for Machine Design is currently still in development. The eleven SUN processors are in place and connected to the grid, but the eight PCs that will contain the FPGAs are not yet in the cluster. Until the PCs and the attached reconfigurable hardware are in place, the cluster will not be fully operational. Even once all of the hardware is in place, developing this cluster into a truly powerful resource will take time as new C and VDHL codes and PDFs are being designed and implemented for this GSC.

The FFT function described in the previous section serves as our prototype application and eventually, once the hardware is in place, both software and hardware versions of this code will be available on the ECE cluster. Currently, only the software version of FFT is complete. As secondary prototypes, the Data Encryption Standard (DES) function and a backprojection algorithm are being implemented for the ECE GSC as well. The DES implementation is complete and both the hardware and software version of this function are available. Testing results regarding the DES functions, both

hardware and software versions, are included in the next section. The PDFs and codes for the DES functions can be found in the appendices of this document. None of the necessary backprojection codes are yet available, but the PDFs for these functions can be found in appendix B.

## 4.4 Results

Below the results of preliminary timing tests on the DES functions are given. Before these results are listed though, some discussion of the DES functions and the testing environment is necessary.

The DES functions used do not represent a fully functional DES implementation. The hardware version of the DES function was designed to be a test of the interface to the FPGA hardware, and the software version was designed to offer a comparison when benchmarking the hardware acceleration achieved. As a result of these intentions, the customary inputs to DES functions, which are the plaintext to be encrypted and the DES keys used in the encryption, are hardcoded into the functions. Due to this hardcoding of the parameters, the calling functions pass no inputs to the DES functions. Since the inputs are known, the output ciphertext is also known, so the output is not returned to the calling function either. Only an integer return value is returned from these functions.

Timing tests were performed on the software and hardware versions of the DES function with NetSolve and without. Regarding the tests that invoke the DES functions

through the NetSolve server, in these examples the agent, client, and server processes all ran on the same processor. As a result, the message transfer time between the different processes, which is a part of the overall execution time, is at a minimum. This testing model does not represent most real world applications of NetSolve, however these results are the first to be obtained from this system and the need for more types of testing is realized. Other students who continue this project in the future will have to adjust this testing model for a more complete view of the system.

Figure 4.1 displays the results found in the preliminary DES function testing described above. In this figure the values on the edges represent the respective runtimes of the four different tests conducted. The fastest runtime is found when the user invokes the hardware version of the DES function without using NetSolve. In this case, no NetSolve server, client, or agent is involved and the user calls the hardware DES function as it would any function available in its local libraries. The other three runtimes are then normalized to the hardware DES runtime.

As shown in the figure, the software version of DES has a runtime that is three times as long as the hardware version, so the use of the FPGA in the hardware version offers a measurable speedup over the software implementation. However when the user invokes either of these functions through the NetSolve interface, a dramatic increase in the runtimes is observed. When NetSolve is used, a number of network connections are established between the agent, client, and server. As a result of the time needed to establish these connections, a significant overhead is added to every NetSolve call.

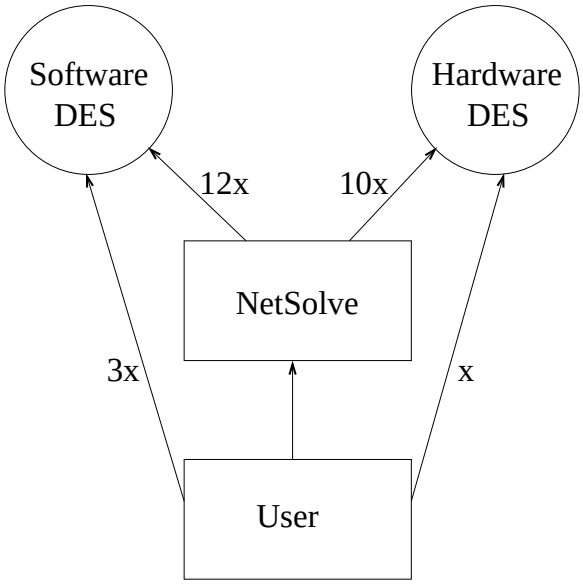


Figure 4.1: *Normalized DES Results.*

Even though the NetSolve overhead could be observed in any function's implementation, the overhead is particularly overwhelming for this function due to it's relatively fast execution time. The execution times for the four different invokations of the DES function are given in Table 4.1. As shown in this table, the execution time for the hardware DES is roughly 0.3 seconds. This is the value represented by x in Figure 4.1 to which all other runtimes were normalized. Looking at the hardware version of DES with NetSolve runtime, the nearly 3 second overhead added by NetSolve is obvious.

As stated before, these DES functions require no input and return no output and the agent, server, and client are all running on the same machine, so the data transfer times involved in the NetSolve function runtimes are at a minimum. For other functions that require inputs of large size or return outputs of large size, this NetSolve overhead

Table 4.1: *Preliminary DES Runtimes*

| | Function call without NetSolve | Function call with NetSolve |
|---|---|---|
| Hardware version of DES | 328,759 usec | 3,426,360 usec |
| Software version of DES | 892,989 usec | 3,995,576 usec |

will only increase. Also, for runtime environments in which the agent, client, and server are each running on separate machines, as is expected for most users, the transfer time included in the overhead will also increase as the machines will be more physically removed from each other. However, the overhead involved in establishing the necessary network connections for a NetSolve call may decrease if the agent, client, and server are run on separate processors. When the three components are run on the same processor, they must share that processor. If they run on separate machines though, the components may achieve some parallel execution and therefore reduce the overhead by some amount. The additional testing required to determine how much each of these factors will affect the runtimes is left to those who will continue this work in the future.

# Chapter 5

# Conclusions

## 5.1 Roadblocks Encountered

The Cluster for Advanced Machine Design is the first non-Computer Science GSC to be brought online in the SInRG project and so a number of problems were faced as we began this new work. A list of some of the roadblocks encountered is given below.

1. Learning to use the new NetSolve software and learning how to create the function interfaces posed a formidable problem in the beginning of this project.

2. Delays in the hardware shipment, specifically the PCs with their reconfigurable Pilchard boards which are being built in Hong Kong, have made creating the NetSolve software interface more difficult since the hardware has not been available for a full test of the interface. Only recently have the PCs become accessible online. The DES testing described in section 4.4 was performed remotely on the

PCs which still reside in Hong Kong.

3. Determining how to copy a configuration file onto the FPGA's memory from within a software function.

4. The long process of writing the VHDL code and then mapping it onto the new Pilchard FPGAs. It's been the case in this project that the PDFs and the client test codes for the functions are ready long before the hardware implementation of the function is complete.

## 5.2 Objectives Achieved

The goals for this thesis were simply to provide an efficient and easy-to-use NetSolve interface to the special-purpose hardware available in the Electrical and Computer Engineering department's GSC. Following the software design implemented by NetSolve, these goals were achieved and an interface to the hardware that will become available in the ECE cluster was created. The number of applications that can take advantage of the hardware acceleration available will increase as work on this project continues and as a result of this work, general users will be able to take advantage of the hardware acceleration found in FPGAs.

# Chapter 6

# Future Work

## 6.1 Increasing Functionality

This thesis only represents the beginning of the work to be done on the Cluster for Advanced Machine Design. More work is required in the future in order to make this GSC as useful and efficient to users as possible. Below is a list of possible work that can be done in the future in order to achieve this.

1. Now that the prototype application is in place, more functions need to be added to the the ECE servers' capabilities in order to make more applications available to users.

2. PDFs have been created for the backprojection algorithm and a DES encryption algorithm, but the PDFs have not been finalized because the function interfaces to these two functions continue to change. Once the interfaces are finalized for

these functions, the PDFs can then be finalized as well. The current version of the PDFs for these algorithms based on the current function interfaces can be found in Appendix B of this document.

3. As stated in section 3.5 on NetSolve Load Balancing, the LINPACK benchmarks are performed when a server starts up in order to get a raw performance value for that machine. LINPACK tests the MFLOPS rate on the machine, but unfortunately, this rating is not valid for machines that offer special-purpose hardware as their resource since the MFLOPS rate does not represent anything about the FPGA's speed or capabilities. In the future, benchmarks that can measure FPGA statistics should be included in the NetSolve software.

4. Currently there is no method for a server to indicate to the agent or a client that it contains specialized hardware that offers function acceleration. Some method of expressing and quantifying this capability is needed in future NetSolve releases.

5. When a user's application runs on an FPGA, part of the execution time is overhead incurred by the time it takes to load a function configuration onto the FPGA. Currently, each time a user makes a function request the corresponding function configuration is loaded onto the FPGA and then processing begins. Some time could be saved and performance increased if the system could detect if the new function request is the same as the previous one, and therefore there would be no need to load the same configuration again and spend that time on overhead. Right now, that type of configuration detection is not available, but it's creation

would be beneficial to the performance of the system.

## 6.2 Open Questions

1. How much acceleration is gained by implementing functions in hardware rather than software? How much of a speedup is found for the FFT and backprojection functions specifically? Once the PCs with the attached Pilchard boards are available and installed in the ECE GSC, these times can be found.

2. While the FPGAs offer acceleration to users, how much time is spent in NetSolve overhead such as transfer time when making a NetSolve call to functions other than DES? And at what point does it become faster to run the software version on your local machine rather than the accelerated hardware version on a remote machine?

3. The process of mapping a VHDL application onto the FPGA is a difficult process and one of the bottlenecks in the progress of this project. One way to make this process easier and faster is to automate the mapping process. But how much of this process can be automated and how could that be done most efficiently?

4. Some applications may require more FPGA chip area to be implemented than is available on one FPGA chip. In this case, the application will need to be partitioned over two or more FPGAs. When this happens, what type of communication should occur between the different PCs over which the problem is partitioned?

And how will this kind of job scheduling interfere with the job scheduling and workload management performed by the NetSolve agent?

5. As stated in the previous section, each NetSolve server generates processor speed benchmark data that is used by the NetSolve agent for load balancing among all of the servers. Since this type of information does not represent the performance of the FPGAs, which is where the processing is done for these applications, which benchmarking techniques are appropriate for reconfigurable hardware and how well could the NetSolve agent understand these ratings?

# Bibliography

# Bibliography

[AAB+] Dorian Arnold, Sudesh Agrawal, Susan Blackford, Jack Dongarra, Michelle Miller, and Sathish Vadhiyar. *Users' Guide to NetSolve V1.4*. Innovative Computing Laboratory, Department of COmputer Science, University of Tennessee. http://icl.cs.utk.edu.

[CD99] Henri Casanova and Jack Dongarra. Netsolve's Network Enabled Server: Examples and Applications. Proceedings of the Heterogeneous Computing Workshop, Orlando, FL 1998, February 1999.

[CD00] Henri Casanova and Jack Dongarra. Netsolve: A Network-enabled Server for Solving Computational Science Problems. Department of Computer Science, University of Tennessee, May 2000.

[Cha] Champion: A Software Design Environment for Adaptive Computing Systems and ASICs. http://microsys6.engr.utk.edu/ bouldin/darpa/acs/index.html.

[Con] Overview of the Condor High-Throughput Computing System. http://www.cs.wisc.edu/condor.

[FK97] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[GW96] Andrew S. Grimshaw and Wm A. Wulf. Legion - A View from 50,000 Feet. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, August 1996.

[GWF+94] Andrew S. Grimshaw, William A. Wulf, James C. French, Alfred C. Weaver, and Paul F. Reynolds Jr. A Synopsis of the Legion Project. Technical report, University of Virginia, 1994.

[KM98] Ralph Kohler and Richard C. Metzger. Benchmarking Tools and Assessment Environment for Configurable Computing; Scalability Stressmark. Benchmark Specification Document CDRL A100, USA Intelligence Center and Fort Huachuca; Rome Laboratory; Honeywell, Inc., August 1998.

[LLC+] P.H.W. Leong, M.P. Leong, O.Y.H Cheung, T. Tung, C.M. Kwok, and K.H. Lee. Pilchard - A Reconfigurable Computing Platform with Memory Slot Interface. Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT Hong Kong.

[Nin] Ninf: A Network Based Information Library for Global World-Wide Computing Infrastructure. http://ninf.apgrid.org.

[OKS+00] S. Ong, N. Kerkiz, B. Srijanto, C. Tan, M.A. Langston, D.F. Newport, and D.W. Bouldin. Design Flow for Automatic Mapping of Graphical Programming Applications to Adaptive Computing Systems. Workshop on High Performance Embedded Computing, September 2000.

[OKS+01] S. Ong, N. Kerkiz, B. Srijanto, C. Tan, M.A. Langston, D.F. Newport, and D.W. Bouldin. Automatic Mapping of Multiple Applications to Multiple Adaptive Computing Systems. IEEE Symposium on Filed-Programmable Custom Computing Systems, April 2001.

[SIn] Scalable Intracampus Research Grid. http://icl.cs.utk.edu/sinrg.

[VBR+] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard. Programmable Active Memories: Reconfigurable Memories: Reconfigurable Systems Come of Age. Digital Equipment Corporation's Paris Research Laboratory.

# Appendix

# Appendix A

# User's Guide

This appendix gives an overview of how to use and manage different components of the NetSolve V1.4 software and highlights some important details of the software's use. This section's guide is not complete and for a more detailed guide to NetSolve please download the official NetSolve Users' Guide V1.4 available at http://icl.cs.utk.edu/netsolve.

In the following sections two variables are used within the directions to define specific directories within your home area. The first is NETSOLVE_ROOT. The NetSolve software is by default installed in a directory in your home directory called NetSolve-1.4. This directory is referred to as NETSOLVE_ROOT. The second variable used is NETSOLVE_ARCH which defines which operating system on which you are running the NetSolve software. To determine which operating system you are using, you can run the following command: $NETSOLVE_ROOT/conf/config.guess. The string returned by config.guess is assigned to NETSOLVE_ARCH except that the string output from

the config.guess command contains hyphens, but in the NETSOLVE_ARCH string all of the hyphens are replaced by underscores.

No superuser or root priviledges are required to run any part of the NetSolve software.

## A.1 The Client

In order to run a NetSolve client, you must first download the NetSolve client software available at http://icl.cs.utk.edu/netsolve. Two different downloads are available at this site. One download contains only the client source code and the second contains the client, server, and agent code. The client software can run on UNIX, Linux, and Windows operating systems. However only the Matlab client interface is available on the Windows platform. To run client code on your local machine that makes NetSolve calls to remote resources, follow these steps. Please note though that these steps only apply to running clients on the Linux and UNIX operating system. For directions on running a client in Windows please refer to the Users' Guide.

1. Download the client software from the NetSolve website.

2. Follow the directions for unzipping, configuration, and installation given in the NetSolve Users' Guide V1.4. The NetSolve installation is configured to the operating sytem on which you are running by using the GNU tool "configure". More details on this tool are given in the Users' Guide.

3. NetSolve client interfaces are available in C, Fortran, Matlab, and Mathematica. Based on which interface you want to use, compile the client code accordingly. More than one interface may be compiled into the client code. More details on this step can be found in the NetSolve Users' Guide. The following steps are based around using the C code client interface.

4. Each NetSolve call within your client code must correspond to a function on the list of available NetSolve functions. This list can be found on the NetSolve website through the use of a CGI script or by using one of the NetSolve management tools which is invoked with the following command available in $NETSOLVE_ROOT/bin/$NETSOLVE_ARCH:

   NS_problems agenthostname.domain

   More information on this and other management tools can be found in the Users' Guide.

5. Each NetSolve call must also match the calling sequence that is defined for the inputs and outputs to that function. Information on the calling sequence and inputs and outputs to a function can be found online as well or with the following management tool available again in $NETSOLVE_ROOT/bin/$NETSOLVE_ARCH:

   NS_problemdesc agenthostname.domain problem_name

6. The prototype for a non-blocking function call in C looks like:

   int netsl(char *problem_name, ..argument list..);

   The first argument is the name of the problem you want to run and the rest of the

arguments are the inputs and outputs to the function and are listed in the order specified by the problem's predefined calling sequence. An important key to note is that all of the input and output parameters passed to the NetSolve call should be pointers to the input and output values. The following example will clarify this point. Consider the function named add and its problem description given below, which can be found using the NetSolve tools mentioned above:

```
-- add -- This functions takes 2 integers as input and returns
          the interger sum of the two.
* 2 objects in INPUT
 - input 0: Scalar Integer.
 pointer to integer a
 - input 1: Scalar Integer.
 pointer to integer b
* 1 objects in OUTPUT
 - output 0: Scalar Integer.
 pointer to sum of two input integers
* Calling sequence from C or Fortran
3 arguments
 - Argument #0:
   - pointer to input object #0 (a)
 - Argument #1:
   - pointer to input object #1 (b)
 - Argument #2:
   - pointer to output object #0 (sum)
```

One correct way to call this function through the C interface is:

```
int a, b, result;
a=2; b=4;
info = netsl("add()", &a, &b, &result);
```

where a and b are the inputs and result is the output. The only exception to passing pointers to the input and output values is when the parameter represents the size of a matrix or vector. In that case you should pass an interger value.

7. Once your code is written, you must include the appropriate NetSolve header file and link to the appropriate NetSolve library in order to compile your code. The appropriate header file and library to use is based on which operating system you want to run your client code. One way to include and link to the correct files is as follows. Include the following line at the beginning of your client code:

   #include "netsolve.h"

   And then to compile a code called test-add.c, for example, use a Makefile similar to the following:

```
NETSOLVE_ROOT = /home/lehrter/NetSolve-1.4
NETSOLVE_ARCH = i686_pc_linux_gnu

CC = gcc
INCDIR = -I$(NETSOLVE_ROOT)/include
NETSOLVECLIB = $(NETSOLVE_ROOT)/lib/$(NETSOLVE_ARCH)/libnetsolve.a
CFLAGS = $(INCDIR) -g
#LIBS = -lsocket -lnsl  When running on UNIX use this line instead.
LIBS = -lnsl

test-add: test-add.c
        $(CC) $(CFLAGS) -o $@ test-add.c $(NETSOLVECLIB) $(LIBS)
```

8. Before you can run your code, you must specify with which NetSolve agent you want your client code to communicate. An agent is always running at net-

solve.cs.utk.edu so that is a good choice to use, but any running agent may be specified. To specify your agent from a csh enter the following or replace the agent's name with another:

setenv NETSOLVE_AGENT netsolve.cs.utk.edu

9. Now you can start your code and get access to some remote resource by simply entering your executable name and any required arguments at the command line prompt.

10. If you want to run your client code on a different operating system from what you compiled your code on originally, you must recompile your code on the new operating system before running it. Each compilation links to files specific to a certain architecture and so recompliation is required everytime the architecture changes. If you use the makefile given above to compile your code, then to recompile you only need to edit the line in which NETSOLVE_ARCH is defined in order to link to the now appropriate files.

## A.2 The Server

In order to run a NetSolve server, you must first download the NetSolve server software available at http://icl.cs.utk.edu/netsolve. The server software download also includes the agent and client software. The server software can run only on the Linux and UNIX operating systems.

1. Download the server software from the NetSolve website.

2. Follow the directions for unzipping, configuration, and installation given in the NetSolve Users' Guide V1.4. The NetSolve installation is configured to the operating sytem on which you are running by using the GNU tool "configure". More details on this tool are given in the Users' Guide.

3. To compile your server, go to the $NETSOLVE_ROOT directory and enter either "make server" to compile only the server or "make standard" which compiles the server, client, agent, tools, and tests. Consult the official Users' Guide for more directions on compiling and testing the server software.

4. The server software download contains some numerical software libraries and so you could start your server at this point and offer only these included functions to remote users. Before you start your server though you must edit the $NETSOLVE_ROOT/server_config file and specify with which agent you want your server to register. An agent process is always running at netsolve.cs.utk.edu, so that is a good choice to use. Now you can start the server by going to the $NETSOLVE_ROOT/bin/$NETSOLVE_ARCH directory and entering the command "server" to start the server process. You can pass arguments to the server when starting it; details on these arguents can be found in the Users' Guide. Unless otherwise specified in the server command arguments, a log of the server's activities is kept in the file named $NETSOLVE_ROOT/nsserver.log.

5. To end the server process, go to the $NETSOLVE_ROOT/bin/$NETSOLVE_ARCH directory and enter:

   NS_killserver agenthostname.domain serverhostname.domain

While the distribution server software download includes some numerical software, you can add more functions to your server and expand its functionality. This process can be confusing, but the following steps will hopefully clarify the critical points in the process while deferring to the official Users' Guide for a more complete explanation.

1. First, to offer a function to users through NetSolve, the function's source or object code that is compiled on that machine's architecture must be located on the machine on which the NetSolve server will run. Currently, NetSolve can only handle functions and libraries written in C or Fortran.

2. For each function you want to offer through your NetSolve server you must create a Problem Description File or PDF. The PDF defines the interface between the NetSolve server, which will receive the client code request, and the local function code, which will execute the client code request. When a function is added to the server, during recompilation the function's PDF goes through a "source code generator" which creates the service function that makes the actual call to the local function when a client request is received. The PDF is written in a special syntax or mnemonics in order to facilitate the conversion from the high-level data structures used by the NetSolve system to the low-level language dependent data types used in the service function and the library source code. The use of these

mnemonics makes the PDF difficult to read and hard to write correctly, but NetSolve V1.4 includes a GUI interface to writing PDFs that helps to ease some of these difficulties. So you can create the PDF in an editor without the use of the GUI and follow the descriptions and examples given in the official Users' Guide or you can use the GUI interface. More information on using the GUI will be given shortly.

3. When creating the PDF most mistakes are made in the section named Code. The Code section defines the body of the service function which will be run on the server when a request is made for that function by some client code. There are two important keys to remember when writing the Code section. The first is that all function parameters are pointers to the input and output data except for parameters that represent the size of a matrix or vector. So when passing the parameters to the library function call in the Code section of the PDF, for parameters that correspond to inputs or outputs in the library function prototype that aren't pointers, you must dereference these parameters with the * operator. Second, when the client call is sent to the server, the input values are sent to the server as well and stored by the service function, but no space is ever allocated for the output values. In order to avoid a segmentation fault on the server, before the library function call in the Code section you must allocate space for all output values that are not pointers themselves. These two keys are illustrated by the following PDF for the function add described in the client section above.

```
@PROBLEM add
@DASHI /avocado/homes/lehrter/my_netfuncs/testadd
@INCLUDE "libadd.h"
@LIB /avocado/homes/lehrter/my_netfuncs/testadd/libadd.a
@LANGUAGE C
@MAJOR ROW
@PATH /test/
@DESCRIPTION
This functions takes 2 integers as input and returns
the interger sum of the two.
@INPUT 2
@OBJECT SCALAR I a
pointer to integer a
@OBJECT SCALAR I b
pointer to integer b
@OUTPUT 1
@OBJECT SCALAR I sum
pointer to sum of two input integers
@COMPLEXITY 1,1
@CALLINGSEQUENCE
@ARG I0
@ARG I1
@ARG O0
@CODE
@O0@ = (int*)malloc(sizeof(int));
*@O0@ = add(*@I0@,*@I1@);

@END_CODE
```

Notice that the library function add which is called in the Code section receives two integers as input and returns one integer as ouput. However since NetSolve passes pointers to the service function, you must dereference the pointers to obtain the integer values passed from the client. This PDF file is simply named "add" after the function for whose interface this file defines. No extension is added to the PDF filenames. Other PDFs are given in Appendix B of this document and may serve as examples to new users.

4. This step refers to the use of the GUI to write the PDF and to some problems found with this approach. To use the GUI, you must first go to $NETSOLVE_ROOT and enter "make pdfgui" to compile the GUI for your current architecture. Once compilation is complete, you can start the GUI application by going to the $NETSOLVE_ROOT/bin/$NETSOLVE_ARCH directory and entering "NS_pdfgui". Refer to the Users' Guide for a description of the different sections of the PDF. The following gives a listing of hints to creating the PDF with the GUI and how to avoid some problems associated with the GUI PDF generation.

    - When on the "Input" or "Ouput" screen, for a parameter that is a file, the object type should be FILE and the data type should be CHAR. However, the GUI does not offer the option of specifying this combination. So temporarily select some other parameter specification and then after saving the information to a PDF file be sure to edit the file and change the parameter specification to the correct values.
    - In the Input and Output sections of the PDF the object types must be printed in all capital letters. The GUI prints them out to the PDF file with the first letter capitalized and the rest in lower case. Be sure to edit these strings to be all capitals before trying to recompile your server.
    - Sometimes the parameters' names and parameter dereferencing in the commands entered in the Code section of the GUI aren't translated correctly into

the NetSolve syntax used in the PDF. Again, after you save the PDF check the Code section and make sure the correct syntax and dereferencing is used.

- When you save the information to the PDF file, go to the menu and select "Save As". From the "Files of type" box choose to save it as a "nspdf" file, which is the format that the server software can understand. The other format "xmlpdf" cannot be compiled into the server.

5. Once the PDF is complete, it should be copied to the $NETSOLVE_ROOT/problems directory, dropping the "nspdf" extension if the GUI was used to create the PDF, which is where the other PDFs are located.

6. All functions that are to be compiled into the server need to appear in the problems list given in the $NETSOLVE_ROOT/server_config file. To include the add function to the server, the server_config file should have a problems list similar to this.

```
@PROBLEMS:
./problems/testing
./problems/qsort
./problems/mandelbrot
./problems/blas_subset
./problems/lapack_subset
./problems/add
```

The first five problems listed above are included in the NetSolve distribution, but the sixth listing corresponds to the add function to be added to the server.

The other sections of the server_config file define several other characteristics of how the server will function when running. A description of this is left to the official Users' Guide.

7. Now you are ready to recompile the server and include the new function in the server's capabilities. To recompile go to the directory $NETSOLVE_ROOT and enter "make server".

8. Once recompilation is complete, you can start your server again and now the list of available functions on your server will have increased.

## A.3 The Agent

In order to run a NetSolve agent, you must first download the NetSolve agent software available at http://icl.cs.utk.edu/netsolve. The agent software is included in the download that contains the server and client software as well. The agent software can run only on the Linux and UNIX operating systems. In order to run your NetSolve client or server, you do not need to run your own NetSolve agent. Only one agent per grid is necessary, and there is always an agent running on netsolve.cs.utk.edu. If you choose to run your own agent though you must do the following:

1. Download the software from the NetSolve website mentioned above.

2. Follow the directions for unzipping and installation given in the NetSolve Users' Guide V1.4.

3. Compile the agent code by entering “make agent” or “make standard”, which also compiles the client, server, tools, and tests, at your prompt from your $NETSOLVE_ROOT directory.

4. Enter the command “agent” at the prompt from your $NETSOLVE_ROOT/bin/$NETSOLVE_ARCH directory to start your agent process. You can pass some arguments to the agent software when you start it. A description of these arguments is available in the official NetSolve Users’ Guide.

Once you start the agent, the agent will keep a log of its activity in a file named $NETSOLVE_ROOT/nsagent.log, unless another name is specified at instantiation. The agent does not require any interaction from its administrator, so you should not have to deal with it again until you want to end the agent process. To do this enter either of the following commands from the $NETSOLVE_ROOT/bin/$NETSOLVE_ARCH directory:

- NS_killagent agenthostname.domain
- NS_killall agenthostname.domain

The first option above kills only the agent process, while the second option kills all servers and agents connected to the named agent.

# Appendix B

# PDFs

Since both a software and an accelerated or hardware version of the functions will be offered on the ECE cluster, two PDFs are required to implement these two functions. The two PDFs will look alike except in the Code section where different functions will be called. The PDFs for the additional functions that will be included in the ECE server are given in this section.

## B.1 FFT

The problem description file for the software version of FFT, called fft:

```
@PROBLEM fft
@INCLUDE </avocado/homes/lehrter/my_netfuncs/libfpga.h>
@LIB /avocado/homes/lehrter/my_netfuncs/libfpga.a
@LANGUAGE C
@MAJOR ROW
@PATH /fpga/
```

```
@DESCRIPTION
It performs an in-place decimation-in-frequency operation
(Sande-Tukey FFT), leaving the output shuffled. Bit reversal
is not performed.
@INPUT 2
@OBJECT FILE CHAR Input1
input file
@OBJECT SCALAR I  Input2
size
@OUTPUT 1
@OBJECT FILE CHAR Output1
output file
@COMPLEXITY 1,1
@CALLINGSEQUENCE
@ARG I0
@ARG O0
@ARG I1
@CODE
fft(@I0@, @O0@,*@I1@);
@END_CODE
```

The problem description file for the hardware version of FFT, called ffta which means the accelerated version of fft:

```
@PROBLEM ffta
@INCLUDE </avocado/homes/lehrter/my_netfuncs/libfpga.h>
@LIB /avocado/homes/lehrter/my_netfuncs/libfpga.a
@LANGUAGE C
@MAJOR ROW
@PATH /fpga/
@DESCRIPTION
It performs an in-place decimation-in-frequency operation
(Sande-Tukey FFT), leaving the output shuffled. Bit reversal
is not performed.  This function is implemented is hardware on
an FPGA and offers better performance than the FFT available
in software.
NOTE:  This function can only accept input files of size 512,
4096, or 16384.
```

```
@INPUT 2
@OBJECT FILE CHAR Input1
input file
@OBJECT SCALAR I  Input2
size
@OUTPUT 1
@OBJECT FILE CHAR Output1
output file
@COMPLEXITY 1,1
@CALLINGSEQUENCE
@ARG I0
@ARG O0
@ARG I1
@CODE
ffta(@I0@, @O0@,*@I1@);
@END_CODE
```

## B.2 Backprojection

The PDF for the software version of the backprojection function, called backprojection.

```
@INCLUDE </avocado/homes/lehrter/my_netfuncs/libfpga.h>
@LIB /avocado/homes/lehrter/my_netfuncs/libfpga.a
@LANGUAGE C
@MAJOR ROW
@PATH /fpga/
@DESCRIPTION
This function implements the software version of the
backprojection function.  The input sinogram data
file is a 3D data file that represents 128 points by
128 angles by 63 planes deep.
@INPUT 2
@OBJECT FILE CHAR sinogram
name of input sinogram data file
@OBJECT FILE CHAR offset
name of input offset data file
@OUTPUT 1
```

```
@OBJECT FILE CHAR out
output image filename
@COMPLEXITY 1,1
@CALLINGSEQUENCE
@ARG I0
@ARG I1
@ARG O0
@CODE
backprojection(@I0@, @I1@, @O0@);
@END_CODE
```

The PDF for the hardware version of the backprojection function, called backprojectiona.

```
@PROBLEM backprojectiona
@INCLUDE </avocado/homes/lehrter/my_netfuncs/libfpga.h>
@LIB /avocado/homes/lehrter/my_netfuncs/libfpga.a
@LANGUAGE C
@MAJOR ROW
@PATH /fpga/
@DESCRIPTION
This function implements the hardware accelerated version
of the backprojection function.  The input sinogram data
file is a 3D data file that represents 128 points by
128 angles by 63 planes deep.
@INPUT 2
@OBJECT FILE CHAR sinogram
name of input sinogram data file
@OBJECT FILE CHAR offset
name of input offset data file
@OUTPUT 1
@OBJECT FILE CHAR out
output image filename
@COMPLEXITY 1,1
@CALLINGSEQUENCE
@ARG I0
@ARG I1
@ARG O0
@CODE
```

```
backprojectiona(@I0@, @I1@, @OO@);
@END_CODE
```

## B.3 DES Encryption

Note that in the following two PDFs, both CODE sections make calls to functions that have no inputs. No input needs to be passed to the functions because the inputs are hardcoded in the function code and therefore the output is already known. These functions are currently designed this way because they are being used to test the system. The lack of inputs caused a problem with NetSolve which requires at least one input and one output for a function. In order to work with this requirement, the PDFs do define an interface that requires one integer as input, but as you can see in the PDFs below, this parameter will be ignored by the service function, but the client code must pass it nonetheless.

The PDF for the software version of the DES function, called des.

```
@PROBLEM des
@INCLUDE </avocado/homes/lehrter/my_netfuncs/libfpga.h>
@LIB /avocado/homes/lehrter/my_netfuncs/libfpga.a
@LANGUAGE C
@MAJOR ROW
@PATH /fpga/
@DESCRIPTION
@INPUT 1
@OBJECT SCALAR I go
ignored - dummy variable used to make NetSolve interface work since
it requires at least one input to be listed
```

```
@OUTPUT 1
@OBJECT SCALAR I status
status value returned from the dessw function
@COMPLEXITY 1,1
@CALLINGSEQUENCE
@ARG IO
@ARG OO
@CODE
@OO@ = (int*)malloc(sizeof(int));
*@OO@=dessw();
@END_CODE
```

The PDF for the hardware version of the DES function, called desa.

```
@PROBLEM desa
@INCLUDE </avocado/homes/lehrter/my_netfuncs/libfpga.h>
@LIB /avocado/homes/lehrter/my_netfuncs/libfpga.a
@LANGUAGE C
@MAJOR ROW
@PATH /fpga/
@DESCRIPTION
@INPUT 1
@OBJECT SCALAR I go
ignored - dummy variable used to make NetSolve interface work since
it requires at least one input to be listed
@OUTPUT 1
@OBJECT SCALAR I status
status value returned from the deshw function
@COMPLEXITY 1,1
@CALLINGSEQUENCE
@ARG IO
@ARG OO
@CODE
@OO@ = (int*)malloc(sizeof(int));
*@OO@=deshw();
@END_CODE
```

# Appendix C

# C Codes

## C.1 FFT

The C source code of the software version of the FFT algorithm is shown in section 4.2.1 of this document and will not be reprinted here. This is the code that will be run on the NetSolve server when a user requests the software version of the FFT algorithm. The code that requests the hardware version of the FFT function is not yet available.

## C.2 DES

The following code is the software implementation of the DES function. The dessw.c file is listed first and then the header file included by the function code, dessw.h, is listed.

`dessw.c:`

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <sys/fcntl.h>

#include "dessw.h"
#define BUFSIZE ((long)1024)

typedef struct
{
        int w[2];
}int64;

int dessw()
{

        float sw_time;
        long sw_ttime;
        float bps;
        int fd;
        int rc;
        int64   data[512],control;
        int64   ct[512],t;
        int i,j,k,l;
        long total_time;
        struct timeval t1,t2;
        char * mempwp,* memprp, * reg;
        long max_tran;

        static unsigned char sw_buf[BUFSIZE];
        static des_cblock
        sw_key1={0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef};
        static des_cblock sw_key2=
{0x34,0x56,0x78,0x9a,0xbc,0xde,0xf0,0x12};
        static des_cblock sw_key3=
{0x56,0x78,0x9a,0xbc,0xde,0xf0,0x12,0x34};
        des_key_schedule sw_sch1,sw_sch2,sw_sch3;
```

```
        DES_LONG sw_data[2];

for(i=0; i<512; i++)
{
data[i].w[0]=i;
data[i].w[1]=0;
}
max_tran = 1048576;

        des_set_key((C_Block *)sw_key1,sw_sch1);
        des_set_key((C_Block *)sw_key2,sw_sch2);
        des_set_key((C_Block *)sw_key3,sw_sch3);

        gettimeofday(&t1,NULL);

        for(i=0;i<max_tran;i++)
        {
                sw_data[0]=ntohl(data[0].w[1]);
                sw_data[1]=ntohl(data[0].w[0]);
                des_encrypt(sw_data,&(sw_sch1),DES_ENCRYPT);
        }
        gettimeofday(&t2,NULL);

        sw_data[1]=ntohl(sw_data[1]);
        sw_data[0]=ntohl(sw_data[0]);

        sw_ttime=(t2.tv_sec-t1.tv_sec)*1000000+(t2.tv_usec-t1.tv_usec);
        sw_time = (float)max_tran*8.0/(float)sw_ttime;

        printf("[Plaintext] %08X %08X\n",data[0].w[1],data[0].w[0]);
        printf("[Libdes ciphertext] %08X %08X\n",sw_data[0],sw_data[1]);
        printf("Software :Libdes perform %d DES encryption in %d usec\n",
max_tran,sw_ttime);
        printf("Software :Libdes %f MByte per sec\n",sw_time);
return 0;
}


dessw.h:

/* crypto/des/des.org */
/* Copyright (C) 1995-1997 Eric Young (eay@cryptsoft.com)
```

```
*  All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to.  The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code.  The SSL documentation
* included with this distribution is covered by the same copyright terms
* except that the holder is Tim Hudson (tjh@cryptsoft.com).
*
* Copyright remains Eric Young's, and as such any Copyright notices in
* the code are not to be removed.
* If this package is used in a product, Eric Young should be given
* attribution as the author of the parts of the library used.
* This can be in the form of a textual message at program startup or
* in documentation (online or textual) provided with the package.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the copyright
*    notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the distribution.
* 3. All advertising materials mentioning features or use of this software
*    must display the following acknowledgement:
*    "This product includes cryptographic software written by
*     Eric Young (eay@cryptsoft.com)"
*    The word 'cryptographic' can be left out if the rouines from the
*    library being used are not cryptographic related :-).
* 4. If you include any Windows specific code (or a derivative thereof)
*    from the apps directory (application code) you must include an
*    acknowledgement:
*"This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
*
* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS
* OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
```

```
* DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR
* ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
* IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
*
* The licence and distribution terms for any publically available version
* or derivative of this code cannot be changed.  i.e. this code cannot
* simply be copied and put under another distribution licence
* [including the GNU Public Licence.]
*/

/* WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
*
* Always modify des.org since des.h is automatically generated from
* it during SSLeay configuration.
*
* WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
*/

#ifndef HEADER_DES_H
#define HEADER_DES_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <stdio.h>

/* If this is set to 'unsigned int' on a DEC Alpha, this gives about a
 * %20 speed up (longs are 8 bytes, int's are 4). */
#ifndef DES_LONG
/* RCSID $Id: des.h,v 1.5 1999/04/05 22:02:01 henry Exp $ */
/* This conditional for FreeS/WAN project. */
#ifdef __alpha
#define DES_LONG unsigned int
#else
#define DES_LONG unsigned long
#endif
```

```
#endif

int dessw();

typedef unsigned char des_cblock[8];
typedef struct des_ks_struct
        {
        union   {
                des_cblock _;
                /* make sure things are correct size on machines with
                 * 8 byte longs */
                DES_LONG pad[2];
                } ks;
#undef _
#define _       ks._
        } des_key_schedule[16];

#define DES_KEY_SZ      (sizeof(des_cblock))
#define DES_SCHEDULE_SZ (sizeof(des_key_schedule))

#define DES_ENCRYPT     1
#define DES_DECRYPT     0

#define DES_CBC_MODE    0
#define DES_PCBC_MODE   1

#define des_ecb2_encrypt(i,o,k1,k2,e) \
        des_ecb3_encrypt((i),(o),(k1),(k2),(k1),(e))

#define des_ede2_cbc_encrypt(i,o,l,k1,k2,iv,e) \
        des_ede3_cbc_encrypt((i),(o),(l),(k1),(k2),(k1),(iv),(e))

#define des_ede2_cfb64_encrypt(i,o,l,k1,k2,iv,n,e) \
        des_ede3_cfb64_encrypt((i),(o),(l),(k1),(k2),(k1),(iv),(n),(e))

#define des_ede2_ofb64_encrypt(i,o,l,k1,k2,iv,n) \
        des_ede3_ofb64_encrypt((i),(o),(l),(k1),(k2),(k1),(iv),(n))

#define C_Block des_cblock
#define Key_schedule des_key_schedule
#ifdef KERBEROS
#define ENCRYPT DES_ENCRYPT
```

```
#define DECRYPT DES_DECRYPT
#endif
#define KEY_SZ DES_KEY_SZ
#define string_to_key des_string_to_key
#define read_pw_string des_read_pw_string
#define random_key des_random_key
#define pcbc_encrypt des_pcbc_encrypt
#define set_key des_set_key
#define key_sched des_key_sched
#define ecb_encrypt des_ecb_encrypt
#define cbc_encrypt des_cbc_encrypt
#define ncbc_encrypt des_ncbc_encrypt
#define xcbc_encrypt des_xcbc_encrypt
#define cbc_cksum des_cbc_cksum
#define quad_cksum des_quad_cksum

/* For compatibility with the MIT lib - eay 20/05/92 */
typedef des_key_schedule bit_64;
#define des_fixup_key_parity des_set_odd_parity
#define des_check_key_parity check_parity

extern int des_check_key;       /* defaults to false */
extern int des_rw_mode;         /* defaults to DES_PCBC_MODE */

/* The next line is used to disable full ANSI prototypes, if your
 * compiler has problems with the prototypes, make sure this line always
 * evaluates to true :-) */
#if defined(MSDOS) || defined(__STDC__)
#undef NOPROTO
#endif
#ifndef NOPROTO
char *des_options(void);
void des_ecb3_encrypt(des_cblock *input,des_cblock *output,
        des_key_schedule ks1,des_key_schedule ks2,
        des_key_schedule ks3, int enc);
DES_LONG des_cbc_cksum(des_cblock *input,des_cblock *output,
        long length,des_key_schedule schedule,des_cblock *ivec);
void des_cbc_encrypt(des_cblock *input,des_cblock *output,long length,
        des_key_schedule schedule,des_cblock *ivec,int enc);
void des_ncbc_encrypt(des_cblock *input,des_cblock *output,long length,
        des_key_schedule schedule,des_cblock *ivec,int enc);
void des_xcbc_encrypt(des_cblock *input,des_cblock *output,long length,
```

```
        des_key_schedule schedule,des_cblock *ivec,
        des_cblock *inw,des_cblock *outw,int enc);
void des_cfb_encrypt(unsigned char *in,unsigned char *out,int numbits,
        long length,des_key_schedule schedule,des_cblock *ivec,int enc);
void des_ecb_encrypt(des_cblock *input,des_cblock *output,
        des_key_schedule ks,int enc);
void des_encrypt(DES_LONG *data,des_key_schedule ks, int enc);
void des_encrypt2(DES_LONG *data,des_key_schedule ks, int enc);
void des_encrypt3(DES_LONG *data, des_key_schedule ks1,
        des_key_schedule ks2, des_key_schedule ks3);
void des_decrypt3(DES_LONG *data, des_key_schedule ks1,
        des_key_schedule ks2, des_key_schedule ks3);
void des_ede3_cbc_encrypt(des_cblock *input, des_cblock *output,
        long length, des_key_schedule ks1, des_key_schedule ks2,
        des_key_schedule ks3, des_cblock *ivec, int enc);
void des_ede3_cfb64_encrypt(unsigned char *in, unsigned char *out,
        long length, des_key_schedule ks1, des_key_schedule ks2,
        des_key_schedule ks3, des_cblock *ivec, int *num, int enc);
void des_ede3_ofb64_encrypt(unsigned char *in, unsigned char *out,
        long length, des_key_schedule ks1, des_key_schedule ks2,
        des_key_schedule ks3, des_cblock *ivec, int *num);

void des_xwhite_in2out(des_cblock (*des_key), des_cblock (*in_white),
        des_cblock (*out_white));

int des_enc_read(int fd,char *buf,int len,des_key_schedule sched,
        des_cblock *iv);
int des_enc_write(int fd,char *buf,int len,des_key_schedule sched,
        des_cblock *iv);
char *des_fcrypt(const char *buf,const char *salt, char *ret);
#ifdef PERL5
char *des_crypt(const char *buf,const char *salt);
#else
/* some stupid compilers complain because I have declared char instead
 * of const char */
#ifdef HEADER_DES_LOCL_H
char *crypt(const char *buf,const char *salt);
#else
char *crypt();
#endif
#endif
void des_ofb_encrypt(unsigned char *in,unsigned char *out, int
```

```
        numbits,long length,des_key_schedule schedule,des_cblock *ivec);
void des_pcbc_encrypt(des_cblock *input,des_cblock *output,long length,
        des_key_schedule schedule,des_cblock *ivec,int enc);
DES_LONG des_quad_cksum(des_cblock *input,des_cblock *output,
        long length,int out_count,des_cblock *seed);
void des_random_seed(des_cblock key);
void des_random_key(des_cblock ret);
int des_read_password(des_cblock *key,char *prompt,int verify);
int des_read_2passwords(des_cblock *key1,des_cblock *key2,
        char *prompt,int verify);
int des_read_pw_string(char *buf,int length,char *prompt,int verify);
void des_set_odd_parity(des_cblock *key);
int des_is_weak_key(des_cblock *key);
int des_set_key(des_cblock *key,des_key_schedule schedule);
int des_key_sched(des_cblock *key,des_key_schedule schedule);
void des_string_to_key(char *str,des_cblock *key);
void des_string_to_2keys(char *str,des_cblock *key1,des_cblock *key2);
void des_cfb64_encrypt(unsigned char *in, unsigned char *out, long length,
        des_key_schedule schedule, des_cblock *ivec, int *num, int enc);
void des_ofb64_encrypt(unsigned char *in, unsigned char *out, long length,
        des_key_schedule schedule, des_cblock *ivec, int *num);
int des_read_pw(char *buf, char *buff, int size, char *prompt, int verify);

/* Extra functions from Mark Murray <mark@grondar.za> */
void des_cblock_print_file(des_cblock *cb, FILE *fp);
/* The following functions are not in the normal unix build or the
 * SSLeay build.  When using the SSLeay build, use RAND_seed()
 * and RAND_bytes() instead. */
int des_new_random_key(des_cblock *key);
void des_init_random_number_generator(des_cblock *key);
void des_set_random_generator_seed(des_cblock *key);
void des_set_sequence_number(des_cblock new_sequence_number);
void des_generate_random_block(des_cblock *block);

#else

char *des_options();
void des_ecb3_encrypt();
DES_LONG des_cbc_cksum();
void des_cbc_encrypt();
void des_ncbc_encrypt();
void des_xcbc_encrypt();
```

```
void des_cfb_encrypt();
void des_ede3_cfb64_encrypt();
void des_ede3_ofb64_encrypt();
void des_ecb_encrypt();
void des_encrypt();
void des_encrypt2();
void des_encrypt3();
void des_decrypt3();
void des_ede3_cbc_encrypt();
int des_enc_read();
int des_enc_write();
char *des_fcrypt();
#ifdef PERL5
char *des_crypt();
#else
char *crypt();
#endif
void des_ofb_encrypt();
void des_pcbc_encrypt();
DES_LONG des_quad_cksum();
void des_random_seed();
void des_random_key();
int des_read_password();
int des_read_2passwords();
int des_read_pw_string();
void des_set_odd_parity();
int des_is_weak_key();
int des_set_key();
int des_key_sched();
void des_string_to_key();
void des_string_to_2keys();
void des_cfb64_encrypt();
void des_ofb64_encrypt();
int des_read_pw();
void des_xwhite_in2out();

/* Extra functions from Mark Murray <mark@grondar.za> */
void des_cblock_print_file();
/* The following functions are not in the normal unix build or the
 * SSLeay build.  When using the SSLeay build, use RAND_seed()
 * and RAND_bytes() instead. */
#ifdef FreeBSD
```

```
int des_new_random_key();
void des_init_random_number_generator();
void des_set_random_generator_seed();
void des_set_sequence_number();
void des_generate_random_block();
#endif

#endif

#ifdef  __cplusplus
}
#endif

#endif
```

The following is the DES function code that implements the hardware version of the function. Again, the deshw.c file is given first, followed by its header file deshw.h.

deshw.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <sys/fcntl.h>

#include "deshw.h"

#define USEMOVQ

char *memp = NULL;

void write64(int64 twrite, char *addr)
{
#ifdef  USEMOVQ
```

```
        __asm__ __volatile__(
        "
                movl %1,%%ecx\n
                movq %0,%%mm0\n
                movq %%mm0,(%%ecx)\n
        "
        :
        : "m" (twrite), "g" (addr)
        );
#else
        *((int *)(addr + 4)) = twrite.w[1];
        *((int *)addr) = twrite.w[0];
#endif
}

void read64(int64 *data, char *addr)
{
#ifdef  USEMOVQ
        int64   tread;

        __asm__ __volatile__(
        "
                movl %1,%%ecx\n
                movq (%%ecx),%%mm1\n
                movq %%mm1,%0\n
        "
        : "=m" (tread)
        : "g" (addr)
        );
        data->w[0] = tread.w[0];
        data->w[1] = tread.w[1];
#else
        data->w[0] = *((int *)addr);
        data->w[1] = *((int *)(addr + 4));
#endif
}

void write32(int twrite, char *addr)
{
        *((int *)addr) = twrite;
}
```

```
void read32(int *data, char *addr)
{
        *data = *((int *)addr);
}

int deshw()
{
        float bps;
        int fd;
        int rc;
        int64   data[512],control;
        int64   ct[512],t;
        int i,j,k;
        long total_time;
        struct timeval t1,t2;
        char * mempwp,* memprp, * reg;
        long max_tran;

        fd = open(DEVICE, O_RDWR);
        if ((memp = (char *)mmap(NULL,MTRRZ,PROT_READ,MAP_PRIVATE,
                fd,0)) == MAP_FAILED)
        {
                perror(DEVICE);
                return 1;
        }

        mempwp=memp;
        memprp=memp;
        reg=memp+32*8;

        for(i=0;i<512;i++)
        {
                data[i].w[0]=i;
                data[i].w[1]=0;
        }
        max_tran = 1048576;
        gettimeofday(&t1,NULL);

/* Main loop for DES core */
for(k=0;k<max_tran;k+=512)
{
        for(j=0;j<512;j+=32)
```

```
        {
                /* reset control register for initialize
                 buffer address in pilchard */
                control.w[0]=0;
                control.w[1]=0;
                write64(control,reg);

                /* filling data to buffer in pilchard (32) */
                for(i=0;i<32;i++)
                {
                        write64(data[i+j],memp+i*8);
                        /* early triggering DES core to start */
                        if(i==1)
                        {
                               control.w[0]=0xffffffff;
                                control.w[1]=0xffffffff;
                                write64(control,reg);
                        }
                }

                /* read back data from pilchard */
                for(i=0;i<32;i++)
                {
                        read64(&t,memp+i*8);
                        ct[i+j].w[1]=t.w[1];
                        ct[i+j].w[0]=t.w[0];
                }
        }
}
        gettimeofday(&t2,NULL);
        total_time=(t2.tv_sec-t1.tv_sec)*1000000+(t2.tv_usec-t1.tv_usec);

        for(i=0;i<32;i++)
        {
                printf("plaintext [%d] %x %x\n",
                i,data[i].w[1],data[i].w[0]);
                printf("crypttext [%d] %x %x\n",
                i,ct[i].w[1],ct[i].w[0]);
        }
        printf("%d usec\n",total_time);
        munmap(memp, MTRRZ);
        close(fd);
```

```
        return 0;
}
```

deshw.h:

```
#define DEVICE          "/dev/pilchard"
#define MTRRZ           0x100000

typedef struct
{
        int w[2];
} int64;

extern int pilchard(int, char **);
extern void write64(int64, char *);
extern void read64(int64 *, char *);
extern void write32(int, char *);
extern void read32(int *, char *);

extern char *memp;

int deshw();
```

# Appendix D

# VHDL Codes

## D.1 FFT

The FFT VHDL code for the Pilchard board is still being written. Currently, it lacks the functions in which the code interfaces with the Pilchard's RAM. The existing FFT VHDL code is presented below. This FFT code is designed to run on one FPGA only.

Four files are presented below. They are pe1lca.vhd which is the top level VHDL file for FPGA implementation, fft1_top.vhd which instantiates all blocks needed to perform the FFT, fft1.vhd which performs the FFT computations, and state.vhd which implements the state machine for the FFT.

pe1lca.vhd:

```
------------------------------------------------------
--
--  Copyright (C) 1996-1997 Annapolis Micro Systems, Inc.
```

```
--
------------------------------------------------------
-
--
--  Entity        : PE1_Logic_Core
--
--  Architecture  : Blank
--
--  Filename      : pe1lca.vhd
--
--  Description   : "Blank" PE1_Logic_Core part.  Ties all
--                  outputs to default (inactive) states.
--
--  Date          : 8/29/96
--
--  Author        : Brad Fross, AMS; Chris Esser, AMS
--
-------------------------------------------------------------------
---------------------------- Library Declarations ----------------
-

library IEEE;
use IEEE.std_logic_1164.all;
 use IEEE.std_logic_misc.all;
 use IEEE.std_logic_arith.all;
 use IEEE.std_logic_unsigned.all;

-------------------------- Architecture Declaration --------------
-

architecture Blank of PE1_Logic_Core is

constant Passes : integer := 12;
signal resetn : std_logic;
signal bus_grant : std_logic;
signal term : std_logic;

 component interface
    port(clk                  :       IN      std_logic;
         resetn               :       IN      std_logic;
         interrupt_ackn       :       IN      std_logic;
         bus_grantn           :       IN      std_logic;
```

```
        done                :       IN      std_logic;
        bus_reqn            :       OUT     std_logic;
        interrupt_reqn      :       OUT     std_logic);
 end component;

 component fft1_top
    generic(Passes : INTEGER);
    port(  Clk              :    IN       std_logic;
           Resetn           :    IN       std_logic;
           BusGrant         :    IN       std_logic;
           Complete         :    IN       std_logic;
           WEn              :    OUT      std_logic;
           Strobe           :    OUT      std_logic;
           Address          :    OUT      std_logic_vector(Passes downto 0);
           MemoryDataIn     :    IN       std_logic_vector(31 downto 0);
           MemoryDataOut    :    OUT      std_logic_vector(31 downto 0));
 end component;
begin

-------------------------------------------------------------------
--  "Inactive" output port signal assignments
-------------------------------------------------------------------

        PE_Left_OE <= ( others => '0' );    -- Disable left port output

        PE_Right_OE <= (others => '0');

        PE_XbarData_OE <= (others => '0');
        PE_XbarData_WE <= (others => '0');

        PE_FifoSelect <= "00";

        PE_Fifo_WE_n <= '1';
        PE_FifoPtrIncr_EN <= '0';

        PE_CPE_Bus_OE <= ( others => '0' ); -- Disable CPE bus output

 resetn <= not(PE_Reset);
 bus_grant <= not(PE_MemBusGrant_n);

 PE_MemAddr_OutReg(21 downto Passes+1) <= (others => '0');
```

```
 host_interface : interface
  port map(clk => PE_Pclk, resetn => resetn,
           interrupt_ackn => PE_InterruptAck_n,
           bus_grantn => PE_MemBusGrant_n,
           done => term,
           bus_reqn => PE_MemBusReq_n,
           interrupt_reqn => PE_InterruptReq_n);

 fft1_block : fft1_top
  generic map(Passes => Passes)
  port map(Clk => PE_Pclk, Resetn => resetn,
           BusGrant => bus_grant, complete => term,
           WEn => PE_MemWriteSel_n,
           Strobe => PE_MemStrobe_n,
           Address => PE_MemAddr_OutReg(Passes downto 0),
           MemoryDataIn => PE_MemData_InReg,
           MemoryDataOut => PE_MemData_OutReg);

end Blank;
```

fft1_top.vhd:

```
-------------------------------------------------------------------
--  Copyright (September 1999) Honeywell Inc. Unpublished - All rights
--  reserved. This material may be reproduced by, or for, the U.S.
--  Government pursuant to the copyright license under the clause at
--  DFARS 252.227-7013 (Oct. 1988).
-------------------------------------------------------------------
--
-- Filename: fft1_top.vhd
-- Title: FFT PE1 top level behavioral description
--
-- Description:
--
-------------------------------------------------------------------
LIBRARY IEEE;
 use IEEE.std_logic_1164.all;
 use IEEE.std_logic_misc.all;
 use IEEE.std_logic_arith.all;
 use IEEE.std_logic_unsigned.all;
```

```
LIBRARY DEFAULT;
 use DEFAULT.fftpack.all;

ENTITY fft1_top IS
   GENERIC (Passes : INTEGER);
   PORT (
           Clk             :  IN    std_logic;
           Resetn          :  IN    std_logic;
           BusGrant        :  IN    std_logic;
           Complete        :  OUT   std_logic;
           WEn             :  OUT   std_logic;
           Strobe          :  OUT   std_logic;
           Address         :  OUT   std_logic_vector(Passes downto 0);
           MemoryDataIn    :  IN    std_logic_vector(31 downto 0);
           MemoryDataOut   :  OUT   std_logic_vector(31 downto 0)
   );
END fft1_top;

ARCHITECTURE fft1_top_arch OF fft1_top IS

COMPONENT control1
   GENERIC (Passes : INTEGER);
   PORT(
         Clk             :   IN      std_logic;
         Resetn          :   IN      std_logic;
         current_state   :   IN      std_logic_vector(3 downto 0);
         term            :   OUT     std_logic;
         pass            :   BUFFER  integer RANGE 0 to Passes-1;
         address         :   BUFFER  std_logic_vector(Passes downto 0)
   );
END COMPONENT;

COMPONENT state
   PORT (
         Clk            :       IN      std_logic;
         Resetn         :       IN      std_logic;
         BusGrant       :       IN      std_logic;
         Term           :       IN      std_logic;
         current_state  :       BUFFER  std_logic_vector(3 downto 0)
   );
END COMPONENT;
```

```
COMPONENT fft1
   GENERIC (Passes : INTEGER);
   PORT (
           Clk             :    IN       std_logic;
           Resetn          :    IN       std_logic;
           current_state   :    IN       std_logic_vector(3 downto 0);
           pass            :    IN       integer RANGE 0 to Passes-1;
           memory_data_in  :    IN       std_logic_vector(31 downto 0);
           memory_data_out :    OUT      std_logic_vector(31 downto 0)
   );
END COMPONENT;

signal pass : integer RANGE 0 to Passes-1;
signal term : std_logic;
signal buff_address : std_logic_vector(Passes downto 0);

signal current_state : std_logic_vector(3 downto 0);

begin

ctrl : control1
               GENERIC MAP(Passes => Passes)
               PORT MAP(Clk => Clk, Resetn => Resetn,
                        current_state => current_state,
                        term => term, pass => pass,
address => buff_address);

state_mach: state
                  PORT MAP(Clk => Clk, Resetn => Resetn,
                           BusGrant => BusGrant,
                           Term => term,
                           current_state => current_state);

fft_block : fft1
                 GENERIC MAP(Passes => Passes)
                 PORT MAP(Clk => Clk, Resetn => Resetn,
                          current_state => current_state, pass => pass,
                          memory_data_in => MemoryDataIn,
                          memory_data_out => MemoryDataOut);

Address <= buff_address;
```

```
PROCESS(current_state)
 begin
    if (current_state = write0 or current_state = write1) then
       WEn <= '0';
    else
       WEn <= '1';
    end if;
END PROCESS;

PROCESS(current_state)
 begin
    if (current_state = strobe1 or current_state = strobe2) then
       Strobe <= '1';
    else
       Strobe <= '0';
    end if;
END PROCESS;

PROCESS(current_state)
 begin
    if (current_state = done) then
       Complete <= '1';
    else
       Complete <= '0';
    end if;
END PROCESS;

end fft1_top_arch;

CONFIGURATION config_fft1_top OF fft1_top IS
   FOR fft1_top_arch
   END FOR;
END config_fft1_top;
```

fft1.vhd:

```
--------------------------------------------------------------------
--  Copyright (September 1999) Honeywell Inc. Unpublished - All rights
--  reserved. This material may be reproduced by, or for, the U.S.
--  Government pursuant to the copyright license under the clause at
```

```
--  DFARS 252.227-7013 (Oct. 1988).
-------------------------------------------------------------------------
--
-- Filename: fft1.vhd
-- Title: FFT PE1 behavioral description
--
-- Description: The following code performs the complex multiply and
--       complex butterfly add of the FFT operation. The FFT is
--       implemented on a single processing elements. Acquistion and
--       transmission of complex data from/to other processing
--       elements is also handled by this code. The data is assumed to
--       be intially loaded into memory.
-------------------------------------------------------------------------
LIBRARY IEEE;
 use IEEE.std_logic_1164.all;
 use IEEE.std_logic_misc.all;
 use IEEE.std_logic_arith.all;
 use IEEE.std_logic_unsigned.all;

LIBRARY DEFAULT;
 use DEFAULT.fftpack.all;

ENTITY fft1 IS
   GENERIC (Passes : INTEGER);
   PORT (
           Clk               :    IN   std_logic;
           Resetn            :    IN   std_logic;
           current_state     :    IN   std_logic_vector(3 downto 0);
           pass              :    IN   integer RANGE 0 to Passes-1;
           memory_data_in    :    IN   std_logic_vector(31 downto 0);
           memory_data_out   :    OUT  std_logic_vector(31 downto 0)
   );
END fft1;

ARCHITECTURE fft1_arch OF fft1 IS

signal x0r, x0i, x1r, x1i, wr, wi : signed(15 downto 0);
signal zr, zi : signed(15 downto 0);
signal wr_xr, wi_xi, wr_xi, wi_xr : signed(31 downto 0);
signal z : signed(31 downto 0);
signal cmult : signed(31 downto 0);
signal real32, imag32 : signed(31 downto 0);
```

```
signal w, x0, x1 : signed(15 downto 0);

CONSTANT last_pass : integer := Passes-1;
begin

memory_data_out <= STD_LOGIC_VECTOR(zi & zr);

PROCESS(clk)
   begin
      if (clk'event and clk='1') then
         if (current_state = read_weight) THEN
            x0r <= SIGNED(memory_data_in(15 downto 0));
            x0i <= SIGNED(memory_data_in(31 downto 16));
         end if;
         if (current_state = strobe1) THEN
            x1r <= SIGNED(memory_data_in(15 downto 0));
            x1i <= SIGNED(memory_data_in(31 downto 16));
         end if;
        if (current_state = transfer1) THEN
            wr <= SIGNED(memory_data_in(15 downto 0));
            wi <= SIGNED(memory_data_in(31 downto 16));
         end if;
      end if;
END PROCESS;

 -- muliplier data and weights are selected based on the stage
 -- of the FFT and also the current state
 process(current_state, wr, wi, x0r, x0i, x1r, x1i, pass, memory_data_in)
    begin
       case current_state IS
          when transfer1 =>
             x0 <= x0r;
             x1 <= x1r;
             w <= SIGNED(memory_data_in(15 downto 0));
          when transfer2 =>
             x0 <= x0i;
             x1 <= x1i;
             w <= wi;
          when transfer3 =>
             x0 <= x0r;
             x1 <= x1r;
             w <= wi;
```

```
            when transfer4 =>
               x0 <= x0i;
               x1 <= x1i;
               w <= wr;
            when others =>
               x0 <= "----------------";
               x1 <= "----------------";
               w <= "----------------";
         end case;
   end PROCESS;

   process(x0, x1, w)
      begin
         cmult <= w*(x0-x1);
   end process;

   -- individual multiply operations needed for complex multiply
   process(clk)
      begin
         if (clk'event and clk='1') then
            if (current_state = transfer1) then
               real32 <= cmult;
            elsif (current_state = transfer2) then
               real32 <= real32 - cmult;
            end if;

            if (current_state = transfer3) then
               imag32 <= cmult;
            elsif (current_state = transfer4) then
               imag32 <= imag32 + cmult;
            end if;
         end if;
   end PROCESS;

   -- Completion of complex multiplications using previous
   -- individual multiplies.
   -- Complex butterfly add also performed.
   process(pass, current_state, x0r, x0i, x1r, x1i, real32, imag32)
      variable temp : signed(31 downto 0);
      constant round_up :
  signed(31 downto 0) := "00000000000000000100000000000000";
      begin
```

```
          if (current_state = write0) then
             zr <= x0r + x1r;
             zi <= x0i + x1i;
          else
             temp := real32 + round_up;
             zr <= temp(30 downto 15);
             temp := imag32 + round_up;
             zi <= temp(30 downto 15);
          end if;
 end PROCESS;
end fft1_arch;

CONFIGURATION config_fft1 OF fft1 IS
   FOR fft1_arch
   END FOR;
END config_fft1;
```

state.vhd:

```
-----------------------------------------------------------------------
--  Copyright (September 1999) Honeywell Inc. Unpublished - All rights
--  reserved. This material may be reproduced by, or for, the U.S.
--  Government pursuant to the copyright license under the clause at
--  DFARS 252.227-7013 (Oct. 1988).
-----------------------------------------------------------------------
--
-- Filename: state.vhd
-- Title: FFT state machine behavioral description
--
-- Description: The following VHDL code performs the state machine
--              operation of the FFT implementation.
--
-----------------------------------------------------------------------
LIBRARY IEEE;
 use IEEE.std_logic_1164.all;
 use IEEE.std_logic_misc.all;
 use IEEE.std_logic_arith.all;
 use IEEE.std_logic_unsigned.all;

LIBRARY DEFAULT;
```

```
 use DEFAULT.fftpack.all;

ENTITY state IS
   PORT (
         Clk              :       IN      std_logic;
         Resetn           :       IN      std_logic;
         BusGrant         :       IN      std_logic;
         Term             :       IN      std_logic;
         current_state    :       BUFFER  std_logic_vector(3 downto 0)
   );
END state;

ARCHITECTURE state_arch OF state IS

signal next_state : std_logic_vector(3 downto 0);

begin

statetrans: PROCESS(Clk,Resetn)
 begin
      if (Resetn = '0') then
         current_state <= idle;
      elsif (Clk'event and Clk='1') then
        if (BusGrant = '1') then
         current_state <= next_state;
        end if;
      end if;
end process;

state_mach: PROCESS(current_state, Term)
 begin
  case current_state is
    when idle =>
       next_state <= read_data0;
    when read_data0 =>
       next_state <= read_data1;
    when read_data1 =>
       next_state <= read_weight;
    when read_weight =>
       next_state <= strobe1;
    when strobe1 =>
       next_state <= transfer1;
```

```
    when transfer1 =>
       next_state <= transfer2;
    when transfer2 =>
       next_state <= transfer3;
    when transfer3 =>
       next_state <= transfer4;
    when transfer4 =>
       next_state <= write0;
    when write0 =>
       next_state <= write1;
    when write1 =>
       if (Term = '1') then
          next_state <= done;
       else
          next_state <= strobe2;
       end if;
    when strobe2 =>
       next_state <= read_data0;
    when done =>
       next_state <= done;
    when others =>
       next_state <= done;
  end case;
end process;

end state_arch;

CONFIGURATION config_state OF state IS
   FOR state_arch
   END FOR;
END config_state;
```

## D.2 DES

The first file listed below is the VHDL library used by the second VHDL code below which implements the DES function on one FPGA.

des_lib.vhd:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity m2_1 is
port (
d0: in std_logic;
d1: in std_logic;
s0: in std_logic;
o: out std_logic);
end m2_1;

architecture behaviour of m2_1 is
begin
o <= d0 when s0 = '0' else d1;
end behaviour;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity fd is
port (
c: in std_logic;
ce: in std_logic;
d: in std_logic;
q: out std_logic);
end fd;

architecture behaviour of fd is
begin
-- process(c, ce)
-- begin
-- if (rising_edge(c) and ce = '1') then
-- if rising_edge(c) then
q <= d;
-- end if;
-- end process;
end behaviour;
```

des_pc0.vhd:

```
--Name: optimised-des.vhdl
--Author: Chris Eilbeck, chris@yordas.demon.co.uk
--Purpose:Fully pipelined implementation of a DES encryptor to the
--FIPS46-2 standard.
--Performance: Xilinx XC4062XLA-08 - 28MHz max clock rate, 1700 CLBs,
--1800 mbps.
--IP Status:Free use is hereby granted for all civil use including
--personal, educational and commercial use.
--The use of this code for military, diplomatic or governmental purposes
--is specifically forbidden.
--Copyright is retained by the author.
--Vers Info:v0.1 20/1/1999 - tested using supplied VHDL testbench with
--vectors taken from des-linux package.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity s1 is port
(
i : in std_logic_vector(1 to 6);
clk : in std_logic;
ce : in std_logic;
o : out std_logic_vector(1 to 4)
);
end s1;

architecture behaviour of s1 is
component m2_1
port (
d0: in std_logic;
d1: in std_logic;
s0: in std_logic;
o: out std_logic);
end component;
component fd
port (
```

```
c: in std_logic;
ce: in std_logic;
d: in std_logic;
q: out std_logic);
end component;
component rom32x1
generic (
init: std_logic_vector(31 downto 0));
port(
a0: in std_logic;
a1: in std_logic;
a2: in std_logic;
a3: in std_logic;
a4: in std_logic;
o:out std_logic);
end component;


attribute init: string;
attribute init of u1: label is "86e67619";
attribute init of u2: label is "869d497a";
attribute init of u3: label is "497826bd";
attribute init of u4: label is "b0c7871b";
attribute init of u5: label is "609f1f29";
attribute init of u6: label is "27e9d492";
attribute init of u7: label is "6f81b478";
attribute init of u8: label is "917be906";

signal lower1,upper1,lower2,upper2,lower3,upper3,lower4,upper4
: std_logic;
signal d1,d2,d3,d4 : std_logic;

begin


--u1:rom32x1 generic map (init=>16#86e67619# ) port map
(a0=>i(6),a1=>i(5),a2=>i(4),a3=>i(3),a4=>i(2),o=>lower1);
u1: rom32x1 generic map ( init=>"10000110111001100111011000011001" )
port map(a0=>i(6),a1=>i(5),a2=>i(4),a3=>i(3),a4=>i(2),o=>lower1);
--u2: rom32x1 generic map ( init=>16#869d497a# ) port map
( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
u2: rom32x1 generic map ( init=>"10000110100111010100100101111010" )
```

```
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
--u3: rom32x1 generic map ( init=>16#497826bd# )
port map(a0=>i(6),a1=>i(5),a2=>i(4),a3=>i(3),a4=>i(2), o=>lower2);
u3: rom32x1 generic map ( init=>"01001001011110000010011010111101" )
port map(a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
--u4: rom32x1 generic map ( init=>16#b0c7871b# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
u4:rom32x1 generic map (init=>"10110000110001111000011100011011" )
port map(a0=>i(6),a1=>i(5),a2=>i(4),a3=>i(3),a4=>i(2),o=>upper2 );
--u5: rom32x1 generic map ( init=>16#609f1f29# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
u5: rom32x1 generic map ( init=>"01100000100111110001111100101001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
--u6: rom32x1 generic map ( init=>16#27e9d492# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
u6: rom32x1 generic map ( init=>"00100111111010011101010010010010" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
--u7: rom32x1 generic map ( init=>16#6f81b478# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
u7: rom32x1 generic map ( init=>"01101111100000011011010001111000" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
--u8: rom32x1 generic map ( init=>16#917be906# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );
u8: rom32x1 generic map ( init=>"10010001011110111110100100000110" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );

u9: m2_1 port map ( d0=>lower1, d1=>upper1, s0=>i(1), o=>d1 );
u10: m2_1 port map ( d0=>lower2, d1=>upper2, s0=>i(1), o=>d2 );
u11: m2_1 port map ( d0=>lower3, d1=>upper3, s0=>i(1), o=>d3 );
u12: m2_1 port map ( d0=>lower4, d1=>upper4, s0=>i(1), o=>d4 );

u13: fd port map ( c=>clk, ce=>ce, d=>d1, q=>o(1) );
u14: fd port map ( c=>clk, ce=>ce, d=>d2, q=>o(2) );
u15: fd port map ( c=>clk, ce=>ce, d=>d3, q=>o(3) );
u16: fd port map ( c=>clk, ce=>ce, d=>d4, q=>o(4) );

end; -- s1

library ieee;
use ieee.std_logic_1164.all;

entity s2 is port
```

```
(
i : in std_logic_vector(1 to 6);
clk : in std_logic;
ce : in std_logic;
o : out std_logic_vector(1 to 4)
);
end s2;

architecture behaviour of s2 is

attribute init: string;
attribute init of u1: label is "69c3a659";
attribute init of u2: label is "e196196e";
attribute init of u3: label is "9346c3e9";
attribute init of u4: label is "68f93c16";
attribute init of u5: label is "62949fc3";
attribute init of u6: label is "746a8b74";
attribute init of u7: label is "b865168f";
attribute init of u8: label is "cd235ad2";

component m2_1
port (
d0: in std_logic;
d1: in std_logic;
s0: in std_logic;
o: out std_logic);
end component;

component fd
port (
c: in std_logic;
ce: in std_logic;
d: in std_logic;
q: out std_logic);
end component;

component rom32x1
generic (
init: std_logic_vector(31 downto 0));
port(
a0: in std_logic;
a1: in std_logic;
```

```
a2: in std_logic;
a3: in std_logic;
a4: in std_logic;
o:out std_logic);
end component;
signal lower1,upper1,lower2,upper2,lower3,upper3,lower4,
upper4 : std_logic;
signal d1,d2,d3,d4 : std_logic;

begin

--u1: rom32x1 generic map ( init=>16#69c3a659# )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
u1: rom32x1 generic map ( init=>"01101001110000111010011001011001" )
port map( a0=>i(6),a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
--u2: rom32x1 generic map ( init=>16#e196196e# )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
u2: rom32x1 generic map ( init=>"11100001100101100001100101101110" )
port map( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
--u3: rom32x1 generic map ( init=>16#9346c3e9# )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
u3: rom32x1 generic map ( init=>"10010011010001101100001111101001" )
port map( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
--u4: rom32x1 generic map ( init=>16#68f93c16# )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
u4: rom32x1 generic map ( init=>"01101000111110010011110000010110" )
port map( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
--u5: rom32x1 generic map ( init=>16#62949fc3# )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
u5: rom32x1 generic map ( init=>"01100010100101001001111111000011" )
port map( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
--u6: rom32x1 generic map ( init=>16#746a8b74# )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
u6: rom32x1 generic map ( init=>"01110100011010101000101101110100" )
port map( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
--u7: rom32x1 generic map ( init=>16#b865168f# )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
u7: rom32x1 generic map ( init=>"10111000011001010001011010001111" )
port map( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
--u8: rom32x1 generic map ( init=>16#cd235ad2# )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );
u8: rom32x1 generic map ( init=>"11001101001000110101101011010010" )
```

```
port map( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );

u9: m2_1 port map ( d0=>lower1, d1=>upper1, s0=>i(1), o=>d1 );
u10: m2_1 port map ( d0=>lower2, d1=>upper2, s0=>i(1), o=>d2 );
u11: m2_1 port map ( d0=>lower3, d1=>upper3, s0=>i(1), o=>d3 );
u12: m2_1 port map ( d0=>lower4, d1=>upper4, s0=>i(1), o=>d4 );

u13: fd port map ( c=>clk, ce=>ce, d=>d1, q=>o(1) );
u14: fd port map ( c=>clk, ce=>ce, d=>d2, q=>o(2) );
u15: fd port map ( c=>clk, ce=>ce, d=>d3, q=>o(3) );
u16: fd port map ( c=>clk, ce=>ce, d=>d4, q=>o(4) );

end; -- s2

library ieee;
use ieee.std_logic_1164.all;

entity s3 is port
(
i : in std_logic_vector(1 to 6);
clk : in std_logic;
ce : in std_logic;
o : out std_logic_vector(1 to 4)
);
end s3;

architecture behaviour of s3 is

attribute init: string;
attribute init of u1: label is "6b9c90d3";
attribute init of u2: label is "96692d69";
attribute init of u3: label is "26f4794a";
attribute init of u4: label is "d96a8635";
attribute init of u5: label is "39c2b749";
attribute init of u6: label is "76b9960c";
attribute init of u7: label is "a965569a";
attribute init of u8: label is "4b8d9c63";

component m2_1
port (
d0: in std_logic;
d1: in std_logic;
```

```
s0: in std_logic;
o: out std_logic);
end component;
component fd
port (
c: in std_logic;
ce: in std_logic;
d: in std_logic;
q: out std_logic);
end component;
component rom32x1
generic (
init: std_logic_vector(31 downto 0));
port(
a0: in std_logic;
a1: in std_logic;
a2: in std_logic;
a3: in std_logic;
a4: in std_logic;
o:out std_logic);
end component;
signal lower1,upper1,lower2,upper2,lower3,upper3,lower4,
upper4 : std_logic;
signal d1,d2,d3,d4 : std_logic;

begin

--u1: rom32x1 generic map ( init=>16#6b9c90d3# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
u1: rom32x1 generic map ( init=>"01101011100111001001000011010011" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
--u2: rom32x1 generic map ( init=>16#96692d69# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
u2: rom32x1 generic map ( init=>"10010110011010010010110101101001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
--u3: rom32x1 generic map ( init=>16#26f4794a# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
u3: rom32x1 generic map ( init=>"00100110111101000111100101001010" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
--u4: rom32x1 generic map ( init=>16#d96a8635# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
u4: rom32x1 generic map ( init=>"11011001011010101000011000110101" )
```

```
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
--u5: rom32x1 generic map ( init=>16#39c2b749# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
u5: rom32x1 generic map ( init=>"00111001110000101011011101001001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
--u6: rom32x1 generic map ( init=>16#76b9960c# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
u6: rom32x1 generic map ( init=>"01110110101110011001011000001100" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
--u7: rom32x1 generic map ( init=>16#a965569a# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
u7: rom32x1 generic map ( init=>"10101001011001010101011010011010" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
--u8: rom32x1 generic map ( init=>16#4b8d9c63# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );
u8: rom32x1 generic map ( init=>"01001011100011011001110001100011" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );

u9: m2_1 port map ( d0=>lower1, d1=>upper1, s0=>i(1), o=>d1 );
u10: m2_1 port map ( d0=>lower2, d1=>upper2, s0=>i(1), o=>d2 );
u11: m2_1 port map ( d0=>lower3, d1=>upper3, s0=>i(1), o=>d3 );
u12: m2_1 port map ( d0=>lower4, d1=>upper4, s0=>i(1), o=>d4 );

u13: fd port map ( c=>clk, ce=>ce, d=>d1, q=>o(1) );
u14: fd port map ( c=>clk, ce=>ce, d=>d2, q=>o(2) );
u15: fd port map ( c=>clk, ce=>ce, d=>d3, q=>o(3) );
u16: fd port map ( c=>clk, ce=>ce, d=>d4, q=>o(4) );

end; -- s3

library ieee;
use ieee.std_logic_1164.all;

entity s4 is port
(
i : in std_logic_vector(1 to 6);
clk : in std_logic;
ce : in std_logic;
o : out std_logic_vector(1 to 4)
);
end s4;
```

```
architecture behaviour of s4 is

attribute init: string;
attribute init of u1: label is "ed90583e";
attribute init of u2: label is "92c3e719";
attribute init of u3: label is "74ca0e97";
attribute init of u4: label is "cb69718c";
attribute init of u5: label is "692cce71";
attribute init of u6: label is "acd1168f";
attribute init of u7: label is "c34998e7";
attribute init of u8: label is "09b77c1a";

component rom32x1
generic (
init: std_logic_vector(31 downto 0));
port(
a0: in std_logic;
a1: in std_logic;
a2: in std_logic;
a3: in std_logic;
a4: in std_logic;
o:out std_logic);
end component;
component m2_1
port (
d0: in std_logic;
d1: in std_logic;
s0: in std_logic;
o: out std_logic);
end component;
component fd
port (
c: in std_logic;
ce: in std_logic;
d: in std_logic;
q: out std_logic);
end component;
signal lower1,upper1,lower2,upper2,lower3,upper3,lower4,
upper4 : std_logic;
signal d1,d2,d3,d4 : std_logic;

begin
```

```
--u1: rom32x1 generic map ( init=>16#ed90583e# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
u1: rom32x1 generic map ( init=>"11101101100100000101100000111110" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
--u2: rom32x1 generic map ( init=>16#92c3e719# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
u2: rom32x1 generic map ( init=>"10010010110000111110011100011001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
--u3: rom32x1 generic map ( init=>16#74ca0e97# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
u3: rom32x1 generic map ( init=>"01110100110010100000111010010111" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
--u4: rom32x1 generic map ( init=>16#cb69718c# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
u4: rom32x1 generic map ( init=>"11001011011010010111000110001100" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
--u5: rom32x1 generic map ( init=>16#692cce71# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
u5: rom32x1 generic map ( init=>"01101001001011001100111001110001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
--u6: rom32x1 generic map ( init=>16#acd1168f# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
u6: rom32x1 generic map ( init=>"10101100110100010001011010001111" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
--u7: rom32x1 generic map ( init=>16#c34998e7# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
u7: rom32x1 generic map ( init=>"11000011010010011001100011100111" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
--u8: rom32x1 generic map ( init=>16#09b77c1a# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );
u8: rom32x1 generic map ( init=>"00001001101101110111110000011010" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );

u9: m2_1 port map ( d0=>lower1, d1=>upper1, s0=>i(1), o=>d1 );
u10: m2_1 port map ( d0=>lower2, d1=>upper2, s0=>i(1), o=>d2 );
u11: m2_1 port map ( d0=>lower3, d1=>upper3, s0=>i(1), o=>d3 );
u12: m2_1 port map ( d0=>lower4, d1=>upper4, s0=>i(1), o=>d4 );

u13: fd port map ( c=>clk, ce=>ce, d=>d1, q=>o(1) );
u14: fd port map ( c=>clk, ce=>ce, d=>d2, q=>o(2) );
u15: fd port map ( c=>clk, ce=>ce, d=>d3, q=>o(3) );
```

```
u16: fd port map ( c=>clk, ce=>ce, d=>d4, q=>o(4) );

end; -- s4

library ieee;
use ieee.std_logic_1164.all;

entity s5 is port
(
i : in std_logic_vector(1 to 6);
clk : in std_logic;
ce : in std_logic;
o : out std_logic_vector(1 to 4)
);
end s5;

architecture behaviour of s5 is

attribute init: string;
attribute init of u1: label is "79e1348e";
attribute init of u2: label is "429dcd6a";
attribute init of u3: label is "91666b96";
attribute init of u4: label is "695b9ca1";
attribute init of u5: label is "92f05d2b";
attribute init of u6: label is "c70b39c6";
attribute init of u7: label is "4b76b948";
attribute init of u8: label is "a4cd96d2";

component rom32x1
generic (
init: std_logic_vector(31 downto 0));
port(
a0: in std_logic;
a1: in std_logic;
a2: in std_logic;
a3: in std_logic;
a4: in std_logic;
o:out std_logic);
end component;
component m2_1
port (
d0: in std_logic;
```

```
d1: in std_logic;
s0: in std_logic;
o: out std_logic);
end component;
component fd
port (
c: in std_logic;
ce: in std_logic;
d: in std_logic;
q: out std_logic);
end component;
signal lower1,upper1,lower2,upper2,lower3,upper3,lower4,
upper4 : std_logic;
signal d1,d2,d3,d4 : std_logic;

begin

--u1: rom32x1 generic map ( init=>16#79e1348e# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
u1: rom32x1 generic map ( init=>"01111001111000010011010010001110" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
--u2: rom32x1 generic map ( init=>16#429dcd6a# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
u2: rom32x1 generic map ( init=>"01000010100111011100110101101010" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
--u3: rom32x1 generic map ( init=>16#91666b96# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
u3: rom32x1 generic map ( init=>"10010001011001100110101110010110" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
--u4: rom32x1 generic map ( init=>16#695b9ca1# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
u4: rom32x1 generic map ( init=>"01101001010110111001110010100001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
--u5: rom32x1 generic map ( init=>16#92f05d2b# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
u5: rom32x1 generic map ( init=>"10010010111100000101110100101011" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
--u6: rom32x1 generic map ( init=>16#c70b39c6# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
u6: rom32x1 generic map ( init=>"11000111000010110011100111000110" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
--u7: rom32x1 generic map ( init=>16#4b76b948# ) port map ( a0=>i(6),
```

```
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
u7: rom32x1 generic map ( init=>"01001011011101101011100101001000" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
--u8: rom32x1 generic map ( init=>16#a4cd96d2# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );
u8: rom32x1 generic map ( init=>"10100100110011011001011011010010" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );

u9: m2_1 port map ( d0=>lower1, d1=>upper1, s0=>i(1), o=>d1 );
u10: m2_1 port map ( d0=>lower2, d1=>upper2, s0=>i(1), o=>d2 );
u11: m2_1 port map ( d0=>lower3, d1=>upper3, s0=>i(1), o=>d3 );
u12: m2_1 port map ( d0=>lower4, d1=>upper4, s0=>i(1), o=>d4 );

u13: fd port map ( c=>clk, ce=>ce, d=>d1, q=>o(1) );
u14: fd port map ( c=>clk, ce=>ce, d=>d2, q=>o(2) );
u15: fd port map ( c=>clk, ce=>ce, d=>d3, q=>o(3) );
u16: fd port map ( c=>clk, ce=>ce, d=>d4, q=>o(4) );

end; -- s5

library ieee;
use ieee.std_logic_1164.all;

entity s6 is port
(
i : in std_logic_vector(1 to 6);
clk : in std_logic;
ce : in std_logic;
o : out std_logic_vector(1 to 4)
);
end s6;

architecture behaviour of s6 is

attribute init: string;
attribute init of u1: label is "c9a4695b";
attribute init of u2: label is "b44ab695";
attribute init of u3: label is "15e69a69";
attribute init of u4: label is "c69938d6";
attribute init of u5: label is "6d9216da";
attribute init of u6: label is "52cbe13c";
attribute init of u7: label is "7c3ca34c";
```

```
attribute init of u8: label is "95a36a59";

component rom32x1
generic (
init: std_logic_vector(31 downto 0));
port(
a0: in std_logic;
a1: in std_logic;
a2: in std_logic;
a3: in std_logic;
a4: in std_logic;
o:out std_logic);
end component;
component m2_1
port (
d0: in std_logic;
d1: in std_logic;
s0: in std_logic;
o: out std_logic);
end component;
component fd
port (
c: in std_logic;
ce: in std_logic;
d: in std_logic;
q: out std_logic);
end component;
signal lower1,upper1,lower2,upper2,lower3,upper3,lower4,
upper4 : std_logic;
signal d1,d2,d3,d4 : std_logic;

begin

--u1: rom32x1 generic map ( init=>16#c9a4695b# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
u1: rom32x1 generic map ( init=>"11001001101001000110100101011011" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
--u2: rom32x1 generic map ( init=>16#b44ab695# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
u2: rom32x1 generic map ( init=>"10110100010010101011011010010101" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
--u3: rom32x1 generic map ( init=>16#15e69a69# ) port map ( a0=>i(6),
```

```
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
u3: rom32x1 generic map ( init=>"00010101111001101001101001101001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
--u4: rom32x1 generic map ( init=>16#c69938d6# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
u4: rom32x1 generic map ( init=>"11000110100110010011100011010110" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
--u5: rom32x1 generic map ( init=>16#6d9216da# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
u5: rom32x1 generic map ( init=>"01101101100100100001011011011010" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
--u6: rom32x1 generic map ( init=>16#52cbe13c# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
u6: rom32x1 generic map ( init=>"01010010110010111110000100111100" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
--u7: rom32x1 generic map ( init=>16#7c3ca34c# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
u7: rom32x1 generic map ( init=>"01111100001111001010001101001100" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
--u8: rom32x1 generic map ( init=>16#95a36a59# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );
u8: rom32x1 generic map ( init=>"10010101101000110110101001011001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );

u9: m2_1 port map ( d0=>lower1, d1=>upper1, s0=>i(1), o=>d1 );
u10: m2_1 port map ( d0=>lower2, d1=>upper2, s0=>i(1), o=>d2 );
u11: m2_1 port map ( d0=>lower3, d1=>upper3, s0=>i(1), o=>d3 );
u12: m2_1 port map ( d0=>lower4, d1=>upper4, s0=>i(1), o=>d4 );

u13: fd port map ( c=>clk, ce=>ce, d=>d1, q=>o(1) );
u14: fd port map ( c=>clk, ce=>ce, d=>d2, q=>o(2) );
u15: fd port map ( c=>clk, ce=>ce, d=>d3, q=>o(3) );
u16: fd port map ( c=>clk, ce=>ce, d=>d4, q=>o(4) );

end; -- s6

library ieee;
use ieee.std_logic_1164.all;

entity s7 is port
(
i : in std_logic_vector(1 to 6);
```

```
clk : in std_logic;
ce : in std_logic;
o : out std_logic_vector(1 to 4)
);
end s7;

architecture behaviour of s7 is

attribute init: string;
attribute init of u1: label is "2c96d966";
attribute init of u2: label is "92c761f8";
attribute init of u3: label is "99e643c3";
attribute init of u4: label is "869cd966";
attribute init of u5: label is "9e4b81f4";
attribute init of u6: label is "6a95f41a";
attribute init of u7: label is "497969a6";
attribute init of u8: label is "348e9679";

component m2_1
port (
d0: in std_logic;
d1: in std_logic;
s0: in std_logic;
o: out std_logic);
end component;
component rom32x1
generic (
init: std_logic_vector(31 downto 0));
port(
a0: in std_logic;
a1: in std_logic;
a2: in std_logic;
a3: in std_logic;
a4: in std_logic;
o:out std_logic);
end component;
component fd
port (
c: in std_logic;
ce: in std_logic;
d: in std_logic;
q: out std_logic);
```

```
end component;
signal lower1,upper1,lower2,upper2,lower3,upper3,lower4,
upper4 : std_logic;
signal d1,d2,d3,d4 : std_logic;

begin

--u1: rom32x1 generic map ( init=>16#2c96d966# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
u1: rom32x1 generic map ( init=>"00101100100101101101100101100110" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
--u2: rom32x1 generic map ( init=>16#92c761f8# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
u2: rom32x1 generic map ( init=>"10010010110001110110000111111000" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
--u3: rom32x1 generic map ( init=>16#99e643c3# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
u3: rom32x1 generic map ( init=>"10011001111001100100001111000011" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
--u4: rom32x1 generic map ( init=>16#869cd966# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
u4: rom32x1 generic map ( init=>"10000110100111001101100101100110" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
--u5: rom32x1 generic map ( init=>16#9e4b81f4# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
u5: rom32x1 generic map ( init=>"10011110010010111000000111110100" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
--u6: rom32x1 generic map ( init=>16#6a95f41a# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
u6: rom32x1 generic map ( init=>"01101010100101011111010000011010" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
--u7: rom32x1 generic map ( init=>16#497969a6# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
u7: rom32x1 generic map ( init=>"01001001011110010110100110100110" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
--u8: rom32x1 generic map ( init=>16#348e9679# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );
u8: rom32x1 generic map ( init=>"00110100100011101001011001111001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );

u9: m2_1 port map ( d0=>lower1, d1=>upper1, s0=>i(1), o=>d1 );
u10: m2_1 port map ( d0=>lower2, d1=>upper2, s0=>i(1), o=>d2 );
```

```
u11: m2_1 port map ( d0=>lower3, d1=>upper3, s0=>i(1), o=>d3 );
u12: m2_1 port map ( d0=>lower4, d1=>upper4, s0=>i(1), o=>d4 );

u13: fd port map ( c=>clk, ce=>ce, d=>d1, q=>o(1) );
u14: fd port map ( c=>clk, ce=>ce, d=>d2, q=>o(2) );
u15: fd port map ( c=>clk, ce=>ce, d=>d3, q=>o(3) );
u16: fd port map ( c=>clk, ce=>ce, d=>d4, q=>o(4) );

end; -- s7

library ieee;
use ieee.std_logic_1164.all;

entity s8 is port
(
i : in std_logic_vector(1 to 6);
clk : in std_logic;
ce : in std_logic;
o : out std_logic_vector(1 to 4)
);
end s8;

architecture behaviour of s8 is

attribute init: string;
attribute init of u1: label is "38C716B9";
attribute init of u2: label is "C17ABD24";
attribute init of u3: label is "596AA569";
attribute init of u4: label is "394e96b1";
attribute init of u5: label is "C8F13F0c";
attribute init of u6: label is "A71658A7";
attribute init of u7: label is "619C7C2b";
attribute init of u8: label is "9F6281Cd";

component m2_1
port (
d0: in std_logic;
d1: in std_logic;
s0: in std_logic;
o: out std_logic);
end component;
component fd
```

```
port (
c: in std_logic;
ce: in std_logic;
d: in std_logic;
q: out std_logic);
end component;
signal lower1,upper1,lower2,upper2,lower3,upper3,lower4,
upper4 : std_logic;
signal d1,d2,d3,d4 : std_logic;

component rom32x1
generic (
init: std_logic_vector(31 downto 0));
port(
a0: in std_logic;
a1: in std_logic;
a2: in std_logic;
a3: in std_logic;
a4: in std_logic;
o:out std_logic);
end component;
begin

--u1: rom32x1 generic map ( init=>16#38C716B9# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
u1: rom32x1 generic map ( init=>"00111000110001110001011010111001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower1 );
--u2: rom32x1 generic map ( init=>16#C17ABD24# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
u2: rom32x1 generic map ( init=>"11000001011110101011110100100100" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper1 );
--u3: rom32x1 generic map ( init=>16#596AA569# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
u3: rom32x1 generic map ( init=>"01011001011010101010010101101001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower2 );
--u4: rom32x1 generic map ( init=>16#394e96b1# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
u4: rom32x1 generic map ( init=>"00111001010011101001011010110001" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper2 );
--u5: rom32x1 generic map ( init=>16#C8F13F0c# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
u5: rom32x1 generic map ( init=>"11001000111100010011111100001100" )
```

```
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower3 );
--u6: rom32x1 generic map ( init=>16#A71658A7# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
u6: rom32x1 generic map ( init=>"10100111000101100101100010100111" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper3 );
--u7: rom32x1 generic map ( init=>16#619C7C2b# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
u7: rom32x1 generic map ( init=>"01100001100111000111110000101011" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>lower4 );
--u8: rom32x1 generic map ( init=>16#9F6281Cd# ) port map ( a0=>i(6),
a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );
u8: rom32x1 generic map ( init=>"10011111011000101000000111001101" )
port map ( a0=>i(6), a1=>i(5), a2=>i(4), a3=>i(3), a4=>i(2), o=>upper4 );

u9: m2_1 port map ( d0=>lower1, d1=>upper1, s0=>i(1), o=>d1 );
u10: m2_1 port map ( d0=>lower2, d1=>upper2, s0=>i(1), o=>d2 );
u11: m2_1 port map ( d0=>lower3, d1=>upper3, s0=>i(1), o=>d3 );
u12: m2_1 port map ( d0=>lower4, d1=>upper4, s0=>i(1), o=>d4 );

u13: fd port map ( c=>clk, ce=>ce, d=>d1, q=>o(1) );
u14: fd port map ( c=>clk, ce=>ce, d=>d2, q=>o(2) );
u15: fd port map ( c=>clk, ce=>ce, d=>d3, q=>o(3) );
u16: fd port map ( c=>clk, ce=>ce, d=>d4, q=>o(4) );

end; -- s8

library ieee;
use ieee.std_logic_1164.all;

entity pc1 is port
(
key : in std_logic_vector(1 TO 64);
c0x,d0x : out std_logic_vector(1 TO 28)
);
end pc1;

architecture behaviour of pc1 is
signal XX : std_logic_vector(1 to 56);
begin
XX(1)<=key(57); XX(2)<=key(49); XX(3)<=key(41); XX(4)<=key(33);
XX(5)<=key(25); XX(6)<=key(17); XX(7)<=key(9);
XX(8)<=key(1); XX(9)<=key(58); XX(10)<=key(50); XX(11)<=key(42);
```

```
XX(12)<=key(34); XX(13)<=key(26); XX(14)<=key(18);
XX(15)<=key(10); XX(16)<=key(2); XX(17)<=key(59); XX(18)<=key(51);
XX(19)<=key(43); XX(20)<=key(35); XX(21)<=key(27);
XX(22)<=key(19); XX(23)<=key(11); XX(24)<=key(3); XX(25)<=key(60);
XX(26)<=key(52); XX(27)<=key(44); XX(28)<=key(36);
XX(29)<=key(63); XX(30)<=key(55); XX(31)<=key(47); XX(32)<=key(39);
XX(33)<=key(31); XX(34)<=key(23); XX(35)<=key(15);
XX(36)<=key(7); XX(37)<=key(62); XX(38)<=key(54); XX(39)<=key(46);
XX(40)<=key(38); XX(41)<=key(30); XX(42)<=key(22);
XX(43)<=key(14); XX(44)<=key(6); XX(45)<=key(61); XX(46)<=key(53);
XX(47)<=key(45); XX(48)<=key(37); XX(49)<=key(29);
XX(50)<=key(21); XX(51)<=key(13); XX(52)<=key(5); XX(53)<=key(28);
XX(54)<=key(20); XX(55)<=key(12); XX(56)<=key(4);

c0x<=XX(1 to 28); d0x<=XX(29 to 56);
end behaviour;

library ieee;
use ieee.std_logic_1164.all;

entity pc2 is port
(
c,d : in std_logic_vector(1 TO 28);
k : out std_logic_vector(1 TO 48)
);
end pc2;

architecture behaviour of pc2 is
signal YY : std_logic_vector(1 to 56);
begin
YY(1 to 28)<=c; YY(29 to 56)<=d;

k(1)<=YY(14); k(2)<=YY(17); k(3)<=YY(11); k(4)<=YY(24); k(5)<=YY(1);
k(6)<=YY(5);  k(7)<=YY(3); k(8)<=YY(28); k(9)<=YY(15); k(10)<=YY(6);
k(11)<=YY(21); k(12)<=YY(10);
k(13)<=YY(23); k(14)<=YY(19); k(15)<=YY(12); k(16)<=YY(4); k(17)<=YY(26);
k(18)<=YY(8);
k(19)<=YY(16); k(20)<=YY(7); k(21)<=YY(27); k(22)<=YY(20); k(23)<=YY(13);
k(24)<=YY(2);
k(25)<=YY(41); k(26)<=YY(52); k(27)<=YY(31); k(28)<=YY(37); k(29)<=YY(47);
k(30)<=YY(55);
k(31)<=YY(30); k(32)<=YY(40); k(33)<=YY(51); k(34)<=YY(45); k(35)<=YY(33);
```

```
k(36)<=YY(48);
k(37)<=YY(44); k(38)<=YY(49); k(39)<=YY(39); k(40)<=YY(56); k(41)<=YY(34);
k(42)<=YY(53);
k(43)<=YY(46); k(44)<=YY(42); k(45)<=YY(50); k(46)<=YY(36); k(47)<=YY(29);
k(48)<=YY(32);
end behaviour;
```

## Vita

Jeanne Marie Lehrter was born in Nuremburg, Germany in 1975. Her father was in the Army and as a result, her family moved many times. In 1987, Jeanne's father retired from the military and the family moved to Florence, Alabama. Jeanne graduated high school in 1993 from the Alabama School of Math and Science in Mobile, AL. She continued school and received her undergraduate degree in Computer Engineering with a minor in French from Auburn University, Auburn, AL. She will receive a Master's degree in Computer Science from the University of Tennessee in Knoxville in May, 2002.