THE USE OF VHDL IN COMPUTER-AIDED SUPPORT OF LIFE-CYCLE COMPLETE PRODUCT DESIGN

by

Rhett D. Hudson

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE
in
Electrical Engineering

APPROVED:

[Signature]
Dr. Scott F. Midkiff, Chairperson

[Signature]
Dr. Wolter J. Fabrycky

[Signature]
Dr. Walling Cyre

June, 1994
Blacksburgh, Virginia
THE USE OF VHDL IN COMPUTER-AIDED SUPPORT OF LIFE-CYCLE COMPLETE PRODUCT DESIGN

by

Rhett Hudson

Dr. Scott F. Midkiff

Bradley Department of Electrical Engineering

(ABSTRACT)

Successful competition in the computer systems industry depends on a firm's ability to bring profitable products to market. The success of a product is measured by its future worth to the company. Life-cycle complete design attempts to engineer products that provide maximum future worth. Many components contribute to the overall cost of developing a product. Designing merely to reduce the cost of the components that make up the system is insufficient.

A product must be engineered in a manner that addresses all pertinent issues over its complete life cycle. This research examines the use of the VHSIC Hardware Description Language as a computer-aided engineering tool for life-cycle complete engineering. VHDL is traditionally used to model the functional behavior of digital systems. This thesis provides an overview of a life-cycle complete design process and describes the use of VHDL to support that process. A case study is presented to illustrate the use of VHDL for life-cycle complete modeling.
Dedication

This work is dedicated to Wayne R. Hudson and Bonnie O. Hudson, my parents. I am lucky to have been born to two such exceptional people. They have given me a love of science and the creativity to explore its bounds.
Acknowledgements

First and foremost I would like to thank my advisor Dr. Scott F. Midkiff. Dr. Midkiff has provided guidance and support for my graduate career from the very beginning. His dedication to quality and completeness have provided me with an excellent role model. If I can live up to his example I will count myself lucky.

I would like to thank Dr. Fabrycky for his support in hiring me to work on the PP/E1 DF, and for providing me with a broad insight into life-cycle complete engineering. I would like to thank Dr. Cyre for taking the time to be on my committee and for teaching me digital design, networks and conceptual knowledge representation theory.

I would like to thank my comrades in arms: Jim Peterson, who taught me about loose ends; Rob Osberg, who taught me when to hold my touange; Ray Bittner, who taught me not to compare myself to others; and Michael Hopson, taught me how to fight chaos.

Finally I would like to thank Whitney Emch, who has so far given me the best year of my life, and has taught me so many things about how to live that I could not list them here if I used all the pages between these two covers.
1. Introduction ................................................................. 1
  1.1. The Life Cycle ......................................................... 1
  1.2. Life-Cycle Complete Design ....................................... 2
  1.3. VHDL as a CAE Tool for Life-Cycle Complete Design ........ 3

2. The Design Process ..................................................... 5
  2.1. The System Life Cycle Process .................................... 5
      2.1.1. Definition of Need ........................................... 7
      2.1.2. Conceptual Design .......................................... 10
          2.1.2.1. Feasibility Studies .................................... 11
          2.1.2.2. System Operational Requirements .................... 12
          2.1.2.3. System Maintenance Concept ......................... 13
          2.1.2.4. Preliminary Systems Analysis ....................... 14
          2.1.2.5. Advance System Planning ............................ 15
          2.1.2.6. Conceptual Design Review ............................ 16
      2.1.3. Preliminary Design .......................................... 16
          2.1.3.1. System Functional Analysis .......................... 17
          2.1.3.2. Preliminary Synthesis and Allocation of Design Criteria .......................... 17
          2.1.3.3. System Optimization .................................. 19
          2.1.3.4. System Synthesis and Definition ................... 20
          2.1.3.5. Preliminary Design Review ........................... 20
      2.1.4. Detail Design and Development ............................ 21
          2.1.4.1. Feedback from Detailed Design ...................... 21
          2.1.4.2. The Prototype ........................................ 23
      2.1.5. Production and Construction .............................. 23
      2.1.6. Utilization and Support ................................... 24
      2.1.7. Phaseout and Disposal ................................... 25
  2.2. Inter-Product Competition for Scarce Resources .............. 25
      2.2.1. Product Lifetime Resource Requirements ................. 26
          2.2.1.1. Increasingly Accurate Predictions .................. 26
          2.2.1.2. Resource Requirement Functions .................... 27
          2.2.1.3. Organizational Constraint Functions ............... 33
3. Using VHDL to Support the Design Process .......... 38
   3.1. Parts Cost ........................................... 39
       3.1.1. Parts Cost Resolution Function ................. 39
       3.1.2. Parts Cost Output ................................ 40
   3.2. Mean Time Before Failure ............................. 40
       3.2.1. MTBF Resolution Function ....................... 40
       3.2.2. Effectiveness Measures Output .................. 41
           3.2.2.1. Reliability ................................... 41
           3.2.2.2. Availability ................................. 42
           3.2.2.3. Maintainability ............................. 43
       3.2.3. Real Valued Functions in VHDL .................. 43
   3.3. Resource Requirement Functions ...................... 44
       3.3.1. Resource Requirement Resolution Function ....... 45
           3.3.1.1. Resource Requirement Function Data Type .... 45
           3.3.1.2. Aggregate Initialization of Requirement Functions ... 46
           3.3.1.3. Checking Requirements Versus Limits .......... 47
   3.4. Design Criteria in the Hierarchy ..................... 48
       3.4.1. Connecting the Signals ........................... 48
       3.4.2. Checking the Results ............................ 50
   3.5. Organizational Information Flow ...................... 50
   3.6. Chapter Summary .................................. 53

4. VHDL in the Design Process: A Case Study .............. 54
   4.1. Definition of Need ................................ 54
       4.1.1. Defining Boundaries ............................ 56
           4.1.1.1. Cost and Reliability ........................ 56
           4.1.1.2. Time to Market ............................. 57
           4.1.1.3. Resource Requirements ...................... 58
## A. VHDL Packages for Life-Cycle Modeling

- A.1 Resource Requirements ........................................ 110
- A.2 Reliability ...................................................... 113
- A.3 Parts Costs .................................................... 115
- A.4 Real Valued Functions ........................................ 116

## B. VHDL Model of Definition of Need

- B.1 LAN_PRODUCT Model ........................................ 119
- B.2 Configuration for LAN_PRODUCT .......................... 120
- B.3 Testbench for LAN_PRODUCT ............................... 121

## C. VHDL Model of Preliminary Design

- C.1 Bridge Model .................................................. 122
- C.2 Bridge Configuration for Preliminary Design .......... 124
- C.3 Testbench for Preliminary Design Model ............... 129
- C.4 Microprocessor Model ....................................... 130
- C.5 Ethernet Connection Model ................................. 131
- C.6 RAM Model .................................................... 133
- C.7 ROM Model .................................................... 134
- C.8 Manual Model ................................................ 135
- C.9 Mounting Model .............................................. 137

## D. VHDL Model of Detailed Design

- D.1 Bridge Model .................................................. 139
- D.2 Bridge Configuration for Preliminary Design .......... 142
- D.3 Testbench for Preliminary Design Model ............... 148
- D.4 Microprocessor Model ....................................... 149
- D.5 Ethernet Connection Model ................................. 150
- D.6 Etherchip Model .............................................. 151
- D.7 RAM Model .................................................... 152
- D.8 DATASPACE Model .......................................... 153
- D.9 Memchips Model ............................................. 155
- D.10 Search Model ............................................... 156
- D.11 Manuals Model .............................................. 157
- D.12 Mounting Model ............................................ 158
List of Figures

1-1. Design, manufacturing, and support life cycles........................................... 2

2-1. The life-cycle process.................................................................................. 6

2-2. A sample product life cycle........................................................................ 8

2-3. Conceptual design activities................................................................. 11

2-4. Hierarchical cost and reliability breakdowns........................................ 19

2-5. Concurrent design issues........................................................................ 22

2-6. Resource requirement functions......................................................... 30

2-7. A sample budget requirement curve................................................... 32

2-8. Functions used in sample budget curve............................................. 32

3-1. Organizational information flows........................................................ 52

4-1. Different LAN environments............................................................... 55

4-2. Life-cycle phases of LAN product..................................................... 58

4-3. Budget requirements predicted during definition of need phase.......... 66

4-4. Revenue requirements predicted during definition of need phase...... 67

4-5. Staff requirements predicted during definition of need phase.......... 67

4-6. Sources of bridge congestion.............................................................. 77
4-7. Functional flow diagram of normal bridge operation ............................................. 81
4-8. Initial block diagram of bridge ............................................................................. 84
4-9. Allocation of parts cost ....................................................................................... 85
4-10. Allocation of reliability ...................................................................................... 87
4-11. Budget requirements for preliminary design phase .......................................... 91
4-12. Revenue requirements for preliminary design phase ....................................... 91
4-13. Staff requirements for preliminary design phase .............................................. 92
4-14. Final bridge block diagram .............................................................................. 95
4-15. Budget requirements for detailed design phase ................................................ 99
4-16. Revenue requirements for detailed design phase ............................................. 100
4-17. Staff requirements for detailed design phase .................................................. 100
1. Introduction

Successful competition in the computer systems industry depends on a firm's expertise in bringing profitable products to market. Profitability depends on providing the customer superior performance for less cost. Superior performance stems from reliable, well-conceived designs, while low cost depends on a short and efficient design process. These two issues motivate life-cycle complete design.

1.1 The Life Cycle

A product exists across a span of time called its life cycle. The life cycle begins when someone perceives a need for a new product in the marketplace. If corporate executives decide that the need really exists and that the firm has an interest in fulfilling that need, they designate engineers to design the product. When the engineers have finished the design and tested it satisfactorily, it enters production. Once it enters the marketplace, consumers can purchase the product and the product provides service to the consumer. As time passes, the product becomes obsolete and the need for the product diminishes. The company removes the product from the market, phases out customer support, and fulfills any responsibility it has for disposing of the physical product itself. These are phases in a product's complete life cycle.

Products require a manufacturing process for production. Once produced, products require support for the customer. The development of a manufacturing process is a product's manufacturing life cycle. The development of a product's customer support
service is its support life cycle. Figure 1-1 shows that these cycles occur simultaneously with the design life cycle.

![Diagram of life cycles](image)

**Figure 1-1.** Design, manufacturing, and support life cycles [BlaF90].

### 1.2 Life-Cycle Complete Design

The success of a product is measured by its future worth to the company. The future worth of a product is calculated by summing all the future equivalent revenues derived from the product and subtracting all the future equivalent costs incurred by the product. Future equivalent revenues and costs are computed by adjusting for the time value of money. Successful products have maximized revenues and minimized costs. Life-cycle complete design is a design philosophy that attempts to engineer products with this in mind.
Many components contribute to the overall cost of developing a product. A partial list includes: design costs, prototype costs, manufacturing costs, warranty repair costs, and staff costs for marketing and customer support. All of these factors must be considered when designing products. Designing merely to reduce the cost of the components that make up the system is insufficient. A life-cycle complete product must be engineered in a manner that addresses all of these issues.

A general framework for the design process can be used to address these issues. This framework divides the life cycle of a product into life-cycle phases, during which certain kinds of design activities occur. At the end of each of these life-cycle phases, a review process occurs before the product may enter the next life-cycle phase. These review processes are referred to as life-cycle gates. Chapter 2 reviews a specific life-cycle design process by Blanchard and Fabrycky [BlaF90].

1.3 VHDL as a CAE Tool for Life-Cycle Complete Design

Life-cycle complete design uses all available information to make decisions in product development. Since "all available information" could easily become intractable to a human engineer, computer-aided engineering (CAE) tools are an important part of the process.

This research specifically examines the use of the VHSIC Hardware Description Language (VHDL) as a CAE tool for life-cycle complete engineering. VHDL is
traditionally used to model the functional behavior of digital systems. It would be desirable if the same tool can be used to simulate and document the product’s life-cycle behavior. This would provide a common form of documentation for all information regarding a product. VHDL signals can be defined that represent the allocation of design criteria in a system rather than the flow of digital signals. By appropriately defining bus resolution functions for these signals, they can be used to document and calculate the life-cycle design metrics of a system.

Chapter 3 discusses the specific contributions that VHDL can make to the life-cycle design process. Chapter 4 develops a digital system, a local area network product, as a case study using the process described by Chapters 2 and 3. Chapter 5 summarizes the results of this work.
2. The Design Process

When engineers design a life-cycle complete product, they employ a design philosophy that examines all issues pertinent to a product's life cycle. The system life-cycle process discussed in this chapter is a design procedure that focuses on these issues. The first section of this chapter describes the life-cycle design process used in this research.

The second section of this chapter discusses products competing for scarce corporate resources. Life-cycle complete design does not end with the life-cycle of single products. It also examines the interactions of all product life cycles within a company. The consideration of these interactions to provide the maximum possible benefit to the company is referred to as enterprise integration.

2.1 The System Life Cycle Process

The system life-cycle process discussed in this chapter comes from Blanchard and Fabrycky [BlaF90]. Figure 2-1 illustrates the system life-cycle process. A product's life cycle is broken down into life-cycle phases. Each of the boxes in Figure 2-1 represents a life cycle phase. When a product passes from one phase to the next it is said to have passed a life-cycle gate. Figure 2-2 provides a timeline of a product's life cycle with the life-cycle gates labeled. The life-cycle gating structure is based on the product development method used by Northern Telecom [TerJ91]. Research conducted at Virginia Tech has adapted the Northern Telecom gating structure to produce the more generalized form described here [HoeW93].
Figure 2-1. The life-cycle process [BlaF90].
2.1.1 Definition of Need

Companies create products to fulfill needs in the marketplace. Products fill gaps in existing systems using novel or more cost-effective solutions than other available products. After recognizing a need, the company must formally define the boundaries of the niche it wants to fill. This definition directs the design process toward specific goals.

The planning team must set a cost goal for a basic system unit. System cost influences the design process by limiting implementation options. This boundary condition becomes an important part of the remaining design process, when engineers search the design space for practical implementations.

System planners should establish effectiveness requirements for the system. To set effectiveness goals, the designers must first determine the expected lifetime of an individual system. This allows them to define measures of system reliability, availability, maintainability, and supportability [BarJ89]. Not all of these measures may apply to a given system. Some systems, like some produced for the military, may have their entire design process governed by the establishment of a very high reliability requirement. In the more likely case, reliability plays an important role as a boundary condition in the design space. The earlier the planners establish these requirements, the earlier they can perceive their effect on the system as a whole.
The system also has other, more qualitative, operational requirements. Will the system's users require operation under extreme temperatures? Does the system have to operate on the floor of an electrically noisy manufacturing facility? Do the users need a portable system? The answers to these questions could have significant repercussions on the design of the final system.

System engineers must consider other, more fiscal, issues as well. They must consider how the system's life cycle breaks down into fiscal periods. Companies allocate resources, like payroll, to products by fiscal periods. These periods may last for a year, a quarter, or even a month. A product's life cycle divides into life-cycle phases. Creating a system life-cycle timeline aids the system designers in understanding the time and resource constraints that the product may experience. Figure 2-2 illustrates a sample timeline.

![Figure 2-2. A sample product life cycle.](image)
The beginning of the utilization and support phase marks the date that the product becomes available to the consumer. If consumers require a certain date for the product's viability in the marketplace, then this becomes an important constraint on the design process. The product availability deadline may constrain the length of the preliminary design phase and thus lower the system's cost/performance ratio below its potential. Designers must remember, however, that if consumers cannot use a product that arrives late the entire process has wasted time and resources. Alternatively, if the date does not constrain the design process, the system engineers may allocate extra time to the preliminary phase and increase the quality of the design.

Products have resource requirements over their life cycle. Products consume resources like personnel, payroll, laboratory facilities, and computer simulation time among many others. In the design phases products require engineers, managers, and programmers. Later, during prototype construction, products require technicians, parts, and testing equipment. After the product has entered the market, it demands sales people, support staff, and toll-free telephone lines. The system engineers must consider and document these resource demands.

Earlier, product designers set a reliability requirement for a product. By assigning a product a reliability rating less than 1.0, the system engineers have admitted that the product will fail. The philosophy of concurrent engineering requires that if the engineers
know it will happen, they should plan for the eventuality. Some systems, like those
composed entirely of TTL devices, may have mean time to failures (MTTF) that last well
beyond the product's life expectancy. Others, like interconnection networks, may have
MTTF's much smaller than the product's life expectancy. Engineers should establish
preliminary policies governing repair, maintenance, or replacement of failed systems early
in the design process. This allows the resource requirements for maintenance to enter into
design decisions from the beginning of the design process.

The design process uses the definition of need as its first objective. As the design process
continues, engineers must insure that the designs under consideration address all issues
enumerated in the definition of need.

2.1.2 Conceptual Design
When the designers have fully established the definition of need, the conceptual design
phase can begin. In terms of life-cycle gates the product has passed Gate 0. Conceptual
design refines the definition of need into a detailed high level description of the new
system. When complete, the conceptual design documentation specifies the functional
behavior of the system. This includes its I/O ports, performance requirements, and timing
characteristics. The conceptual design describes the "black box" operation of the new
system. Figure 2-3 illustrates the issues involved in conceptual design.
Figure 2-3. Conceptual design activities [BlaF90].

2.1.2.1 Feasibility Studies
Feasibility studies answer some of the following questions. Does the technology needed to solve this problem currently exist? Do different technologies solve the problem in different ways? If so, what issues need consideration to choose between the different technologies? If designers find a well-established technology that solves the problem, then product development can continue along that path without much worry over how fast technology develops. If a useful technology exists, but remains largely untested by industry, the system planners must remain somewhat skeptical. In another case, designers can discover
that a technology does not exist. This requires serious attention since developing a product that uses nonexistent technology involves high levels of risk. If designers perceive the nonexistent technology as emerging soon, then product development can continue with an understanding of the risks involved. If the technology does not appear emergent, designers should drop the solutions involving those technologies from consideration.

The results of the feasibility studies should indicate what technologies apply to the product's development. The studies should also provide important criteria for evaluating the cost and performance of the differing technological solutions.

2.1.2.2 System Operational Requirements

The definition of need gives a general idea of the niche that a new product will fill. During conceptual design, analysis of how the product fills the niche produces a detailed mission description. The mission description details the exact functionality that the new product must provide. This description provides a baseline requirement for product designs.

Once designers understand the tasks that a product must perform, they must decide how well the product needs to perform these tasks. Engineers set performance goals for the product based on the environment in which it operates. General performance goals for digital systems are system throughput, accuracy, power dissipation, and size. These performance goals constrain the design process by making specific requirements on the quality of different design solutions.
The definition of need can specify some effectiveness requirements for the product. These include things like reliability, availability, and repairability. These metrics constrain mean time to failure (MTTF), another system parameter. The definition MTTF requires a use profile for the system. The use profile describes how the consumer uses the system and answers questions such as the following. How many hours per day does the system function? How often does the system get turned on and off?

Designers must anticipate the operational life of a system. How long will the system function effectively for the consumer? Will the system become obsolete before it wears out? Will the moving parts wear out while the consumer still needs the system? Answering these questions requires an understanding of how long the system will remain in place.

2.1.2.3 System Maintenance Concept
Systems break down and require maintenance. Product maintenance demands the designer’s attention early in development. A product’s maintainability depends on its system maintenance concept. The system maintenance concept details the plans for providing repair and upkeep to systems in the field.

The system maintenance concept establishes the producer’s basic repair and maintenance policy. Does the system have consumer repairable subsystems? If it does, will the producer expect the consumer to make on-site repairs to those systems? If it does not have consumer repairable subsystems, does the producer supply repair personnel to repair it
on-site, or does the consumer ship the system to an intermediate repair facility? The answers to these questions affect the cost and effectiveness of the system.

Making these policy decisions early in the design process creates critical constraints on the design process. If the consumer services some subsystems, the product should contain provisions for self-testing and support for non-expert repair. If the consumer cannot repair the system, these test systems only need to exist in repair facilities.

2.1.2.4 Preliminary Systems Analysis
As shown in Figure 2-3, the results from the previous steps all feed into preliminary systems analysis. This step begins with the operational requirements, maintenance concept, and technological alternatives, and brings them together into a system model. Engineers design systems to produce an optimized result. They begin by explicitly defining what they wish to optimize. Without this statement of purpose, engineers might as well design the simplest and thus possibly the most expensive system they can imagine.

The feasibility studies have, hopefully, produced multiple options for implementation. To limit the design space to a tractable volume, engineers must identify the different paths possible to achieve the design goals. Designers should codify these options into specific alternatives. Once the designers have a firm understanding of the alternatives, they must choose between them.
To choose between alternatives the designers use design goals as decision criteria. This requires metrics to measure the success of a particular design alternative in meeting the goals. Metrics measure the performance of models that simulate the behavior and properties of the alternate designs. By applying good input data, engineers can examine the performance of the models, and based on the metrics, choose the design alternative that best satisfies the design goals.

2.1.2.5 Advance System Planning
When designers identify the best design alternative, they collect all the information gathered up to this point. This information forms the system specification. It contains a description of the system's function and operation, the maintenance concept, functional interfaces, performance requirements, and effectiveness requirements. After approval, this document supplies the starting point for the preliminary design phase.

The product manager must also provide a management plan for implementing the rest of product development. Improved information about the future of the design process necessitates revision of the product life-cycle diagram. Newly uncovered information concerning the viability of the product may influence management's support of the product.
2.1.2.6 Conceptual Design Review

The conceptual design review serves several purposes. It provides a formal presentation of the design methodology, and results. It allows all interested parties to see the current design, and to comment on the results. This interaction allows people not directly involved in the design process, like support technicians or marketing personnel, to voice comments from their perspectives.

The design review insures that all the people involved in the design effort communicate about their individual responsibilities. The interfaces from various portions of the design must mesh together into a coherent whole. The design review enforces the compatibility of these interfaces, and provides some assurance that subsystems will mesh together correctly.

If the design fails to meet any of the requirements set by the reviewers, then some part of conceptual design requires revision. Design reviewers may require several design iterations before passing a product out of the conceptual design phase. Once a product passes the design review, it moves through Gate 1 and enters preliminary design.

2.1.3 Preliminary Design

Preliminary design takes the "black box" produced by conceptual design and begins to fill it, mostly with other black boxes. The designers have specified the device's input and output characteristics and the data transformation that takes place between them. The
input and output specifications should remain fixed. Some redesign may occur but it should only occur as a result of knowledge gained through preliminary design. The main concern of preliminary design is the structure and function of the data transformation.

2.1.3.1 System Functional Analysis

Analysis of the data transformation leads to a functional breakdown of the system. Blanchard and Fabrycky detail a system for conducting functional analysis that uses functional flow diagrams. The functional flow diagram divides the data transformation into a step-by-step process for producing the outputs from the inputs. The ordering of the steps provides a flow chart of the data conversion process. For digital systems, this flow chart is similar to a data flow graph. The basic operations performed in each step form a set of system functions necessary to perform the global transformation [BlaF96].

2.1.3.2 Preliminary Synthesis and Allocation of Design Criteria

Once the functional analysis produces a set of necessary data operations, preliminary synthesis takes place. Preliminary synthesis produces a block diagram of the system. The system designers assign the data operations generated above to subsystems. Top-down design reduces the more complex subsystems into their own hierarchy of subsystems. The members of this hierarchy form the basic functional units of the system. The functional units become the blocks in the diagram. The data flow graph produced by the functional analysis step specifies the basic interconnection between the functional units. Engineers use this block diagram as the basis for further system design.
During conceptual design, system engineers established performance and cost criteria for the system as a whole. The next step in design allocates the system wide performance and cost measures among the newly created hierarchy. Figure 2-4 illustrates the distribution of system unit cost among various levels of the hierarchy. If the designers consider the hierarchy as a directed tree, then for any node in the hierarchy, the sum of the costs of its immediate successors equals the cost assigned to it. The cost assignments made in this phase guide engineers in detailed design toward subsystem costs that satisfy the overall system unit cost objective.

Designers allocate other performance criteria as well. In a system that requires all components for proper system function, the overall system unit mean time to failure (MTTF) is given by the equation below, where n is the number of subsystems. $MTTF_{tot}$ is the overall system MTTF and the MTTF values of the subsystems are represented by the enumerated subscripts [JohB89].

$$MTTF_{tot} = \frac{1}{\frac{1}{MTTF_1} + \frac{1}{MTTF_2} + \ldots + \frac{1}{MTTF_n}}$$

In a system hierarchy like that shown in Figure 2-4, the MTTF of a given node is related to the MTTF values of its immediate successors by the equation above. System designers allocate other criteria based on rules appropriate for the given metric. These other criteria could include weight, throughput, and size.
2.1.3.3 System Optimization

Optimization of the various subsystems requires an objective function that evaluates the system as a whole with respect to the system's design goals. The objective function chosen for the task can vary widely depending on the system. Numerous factors play a part in choosing the specific function to use. A system with a low MTTF will tend to break down often. To lower the product's mean time to repair (MTTR), and thus increase its availability, designers might use an objective function that evaluates the system's ease of maintenance. Alternatively, if the system description calls for portability, system engineers would choose an objective function that examines weight and size. In general, the objective function operates on several of these criteria, with a heavy emphasis on life-cycle cost compared to predicted performance.
With the boundaries in place and the objective functions as a guide, the designers can explore the design space. Many different implementations might exist for each subsystem. To explore the design space, engineers examine the impact that different implementation options have on the objective functions. Simulations of the differing choices yield information on technical performance measures like throughput and system utilization. Engineers make estimates of cost and reliability to use in the objective function. The output of the objective function points to the design that best satisfies the system design goals.

2.1.3.4 System Synthesis and Definition
When the objective functions and engineers have done their work, a set of subsystem implementation options emerges as the best choice for the final design. The last step of preliminary design is to collect all the information gathered about this best design and create a system description to drive the detailed design process.

2.1.3.5 Preliminary Design Review
Before passing Gate 2 and entering the detailed design phase, the product must pass the preliminary design review. The objectives of this review mirror those of the conceptual design review. It provides a formal setting for reviewing the work completed thus far. The design engineers present the results of the preliminary design to people not directly involved in the design effort, but whose opinions and ideas support the process. When the
design satisfies the members of the review committee, the product passes Gate 2 and enters detailed design.

2.1.4 Detail Design and Development
After preliminary design provides specifications for an implementation, detailed design fills the black boxes with actual components. The principles of life-cycle complete design must continue during the detailed design phase. All the considerations examined in the previous phases merit further attention during this phase.

Blanchard and Fabrycky list twenty-two considerations for detailed design, reproduced here as Figure 2-5. As the details of the final design emerge, these considerations require greater and greater attention since the scope of managing the design increases with every addition.

2.1.4.1 Feedback from Detailed Design
As the detailed design process progresses, feedback from various completed systems may affect other subsystems in the unit. Subsystems that come in under some budgeted attribute may allow reallocation of the excess budget to other subsystems. This allows some flexibility in the design of subsystems, while retaining the overall budgeted system specification. Consider the following example.
1. Accessibility  
2. Adjustments and alignments  
3. Cables and connectors  
4. Calibration  
5. Packaging and mounting  
6. Disposability  
7. Environment  
8. Fasteners  
9. Handling  
10. Human factors  
11. Interchangeability  
12. Maintainability  
13. Panel displays and controls  
14. Producibility  
15. Reliability  
16. Safety  
17. Selection of parts/materials  
18. Servicing and lubrication  
19. Software  
20. Standardization  
21. Supportability  
22. Testability

Figure 2-5. Concurrent design issues [BlaF90].

Suppose that Subsystem A has a budgeted MTTF of 10,000 hours. Further, suppose that Subsystem B and Subsystem C comprise the only two subsystems of Subsystem A and have budgeted MTTFs of 17,000 hours and 25,000 hours, respectively. Subsystem A only functions correctly when both Subsystem B and Subsystem C work correctly. The MTTF of Subsystem A, calculated from the MTTFs of its subsystems, is the inverse of the sum of the reciprocals of the budgeted MTTF values of Subsystems B and C. The calculation yields a MTTF for Subsystem A of 10,119 hours which slightly exceeds the requirement.

Suppose that the most reliable design, within cost constraints, found for Subsystem C has a MTTF of 18,000 hours. Analysis of Subsystem B reveals that it has a MTTF of 30,000 hours. Since Subsystem B has exceeded its MTTF requirement the overall MTTF of Subsystem A equals \([(30,000)^{-1} + (18,000)^{-1}]^{-1} = 11,250\) hours, which exceeds the required...
MTTF. This excess MTTF could then in turn allow a slight decrease in the MTTFs of subsystems whose place in the hierarchy links their MTTFs to that of A.

2.1.4.2 The Prototype
When the designers have completed all the subsystems and verified the design through simulations, prototype production begins. The prototype verifies the functionality of the design. It is the final test of the detailed design. If the comparison of output from the prototype does not meet the detailed design specification, the discrepancy feeds back into the detailed design process, where engineers diagnose and correct the problem. If the output meets the description the design is ready to enter production. The product has passed Gate 3.

2.1.5 Production and Construction
When a product enters production, its design remains relatively fixed. Production may reveal flaws in a design's producibility. If the flaws severely hamper production, redesign of the flawed areas may become necessary. Outside influences, such as parts availability, may prevent production. The detailed design phase should have addressed these issues as part of the design for producibility, but outside forces may compel changes.

Production versions of the product require testing and evaluation to insure that any differences between the production model and the prototype remain within the product specification. Testing must verify printed circuit board designs for the production version.
If the prototype experienced noise problems because it was implemented on a protoboard, and system designers assumed the noise would go away in the printed circuit board implementation, they must verify this assumption. All these issues feed back to influence the design. When the production model passes these final tests it passes Gate 4 and enters the marketplace.

2.1.6 Utilization and Support
During utilization the product suffers testing by the ultimate critic, the customer. Comments from the customer represent the final feedback loop in the design process. Engineers may have to correct glaring problems found by users by altering the detailed design. In some cases, designers may correct problems easily. Programmers can sometimes alter software relatively easily. Problems in the hardware design tend to present much greater difficulties. Major revisions of the design at this stage cost a great deal since changes in the hardware design require updating all the existing units and redesigning the production facilities to accommodate the change. If the required hardware modification changes a mask for an integrated circuit implementation, for instance, the cost of producing the new mask could make the product untenable. If the problem requires major revisions of the hardware design, the design process failed to do its job correctly.

Some problems may become product enhancements that feed forward into the next generation of the system. This information flow provides a rich source of suggestions for improvements in any successor systems. The successor system enters the market when the
original product's lifetime has expired, or it has become obsolete. When this happens, the company phases the original product out of the marketplace.

2.1.7 Phaseout and Disposal
The final phase of a product's life removes it from useful service. When this has occurred the product has passed Gate 5. In planning a new product, engineers must have a plan for its eventual obsolescence. Removing a product from service may incur costs to the company that produced it. The company must prepare the product's customers for termination of maintenance availability, spare parts availability, and customer support. Just disposing of the product physically may also incur a cost to the producer. Designers must account for these concerns and costs during the design process.

2.2 Inter-Product Competition for Scarce Resources
Companies that produce more than one product must consider the interactions between those products. Companies do not have unlimited resources. Executive decision makers budget funds for product development, product support, marketing, and so on. Products in the design stages of their life cycles must share the budget for product development. New products cannot enter the design phase when existing products have consumed the product development budget. Products not only interact by consuming funds, they also consume other less obvious resources. Products in development require engineers, programmers, and CAD workstations, for example. Products in production require
distribution resources like trucks and storage space. Successful enterprise integration encompasses all of these interactions and produces the best possible set of products.

2.2.1 Product Lifetime Resource Requirements

The Product Portfolio/Enterprise Integration Demonstration Facility (PP/EI DF see section 2.2.2) optimizes the future worth of a company's portfolio of products constrained by the interactions of six generalized resource requirements. PP/EI DF examines product interaction in terms of budget, revenue, staffing, storage space, production facilities, and design facilities. These interactions do not form a complete set, but they make up a good example set. Analysis of interactions between products requires that each product have a complete description of its resource requirements for its lifetime. Initially this description represents an educated guess, but as the product progresses through its life cycle, the predictions improve.

2.2.1.1 Increasingly Accurate Predictions

As early as the beginning of the conceptual design phase, designers need to make predictions for a product's resource requirements. System designers must treat these initial predictions with the utmost concern since they may help decide whether a company funds a product's development.

The resource requirement functions predicted during conceptual design constitute educated guesses based on the company's experience. The accuracy of the predictions
improve with every gate that a product passes. After passing Gate 1, the product has taken on an identifiable shape. Designers have defined its functionality and interface. Experience with similar systems may produce more accurate predictions of potential resource requirements. Passing Gate 2 the product enters detailed design. Now the product has a shape and a direction for implementation. These details further enhance the predictions by indicating not only the basic functionality, but also the underlying technology. Proceeding through Gate 3 reveals the complete design for the product. The detailed design defines the system unit cost and its potential maintenance requirements become clearer. Revenue information predictions improve based on customer response to the now well defined product. Gate 4 leads to entrance into the marketplace. Revenue predictions become even more precise.

Predictions improve with each life-cycle phase. More information is available for resource prediction the further along a product is in its life cycle. Designers should update resource requirement predictions for every fiscal period to reflect new information about the product. The resulting resource requirement functions feed into the executive decision making process that designs product interaction.

2.2.1.2 Resource Requirement Functions

Formal treatment of the resource requirement functions requires a formal definition. The domain of the resource requirement functions spans fiscal periods. The function maps fiscal periods to resource requirements for that period. Product managers build the
requirement functions from five simple functions: uniform, gradient, growth, decay, and exponential. These five functions describe typical resource requirement behavior [BucH75, SkiL85]. Figure 2-6 illustrates these functions.

The numbering of the fiscal periods associates a number with a period of time instead of an instant of time. To define the continuous independent variable \( t \) over a fiscal period \( m \) the value of \( t \) begins at \( m-1 \) and ends at \( m \). For example, for fiscal period 3, \( m=3 \), \( t \) ranges from 2 to 3, inclusive.

The constant function has one parameter, \( C \). Product managers use the constant function to represent costs that do not change over time. Constant functions could model lease payments on office space, for example. The gradient function has one parameter, \( G \). Product managers define a gradient function over a time period to represent a cost that increases linearly over time. The \( G \) parameter defines the change of the function over one fiscal period. Examples of constant and gradient functions can be found in Figures 2-7 and 2-8.

The remaining three exponential functions model more complex resource requirements. Each has two parameters, \( A \) and \( r \). In all three of the exponential functions the \( r \) parameter controls the rate of change. The exponential function produces a simple exponential curve. The growth function models a resource that increases relatively quickly and then levels off.
to approach a certain value. The $A$ parameter in the growth function becomes the horizontal asymptote of the function. The decay function models a resource that begins at a given level and then drops off quickly at first and then slowly approaches zero. The $A$ parameter in the decay function controls the initial value of the resource function. An example of the growth and decay functions can be found in Figures 2-7 and 2-8.

The product manager constructs a resource requirement curve for a product by superimposing instances of these five simple functions. The manager begins with the life-cycle timeline for the product. The product manager uses one such timeline for each modeled resource. The manager then places instances of the five simple functions onto the timeline to represent the requirements of a particular resource.

The functions begin and end during particular fiscal periods. Formally, each simple function becomes a piecewise function, $f(x)$, of the following form.

$$f(x) = \begin{cases} 
0 & \text{for } x < m - 1 \\
 g(x - m) & \text{for } m - 1 \leq x \leq n \\
0 & \text{for } x > n 
\end{cases}$$

The function $g(t)$ holds the place of one of the five simple functions. The constants $m$ and $n$ represent the beginning and ending fiscal periods of the function. Figure 2-7 shows a sample graph for a product's budget resource requirement function.
Figure 2-6. Resource requirement functions.

Uniform\( (t) = C \)

Gradient\( (t) = Gt \)

Exponential\( (t) = Ae^{rt} \)

Growth\( (t) = A(1 - e^{-rt}) \)

Decay\( (t) = Ae^{-rt} \)
The budget resource requirement function in Figure 2-7, consists of seven component functions. Figure 2-8 details the seven functions and the parameters that define them. The product's budget starts with the conceptual design phase. The curve indicates that the budget remains fixed during this phase. The funds pay mostly for a small staff of systems engineering and design personnel. When the product passes Gate 1 and enters the preliminary design phase, the budget begins to increase. As the product development progresses, the design staff increases to deal with the added complexity. Designs require more simulation time and CAD resources. By the end of the preliminary design phase, the budget requirements for design level off as detailed design begins. The budget for the detailed design phase is represented by a single constant function. When the product passes Gate 3 and enters production, the budget again increases to handle the costs of setting up manufacturing facilities and adding production staff to the payroll. Passing Gate 4, the product enters utilization and support. The budget starts at the same level as the end of production, as the design team resolves any remaining bugs in the system and then drops off quickly as the design team members move to other projects. The product now only incurs the costs of supporting the product's users. Finally, passing Gate 5, the curve indicates a flat cost for the final phase out and disposal of the product.
**Figure 2-7.** A sample budget requirement curve.

<table>
<thead>
<tr>
<th>Function</th>
<th>$m$</th>
<th>$n$</th>
<th>$C$</th>
<th>$G$</th>
<th>$A$</th>
<th>$r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>1</td>
<td>1</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>2</td>
<td>3</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gradient</td>
<td>2</td>
<td>3</td>
<td></td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>4</td>
<td>5</td>
<td></td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>6</td>
<td>6</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Growth</td>
<td>6</td>
<td>6</td>
<td></td>
<td>100</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Decay</td>
<td>7</td>
<td>10</td>
<td></td>
<td>400</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2-8.** Functions used in sample budget curve.

The product manager generates a similar curve to describe each of the product's other resource requirements. In addition, the manager must produce the product's estimated revenue curve. The revenue curve mimics the resource curves in construction. The revenue curve distinguishes itself from other curves by predicting incoming funds, which presumably the company does not wish to constrain. Together these curves describe the
product's demand for limited company resources and its contribution to company revenue over the whole of its life cycle. When company executives combine this information with similar information from other products, the sum of the resource requirements must not exceed the organizational constraints. Executives may also require the summed revenue curve to exceed some value in each fiscal period.

2.2.1.3 Organizational Constraint Functions
Corporate-level executives define the organizational constraint functions. These functions represent resource availability on a fiscal period by fiscal period basis. The company's production, research and development, and support services cannot exceed these global constraints. Executives construct the organizational constraint curves from the same five simple functions used to define product resource requirement functions. Decision makers detail the company's resources on a timeline similar to the one used for resource requirements. The constraint timeline does not include life-cycle phases. It simply has as many fiscal periods on it as necessary to provide resources for the company's longest lived product. Executives create one of these resource constraint curves for each resource demanded by the products. Additionally, executives might establish a minimum revenue curve that defines the minimum acceptable revenue in a given fiscal period.

2.2.1.4 Optimizing Future Worth
A company's global resource constraints limit the number of products that a company can research, develop and produce. Usually more potential products exist than a company can
produce. The company should produce the set of products that has the maximum future
worth. Selection of the best set of products to produce is thus an optimization problem.

2.2.1.5 The Objective Function
Computation of the future worth of a set of products requires comparison of products
with unequal lives. The solution adopted here uses a minimum acceptable rate of return
(MARR) to compare products with unequal lives. The horizon for a set of products is the
end of the life cycle of the longest lived product in the set. After computing the future
worth of each product, analysts use the MARR to move the discrete future worth from the
end of the product's life cycle to the horizon.

Calculation of an individual product's future worth involves uncertainty, since product
managers estimate the revenue and the resource requirement functions. To model the risk,
the analyst associates a risk adjusted rate of return (RARR) with each life-cycle phase of
the product's life cycle. This rate represents the relative uncertainty of the predicted
functions. Analysts calculate the future worth of a product by determining the area under
the revenue prediction and budget requirement for each fiscal period in the product's life
cycle. Then, beginning at the current fiscal period, the analyst sums each of these discrete
amounts, moving the sum forward in time using the RARR values associated with each
life-cycle phase. When the sum reaches the end of the product's life cycle it represents the
future worth of the product.
Analysts use the future worth of the product at the horizon as the objective function for optimizing the product portfolio. Companies should select the set of products that produces the highest future worth without exceeding the company's resource constraints.

2.2.1.6 Searching for Feasible Sets
If a company has a set of $P$ products to choose from, then $2^P$ possible combinations exist. The most straightforward way to determine which has the highest future value involves examining each combination. For each of the $2^P$ possibilities, analysts, or an automated program, must sum the resource requirements and insure that they do not exceed the organizational constraint function. Executives fund the portfolio that has the highest future worth and satisfies the resource constraints.

2.2.2 Product Portfolio/Enterprise Integration Demonstration Facility
The Product Portfolio/Enterprise Integration Demonstration Facility (PP/EI DF) provides an automated process for determining optimal product portfolios. It provides a user interface for constructing resource requirement/organizational constraint functions. These functions automatically feed into an optimization module that searches all $2^P$ portfolio combinations for the one with maximum future worth [HoeW93].

The PP/EI DF introduces several new features into the optimization process. Executives can create interrelationships among products that do not depend on resource constraints. Executives can flag products that would directly compete with each other as mutually
exclusive. This removes portfolios with both products from consideration. For example, if a corporate executives were examining two different low-cost line printer products, the executives would indicate that the products were mutually exclusive. The company might wish to produce one low-cost line printer, but not two. Executives can use a contingent relationship to consider only portfolios that contain interrelated products. Continuing with the previous example, a printer ribbon cartridge product associated with one of the line printer products would be contingent upon that line printer's acceptance. The company would not produce the line printer ribbons if the company was not going to produce the printer.

2.3 Chapter Summary

This chapter provides background material that explains the philosophy of life-cycle complete product design. The information presented covers both the general principles of concurrent engineering, and a specific life-cycle design process due to Blanchard and Fabrycky. This process motivates the remainder of this research.

Life-cycle complete design attempts to minimize the cost and design time required to perform product development. Classical design techniques tend to ignore some issues of a product's life cycle until the opportunity to change the design to accommodate those issues has passed. By studying the behavior of product development, life-cycle complete
design anticipates costs that occur throughout a product's life-cycle and recognizes design issues that can minimize these costs.

Further, corporate executives can optimize the competition between products for scarce resources within the corporation. By defining corporate resource constraints and resource requirements for individual product life cycles, executives can study the interaction of the product's resource requirements over a period of study. The results of this study can produce an optimal product funding decision that maximizes the future worth of the corporation.
3. Using VHDL to Support the Design Process

This chapter investigates the utility of VHDL as an aid for the life-cycle complete design of products. VHDL has been used effectively to simulate and document the functional aspects of hardware design [ArmG93, ArmJ89]. It would be desirable if the same mechanism could be used to simulate and document the product’s life-cycle behavior. This would provide a common form of representation for all information regarding a product. This chapter examines the use of VHDL to address three aspects of life-cycle complete design: system parts cost, system reliability and product resource requirements.

VHDL simulates the functional behavior of digital systems by using a special kind of data object called a signal. Signal data objects simulate wires by interconnecting the ports of VHDL models. VHDL does not restrict the values that can be sent across its simulated wires to merely logical 1’s and 0’s. The signals that propagate throughout a system can be of any data type, even complex ones like records and arrays. There is no reason why these signals flowing inside the system must be related to electrical phenomena describing the circuit. The signals can just as easily be cost, mean time before failure, and resource requirement values. The VHDL paradigm presented here exploits VHDL’s hierarchical programming structures and specialized signals to model life-cycle complete behavior.

Before cost and resource requirements can propagate through circuits, special data types and functions must be created to handle them. This chapter describes three VHDL
packages that implement these functions for the three design criteria under study. Each package provides data types for storing life-cycle complete information, functions that "probe" the life-cycle signals as they traverse the system's hierarchy and bus resolution functions that tell VHDL simulator how to handle these signals.

3.1 Parts Cost
A system's parts cost is the cost of the components that make up the system description. The "parts cost" signal designed here propagates the cost of components throughout a system's design hierarchy. The code for the parts cost package can be found in Appendix A.3. When signal assignment is performed in VHDL the default bus resolution function assumes that the signals should behave like electrical signals. When a component with a cost assigns its cost to the global cost signal, the default resolution function does not provide the appropriate response. Each time a new assignment is made to the global cost signal the resulting value should be the sum of all the assignments made to the signal so far. To implement this, a new bus resolution function must be created to resolve cost signals.

3.1.1 Parts Cost Resolution Function
Bus resolution functions are associated with particular data types. The parts cost package defines a data type called ResolvedCost as a subtype of real. All signals within a VHDL model that wish to use the parts cost resolution function are of type ResolvedCost. The function itself is defined in the package body.
The resolution function receives a list of all the values that signal drivers have assigned to a signal. The parts cost resolution function uses a for loop to sum all the values. The function returns the sum as the final value of the signal.

3.1.2 Parts Cost Output
The parts cost package also provides an output procedure for monitoring the value of cost signals in the system. The OUTCOST procedure has two parameters. The first is the cost signal to monitor. The second is a file name used to store the results. The procedure opens the specified file and outputs the current value of the cost signal.

3.2 Mean Time Before Failure
Mean time before failure (MTBF) is a measure of the reliability of components. In a similar fashion to parts cost, a MTBF signal can be defined that automatically calculates the overall MTBF of a system based on the MTBF values of its component parts. This metric can then be used to calculate reliability, availability and maintainability metrics for the system. The code for the reliability package can be found in Appendix A.2.

3.2.1 MTBF Resolution Function
MTBF values do not add together like parts costs. For systems that require all components to function for the system to function, MTBF values are combined like resistors in parallel. The overall MTBF of a system is the reciprocal of the sum of the reciprocals of MTBF values for all components as described in Section 2.1.3.2.
The MTBF resolution function provided in the reliability package performs this operation. The bus resolution function is passed an array of all the MTBF values that have been assigned to the signal. A for loop sums the reciprocals of the values and when finished, inverts the answer to produce a comprehensive MTBF. The resolved MTBF signal type is called ResolvedMTBF.

3.2.2 Effectiveness Measures Output

MTBF can be used as the basis for many common effectiveness ratings. The output functions for the reliability package provide three such functions: reliability, availability and maintainability. Each describes a particular aspect of a system's effectiveness.

3.2.2.1 Reliability

Reliability is usually expressed as a percentage. Sometimes called the survival function, reliability is the probability that a system will function correctly for a given period of time. When discussing the reliability of a system, the period of time is usually the expected service lifetime of the system [BlalF90]. The reliability package provides two functions that give reliability information for a system. CHECKREL calculates the reliability as a decimal number based on a supplied MTBF value and a period of time. The units of time used to describe the period of time must be the same units used to calculate the MTBF. CHECKREL returns the reliability metric as a real value. CHECKREL can be used to setup automatic effectiveness monitoring within a model. This principle is discussed in more detail in later sections. OUTREL performs the same reliability calculation based on
MTBF and lifetime, and then writes the results to a filename that is specified in the function call. The equation for reliability is shown below where \( t \) is the lifetime of the device.

\[
Reliability(t) = e^{-\frac{t}{MTBF}}
\]

This function assumes an exponential distribution of failure times for system components. Analysis of real world systems indicates that the exponential failure distribution models the failure of electrical components well. Other failure distributions are possible. The Weibull distribution, for instance, is used to model the reliability of software systems [JohB89]. The exponential failure distribution is used here exclusively.

### 3.2.2.2 Availability

Availability is also expressed as a percentage. Availability is the probability that a system will be functioning correctly at any given moment during its lifetime [BlaF90]. The reliability package provides two functions for working with availability. CHECKAVA is passed the system's MTBF and the mean time to repair (MTTR). Based on these two values, CHECKAVA returns a real number representing the system's availability. OUTAVA performs the same calculation and writes its results to a file specified in the function call. The equation for availability is given below.

\[
Availability = \frac{MTBF}{MTBF + MTTR}
\]
3.2.2.3 Maintainability

Maintainability is the probability that a system can be repaired in a certain period of time [JohB89]. Two functions are provided for working with maintainability. The first, CHECKMAN calculates the maintainability based on the MTTR and a given time. It returns the maintainability value as a real number. OUTMAN performs the same calculation and writes the results to a filename specified in the function call. The formula for maintainability is given below.

\[ \text{Maintainability}(t) = e^{(\frac{t}{\text{MTTR}})} \]

3.2.3 Real Valued Functions in VHDL

One weakness of VHDL is its lack of real valued functions. The calculations above require exponentiation to fractional powers. VHDL does not support this as part of its standard libraries. To create the reliability package another package containing real valued functions was had to be implemented. This real valued function package is listed in Appendix A.4.

VHDL provides limited support for implementing real valued functions. VHDL's real data type only supports single precision floating point numbers, which are generally inadequate for performing accurate numeric processing. The real functions provided by the package use single precision arithmetic. The table below summarizes the functions provided by the floating point math package.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp(x)</td>
<td>returns $e^x$</td>
</tr>
<tr>
<td>ln(x)</td>
<td>returns $\log x$</td>
</tr>
<tr>
<td>fabs(x)</td>
<td>returns $\text{abs}(x)$</td>
</tr>
<tr>
<td>pow(A,x)</td>
<td>returns $A^x$, where $x$ is an integer</td>
</tr>
<tr>
<td>fpow(A,x)</td>
<td>returns $A^x$, where $x$ is a real number</td>
</tr>
<tr>
<td>fact(x)</td>
<td>returns $x!$</td>
</tr>
<tr>
<td>iatfrac(x,w,f)</td>
<td>returns the integer part of $x$ in $w$, and the fractional part of $x$ in $f$</td>
</tr>
</tbody>
</table>

The functions are implemented as Maclurin series expansions [SwoE88]. The series are calculated iteratively. Iterations continue until either the results of adding a new term does not change the previous iteration's result or a maximum number of iterations has been reached. These functions are, unfortunately, slow. In the future, VHDL will require external language interfaces to C to provide access to optimized floating point routines.

### 3.3 Resource Requirement Functions

Resource requirements for a system are described using resource requirement functions. These functions are described in Chapter 2. Complex systems are usually divided into less complex subsystems. As this hierarchy is formed each new subsystem contributes a portion of the resource demand for the whole system. In a manner similar to summing parts costs, a method is required to sum the resource requirement functions for subsystems at the system level, or any other level of the hierarchy.
3.3.1 Resource Requirement Resolution Function

Resource requirement signals are arrays that contain records as elements. Each record contains a description of a resource requirement function. Together, the records in the array form the total resource requirement function. When a signal assignment is made to a resource requirement signal, the array of functions being assigned is appended to the list of functions previously assigned to the signal. The functions are thus accumulated across all subsystems until a complete list of resource requirement functions is available at the system level.

The bus resolution function for resource requirements is defined in the package body. It receives as input an array of arrays containing all the arrays being driven by the signal drivers associated with a given signal. It then produces one output array that contains all elements of each array passed to it. The final value of the signal is an array that contains all resource requirement functions assigned to a given signal across the entire system. The resolved signal type is called ResolvedConst.

3.3.1.1 Resource Requirement Function Data Type

The resource requirement arrays are made up of records that contain the resource requirement functions as described in Section 2.2.1. The records are called ConstParams and are defined as follows.

```vhdl
type ConstParams is record
  PTYPE: Functype;
  PBEGIN: natural;
end record;
```

Chapter 3. Using VHDL to Support the Design Process 45
FEND: natural;
A: real;
J: real;
end record;

The first field, FTYPE, is of an enumerated data type. It can take on one of six values: NONE, UNIFORM, GRADIENT, GROWTH, DECAY and EXPONENTIAL. NONE is a placeholder value that indicates no function occupies that space in the array. The remaining five correspond to one of the five resource requirement function types.

The next two fields, FBEGIN and FEND, are natural numbers. They specify the beginning and ending fiscal period for the function being described. Following those fields are the function parameters. The A field is used by the UNIFORM and GRADIENT function to indicate the magnitude and slope of the function, respectively. The A field is used by the GROWTH, DECAY and EXPONENTIAL functions as a scaling value. The J field is only used by the GROWTH, DECAY and EXPONENTIAL functions. It is the constant that appears in the exponent of each of those functions.

3.3.1.2 Aggregate Initialization of Resource Requirement Functions
It is often convenient to make signal assignments to resource requirement functions using aggregate initialization. In the case study that follows in Chapter 4, configuration files are used to initialize generic variables that contain arrays of resource requirement functions. Such initialization simply follows VHDL's rules for creating aggregates. The example below illustrates assigning an array of two resource requirement functions to a resource requirement signal.
Budget <= \( \begin{cases} 
0 & \leq (\text{UNIFORM}, 2, 9, 150, 0, 0, 0), \\
1 & \leq (\text{GROWTH}, 5, 10, 200, 0, 1.2), \\
others & \leq (\text{NONE}, 0, 0, 0, 0, 0, 0) 
\end{cases} \);

The first function is a gradient with a slope of 150 that spans fiscal periods 2 through 9. The second is a growth function with a scaling value of 200 and an exponential constant of 1.2. It spans fiscal periods 5 through 10. The remaining array slots are assigned as type NONE functions. This is required by the bus resolution function, so that it can detect the end of the function list.

3.3.1.3 Checking Requirements Versus Limits

The resource requirement package provides a function called CHECKCONST that compares resource requirement functions. It takes two arrays of resource requirement functions, one as a limit, and the other as a set of functions to test against the limit. CHECKCONST sets up an array of fiscal periods and then calculates the aggregate requirement for each period based on both requirement arrays. The function then compares each fiscal period from each requirement arrays to insure that the test resource array is either above or below the limit array. A third parameter, DIRFLAG, determines whether the test array should be above or below the limit array. The fiscal periods for which the test array fails to compare with the limit array are saved in a file specified by a fourth function parameter.
3.4 Design Criteria in the Hierarchy

The purpose of the signals discussed in this chapter is to connect levels of the design hierarchy and facilitate modeling and simulation for life-cycle complete design. Using the signals within the design hierarchy allows each subsystem to report its own design dependent metrics and send its data to higher levels so that the same calculations can be made there. This allows designers to understand the allocation of the design criteria among the subsystems.

3.4.1 Connecting the Signals

When VHDL initializes a model, it executes every concurrent statement in the model. The life-cycle analysis of the model occurs during initialization. Signals are given their initial values and then signals that have multiple assignments made to them are resolved. Signals that depend on others signals being resolved first are then resolved and so on until all are completed. By assigning life-cycle signals initial values, the calculation of their aggregates across the system is carried out automatically during initialization.

In addition to normal I/O port connections, subsystems now have life-cycle metric connections. A sample port description from the case study in Chapter 4 demonstrates this.

```vhdl
port(BUDGET: inout ResolvedConst := BUDGETLIMITS;
    REVENUE: inout ResolvedConst := REVENUEMIN;
    STAFF: inout ResolvedConst := STAFFLIMITS;
    MTBF: inout real := 675.0e+3;
```

Chapter 3. Using VHDL to Support the Design Process
PARTSCOST: inout real := PARTSCOSTLIMIT);

This device has three ports for resource requirements, one for MTBF, and one for parts cost. These signals are interconnected at the subsystem level to form a bus that calculates their aggregate value. The components below illustrate this interconnection.

C1:MICROPROCESSOR
    port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

C2:ETHERLINK
    port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

C3:RAM
    port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

C4:ROM
    port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

C5:MOUNTING
    port map (BUDGET, REVENUE, STAFF, PARTSCOST);

C6:MANUALS
    port map (BUDGET, REVENUE, STAFF, PARTSCOST);

All of the component's budget ports are connected at the subsystem level by the BUDGET signal. The same applies to REVENUE, STAFF, MTBF and PARTSCOST signals. These signals, once resolved, contain the aggregate of each design metric. They can be examined at this level of the hierarchy and then passed to levels above for further computation.

It is important to note that signals in VHDL have only one current value associated with them. If all of the signals in the system are connected, then the signal's value after resolution would be the aggregate value for the entire system. To examine values for a given local subsystem, it is necessary to have local design metric signals. These local
signals are interconnected to all components of a subsystem, but are not themselves attached to systems at higher levels. To pass the values of the local signal to higher levels of the hierarchy, the signals from higher levels are assigned the value of the local signal.

### 3.4.2 Checking the Results

Once the system is set to automatically calculate its life-cycle metrics, it can also monitor them. When design entities are created, `assert` statements can be included in their concurrent code section that will automatically provide warnings if the entity exceeds its design criteria allowance. The following code fragment from the case study in Chapter 4 illustrates this.

```vhdl
assert CHECKCONST(BUDGETLIMITS,BUDGET,UPPER,"Budget","Budget.chk")
  report "Bridge has failed to meet Budget.";
assert CHECKCONST(REVENUEMIN,REVENUE,LOWER,"Revenue","Revenue.chk")
  report "Bridge has failed to meet Revenue.";
assert CHECKCONST(STAFFLIMITS,STAFF,UPPER,"Staff","Staff.chk")
  report "Bridge has failed to meet Staff.";

assert (CHECKREL(MTBF,LIFE) > ReliabilityLimit)
  report "Bridge has failed to meet reliability.";

assert (PARTSCOST <= PARTSCOSTLIMIT)
  report "Bridge has exceeded its parts cost.";
```

This technique automatically warns the designer of violations in the allocation of design criteria in a subsystem.

### 3.5 Organizational Information Flow

The corporate executives, product managers and engineers all interact with PP/PI DF and the VHDL models to make decisions. Figure 3-1 graphically illustrates these interactions.
The corporate executives make decisions about organizational resource constraints, product interactions, and risk. These decisions all flow into the PP/EI DF and form the boundary conditions for optimizing the corporation’s product portfolio. Finally, when all the product information is assembled, the PP/EI DF returns an optimal portfolio to the corporate executives.

The product managers predict the resources required by their products. These product resource requirements flow into the PP/EI DF as part of the data that PP/EI DF uses to optimize portfolios. Product managers (with systems engineers) also make decisions regarding overall parts costs and reliability requirements for a product. This information feeds into the VHDL models as design constraints.

Engineers interact with the VHDL models by specifying the design hierarchy. The VHDL models provide feedback to the engineers about how well the design hierarchy meets the design requirements and resource constraints.

The PP/EI DF and the VHDL models also interact. The product resource requirement predictions provided by the product manager are fed into the VHDL models as product resource constraints. In turn, when a design hierarchy is established in the VHDL models, the models feed the actual resource requirements for the product back into the PP/EI DF.
The executives, product managers, and engineers also interact on a personal level. Design meetings and design reviews constitute information flows that take place outside of the interactions via CAE tools and models.

![Diagram showing organizational information flows](image)

**Figure 3-1.** Organizational information flows.
3.6 Chapter Summary

VHDL signals can be of any allowable data type. Thus, signals can be defined that represent the allocation of design criteria in a system rather than the flow of data. By appropriately defining bus resolution functions for these signals, they can be used to document and calculate the life-cycle design metrics of a system. This chapter described the design of three such signals: parts cost, MTBF and resource requirement functions. The chapter developed a conceptual framework for interconnecting the signals to provide information on design criteria at the subsystem and system levels. Methods for automatically checking the design metric requirements of each subsystem were also discussed. Now that a conceptual framework for life-cycle complete system modeling is in place, Chapter 4 develops a case study using the new framework.
4. VHDL in the Design Process: A Case Study

This chapter presents a case study that demonstrates the principles developed in Chapter 3. The design of an Ethernet local area network (LAN) bridge, based on an Digital Equipment Corporation LANBridge 100, forms the basis of the case study. The description of the LANBridge 100 in [HawK86] provides information about the product's design process so that simple manipulation can place it within the conceptual framework discussed here. Most of the original design of the LANBridge 100 remains, but modifications have been made to facilitate discussion.

The chapter is organized into sections covering each phase of the design process. The application of VHDL within each phase is presented. The discussion starts with definition of need and continues through the basics of detailed design. The VHDL model of the LANBridge 100 evolves through the phases of its design. A fully functional model of the LANBridge 100 is beyond the scope of this work. The principles of functional modeling in VHDL are well understood. The model developed here concentrates on issues related to life-cycle complete design.

4.1 Definition of Need

Product design is motivated by the perception of a need in the marketplace. In the early 1980's LAN technology became a quickly expanding area of computer networking. As customers emerged, so did a wide array of networking environments. DEC perceived a need at this time to provide network products that solved the different networking
problems created by the different environments. DEC began the Broadband Project to investigate what sort of LAN products DEC should develop [HawK86]. This case study begins with the perceived need for a LAN product that adapts to this diverse set of customer requirements.

Customer requirements for LAN technology span a wide range of applications. A LAN product to serve that customer base should provide flexibility to both the consumer, whose LAN requirements may change, and the producer, who would like to produce one low-cost product that solves multiple problems. LAN applications divide into three basic markets. The table below, from [HawK86], summarizes the markets for LAN technology and demonstrates the diversity of their requirements.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Extent</th>
<th>Number of Stations</th>
<th>Physical Environments</th>
<th>Frequency of Station Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Office</td>
<td>Less than 3 km</td>
<td>Less than 130</td>
<td>Benign</td>
<td>Occasional</td>
</tr>
<tr>
<td>Campus</td>
<td>Less than 25 km</td>
<td>Less than 10,000</td>
<td>Benign within a building, harsh between buildings</td>
<td>Possibly frequent</td>
</tr>
<tr>
<td>Factory</td>
<td>Less than 8 km</td>
<td>Less than 2200</td>
<td>Harsh</td>
<td>Rare</td>
</tr>
</tbody>
</table>

*Figure 4-1. Different LAN environments [HawK86].*

The three application groups produce three different types of data traffic: terminal-to-computer, computer-to-computer, and real-time. Terminal-to-computer data
traffic produces bursty relatively low bandwidth traffic from many terminals to a smaller number of computers. Computer-to-computer traffic produces higher bandwidth bursty traffic between any two computers on the network. Real-time communications produce continuous high-bandwidth traffic that flows from one station to other stations in a structured hierarchical manner [HawK86].

The different application also lead to different operating environments. LANs in office buildings operate within a benign environment. Office building have generally stable temperatures and low electromagnetic (EM) emissions. Factory floors have extremely noisy EM emissions. Campuses provide a mixture of operating conditions. Inside buildings the environment is similar to that of an office building. Between buildings temperature can range widely and EM noise can be large.

4.1.1 Defining Boundaries
System boundaries specify the niche that the LAN technology provider would like to fill. Boundary conditions must be established to guide engineers in the design process. Boundaries are necessary to limit design choices to a spectrum where the product can compete in the marketplace.

4.1.1.1 Cost and Reliability
To compete with existing LAN products, the new product must provide added flexibility at a comparable price or comparable flexibility at a lower price.
Similarly, effectiveness requirements for the new product should equal or exceed the rated effectiveness of existing products. Effectiveness measures for LAN products emphasize reliability and availability. The requirements for cost and reliability should be chosen by surveying the costing and reliability of existing systems.

4.1.1.2 Time to Market

Next, temporal boundaries are set. This new LAN offering needs to enter the market in two years, i.e. the product must pass Gate 4 and enter the utilization and support phase in two years. To reach the utilization and support phase, the product must pass through four other stages: conceptual design, preliminary design, detail design, and production. The twenty-four months available need to be divided among these four phases.

Assuming that a well-understood technology is chosen to implement the new product, engineers should have little trouble producing a detailed design that implements the chosen network scheme. Previously existing production facilities should require little time to retool and produce the final design, again assuming that no radically new technologies are used. This allows most of the twenty-four months to be allocated to conceptual and preliminary design. Making this assignment emphasizes the company's commitment to producing the most cost-effective product possible.

After entering the market, the product is expected to remain in use for fifteen years before becoming obsolete. After the product's market life has expired, there is a short phase out
period. The time line in Figure 4-2 shows the exact distribution of time among the different life-cycle phases. This time line will be used in the next section to aid in predicting product resource requirements and revenues during the life of the product.

![Life-cycle phases of LAN product](image)

**Figure 4-2.** Life-cycle phases of LAN product.

### 4.1.1.3 Resource Requirements

Once the product's life cycle is divided into months, these months can be regarded as fiscal periods in the product's life cycle. The product will consume corporate resources during its life cycle. Predictions must be made about the magnitudes of the resource requirements so that the feasibility of designing, producing and marketing the product can be assessed. These predictions also define the product's predicted future worth to the corporation. The predictions take the form of resource requirement functions as discussed in Chapters 2 and 3. These resource requirement functions are used by the PP/EI DF to determine a company's optimal portfolio of products.

The LAN product has three types of resources associated with it: Budget, Revenue and Staff Hours. When given to the PP/EI DF, these resource requirements are compared with
the resource requirements of other products under consideration. The aggregates of the various possible combinations of products are compared with corporate resource constraints. Sets of products whose aggregate resource requirements exceed the corporate resource constraints are not considered as viable production decisions.

Budget and Revenue represent the direct fiscal impact of the product. Budget represents the cost, in dollars, of the project during each fiscal period. Revenue is the income, in dollars, generated by the product during each fiscal period.

Six functions define the budget for the LAN bridge. At the beginning of conceptual design three engineers will begin working on the project. A growth function is used to represent the cost of the engineers as they finish their previous projects and begin billing hours to this one. These three engineers will remain attached to the product through the end of the production phase. When the product enters the preliminary design phase another engineer will be added to help with the design effort. This engineer remains on the product until the end of the detailed design phase when the work load diminishes. Two more engineers are added at the beginning of the detailed design phase. Both remain until the end of the detailed design phase. A uniform function is used near the end of the detailed design phase to represent the cost of building a prototype system. When the product enters the production phase a decay function is used to represent the cost of producing units. It starts off large, but as time passes and demand lowers, production eventually ends. Finally,
two support staff are added during the production phase through the end of the product's life. Their purpose is to provide technical support to customers.

Two functions make up the revenue curve. When the product enters utilization and support, a growth function represents the income generated by selling units. After time, the product becomes less cost-effective than its competition. A decay function is used to show the declining revenue generated as the product is supplanted in the marketplace.

Staff Hours accounts for personnel dedicated to the product during each fiscal period. Different types of staff may be required during different phases of a product's life. Products in design phases require engineers, while products in production phase require marketing, sales, and technical support staff. PP/EI DF captures this by allowing corporate resource constraints to be made for Staff Hours with respect to the product's current life-cycle phase. When PP/EI DF sums Staff Hours for a given fiscal period it compares the staff requirements of all products currently in a given life-cycle phase with the resource constraint for that phase.

The staff resource follows the personnel model established by budget. Three engineers enter the project during conceptual design. Another is added at the beginning of preliminary design and two others are added during the detailed design phase. The three engineers who began the product remain through the end of the production phase. The
other three leave the product at the end of detailed design. Finally, two support staff are added for the life of the product to provide technical support to customers.

4.1.2 VHDL Model of Definition of Need

The needs that a product will fill must be documented. A VHDL model can aid in this documentation effort. A definition of need VHDL model for the product under discussion is provided in Appendix B.1. It does not document functionality, because there has been none defined. It does document the cost, reliability, and resource requirements that have been determined so far.

The code begins by including several packages whose development was discussed in Chapter 3. The code uses the: PRODUCT_CONSTRAINT, RELIABILITY_FUNCTIONS and PARTSCOST_FUNCTIONS packages. These packages provide subprograms and data types used to deal with product constraints, reliability and cost of parts.

The entity is entitled LAN_PRODUCT. As of yet, the exact nature of the product is undefined. So for the moment it receives a general name. The entity has three sets of generic variables, or generics, associated with it.
The first set of generics relates to product constraints. Each product constraint has two
generic variables associated with it: a list of resource requirement functions for the
product, and a list of resource requirement functions that represent limits imposed on the
product's resource requirements. The limits are the product's predicted requirements. This
is the list of functions that is passed to the PP/EI DF for evaluation at the corporate level.
When the model becomes more complex a list of resource requirements based on the
product's existing design hierarchy is formulated. These computed resource requirements
are compared to the limits to ensure that the current design does not exceed them. The first
generic associated with each requirement lists resource requirements that are global to the
product. Requirements that are not associated with a particular portion of the design like
managers, marketing personnel and secretarial staff should be included in that list.

The second generic is RELIABILITYLIMIT. This generic specifies the minimum
reliability that is acceptable for the product. As a design hierarchy is formed, automatic
reliability estimates are produced and compared to this limit.

The last generic variable, PARTSCOSTLIMITS, specifies the maximum parts cost
acceptable for the product. As estimates for costs of subsystems are included in the design
hierarchy, they are summed over the entire product and compared with this value to
ensure that the product is under its maximum parts cost.
Next in the entity description is the product's port description. The first set of ports deals with product constraints. These signals are connected throughout the design hierarchy with other signals representing the subsystem's resource requirements. Bus resolution functions maintain a complete list of the resource requirement functions generated across the design hierarchy. The signals are initialized to the global product resource requirements defined by the generics discussed above. When a design hierarchy is formed these signals connect the subsystems to the overall system.

The next signal is used for reliability calculations. The MTBF signal maintains the value for the product's mean time before failure, in this case represented by hours to failure. As the product's design hierarchy forms, the MTBFs of the subsystems are automatically combined to determine the MTBF of the overall system. At the moment, since no subsystems currently exist, the signal is simply initialized to an appropriate value.

The last signal is PARTSCOST. This signal is connected throughout the design hierarchy and automatically sums the parts costs of all subsystems, allowing the designers to monitor the fluctuations in system cost as subsystems are designed.

After the port description, the constant LIFE is defined. LIFE defines the expected service lifetime of the product in hours of continuous operation. This value is used in conjunction with MTBF to calculate the system's reliability.
Following the entity's declarative section, a set of OUT functions are invoked in its concurrent section. These functions create data files that advise the designers of the computed values for the resource requirement functions, reliability and parts cost. The OUT functions for the resource requirements produce tables that give the product's resource requirement for each fiscal period in the product's life cycle. The OUTREL and OUTCOST functions simply provide a short message that give the system's computed reliability and parts cost. The outputs generated by the definition of need model are discussed at the end of this section.

Following the OUT functions are a set of ASSERT statements that monitor the product's status with regard to the limits set by the designers. The first set of asserts checks for violations of the product's predicted resource requirements by the product's actual resource requirements. If a subsystem is added to the product whose resource requirements when combined with the rest of the product exceed the predicted resource requirements for some fiscal period, this ASSERT generates a warning message. The next ASSERT monitors the product's actual reliability with respect to the set requirement. If the product's design fails to meet the effectiveness requirement, a warning message is generated. The last assert provides a similar warning if the product's parts cost exceeds the set limit.
The generics are bound in a separate configuration file. Configuration files are provided to allow generics to be bound to different values without having to recompile entire architectures. In this case it allows designers to adjust the values of the resource requirement functions or reliability ratings so that the product meets its goals. The configuration file for the definition of need entity is provided in Appendix B.2.

The configuration file initializes the generics of the model to values discussed in previous sections. Each of the budget resource requirement functions provides a short comment explaining what that function represents. Personnel is budgeted on the assumptions that the cost of devoting an engineer to a project for one fiscal period is $100,000, and the cost of support staff, including technical support people and technical writers, is $50,000. Revenue and staff are estimated as discussed previously. The overall reliability requirement for the product is set to 0.8. The total parts cost of the system is set to $500.

Finally, to compile and simulate the model a test bench is required. The test bench used to simulate the model is included in Appendix B.3. It instantiates an instance of LAN_PRODUCT. There are no required input signals. Simulating the test bench generates the resource requirement tables, reliability, and parts cost. Warnings regarding violation of the resource requirements are produced automatically during simulation.
Figures 4-3, 4-4, and 4-5 are graphs of the resulting resource requirement tables. At this early stage in the life cycle, the VHDL model does not contain a subsystem hierarchy to generate requirements, so in the graphs the requirement curves are identical to the limit curves. Output from the reliability and parts cost are also just the values assigned as limits since no hierarchy exists. The reliability and parts cost output routines generated the following output:

System Reliability for Lifetime of 1.500000e+05 million hours is:
8.007374e-01
Parts Cost: 4.000000e+02

**Figure 4-3.** Budget requirements predicted during definition of need phase.
Figure 4-4. Revenue requirements predicted during definition of need phase.

Figure 4-5. Staff requirements predicted during definition of need phase.
4.2 Conceptual Design

The conceptual design process provides a means to the end specified by the definition of need. The process begins with feasibility studies that determine the possible technologies that can be applied to the need. In this case, the feasibility studies lead to a single option that satisfies all stated requirements from the definition of need. This simplifies the process somewhat, since it becomes unnecessary to pursue multiple design paths. Once this path is identified, system operational requirements are defined. System operational requirements produce a specific definition of how the final system should behave. Finally, advance system planning generates an algorithmic VHDL description of the product.

4.2.1 Feasibility Studies

The first question to ask is "Does the technology to solve this problem currently exist?" In terms of the case study this question is "Can currently existing LAN products provide the performance and flexibility required to satisfy the LAN customer as defined?" The following sections cover currently existing LAN technologies and examine the advantages and disadvantages of each.

4.2.1.1 Media Access Methods

The first avenue of investigation leads to LAN media access methods. A LAN's media access method describes how the nodes share the transmission media interconnecting them. There are three important considerations to note while evaluating these techniques. A LAN's extent is the longest propagation path between two nodes along the transmission
media. The LAN product being designed requires a large extent. The campus implementation described earlier requires an extent around 25 kilometers.

The campus implementation also requires a large number of network stations. Network stations, or nodes, are computers or terminals that access the network. The LAN product being designed must support on the order of thousands of nodes. Cost is also an issue. When the number of nodes on the network is in the thousands, the cost of each node's interface to the transmission media needs to be low.

Based on these issues the question to answer is "Can a currently existing single LAN provide the flexibility required?" The following sections survey currently available network technology. A brief description of the media access method is provided along with an evaluation of its usefulness for designing the LAN product under consideration.

4.2.1.1 Carrier Sense Multiple Access with Collision Detection (CSMA/CD)
When a CSMA/CD node wishes to transmit it listens for a carrier to see if another node is currently transmitting. If no carrier is present, the node begins transmitting. It is still possible that another node was listening for no carrier and decided to transmit at approximately the same time. This creates a collision as both nodes simultaneously transmit. A collision detection mechanism is used to sense when this has occurred and resend the lost transmission [ComD91].
CSMA/CD performance does not degrade quickly as nodes are added to the network. Unfortunately, finite propagation delays along the transmission media limit the physical extent of the LAN. A LAN's extent refers to the longest physical distance between nodes along the transmission media. If the distance is large enough to introduce propagation delays that are longer than the total time necessary to transmit a packet, then it is possible for two nodes at opposite ends of the LAN to transmit their packets without being able to sense a collision. So, to increase the extent, either the minimum packet size must be increased or the transmission rate must be decreased. Both of these options reduce LAN performance [HawK86].

4.2.1.1.2 Carrier Sense Multiple Access
CSMA LANs function like CSMA/CD LANs but without collision detection. Without physical layer collision detection, the LAN must depend on high-level protocols to detect any transmission errors. This increases delay variance across the network as high-level protocols wait for time-outs caused by collisions. Increasing the transmission rate can decrease the probability of collisions and thus reduce delay variance, but this significantly increases the cost of each node since it requires faster hardware [HawK86].

4.2.1.1.3 Token Passing Bus
On a token passing bus only the node that currently has the token can transmit. When a node on a token passing bus wishes to transmit, it needs to "own" the bus's token. The token is passed around the bus and shared equally among the nodes. Since only one node
ever has the right to transmit on the bus, there is no chance for a collision to occur. This means that the LAN's performance does not decrease considerably with physical extent. However, since in the worst case a station that wishes to transmit may have to wait for every other station on the network to pass the token, the network's performance does decrease considerably as large numbers of nodes are added to the network [HawK86].

4.2.1.4 Token Ring
The token ring LAN is similar to the token passing bus LAN. It encounters the same limitations in that as the number of nodes attached to the network increases performance degrades. The token ring bus is also somewhat more sensitive to extent than the token passing bus [HawK86].

4.2.1.5 Time Division Multiplexing (TDM) Bus
The TDM bus allocates to each node on the network a time-slice of the available transmission bandwidth. This allows the bus to avoid collisions which increases the extent of the network. The time-slice is allocated to nodes whether they use it or not. This means that the delay between when a node wishes to transmit and its next window on the bus may be large. It also means that stations are limited to a fixed amount of data throughput. If one station has a large quantity of data to transmit, it takes just as long to transmit if the other stations are idle as it does if they all wish to transmit [HawK86].
4.2.1.6 Frequency Division Multiplexing (FDM) Bus

A FDM bus uses different frequencies for each station. It has the advantage that it allows different stations to be assigned more or less bandwidth depending on their assigned frequency. The disadvantage is that every station must monitor every frequency channel for packets addressed to it. The cost of the node interfaces becomes prohibitive as the number of nodes on the network increases [HawK86].

4.2.1.2 LAN Interconnection Alternatives

Unfortunately no single LAN technology is capable of providing the extent, stations and cost required by the three environments to be supported. Another alternative is to interconnect individual LANs to provide a larger extent and greater number of stations. The important considerations here are cost and routing protocol. The diversity of customers being served by the theoretical LAN product may use a variety of nonstandard protocols on their systems. This requires the new LAN product to transparently support nonstandard protocols between stations. Two strategies for LAN interconnection are presented below. The distinguishing feature of LAN interconnection strategies is the protocol layer at which they choose to interconnect.

4.2.1.2.1 Router

A router is an intelligent network device that interconnects LANs at the network layer. It utilizes standard high-level protocols to determine the best possible routes through the networks it is connected to. This is the technology used to implement the worldwide Internet [ComD91]. The Internet is an example of interconnected networks that increase
the number of stations; over a million stations are connected to the Internet [ComD91]. Unfortunately, it requires the use of a standard routing protocol, usually implemented at the network layer. This requirement disqualifies router technology from consideration since it does not support proprietary customer routing protocols.

4.2.1.2.2 Bridge
A bridge connects LANs together to form an extended LAN. The bridge listens to all data frames that occur on both LANs and forwards them to the opposite side. The bridge performs intelligent filtering on the data frames so that only frames whose data link layer destination address is on the opposite side of the bridge cross. This effectively isolates traffic internal to each network, but allows communication with nodes on other networks. The bridge allows packets from different network and higher level protocols to pass successfully since it performs address filtering at the data link layer. It also allows differing LAN technologies to be intermixed, which in turn allows networks to be optimized for certain performance characteristics. Bridges appear to be simply another network station to the LANs which they interconnect. This means that differing LANs can be used together to overcome each other's shortfalls in terms of extent and number of nodes. For instance a token bus which has a large extent, but is sensitive to the number of nodes attached to it, could be used to interconnect a series of bridges that are physically widely separated. The bridges in turn would be connected to CSMA/CD LANs that are sensitive to large extents, but which scale well with increasing numbers of nodes. The CSMA/CD
LANs would connect to the network stations and thus network users. The CSMA/CD
LANs are interconnected through the bridges, which are interconnected by the token bus.
There is only one bridge on the token bus for each LAN in the extended LAN, so the
token bus's sensitivity to large numbers of nodes does not become a performance penalty.

4.2.1.3 Choosing a Direction
Having surveyed the available technology, it is clear that no single LAN is capable of
satisfying the needs of the theoretical LAN product due to the large extent and large
number of nodes required. The router will not allow transparent operation to the network
stations since it cannot support nonstandard routing protocols. That leaves the bridge.

The bridge allows nonstandard routing protocols since it makes forwarding decisions
based on physical addresses instead of higher-layer addresses. Bridges allow
interconnection of LANs to provide a significant increase in the total number of nodes on
the extended LAN and in the physical extent which it can support. Clearly baring the
investigation of some radical new technologies, forbidden by earlier assumptions, the
extended LAN is the choice for the new LAN offering.

4.2.2 System Operational Requirements
Having made the decision to pursue the extended LAN architecture, exact definitions of
the functionality of components must be provided. Single LAN technologies already exist.
Improvements in single LAN technology are not necessary for the purposes of the
extended LAN. Therefore, the remainder of this case study devotes itself to studying the design of the bridge. To define the bridge's functionality a mission description is required.

4.2.2.1 Mission Description
In normal operation the bridge "listens" to all data frames being transmitted on both LANs that it is connected to. The bridge has a forwarding table that is a list of node addresses associated with the LAN that contains them. When a data frame is received on one LAN, the bridge checks to see if the destination address is in the forwarding table. If the address is in the forwarding table, the bridge checks the associated LAN. If the associated LAN is not the same LAN that sent the frame, the bridge forwards the frame on the associated LAN. If the associated LAN is the same LAN that sent the packet, no action is taken. If the destination address is not in the forwarding table the bridge forwards the frame since it has no way of knowing which side the destination station is on. Multicast addresses are similarly forwarded since the bridge cannot know which stations are to receive the frame.

The forwarding table is maintained while the bridge is in normal operation. When frames are received, the source address and the LAN from which it was received are added to the forwarding table. If the source address is already in the table, the bridge simply checks to insure that the station that sent the message has not been moved to a different LAN. If it has been moved the bridge modifies the forwarding table appropriately. The table is a caching system. For the bridge to react to changes in the locations or presence of stations, addresses that have not been heard from for a time are "aged" out of the list.
During initialization just after power is applied to the bridge, the forwarding table is empty. Since the forwarding algorithm forwards frames whose address is not in the table, this means that the bridge forwards all frames that it receives. To prevent this unnecessary congestion the bridge does not immediately begin forwarding frames. It waits for a period of time and passively listens to the LANs connected to it. While doing so it accumulates a cache of forwarding addresses. When it is finished learning, it begins normal operation.

The bridge maintains the integrity of data frames. When a bridge forwards a frame between two similar LANs the data frame is passed unmodified. When a bridge forwards a frame between two dissimilar LANs the incoming data frame is reformatted to the frame specification for the destination LAN. This allows the bridges in the extended LAN to remain transparent to the network stations.

The bridge must prevent looping in the extended LAN. Forwarding messages around a loop would quickly saturate all the LANs in the loop. To prevent this, the bridges in the extended LAN communicate with each other to insure that the topology is logically a tree. A tree topology prevents looping. This strategy also allows redundancy in the extended LAN. Availability is extremely important in a LAN. Having redundant bridges in the LAN insures that if one bridge goes down, another can almost immediately take over. The bridges communicate to insure that the tree topology is maintained. If a fault is detected, the bridges immediately reorganize into a new tree using a redundant bridge.
4.2.2.2 Performance Goals
Bridge performance is an important issue in extended LAN design. To maintain performance similar to a single LAN across the extended LAN, the bridges used to join LANs must not constitute bottlenecks. The figure below illustrates the possible areas of congestion within the bridge.

![Bridge Diagram]

**Figure 4-6.** Sources of bridge congestion.

Congestion can occur in the receiving buffers, the forwarding queue or the transmitting buffers. To eliminate congestion at the receiving and forwarding queues, the forwarding operation must take less time than the sum of the interarrival times of inbound LAN traffic [HawK86]. Therefore processing speed of incoming packets is an important performance criteria.
Congestion at the outbound LAN connections is determined by the availability of the outgoing LAN. If a packet is queued for transmission on the outgoing LAN, and the LAN is crowded with local traffic, the packet must wait. If the traffic on the outgoing LAN is extremely high, then packets waiting for transmission will eventually exhaust the queue and the bridge will be forced to drop packets. No amount of clever bridge design can eliminate this type of congestion. It is caused by limitations of the underlying LAN, not the bridge [HawK86].

The bridge should not be a source of bit errors across the extended LAN. Individual LANs have low bit error rates. Some high-level protocols rely upon this assumed accuracy. Introduction of bridges between the individual LANs should not significantly increase this bit error rate.

### 4.2.2.3 Effectiveness Requirements

Reliability and availability are the important effectiveness metrics for designing an extended LAN. When a bridge forms the information link between different parts of a company, loosing the link can be costly. Reliability is a general measure of how often a company can expect to loose the link. Availability measures the percentage of time that the extended LAN is available for use. Following the same design principle as before, the reliability and availability of the extended LAN should not be significantly different from the reliability and availability of the underlying individual LANs.
4.2.3 Advance System Planning
The purpose of advance system planning is to bring together all the information generated by the conceptual design phase into a coherent product description. A well-crafted algorithmic model of the proposed system can fulfill this goal.

4.2.3.1 Algorithmic Bridge Model
An algorithmic model of the bridge summarizes all of the information that was gathered in the operational requirements analysis. It provides a model for comparison with later design stages. The model captures functionality in its algorithmic description of the system's behavior. It captures performance issues in its timing response. VHDL modeling of these issues is well understood and documented completely in other sources [ArmG93].

4.2.3.2 Life-Cycle Completeness Issues
The life cycle issues modeled by the conceptual design model are essentially the same as the issues modeled in the definition of need model. Since the model still supports no hierarchy, the model requires no new elements. The model will change to reflect issues discovered during conceptual design. The product under development is now understood to be a single device, a bridge, instead of a complete LAN system. This revelation will affect on the life-cycle issues already being modeled.

Since the system in question is now just a bridge and not a complete LAN system, its failure can be readily detected. This shortens the diagnosis period for extended LAN
failure and thus reduces the MTTR of the system. Lowering the MTTR means that a higher availability can be expected from the system. Similarly since the scope of the system is reduced to failure of a single device the MTBF of the bridge can be increased, and a higher reliability expected.

Resource requirements may also be affected. The scope of the design effort and potential market for the product are much better understood now. This may have dramatic effects on the required budget and the predicted revenue. Limiting the scope of the design to a single device could lower the staff hours requirement significantly. All these modifications can be made to the existing definition of need model by altering the configuration file that instantiates its generic variables.

4.3 Preliminary Design
Preliminary design examines the operational behavior defined by conceptual design and produces a functional description of the device's operation. This functional description is then mapped onto subsystems that compose the system. Design criteria are then allocated among the subsystems so that the aggregate of the subsystems meets the requirements placed on the whole system. Finally a structural model of the system is generated for evaluation and optimization.
4.3.1 Functional Analysis

The purpose of functional analysis is to identify all the operations that a system must perform to satisfy its mission. This analysis creates a list of functionality that the system must provide. Once the analysis is complete, each of the enumerated functions must be allocated to a piece of hardware that is capable of performing the operation.

![Functional Flow Diagram]

**Figure 4-7.** Functional flow diagram of normal bridge operation.
Figure 4-7 illustrates a functional analysis of the bridge using a functional flow diagram. Functional flow diagrams are explained in detail in [BlaF90]. The functional flow diagram illustrates functional behavior for traffic going one way across the bridge during normal operation. The functional behavior of traffic going the opposite direction is the same with LAN A and LAN B switched. For simplicity, the diagram does not attempt to document the functionality of higher level bridge functions like bridge maintenance or the distributed spanning tree algorithm.

As the name indicates this diagram indicates function only. It does not wholly specify operation. Note, for instance, that there are no conditions given for the or branches in the diagram. The diagram simply indicates that two branches are possible, not how to pick which branch should be executed. The next step is to provide devices capable of providing the functionality specified in the diagram.

### 4.3.2 Preliminary Synthesis

The purpose of preliminary synthesis is to allocate functional requirements identified in the system functional analysis process to hardware devices. Block 2, "Wait for frame on LAN A," and block 5.3, "Transmit frame on LAN B," require LAN physical layer interfaces. To prevent the bridge from being a bottleneck, these devices need to be able to read incoming frames as fast as they arrive. Therefore, since both LANs can receive and transmit frames simultaneously, two such interfaces are required.
The remaining functions are purely digital processing operations. So far there have been no requirements on the processing functions that require any of them to occur in parallel as long as they can be executed fast enough. In general, a microprocessor or microcontroller provides an extremely cost-effective method for implementing multiple process and control functions. So, the remaining functions are allocated to a microprocessor with the understanding that the processor must be able to perform the operations within real-time constraints.

By introducing a microprocessor several other devices are implicitly added as well. Microprocessors require program and data memory. Since the bridge is required to operate independently of other devices, its program must be permanently stored inside it. This requires a ROM device for program storage. The data storage requires RAM. Two more devices are added to the list. Adding a bus to interconnect all the devices creates the initial block diagram of the bridge in Figure 4-8.

Now that the hardware devices for the system have been identified, the design criteria that were allocated at the system level must be distributed among the newly created hierarchy.

4.3.3 Allocation of Design Criteria
There are three design criteria to allocate among the subsystems: parts cost, resource requirements and reliability. Each subsystem consumes part of the whole system's budget
for each criteria. Some subsystems will have a larger requirement than others. Analysis of the possible distribution of criteria budgets among the subsystems creates a better understanding of how each subsystem needs to be designed.

![Diagram showing Microprocessor, Program Memory (ROM), Data Memory (RAM), Ethernet Interface, and Ethernet Interface connected via LAN A and LAN B.]

**Figure 4-8.** Initial block diagram of bridge [HawK86].

### 4.3.3.1 Parts Cost

The total system parts cost is $500. Parts cost is the simplest of the resources to allocate. At this point in the design process, the criteria must be divided among four hardware subsystems. There are, however, two other places that to which cost must be allocated. When the hardware subsystems are designed and fabricated, they must be mounted in a box of some sort. The cost of the box must be included in the total parts cost. When the bridge is shipped to customers it must include a user's manual. The cost of printing and binding these manuals is also included in the parts cost. The block diagram in Figure 4-9 indicates the initial allocation of parts cost.
Ethernet interfaces are prevalent in the market place. Most likely, an existing device will handle almost all of the physical interface requirements. Two are required. An estimate of $200 each indicates a subsystem cost of $400. Based on knowledge of the microprocessor market, a processor capable of performing the functions required by the bridge should cost under $50. The cost of the RAM is somewhat uncertain. The cost of RAM is volatile, and so far no decision has been made to fix the size of the RAM required. To be safe the cost of the RAM and memory controller is assumed to be under $10. ROM has to be custom fabricated, but the process is not that expensive, so $5 is allocated for the cost of the program ROM. The metal box, cooling fan and power supply constitute mountings and are estimated to cost $25. The cost of printing and binding the manuals is around $10.

4.3.3.2 Reliability
Allocation of reliability among subsystems is more complicated. The reliabilities of systems in parallel are products not sums. Intuitively it is difficult to understand what it means to
divide reliability among subsystems. MTBF is also difficult to divide since it adds reciprocally. However, the dual of MTBF, the failure rate, is additive. Since the lifetime of the product is set, the overall MTBF of the system, and therefore the overall failure rate, can be calculated directly from reliability. Assuming that failure times are exponentially distributed, the failure rate, designated \( \lambda \), is related to reliability by the equation \( R(t) = e^{-\lambda t} \) where \( R \) is reliability and \( t \) is the life of the system [JohB89].

Once an overall failure rate is established, it can be allocated by assigning a percentage of the overall failure rate to each subsystem. The ethernet connection consumes the most power of any subsystem in the bridge. Assume that the ethernet connection is responsible for 30 percent of system failures. The microprocessor system is responsible for 20 percent of system failures. That leaves the ROM and RAM systems, which are each responsible for 25 percent of the overall failures. The calculation below illustrates the allocation of reliability based on these figures.

First, the overall failure rate is calculated,

\[
\lambda = \frac{\ln(R)}{-t} = \frac{\ln(0.80)}{-150000} = 1.4876e^{-6}.
\]

then the failure rates of the individual systems are calculated from it based on the allocated percentages discussed above.
\[
\begin{align*}
\lambda(.30) &= 446.29e - 9 \Rightarrow R_{Ether} = .95 \\
\lambda(.20) &= 297.52e - 9 \Rightarrow R_{MP} = .935 \\
\lambda(.25) &= 371.91e - 9 \Rightarrow R_{RAM} = .94 \\
\lambda(.25) &= 371.91e - 9 \Rightarrow R_{ROM} = .94
\end{align*}
\]

Figure 4-10 illustrates the allocation of reliability among the subsystems of the bridge.

![Figure 4-10. Allocation of reliability.](image)

4.3.3.3 Resource Requirements

The subsystems of the bridge do not consume a specific part of the budget, nor do they generate specific portions of the revenue. The budget and revenue functions can be left safely allocated at the system level. The subsystems do require staff.

At the beginning of the conceptual design phase three engineers are available. One of them is the product manager. The product manager's responsibility is to supervise the other engineers and provide life-cycle complete analysis of the product. The two remaining engineers are allocated to the microprocessor subsystem and the ethernet subsystem.
When the fourth engineer arrives during the beginning of preliminary design, that engineer is allocated to the RAM subsystem. At the beginning of detailed design two more engineers arrive. The first is assigned to the microprocessor subsystem as a programmer. The second remains allocated at the system level and is able to lend a hand where necessary. At the end of detailed design only the original three engineers remain assigned to the product. They remain until the product enters the utilization and support phase. At the beginning of production, two support staff are assigned. During the production phase, these two staff members are assigned to the manuals subsystem to write the manuals. At the beginning of the utilization and support phase, these two staff members are assigned back to the system level, where they provide technical support for the product.

### 4.3.4 VHDL Model for Preliminary Design

A VHDL model of the preliminary design serves to document the results of the preliminary design phase. This model has a hierarchy of subsystems. Each of the subsystems is implemented as an algorithmic VHDL model. The subsystems are interconnected into a structural model of the bridge. The model documents both the basic block diagram of the system and the interconnection between the blocks. Simulation of the model verifies the functional interconnection of the subsystems.

The preliminary design model listed in Appendix C demonstrates the VHDL model's usefulness for modeling the life-cycle issues of preliminary design. The BRIDGE entity is similar to the previous LAN_PRODUCT entity. The primary difference is the presence of
the subsystems. In the LAN_PRODUCT entity all the design criteria were assigned at the LAN_PRODUCT level. The BRIDGE entity has some design criteria assigned to it, but it also receives requirements for design criteria from the subsystems below it.

Each subsystem is implemented as a separate entity. The subsystem entities are similar to the BRIDGE entity itself. Since there is only one layer of hierarchy in the preliminary design, each of the subsystem entities has all of its design criteria assigned at its own level. When the VHDL model is initialized, each subsystem entity calculates its own design requirements and sends them to the BRIDGE entity above. The results are compared to the previously set limits for each subsystem. If the limits are exceeded, assertion warnings are generated to inform the modeller. When the BRIDGE entity receives all the design criteria requirements it combines them and determines if the aggregate requirement exceeds the limits placed on the bridge as a whole. If the requirements exceed the limits, assertion warnings inform the designer.

Appendix C.1 is a listing of the VHDL description of the BRIDGE entity. Appendices C.4-C.9 list the VHDL code for the various subsystems. Appendix C.2 lists the configuration file for the preliminary design model. This configuration file contains all the limits and design criteria requirements for the BRIDGE entity and all the subsystems. Appendix C.3 lists the test bench used to simulate the model.
The model can provide detailed feedback about the allocation of criteria among subsystems. Each subsystem in the preliminary design model includes an OUTPARTSCOST and OUTREL function call. When the model is simulated it produces a summary of the allocation of parts cost and reliability among BRIDGE's four subsystems. The system's overall parts cost and reliability are given on lines beginning with the label PRELIM. The summary for the BRIDGE preliminary design model is:

MICROPROCESSOR Parts Cost: 5.000000e+01
ETHERLINK Parts Cost: 4.000000e+02
RAM Parts Cost: 1.000000e+01
ROM Parts Cost: 5.000000e+00
MANUALS Parts Cost: 1.000000e+01
MOUNTING Parts Cost: 2.500000e+01
PRELIM Parts Cost: 5.000000e+02
MICROPROCESSOR System Reliability for Lifetime of 1.500000e+05 million hours is: 9.602703e-01
ETHERLINK System Reliability for Lifetime of 1.500000e+05 million hours is: 9.439404e-01
RAM System Reliability for Lifetime of 1.500000e+05 million hours is: 9.439404e-01
ROM System Reliability for Lifetime of 1.500000e+05 million hours is: 9.439404e-01
PRELIM System Reliability for Lifetime of 1.500000e+05 million hours is: 8.076573e-01

Figures 4-11, 4-12, and 4-13 are graphs of the resource requirement tables generated by the BRIDGE preliminary design model. The preliminary design model computes the requirements based on the aggregate requirements of the subsystems. The results shown in the figures indicate that currently the system's overall resource requirements are below the budgeted values.
Figure 4-11. Budget requirements for preliminary design phase.

Figure 4-12. Revenue requirements for preliminary design phase.
4.4 Detailed Design
Detailed design maps the functionality of the subsystems specified in earlier stages onto real devices. Lengthy coverage of a complete detailed design for the bridge is beyond the scope of this case study. The life-cycle complete principles being investigated still apply during this phase. The discussion below concentrates on those issues.

4.4.1 Feedback from Detailed Design
As design moves forward and the algorithmic models specified in previous phases are replaced with models of real devices, two problems in the design of the bridge become apparent. First, the bus interconnecting the microprocessor to the memory and the ethernet connections constitutes a bottleneck. The speeds that are required are impractical
for a single bus. Second, the time required for the microprocessor to conduct the address search is too lengthy. No suitable processor is capable of performing the search within the time allowed. These two problems are addressed below.

4.4.1.1 Bus Bottleneck
Fixing the bus bottleneck requires some restructuring of the bridge's internal architecture. The RAM device needs to be restructured into a four-port memory device, with one port each for the two ethernet devices, one for the microprocessor and one for the dynamic RAM's own refresh cycle. This also means that the RAM used by the processor for internal operations must be located elsewhere.

The software development effort has revealed that there are certain software dependent values that need to be saved when power is lost. To solve this problem and relocate the processor's RAM, the ROM subsystem is reconfigured to contain ROM for the program memory, RAM for internal processing and non-volatile RAM (NVRAM) for storing values when power is lost. The ROM subsystem is appropriately renamed DATASPACE.

4.4.1.2 Address Search
The processor requires some sort of hardware assist to accomplish the search in an acceptable amount of time. Another issue arises here as a result of the bus bottleneck problem. The addresses cannot be stored in the RAM unit, since the search algorithm would cause the RAM unit to again be a bottleneck. A new subsystem must be created
that contains the hardware to conduct the search and that contains the RAM for storing the addresses. The new subsystem is called SEARCH. At the moment it does not have any resources allocated to it. Resources from another must be reallocated to supply this new subsystem. It is possible that other subsystems may have resources requirements less than originally predicted. Such resources could be used to supply the new subsystem.

Initially, content addressable memories (CAMs) are investigated as a means to solve the search speed problem. Unfortunately, these devices are designed to be used in high-speed memory caching systems, which renders them must faster than necessary for the address search. Initial simulations indicate that the CAMs are simply too expensive to implement as the hardware assist. This leaves special-purpose digital logic.

Simulations indicate that a register-level transistor-transistor logic (TTL) implementation of binary search is sufficiently fast for the bridge's searching mechanism. TTL parts are highly reliable and relatively inexpensive. Therefore, SEARCH is implemented as a TTL binary search circuit. A new block diagram of the system implementing both the bus solution and the search solution are shown in Figure 4-14.
4.4.2 VHDL Model for Detailed Design
The model is similar to previous models. The difference is that some subsystems now themselves have subsystems. The behavior of the "subsubsystems" is identical to that of the subsystems. Each subsubsystem creates an aggregate requirement for itself and passes it up to the subsystem above it.

Some of the subsystems are mapped as actual parts. These subsystems differ from the previous subsystems. The subsystems that are mapped as actual parts do not have resource requirement functions. Since actual parts are the final step in the design process, they do not require any resources; they are complete. They do have parts costs and MTBFs associated with them. These connections to the systems above are still in place. As more and more parts are mapped as actual devices, the system-wide parts cost and reliability calculations become more accurate. Finally, when all the subsystems are
designed with actual devices, final values for parts cost and reliability are automatically calculated.

The detailed design model inherits most of its structure from the preliminary design model. As discussed, the ROM subsystem becomes the DATASPACE subsystem and a new subsystem, the SEARCH subsystem, is added to the hierarchy. The DATASPACE subsystem inherits all the resources of the ROM subsystem. The SEARCH subsystem, however, initially does not have any resources allocated to it. Simulation of the detailed design model produces an appropriate assertion error:

```
Assertion ERROR at 0 NS in design unit DETAIL from process /TB/C1/C5/_P4: 
"Search Logic has exceeded its parts cost."
```

An examination of the other subsystems reveals that the ETHERLINK subsystem has been implemented with a VLSI chip set that costs $80. There are two such chip sets in the ETHERLINK subsystem, which makes the total parts cost of the ETHERLINK subsystem $160. That leaves $40 dollars of unused parts cost. The SEARCH subsystem only costs $10 to implement, so $10 of this excess from the ETHERLINK subsystem is reallocated to the SEARCH subsystem. This solves the parts cost problem. However, the next simulation run produces the assertion violation:

```
Assertion ERROR at 0 NS in design unit DETAIL from process /TB/C1/_P11: 
"Bridge has failed to meet reliability."
```
The simulation run produces a summary of the parts cost/reliability allocation among subsystems that can be used to understand what is causing this assertion violation. The summary produced is:

MICROPROCESSOR Parts Cost: 5.000000e+01
ETHERLINK Parts Cost: 1.600000e+02
RAM Parts Cost: 1.000000e+01
DATASPACE Parts Cost: 9.900000e+00
SEARCH Parts Cost: 1.000000e+01
MANUALS Parts Cost: 1.000000e+01
MOUNTING Parts Cost: 2.500000e+01
DETAIL Parts Cost: 2.749000e+02
MICROPROCESSOR System Reliability for Lifetime of 1.500000e+05 million hours is: 9.602703e-01
ETHERLINK System Reliability for Lifetime of 1.500000e+05 million hours is: 9.439404e-01
RAM System Reliability for Lifetime of 1.500000e+05 million hours is: 9.439404e-01
DATASPACE System Reliability for Lifetime of 1.500000e+05 million hours is: 9.402787e-01
SEARCH System Reliability for Lifetime of 1.500000e+05 million hours is: 9.700323e-01
DETAIL System Reliability for Lifetime of 1.500000e+05 million hours is: 7.804145e-01

Again, the problem is the introduction of the new SEARCH subsystem. The preliminary design model already shows that the system is operating on the edge of its reliability requirement. Examining the preliminary design reliability summary reveals that the system's overall reliability is only 0.807, only slightly above the system requirement of 0.8. To maintain the overall system reliability requirement any new system added to the hierarchy would require a reliability very close to 1.0. The SEARCH system's reliability is 0.97 and it is designed using TTL logic, which is highly reliable, so the SEARCH system cannot be easily redesigned for higher reliability. This means that the extra reliability should come from one of the other subsystems.
Examining the ETHERLINK subsystem reveals an alternate implementation of the ETHERLINK using a different chip set that has a higher reliability. This alternative was originally discarded because the less reliable chip set was less expensive and still satisfied the ETHERLINK's reliability requirement. The more reliable chipset costs $95, but the resulting ETHERLINK subsystem has a reliability around 0.97. Simulating the bridge model with the new ETHERLINK subsystem produces the following system summary:

MICROPROCESSOR Parts Cost: 5.000000e+01
ETHERLINK Parts Cost: 1.900000e+02
RAM Parts Cost: 1.000000e+01
DATSPACE Parts Cost: 9.900001e+00
SEARCH Parts Cost: 1.000000e+01
MANUALS Parts Cost: 1.000000e+01
MOUNTING Parts Cost: 2.500000e+01
DETAIL Parts Cost: 3.049000e+02
MICROPROCESSOR System Reliability for Lifetime of 1.500000e+05 million hours is: 9.602703e-01
ETHERLINK System Reliability for Lifetime of 1.500000e+05 million hours is: 9.704456e-01
RAM System Reliability for Lifetime of 1.500000e+05 million hours is: 9.439404e-01
DATSPACE System Reliability for Lifetime of 1.500000e+05 million hours is: 9.402787e-01
SEARCH System Reliability for Lifetime of 1.500000e+05 million hours is: 9.700323e-01
DETAIL System Reliability for Lifetime of 1.500000e+05 million hours is: 8.023281e-01

This new system design simulates without errors. The redesign of the ETHERLINK subsystem has exceeded its own reliability requirement, which is 0.94, but the higher reliability allows the SEARCH subsystem to be included without exceeding the global reliability requirement. The tradeoff is parts cost. The new ETHERLINK subsystem costs $30 dollars more than the original, and raises the system's parts cost by $30.
The code for the VHDL model of the detailed design can be found in Appendix D. The model for the bridge is in Appendix D.1. The configuration file for the detailed design model is in Appendix D.2. The test bench used for simulation is in Appendix D.3. The code for the various members of the design hierarchy can be found in Appendices D.4-D.12. Figure 4-15, 4-16, and 4-17 are graphs of the resource requirements tables produced by the detailed design model.

Figure 4-15. Budget requirements for detailed design phase.
Figure 4-16. Revenue requirements for detailed design phase.

Figure 4-17. Staff requirements for detailed design phase.
4.5 Chapter Summary

This chapter presented a case study of a LAN bridge design that demonstrated the principles discussed in Chapter 3. The chapter discussed the different phases of the design process and the applications of VHDL within each. The discussion starts with definition of need and continues through the basics of detailed design. The sections evolve a VHDL model of the LAN bridge as it passes through the phases of its design. The model developed here concentrates on modeling life-cycle complete design issues. The source code for the various models can be found in the appendices.
5. Summary, Contributions, and Extensions
This chapter summarizes the work presented and its contributions. Conclusions concerning practical implementations of the CAE tools are discussed and suggestions about future research are also provided.

5.1 VHDL as a CAE Tool for Life-Cycle Complete Engineering
Classical design techniques tend to ignore some issues of a product's life cycle until the opportunity to change the design to accommodate those issues has passed. By studying the behavior of product development, life-cycle complete design anticipates costs that occur throughout a product's life-cycle and recognizes design issues that can minimize these costs.

Corporate executives can optimize the competition between products for scarce resources within the corporation. By defining corporate resource constraints and resource requirements for individual product life cycles, executives can study the interaction of the product's resource requirements over a period of interest. The results of this study can produce an optimal product funding decision that maximizes the future worth of the corporation's product portfolio.

Since monitoring all of the information necessary for life-cycle engineering can become intractable for humans, it is useful to have computer-aided engineering tools that simplify the task. VHDL has been used effectively to simulate and document the functional aspects
of digital hardware design [ArmG93,ArmJ89]. It is desirable to use the same representation and associated tools to simulate and document the product's life-cycle behavior. This would provide a common form of documentation for all information regarding a product.

VHDL signals can be defined that represent the allocation of design criteria in a system rather than the flow of digital signals. By appropriately defining bus resolution functions for these signals, they can be used to document and calculate the life-cycle design metrics of a system. Three such signals were examined in this research: parts cost, MTBF and resource requirement functions. A conceptual framework for interconnecting the signals to provide information on design criteria at the subsystem and system levels was developed. Methods for automatically checking the design metric requirements of each subsystem were also discussed.

A case study was developed using the principles presented. The case study covered the different phases of the design process and the applications of VHDL within each. The case study started with definition of need and continued through the basics of detailed design. The case study developed a VHDL model of a local area network bridge and demonstrated the evolution of the model as it passed through the design phases.
5.2 Research Contributions

This research originated with the author's participation in the design and implementation of the Product Portfolio/Enterprise Integration Demonstration Facility (PP/EI DF). The author implemented the user interface and a portion of the computation engine. This work identified the need for CAE tools to support the PP/EI DF by automatically gathering design information from lower levels of the design process.

Examination of the requirements for CAE tools that could perform this function suggested VHDL as a common solution for both simulating the functional behavior of digital hardware and calculating life-cycle information. This research has led to a framework for modeling life-cycle behavior. The framework provides automated calculation of product life-cycle information and demonstrates principles and techniques that can be used to extend the framework to model information that is not explicitly discussed here.

Designers using the framework are provided with feedback about life-cycle issues at the same time that they are provided with functional feedback. When designs are simulated the output contains not only the functional behavior of the system but also the associated life-cycle parameters. Each module or device added to the design contributes to the overall system cost, reliability, and resource requirements. This provides the designers
with a direct understanding of how the functional implementation of the design affects life-cycle issues.

5.3 Further Research

The VHDL code presented here is tractable only for simple examples. Hand coding of the life-cycle criteria allocation configurations and subsystem links would quickly become a source of errors for a large-scale project. The code is highly repetitive and ordered which makes it an ideal candidate for automation.

A practical automated system for applying these techniques would have several major components. At the highest corporate level, executives would require a user interface that allows them to define corporate resource constraint functions and analyze product resource requirement functions to produce an optimal product portfolio. The PP/EI DF is an example of a prototype implementation of such a user interface.

Product managers would require a user interface that allows them to create product resource requirement functions that feed into the executive decision facility described above. These functions also would feed into the automated VHDL modeling tool as the resource requirement limits discussed in Chapter 3.
Product designers need an interface that allows them to graphically define a system hierarchy. This tool would also aid in the allocation of design criteria among subsystems. The results of this tool would feed into the automated VHDL code generation tool. The code generation tool would produce a VHDL shell that contains code similar to that described here. The tool would automatically setup the design hierarchy and signal interconnections.

The last step would be to create the configuration file that binds actual values to the generic variables. A centralized database the stores cost and reliability information about the parts library would provide values associated with the actual parts used in the model. Integrating these tools would provide a powerful set of CAE tools for life-cycle complete design.

An obvious avenue for further study would be to implement one or more of the tools described above. The PP/EI DF is relatively primitive in its current state; many enhancements are possible that would increase its flexibility and functionality. The user interface for designing the hierarchy could implemented using a proprietary extendible CAD package. The Mentor Graphics package has elaborate extendibility with its AMPLEx programming language. Implementing both the hierarchy tool and the automated VHDL generation tool could be done using AMPLEx [MenG91]. Finally, perhaps the most
important step would be integrating the tools together to form a complete CAE system for life-cycle complete engineering.
References


A. VHDL Packages for Life-Cycle Modeling

A.1 Resource Requirements

use work.real_FUNCTIONS.all;
use std.textio.all;
package PRODUCT_CONSTRAINTS is

    constant MAXHORIZON: integer := 206;
    constant MAXFUNCS: integer := 50;

    type Functype is (NONE, UNIFORM, GRADIENT, GROWTH,
                     DECAY, EXPONENTIAL);
    type Dirflag is (UPPER, LOWER);
    type ConstParams is record
        FTYPE: Functype;
        FBEGIN: natural;
        FEND: natural;
        A: real;
        J: real;
    end record;
    type Constraint is array (0 to MAXFUNCS) of ConstParams;
    typeListOfConst is array (natural range <>) of Constraint;
    type TotalConst is array (0 to MAXHORIZON) of real;

    procedure OUTCONST(CONST: in Constraint;
                        FName: in string);

    function CHECKCONST(LIMIT: in Constraint;
                         TEST: in Constraint;
                         DIRECTION: in Dirflag;
                         CONSTNAME: in string;
                         FName: in string) return boolean;

    procedure TOTALFUNCS(CONST: in Constraint;
                          TOTAL: inout TotalConst;
                          HORIZON: inout integer);

    function fResolveConst(SIGNAL: ListOfConst) return Constraint;

    subtype ResolvedConst is fResolveConst Constraint;

end PRODUCT_CONSTRAINTS;

package body PRODUCT_CONSTRAINTS is

    procedure OUTCONST(CONST: in Constraint;
                        FName: in string) is
        file OUTLIST: TEXT is out FName;
        variable TEXTSETUP: LINE;
        variable Total: TotalConst := (others => 0.0);
        variable N: real;
        variable J,A: real;
        variable HORIZON: integer := MAXHORIZON;
begin
TOTALFUNCS(CONST, Total, HORIZON);
HORIZON := MAXHORIZON;
write(TEXTSETUP,string("Fiscal Period, Requirement"));
write(TEXTSETUP,TEXTSETUP);
for i in 0 to (HORIZON - 1) loop
  write(TEXTSETUP,i+1);
  write(TEXTSETUP,string("",""));
  write(TEXTSETUP,Total(i));
  writeln(OUTLIST,TEXTSETUP);
end loop;
end OUTCONST;

function CHECKCONST(LIMIT: in Constraint;
  TEST: in Constraint;
  DIRECTION: in Dirflag;
  CONSTNAME: in string;
  FNAME: in string) return boolean is

  variable Total: TotalConst := (others => 0.0);
  variable TotalLimit: TotalConst := (others => 0.0);
  variable N: real;
  variable J,A: real;
  variable HORIZON: integer := 0;
  file OUTLIST: TEXT is out FNAME;
  variable TEXTSETUP: LINE;
  variable PASSED: boolean := TRUE;

begin
  TOTALFUNCS(LIMIT,TotalLimit,HORIZON);
  TOTALFUNCS(TEST,Total,HORIZON);
  for i in 0 to (HORIZON - 1) loop
    case DIRECTION is
      when UPPER =>
        if Total(i) > TotalLimit(i) then
          write(TEXTSETUP,CONSTNAME);
          write(TEXTSETUP,string(" has exceeded its
constraint in fiscal period "));
          write(TEXTSETUP,i+1);
          writeln(OUTLIST,TEXTSETUP);
          PASSED := FALSE;
        end if;
      when LOWER =>
        if TotalLimit(i) > Total(i) then
          write(TEXTSETUP,CONSTNAME);
          write(TEXTSETUP,string(" has failed to meet its
requirement in fiscal period "));
          write(TEXTSETUP,i+1);
          writeln(OUTLIST,TEXTSETUP);
          PASSED := FALSE;
        end if;
    end case;
  end loop;
return PASSED;
end CHECKCONST;
procedure TOTALFUNCS\{CONST: in Constraint;
    TOTAL: inout TotalConst;
    HORIZON: inout integer\} is

    variable N: real;
    variable J,A: real;

begin
    for i in CONST'range loop
        for cj in (CONST(i).FBEGIN-1) to (CONST(i).FEND-1) loop
            J := CONST(i).J;
            A := CONST(i).A;
            N := real(2+cj) - CONST(i).FBEGIN;
            if HORIZON < CONST(i).FEND then
                HORIZON := CONST(i).FEND;
            end if;
            case CONST(i).FTYPE is
            when NONE => null;
            when UNIFORM =>
                TOTAL(cj) := TOTAL(cj) + A;
            when GRADIENT =>
                TOTAL(cj) := TOTAL(cj) +
                A * (N - 0.5);
            when DECAY =>
                TOTAL(cj) := TOTAL(cj) +
                A/(J * exp(J*N)) * ( exp(J) - 1.0 );
            when EXponential =>
                TOTAL(cj) := TOTAL(cj) +
                A*exp(J*N)*J * (1.0 - (1.0/exp(J)));
            when GROWTH =>
                TOTAL(cj) := TOTAL(cj) +
                (A/(J*exp(J*N)) * ( J*exp(J*N) - exp(J) + 1.0 ));
            end case;
        end loop;
    end loop;
end TOTALFUNCS;

function fResolveConst(SIGLIST: ListOfConst) return Constraint is

    variable POS: integer := 0;
    variable J: integer := 0;
    variable NEWLIST: Constraint;

begin
    for i in SIGLIST'range loop
        J := 0;
        while SIGLIST(i).J.FTYPE /= NONE and J < MAXFUNCS loop
            NEWLIST(POS) := SIGLIST(i)(J);
            J := J + 1;
            POS := POS + 1;
        end loop;
    end loop;
    return NEWLIST;
end fResolveConst;
end PRODUCT_CONSTRAINTS;
A.2 Reliability

use std.TEXTIO.all;
use work.real FUNCTIONS.all;
package RELIABILITY_FUNCTIONS is

  type ListOfFail is array (natural range <>) of real;

  procedure OUTMTBF(MTBF: in real; IDENT: in string;
                      FNAME: in string);
  procedure OUTMTTR(MTTR: in real; IDENT: in string;
                    FNAME: in string);

  function CHECKREL(MTBF: in real; LIFE: in real) return real;
  function CHECKAVA(MTBF: in real; MTTR: in real) return real;
  function CHECKMAN(MTTR: in real; LIFE: in real) return real;

  procedure OUTREL(MTBF: in real; LIFE: in real;
                   IDENT: in string; FNAME: in string);
  procedure OUTAVA(MTBF: in real; MTTR: in real;
                 IDENT: in string; FNAME: in string);
  procedure OUTMAN(MTTR: in real; LIFE: in real;
                 IDENT: in string; FNAME: in string);

  function fResolveMTBF(FAILLIST: ListOfFail) return real;

  subtype ResolvedMTBF is fResolveMTBF real;

end RELIABILITY_FUNCTIONS;

package body RELIABILITY_FUNCTIONS is

  procedure OUTMTBF(MTBF: in real; IDENT: in string;
                    FNAME: in string) is
    file OUTNUM; TEXT is out FNAME;
    variable TEXTSETUP: LINE;
  begin
    write(TEXTSETUP,string'("MTBF (in millions of hours) for "));
    write(TEXTSETUP,IDENT);
    write(TEXTSETUP,string'(" "));
    write(TEXTSETUP,MTBF);
    writeln(OUTNUM,TEXTSETUP);
  end OUTMTBF;

  procedure OUTMTTR(MTTR: in real; IDENT: in string;
                    FNAME: in string) is
    file OUTNUM; TEXT is out FNAME;
    variable TEXTSETUP: LINE;
  begin
    write(TEXTSETUP,string'("MTTR (in millions of hours) for "));
    write(TEXTSETUP,IDENT);
    write(TEXTSETUP,string'(" "));
    write(TEXTSETUP,MTTR);
    writeln(OUTNUM,TEXTSETUP);
  end OUTMTTR;

Appendix A. VHDL Packages for Life Cycle Modeling. 113
function CHECKREL(MTBF: in real;
LIFE: in real) return real is
    variable REL: real;
begin
    if MTBF /= 0.0 then
        REL := exp(-1.0/MTBF * LIFE);
    else
        REL := 0.0;
    end if;
    return REL;
end CHECKREL;

procedure OUTREL(MTBF: in real; LIFE: in real;
IDENT: in string; NAME: in string) is
    file OUTNUM: TEXT is out NAME;
    variable TEXTSETUP: LINE;
    variable REL: real;
begin
    REL := CHECKREL(MTBF,LIFE);
    write(TEXTSETUP,IDENT);
    write(TEXTSETUP,string" System Reliability for Lifetime of ");
    write(TEXTSETUP,LIFE);
    write(TEXTSETUP,string" million hours is ");
    write(TEXTSETUP,REL);
    writeln(OUTNUM,TEXTSETUP);
end OUTREL;

function CHECKAVA(MTBF: in real;
MTR: in real) return real is
    variableAVA: real;
begin
    if MTBF /= 0.0 and MTR /= 0.0 then
        AVA := 1.0 / ( 1.0 + ((1.0/MTBF)/(1.0/MTR)));
    else
        AVA := 0.0;
    end if;
    return AVA;
end CHECKAVA;

procedure OUTAVA(MTBF: in real; MTR: in real;
IDENT: in string; NAME: in string) is
    file OUTNUM: TEXT is out NAME;
    variable TEXTSETUP: LINE;
    variableAVA: real;
begin
    AVA := CHECKAVA(MTBF,MTR);
    write(TEXTSETUP,IDENT);
    write(TEXTSETUP,string" System Availability is ");
    write(TEXTSETUP,AVA);
    writeln(OUTNUM,TEXTSETUP);
end OUTAVA;

function CHECKMAP(MTR: in real;
LIFE: in real) return real is
variable MAN: real;

begin
    if MTTR /= 0.0 then
        MAN := 1.0 - exp(-1.0/MTTR * LIFE);
    else
        MAN := 0.0;
    end if;
    return MAN;
end CHECKMAN;

procedure OUTMAN(MTTR: in real; LIFE: in real;
                  IDENT: in string; NAME: in string) is
    file OUNUM: TEXT is out NAME;
    variable TEXTSETUP: LINE;
    variable MAN: real;
begin
    MAN := CHECKMAN(MTTR, LIFE);
    write(TEXTSETUP, IDENT);
    write(TEXTSETUP, string('" System Maintainability for "));
    write(TEXTSETUP, LIFE);
    write(TEXTSETUP, string('" million hours is ");
    write(TEXTSETUP, MAN);
    writeln(OUNUM, TEXTSETUP);
end OUTMAN;

function fResolveMTBF(FAILLIST: ListOfFail) return real is
    variable sum: real := 0.0;
begin
    for i in FAILLIST'range loop
        sum := sum + 1.0 / FAILLIST(i);
    end loop;
    sum := 1.0 / sum;
    return sum;
end fResolveMTBF;

end RELIABILITY_FUNCTIONS;

A.3 Parts Costs

use std.TEXTIO.all;
package PARTSCOST_FUNCTIONS is

    type ListOfCosts is array (natural range <>) of real;

    procedure OUTCOST(COST: in real; IDENT: in string; NAME: in string);

    function fResolveCost(COSTLIST: ListOfCosts) return real;

    subtype ResolvedCost is fResolveCost real;

end PARTSCOST_FUNCTIONS;

package body PARTSCOST_FUNCTIONS is

Appendix A. VHDL Packages for Life Cycle Modeling.
procedure OUTCOST(COST: in real; IDENT: in string;
   NAME: in string) is
  file OUTNUM: TEXT is out NAME;
  variable TEXTSETUP: line;
begin
  write(TEXTSETUP,IDENT);
  write(TEXTSETUP,string'(" Parts Cost: "));
  write(TEXTSETUP,COST);
  writeln(OUTNUM,TEXTSETUP);
end OUTCOST;

function fResolveCost(COSTLIST: ListOfCosts) return real is
  variable sum: real := 0.0;
begin
  for i in COSTLIST'range loop
    sum := sum + COSTLIST(i);
  end loop;
  return sum;
end fResolveCost;

end PARTSCOST FUNCTIONS;

A.4 Real Valued Functions

package REAL_FUNCTIONS is

  constant MAXPREC: real := 0.0000001;
  constant MAXITER: real := 20.0;

  procedure intfrac(FLOAT: in real; INT: out real; FRACT: out real);
  function fact(A: real) return real;
  function pow(A: real; X: real) return real;
  function exp(X: real) return real;
  function ln(X: real) return real;
  function fabs(X: real) return real;
  function fpow(A:real; X: real) return real;

end REAL_FUNCTIONS;

package body REAL_FUNCTIONS is

  function fact(A: real) return real is
    variable RESULT: real := 1.0;
  begin
    if A < 0.0 then
      return RESULT;
    end if;
    RESULT := real(A);
    for i in integer(A-2.0) downto 1 loop
      RESULT := RESULT * (A - real(i));
    end loop;
    return RESULT;
  end fact;

  function pow(A: real; X: real) return real is
variable RESULT: real := 1.0;
begin
if X = 0.0 then
return RESULT;
end if;
RESULT := 1.0;
for i in integer(fabs(X)) downto 1 loop
if X > 0.0 then
    RESULT := RESULT * A;
else
    RESULT := RESULT / A;
end if;
end loop;
return RESULT;
end pow;

function exp(X: real) return real is
variable WHOLE,FRACT: real;
variable WHOLERES: real;
variable RESULT: real := 0.0;
variable TEMP: real;
variable I: real;
begin
if X > 87.0 then
    return real'HIGH;
end if;
INTFRACT(X,WHOLES,FRACT);
WHOLERES := pow(2.71281728,WHOLES);
I := 0.0;
TEMP := -real'high;
while fabs(RESULT - TEMP) > MAXPREC and I < MAXITER loop
    TEMP := RESULT;
    RESULT := RESULT + pow(FRACT,I)/fact(I);
    I := I + 1.0;
end loop;
RESULT := WHOLERES * RESULT;
return RESULT;
end exp;

function ln(X: real) return real is
variable RESULT: real := 0.0;
variable TEMP: real;
variable I: real;
variable SCALE: real := 0.0;
variable OP: real;
begin
OP := X;
while OP > 1.0 loop
    OP := OP / 2.718281828;
    SCALE := SCALE + 1.0;
end loop;
I := 2.0;
TEMP := OP;
RESULT := OP - 1.0;
while fabs(RESULT - TEMP) > MAXPREC and I < MAXITER loop
    TEMP := RESULT;
end loop;
return RESULT;
end ln;

Appendix A. VHDL Packages for Life Cycle Modeling.
RESULT := RESULT + pow(-1.0, I-1.0)*(1.0/I)*pow(OP-1.0, I); 

I := I + 1.0;
end loop;
return RESULT + SCALE;
end in;

function fabs(X: real) return real is
begin
  if X < 0.0 then
    return -X;
  else
    return X;
  end if;
end fabs;

function fpow(A:real; X:real) return real is
  variable RESULT: real;
begin
  RESULT := exp(X*ln(A));
  return RESULT;
end fpow;

procedure intfrac(FLOAT: in real; INT: out real; FRACT: out real) is
  variable OP: real;
  variable OPROUND: real;
  variable OPINT: integer;
  variable DIF: real;
begin
  OP := FLOAT;
  OPINT := integer(OP);
  OPROUND := real(OPINT);
  DIF := OP - OPROUND;
  if OP >= 0.0 then
    if DIF >= 0.0 then
      INT := OPROUND;
      FRACT := OP - OPROUND;
    else
      INT := OPROUND - 1.0;
      FRACT := OP - OPROUND - 1.0;
    end if;
  else
    if DIF >= 0.0 then
      INT := OPROUND + 1.0;
      FRACT := -(1.0 - DIF);
    else
      INT := OPROUND;
      FRACT := DIF;
    end if;
  end if;
end intfrac;
end REAL_FUNCTIONS;

Appendix A. VHDL Packages for Life Cycle Modeling.
B. VHDL Model of Definition of Need

B.1 LAN_PRODUCT Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTS_COST_CONSTRAINTS.all;

entity LAN_PRODUCT is

generic(BUDGETFUNCS: Constraint;
        BUDGETLIMITS: Constraint;
        REVENUEFUNCS: Constraint;
        REVENUEMIN: Constraint;
        STAFFFUNCS: Constraint;
        STAFFLIMITS: Constraint;
        RELIABILITYLIM: real;
        PARTS_COSTLIM: real);

port(BUDGET: inout ResolvedConst := BUDGETLIMITS;
     REVENUE: inout ResolvedConst := REVENUEMIN;
     STAFF: inout ResolvedConst := STAFFLIMITS;
     MTBF: inout real := 675.0e+3;
     PARTS_COST: inout real := PARTS_COSTLIM);

constant LIFE: real := 150000.0;

begin

OUTCONST(BUDGET,"budget.req");
OUTCONST(BUDGETLIMITS,"budget.lim");
OUTCONST(REVENUE,"revenue.req");
OUTCONST(REVENUEMIN,"revenue.lim");
OUTCONST(STAFF,"staff.req");
OUTCONST(STAFFLIMITS,"staff.lim");

OUTREL(MTBF,LIFE,"rel.out");

OUTCOST(PARTS_COST,"parts_cost.out");

assert CHECKCONST(BUDGETLIMITS,BUDGET,UPPER,"Budget","Budget.chk")
    report "LAN Product has failed to meet Budget.";
assert CHECKCONST(REVENUEMIN,REVENUE,LOWER,"Revenue","Revenue.chk")
    report "LAN Product has failed to meet Revenue.";
assert CHECKCONST(STAFFLIMITS,STAFF,UPPER,"Staff","Staff.chk")
    report "LAN Product has failed to meet Staff.";

assert (CHECKREL(MTBF,LIFE) > RELIABILITYLIM)
    report "LAN Product has failed to meet reliability.";

assert (PARTS_COST <= PARTS_COSTLIM)
    report "LAN Product has exceeded its parts cost.";

Appendix B. VHDL Model of Definition of Need.
architecture DEFONEED of LAN_PRODUCT is
begin DEFONEED;
end;

B.2 Configuration for LAN_PRODUCT

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
configuration DEFONEED of tb is
  for LAN_PRODUCT
    for all: LAN_PRODUCT use entity work.LAN_PRODUCT(DEFONEED)
      generic map(
        --
        -- Budget Resource Requirement Functions
        --
        -- Requirements: 
        (others => (NONE,0,0,0,0,0,0),
        -- Limits:
        (0 => (GROWTH,1,24,300.0,0,1.0), -- 3 engineers
        1 => (GROWTH,6,22,100.0,1.0), -- +1 engineer
        2 => (GROWTH,17,22,200.0,1.0), -- +2 engineer
        3 => (UNIFORM,21,22,50.0,0.0), -- prototype cost
        4 => (DECAY,23,28,400.0,0.25), -- production run
        5 => (GROWTH,25,206,100.0,1.0), -- 2 support staff
        others => (NONE,0,0,0,0,0,0)
      ),
        --
        -- Revenue Resource Estimation Functions
        --
        -- Requirements: 
        (others => (NONE,0,0,0,0,0,0),
        -- Limits:
        (0 => (GROWTH,25,28,6000.0,1.0), -- peak sales
        1 => (DECAY,28,31,6000.0,1.0), -- obsolescence
        others => (NONE,0,0,0,0,0,0)
      ),
        --
        -- Staff Resource Requirement Functions
        --
        -- Requirements: 
        (others => (NONE,0,0,0,0,0,0),
        -- Limits:
        (0 => (GROWTH,1,24,3.0,1.0), -- 3 engineers
        1 => (GROWTH,6,22,1.0,1.0), -- +1 engineer
        2 => (GROWTH,17,22,2.0,1.0), -- +2 engineer
        3 => (GROWTH,25,206,2.0,1.0), -- 2 support staff
        others => (NONE,0,0,0,0,0,0)
      ),
        --
        -- Reliability Requirement:
        0.80,
        --
        -- Total Parts Cost: 
        400.0)
    end for;

Appendix B. VHDL Model of Definition of Need.
B.3 Testbench for LAN_PRODUCT

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity tb is
end tb;

architecture LAN_PRODUCT of tb is

    signal BUDGET,REVENUE,STAFF: ResolvedConst;
    signal MTBF: ResolvedMTBF;
    signal PARTSCOST: ResolvedCost;

    component LAN_PRODUCT
        port (BUDGET: inout ResolvedConst;
             REVENUE: inout ResolvedConst;
             STAFF: inout ResolvedConst;
             MTBF: inout ResolvedMTBF;
             PARTSCOST: inout ResolvedCost);
    end component;

begin
    C1: LAN_PRODUCT
        port map (BUDGET,REVENUE,STAFF,MTBF,PARTSCOST);

end LAN_PRODUCT;
C. VHDL Model of Preliminary Design

C.1 Bridge Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;

entity BRIDGE is

    generic(BUDGETFUNCS: Constraint;
             BUDGETLIMITS: Constraint;
             REVENUEFUNCS: Constraint;
             REVENUEMIN: Constraint;
             STAFFFUNCS: Constraint;
             STAFFLIMITS: Constraint;

             ReliabilityLimit: real;
             PARTSCOSTLIMIT: real);

    port(BUDGET: inout ResolvedConst := BUDGETFUNCS;
         REVENUE: inout ResolvedConst := REVENUEFUNCS;
         STAFF: inout ResolvedConst := STAFFFUNCS;

         MTBF: inout ResolvedMTBF;

         PARTSCOST: inout ResolvedCost := 0.0);

    constant LIFE: real := 150000.0;

begin

    OUTCONST(BUDGET,"budget.req");
    OUTCONST(BUDGETLIMITS,"budget.lim");
    OUTCONST(REVENUE,"revenue.req");
    OUTCONST(REVENUEMIN,"revenue.lim");
    OUTCONST(STAFF,"staff.req");
    OUTCONST(STAFFLIMITS,"staff.lim");

    OUTREL(MTBF,LIFE,"PRELIM","rel.out");

    OUTCOST(PARTSCOST,"PRELIM","partscost.out");

    assert CHECKCONST (BUDGETLIMITS, BUDGET, UPPER, "Budget","Budget.chk")
    report "Bridge has failed to meet Budget.";

    assert CHECKCONST (REVENUEMIN, REVENUE, LOWER, "Revenue", "Revenue.chk")
    report "Bridge has failed to meet Revenue.";

    assert CHECKCONST (STAFFLIMITS, STAFF, UPPER, "Staff", "Staff.chk")
    report "Bridge has failed to meet Staff.";

    assert (CHECKREL(MTBF,LIFE) > ReliabilityLimit)
    report "Bridge has exceeded its parts cost.";

    assert (PARTSCOST <= PARTSCOSTLIMIT)
    report "Bridge has exceeded its parts cost.";

Appendix C. VHDL Model of Preliminary Design.
end BRIDGE;

architecture PRELIMINARY of BRIDGE is

component MICROPROCESSOR

port(BUDGET: inout ResolvedConst;
    REVENUE: inout ResolvedConst;
    STAFF: inout ResolvedConst;
    MTBF: inout ResolvedMTBF;
    PARTSCOST: inout ResolvedCost);

end component;

component ETHERLINK

port(BUDGET: inout ResolvedConst;
    REVENUE: inout ResolvedConst;
    STAFF: inout ResolvedConst;
    MTBF: inout ResolvedMTBF;
    PARTSCOST: inout ResolvedCost);

end component;

component RAM

port(BUDGET: inout ResolvedConst;
    REVENUE: inout ResolvedConst;
    STAFF: inout ResolvedConst;
    MTBF: inout ResolvedMTBF;
    PARTSCOST: inout ResolvedCost);

end component;

component ROM

port(BUDGET: inout ResolvedConst;
    REVENUE: inout ResolvedConst;
    STAFF: inout ResolvedConst;
    MTBF: inout ResolvedMTBF;
    PARTSCOST: inout ResolvedCost);

end component;

component MOUNTING

port(BUDGET: inout ResolvedConst;

Appendix C. VHDL Model of Preliminary Design.
begin
   BUDGET <= BUDGETFUNCS;
   REVENUE <= REVENUEFUNCS;
   STAFF <= STAFFFUNCS;

   C1:MICROPROCESSOR
      port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

   C2:ETHERLINK
      port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

   C3:RAM
      port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

   C4:ROM
      port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

   C5:MOUNTING
      port map (BUDGET, REVENUE, STAFF, PARTSCOST);

   C6:MANUALS
      port map (BUDGET, REVENUE, STAFF, PARTSCOST);

end PRELIMINARY;

C.2 Bridge Configuration for Preliminary Design

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
configuration PRELIMINARY of tb is
   for BRIDGE
      for all: BRIDGE use entity work.BRIDGE(PRELIMINARY)
         generic map;
--
--         Budget Resource Requirement Functions

Appendix C. VHDL Model of Preliminary Design.
--

Requirements:
(0 => (GROWTH,1,24,100.0,0.1,0), -- product manager
  1 => (UNIFORM,21,22,50.0,0.0), -- prototype cost
  2 => (DECAY,23,28,400.0,0.25), -- production run
  3 => (GROWTH,25,206,100.0,1.0), -- 2 support staff
  others => (NONE,0,0,0,0,0,0) ),

Limits:
(0 => (GROWTH,1,24,300.0,1.0), -- 3 engineers
  1 => (GROWTH,6,22,100.0,1.0), -- +1 engineer
  2 => (GROWTH,17,22,200.0,1.0), -- +2 engineer
  3 => (UNIFORM,21,22,50.0,0.0), -- prototype cost
  4 => (DECAY,23,28,400.0,0.25), -- production run
  5 => (GROWTH,23,206,101.0,1.0), -- 2 support staff
  others => (NONE,0,0,0,0,0,0) ),

Revenue Resource Estimation Functions
--

Requirements:
(0 => (GROWTH,25,28,6000.0,1.0), -- peak sales
  1 => (DECAY,28,31,6000.0,1.0), -- obsolescence
  others => (NONE,0,0,0,0,0,0) ),

Limits:
(0 => (GROWTH,25,28,6000.0,1.0), -- peak sales
  1 => (DECAY,28,31,6000.0,1.0), -- obsolescence
  others => (NONE,0,0,0,0,0,0) ),

Staff Resource Requirement Functions
--

Requirements:
(0 => (GROWTH,1,24,1.0,1.0), -- product manager
  1 => (GROWTH,25,206,2.0,1.0), -- 2 support staff
  2 => (GROWTH,17,22,1.0,1.0), -- swing man
  others => (NONE,0,0,0,0,0,0) ),

Limits:
(0 => (GROWTH,1,24,3.1,1.0), -- 3 engineers
  1 => (GROWTH,6,22,1.1,1.0), -- +1 engineer
  2 => (GROWTH,17,22,2.1,1.0), -- +2 engineer
  3 => (GROWTH,23,206,2.1,1.0), -- 2 support staff
  others => (NONE,0,0,0,0,0,0) ),

Reliability Requirement:
0.80,

Parts Cost Limit:
500.0);

for PRELIMINARY:
for all: MICROPROCESSOR use entity work.MICROPROCESSOR(PRELIMINARY)
generic map(

Budget Resource Requirement Functions
--

Requirements:
(others => (NONE,0,0,0,0,0,0) ),

Limits:
(others => (NONE,0,0,0,0,0,0,0) ),
--
Revenue Resource Estimation Functions
--
Requirements:
(others => (NONE,0,0,0,0,0,0,0) ),
--
Limits:
(others => (NONE,0,0,0,0,0,0,0) ),
--
Staff Resource Requirement Functions
--
Requirements:
(0 => (GROWTH,1,24,1.0,1.0),   -- microproc engnr
  1 => (GROWTH,17,22,1.0,1.0),   -- programmer
  others => (NONE,0,0,0,0,0,0,0) ),
--
Limits:
(0 => (GROWTH,1,24,1.0,1.0),   -- microproc engnr
  1 => (GROWTH,17,22,1.0,1.0),   -- programmer
  others => (NONE,0,0,0,0,0,0,0) ),
--
Reliability Requirement:
0.96,
--
Total Parts Cost:
50.0,
--
Mean time before failure:
3.7e+6,
--
Parts Cost
50.0);
end for;
for all: ETHERLINK use entity work.ETHERLINK(PRELIMINARY)
generic map{
--
Budget Resource Requirement Functions
--
Requirements:
(others => (NONE,0,0,0,0,0,0,0) ),
--
Limits:
(others => (NONE,0,0,0,0,0,0,0) ),
--
Revenue Resource Estimation Functions
--
Requirements:
(others => (NONE,0,0,0,0,0,0,0) ),
--
Limits:
(others => (NONE,0,0,0,0,0,0,0) ),
--
Staff Resource Requirement Functions
--
Requirements:
(0 => (GROWTH,1,24,1.0,1.0),   -- ethernet engnr
  others => (NONE,0,0,0,0,0,0,0) ),
--
Limits:
(0 => (GROWTH,1,24,1.0,1.0), -- ethernet engnr
    others => (NONE,0,0,0,0,0,0) ),

--
Reliability Requirement:
  0.94,
--
Total Parts Cost:
  400.0,
--
Mean time before failure:
  2.6e+6,
--
Parts Cost
  400.0);
end for;

for all: RAM use entity work.RAM(PRELIMINARY)
genric map(
--
Budget Resource Requirement Functions
--
Requirements:
 (others => (NONE,0,0,0,0,0,0) ),
 Limits:
 (others => (NONE,0,0,0,0,0,0) ),
--
Revenue Resource Estimation Functions
--
Requirements:
 (others => (NONE,0,0,0,0,0,0) ),
 Limits:
 (others => (NONE,0,0,0,0,0,0) ),
--
Staff Resource Requirement Functions
--
Requirements:
 (0 => (GROWTH,6,22,1.0,1.0), -- memory engnr
    others => (NONE,0,0,0,0,0,0) ),
 Limits:
 (0 => (GROWTH,6,22,1.0,1.0), -- memory engnr
    others => (NONE,0,0,0,0,0,0) ),
--
Reliability Requirement:
  0.94,
--
Total Parts Cost:
  10.0,
--
Mean time before failure:
  2.6e+6,
--
Parts Cost
  10.0);
end for;

for all: ROM use entity work.ROM(PRELIMINARY)
generic map;

Budget Resource Requirement Functions

Requirements:
  (others => (NONE,0,0,0,0,0,0) ),
Limits:
  (others => (NONE,0,0,0,0,0,0) ),

Revenue Resource Estimation Functions

Requirements:
  (others => (NONE,0,0,0,0,0,0) ),
Limits:
  (others => (NONE,0,0,0,0,0,0) ),

Staff Resource Requirement Functions

Requirements:
  (others => (NONE,0,0,0,0,0,0) ),
Limits:
  (others => (NONE,0,0,0,0,0,0) ),

Reliability Requirement:
  0.94,

Total Parts Cost:
  5.0,

Mean time before failure:
  2.5e+6,

Parts Cost
  5.0);
end for;

for all: MOUNTING use entity work.MOUNTING(PRELIMINARY)
generic map;

Budget Resource Requirement Functions

Requirements:
  (others => (NONE,0,0,0,0,0,0) ),
Limits:
  (others => (NONE,0,0,0,0,0,0) ),

Revenue Resource Estimation Functions

Requirements:
  (others => (NONE,0,0,0,0,0,0) ),
Limits:
  (others => (NONE,0,0,0,0,0,0) ),

Staff Resource Requirement Functions

Requirements:
(others => (NONE,0,0,0,0,0,0,0)) ;

Limits:
(others => (NONE,0,0,0,0,0,0,0)) ;

Total Parts Cost:
25.0,

Parts Cost
25.0);
end for;

for all: MANUALS use entity work.MANUALS(PRELIMINARY)
generic map(

--
Budget Resource Requirement Functions
--
Requirements:
(others => (NONE,0,0,0,0,0,0,0)) ;

Limits:
(others => (NONE,0,0,0,0,0,0,0)) ;

Revenue Resource Estimation Functions
--
Requirements:
(others => (NONE,0,0,0,0,0,0,0)) ;

Limits:
(others => (NONE,0,0,0,0,0,0,0)) ;

Staff Resource Requirement Functions
--
Requirements:
(0 => (GROWTH,23,24,2.0,1.0), -- technical writers
others => (NONE,0,0,0,0,0,0,0)) ;

Limits:
(0 => (GROWTH,23,24,2.0,1.0), -- technical writers
others => (NONE,0,0,0,0,0,0,0)) ;

Total Parts Cost:
10.0,

Parts Cost
10.0);
end for;

end for;
end for;
end PRELIMINARY;

C.3 Testbench for Preliminary Design Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTS_COST_FUNCTIONS.all;

Appendix C. VHDL Model of Preliminary Design
entity tb is
end tb;

architecture BRIDGE of tb is

signal BUDGET, REVENUE, STAFF: ResolvedConst;
signal MTBF: ResolvedMTBF;
signal PARTSCOST: ResolvedCost;

component BRIDGE
  port (BUDGET: inout ResolvedConst;
       REVENUE: inout ResolvedConst;
       STAFF: inout ResolvedConst;
       MTBF: inout ResolvedMTBF;
       PARTSCOST: inout ResolvedCost);
end component;

begin
  C1: BRIDGE
    port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);
end BRIDGE;

C.4 Microprocessor Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity MICROPROCESSOR is

  generic (BUDGETFUNCS: Constraint;
           BUDGETLIMITS: Constraint;
           REVENUEFUNCS: Constraint;
           REVENUELIMITS: Constraint;
           STAFFFUNCS: Constraint;
           STAFFLIMITS: Constraint;
           ReliabilityLimit: real;
           PARTSCOSTLIMIT: real;
           DEVMENUMTB: real;
           DEVMENUPARTSCOST: real);

  port (BUDGET: inout ResolvedConst;
       REVENUE: inout ResolvedConst;
       STAFF: inout ResolvedConst;
       MTBF: inout ResolvedMTBF;
       PARTSCOST: inout ResolvedCost := 0.0);

  constant LIFE: real := 150000.0;

Appendix C. VHDL Model of Preliminary Design.
signal LBUDGET: ResolvedConst := BUDGETFUNCS;
signal LREVENUE: ResolvedConst := REVENUEFUNCS;
signal LSTAFF: ResolvedConst := STAFFFUNCS;

signal LMTBF: ResolvedMTBF := DEVMETBF;
signal LPARTSCOST: ResolvedCost := DEVICEPARTSCOST;

begin

assert CHECKCONST(BUDGETLIMITS, LBUDGET, UPPER, "Budget","Budget.mic")
report "Microprocessor has failed to meet Budget.";
assert CHECKCONST(REVENUEMIN, LREVENUE, LOWER, "Revenue","Revenue.mic")
report "Microprocessor has failed to meet Revenue.";
assert CHECKCONST(STAFFLIMITS, LSTAFF, UPPER, "Staff","Staff.mic")
report "Microprocessor has failed to meet Staff.";

assert (CHECKREL(LMTBF, LIFE) > ReliabilityLimit)
report "Microprocessor has failed to meet reliability.";

assert (LPARTSCOST <= PARTSCOSTLIMIT)
report "Microprocessor has exceeded its parts cost.";

OUTCOST(LPARTSCOST, "MICROPROCESSOR", "micpart.out");
OUTREL(LMTBF, LIFE, "MICROPROCESSOR", "micmtbf.out");

end MICROPROCESSOR;

architecture PRELIMINARY of MICROPROCESSOR is
begin

BUDGET <= LBUDGET;
REVENUE <= LREVENUE;
STAFF <= LSTAFF;

MTBF <= LMTBF;
PARTSCOST <= LPARTSCOST;

end PRELIMINARY;

C.5 Ethernet Connection Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_CONSTRAINTS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity ETHERLINK is

generic(BUDGETFUNCS: Constraint;
BUDGETLIMITS: Constraint;
REVENUEFUNCS: Constraint;
REVENUEMIN: Constraint;
STAFFFUNCS: Constraint;
STAFFLIMITS: Constraint;

Appendix C. VHDL Model of Preliminary Design.
ReliabilityLimit: real;
PARTSCOSTLIMIT: real;

DEVICE MTBF: real;
DEVICEPARTSCOST: real);

port (BUDGET: inout ResolvedConst;
REVENUE: inout ResolvedConst;
STAFF: inout ResolvedConst;

MTBF: inout ResolvedMTBF;

PARTSCOST: inout ResolvedCost := 0.0);

constant LIFE: real := 150000.0;

signal LBUDGET: ResolvedConst := BUDGETFUNCS;
signal LREVENUE: ResolvedConst := REVENUEFUNCS;
signal LSTAFF: ResolvedConst := STAFFFUNCS;

signal LMTBF: ResolvedMTBF := DEVICE MTBF;
signal LPARTSCOST: ResolvedCost := DEVICEPARTSCOST;

begin

assert CHECKCONST (BUDGETLIMITS, LBUDGET, UPPER, "Budget", "Budget.eth")
  report "Etherlink has failed to meet Budget.";
assert CHECKCONST (REVENUELIMITS, LREVENUE, LOWER, "Revenue", "Revenue.eth")
  report "Etherlink has failed to meet Revenue.";
assert CHECKCONST (STAFFLIMITS, LSTAFF, UPPER, "Staff", "Staff.eth")
  report "Etherlink has failed to meet Staff.";

assert (CHECKREL(LMTBF, LIFE) > ReliabilityLimit)
  report "Etherlink has failed to meet reliability.";

assert (PARTSCOST <= PARTSCOSTLIMIT)
  report "Etherlink has exceeded its parts cost.";

OUTCOST (LPARTSCOST, "ETHERLINK","ethpart.out");
OUTREL (LMTBF, LIFE, "ETHERLINK","ethmtbf.out");

end ETHERLINK;

architecture PRELIMINARY of ETHERLINK is
begin

BUDGET <= LBUDGET;
REVENUE <= LREVENUE;
STAFF <= LSTAFF;

MTBF <= LMTBF;
PARTSCOST <= LPARTSCOST;

end PRELIMINARY;

Appendix C. VHDL Model of Preliminary Design.
C.6 RAM Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity RAM is

    generic(BUDGETFUNCS: Constraint;
            BUDGETLIMITS: Constraint;
            REVENUEFUNCS: Constraint;
            REVENUEMINS: Constraint;
            STAFFFUNCS: Constraint;
            STAFFLIMITS: Constraint;
            
            ReliabilityLimit: real;
            PARTSCOSTLIMIT: real;
            
            DEVICEMTBF: real;
            DEVICESPARENCOST: real);

    port(BUDGET: inout ResolvedConst;
         REVENUE: inout ResolvedConst;
         STAFF: inout ResolvedConst;
         
         MTBF: inout ResolvedMTBF;
         
         PARTSCOST: inout ResolvedCost := 0.0);

    constant LIFE: real := 150000.0;

    signal LBUDGET: ResolvedConst := BUDGETFUNCS;
    signal LREVENUE: ResolvedConst := REVENUEFUNCS;
    signal LSTAFF: ResolvedConst := STAFFFUNCS;
    
    signal LMTBF: ResolvedMTBF := DEVICEMTBF;
    signal LPARTSCOST: ResolvedCost := DEVICESPARENCOST;

begin

    assert CHECKCONST (BUDGETLIMITS, LBUDGET, UPPER, "Budget", "Budget."")
            report "RAM has failed to meet Budget.";
    assert CHECKCONST (REVENUEMINS, LREVENUE, LOWER, "Revenue", "Revenue."")
            report "RAM has failed to meet Revenue.";
    assert CHECKCONST (STAFFLIMITS, LSTAFF, UPPER, "Staff", "Staff."")
            report "RAM has failed to meet Staff.";
    
    assert (CHECKREL(LMTBF,LIFE) > ReliabilityLimit)
            report "RAM has failed to meet reliability.";
    
    assert (LPARTSCOST <= PARTSCOSTLIMIT)
            report "RAM has exceeded its parts cost.";

    OUTCOST(LPARTSCOST, "RAM", "rampart.out");
    OUTREL(LMTBF,LIFE, "RAM", "rammtbf.out");

end RAM;

Appendix C. VHDL Model of Preliminary Design.
architecture PRELIMINARY of RAM is
begin

   BUDGET <= LBUDGET;
   REVENUE <= LREVENUE;
   STAFF <= LSTAFF;

   MTBF <= LMTBF;
   PARTSCOST <= LPARTSCOST;

end PRELIMINARY;

C.7 ROM Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity ROM is

   generic(BUDGETFUNCS: Constraint;
           BUDGETLIMITS: Constraint;
           REVENUEFUNCS: Constraint;
           REVENUEFUNCS: Constraint;
           STAFFFUNCS: Constraint;
           STAFFLIMITS: Constraint;
           ReliabilityLimit: real;
           PARTSCOSTLIMIT: real;
           DEVICEMTB: real;
           DEVICEPARTSCOST: real);

port(BUDGET: inout ResolvedConst;
     REVENUE: inout ResolvedConst;
     STAFF: inout ResolvedConst;
     MTBF: inout ResolvedMTBF;
     PARTSCOST: inout ResolvedCost := 0.0);

constant LIFE: real := 150000.0;

signal LBUDGET: ResolvedConst := BUDGETFUNCS;
signal LREVENUE: ResolvedConst := REVENUEFUNCS;
signal LSTAFF: ResolvedConst := STAFFFUNCS;

signal LMTBF: ResolvedMTBF := DEVICEMTB;
signal LPARTSCOST: ResolvedCost := DEVICEPARTSCOST;

begin

   assert CHECKCONST(BUDGETLIMITS, LBUDGET, UPPER,"Budget","Budget.com")
   report "ROM has failed to meet Budget.";

Appendix C. VHDL Model of Preliminary Design. 134
assert CHECKCONST(REVENUEMIN, LREVENUE, LOWER, "Revenue", "Revenue"rom")
   report "ROM has failed to meet Revenue.";
assert CHECKCONST(STAFFLIMITS, LSTAFF, UPPER, "Staff", "Staff"rom")
   report "ROM has failed to meet Staff.";
assert (CHECKREL(IMTBF, LIFE) > ReliabilityLimit)
   report "ROM has failed to meet reliability.";
assert (PARTSCOST <= PARTSCOSTLIMIT)
   report "ROM has exceeded its parts cost.";
OUTCOST(PARTSCOST,"ROM","rompart.out");
OUTREL(IMTBF, LIFE,"ROM","rommtbf.out");
end ROM;

architecture PRELIMINARY of ROM is
begin

BUDGET <= LBUDGET;
REVENUE <= LREVENUE;
STAFF <= LSTAFF;

IMTBF <= IMTBF;
PARTSCOST <= LPARTSCOST;

end PRELIMINARY;

C.8 Manual Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity MANUALS is

   generic(BUDGETFUNCS: Constraint;
           BUDGETLIMITS: Constraint;
           REVENUEFUNCS: Constraint;
           REVENUEMIN: Constraint;
           STAFFFUNCS: Constraint;
           STAFFLIMITS: Constraint;

           PARTSCOSTLIMIT: real;
           DEVICEPARTSCOST: real);

   port(BUDGET: inout ResolvedConst;
        REVENUE: inout ResolvedConst;
        STAFF: inout ResolvedConst;

        PARTSCOST: inout ResolvedCost := 0.0);

   signal LBUDGET: ResolvedConst := BUDGETFUNCS;

Appendix C. VHDL Model of Preliminary Design
signal LREVENUE: ResolvedConst := REVENUEFUNCS;
signal LSTAFF: ResolvedConst := STAFFFUNCS;

signal LPARTSCOST: ResolvedCost := DEVICEPARTSCOST;

begin

assert CHECKCONST(BUDGETLIMITS, LBUDGET, UPPER, "Budget", "Budget.man")
    report "Manuals has failed to meet Budget.";
assert CHECKCONST(REVENUEMINs, LREVENUE, LOWER, "Revenue", "Revenue.man")
    report "Manuals has failed to meet Revenue.";
assert CHECKCONST(STAFFLIMITS, LSTAFF, UPPER, "Staff", "Staff.man")
    report "Manuals has failed to meet Staff.";

assert (LPARTSCOST <= PARTSCOSTLIMIT)
    report "Manuals has exceeded its parts cost.";

end MANUALS;

architecture PRELIMINARY of MANUALS is
begin

    BUDGET <= LBUDGET;
    REVENUE <= LREVENUE;
    STAFF <= LSTAFF;

    PARTSCOST <= LPARTSCOST;

end PRELIMINARY;

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;

entity MANUALS is

    generic (BUDGETFUNCS: Constraint;
        BUDGETLIMITS: Constraint;
        REVENUEFUNCS: Constraint;
        REVENUEMINs: Constraint;
        STAFFFUNCS: Constraint;
        STAFFLIMITS: Constraint;
        PARTSCOSTLIMIT: real;
        DEVICEPARTSCOST: real);

    port (BUDGET: inout ResolvedConst;
        REVENUE: inout ResolvedConst;
        STAFF: inout ResolvedConst;
        PARTSCOST: inout ResolvedCost := 0.0);

    signal LBUDGET: ResolvedConst := BUDGETFUNCS;
    signal LREVENUE: ResolvedConst := REVENUEFUNCS;

Appendix C. VHDL Model of Preliminary Design.
signal LSTAFF: ResolvedConst := STAFFFUNCS;
signal LPARTSCOST: ResolvedCost := DEVICEPARTSCOST;

begin

assert CHECKCONST(BUDGETLIMITS, LBUDGET, UPPER, "Budget", "Budget.menu")
report "Manuals has failed to meet Budget.";
assert CHECKCONST(REVENUEFUNCS, LREVENUE, LOWER, "Revenue", "Revenue.menu")
report "Manuals has failed to meet Revenue.";
assert CHECKCONST(STAFFLIMITS, LSTAFF, UPPER, "Staff", "Staff.menu")
report "Manuals has failed to meet Staff.";

assert (LPARTSCOST <= PARTSCOSTLIMIT)
report "Manuals has exceeded its parts cost.";

OUTCOST(LPARTSCOST,"MANUALS","manpart.out");

end MANUALS;

architecture PRELIMINARY of MANUALS is
begin

BUDGET <= LBUDGET;
REVENUE <= LREVENUE;
STAFF <= LSTAFF;

PARTSCOST <= LPARTSCOST;
end PRELIMINARY;

C.9 Mounting Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity MOUNTING is

generic
(BUDGETFUNCS: Constraint;
BUDGETLIMITS: Constraint;
REVENUEFUNCS: Constraint;
REVENUEFUNCS: Constraint;
REVENUEFUNCS: Constraint;
STAFFFUNCS: Constraint;
STAFFLIMITS: Constraint;

PARTSCOSTLIMIT: real;

DEVICEPARTSCOST: real);

port
(BUDGET: inout ResolvedConst;
REVENUE: inout ResolvedConst;
STAFF: inout ResolvedConst;

PARTSCOST: inout ResolvedCost := 0.0);
signal LBUDGET: ResolvedConst := BUDGETFUNCS;
signal LREVENUE: ResolvedConst := REVENUEFUNCS;
signal LSTAFF: ResolvedConst := STAFFFUNCS;

signal LPARTSCOST: ResolvedCost := DEVICEPARTSCOST;

begin

assert CHECKCONST(BUDGETLIMITS, LBUDGET, UPPER, "Budget", "Budget.mnt")
report "Mounting has failed to meet Budget.";
assert CHECKCONST(REVENUELIMITS, LREVENUE, LOWER, "Revenue", "Revenue.mnt")
report "Mounting has failed to meet Revenue.";
assert CHECKCONST(STAFFLIMITS, LSTAFF, UPPER, "Staff", "Staff.mnt")
report "Mounting has failed to meet Staff.";

assert (LPARTSCOST <= PARTSCOSTLIMIT)
report "Mounting has exceeded its parts cost."

OUTCOST(LPARTSCOST,"MOUNTING","mntpart.out");

end MOUNTING;

architecture PRELIMINARY of MOUNTING is
begin

BUDGET <= LBUDGET;
REVENUE <= LREVENUE;
STAFF <= LSTAFF;

PARTSCOST <= LPARTSCOST;

end PRELIMINARY;
D. VHDL Model of Detailed Design

D.1 Bridge Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;

entity BRIDGE is

    generic (BUDGETFUNCS: Constraint;
             BUDGETLIMITS: Constraint;
             REVENUEFUNCS: Constraint;
             REVENUEMINS: Constraint;
             STAFFFUNCS: Constraint;
             STAFFLIMITS: Constraint;
             
             ReliabilityLimit: real;
             PARTSCOSTLIMIT: real);

    port (BUDGET: inout ResolvedConst := BUDGETFUNCS;
          REVENUE: inout ResolvedConst := REVENUEFUNCS;
          STAFF: inout ResolvedConst := STAFFFUNCS;
          
          MTBF: inout ResolvedMTBF;
          
          PARTSCOST: inout ResolvedCost := 0.0);

    constant LIFE: real := 150000.0;

begin
    OUTCONST (BUDGET, "budget.req");
    OUTCONST (BUDGETLIMITS, "budget.lim");
    OUTCONST (REVENUE, "revenue.req");
    OUTCONST (REVENUEMINS, "revenue.lim");
    OUTCONST (STAFF, "staff.req");
    OUTCONST (STAFFLIMITS, "staff.lim");

    OUTREL (MTBF, LIFE, "DETAIL", "rel.out");

    OUTCOST (PARTSCOST, "DETAIL", "partscst.out");

    assert CHECKCONST (BUDGETLIMITS, BUDGET, UPPER, "Budget", "Budget.chk")
        report "Bridge has failed to meet Budget.";
    assert CHECKCONST (REVENUEMINS, REVENUE, LOWER, "Revenue", "Revenue.chk")
        report "Bridge has failed to meet Revenue.";
    assert CHECKCONST (STAFFLIMITS, STAFF, UPPER, "Staff", "Staff.chk")
        report "Bridge has failed to meet Staff.";

    assert (CHECKREL (MTBF, LIFE) > ReliabilityLimit)
        report "Bridge has failed to meet reliability.";

    assert (PARTSCOST <= PARTSCOSTLIMIT)
        report "Bridge has exceeded its parts cost.";
end BRIDGE;

architecture DETAIL of BRIDGE is

    component MICROPROCESSOR

        port(BUDGET: inout ResolvedConst;
             REVENUE: inout ResolvedConst;
             STAFF: inout ResolvedConst;

             MTBF: inout ResolvedMTBF;

             PARTSCOST: inout ResolvedCost);

    end component;

    component ETHERLINK

        port(BUDGET: inout ResolvedConst;
             REVENUE: inout ResolvedConst;
             STAFF: inout ResolvedConst;

             MTBF: inout ResolvedMTBF;

             PARTSCOST: inout ResolvedCost);

    end component;

    component RAM

        port(BUDGET: inout ResolvedConst;
             REVENUE: inout ResolvedConst;
             STAFF: inout ResolvedConst;

             MTBF: inout ResolvedMTBF;

             PARTSCOST: inout ResolvedCost);

    end component;

    component DATASPACE

        port(BUDGET: inout ResolvedConst;
             REVENUE: inout ResolvedConst;
             STAFF: inout ResolvedConst;

             MTBF: inout ResolvedMTBF;

             PARTSCOST: inout ResolvedCost);

    end component;

    component SEARCH

        port(BUDGET: inout ResolvedConst;
REVENUE: inout ResolvedConst;
STAFF: inout ResolvedConst;

MTBF: inout ResolvedMTBF;
PARTSCOST: inout ResolvedCost);
end component;

component MOUNTING
port (BUDGET: inout ResolvedConst;
REVENUE: inout ResolvedConst;
STAFF: inout ResolvedConst;

PARTSCOST: inout ResolvedCost);
end component;

component MANUALS
port (BUDGET: inout ResolvedConst;
REVENUE: inout ResolvedConst;
STAFF: inout ResolvedConst;

PARTSCOST: inout ResolvedCost);
end component;

begin
BUDGET <= BUDGETFUNCS;
REVENUE <= REVENUEFUNCS;
STAFF <= STAFFFUNCS;

C1:MICROPROCESSOR
port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

C2:ETHERLINK
port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

C3:RAM
port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

C4:DATASPACE
port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

C5:SEARCH
port map (BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);

C6:MOUNTING
port map (BUDGET, REVENUE, STAFF, PARTSCOST);

C7:MANUALS
port map (BUDGET, REVENUE, STAFF, PARTSCOST);

Appendix D. VHDL Model of Detailed Design.
D.2 Bridge Configuration for Preliminary Design

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
configuration DETAIL of tb is
  for BRIDGE
    for all: BRIDGE use entity work.BRIDGE(Detail)
      generic map(
        --
        --
        --
        --
        Requirements:
        (0 => (GROWTH,1,24,100.0,1.0), -- product manager
        1 => (UNIFORM,21,22,50.0,0.0), -- prototype cost
        2 => (DECAY,23,28,400.0,0.0,25), -- production run
        3 => (GROWTH,25,206,100.0,1.0), -- 2 support staff
        others => (NONE,0,0,0,0,0,0) ),
        --
        Limits:
        (0 => (GROWTH,1,24,300.0,1.0), -- 3 engineers
        1 => (GROWTH,6,22,100.0,1.0), -- +1 engineer
        2 => (GROWTH,17,22,200.0,1.0), -- +2 engineer
        3 => (UNIFORM,21,22,50.0,0.0), -- prototype cost
        4 => (DECAY,23,28,400.0,0.0,25), -- production run
        5 => (GROWTH,23,206,101.0,1.0), -- 2 support staff
        others => (NONE,0,0,0,0,0,0) ),
        --
        --
        --
        --
        Requirements:
        (0 => (GROWTH,25,28,6000.0,1.0), -- peak sales
        1 => (DECAY,28,31,6000.0,1.0), -- obsolescence
        others => (NONE,0,0,0,0,0,0) ),
        --
        Limits:
        (0 => (GROWTH,25,28,6000.0,1.0), -- peak sales
        1 => (DECAY,28,31,6000.0,1.0), -- obsolescence
        others => (NONE,0,0,0,0,0,0) ),
        --
        --
        Staff Resource Requirement Functions
        --
        Requirements:
        (0 => (GROWTH,1,24,1.0,1.0), -- product manager
        1 => (GROWTH,25,206,2.0,1.0), -- 2 support staff
        2 => (GROWTH,17,22,1.0,1.0), -- swing man
        others => (NONE,0,0,0,0,0,0) ),
        --
        Limits:
        (0 => (GROWTH,1,24,3.1,1.0), -- 3 engineers
        1 => (GROWTH,6,22,1.1,1.0), -- +1 engineer
        2 => (GROWTH,17,22,2.1,1.0), -- +2 engineer
        3 => (GROWTH,23,206,2.1,1.0), -- 2 support staff
        others => (NONE,0,0,0,0,0,0) ),
Appendix D. VHDL Model of Detailed Design. 142
Reliability Requirement:
0.80,

Parts Cost Limit:
500.0);

for DETAIL
for all: MICROPROCESSOR use entity work.MICROPROCESSOR(PRELIMINARY)
generic map(

Budget Resource Requirement Functions

Requirements:
(others => (NONE,0,0,0,0,0,0) ),

Limits:
(others => (NONE,0,0,0,0,0,0) ),

Revenue Resource Estimation Functions

Requirements:
(others => (NONE,0,0,0,0,0,0) ),

Limits:
(others => (NONE,0,0,0,0,0,0) ),

Staff Resource Requirement Functions

Requirements:
(0 => (GROWTH,1,24,1.0,1.0), -- microproc engnr
  1 => (GROWTH,17,22,1.0,1.0), -- programmer
  others => (NONE,0,0,0,0,0,0) ),

Limits:
(0 => (GROWTH,1,24,1.0,1.0), -- microproc engnr
  1 => (GROWTH,17,22,1.0,1.0), -- programmer
  others => (NONE,0,0,0,0,0,0) ),

Reliability Requirement:
0.96,

Total Parts Cost:
50.0,

Mean time before failure:
3.7e+6,

Parts Cost
50.0);

end for;

for all: ETHERLINK use entity work.ETHERLINK(DETAIL)
generic map:

Budget Resource Requirement Functions

Requirements:
(others => (NONE,0,0,0,0,0,0) ),
Limits:
  (others => (NONE,0,0,0,0,0) ),

Revenue Resource Estimation Functions

Requirements:
  (others => (NONE,0,0,0,0,0) ),

Limits:
  (others => (NONE,0,0,0,0,0) ),

Staff Resource Requirement Functions

Requirements:
  (0 => (GROWTH,1,24,1.0,1.0), -- ethernet engnr
  others => (NONE,0,0,0,0,0) ),

Limits:
  (0 => (GROWTH,1,24,1.0,1.0), -- ethernet engnr
  others => (NONE,0,0,0,0,0) ),

Reliability Requirement:
  0.94,

Total Parts Cost:
  390.0);

for DETAIL
  for all: ETHERCHIP use entity work.ETHERCHIP(DETAIL)
generic map(

Mean time before failure:
  10.0e+5,

Parts Cost
  95.0); end for;
end for;
end for;

for all: RAM use entity work.RAM(PRELIMINARY)
generic map(

Budget Resource Requirement Functions

Requirements:
  (others => (NONE,0,0,0,0,0) ),

Limits:
  (others => (NONE,0,0,0,0,0) ),

Revenue Resource Estimation Functions

Requirements:
  (others => (NONE,0,0,0,0,0) ),

Limits:
  (others => (NONE,0,0,0,0,0) ),

Staff Resource Requirement Functions

Appendix D. VHDL Model of Detailed Design.
--

    Requirements:
    (0 => (GROWTH,6,22,1.0,0.0), -- memory engnr
     others => (NONE,0,0,0.0,0.0) ),
    Limits:
    (0 => (GROWTH,6,22,1.0,0.0), -- memory engnr
     others => (NONE,0,0,0.0,0.0) ),

    Reliability Requirement:
    0.94,

    Total Parts Cost:
    10.0,

    Mean time before failure:
    2.6e+6,

    Parts Cost
    10.0);
end for;
for all: DATASPACE use entity work.DATASPACE(DETAIL)
generic map(

    --

    Budget Resource Requirement Functions

    --

    Requirements:
    (others => (NONE,0,0,0.0,0.0) ),
    Limits:
    (others => (NONE,0,0,0.0,0.0) ),

    Revenue Resource Estimation Functions

    --

    Requirements:
    (others => (NONE,0,0,0.0,0.0) ),
    Limits:
    (others => (NONE,0,0,0.0,0.0) ),

    Staff Resource Requirement Functions

    --

    Requirements:
    (others => (NONE,0,0,0.0,0.0) ),
    Limits:
    (others => (NONE,0,0,0.0,0.0) ),

    Reliability Requirement:
    0.94,

    Total Parts Cost:
    10.0);

    for DETAIL
    for all: RAMCHIP use entity work.RAMCHIP(DETAIL)
generic map(

    --

    Mean time before failure:
19.0e+6,

--
Parts Cost
  1.1);
end for;

for all: ROMCHIP use entity work.ROMCHIP(DETAIL)
  generic map(

--
Mean time before failure:
  10.0e+6,

--
Parts Cost
  3.5);
end for;

for all: NVRAMCHIP use entity work.NVRAMCHIP(DETAIL)
  generic map(

--
Mean time before failure:
  10.0e+6 ,

--
Parts Cost
  2.0);
end for;
end for;

for all: SEARCH use entity work.SEARCH(DETAIL)
  generic map(

--
Budget Resource Requirement Functions

--
Requirements:
  (others => (NONE,0,0,0,0,0,0,0) ),

--
Limits:
  (others => (NONE,0,0,0,0,0,0,0) ),

--
Revenue Resource Estimation Functions

--
Requirements:
  (others => (NONE,0,0,0,0,0,0) ),

--
Limits:
  (others => (NONE,0,0,0,0,0,0) ),

--
Staff Resource Requirement Functions

--
Requirements:
  (others => (NONE,0,0,0,0,0,0) ),

--
Limits:
  (others => (NONE,0,0,0,0,0,0) ),

--
Reliability Requirement:
  0.97,

--
Total Parts Cost:
10.0,

-- Mean time before failure:
4.93e+6,

-- Parts Cost
10.0);
end for;

for all: MOUNTING use entity work.MOUNTING(PRELIMINARY)
generic map(

--
-- Budget Resource Requirement Functions
--
-- Requirements:
{others => (NONE,0,0,0,0,0,0) },
-- Limits:
{others => (NONE,0,0,0,0,0,0) },

-- Revenue Resource Estimation Functions
--
-- Requirements:
{others => (NONE,0,0,0,0,0,0) },
-- Limits:
{others => (NONE,0,0,0,0,0,0) },

-- Staff Resource Requirement Functions
--
-- Requirements:
{others => (NONE,0,0,0,0,0,0) },
-- Limits:
{others => (NONE,0,0,0,0,0,0) },

-- Total Parts Cost:
25.0,

-- Parts Cost
25.0);
end for;

for all: MANUALS use entity work.MANUALS(PRELIMINARY)
generic map(

--
-- Budget Resource Requirement Functions
--
-- Requirements:
{others => (NONE,0,0,0,0,0,0) },
-- Limits:
{others => (NONE,0,0,0,0,0,0) },

-- Revenue Resource Estimation Functions
--
-- Requirements:
{others => (NONE,0,0,0,0,0,0) },
-- Limits:
{others => (NONE,0,0,0,0,0,0) },
D.3 Testbench for Preliminary Design Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity tb is
end tb;

architecture BRIDGE of tb is
begin
    component BRIDGE
        port(BUDGET: inout ResolvedConst;
             REVENUE: inout ResolvedConst;
             STAFF: inout ResolvedConst;
             MTBF: inout ResolvedMTBF;
             PARTSCOST: inout ResolvedCost);
    end component;

    C1: BRIDGE
        port map(BUDGET, REVENUE, STAFF, MTBF, PARTSCOST);
end BRIDGE;
D.4 Microprocessor Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity MICROPROCESSOR is

  generic (BUDGETFUNCS: Constraint;
           BUDGETLIMITS: Constraint;
           REVENUEFUNCS: Constraint;
           REVENUEMIN: Constraint;
           STAFFFUNCS: Constraint;
           STAFFLIMITS: Constraint;
           ReliabilityLimit: real;
           PARTSCOSTLIMIT: real;
           
           DEVICEMTBF: real;
           DEVICEPARTSCOST: real);

  port (BUDGET: inout ResolvedConst;
        REVENUE: inout ResolvedConst;
        STAFF: inout ResolvedConst;
        
        MTBF: inout ResolvedMTBF;
        PARTSCOST: inout ResolvedCost := 0.0);

  constant LIFE: real := 150000.0;

  signal LBUDGET: ResolvedConst := BUDGETFUNCS;
  signal LREVENUE: ResolvedConst := REVENUEFUNCS;
  signal LSTAFF: ResolvedConst := STAFFFUNCS;
  
  signal LM: ResolvedMTBF := DEVICEMTBF;
  signal LPARTSCOST: ResolvedCost := DEVICEPARTSCOST;

begin

  assert CHECKCONST (BUDGETLIMITS, LBUDGET, UPPER, "Budget", "Budget.mic")
              report "Microprocessor has failed to meet Budget.");
  assert CHECKCONST (REVENUEMIN, LREVENUE, LOWER, "Revenue", "Revenue.mic")
              report "Microprocessor has failed to meet Revenue.");
  assert CHECKCONST (STAFFLIMITS, LSTAFF, UPPER, "Staff", "Staff.mic")
              report "Microprocessor has failed to meet Staff.");

  assert (CHECKREL (LM, LIFE) > ReliabilityLimit)
              report "Microprocessor has failed to meet reliability.");

  assert (LPARTSCOST <= PARTSCOSTLIMIT)
              report "Microprocessor has exceeded its parts cost.");

  OUTM (LPARTSCOST, "MICROPROCESSOR", "micpart.out");
  OUTR (LM, LIFE, "MICROPROCESSOR", "mictbf.out");

end MICROPROCESSOR;

Appendix D. VHDL Model of Detailed Design.
architecture PRELIMINARY of MICROPROCESSOR is
begin

    BUDGET <= LBUDGET;
    REVENUE <= LREVENUE;
    STAFF <= LSTAFF;

    MTBF <= LMTBF;
    PARTSCOST <= LPARTSCOST;

end PRELIMINARY;

D.5 Ethernet Connection Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity ETHERLINK is

    generic(BUDGETFUNCS: Constraint;
             BUDGETLIMITS: Constraint;
             REVENUEFUNCS: Constraint;
             REVENUEMIN: Constraint;
             STAFFFUNCS: Constraint;
             STAFFLIMITS: Constraint;
             ReliabilityLimit: real;
             PARTSCOSTLIMIT: real);

    port(BUDGET: inout ResolvedConst;
         REVENUE: inout ResolvedConst;
         STAFF: inout ResolvedConst;
         MTBF: inout ResolvedMTBF;
         PARTSCOST: inout ResolvedCost := 0.0);

    constant LIFE: real := 1500000.0;

    signal LBUDGET: ResolvedConst := BUDGETFUNCS;
    signal LREVENUE: ResolvedConst := REVENUEFUNCS;
    signal LSTAFF: ResolvedConst := STAFFFUNCS;

    signal LMTBF: ResolvedMTBF;
    signal LPARTSCOST: ResolvedCost := 0.0;

begin

    assert CHECKCONST(BUDGETLIMITS, LBUDGET, UPPER,"Budget","Budget.eth")
           report "Etherlink has failed to meet Budget.";
    assert CHECKCONST(REVENUEMIN, LREVENUE, LOWER,"Revenue","Revenue.eth")
           report "Etherlink has failed to meet Revenue.";
    assert CHECKCONST(STAFFLIMITS, LSTAFF, UPPER,"Staff","Staff.eth")

Appendix D. VHDL Model of Detailed Design. 150
report "Etherlink has failed to meet Staff."

assert (CHECKREL(LMTBF,LIFE) > ReliabilityLimit)
report "Etherlink has failed to meet reliability."

assert (LPARTSCOST <= PARTSCOSTLIMIT)
report "Etherlink has exceeded its parts cost."

OUTCOST (LPARTSCOST,"ETHERLINK","ethpart.out");
OUTREL (LMTBF,LIFE,"ETHERLINK","ethmtbf.out");

end ETHERLINK;

architecture PRELIMINARY of ETHERLINK is
begin

  BUDGET <= LBUDGET;
  REVENUE <= LREVENUE;
  STAFF <= LSTAFF;

  MTFB <= LMTFB;
  PARTSCOST <= LPARTSCOST;

end PRELIMINARY;

architecture DETAIL of ETHERLINK is

  component ETHERCHIP
  port(MTB: inout ResolvedMTBF; PARTSCOST: inout ResolvedCost);
  end component;

begin

  BUDGET <= LBUDGET;
  REVENUE <= LREVENUE;
  STAFF <= LSTAFF;

  MTFB <= LMTFB;
  PARTSCOST <= LPARTSCOST;

  C1: ETHERCHIP
  port map(LMTFB,LPARTSCOST);
C2: ETHERCHIP
  port map(LMTFB,LPARTSCOST);

end DETAIL;

D.6 Etherchip Model

use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity ETHERCHIP is

  generic(DEVICEPARTSCOST: real;
           DEVICEMTBF: real);

Appendix D. VHDL Model of Detailed Design.
port (MTBF: inout ResolvedMTBF := DEVICEMTBF;
     PARTSCOST: inout ResolvedCost := DEVICEPARTSCOST);
end ETHERCHIP;
architecture DETAIL of ETHERCHIP is
begin
end DETAIL;

D.7 RAM Model

use work.PRODUCT CONSTRAINTS.all;
use work.RELIABILITY FUNCTIONS.all;
use work.PARTSCOST FUNCTIONS.all;
entity RAM is
  generic (BUDGETFUNCS: Constraint;
           BUDGETLIMITS: Constraint;
           REVENUEFUNCS: Constraint;
           REVENUEMIN: Constraint;
           STAFFFUNCS: Constraint;
           STAFFLIMITS: Constraint;
           ReliabilityLimit: real;
           PARTSCOSTLIMIT: real;
           DEVICEMTBF: real;
           DEVICEPARTSCOST: real);
port (BUDGET: inout ResolvedConst;
      REVENUE: inout ResolvedConst;
      STAFF: inout ResolvedConst;
      MTBF: inout ResolvedMTBF;
      PARTSCOST: inout ResolvedCost := 0.0);
constant LIFE: real := 150000.0;

signal LBUDGET: ResolvedConst := BUDGETFUNCS;
signal LREVENUE: ResolvedConst := REVENUEFUNCS;
signal LSTAFF: ResolvedConst := STAFFFUNCS;

signal LMTBF: ResolvedMTBF := DEVICEMTBF;
signal LPARTSCOST: ResolvedCost := DEVICEPARTSCOST;
begin
  assert CHECKCONST (BUDGETLIMITS,LBUDGET,UPPER,"Budget","Budget. ram")
      report "RAM has failed to meet Budget.");
  assert CHECKCONST (REVENUEMIN,LREVENUE,LOWER,"Revenue","Revenue. ram")
      report "RAM has failed to meet Revenue.");
  assert CHECKCONST (STAFFLIMITS,LSTAFF,UPPER,"Staff","Staff. ram")
      report "RAM has failed to meet Staff.");
D.8 DATASPACE Model

use work.PRODUCTCONSTRAINTS.all;
use work.RELIABILITYFUNCTIONS.all;
use work.PARTSCOSTFUNCTIONS.all;

entity DATASPACE is
  generic(
    BUDGETFUNCS: Constraint;
    BUDGETLIMITS: Constraint;
    REVENUEFUNCS: Constraint;
    REVENUEMINS: Constraint;
    STAFFFUNCS: Constraint;
    STAFFLIMITS: Constraint;
    ReliabilityLimit: real;
    PARTSCOSTLIMIT: real);

  port(
    BUDGET: inout ResolvedConst;
    REVENUE: inout ResolvedConst;
    STAFF: inout ResolvedConst;
    MTBF: inout ResolvedMTBF;
    PARTSCOST: inout ResolvedCost := 0.0);

  constant LIFE: real := 150000.0;

  signal LBOUND: ResolvedConst := BUDGETFUNCS;
  signal LREVENUE: ResolvedConst := REVENUEFUNCS;
signal LSTAFF: ResolvedConst := STAFFFUNCS;

signal LMTBF: ResolvedMTBF;
signal LPARTSCOST: ResolvedCost := 0.0;

begin

assert CHECKCONST(BUDGETLIMITS,LBUDGET,UPPER,"Budget","Budget.dat")
report "DATAPLACE has failed to meet Budget.";

assert CHECKCONST(REVENUEMINSL,LEVENUE,LOWER,"Revenue","Revenue.dat")
report "DATAPLACE has failed to meet Revenue.";

assert CHECKCONST(STAFFLIMITS,LSSTAFF,UPPER,"Staff","Staff.dat")
report "DATAPLACE has failed to meet Staff.";

assert (CHECKREL(LMTBF,LIFE) > ReliabilityLimit)
report "DATAPLACE has failed to meet reliability.";

assert (LPARTSCOST <= PARTSCOSTLIMIT)
report "DATAPLACE has exceeded its parts cost.";

OUTCOST(LPARTSCOST,"DATAPLACE","datapart.out");
OUTREL(LMTBF,LIFE,"DATAPLACE","datamtb.out");

end DATAPLACE;

architecture DETAIL of DATAPLACE is

component RAMCHIP
    port(MTBF: inout ResolvedMTBF;
         PARTSCOST: inout ResolvedCost);
end component;

component ROMCHIP
    port(MTBF: inout ResolvedMTBF;
         PARTSCOST: inout ResolvedCost);
end component;

component NVRAMCHIP
    port(MTBF: inout ResolvedMTBF;
         PARTSCOST: inout ResolvedCost);
end component;

begin

    BUDGET <= LBUDGET;
    REVENUE <= LEVENUE;
    STAFF <= LSSTAFF;

    MTBF <= LMTBF;
    PARTSCOST <= LPARTSCOST;

    C1: RAMCHIP
        port map(IMTBF,LPARTSCOST);
    C2: RAMCHIP
        port map(IMTBF,LPARTSCOST);
    C3: RAMCHIP

Appendix D. VHDL Model of Detailed Design.
D.9 Memchips Model

use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity RAMCHIP is
  generic (DEVICEMTBF: real;
           DEVICEPARTSCOST: real);
  port (MTBF: inout ResolvedMTBF := DEVICEMTBF;
        PARTSCOST: inout ResolvedCost := DEVICEPARTSCOST);
end RAMCHIP;

architecture DETAIL of RAMCHIP is begin end DETAIL;

use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity ROMCHIP is
  generic (DEVICEMTBF: real;
           DEVICEPARTSCOST: real);
  port (MTBF: inout ResolvedMTBF := DEVICEMTBF;
        PARTSCOST: inout ResolvedCost := DEVICEPARTSCOST);
end ROMCHIP;

architecture DETAIL of ROMCHIP is begin end DETAIL;

use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity NVRAMCHIP is
  generic (DEVICEMTBF: real;
           DEVICEPARTSCOST: real);
  port (MTBF: inout ResolvedMTBF := DEVICEMTBF;
        PARTSCOST: inout ResolvedCost := DEVICEPARTSCOST);
end NVRAMCHIP;

architecture DETAIL of NVRAMCHIP is
begin
end DETAIL;

D.10 Search Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity SEARCH is

  generic(BUDGETFUNCS: Constraint;
    BUDGETLIMITS: Constraint;
    REVENUEFUNCS: Constraint;
    REVENUELIMITS: Constraint;
    STAFFFUNCS: Constraint;
    STAFFLIMITS: Constraint;
    ReliabilityLimit: real;
    PARTSCOSTLIMIT: real;
    DEVICEMTBFB: real;
    DEVICEPARTSCOST: real);

  port(BUDGET: inout ResolvedConst;
       REVENUE: inout ResolvedConst;
       STAFF: inout ResolvedConst;
       MTBF: inout ResolvedMTBF;
       PARTSCOST: inout ResolvedCost := 0.0);

constant LIFE: real := 150000.0;

signal LBUDGET: ResolvedConst := BUDGETFUNCS;
signal LREVENUE: ResolvedConst := REVENUEFUNCS;
signal LSTAFF: ResolvedConst := STAFFFUNCS;

signal LMTBF: ResolvedMTBF := DEVICEMTBFB;
signal LPARTSCOST: ResolvedCost := DEVICEPARTSCOST;

begin

  assert CHECKCONST(BUDGETLIMITS, LBUDGET, UPPER, "Budget", "Budget.srh")
    report "Search Logic has failed to meet Budget.";
  assert CHECKCONST(REVENUELIMITS, LREVENUE, LOWER, "Revenue", "Revenue.srh")
    report "Search Logic has failed to meet Revenue.";
  assert CHECKCONST(STAFFLIMITS, LSTAFF, UPPER, "Staff", "Staff.srh")
    report "Search Logic has failed to meet Staff.";

  assert (CHECKREL(LMTBF, LIFE) > ReliabilityLimit)
    report "Search Logic has failed to meet reliability.";

Appendix D. VHDL Model of Detailed Design.
assert (LPARTSCOST <= PARTSCOSTLIMIT)
        report "Search Logic has exceeded its parts cost.");

OUTCOST(LPARTSCOST,"SEARCH","srchpart.out");
OUTREL(IMTBFLIFE,"SEARCH","srchmtbf.out");

end SEARCH;

architecture DETAIL of SEARCH is
begin

    BUDGET <= LBUDGET;
    REVENUE <= LREVENUE;
    STAFF <= LSTAFF;

    MTBF <= IMTBFLIFE;
    PARTSCOST <= LPARTSCOST;

end DETAIL;

D.11 Manuals Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity MANUALS is

generic (BUDGETFUNCS: Constraint;
          BUDGETLIMITS: Constraint;
          REVENUEFUNCS: Constraint;
          REVENUEMIN: Constraint;
          STAFFFUNCS: Constraint;
          STAFFLIMITS: Constraint;
          )

    PARTSCOSTLIMIT: real;
    DEVICEPARTSCOST: real);

port (BUDGET: inout ResolvedConst;
       REVENUE: inout ResolvedConst;
       STAFF: inout ResolvedConst;

       PARTSCOST: inout ResolvedCost := 0.0);

signal LBUDGET: ResolvedConst := BUDGETFUNCS;
signal LREVENUE: ResolvedConst := REVENUEFUNCS;
signal LSTAFF: ResolvedConst := STAFFFUNCS;

signal LPARTSCOST: ResolvedCost := DEVICEPARTSCOST;

begin
assert CHECKCONST (BUDGETLIMITS, LBUDGET, UPPER, "Budget", "Budget.man")
report "Manuals has failed to meet Budget.");
assert CHECKCONST (REVENUEMIN, LREVENUE, LOWER, "Revenue", "Revenue.man")
report "Manuals has failed to meet Revenue.");
assert CHECKCONST (STAFFLIMITS, LSTAFF, UPPER, "Staff", "Staff.man")
report "Manuals has failed to meet Staff.");

assert (LPARTSCOST <= PARTSCOSTLIMIT)
report "Manuals has exceeded its parts cost.");

OUTCOST (LPARTSCOST, "MANUALS", "manpart.out");

end MANUALS;

architecture PRELIMINARY of MANUALS is
begin

BUDGET <= LBUDGET;
REVENUE <= LREVENUE;
STAFF <= LSTAFF;

PARTSCOST <= LPARTSCOST;

end PRELIMINARY;

D.12 Mounting Model

use work.PRODUCT_CONSTRAINTS.all;
use work.RELIABILITY_FUNCTIONS.all;
use work.PARTSCOST_FUNCTIONS.all;
entity MOUNTING is

generic (BUDGETFUNCS: Constraint;
BUDGETLIMITS: Constraint;
REVENUEFUNCS: Constraint;
REVENUEMIN: Constraint;
STAFFFUNCS: Constraint;
STAFFLIMITS: Constraint;
)

PARTSCOSTLIMIT: real;

DEVICEPARTSCOST: real);

port (BUDGET: inout ResolvedConst;
REVENUE: inout ResolvedConst;
STAFF: inout ResolvedConst;

PARTSCOST: inout ResolvedCost := 0.0);

signal LBUDGET: ResolvedConst := BUDGETFUNCS;
signal LREVENUE: ResolvedConst := REVENUEFUNCS;
signal LSTAFF: ResolvedConst := STAFFFUNCS;
signal LPARTSCOST: ResolvedCost := DEVC1EPARTSCOST;

begin

assert CHECKCONST(BUDGETLIMITS, LBUDGET, UPPER, "Budget", "Budget.mnt")
report "Mounting has failed to meet Budget."
assert CHECKCONST(REVENUEMIN, LREVENUE, LOWER, "Revenue", "Revenue.mnt")
report "Mounting has failed to meet Revenue."
assert CHECKCONST(STAFFLIMITS, LSTAFF, UPPER, "Staff", "Staff.mnt")
report "Mounting has failed to meet Staff."

assert (LPARTSCOST <= PARTSCOSTLIMIT)
report "Mounting has exceeded its parts cost."

OUTCOST(LPARTSCOST, "MOUNTING", "mntpart.out");

end MOUNTING;

architecture PRELIMINARY of MOUNTING is
begin

BUDGET <= LBUDGET;
REVENUE <= LREVENUE;
STAFF <= LSTAFF;

PARTSCOST <= LPARTSCOST;

end PRELIMINARY;
Vita

Rhett Hudson completed his Bachelor’s Degree in Electrical Engineering at Virginia Tech in the spring of 1993. After injuring his ankle while attempting a thru-hike of the Appalachian Trail, he finished his master’s thesis and began work on his Ph.D. He has been employed as a graduate research assistant on several projects since beginning his graduate studies. His current technical interests include computer networking applications, cyberspace and artificial intelligences that operate on network information sources.

Rhett Hudson