DESIGN OF THE COMPUTER SUBSYSTEM FOR THE AFIT SIMULATION SATELLITE (*SIMSAT*)

THESIS

Michael P. Hanke
GS-13, USAF

AFIT/GSE/ENY/98D-1

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GSE/ENY/98D-1

DESIGN OF THE COMPUTER SUBSYSTEM FOR THE AFIT SIMULATION SATELLITE (*SIMSAT*)

THESIS
Michael P. Hanke
GS-13, USAF

AFIT/GSE/ENY/98D-1

Approved for public release; distribution unlimited

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

AFIT/GSE/ENY/98D-1

DESIGN OF THE COMPUTER SUBSYSTEM FOR THE AFIT SIMULATION SATELLITE (*SIMSAT*)

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Systems Engineering

Michael P. Hanke, B.S.E.E.

GS-13, USAF

December, 1998

AFIT/GSE/ENY/98D-1

DESIGN OF THE COMPUTER SUBSYSTEM FOR THE AFIT SIMULATION SATELLITE (*SIMSAT*)

Michael P. Hanke, B.S.E.E.
GS-13, USAF

Approved:

	7 DEC 98
Lieutenant Colonel Stuart Kramer Thesis Advisor	Date
	7 Dec 98
Dr. Curtis Spenny Committee Member	Date
	07 Dec 98
Captain Gregory Agnes Committee Member	Date

Preface

While only one author is listed on this volume, the research was part of a larger team effort. The second part of the total system design effort will be completed in March 1999. The computer subsystem design effort documented here was not developed in a vacuum, but required the support of the five authors responsible for the second volume [8]. Ideally the simulation satellite (*SIMSAT*) design effort would have been documented in a single document, but academic imperatives required this author graduate earlier than the rest of the team, so some part of the overall design had to be completed early. As fate would have it, the author's area of expertise and the subsystem to be implemented first coincided.

As will be seen in this document, maintaining a total system perspective in the design process was not sacrificed to provide an expedient research topic. In fact, based upon the way the process was executed over the past few months, the computer subsystem would likely have still been a little ahead of the rest of the system design effort because a significant portion of the computer subsystem became available before the team began their design effort. The availability of DSPACE equipment early in the process established it as the baseline for all the engineering design studies the team conducted. The final design outcome was to use the DSPACE system, but not until every design decison confirmed DSPACE as the best alternative available.

Acknowledgements

Anyone that thinks they can conduct research without any outside assistance deceives themself. Support is needed in a number of areas, among them are the establishment of reasonable yet meaningful research goals, maintaining focus on those goals, finding the obvious-after-the-fact answers to seemingly tough problems, choosing (and using) the right tools, and dealing effectively with the inevitable difficulties of everyday life.

The first group that deserves recognition is the rest of the *SIMSAT* design team. Beyond covering for my ignorance in satellite system design, everyone's willingness to support the team and work through the difficult personal and academic times we all faced was invaluable. To accomplish this without the rest of the team would have been unimaginable.

Each member's willingness to find value in others' perspectives and help each other was what made this possible. The fact that a significant portion of the first two chapters of both documents were co-developed reflects that spirit of trust and cooperation. It would have been easy for any of us to insist on doing things "my" way. But we all learned that working as a team made dealing with complex problems more tolerable.

As advisors, LtCol Kramer and Capt Agnes provided the technical and academic support required to effectively address the issues and develop a meaningful product. Unflappable, LtCol Kramer found the perfect balance between teacher, mentor, and customer, effectively shepherding the team through the complicated development process. Capt Agnes always managed to challenge our technical understanding of the design effort without making us feel inadequate. Their ability to help us balance the many obligations we had vying for our attention never ceased to amaze me.

Without the right tools, a design effort such as this would have been difficult, if not impossible. Jay Anderson, Bob Bacon, and the other lab technicians provided invaluable assistance in finding the right equipment, ordering it, and helping us set it up in a timely fashion. Their efforts made the implementation of the system smoother. And since nothing ever goes the way it should, I cannot thank the experts at dSPACE, Inc. (Vivek, Sukesh, and Joe) and in AFIT/SC (SrA Ragsdale, Lisa Gilbert, and others) enough for pulling us back from the brink of destruction as many times as they did.

And last, but certainly not least, my family. Judy, Justin, and Alicia—words cannot begin to express my appreciation for all your support and encouragement through this long, arduous process. While you may not have been doing any of the head-scratching or keyboard pounding, this would not have been possible without you picking up the slack my frequent absences and long hours created. Your saintly patience throughout this seemingly endless process did not go unnoticed. Finally, our Maggie Mae. While she wasn't here to see the rest of the family "return to normal," her obvious daily joy at being one of us all those years helped us get through the tough times—here's to you, "Mooners!"

Michael P. Hanke

Table of Contents

List of Figures

List of Tables

Abstract

This document details the systematic development of the computer subsystem for the AFIT Simulation Satellite (SIMSAT) from Concept Exploration to Implementation. This subsystem design effort was conducted as part of a larger four-phase team design effort to implement a fully functional, user-friendly test facility to meet current AFIT teaching and research requirements. Once *SIMSAT* system integration is complete, control laws will execute on the free-floating "satellite" to reduce communication systems overhead and its impact on control law execution, thereby allowing for more robust control system development and execution. Control law development, simulation, and interactive control of the system will occur through a "ground station" graphical user interface to enhance simulation capability and provide for more intuitive system control. The final subsystem design decision made in this effort was to implement the computer subsystem with an integrated hardware/software, commercial, off-the-shelf system. Detailed information on the design process, computer subsystem implementation, lessons learned, suggestions for future work, and procedures for system use are included in this document.

DESIGN OF THE COMPUTER SUBSYSTEM FOR THE AFIT SIMULATION SATELLITE (*SIMSAT*)

I. Introduction

1.1 Overview

In recent years, the U. S. military has increasingly realized the importance of space as a strategic and tactical resource. Satellite imagery, GPS-guided munitions, missile warning, and global communications for deployed forces exemplify the value of space assets in defense planning. Recognizing the value of these significant resources, the USAF has shifted its focus from being strictly an air force to becoming the premiere air *and SPACE* force by the year 2025 [62]. Accordingly, the Air Force Institute of Technology (AFIT) responded by developing curriculum and conducting research to support space operations and cutting-edge space technologies [34]. Unfortunately, much of the space-related work at AFIT has been limited to computer simulation or stationary laboratory experiments due to a lack of representative space hardware. Consequently, AFIT needed to augment its laboratory facilities with a more realistic space-platform simulator. With a realistic satellite simulator, AFIT would be able to implement practical experiments, demonstrate fundamental motion principles, and extend Air Force research capabilities. Additionally, a realistic satellite simulator would provide a hands-on learning tool to enhance the educational experience of AFIT students, particularly in the area of satellite attitude dynamics.

To initiate the development of the simulator, AFIT purchased a Tri-Axis Air Bearing System from Space Electronics, Inc. of Berlin, Connecticut. The system consists of an air bearing (spherical rotor, hollow shaft, and mounting flanges), a pedestal, and an air compressor. Figure 1.1 shows the free-floating portion of the air bearing assembly. Compressed air flows inside the pedestal (at approximately 75 psi) to six small jets in the air bearing

cup. The spherical rotor then floats above the cup on a gas film less than 0.0005 inch thick. The air bearing is capable of 360° of yaw (rotation about the vertical axis), 360° of roll (rotation about the horizontal axis), and ±25° of pitch (tilt in the vertical plane). The air bearing can support objects weighing up to 300 pounds [57]. Not capable of any motion without attached components, the Tri-Axis Air Bearing System served as the "backbone" for the development of the AFIT satellite simulator.

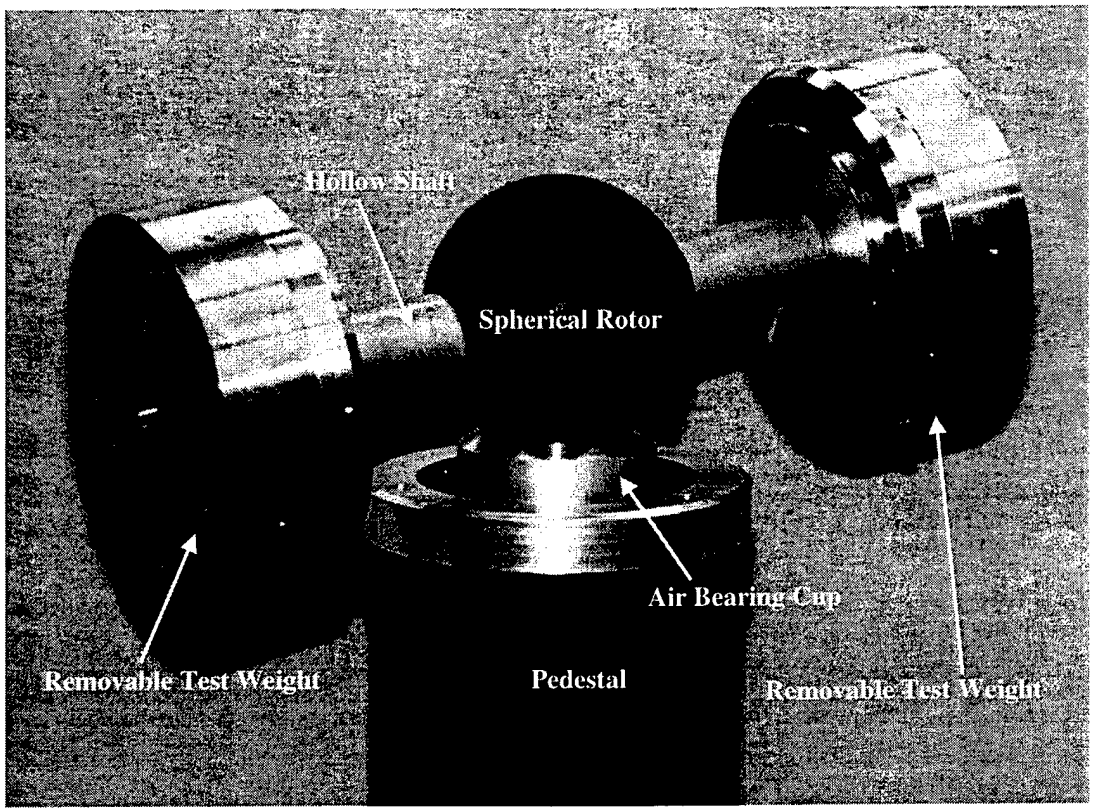


Figure 1.1 Air Bearing Assembly [8]

To implement the total system, two independent, yet synergistic, design efforts were undertaken by a team of six AFIT Graduate Systems Engineering students: the computer subsystem design effort documented here and an overall design effort documented in [8]. Throughout the system design process, the two efforts were closely tied, but the logistics

that drove the need to separate the efforts added some interesting twists to implementing the design.

This chapter provides a top-level description of the problem AFIT faced, the path taken to solve that problem, and a short list of assumptions made to narrow the scope of this research. The chapter concludes with a concise description of the problem and objectives for this research effort, a summary of the final design solution, and an overview of the rest of the document.

1.2 Initial Design Requirements

To close the gap between space operations theory and application, AFIT tasked the 1999 Graduate Systems Engineering Team to develop a satellite simulator (which became known as the SIMulation SATellite—*SIMSAT*) using the air bearing assembly (Figure 1.1) as the foundation of the solution. The team's charter was to formalize the customer's requirements for the simulator, design a system to meet those requirements, order parts, and perform as much of the system integration/implementation as possible before March 1999.

As with any development effort, this one began with a problem to be solved, a need to be filled. To provide the team an initial focus, the customer (the team's academic advisors) developed a "statement of need" as a one-line summary of the problem to be tackled:

AFIT needs to simulate satellite behavior with as much fidelity as possible

This needs statement served as a starting point from which the team developed more detailed design requirements.

The first meeting of all the team members and their AFIT advisors allowed the team to "sit down with the customer" and determine more details of the system requirements. The principal users of the simulator, the team's academic advisors, indicated *SIMSAT* should [33]:

- support research and educational needs

  - perform pure and "dual" spin experiments
  - support three-axis rigid and flexible structure experiments
  - host a variety of experimental payloads

- be simple to use
  - setup and run meaningful experiments with only one researcher and one technician in less than a week
  - display experiment data to allow intuitive real-time data analysis
  - easily store data to allow for future replay/analysis

- be safe—as required, the following design techniques should be employed:
  - highly reliable and/or redundant critical components
  - containment of components that could fail catastrophically

- stay within budgetary and scheduling constraints
  - total initial costs should remain under $100K (additional funding may be possible, but will add to the total development time)
  - the facility should available by 1st Qtr, FY 00

Even from these top-level needs, it is clear the *SIMSAT* problem is multi-faceted. So how does one ensure simultaneous coverage of all the concerns that need to be addressed? How can one make sure that no critical aspects of the system get overlooked? Given several alternative solutions, how can the "best" alternative be selected? The answers to these questions, and others, form the foundation of the formal process known as "Systems Engineering." Some of the "variations on a theme" include "Systems Engineering is ..."

> ...a process employed in the evolution of systems from the point when a need is identified through production and/or construction and ultimate deployment of that system for consumer use. [6:21]

> ...concerned with the design and analysis of the whole system to achieve optimum results. [45:US Naval Academy Definition]

> ... a top-down, life-cycle approach to the design, development, and deployment of large-scale systems ... to meet the effective needs of users ... in a cost-effective, high-quality way. It uses the engineering thought process to analyze, model, and solve inter-disciplinary problems. [45:West Point Definition]
>
> ... concerned with understanding the entire system, including its internal structure, and its interaction with external variables. [45:AFIT Definition]
>
> ... [a holistic attack of problems] to ensure a balanced treatment of all components and their interactions ... [45:Case Western Reserve Definition]
>
> ... an inter-disciplinary approach encompassing the entire technical effort to evolve and verify an integrated life-cycle balanced set of system, people, product, and process solutions that satisfy customer needs. [45:MIL-STD-499B Definition]
>
> ... [a process that] must ensure delivery of a system optimized to satisfy mission requirements that has the greatest possibility of success at the lowest cost. ... [It] can be viewed as the technical arm of program management. [45:Martin Marietta Definition]
>
> ... [an] iterative, but controlled process, in which user needs are understood and evolved through increasingly detailed levels of requirement specification and system design, to an operational state. [45:IBM Definition]
>
> ... [a process that] integrates all the [appropriate] disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation. [It] considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs. [64]
>
> ... the design, production, and maintenance of trustworthy systems within cost and time constraints. [49:10]

Clearly, there is no single commonly held definition for Systems Engineering, but they all share a common theme: Systems Engineering is not simply another engineering discipline [6:20]. Rather, it is a formal, multi-disciplined, iterative *process* for systematically developing customer needs, generating potential solutions to those needs, and determining which alternative in that solution space would most effectively satisfy the customer. An effectively implemented Systems Engineering (SE) process would help ensure critical issues are addressed at appropriate times, and would aid any design team in managing complexity [49:9].

1.3 Scope

As already mentioned, the ultimate goal of this total design effort was to implement a high-fidelity satellite simulation facility. Since the list of desires in Section 1.2 was the only information initially provided to the team, the team's first task was to step back and determine what AFIT's detailed requirements were. To accomplish this task, the team's design work was divided into two major efforts: the computer subsystem design, and the design of the other subsystems, culminating in *SIMSAT* integration. This report documents the computer subsystem design effort, which directly supports the remaining satellite design effort (to be published in early 1999 [8]). The reason for this split was manifold and is explained in more detail in Chapter III, starting on page 3-2.

With the problem thus limited in scope, it needed to be further constrained by what issues the team could control in the design and which issues were outside the domain of potential solutions they could develop. Determining the context of the system provided that focus.

1.4 System Context

To even the most casual observer, it is clear that satellite systems have two principal parts: the portion based in space (or the "Satellite") and the earth-bound portion (or the "Ground Station"). With this intuitive separation in traditional satellite systems, it made sense to notionally consider *SIMSAT* the same way to establish its boundaries. Figure 1.2 pictorially represents the top-level functionality of these system elements. The only issue to resolve was where the software controlling the satellite would execute: "on the satellite" or "on the ground"—there are tradeoffs with either choice. Determining the correct solution to that problem became the primary focus for this research effort and is addressed during *Detailed Design*, Chapter IV.

Once the system context is defined, the next step in most SE processes is to effectively decompose the large, complex problem into a smaller set of more manageable problems. To ensure no specific space system design requirements were overlooked, the team began their efforts by considering the information, data, and techniques presented in

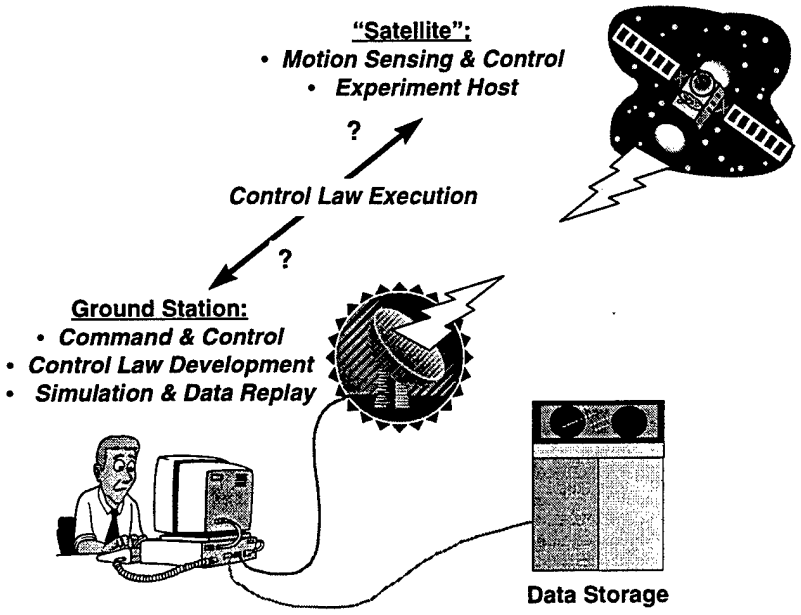


Figure 1.2 Top-Level Representation of *SIMSAT*

the *Space Mission Analysis and Design* (SMAD) text[1]. As Figure 1.2 infers a system context, the team decided to functionally decompose the system along the lines specified in the SMAD text [35:287]: Attitude Determination and Control (ADACS), Power, Command and Data Handling (C&DH), Communications, and Structures. The *SIMSAT* subsystems were defined as:

- **ADACS:** provides determination and control of attitude and orbit position, plus pointing of spacecraft and appendages (in the *SIMSAT* design, this subsystem also includes the Propulsion functions required to provide thrust for adjusting attitude and managing angular momentum)
- **Power:** generates, stores, regulates, and distributes electrical power
- **C&DH:** processes and allocates commands; processes, stores, and formats data[2]

[1]Even though the SMAD book [35] was intended for systems that actually deploy in space, the team frequently consulted it throughout the design effort.

[2]As *C&DH* is a more problem-specific term for the *SIMSAT* computer subsystem, that designation will be used throughout the rest of the document to refer to the computer subsystem

- **Communications:** communicates with the ground (in *SIMSAT*, conveys the signals to/from the C&DH subsystem and the sensors/effectors[3])
- **Structures:** provides support structure and moving parts

Section 2.2.4.1 (page 2-15) further elaborates on these subsystem definitions and the system context.

Figure 1.3 provides a more complete representation of the top-level context diagram of the *SIMSAT* system. The figure illustrates the functional decomposition of the system and identifies obvious aspects of the environment (those things the team had to consider, but had little or no control over). Further documentation of each part of this figure will be addressed in Section 2.2.2.5 (starting on page 2-6).

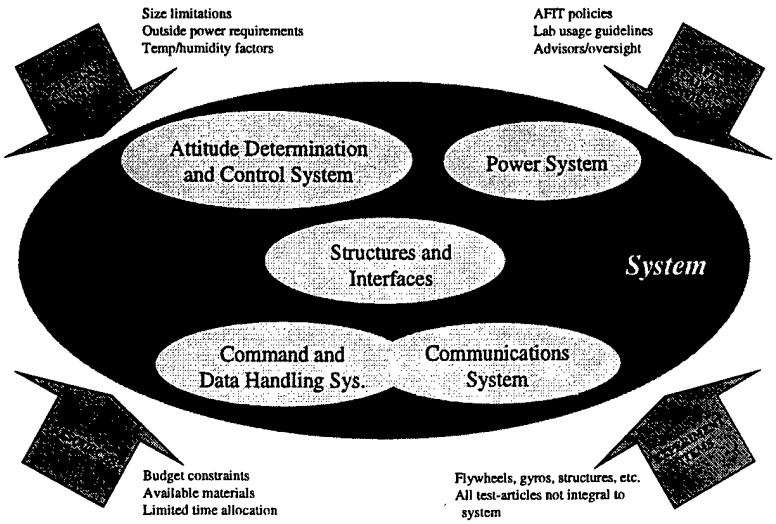


Figure 1.3 *SIMSAT* Context Diagram [8]

One item of interest in Figure 1.3 is the overlap shown between the C&DH and Communications subsystems. After working on the *SIMSAT* design for short time, it became

[3]Effectors are those parts of the system that cause things to happen. In this case, they could be momentum exchange devices, thrusters, etc.

clear there was an inexorable link between the two subsystems. The C&DH subsystem was so dependent upon the Communications subsystem that it drove many of the Communication subsystem requirements—after the *Concept Exploration Phase*, the C&DH system was designed with the assumption that the communications subsystem would support it[4]. Further rationale for the decision to unify the C&DH and communications subsystems will be covered in Chapter II.

1.5 Approach

While considering each subsystem as a part of the integrated whole, the subsystem designs went through a collaborative, systematic process to find the best way to implement the system. Each subsystem was designated at least one subject matter expert to refine the subsystem requirements, research alternatives, and gather data for each potential alternative. Alternative *SIMSAT* solutions were then defined by permutation of the subsystem alternatives using the Strategy Generation Table (SGT) approach mentioned in [28:47]—system solutions are developed by choosing one of each of the subsystem alternatives and integrating them. These composite, integrated *SIMSAT* system solutions were then evaluated to determine how well user values were met—attempting to select "optimal" individual subsystem alternatives was deemed inappropriate.

Using this technique, the pool of potential system solutions could be enlarged by adding different subsystem alternatives and generating additional *SIMSAT* permutations (an evolutionary approach to finding better solutions). This process could also have led the team to revolutionary solutions: if their research to improve sub-optimal subsystem alternatives led to innovative solutions that crossed the arbitrary functional boundaries the team originally established, a whole new set of solutions could result. For example, if they decided that "smart" motors and sensors, with their own dedicated control software, solved some nagging ADACS problems, adding that subsystem point solution would generate an

[4]In other words, aspects of the Communication subsystem design evaluated, selected, and integrated into *SIMSAT* in the follow-on effort [8] were effectively constrained by the requirements imposed by the C&DH subsystem selected in this portion of the design effort.

entirely new set of alternatives—the original "digital, computer-assisted control system" decision would likely no longer be appropriate.

As mentioned before, the formal, iterative process to go from requirements to an implementation of the best integrated system is covered by the SE process. How the team provided for "independent" assessment of the computer subsystem is discussed in Section 3.2, starting on page 3-2.

1.5.1 Systems Engineering Processes.

In practice, most complex problems are multi-faceted. The SE Process is intended to help solve these types of problems. One of the purposes of a systematic process is to effectively manage problem complexity [49:9], so it is clear the complexity of considering all the issues, across every subsystem, from identification of requirements to system implementation, must be addressed. Clearly, partioning problems functionally helps, but as the design evolves, the level of design detail increases, and a different set of problems need to be solved. But, if one could develop a standardized way of dealing with all problems, the complexity of problem resolution would be reduced to manageable, systematic sub-problem resolution, each defined by the expertise required, the level of design detail, and what part of the "problem-solving process" is being applied.

A classic approach, referenced by much of the literature on SE processes, was developed by Hall nearly thirty years ago [26]. As mentioned in that article, "morphological analysis" was a term coined to describe a process whereby one could "decompose a general problem or system into its basic variables, each variable becoming a dimension on a morphological box." Hall defined the dimensions of his "box" as:

- **Knowledge:** those "body of facts, models, and procedures" utilized during the process.
- **Time:** a coarse division of the total process into major decision phase points reflecting the systems life-cycle.
- **Logic:** a fine division of the total process into a set of problem-solving steps to be followed in every phase of the design life-cycle.

While Hall is not the only person that developed a comprehensive SE process (Sage, in [48], presents a comprehensive review of other processes developed through the late 1970's), it is frequently the standard against which others are compared.

1.5.1.1 Knowledge Dimension. Hall's article defined this dimension to reflect an ever-increasing level of detailed knowledge as one advanced along this axis. However, the team determined there would be no impact to the design effort if they strictly used this dimension to list the technical specialties brought to bear on the problem. While Section 2.2.2.9 (page 2-10) provides a complete list of expertise used for the total *SIMSAT* effort, the principle disciplines required for the total design effort included:

- **Systems Engineering Management**
- **Electrical**
- **Computers**
- **Space Operations**
- **Structures**

Collectively, these skills ensured key issues were not overlooked, and system integration difficulties were resolved quickly and completely.

1.5.1.2 Time Dimension. The next dimension in the Hall process is the coarse, top-level "time" dimension, also known as the life-cycle phases for the development effort. Hall's life-cycle phases, as defined in [49:42–43] and [26:156], are:

- **Program Planning:** identification of the system requirements and types of projects necessary to meet those requirements
- **Project Planning:** focuses on the completion of a number of specific projects that, when brought together, will fulfill the intents of the program
- **System Development:** tasks in this step complete the subsystem designs identified in the project plans above, typically resulting in drawings, interface specifications, and bills of material

- **Production:** all the activities required to physically manifest the design solutions developed in the previous steps
- **Distribution:** taking whatever steps are required to deploy the system to the field
- **Operations:** the focus of the rest of the phases; typically includes periodic system maintenance
- **Retirement:** taking the system out of service, perhaps due to obsolesence (replacement with a more modern system) or life-limitations. If the system was mass-produced, this phase-out will typically occur over a period of time.

1.5.1.3 Logic Dimension. The final aspect of Hall's SE process is the "logic" dimension which describes a standardized "problem-solving process": how do we make sure we are making good decisions? The idea behind describing such a process is to make problem-solving easier—having a consistent process that can be applied no matter what phase of the life-cycle the design effort is in reduces the confusion of process implementation. The steps of Hall's problem-solving process[5], and a summary of the activities in each step (detailed in [49:44–45] and [26:156–157]), are[6]:

1. **Problem Definition:** define the scope of the problem, those involved in helping to solve the problem, the constraints associated with the problem, and any assumptions being made as part of the SE process
2. **Value System Design:** develop a hierarchy of values to allow for analytical comparison of alternative tradable attributes
3. **System Synthesis:** use whatever tools may be applicable to generate as many alternatives as possible. For most *SIMSAT* system-level solutions, the team used the previously mentioned Strategy Generation Table (SGT) technique mentioned in [28:47]. The only alternatives that should be disregarded during this step are those that violate the constraints identified in the first step

[5]Hall states that these steps can be followed in any order. But since each step is based on the results of previous steps, a more realistic approach is to follow the steps sequentially, allowing iteration back to previous steps when corrections need to be made in the products of those steps.

[6]These short-hand descriptions of the *Logic* dimension are taken directly from Sage [49:44–45]. Section 1.5.2.2 (page 1-17) summarizes how Sage's process effectively encapsulates the Hall seven step process.

4. **System Modeling:** determine how to measure each tradable attribute, assessing the range, scale, and units to be measured
5. **System Analysis:** (also called 'Optimization' in some of the literature) collect raw data on the tradable attributes of each alternative, determine how to convert the raw data into common units (or "value"), and determine if any alternatives are dominated (and can be discarded) or dominate others (are at least as good, or better than, other alternatives in every attribute, and always better in at least one attribute) ; consider solution modifications to make "weak" solutions more competitive
6. **Decision Making:** determine the weighting factors assigned to each attribute measured, and the values they support. Uncertainty about the fidelity of the raw data is also factored into the analysis that leads to the recommended ranking of solutions
7. **Planning for Action/Implementation:** document the design solution(s) selected and finalize the information developed during the process/phase to help prepare for the next life-cycle phase (as required)

While these steps were introduced as a part of an elaborate SE process, careful consideration of this "logical" dimension to Hall's Morphological process reveals a procedure that could prove valuable no matter what the problem is. Even "simple" issues can benefit from systematically formulating the problem, analyzing the potential solutions to the problem, and then interpreting the results of that analysis. Such a process can help make any decision a good one (unfortunately, it cannot guarantee a good outcome). Figure 1.4 (page 1-14) pictorially represents a total SE process, adapted from the Hall morphological box [26:159] to reflect a subset of the skills the team applied to this problem.

1.5.1.4 Systems Engineering Processes Summary. In summary, the SE process is multi-faceted: several disciplines cooperatively iterate through clearly defined problem solving steps in every phase of the system life. The "column" in Figure 1.4 illustrates that, for a given process step (i.e., *System Modeling*) in a given life phase (i.e., *System Development*), all relevant disciplines are concurrently brought to bear. Having all the experts involved in addressing the sub-problems reduces the likelihood key issues will

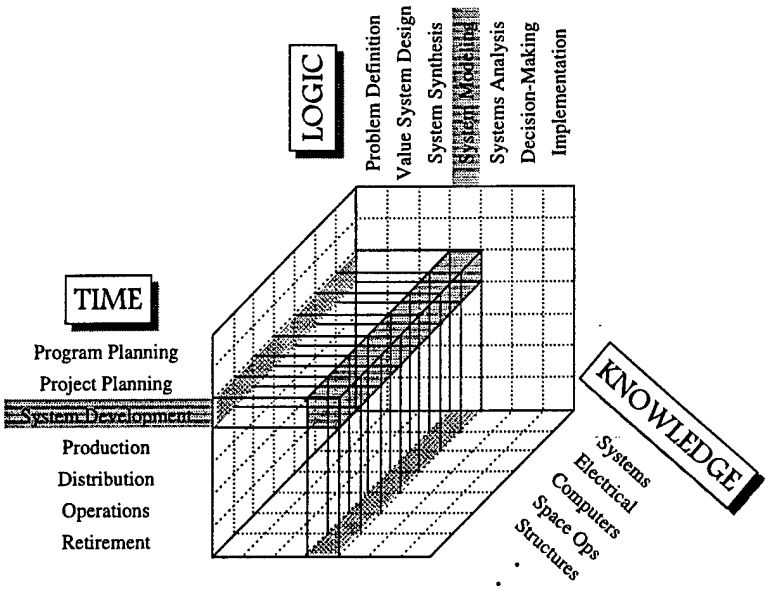


Figure 1.4 Systems Engineering Process Morphology

be overlooked. It also helps the team become more sensitive to system-level interactions they might have otherwise not been considered. This figure also makes it clear that, to actually implement a design, the problem-solving process will need to be applied more than once to evolve a design from a set of needs to an integrated, working system.

1.5.2 Process Considerations. While all the life-cycle phases and problem-solving steps listed above could have been used in the team's development effort, not all of them were really necessary or appropriate. Due to a limited planning horizon (design completion by March 1999), the *Distribution*, *Operations*, and *Retirement* phases were not applicable to the *SIMSAT* design effort. In addition, *Program Planning* was essentially accomplished by the customer before the team was established due to their familiarity with the SE process and the need to procure long lead-time items—the team only had to formalize the requirements the users had already identified. Similarly, the team found the fine resolution of Hall's *Logic* dimension to be excessive for the task at hand.

Clearly all the problem-solving steps, applied during each of the life-cycle phases, would help ensure the results of the design process would provide a great level of detail about the design (a critical concern in high risk projects). However, such a complicated process would be very cumbersome for simpler projects [49:33]. Since one goal of a systematic process is to effectively manage a design's inherent complexity, any process that encumbers the design team and impedes design evolution is undesirable.

An overly detailed or rigid process can lead to overemphasis of process objectives at the expense of completing design objectives in a timely fashion. A simple test of an overly constrained design process is to determine whether the process steps are significantly contributing to a better final design. If process steps are being accomplished strictly for their own sake, they are wasting the design teams valuable time and the clients resources.

Conversely, a design process which is too flexible and unstructured provides an inadequate framework for the design team to conceptualize solutions, compare alternatives, and finally choose a "preferred" system. As all the information above indicates, existence of a formal process can be a significant driver in keeping the design team on track and providing bounds to the seemingly unbounded world of systems design. Thus, a formal design process is a useful tool for managing the complexity inherent in any design problem [49:9], but only if properly selected and implemented.

Somewhat abstract processes are sufficient for simpler projects, especially if the risks associated with the design decisions are fairly low. For example, the risks associated with choosing the wrong location for a nuclear waste site would likely justify the time and money required to implement the full Hall process. Military acquisitions likewise justify using the full Hall process. But designing a kite is not nearly as critical and would not justify the same level of resource commitment. Developing a laboratory test facility, as in this effort, falls somewhere in between. The question indeed arises, what is the best systematic design process for the problem at hand?

Sage suggests the following considerations when assessing the appropriateness of candidate SE processes [49:68]:

1. proximity and number of participants

2. their experience with SE tasks and the anticipated systems usage

3. cost and schedule budgets

In this design effort, the key participants were all in the same building and were either intimately familiar with SE, the domain of the potential solutions and their usage, or both. And as mentioned before, a significant objective identified by the users was the actual operation of *SIMSAT* prior to March 1999. Thus, the design team was faced with a moderately complex problem to be taken from conceptualization to integration (and operation, if possible) in a very short period of time. The team needed to make several iterations through whatever design process was chosen/adapted. A straight-forward, formal (yet flexible), design process had to be developed to handle this schedule-driven design problem. The next two subsections detail the *Time* and *Logic* dimensions of the process they defined.

1.5.2.1 Time Dimension. Largely due to time constraints, a simplified variation of the morphology depicted in Figure 1.4, was needed to reduce the process complexity. The team found a partial solution in a process developed by Sage [49:32–33]. There, he points out that "typical" SE efforts can be condensed down to effectively three basic life-cycle phases (using Sage's terms):

- **System Definition:** The user perceives a need to be fulfilled. Requirements for the solution are formalized. If required, this phase would see the requirement put out for bid, evaluation of the responses returned, and selection of a vendor to develop a solution to the problem [49:33–34].
- **System Design and Development:** Between choosing a vendor and the critical design review, the vendor carries the system from requirements to detailed design (a significant effort). Some of the activities performed include system decomposition into subsystems, detailed subsystem specifications (including detailed interface "drawings" and testing requirements), design and fabrication of subsystem components (hardware and software), implementation and testing of subsystems, followed by systems integration and test, and finally, development of documentation and system training for the user [49:35–36].

- **System Operation and Maintenance:** System fielded by the vendor in conjunction with the user. To achieve full capability, the system must complete *Acceptance Testing* (in a "controlled" lab environment) which could lead to system changes to meet the users *REAL* needs (expected to be minor changes since the user should have been adjusting the vendor's course, as required, earlier in the life-cycle). Formal acceptance/operational deployment comes only after *Operational Testing* (duplicating field conditions). During operational use, other needs may be identified, leading to another design life-cycle to modify the system. Documentation of the system development plays a key role in this phase—thorough documentation makes it easier to modify the system and helps prevent unintended consequences [49:36–37].

Clearly, these aggregated life-cycle phases encompass the seven detailed Hall life-cycle phases listed previously. Table 1.1 summarizes the Hall life-cycle phases captured in each phase of the "typical" Sage SE process [49:42].

Life-Cycle Phases	
Sage	**Hall**
System Definition	Program Planning Project Planning
System Design & Development	System Development Production Distribution
System Operation & Maintenance	Operations Retirement

Table 1.1 Sage vs. Hall Life-Cycle Phases

1.5.2.2 Logic Dimension. In addition to the complexity required to implement a full Hall life-cycle approach, his detailed problem-solving steps made the design process very cumbersome. The detailed, seven-step problem-solving process helps make sure all the important issues are considered. But, early in the *SIMSAT* design effort, the team found themselves trying to exhaustively complete one step before moving on to the next—they needed to find another model that would provide structure without being so confining. Again, Sage had a "coarse" problem-solving process based upon the Hall

process [49:44–45]. The Sage process effectively captures the seven step Hall problem-solving process in three sub-categories:

Issue Formulation. As a starting point for any design life-cycle phase, identification of problem characteristics and relevant issues must be accomplished. The following information should be identified or at least considered by the design team at this stage: actors involved in this stage of the design process, groups affected by the issues or proposed solutions, fields of knowledge required to solve problem, specific needs addressed by the problem, design alterables, imposed constraints, and cost and schedule considerations. The problem itself is isolated, quantified, and clarified. The system (or subsystem) to be developed is delineated from its surrounding environment. This abstraction of the environment consists of those elements which significantly interact or affect the system (or subsystem), but are beyond the design teams sphere of control (at this stage). Determination of "what is the system?" and "what is the environment?" allows identification and classification of important interfaces between what the team can control and what it cannot for the *SIMSAT* design.

Once needs are identified, discernment of the user's values for the solution to this problem begins. This process, often called the *Value System Design* (or VSD), is the formulation of key user values that will be used to guide the search for alternatives. The framework developed during this task, frequently called the *values hierarchy*, can then be used for comparison of alternatives. The process helps the user formalize what is important to them.

The value hierarchy itself can vary in form. For some problems, qualitative assessments are sufficient, so formal, quantitative measures are unnecessary. For other problems or subproblems, the value hierarchy may be the enumeration of specific measurables by which all alternatives will be judged. Thus, the determination of a preferred solution must be accomplished quantitatively [28:23]. From a top-level systems architecting perspective, it is highly desirable to create a value hierarchy with associated measurables consistent with the level of detail appropriate to the particular life-cycle phase.

This approach to the VSD process can then be carried to each problem or subproblem encountered as the design evolves and goes through repeated iterations of the design process, adapting it to the level of detail required. In some instances, a formal values hierarchy may not even be required. In these cases, alternatives which are feasible (within constraints) may be chosen without searching for the preferred alternative. This approach is desirable for a variety of reasons: tight schedule constraints prevented detailed alternatives comparisons in each life-cycle phase, lack of reliable modeling data prevented precise comparisons, and the value of a preferred solution may be comparable to that of other feasible solutions. In that case, all the proposed alternatives would progress to the next life-cycle phase.

The last task in *Issue Formulation* is *System Synthesis.* A set of alternative solutions are developed through research, brainstorming, reverse engineering, heuristics, and other means. These alternatives should appear feasible, but need not fully comply with constraints at this stage[7]. Generating these alternatives is at the core of systems architecting [47:12–13]. Functional breakdown of the problem at hand is often a helpful first step, but there is no explicit methodology to generate solutions based on a given problem. This step is at the root of systems design.

Analysis. The Sage *Analysis* step includes the system modeling and evaluation necessary to make decisions regarding which alternatives to pursue further. The first aspect of this step, *System Modeling*, is the development of means to evaluate the "performance" of each alternative. Models are system abstractions used to generate data for each of the measures for each value. The second step, *Systems Evaluation*, is the use of those models to quantify the measures for each alternative. At this stage of the process, alternatives may be refined as required to improve performance.

Analysis can take many different forms. Construction of simulations, itemization of costs, development of prototypes, and engineering estimates are just some of the modeling methods available to the design team to quantify performance measurables. The goal of

[7]Later investigation could reveal a potentially infeasible solution was in fact feasible, or a feasible alternative may prove to be infeasible.

System Analysis is to provide data to the *Decision-Making* sub-step of the *Interpretation* step. Therefore, modeling is only necessary to the degree required to differentiate system alternatives. For the *SIMSAT* design problem, significant mental modeling, engineering estimates, and research were used to quantify the measures at a level sufficient to permit comparison of alternatives.

Interpretation. This step uses the information gained from the *Analysis* step to make decisions and proceed to the next phase of the design process. In the *Decision-Making* sub-step, an alternative (or set of alternatives) is selected based on the analysis data and the values hierarchy defined earlier. Alternatives will typically be better in some aspects, but less desirable in others. But if they exist, dominated solutions should be identified and discarded from consideration[8]. Decision-making tools, such as multi-attribute utility theory, and value weighting are then used to settle on a preferred solution set. Since there is an element of risk and uncertainty in the results obtained through this analysis, these risks and uncertainties must be considered by the CDM when making his decision. Regular interaction with the customer/CDM is critical throughout this stage to determine what techniques he is comfortable with using to help quantify the risk involved in his decision.

Once the set of preferred alternatives is identified, planning for the next life-cycle phase is necessary. The outcome of the design process to this point should be effectively documented to clearly communicate the decisions made and the impact of those decisions. Looking ahead to the next life-cycle iteration, the detailed allocation of resources and the next iteration of the design schedule are generated. The design process then begins the next life-cycle phase, in which the problem is recast to address the next level of detail for the selected solution set. If this is the final design iteration, the process results are documented and implemented.

[8]As "domination" implies at least one of the other alternatives is better than the dominated alternative, it is a waste of time to continue evaluating it—it can never become a competitive alternative. The remaining alternative solutions are naturally termed the "non-dominated solution set."

Table 1.2 (below) summarizes which Hall problem-solving steps are encapsulated by each Sage step[9]. Each iteration through the system or subsystem-level design process incorporates these steps. The tasks (or Hall steps) within each Sage step may be over- or under-emphasized as required, depending on the problem or subproblem. Thus, the design team would not be encumbered by implementation and documentation of the formal seven step process for every problem or subproblem encountered.

Problem-Solving Steps	
Sage	**Hall**
Issue Formulation	Problem Definition Value System Design System Synthesis
Analysis	System Modeling System Analysis
Interpretation	Decision-Making Document/Plan for Next Phase

Table 1.2 Sage vs. Hall Process Problem-Solving Steps

This process clearly accommodates the "time-to-market" approach the team needed to use better than the full Hall process. For example, Sage's approach could support less emphasis on *System Synthesis* or *Analysis* for certain subproblems in favor of requirements determination, a particularly important consideration during early life-cycle phases. Similarly, for later life-cycle phases, the Sage process would support significant effort on the *Synthesis*, *Modeling*, or *Decision-Making* tasks.

It is important to note that although the process appears linear, feedback loops are permitted within every step and between steps. For example, during *Analysis* it may be discovered that a significant user concern was overlooked during earlier discussions. This team could then return to the *Issue Formulation* step and factor the concern into the values hierarchy. Moreover, if requirements prove to be very difficult, or too costly to meet, they could be challenged and the CDM could redefine that aspect of the problem if he felt such an adjustment was more prudent than maintaining the original requirement.

[9]Within the chapters, the Hall steps will be used to further demark section divisions. To avoid confusion regarding the Hall steps within each Sage step, the Hall steps will be called "tasks" throughout the rest of the document.

1.5.3 *Design Process Defined.*

An activity matrix [49:46] is a helpful tool to keep track of where you are in the overall design process. The difference between Hall's 3-D Morphology and this tool is the elimination of the knowledge dimension. A complete listing of disciplines is important to the process to make sure no key area is overlooked (and Section 2.2.2.9, page 2-10 provides such a list). But as can be seen by the way Figure 1.4 is drawn (along with the accompanying explanation of how it is applied), the "active" part of the SE process is the application of the logic dimension steps through each of the time dimension phases. The activity matrix in Figure 1.5 summarizes the logical and time dimensions of Hall's morphology as adapted to the team's particular problem using the Sage technique.

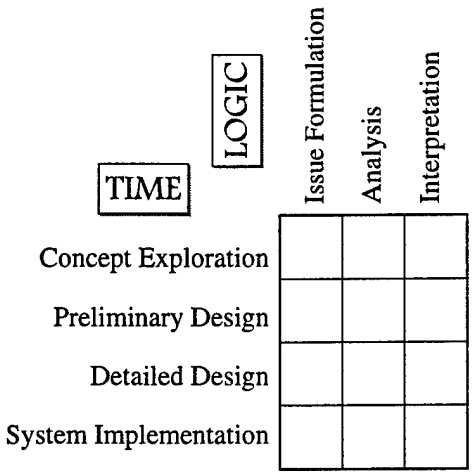


Figure 1.5 *SIMSAT* Team Activity Matrix

This figure will appear near the beginning of each chapter to provide a frame of reference for the aspects of the process covered by that chapter. Each dimension will be addressed in the following subsection.

1.5.3.1 *Time and Logic Dimensions.*

Because of the relatively short planning horizon and resource limitations of this design project, the design team divided the *SIMSAT* development effort into four life-cycle phases: *Concept Exploration*, *Preliminary*

Design[10], *Detailed Design*, and *Implementation*. As mentioned before, other life-cycle phases often seen in the acquisition of large-scale military systems [2] were not addressed. The team defined the "coarse," life-cycle phases of their process as:

Concept Exploration. Once a need has been identified and initial requirements have been defined, the system design process enters the first stage of the system life-cycle. This phase includes refinement of system requirements, along with an exploration into various concepts which can be designed to meet these identified requirements. Emphasis is on top-level system architectures, with detailed design decisions avoided at this point. The focus of this life-cycle phase is on identifying and differentiating broad solution classes. Through initial modeling, research, trade studies, and CDM inputs, a class (or classes) of solutions may be identified which stand out from the rest (i.e., the non-dominated solution set). This solution class(es) can then be further refined and investigated in the next life-cycle phase.

Preliminary Design. In this life-cycle phase, the solution class(es) identified in *Concept Exploration* is (are) further refined. Subsystem level requirements are defined in this phase. Trade studies, research, and system modeling are again used to determine which subsystem types best meet the system goals. The output of this phase includes a system architecture complete with identified subsystem types, subsystem requirements, and notional interface definition.

Detailed Design. The subsystem designs are further refined during this phase. Detailed trade studies could be used to determine the exact subsystem architectures making up the overall system. Integration/interface issues are resolved in this phase and the overall system is completely defined, subject to change as system test and evaluation dictate. The product of this life-cycle phase is a detailed functional system architecture with subsystems designed and integrated. Drawings are finalized to docu-

[10]For this research effort, the *Preliminary Design* phase took the form of a trade study to compare the DSPACE system to other alternative solutions.

ment power, physical, and signal interfaces for each of the subsystems as well as how the integrated system fits together and functions as a unit.

Implementation. The final product is built by purchasing components, fabricating sturctural support elements, and assembling the subsystems into an integrated whole. User's manuals are created and validated. Suggested improvements are described and documented for future user's consideration and to provide a roadmap for future designers. Unresolved design issues are addressed.

During this effort, however, few problems arose, so the documentation of this phase addresses more the programming and interface issues of the implementation than looking at each of the problem-solving steps. The users values and concerns were considered in each C&DH subsystem implementation phase, but the focus of the documentation is on the actual implementation steps taken.

Because the Sage problem-solving steps adequately captured all the steps the team needed, without introducing any additional burden, they were directly implemented (as documented in Section 1.5.2.2, starting on page 1-17). This "fine," logical structure, along with the "coarse" time structure above, defined the activity matrix used to guide the team through this development effort (Figure 1.5).

1.5.3.2 Design Process Summary. Simplification of both the Hall lifecycle phases and problem-solving steps were appropriate for the problem at hand. As such, the Sage consolidation of problem-solving steps reduced process complexity and legitimized the significant recursion that occurs between the closely related Hall tasks within a single Sage step. This abstraction, then, provided some intellectual justification to begin a Hall task with the initial, obvious products of a previous Hall task without waiting until all the products of the former task were available. This saved significant time since many overlooked issues from previous Hall tasks would be detected within the confines of a given Sage step. This is not to say that, once you get to *Analysis* you cannot iterate back to *Issue Formulation* if you happen to find an overlooked issue. It just means that, if you carefully scrutinize the products of the process for completeness at the end of each Sage

step (vice the end of each Hall task), you will likely have nearly the same completeness you would have using the exhaustive Hall process, but with significantly less documentation and process overhead. The Sage aggregation of steps prior to formal product reviews removed some of the encumberances associated with trying to explicitly follow the standard Hall process and reduced the amount of time spent in internal design reviews.

1.6 Assumptions

Before the team was formed, some research funding became available and parts of a DSPACE control system[11] were purchased. The team's efforts did not assume these DSPACE parts were the only possible C&DH solution, but they did have an obvious economic advantage over any other alternatives in any subsystem alternative comparisons—the costs for these parts of the system were treated as sunk costs (money already spent that cannot be recovered). While some of the pieces may have been adaptable to other applications, their immediate reuse was unknown, therefore any opportunity costs were also uncertain. For these reasons, the DSPACE costs were considered negligible for the purposes of subsystem comparisons.

To evaluate subsystem alternatives during *Concept Evaluation* and *Preliminary Design* phases, the team considered integrated system solutions and selected those subsystem alternatives that dominated the others (i.e., were clearly better than the other subsystem alternatives in at least one attribute, and at least as good as the other alternatives in all the rest of the attributes). During *Detailed Design*, the subsystem choices made in this effort were considered separately from the remainder of the system by assuming a baseline system and then analyzing the impact of varying only the C&DH alternatives. To assess the range of potential values for a given C&DH alternative, the analysis was done with the "best" and "worst" subsystems possible (aggregated again to generate the "best" and "worst" alternatives the team were willing to consider for each subsystem).

One crucial technical assumption the team made regarded the processing power requirements of the system. Based upon previous experiences the CDM had with the

[11] For more information on dSPACE, Inc., consult their web-site: www.dspace.de

dSPACE system, and laboratory test environments in general, the 60 MHz Texas Instruments C40 Digital Signal Processor (DSP) was assumed to have sufficient computing power to permit at least the baseline system to function. The CDM recognized that it might not be possible to control the satellite and the most elaborate experiment he could imagine at the same time, but he was willing to assume the class of processors represented by the 60Mhz C40 processor would be sufficient to meet his initial needs. If needed at some later time, he would purchase a processor upgrade to support more complicated experiments.

1.7 Problem Statement

After describing the overall design effort requirements, the process the team used to meet those requirements, and considering the assumptions described in the previous section, the problem statement for this part of the total *SIMSAT* effort could now be effectively expressed:

> *Systematically design a computer control system and development environment to meet AFIT's* SIMSAT *requirements*

While this statement reflects what was to be accomplished in the total research effort, each chapter in this document provides a problem statement that focuses on the work to be accomplished in that life-cycle phase.

1.8 Proposed Hypothesis

To set a target for this effort, the following statement captures the expected research conclusions:

> *The computer control system and development environment for* SIMSAT *was expected to best be accomplished using a fully-integrated hardware/software, commercial, off-the-shelf (COTS) system to support consolidated or distributed computing. In addition, designing the software architecture so the control laws execute on the "satellite" segment of the system was expected to provide the best systems solution.*

The research effort validated this hypothesis, with the exception of the distributed computing aspect: the hardware/software solution chosen, as purchased from the manu-

facturer, could only support consolidated computing. As the DSPACE automatic source code generation routines produce C code, hand-optimizing the code may provide a means to implement a distributed architecture. This effectively became one of the suggestions for future research in Chapter VI (page 6-5). That recommended research extension could be used to determine the benefits (if any) to implementing a distributed processing architecture.

1.9 Objectives

The objectives of this research follow from the issues raised above. In order of importance, this research will:

1. Establish the computer subsystem for *SIMSAT*
2. Determine the communication requirements necessary to support the computer subsystem
3. Implement a user-friendly development and simulation environment for *SIMSAT*

All these objectives were accomplished, as well as establishing the baseline systems engineering and decision-making tools the team would use in subsequent decisions.

1.10 Decision-Making Tools

To accomplish the systematic development and evaluation of alternatives, the team considered and used a multitude of tools and techniques. The tools used in this effort came from several disciplines—SE is not the only field concerned with a systematic approach to decision-making. In fact, many of the techniques and tools used in the qualitative and quantitative analysis portions of the SE process come from Operations Research and Systems Science [49:10,14].

One decision-making tool mentioned by Clements [7:85–88] is DPL, DECISION PROGRAMMING LANGUAGE[12]. While that tool is well-versed at handling probabilistic uncer-

[12]Produced by ADA Decision Systems, Menlo Park, CA. They can be reached at 415-854-7101

tainty and sequential decisions, it is not designed for the multi-criteria decision-making problems the team faced in this effort. For that, LOGICAL DECISIONS [56] provides a more complete and intuitive tool. It also does a good job of handling the one-way sensitivity analysis required later in the process. But even LOGICAL DECISIONS is not perfect—it did not allow much control over the formatting of the graphs and charts it provides. For simple graphing and table manipulation, MICROSOFT EXCEL was the tool of choice.

1.11 Final Subsystem Design

Once this design effort was complete, the following DSPACE system was selected and implemented to satisfy the user's needs:

- a Simulation PC to support control law development and the graphical user interface for monitoring and controlling the system
- a DS1003 processor board (based upon a 60MHz Texas Instruments C40 Digital Signal Processor [DSP]) to execute the control laws
- a DS2003 analog-to-digital conversion board for 32 sensor inputs to the DSP
- a DS2103 digital-to-analog conversion board for 32 effector outputs from the DSP
- a DS400 AutoBox, a self-contained, power-conditioned (auto-ranging 8–36VDC input), ruggedized enclosure to hold all the above boards; also provides an ethernet port to communicate with the Simulation PC
- DSPACE software development environment—all the tools required to build a control system making use of the above hardware in conjunction with MATLAB and SIMULINK
- a RealMotion PC to support 3-D representation of motion; provides a means for rotating an AUTOCAD model of *SIMSAT* either real-time or from stored data
- a second DS1003 processor board for the RealMotion PC to perform 3-D transformations of the AutoBox data
- two DS820 communication link buffer cards to allow greater separation between the AutoBox and the RealMotion PC (one will be housed in the AutoBox along with the cards listed above)

- the REALMOTION software to support the high fidelity representation of *SIMSAT* activity
- all the interconnecting cables

Appendix D details the installation of all these subsystem components. Chapter IV provides additional information about the various interface requirements these components place upon the rest of the system.

Figure 1.6 shows a block diagram of the notional C&DH/communications environment necessary to implement the final *SIMSAT* control system configuration. It provides a top-level depiction of the interfaces between the C&DH subsystems. As this portion of the *SIMSAT* design effort did not select the wireless system to be used, all results reported in this document are based upon a "wired" system using the standard cabling provided by dSPACE, Inc.

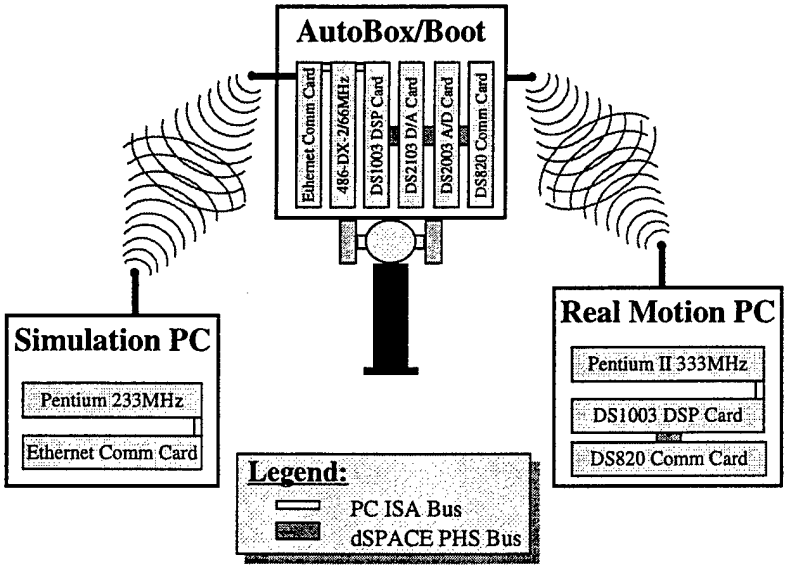


Figure 1.6 *SIMSAT* Computer System Setup

1.12 Document Overview

Like the process the team used to systematically progress from conceptual design to implementation, this report systematically documents the critical steps taken to develop the *SIMSAT* Command and Data Handling (C&DH) subsystem. For documentation purposes, the team evaluated classes of candidate subsystem technologies during *Concept Exploration.* They then went through *Preliminary* and *Detailed Design* phases to select components and finalize subsystem and system-level interfaces. Finally, the team's *Implementation* phase began by assembling the C&DH subsystem (with the ulimate goal of integrating it with the rest of the system as it becomes available) and concluded the C&DH portion of that phase with rudimentary operation of the DSPACE system to validate its functionality. As the rest of the system would likely follow this same evolutionary process, this defined the number of iterations the entire system would need to go through to complete *SIMSAT* implementation. The documentation of the computer subsystem development effort mirrored that evolutionary process—each of the life-cycle phases is documented in a separate chapter.

Chapter II provides the details of the team's *Conceptual Design* effort and lays the groundwork for the rest of the development effort. Chapter III then documents the issues involved with the shift in focus to the C&DH subsystem. Part of that shift was accomplished through a C&DH trade study that assessed subsystems comparable to the baseline DSPACE system purchased before this research began. With the DSPACE choice validated, Chapter IV then documents the *Detailed Design* effort, including the significant quantitative analysis done to support the decision on how to most effectively implement the DSPACE system. An overview of the implementation and operational details for the subsystem is contained in Chapter V. That chapter also includes a description of those aspects of the C&DH system that still need to be handled during overall *SIMSAT* integration. Finally, Chapter VI provides a summary of the entire C&DH research effort and provides recommendations for follow-on research efforts.

The team used the Sage problem-solving steps (without modification) to further segment the documentation. To make Chapters II through V easier to read, each of the Sage problem-solving steps are addressed in a separate section.

Again, as the C&DH subsystem design drove many of the other design decisions, the C&DH design process was broken out into a separate, but coordinated, design effort so it could be completely implemented by the end of Sep 98. Since this document was developed in conjunction with the system-level research effort, significant portions of the first two chapters of this document were co-developed with the 1999 Systems Engineering Team. This was done to ensure some level of consistency in documentation of the team's SE process, and the *Concept Exploration* phase they directly shared. Once this subsystem entered *Preliminary Design*, it began to forge its own path, but was never completely developed in isolation. Sections 3.1 and 3.2 discuss this division of effort in greater detail. To compare the two works, consult [8].

II. Concept Exploration

2.1 Overview

Once a need has been identified and initial requirements have been defined, the system design process enters the first stage of the system life-cycle. This phase included a formal definition of the user's needs and requirements, followed by an exploration into various concepts which can be designed to meet the identified requirements. To effectively manage problem complexity, emphasis during this phase of the design life-cycle was on top-level system architecture issues; detailed design decisions were left for future phases. The focus of this life-cycle phase was on identifying and differentiating broad solution classes for each subsystem. Through mental modeling, research, trade studies, and chief decision maker (or *CDM*; another name for the "customer") inputs, classes of solutions were identified which stood out from the rest. These solution classes were then further refined and investigated in the next life-cycle phase.

While this portion of the C&DH design effort was accomplished at the same time as the rest of the *SIMSAT* design, this document will only summarize those aspects of the total *SIMSAT* effort germane to understanding the decisions made in direct support of the C&DH effort. This portion of the design life-cycle was the first step in the design process, therefore it considered the system at its highest level of abstraction. Figure 2.1 provides a frame of reference for the activities documented in this chapter.

2.2 Issue Formulation

As a precursor to this phase, much of the requirements analysis phase of a typical life-cycle was done by the customer (and documented in Chapter I) before the team was established. This section details those Hall tasks *Problem Definition*, *Value System Design*, and *System Synthesis* appropriate to the design problem during *Concept Exploration.*

2.2.1 Formulation Methodology. To get an understanding of what the CDM expected the system to do, the team had to further define the definition of the

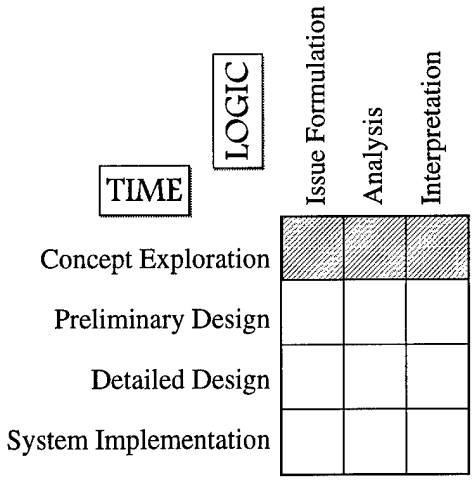


Figure 2.1 Concept Exploration Activity Matrix

problem and what the CDM considered important aspects of that solution. From there, the team began to generate solutions. As this was the first iteration through their System Engineering (SE) process, the preliminary steps of defining the problem and the user's values received the greatest amount of attention, while the solutions the team generated only considered the types of technologies that might be used to solve the problem. Future iterations through the life-cycle addressed greater levels of detail for the *SIMSAT* (and therefore the C&DH subsystem) design.

2.2.2 Problem Definition. This part of the task forms the foundation for the rest of the research effort. While future iterations reconsider the issues developed here, this task in the *Issue Formulation* step in the *Concept Exploration* phase of the SE process requires the most thorough treatment to ensure the problem is completely defined before any signficant design decisions are made.

2.2.2.1 Identification of Actors. To begin the process, the team identified the major players in this design process, and their contributing roles, to ensure success in the design effort through recognition of the concerns, needs, limitations, and expectations of each party.

Lt Col Stuart Kramer and Capt Greg Agnes, USAF. These AFIT instructors were the direct customers of *SIMSAT*, thus their identified needs provided the primary design focus. As sponsors of the project, they also served as the chief decision-maker (CDM).

Systems Design Team. The team, comprised of six AFIT Master's degree candidates in Space Operations and Systems Engineering, was responsible for the design and integration of the system. These responsibilities included the selection of the tools and processes they would use in developing the system and ensuring the system met the needs of the CDM.

Mr. Jay Anderson. An AFIT civilian responsible for resource acquisition and laboratory support, Mr. Anderson was the focal point for facilities management, experimental support, logistics issues, hardware procurement, and safety-related issues.

Suppliers/Vendors. Commercial suppliers were the primary source for hardware and software used in the system design. An understanding of product availability, technological innovations, and customer support of these suppliers was critical to the design effort.

Lab Technicians. A group of AFIT civilians, that work for Mr. Anderson, who were a source of endless experience on what would and would not work for this simulator. They also helped implement the system as the pieces came in. Their expertise was necessary to help find components (from the *Suppliers/Vendors*) to solve some of the design problems the team faced, and avoid purchasing things that would be unsupportable in the AFIT lab environment.

Indirect Customers. Some of the additional customers considered include other AFIT departments, other Air Force agencies, and joint Department of Defense agencies. These indirect considerations play a significant role in the need for a robust and timely design.

These actors, if not directly involved in the process, had to at least be considered in the development of the problem and its potential solutions.

2.2.2.2 Problem Statement. With the "actors" that could potentially be impacted by this effort defined, the team developed a problem statement to capture both the intent of the entire design effort (as specified in Section 1.7), and the goals of this phase of the design life-cycle, in a clear, concise manner. For the entire design effort, the intent of the *SIMSAT* design was to satisfy the needs of the direct AFIT customers (the CDM above), while retaining the capability to meet outside agency needs. Considering then, the previously defined focus for this life-cycle, the team worked against this problem statement:

> *Considering* SIMSAT *is a satellite system simulator to be used as an experimental test bed for AFIT, define the detailed requirements and top-level subsystem technologies for such a simulator to support Air Force/DoD research and provide a sound instructional aid to AFIT instructors teaching satellite dynamics and control.*

This statement provides some initial bounds to the problem, but not many details. The next section defines additional details of the problem context.

2.2.2.3 Problem Scope. The team was tasked to develop a SIMulated SATellite platform (*SIMSAT*), integrating a "flying" test bed with a ground-based control system. The ground-based portion of the system (hereafter referred to as the "ground station") had to be capable of sending commands to and receiving data from the "flying"/space-based portion of the system (hereafter referred to as the "satellite"). In addition, the ground station had to support the development of the control laws, including simulation of the proposed "control laws" prior to actually using them for satellite operation and control. Since this capability will be required any time the satellite characteristics change, the system must provide a user-friendly interface during control law development as well as during system control and the display of the satellite response to commands. The system must also allow for collection and replay of satellite motion and experimental telemetry. The *SIMSAT* design had to be robust enough that a variety of experiments could be accomplished with minimal adjustments. Some experimental areas identified by the customer so far include investigation of control-moment gyros, optimal flywheel orientation design, flexible space structures behavior and vibration control, satellite attitude

and tracking control, and elaborate ground station simulation of the satellite (including hardware-in-the-loop, in-class presentations, technology demonstrations, etc.). The goal was to implement the complete, integrated *SIMSAT* system design by March 1999.

The first iteration through the design process focused on refining requirements and exploration of concepts, and was not intended to result in design, implementation, or evaluation of an actual system. The goal of this life-cycle phase was to understand the problem, gain knowledge in areas needed to attack the problem, identify constraints to the solution, develop fundamental CDM values, and then identify possible classes of solutions based on cost, performance, and other relevant criteria. This identification would then provide inputs into the system design and implementation phases. This first iteration was completed by the end of June 1998.

With the problem statement in hand, and the context of the problem defined, the team worked with the CDM to develop an initial set of needs.

2.2.2.4 ***Initial Needs.*** At this stage of the design, the user's needs were identified in general terms so potential design solutions could be conceived and explored. Actual subsystem requirements and quantitative system specifications were not considered appropriate this early in the design process. The following list summarizes the primary needs for *SIMSAT* at this level of detail, developed from all the previous information.

- Support three-axis orientation control for detailed experimentation. Three-axis stabilization is required for the primary experimental configuration.
- Support dual-spin demonstration experiments, which involve two sections of the satellite simultaneously rotating relative to one another.
- Support pure spinner experiments to demonstrate fundmental principles.
- Robust enough to support unforeseen experiments (include "obvious" support in the way of mass and power margins, flexible mounting points, flexible signal interface capability, etc.).
- Build the system around the air-bearing assembly (the piece of test equipment shown in Figure 1.1 that allows the satellite full rotation in two axes and partial rotation in

the third [pitch] axis). This air-bearing assembly was previously purchased expressly for this project.

- Provide unobstructed satellite rotation—physical connections to the satellite should not exist.
- Develop a ground station to control and monitor the satellite and its experimental payload.
- Provide as much data capability as possible:
  - real-time data acquisition and display to reflect the status of the satellite and its payload.
  - retention of mission data to allow detailed post-mission data analysis of satellite motion and payload information.
- Provide pre-mission satellite simulation capability.
- Provide real-time and post-mission graphical representation of satellite motion. (highly desirable, but not required)

From these system needs, it was clear that some aspects of the system were requirements, while others benefit the user, but were not required. The next two sections address each of those types of needs.

2.2.2.5 Constraints. Constraints generally fall into two categories: "environmental" concerns over which neither the team nor the CDM have any influence, and "design" choices that had already been made. Figure 2.2 identifies those aspects of the environment the team identified early in the design effort. In addition, there is at least one element of the design which has already been made (the choice of the air-bearing assembly). From this perspective, the team identified the following issues as those that MUST be accommodated by any potential system solution:

- facility was physically located in the northwest corner of Room 146, Building 640 (one of the AFIT laboratories)
- system must fit into the available space of approximately 240 square feet

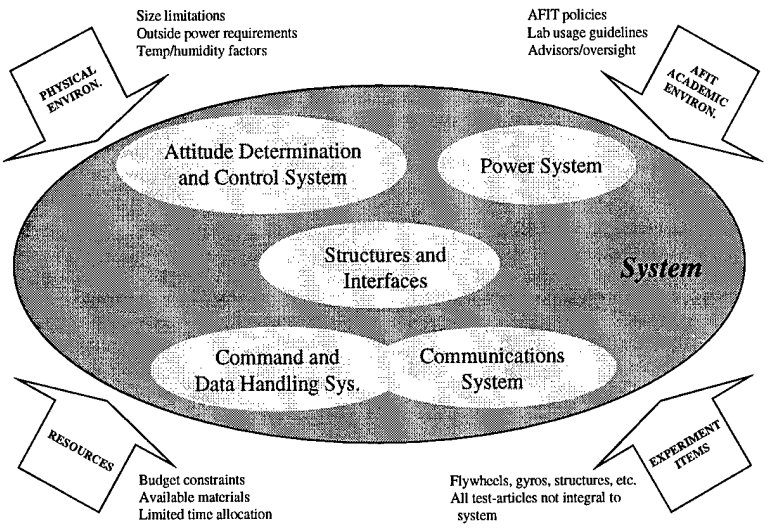


Figure 2.2 *SIMSAT* Context Diagram [8]

- vertical height of any assembly was limited to approximately 15 feet; it had to fit below the overhead laboratory lighting and ventillation ducts in Room 146
- the air-bearing model assembly (Figure 1.1) must be used (the only required design element)
- if used, any computer-based ground station should be accessible from the AFIT network to allow file transfer and use of available software (if possible)
- comply with appropriate AFIT, Wright-Patterson AFB, Air Force, and all other relevant agency policies (pertaining to safety, power, noise, pollution, radio frequency, and hazardous materials issues)
- provide a means to shutdown the system in case of emergency

Two final constraints that did not fit easily into this laundry list are the cost and schedule constraints. At this point in the design, specific cost allocations were not available, but a budget of approximately $100,000 was considered the high-end constraint (although

less expensive would obviously have been better). A design timeline was also formulated to provide a schedule at this stage. The significant events in the overall *SIMSAT* design timeline were:

Apr 98	Refine problem definition and attain initial needs and constraints.
Apr 98	Begin value system design and preliminary system synthesis.
May 98	Refine the value system and conduct further research.
Jun 98	Perform overall system evaluations.
Jun 98	Complete Concept Exploration phase.
Sep 98	Complete the Preliminary Design phase Order subsystem components which have long lead times.
Dec 98	Complete Detailed Design phase. Order remaining subsystem components.
Feb 99	Complete system integration; perform system test and evaluation.
Mar 99	Present final system design and associated documentation.

For the C&DH effort, the deadline for the final design was Sep 98. The associated subsystem testing also had to be completed by the end of Sep 98, with final design documentation and presentation in Nov 98.

Other than the budget constraints and acceptability criteria identified above, all elements of the subsystems and components of the overall design were considered alterable—the overall system architecture was left to the discretion of the design team. Obviously, the first three constraints were related, and if no acceptable solution could have been found to fit within those constraints, relocation of the test facility may have been considered.

2.2.2.6 Alterables. The team then considered what items they had control over in the system design. The items in this list directly influenced the system alternatives the team created in *System Synthesis*, Section 2.2.4.

1. power sources
2. control systems
3. structural support techniques
4. attitude determination systems
5. satellite movement techniques

6. communication systems

While not the original intent of this list, it is interesting to note that it mirrors the functional decomposition of the system shown in Figure 2.2.

2.2.2.7 Tradeables. The items in this list are the characteristics, or consequences, of design choices. The "best" systems solution provides the best balance of these characteristics, based upon the inputs of the CDM, without violating any of the above constraints.

1. system cost
2. development time
3. degree of safety enhancement
4. level of performance, considering
    - ease of use
    - robustness
    - simulation fidelity
    - command & control vs autonomy
    - data analysis capability
    - satellite movement

In Section 2.2.4, *System Synthesis*, the first six alterables were the design decisions to be used to define the potential system solutions. The rest of the alterables were the characteristics of those candidate solutions (i.e., the consequences of the design decisions) which factored into the values hierarchy defined in Section 2.2.3, *Value System Design.* The next section defines some additional elements of the problem.

2.2.2.8 Problem Elements. Once the needs and the system boundaries were defined, the next step in the design process was to identify specific elements of the overall problem which had to be addressed by the *SIMSAT* system architecture. These problem elements were determined to be:

- Moving the satellite.
- Powering the satellite.
- Communicating with the satellite.
- Commanding the satellite.
- Accommodating a variety of experiments.
- Collecting and analyzing satellite telemetry and experimental data.
- Representing the behavior of the satellite (how to depict the satellite on the user interface).
- Predicting satellite/payload behavior (how to estimate the satellite response to inputs).
- Providing emergency shutdown of the satellite.

To address these elements, the team utilized the expertise available on the team, developing additional expertise in those areas they were less comfortable in. The next section delineates the expertise the team considered appropriate to address these problem elements.

2.2.2.9 Relevant Disciplines. Based upon the elements of the problem the team had to address, they identified the following disciplines as relevant to solving this problem, incorporating knowledge from the disciplines as required:

- Systems engineering.
- Space operations.
- Astronautical engineering.
- Mechanical engineering.
- Electrical engineering.
- Program management.
- Simulation/control theory.

- Software design and integration.
- Telemetry and data acquisition system design.
- Computer architecture design.

These disciplines represent the team's implementation of the Knowledge axis of the three-dimensional systems engineering morphology developed by Hall [26:159], with the logic (systems engineering process) and time (lifecycle phases) comprising the other two axes (as adapted for this particular application in Figure 1.4).

2.2.2.10 Problem Definition Summary. By the time the team got to the end of the problem-solving process for this life-cycle phase, this step did not seem that important since it does not produce as many fancy charts, graphs, or other "output" as the other tasks. But its importance in the overall process can not be overlooked. Without this step, the problem might have been intuitively understood, but would not have been as well-defined causing the rest of the process to be much more difficult. The team would not have been able to build as effective a value hierarchy in Section 2.2.3 or develop alternatives as sound as those in Section 2.2.4. The rest of the design effort would have floundered as a result.

The next section documents how the team took the the fundamental needs of this section and elaborated them to one more level of detail to qualitatively compare system alternatives. That structured hierarchy provided the foundation for all the subsequent chapters and, eventually, led to a prioritized list of possible alternatives to meet the CDM's needs.

2.2.3 Value System Design. In this section, the focus is on how the team developed the framework for determining how well a given solution fulfills the needs, values, requirements, and objectives of the CDM. The framework chosen must consider what is important to the CDM (his or her values), and provide the way to systematically capture the salient details of the alternatives (those that are "tradable") to assess how well each alternative satisfies the CDM. That framework is typically shown as a hierarchy of values

or objectives [28:13–16]. While they are very similiar, some in the technical community consider the value hierarchy a better tool. The reasons for this preference include [32]:

- In an objectives hierarchy, measure direction specified by "Minimize" and "Maximize" can be confusing when they are mixed—the customer will have to "get used to the idea" that sometimes going up is good, and sometimes it is bad
- Those additional words also visually clutter the hierarchy
- Measure direction is typically implied in a values hierarchy anyway (i.e., no one typically wants to *Minimize Profit*, or *Maximize Waste*)

Since the hierarchy for the entire *SIMSAT* effort is fairly small, the C&DH effort will use a values hierarchy, while the rest of the team documented the users desires by means of an objectives hierarchy. The reader should consult the companion thesis [8] to compare the tools. To develop the framework shown in Figure 2.3, the team took the top-level needs/values discussed in the previous section and worked with the CDM to make them more specific.

At this stage of the design life-cycle, it was not considered productive to identify specific measurables within the value system to quantitatively compare alternative solutions. Because the classes of solutions could have been very different, the solutions might not compare directly using detailed measurables. In addition, the top-level nature of the solutions in this phase made direct assessment of quantitative measurables difficult, if not impossible. Instead, the value system at this stage only needed to provide a framework wherein each broad solution could be distinguished from the others. If a value system without specific measurables allows differentiation between solutions, then it serves the purpose of identifying the most promising class or classes of solutions [28:23]. If no significant differentiation could have been made between solution classes, all of them would have continued into the next design phase(s).

The values hierarchy in Figure 2.3 reflects the final form of the *Concept Exploration* hierarchy, updated from its initial form for shortcomings found while progressing through the problem-solving steps in this life-cycle phase. As can be seen by comparing it to Figure 4.5, this early life-cycle values hierarchy required revision in the next life-cycle

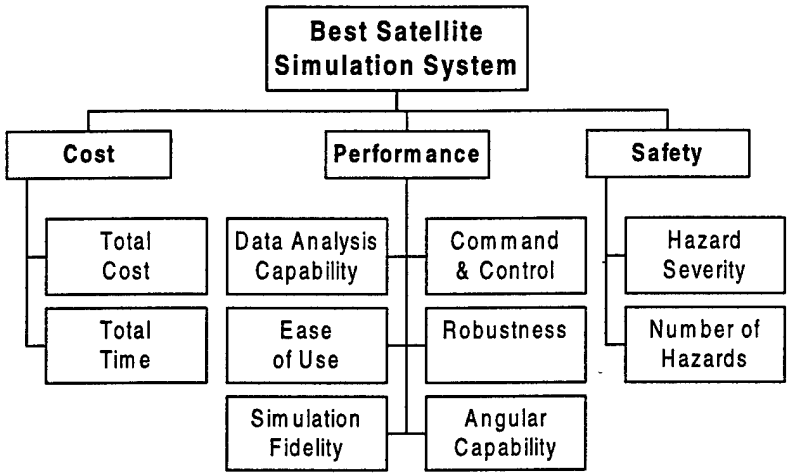


Figure 2.3 *Concept Exploration* Value Hierarchy

phase to more accurately reflect the distinguishing characteristics of more detailed system designs. In fact, each stage of the life-cycle required a new values hierarchy to reflect the level of detail required at that time to differentiate between solutions.

The customer's fundamental concerns (or "values") for the *SIMSAT* design were identified as "Cost" (i.e., impact to resources; the two most critical being money and time), Performance, and Safety. The team (in conjunction with the CDM) then derived some intermediate concerns, or "evaluation considerations"[1] to determine how well a given alternative satisfied those fundamental values. The following list describes the aspect of the alternative each evaluation consideration was trying to capture for each fundamental value at this phase of the design life-cycle.

- **COST**

  Total Cost Consider the purchase and integration costs to bring the system on-line.

[1] *Evaluation Considerations* are those intermediate values that link the fundamental values at the top of the values hierarchy with the quantitative measures at the bottom [28:12–13].

Total Time Consider the purchase and integration time requirements (total time-frame) to bring the system on-line.

- **PERFORMANCE**

  Data Capability Consider both what can be displayed real-time and the post-experiment data analysis capability.

  Ease of Use Consider the user interface and how easy it is to switch components/experiments on the model.

  Simulation Fidelity Consider how well a computer simulation model could represent the physical models behavior and how easy it is to develop the simulation model.

  Command and Control Consider how well the system does what is desired, how responsive it is, and how autonomous it can become.

  Robustness Consider the range of experiments the system can support.

  Angular Capability Consider the system's ability to move rapidly (high slew rate) to the desired position (high sensing accuracy) and stabilize.

- **SAFETY**

  Hazard Severity Estimate the system hazard severity in terms of potential damage to equipment and/or injury to personnel.

  Number of Hazards Estimate the total number of hazards inherent to the system. (a measure to represent the likelihood of failure).

Figure 2.3 captures the *Concept Exploration* hierachy of fundamental values and evaluation considerations. This hierarchy was used in follow-on problem-solving steps to qualitatively measure how well each of the potential classes of solutions measured up against the CDM's values.

2.2.4 System Synthesis. The goal of this task was to develop alternative solutions that, more or less, meet the needs of the CDM. The primary goal, within this life-cycle phase, was to generate as many potential technology solutions as time permitted.

These alternatives were then passed through an initial sanity filter: each candidate system solution must be technologically feasible within the planned time-frame, must be compatible across all the sub-systems, and the solution *Alterables* must be selected to meet the *Constraints* and optimize the *Tradeables* spelled out in the *Problem Definition* task above. Only those solutions that did not make it through this filter were thrown out. In fact, we used this technique to quickly trim the Strategy Generation Table (SGT) permutations (discussed in Section 1.5) from over 960 (represented by the $5 \times 3 \times 4 \times 4 \times 4$ options in Table 2.1) to about 1 technology class for each subsystem using the logic described in the next section.

Using the problem issues discussed in Sections 2.2.2.8 (*Problem Elements*), 2.2.2.5 (*Constraints*), and 2.2.2.6 (*Alterables*), an initial system architecture was developed to meet the needs of the user and provide a basis for *System Synthesis.* The initial system architecture also provided identification of the system boundaries, and by extension, the environment. This resulted in the system context diagram shown in Figure 2.2 (page 2-7).

2.2.4.1 System Decomposition. Initial system decomposition could have taken many forms. The subsystems identified in Figure 2.2 resulted from analysis of the problem elements, and a functional approach to addressing these elements following the method outlined in [35] since it lent itself to assigning indivduals to become subject matter experts on a given function/subsystem.

A software/hardware breakdown was rejected due to the strong interdependencies of computer software and hardware solutions. Furthermore, this breakdown did not aid in development of solution alternatives since the specific problem elements are not addressed using this format. Similarly, a ground station/satellite model breakdown was rejected since aspects of some subsystems resided within each physical divisions, creating an artificial and unnecessary barrier between them.

As stated above, the system decomposition of Figure 2.2 was based on functional breakdown, and was modeled after the spacecraft subsystem breakdown used in the Space Mission Analysis and Design (SMAD) text by Larsen and Wertz. The "Guidance, Navigation, and Control" subsystem identified in that text [35:287] was ignored since the sim-

ulator model is fixed in the laboratory. Thus, guidance and navigation represent attitude positioning only, which is handled by the "Attitude Determination and Control System" (ADACS). The thermal subsystem identified by SMAD was eliminated from initial consideration as a separate subsystem because operating conditions were not expected to require dedicated thermal management. If system operation required cooling of components, later stages of development could address environmental control mechanisms. The other subsystem divisions are very similar to the SMAD text definitions. The following subsystem list defines the system decomposition used for the *SIMSAT* design effort:

Attitude Determination and Control System (ADACS). The ADACS consists of three components: attitude determination equipment, attitude control equipment, and control software that will execute on the processor(s) contained in the C&DH system. The attitude determination equipment determines the system's actual position/orientation and provides information used to develop inputs for the attitude control mechanisms. The attitude control equipment provides the necessary forces and torques on the system based on information from the user or attitude determination and control software inputs. The control software element consists of software code incorporating control laws which provide the logic used to provide the proper inputs to the attitude control equipment. The ADACS subsystem addresses the following problem elements identified in Section 2.2.2.8: "moving the satellite", "predicting satellite behavior", "and representing the behavior of the satellite".

Power System. The power subsystem includes elements associated with the supply, regulation, and distribution of necessary voltages and currents to operate all the onboard subsystems. This subsystem addresses the problem element "powering the satellite".

Command and Data Handling System (C&DH). The C&DH subsystem receives, decodes, processes, and distributes all satellite commands. Moreover, it gathers, formats, stores, and transmits telemetry data from the onboard systems and experiments, as well as the ground station. This definition of the C&DH system is taken from the SMAD text [35:288]. Within the C&DH is the inherent computer archi-

tecture to perform these tasks, addressing the problem elements "commanding the satellite" and "collecting and analyzing satellite telemetry and experimental data".

Communications. The communications subsystem represents the interface between the satellite model and the ground station. The communications system is, in effect, an extension of the C&DH subsystem, linking the simulator C&DH elements with the ground station C&DH elements. This subsystem therefore performs the "communicating with the satellite" function listed in the problem elements. Thus, the communications and C&DH subsystems are shown slightly overlapped in Figure 2.2 (page 2-7).

Structures and Interfaces. This subsystem represents the physical satellite assembly, to include subsystem housing, structural supports, fasteners, and physical interfaces between the base model and experimental hardware. Although each subsystem must consider the following problem element, this subsystem addresses "accomodating a variety of experiments" most directly.

With the broad system architecture now conceived, the next task was the identification of feasible system solutions through the development of a list of potential subsystem technologies, which could be integrated to create feasible system alternatives (using the SGT). These subsystem alternatives were the result of creative brainstorming and research into actual satellite systems, laboratory-based systems, flight test systems, and emerging technologies. Thus, a certain amount of knowledge had to first be developed to generate solutions. While each of the subsystem alternatives are described in the companion thesis [8], the list of subsystem alternatives is summarized in Table 2.1 to provide a context of the system level alternatives the C&DH subsystem was a part of. The focus of this document will be on the C&DH subsystem, its alternatives, and the rationale used to select a class of technologies best suited to the *SIMSAT* application.

2.2.4.2 Classes of Technologies. The C&DH subsystem has two main purposes: interface with the user and, as required, execute the actual control laws determining how the effectors (devices that cause something to happen) are activated to

Subsystem	Alternatives
Attitude Determination	IMUs Sun/Star/Horizon Sensors Magnetometers Radio Interferometers Laser Grid
Attitude Control	Momentum Exchange Mass Expulsion External Forces
Power Generation & Distribution	Photovoltaic Cells Fuel Cells Thermal Batteries Chemical Batteries
Command & Data Handling	Direct Control Analog Computer Digital Computer (Text) Digital Computer (Graphical)
Communications	Flight Test Telemetry Satellite Relays Wireless Modem Wireless LAN
Structures	As Required

Table 2.1 Initial Subsystem Technology Alternatives

respond to commands. Theoretically, the control system could be as simple as a direct connection between the user controls and the effectors (similar to the mechanical control systems of early aircraft). But because this approach creates a tremendous workload for the operator, engineers have found ways to use computers to help the user control the system, improve performance, and reduce life-cycle costs at the same time.

To determine the best C&DH design concept, some point solutions in the spectrum from simple, direct control systems, to the most sophisticated computer control systems available today, were considered. All these C&DH alternatives assume the other *SIMSAT* subsystems (power, communications, and structures) will be sufficiently designed to allow for full C&DH capability. The C&DH subsystem technology alternatives included the following classes:

Direct Control. The direct, unassisted control option considered in this review consisted of a user interface could be comprised of knobs, levers, switches, dials, display panels, and/or lights. The user commands (from the knobs, levers, and/or switches) would be converted to a format that could be transmitted to the satellite, where they would be converted back to a format required by the effectors (those parts of the satellite that cause something to happen). The results of those commands would be sensed, converted to the transmission format, sent back to the ground station where it would be converted to a format consistent with the dials, display panels, and/or lights. This functionality would require substantial research to locate compatible components, integrate them, and then troubleshoot the system when problems arise. These "components" would likely have been at the piece-part level—very little pre-implementation is likely (few implementations like this are being developed any more).

Analog Control. The analog computer-assisted control system consists of a user interface very similar to the "direct" system above. Control law implementation/adaptation is typically handled by changing discrete components. Before the recent advancements in digital computer hardware and software, these computers were considered the preferred solution for real-time control due to reduced execution delay (no software "overhead"). All control box designs provide some flexibility for design optimizations to allow for more "building block"-type implementations (reducing the likelihood of component incompatibilities). This will reduce the research, implementation and troubleshooting time. However, analog computer technology is not as well understood or supportable as digital technology.

Digital Control (Text). The digital computer-assisted control system is based on a command-line (or "text") user interface. Control law implementation and modification is handled with well-defined software packages available for a number of different hardware platforms. Integrated subsystem implementation similar to the analog computer approach, but with more "building block" components available, along with significant in-house expertise.

Digital Control (Graphical). This solution class consists of a digital computer-assisted control system, with a visual/graphically-based user interface. All the design advantages of the above classes of technologies existed with the added intuitiveness of the graphical development environment. Using the integrated hardware/software solution provided by dSPACE, Inc. reduces the "untested" parts required for implementation and troubleshooting to a minimum. A significant portion of the system was already available at AFIT, so the total cost of this system will be fairly low. There was also a lot of in-house and proximate technical support to address any problems that may develop.

2.3 Analysis

This section details those parts of the Hall process steps *System Analysis* and *Optimization* appropriate to the problem at hand. At this stage of the SE Process, these steps were very qualitative—no measures will be defined, but each alternative will be assessed on a relative scale for each value shown in Figure 2.3. Additional detail will be added to this step in follow-on phases.

2.3.1 Analysis Methodology. A primary task of the *Concept Exploration* phase was to eliminate those subsystem alternatives which were determined to be infeasible, impractical, or relatively inferior to other alternatives. "Mental modeling" was the primary method for analyzing subsystem alternatives during this phase. Mental modeling relies on expert opinion, personal experience, research, and common sense rather than sophisticated models or rigorous mathematical analysis. Mental modeling was an efficient tool for making a top level "first cut" between feasible and infeasible alternatives. In order to augment this mental modeling, the SMAD text was often used as a source for expert opinion or rudimentary math calculations. The following paragraphs document the mental modeling used to determine impractical solutions, and allow selection of a set of solution classes for the *Preliminary Design* phase.

While the data was collected for each subsystem independently to support a system-level assessment of integrated solutions, that top-level evaluation proved unnecessary—each

of the subsystems either ended up with a clearly preferred (or *dominated*) solution[2], only one practical/feasible solution, or no decision could be made during this life-cycle phase.

As mentioned before, the ADACS, power, and communications subsystems all had to respond to the choice of C&DH architecture. Thus, an early choice for the C&DH architecture was critical to the overall system architecture. Rather than repeating all the details justifying the choices made for each of the other subsystems, only the detailed analysis for the C&DH subsystem is documented here. To review the complete justification for the non-C&DH subsystem selections made below, consult [8].

2.3.2 Modeling. Since each of the classes of C&DH technologies listed in Section 2.2.4.2 could result in a myriad of alternative solutions, the team needed to qualitatively compare the categories to find the solutions to be investigated further, discontinuing the investigation of the solutions that provide poor value for the required investment. To narrow the focus of the next phase of the team's process, ideally one class of subsystem solutions would dominate the others (i.e., is at least as good in all areas as the rest of the solutions and better than the rest in at least one area). And if there was not a class of solutions that stood out from the rest, hopefully at least one of them would be dominated (eliminating it from consideration). But, how to decide the goodness of the alternatives? The values hierarchy was used to make that assessment.

The value hierarchy in Figure 2.3 represents our first step in trying to conceptualize the CDM's values in determining the best *SIMSAT* design. This hierarchy only includes the first level of evaluation considerations to determine how well the ultimate design goal was met. At this level of design abstraction, there was no point in trying to go all the way down to determining the salient characteristics of the actual alternatives—they have not even been defined yet so it would be impossible to establish actual measures of merit.

Figure 2.4 documents the qualitative assessment of the ability of each class of solutions to satisfy the *System Definition* value hierarchy on a scale 0 (worst) to 5 (best). This evaluation table was used to help the team focus on the technologies that appeared to have

[2]One solution in each area was at least as good as the rest, qualitatively, in all the evaluation considerations shown in 2.2.3, and the best in at least one of them.

Alternative	Total Cost	Total Time	Num. of Hazards	Haz. Severity
Unassisted, direct control	5	3	4	4
Analog computer assisted	1	1	4	4
Digital computer assisted (text)	3	3	5	5
Digital computer assisted (graphical)	5	5	5	5

Alternative	Data Capability	Ease of Use	Simulation Fidelity
Unassisted, direct control	2	1	0
Analog computer assisted	2	2	1
Digital computer assisted (text)	3	4	3
Digital computer assisted (graphical)	5	5	5

Alternative	Command & Ctrl	Robustness	Angular Capability
Unassisted, direct control	1	1	4
Analog computer assisted	2	2	4
Digital computer assisted (text)	3	3	4
Digital computer assisted (graphical)	5	5	4

Figure 2.4 C&DH *Concept Exploration* Evaluation Matrix

the most promise. The data in the table represents the assumption that the other *SIMSAT* subsystems are "the best"—the "optimal system" would score '5' across the board. The scores shown represent how the total system would score by altering the C&DH class of solutions alone—the rest of the subsystem alternatives were assumed to remain constant. High scores for a given evaluation consideration are highlighted.

2.3.3 Analysis. The following descriptions explain why the C&DH alternatives scored the way they did:

Total Cost. While the direct solution would have used simple components, the sunk costs of the DSPACE system means the visual, digital computer solution can directly

compete with the direct solution. The analog computer solution is expected to be the most expensive due to the specialized nature of the technology. The digital (text) solution is moderate because the equivalent of the dSPACE hardware and software would have to be purchased, but are very common COTS components.

Total Time. The dSPACE solution was the best because the parts that may still be required will take little time to come in and integrate into the system—the dSPACE system is assumed to be integrated before the rest of the system is ready since most of the parts are already available. The analog solution is expected to take about as long as the digital (text) solution because both would require piece part selections, orders, and integration troubleshooting. The analog computer would take the longest because they are so rare, the order time would be considerable, and the integration time would likely be the longest—limited in-house expertise available for troubleshooting.

Number of Hazards. The digital computer solutions were ranked the highest because the assumption is that much of the implementation would be based upon "building block" solutions—there were numerous sources for all the components of the subsystem. Such is not the case for the other technologies: the direct system would be nearly entirely fabricated at AFIT and would not be as "tight" as mass produced commercial component solutions. The analog computer solution would have more commercial "building blocks" in it, but would require more fabricated portions than the digital systems.

Hazard Severity. This follows the same logic as the number of hazards—the more commercial building blocks used, the fewer things can fail and the less severe the potential system failure.

Data Capability. The digital solutions assume the data can be viewed real-time and collected directly on the computer hard drive or use typical lab quality data acquisition systems. The other two alternatives assume a totally independent data logging system that will likely not be able to view any "useful" data during the experiment; only oscilloscope traces or strip charts would be available real-time.

Ease of Use. The digital, graphical environment is always the most intuitive for development and control. And the digital computers are the most powerful due to the tools available to it. Again, our lack of expertise in analog computers would make system or payload changes a significant challenge; the possible user interfaces is likely to be similar to what can be used with the digital (text) class of alternatives. The interface and adaptation of the direct control would be the worst—the user would have to re-learn how to handle the system every time some change is made.

Simulation Fidelity. Obviously the direct control system has no simulation capability in and of itself. How an analog computer assisted system could be simulated (or how good the simulation would be) is unclear, but due to the lack of internal expertise, the difficulty of developing such a capability would be great. Digital simulations are well understood and supported with numerous software tools. Of course, a visual environment would be the easiest to build the model and since dSPACE is an integrated solution, it would provide the highest fidelity model.

Command & Control. Any open-loop control system (i.e., the direct control system) is very difficult to deal with, and its responsiveness will be user-dependent. In addition, any computer enhancement will increase the likelihood the system will do what you want it to, no matter how it is configured. The digital control systems will be even more that *SIMSAT* will do what you want it to (and convert to an autonomous system) due to the maturity and availability of development tools and supporting hardware. The dSPACE solution will be the best in all these areas due to the integrated nature of the hardware and software.

Robustness. This category is very correlated to the scores obtained by the Command and Control measure, but represents a different aspect of the system. Actual implementations (as opposed to classes of technologies) may have some divergence with the previous measure, but at this level of detail, the alternative values appear to be the "same" as the command and control values.

Angular Capability. While each technology scored the same, the reasons are different. The direct solution suffers from the requirement of the operator to know how to operate the controls to cause *SIMSAT* to move "fast" and stop "fast", then stay "on

target." The computer solutions all suffer from the added weight/moment of inertia impact the torque system must overcome—the assumption is that, once the control program is optimized, the actual response of the controller is the same.

2.4 *Interpretation*

This section (the equivalent of the Hall *Decision-Making* and *Plan for Next Phase* process steps) provide justification for the choices made during this life-cycle phase. At this stage of the SE Process, *Decision-Making* was very qualitative—only alternatives that dominated others were selected for further evaluation.

2.4.1 Interpretation Methodology. The *Concept Exploration* phase made several significant steps toward the design of an up-and-running satellite simulation system. The problem was refined, with an initial list of needs and top-level requirements generated. A top-level value system, complete with a values hierarchy, was developed. The overall system architecture was framed, with potential subsystem alternatives considered and explored. Finally, these alternatives were compared and analyzed, with decisions made for each subsystem before progressing into the next design phase.

2.4.2 Decision Summary. Upon completion of *Concept Exploration*, some of our subsystems had clear choices, others did not. In general, only the C&DH setup significantly affected the overall system architecture once the infeasible and impractical subsystem alternatives were eliminated. Table 2.2 summarizes the results of the Concept Exploration and definition phase, and sets the stage for the beginning of Preliminary Design. As previously stated, the choice of a C&DH architecture was necessary for the system design to advance to the *Preliminary Design* phase.

2.4.2.1 Attitude Determination and Control System. For the ADACS alternatives, the choice of IMUs for attitude determination was made. Additionally, momentum exchange methods were chosen for further design of the attitude control system. These ADACS alternatives are proven methods, and have fewer associated

Subsystem	Technology Choice
ADACS	IMUs/Momentum Exchange
Power	Chemical Batteries
C&DH	Digital Computer (Graphical)
Communications	Wireless LAN/modem
Structures	As Required

Table 2.2 Concept Exploration—Subsystem Technology Choices

risks in terms of unforeseen problems, such as failure to meet requirements or difficulty in fabrication and integration. The use of gas jets (or other mass expulsion methods) was retained as a possible means for slewing/braking augmentation. However, gas jets alone were determined to be an impractical ADACS solution.

2.4.2.2 Power. The choice of chemical batteries for the power subsystem was also grounded in precedent. Through a range of available types and sizes, chemical batteries allow flexibility in meeting power requirements, quicker integration schedules, and confidence in proper operation.

2.4.2.3 Command and Data Handling. From Figure 2.4, and the above explanations, it is clear that the last alternative is the dominant solution—it has the highest score in each category, and in some categories, it is the only solution with the highest score. As the analysis showed, the dominant solution class for the C&DH was the digital computer with graphical interface. This solution was made even more favorable by the sunk costs of the DSPACE system purchase (assuming that any alternative that might be found more favorable during the next life-cycle phase will have to be even better in other areas to make up for the cost penalties).

2.4.2.4 Communications. The use of wireless LAN/modem architecture was selected based on the use of a digital (graphical) computer C&DH subsystem. This C&DH architecture, namely the probable use of DSPACE, required the use of a wireless network, and allowed for possible onboard computer processing.

2.4.2.5 Structures. No refinement of the structural subsystem was made at this level of design since the "alternatives" available for this subsystem differ depending upon the final decisions made for the other subsystems.

2.5 Summary

The *Concept Exploration* phase made several significant steps towards the design of an up-and-running satellite simulation system. The problem was refined, with an initial list of needs and top-level requirements generated. A top-level value system, complete with objective hierarchy, was developed. The overall system architecture was framed, with potential subsystem alternatives considered and explored. Finally, these alternatives were compared and analyzed, with decisions made for each subsystem before progressing into the *Preliminary Design* phase.

Future chapters document the results of taking these set of technology choices to the next level of detail.

III. Preliminary Design

3.1 Overview

In this chapter, the focus of the design begins to diverge for strictly a *SIMSAT* system focus to more of a C&DH subsystem focus. This *Preliminary Design* iteration of the problem-solving process for the C&DH subsystem developed a trade study to evaluate candidate vendors of the digital control systems identified in the previous life-cycle phase. This does not mean the SE process changed to evaluating C&DH subsystem alternatives in isolation, or strictly on their own merits as C&DH subsystem solutions. Rather, the evaluation process made assumptions regarding the baseline system, where the only changes to the system was to select different C&DH alternatives.

The first section in this chapter will justify the rationale for accelerating the development of the C&DH subsystem, and how the team reduced the likelihood of "sub-optimization" issues associated with developing a subsystem somewhat independently. The next section provides some background on Real-Time issues that needed to be addressed before design decisions regarding real-time systems were made. Particular issues covered in that section include the difference between real-time and general computing systems, techniques that can be used to address those differences, followed by the analytical tools provided by one of those techniques. The intent of that section was to provide justification for some of the measurables required to compare how well potential alternatives can support *SIMSAT* real-time concerns, providing a foundation for the discussions of the real-time measurables used in the rest of this document.

Finally, before making any design decisions, some issues covered in *Conceptual Design* needed further definition to conduct the trade study, while others were left for future steps. First, additional details need to be developed for the *Issue Formulation* step of the *Preliminary Design* phase (see the activity matrix in Figure 3.1). As a minimum, the team needed to refine the *Problem Definition* task based upon some of the details developed in the previous iteration, as well as those required to reflect the shift from requirements to design. In addition, the *Value System Design* task results had to be updated to provide a more

detailed framework to support the choice of digital control system vendors. From there, a pair of alternatives were found to compare to the baseline DSPACE system: one less integrated but perhaps more flexible, the other more robust, but more complicated. Each alternative was then scored against the revised values hierarchy and a final recommendation developed.

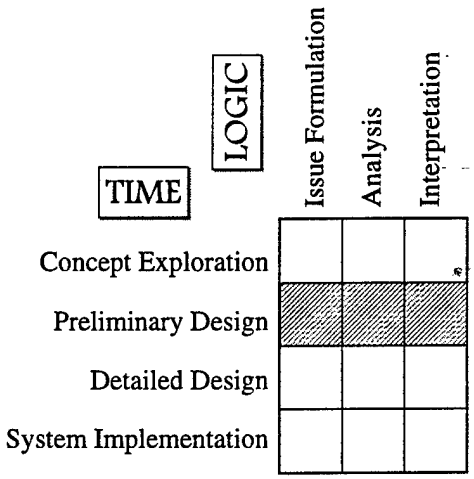


Figure 3.1 Preliminary Design Activity Matrix

As already mentioned, before getting into the trade study, an explanation of why the C&DH subsystem was selected as the subsystem to begin the accelerated development effort detailed in this document is needed. The next section will describe that decision, and how potential system-level pitfalls of seeking subsystem solutions were overcome.

3.2 Design Effort Division

While there were other reasons that necessitated a division in research effort (the author of this document needed to graduate before the rest of the team), there was sound technical justification for making the C&DH the first subsystem to be designed. Considering the subsystem decomposition, and the functions of each of those subsystems, it can be easily seen that there are several subsystem interactions that must be considered:

- Structures supports all the subsystems

- Power supports ADACS, C&DH, and Communications
- C&DH and ADACS, together, form the "brains" and the "brawn" of *SIMSAT*

From this it was clear that the C&DH and ADACS subsystem will effectively influence the other subsystems, and may even constrain them. That leads to the natural conclusion that perhaps the system design could be done one subsystem at a time. One significant problem with that approach is that "optimal" subsystem solutions may lead to sub-optimal system solutions—system level interactions between subsystems cannot be ignored. Another problem with sequential subsystem designs is the additional time required to complete the total system design compared to developing the entire system concurrently.

The team found a way to ameliorate both problems. During each stage of the design effort, each subsystem was assessed to determine if there were potential design decisions to be made. Anytime decisions could be made they were. As the team progressed through the design effort, it became clear the design of some subsystems (such as Structures) would need to wait until the other subsystems were defined (confirming the first observation made in the above list). In fact, over time, it became clear that portions of the core subsystems (C&DH and ADACS) would need to be defined before the design of the remaining subsystems could progress. Since the team would likely have to build the Momentum Wheels for *SIMSAT*, and the C&DH subsystem would need to be an integrated COTS system to meet the usability goals for the system (so the solution space was likely to be much smaller), the decision was made to have the C&DH subsystem lead the rest of the system design. The initial purchase of portions of a DSPACE system, coupled with the author's existing knowledge of embedded, real-time control systems, also influenced the selection of the C&DH subsystem as the *SIMSAT* design pioneer.

To minimize the potential for sub-optimal subsystem solutions, the team made subsystem decisions (for every subsystem) based upon the system-level values hierarchy for that life-cycle phase. When considering alternatives against that hierarchy, the impact of the alternative on other subsystem was factored in. For instance, if a C&DH solution could only be run with AC power, the various impacts of adding a power inverter to that subsystem to make it work on the satellite would have to be factored in. In additon,

weekly internal design reviews provided a forum for the various subject matter experts to interact and reduce the likelihood of unintended consequences for decisions they were trying to make. While these were not flawless techniques for assessing system interactions when considering subsystem design alternatives, it was better than developing subsystems completely in isolation.

With the decision to have the C&DH subsystem be the first subsystem to complete its design effort, research was required to determine those aspects of real-time control systems that need to be compared between potential solutions.

3.3 Computer Subsystem Concerns

As mentioned above, this research effort departed from the rest of the *SIMSAT* design effort after the *Concept Exploration* phase. This effort considered the design, implementation, and testing of the computer control system to support the follow-on *SIMSAT* design and integration effort. Before going farther in discussing the results of the team's efforts, some specific computer control technology issues and concerns need to be addressed.

3.3.1 Real-Time Control Systems. Living in the computer age may lead many to think they can evaluate control computer system correctness (as one of the measures of "goodness") using the same approach they take for determining if an office computer works correctly: "does it produce the logically correct answer?" But real-time applications need the same logically correct results prior to a specific point in time (called its *deadline*) to be "temporally correct." For real-time systems, then, a logically correct answer not available before its time-critical deadline is reached will likely be as useless as an incorrect computation.

Computerized control systems (among others) fall into two categories of "real-time" systems: hard and soft. "Hard" real-time control systems require accurate results from all necessary computations before some well-defined deadline, **guaranteed**. For example, if the tasks that define the behavior of a computerized flight control system miss deadlines, the aircraft could become uncontrollable. For "hard" real-time control systems, accurate results **must** be computed before some pre-defined, application-specific deadline is reached.

For this type of system, a logically correct answer is useless if it is produced too late for the system to respond to user inputs or maintain system stability.

A control system with less stringent timing requirements (such as an environmental control system [ECS]) requires computational results within a "reasonable" period of time (a flexible, or "soft," deadline). If a non-critical ECS task set misses a deadline for optimal temperature control, the pilot may get a little uncomfortable.

The *SIMSAT* control system effectively falls into the first category. We are concerned about supporting a subset of hard real-time tasks: critical control system tasks that must provide logically correct results before some **required** deadline. Understanding more about "hard" real-time systems and scheduling theory is critical to understanding how to determine if a given control system can meet the user's requirements. The following section describe some key *Scheduling Theory* issues used in this evaluation. This is the first chapter that requires some understanding of the attributes of Real-Time Control Systems, so it is treated as a section of background material. Appendix A provides some additional information regarding several other Real-Time Control issues, some of which do not have immediate bearing on this effort, but provide significant foundation for some of the projects suggested in Chapter VI.

3.3.2 Scheduling Theory.

Research into task set schedulability, or "scheduling theory," has attempted to determine the most effective way to schedule a set of tasks and accurately predict whether deadlines can be met. In other words, scheduling theories intend to provide techniques to aid in determining the "schedulability"[1] of a given task set. Locke, in [43], showed a number of different techniques for scheduling a set of tasks. His conclusion was that the only viable theories for most critical real-time systems are the *Cyclic Executive* and *Fixed Priority* techniques. But, as he indicated [43:52], *Fixed Priority* techniques were not even viable until Liu and Layland published their classic paper detailing a fixed priority scheduling technique called *Rate Monotonic Scheduling* (RMS) [39]. Since the time of that seminal paper, many extensions have been developed to make *RMS* a practical tool. In fact, the RMS techniques are some of the simplest tools

[1] A task set is 'schedulable' if the deadlines of the task set can be met.

for determining schedulability, and they can be used to determine "optimal schedulability" of a task set (no other fixed priority scheduling technique can beat RMS processor utilization). This rest of this section is devoted to further elaboration of these issues.

3.3.2.1 Current Knowledge. The Liu and Layland paper intended to provide a set of formal tools to help determine if a set of tasks could meet all its required deadlines. They proved the RMS algorithm was an "optimal" fixed priority scheduling algorithm: if any fixed/static priority scheduling algorithm could schedule a task set, the RMS algorithm could also determine a way to successfully schedule that task set. Using C_i as the execution time of task τ_i, and T_i as the period of the same task, Liu and Layland defined the amount of work done by (or "utilization of") a processor for task τ_i as $U_i = \frac{C_i}{T_i}$. Liu and Layland then showed that, if the sum of all task U_i's is less than 0.6931 (no matter how many tasks there are), the task set can meet all its deadlines.

While this technique for determining task set schedulability is extremely simple and straight-forward, the 69.31% limit is too conservative for most applications. This low processor utilization limit, combined with the restrictive assumption that the task set under evaluation consists strictly of independent, periodic tasks, rendered their basic model nearly useless. The process Liu and Layland went through was noteworthy, but these shortcomings prevented widespread use of RMS in its original form.

Fortunately, some in the real-time community saw the potential power of the basic RMS technique. Numerous papers have been published since the mid-1980's to alleviate some of the limitations of the basic RMS theory. For example, [37] relaxes the Liu and Layland "worst case" 69.31% processor utilization limit. The authors demonstrated that a more realistic limit for a typical real-time workload is closer to 88% processor utilization. Their research indicated that this higher utilization bound is due to a general predominance of harmonic task periods (the task periods are integer multiples of the shortest task period in a given task set) within an "average" task set. One limitation to universal application of this increased bound is the task set size is assumed to be quite large, with a large spread in task periods [37:171]. Many other RMS shortcomings have also been addressed. In [52],

Sha, et. al. provide a good summary of the extensions and tools used to make RMS a more complete environment for real-time system design, which includes:

- Relaxing the Utilization Bound [37]
- Support for Aperiodic Tasks [58]
- Support for Critical Sections/Mutual Exclusion (avoiding Mutual Deadlock and Priority Inversion [25, 42, 40])
- Handling Various Input-Output Paradigms [31]

As already mentioned, RMS is a specific scheduling technique. *Rate Monotonic Analysis* (RMA), on the other hand, consists of a set of tools that can be applied to *any* task set, no matter what preemptive, fixed priority scheduling technique or implementation language is used [50:4,8]. RMA, as a more general analytical set of techniques, intends to provide insight into the temporal behavior of a set of tasks [50:9,12]. To accomplish this evaluation, RMA uses the analytical support tools developed to support RMS and the above extensions. If a set of tasks does not follow RMS priority assignments, the RMA tools can only tell the temporal characteristics of the *optimal* schedule but cannot guarantee the task set will behave that way [46]. The analytical tools (RMA) developed to support and extend RMS can be used to determine if it is possible for a set of fixed priority, preemptively scheduled tasks to meet its critical deadlines, but not if a given non-RMS implementation will be successful.

To further distinguish the analysis technique, RMA, from the application of RMA tools and extensions to a task set whose priorities were assigned according to RMS guidelines, the literature uses the term Generalized Rate Monotonic Scheduling ($GRMS$) for that latter case [55]. The $GRMS$ acronym provides important distinctions from both RMA and the basic, restrictive techniques embodied in the RMS designation. $GRMS$ is the term used in this document to mean RMA techniques applied to a set of RMS-scheduled tasks (i.e., task execution order defined by task period-based priority assignments).

3.3.3 Real-Time Concerns Summary. When control systems were simple, the application of an informal, ad-hoc, "code-and-test" approach to validating control

system design (including logical and temporal correctness) may have been acceptable. But with the advent of integrated and complicated computerized control systems, a disciplined engineering approach to validating system design is necessary (as early in the development life-cycle as possible). Validation must go beyond logical accuracy, to include system reliability and timeliness of response. Separating logical and temporal concerns help bring a more formal, disciplined approach to the development of "hard real-time" computer systems [41:184]. The *GRMS* techniques allow for such a separation of concerns. *GRMS* also supplies simple tools to determine, in a formal, well-defined way, whether a given set of tasks can meet its critical deadlines.

Rather than dealing with the schedulability issue in software, why not just use faster hardware—it would be less of a cultural shift. For real-time systems, *speed* of response is important only so far as it helps ensure tight deadlines are met. An improvement in "computing power" (speed of the calculations) does not necessarily imply a better control system [59:11]. Predictably quick response to unexpected events, a high level of "schedulable utilization,"[2] and stability during overload (critical tasks must always meet their deadlines, even during times of transient overloading) are the most important issues for real-time systems [55:68]. If the result is perfect, but it does not arrive when it is supposed to (i.e., misses a critical deadline), the controllability of the system suffers. The "schedulability" of a task set, then, is concerned with the timeliness and predictability of its set of tasks—can critical tasks meet their deadlines? And if not, can it be predicted which tasks will miss their deadlines? What are the results of missing those deadlines?

But how can one determine if a set of tasks can meet its deadlines? *Testing* for temporal correctness cannot provide 100% task set coverage (i.e., one cannot feasibly test every possible contingency) to ensure the design always meets its critical deadlines [59:11]. Therefore, some other technique for determining temporal correctness must be found to determine the adequacy of a real-time systems design. Scheduling theory seeks to find disciplined, structured alternatives for designing temporal correctness into a system, rather than "testing it in," and analyzing the timing characteristics of an implemented set of tasks.

[2]The processor utilization above which task set schedulability can no longer be guaranteed. Also known as the point of *Breakdown Utilization* [37:171].

3.3.4 Computer Concern Conclusions.

Many papers have been published to extend RMS. The basis for this interest in RMS is its simplicity and power. As already mentioned, since RMS is an "optimal" algorithm for ensuring a given task set can meet its deadlines, it can be used instead of any other preemptive, fixed priority scheduling technique. Also, RMS makes it easy for developers to assign priorities (semantic importance has no role in priority assignment) and determine if a given task set can meet it deadlines (using simple equations related to utilization bound or "scheduling point" calculations). Certain programming restrictions are imposed to implement a set of tasks within true *GRMS* guidelines, but these restrictions do not hinder good software development practices [52:19–24]. As long as the formal, yet simple, rules are followed, *GRMS* has proven to be a well-rounded scheduling technique (perhaps the *only* one [41:182]).

This issue is critical—the team needed to be able to determine the "correctness" of a given control system implementation. Whatever alternatives were considered needed to support schedulability analysis to one extent or another. Obviously, the more support an alternative had for a well-defined analysis technique, the better. As the *RMA* techniques have some of the most well-defined tools, and have a large body of literature supporting the techniques, it was the technique of choice.

3.4 Subsystem Trade Study

Since determining the "best" embedded control system for *SIMSAT* was simply another problem to be solved, the same steps established for the overall research effort was used to solve this problem. A trade study of potential solutions, both vendor-integrated and independent, team-integrated components were considered during this *Preliminary Design* life-cycle phase.

3.4.1 Issue Formulation.

Moving from *Concept Exploration* to *Preliminary Design* requires a natural shift in emphasis from determination of customer needs to considering potential clases of solutions. As a result, some tasks of the *Issue Formulation* step from the previous phase—*Problem Statement*, *Constraints*, and *Alterables*, and the alternatives being considered—had to be revisited.

*3.4.1.1 **Problem Statement.*** The problem statement was again modified to reflect the emphasis for this life-cycle phase:

> *Considering* SIMSAT *is a satellite system simulator to be used as an experimental test bed for AFIT, define those vendors that can meet the original design requirements and select the best C&DH subsystem solution to support Air Force/DoD research and provide a sound instructional aid to AFIT instructors teaching in satellite control and dynamics.*

This statement also formalizes the change from a system-wide definition to a predominant focus on the design of the C&DH subsystem. The other portions of the system went through their *Preliminary Design* phase as well, but not until the C&DH subsystem was in *Detailed Design* (see Section 4.2, page 4-2). For this reason, this phase will be documented as a C&DH exclusive phase.

*3.4.1.2 **Additional Constraints and Alterables.*** The Problem Statement above alluded to the only alternatives to be evaluated during this life-cycle phase: C&DH subsystems. As mentioned at the beginning of the chapter, for a C&DH subsystem trade study to be conducted, nominal values for the rest of the system had to be assumed to allow for meaningful comparison of potential C&DH alternatives. While this would also have required a sensitivity analysis similiar to what will be done in the next life-cycle phase, the "best" solution was so clearly dominant, the team (along with the CDM) determined that no such analysis was required.

Constraints. Numerous embedded controller/single-board computer, real-time operating systems, and control system development software vendors exist in the commercial market today. While the constraints developed in the first life-cycle phase still applied, some additional qualification criteria for selecting eligible C&DH systems was developed as a result of that first iteration through the activity matrix. The following characteristics were determined to be the constraints for the selection of candidate control system solutions:

- graphical, digital development and operating environment
- commercially available

- easily accessible technical support (competency was assumed)
- support a range of operating voltages (expected ADACS would drive the power requirements; a valid C&DH alternative would require voltage regulation/transformation circuitry if not an integral capability)
- operate with battery power (would have required power conversion circuitry if not an integral capability)
- support some level of wireless operation (if control could not be separated between the ground and the satellite, then support for transmitting sensor signals from the satellite and effector signals to the satellite had to be provided)

Alterables. While the alterables developed in the first life-cycle phase also still applied, some additional C&DH attributes (to directly support the *Initial Needs* previously identified) for evaluating eligible C&DH systems were developed. The following characteristics were determined to be tradeable characteristics for the candidate C&DH sub-system solutions:

- totally integrated solutions (processing hardware, operating and development software, support hardware)
- efficient executable code generation (allow larger task sets to be executed on a given system)
- non-proprietary development environment (using MATLAB/SIMULINK or similiar common tool as a foundation is preferred)

3.4.1.3 Value System Design.

With the addition of real-time concerns introduced earlier in this chapter, they needed to be factored into the values hierarchy. The concerns that need to be added include:

- **Schedulability Analysis:** Consider how well the system supports the ability to determine if the planned task set can meet its deadlines. A high score would indicate complete support for the GRMS environment, including operating system support and modeling capability for all components and I/O

- **Communications Latency:** Consider how long it takes from the time a signal gets generated until it reaches the processor. The more parts a signal has to traverse from sensor to processor, the longer it will take, and the lower it scored.
- **Development Environment:** Consider the support for an integrated, graphical development environment: development, simulation, compilation, execution, monitoring, and control are all aspects of this measure.
- **Level of Integration:** Consider how much effort will be involved in pulling the system together. The DSPACE system is the "goal" the other possible alternatives will be measured against.

These issues were added to the value hierarchy in the form of evaluation considerations again, with the hierarchy for the trade study transformed as shown in Figure 3.2.

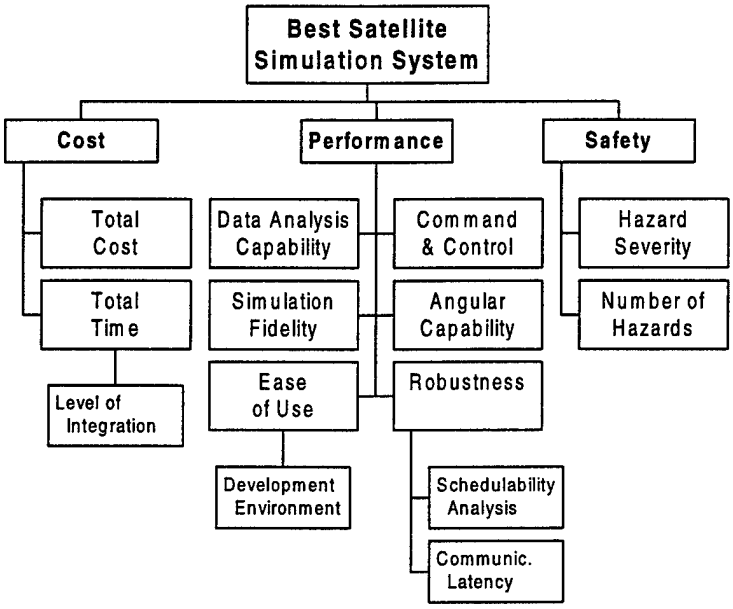


Figure 3.2 C&DH Trade Study Value Hierarchy

The intent at this stage of the design effort is to continue with a qualitative comparison between alternatives to allow for relatively quick completion of this life-cycle phase.

3.4.1.4 System Synthesis. With the framework now updated to support *Preliminary Design*, a list of candidate C&DH subsytems was developed to capture the solutions available on the market today to compete with the DSPACE system previously purchased. The SGT approach used previously was not appropriate for this phase as the team did not look at multiple subsystem components, but rather at point solutions representing the continuum of solutions available for the C&DH subsystem.

To simplify the comparison, the assumed system configuration for the alternatives was equivalent to the DSPACE system already purchased:

- embedded controller
- development software supported by simulation package
- capable of being extended to a satellite-based installation (including development and operating system software support)

Again, due to time constraints, the candidates selected were only a sampling of the systems available today, but the CDM was comfortable those candidates were a sufficient representation to permit adequate comparison of the DSPACE alternative to others. The candidate systems selected for evaluation were:

Integration of Piece Parts. This solution would involve searching tech reports, catalogs, and the Internet for the pieces required to pull together a total system, to include the software (development, compilation, monitoring and control, etc.), electronic components, and supporting hardware. There are a plethora of hardware and software solutions available, the difficulty will come in trying to find the best system, or even a solution comparable to the DSPACE system.

MATRIX$_X$ ***Solution.*** This solution would take advantage the integrated hardware and software environment. Integrated Systems, Inc. (ISI) produces an entire range of products to support aerospace and automotive products. However, its flexibility could be a problem; there may be some additional complexity in implementation useful for high-powered, mass produced, work-group oriented system developments. It is essentially the opposite end of the spectrum from the first alternative.

DSPACE ***Solution.*** This solution struck a balance, whereby a lot of power is available, but with limited, well-defined options. It may not be appropriate for all applications, but for systems that require simple implementation and maintenance, it appears to be the most straight-forward.

The next section will evaluate the alternatives to determine if this list can be narrowed any further.

3.4.2 Analysis. To assess the relative value of each alternative against the *Preliminary Design* values hierarchy, the same technique was used as in the previous life-cycle phase. Figure 3.3 shows a summary of the qualitative scores for the trade study alternatives. For those evaluation considerations that have another level below them, the lower level measures are assumed to provide the means to determine the "value" of that consideration, so those "measures" are addressed in the following discussion. Using the same criteria and assumptions as those used in the C&DH subsystem evaluation of the previous chapter (summarized in Figure 2.4, page 2-22), the alternatives were qualitatively scored in each of the lowest level evaluation considerations in the trade study value hierarchy (Figure 3.2). As in the previous evaluation, '5' is the highest score possible and the scores represent the total system score, assuming the other *SIMSAT* subsystems were the best they could be, and only the C&DH alternatives varied.

The following descriptions explain why the alternatives scored the way they did:

Total Cost Since the comparison is between the DSPACE solution already purchased and the other two solutions, the DSPACE solution got the highest score—it was effectively "given" to the team. Assuming time had permitted, the next best solution would have been the *Piece Part* solution as equivalent parts could be selected until the price was a low as possible. While unlimited time was not available, the assumption was there would be sufficient time to make it cheaper than the MATRIX$_X$ solution. As the MATRIX$_X$ solution is an "industrial strength" development environment, it was extremely expensive. As an aside, if the DSPACE components werer not available, the baseline used in this comparison (i.e., one processor, two I/O cards, connection

Alternative	Total Cost	Total Time	Data Capability	Simulation Fidelity
Integrated Piece Parts	3	1	5	2
MATRIXx Solution	0	3	5	4
dSPACE Solution	5	5	5	5

Alternative	Num. of Hazards	Haz. Severity	Command & Ctrl	Angular Capability
Integrated Piece Parts	3	3	4	3
MATRIXx Solution	5	5	5	3
dSPACE Solution	5	5	5	3

Alternative	Develop. Env.	Latency	Sched. Analysis
Integrated Piece Parts	2	3	2
MATRIXx Solution	5	3	4
dSPACE Solution	5	3	4

Figure 3.3 C&DH Trade Study Evaluation Matrix

panels, software development environment) would have cost less than $23,000, which is still less than a comparable MATRIX$_X$ solution.

Total Time/Level of Integration Since the *Piece Part* solution has little-to-no integration off-the-shelf, integration time for that solution was assumed to be the worst, with the MATRIX$_X$ second due to all the configuration required. The time required to decide what to buy would also be the worst for the *Piece Part* solution, trying to find pieces that will work together at a reasonable price. Since the DSPACE system was already in place, only the installation time was an issue, which was expected to also be less due to in-house expertise on the DSPACE system. In addition, since it was already in-place, the system-level impact of any integration issues for the DSPACE solution were assumed to be negligible.

Number of Hazards This was expected to be about the same for each of the alternatives. But since the *Piece Part* solution was not previously integrated and tested together, that alternative may have some unexpected interactions.

Hazard Severity Again, all the alternatives were expected to mostly have the same failure modes. Again, since the integrated solutions both go through some "burn-in" testing before delivery, and are also qualified to some standard, they were expected to have less things that could go wrong. For the purposes of this measure, the team assumed the *Piece Part* solution could be expected to have more that could go wrong, one or two that could lead to a more significant failure than the integrated solutions.

Data Capability Since there are sound COTS data acquistion packages available, and the dSPACE and MATRIX$_X$ solutions inherently provide that capability, all solutions get the same score.

Ease of Use/Development Environment Again, the dSPACE and MATRIX$_X$ solutions were considered virtually identical due to completeness of the development environment (graphical, integrated solution for creating, compiling, and loading programs onto the target processor, all built upon a well-defined underlying system; graphical control and monitoring also included). Finding the equivalent *Piece Part* was determined to be somewhat unlikely.

Simulation Fidelity The *Piece Part* solution was expected to have the most trouble with this, but there should be components available that could be used with MATLAB, SIMULINK, etc. to generate code, then compile it with the vendor's compiler. Development of the SIMULINK blocks for those components would likely take time though. The MATRIX$_X$ solution, as an "universal" solution, would not have the component problem, but would require some tailoring to establish the level of fidelity available from the dSPACE solution.

Command & Control Most of the alternative distinctions that might contribute to scoring differences in the area of satellite performance had already been addressed with the exception of the extent of control immediately available. The dSPACE and MATRIX$_X$ solutions were clearly equivalent, having a professional staff (and cus-

tomer base) that has developed software/routines/blocks for nearly every imaginable level of control. The *Piece Part* solution would likely have very limited pre-defined and tested routines.

Robustness/Schedulability Analysis The DSPACE operating system manual indicates it schedules task sets using *GRMS* techniques, but the extent of that support is unclear. The MATRIX$_X$ materials available indicates it supports the *Priority Ceiling Protocol* and "pre-emptive, time-based" scheduling, but whether it applies all the techniques (support for aperiodicity, etc.) is unclear. Finding a real-time operating system that would support *GRMS* might have been possible, but ensuring it adequately supported all the rest of the *Piece Part* solution was deemed unlikely.

Robustness/Communications Latency This is a system-level issue that will have more impact during the *Detailed Design* phase of the C&DH life-cycle when trying to decide about where to have the programs execute. Assigned a nominal value for the purposes of this evaluation.

Angular Capability Since the on-satellite equipment was assumed to not change for these three alternatives, they were assumed to be of the same value. They were assigned a nominal value for the purposes of this evaluation.

3.4.3 Interpretation. From the table shown in Figure 3.3, it is clear the DSPACE option is at least as good as the other alternatives in each evaluation consideration, and the best alternative in terms of *Cost*, *Time*, and *Simulation Fidelity*. While the difference in the *Cost* value would have been reduced if the team had to purchase the DSPACE system, it still would have remained the best choice. Since the DSPACE system was such a clear winner, it was not necessary to reconsider the decision to do the comparisons qualitatively. For example, if the MATRIX$_X$ solution had cost nearly the same, as the "nominal" DSPACE system, more quantitative comparisons may have been required to better differentiate between the two alternatives (in particular some of the impacts the robust MATRIX$_X$ solution would have on the "user-friendly" goals desired from the *SIMSAT* C&DH subsystem).

3.4.4 Trade Study Summary.

With confirmation of the validity of the DSPACE choice, the next step was to begin the *Detailed Design* life-cycle phase and define how best to allocate the software tasks.

IV. Detailed Design

4.1 Overview

With the DSPACE system chosen as the system to be used for the C&DH subsystem, this chapter goes to the next level of detail: what design should be implemented in the next life-cycle phase? The activity matrix for this chapter, shown in Figure 4.1, reflects the shift in focus from *Preliminary* to *Detailed Design* for this life-cycle phase.

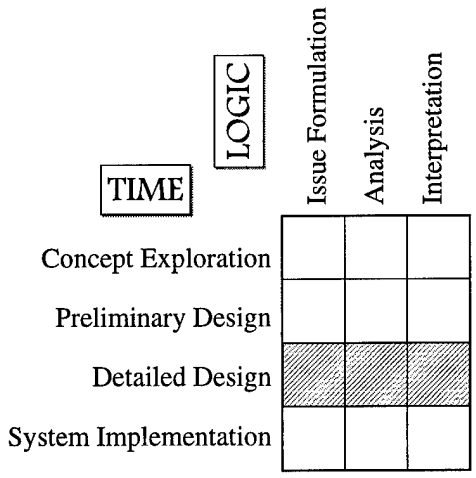


Figure 4.1 Issue Formulation (Design) Activity Matrix

To evaluate specific C&DH subsystem alternatives from a system perspective (rather than as an isolated subsystem) required assuming a baseline for the rest of the subsystems, then considering the system-wide impacts of each C&DH alternatives. This put the alternatives in the appropriate portion of the common units transfer functions[1]. Then, the team provided inputs on the degree of system variation for the various attributes to observe the likely range of values for each C&DH alternative. This variation had the potential of showing that subsystem interactions would cause a change in the recommended C&DH

[1]For instance, the impact of cost to the CDM could not be addressed by comparing the $10,000 cost of one C&DH alternative vs. $9,000 for another, but rather the total system cost for integrating either one, including all the costs of the other subsystems. Looking at Figure C.1, page C-3, it is clear that a $1,000 difference in system cost has more impact on the CDM when the nominal system cost is $90,000 than when it is $20,000.

alternative under adverse conditions. This provided the team some of the insight they needed to evaluate the risks involved with whatever recommendation they made to the CDM. The data for the baseline system combined with the various C&DH alternatives for this phase are contained in Appendix B. This chapter largely concludes the documentation of the "research" part of this system design effort—the next life-cycle phase takes the CDM-selected design and brings it to fruition.

In this hybrid *Design* phase[2], the only significant non-C&DH "decision" reached was for the ADACS subsystem. But since that decision was driven by the cost budget (the Control Moment Gyros far exceeded the available budget), this phase will be documented as a C&DH-exclusive phase. While this chapter will summarize the results of the non-C&DH subsystem "design choices" developed during their *Preliminary Design* phase, consult [8] for additional clarification.

Before making design decisions, some issues covered in *Conceptual Design* and *Preliminary Design* required further definition, while others would be left for future steps. The products of the *Problem Definition* and *Value System Design* tasks require updating to provide a more detailed framework to support specific design decisions. This chapter further documents the additional detail developed for the *Analysis* step of the hybrid *Design* phase in the *SIMSAT* design process. The chapter concludes with interface specification data for the C&DH alternatives that will be implemented in the next life-cycle phase, *Implementation.*

4.2 Issue Formulation

Since the issues typically finalized in this step (interfaces, key players, assumptions, etc.) were already identified in the *Conceptual Design* phase, few things changed. The *Scope* was unchanged, no additional *Relevant Disciplines* were identified, no new *Actors* needed to be considered, and the *Initial Needs* and *Problem Elements* still adequately captured the system boundaries and requirements. The *Problem Statement* only required

[2]With the need for the C&DH subsystem to be completed before the rest of *SIMSAT*, the C&DH design entered *Detailed Design* as the rest of *SIMSAT* design effort entered *Preliminary Design.*

a shift in emphasis to recognize the detailed design nature of this phase. In addition, during the evaluation of the value hierarchy, the list of *Constraints* was again found lacking.

4.2.1 Problem Definition. As this task covered issues for the entire life-cycle when first defined in Chapter II, the only issues that need to be documented in each life-cycle phase are those areas that change as a result of decisions in previous life-cycles, or where short-comings are identified. The parts of this task that need to be updated for this life-cycle phase are the *Problem Statement* (which is updated in every life-cycle phase) and the list of *Constraints.*

4.2.1.1 Problem Statement. To reflect the selection of DSPACE as the C&DH solution, the updated *Problem Statement* became

> *Considering the satellite system simulator is to be used as an experimental test bed for Air Force Institute of Technology (AFIT), define the detailed* DSPACE *architecture to meet the design requirements and select the best* DSPACE *architecture to support Air Force/DoD research and provide a sound teaching aid to AFIT instructors teaching in satellite control and dynamics.*

As before, this statement is based upon the overall intent of the design effort, modified slightly to delineate the focus of this iteration through the problem-solving process.

4.2.1.2 Constraints. Due to the results of the previous life-cycle phase decisions and other process developments, the list of constraints added the following:

- perform all three types of experiments
- implement with a dSPACE, Inc. solution (result of the previous iteration)
- transmitted RF power limited to that allowable in an office environment
- all electrically powered devices will be protected against damage from over-voltage, under-voltage, and short circuits
- slew rate sensing must support the slew capability of the system[3]

[3]This Measure of Merit (MOM) was, until late in this life-cycle phase, considered critical to system-level evaluations. Once the team decided it had little-to-no impact (no system alternatives would address the

- all *SIMSAT* motion must halt within 10 sec of being commanded to stop
- *SIMSAT* must make use of all the motion envelope available to it

4.2.1.3 ***Tradeables.*** Armed with the results of the system *Concept Exploration* life-cycle phase, the CDM worked with the team to develop a more detailed list of desired system attributes. As these are not "requirements" but the range of values he was willing to accept, they are considered design tradeables within the ranges shown.

- run a 10–60 minute experiment
- slew 30°–100° in 10 seconds
- pointing accuracy of 0.001° and 0.01°
- support 100–135 lbs. payload weight
- provide 100–400W to the payload

Also, the CDM assumed that any experimental "requirements" that were more stringent would be handled by the experimenter.

4.2.1.4 ***Detailed System Definition.*** Before updating the values hierarchy, the concept map in Figure 4.2 was developed to ensure all the pertinent elements of *SIMSAT* were captured, along with their inter-relationship. Since the C&DH subsystem was the major design driver at this point in the system development, the concept map is somewhat C&DH-centric.

From this rather free-form diagram, the team developed a more concise, product-oriented functional layout of the system (Figures 4.3 and 4.4). These figures were intended to capture, in more functional detail, the boundaries between subsystems without confining the team to any particular solutions. Since some of the detailed design decisions were still to be made, the team had to make some simplifying assumptions to develop this diagram. For instance, Figure 4.3 refers to a ground-based "Controller Assembly" that executes the control software—a design decision that would not actually be made until later in this

MOM), time limitations prevented revisiting the LOGICAL DECISIONS models to remove the MOM. All the C&DH subssytem results reported in this chapter used 'no value' scores for this MOM

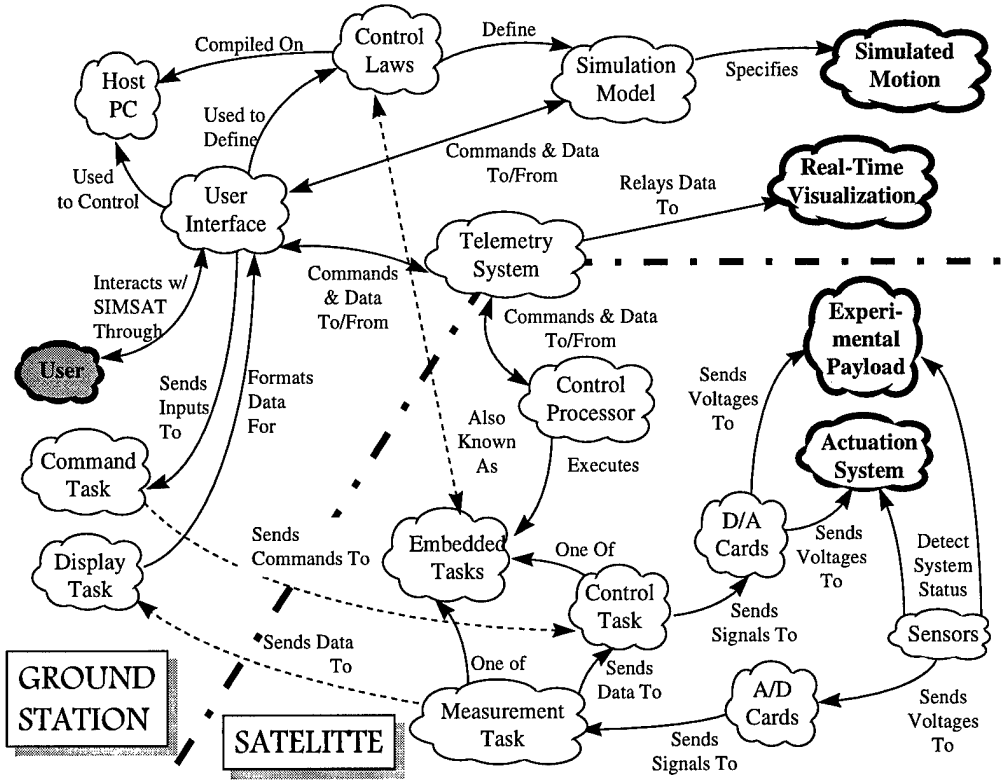


Figure 4.2 *SIMSAT* Concept Map

life-cycle phase. That box (and others) were placeholders in the diagrams to allow the team to work on defining interactions, and interfaces, between subsystems.

4.2.2 Detailed Value System Design. As the values hierarchy for this phase has to become more elaborate to support detailed decision-making, some comments regarding the team's approach to the values hierarchy design are in order. Since this was a hybrid *Design* phase for the overall *SIMSAT* system, the values hierarchy reflects that varying level of detail: significant low-level design attributes for the C&DH subsystem, higher level measures to support the other subsystem design decisions. In addition, the team selected/developed measures to support the desirable value hierarchy properties defined by Kirkwood [28:16–19]:

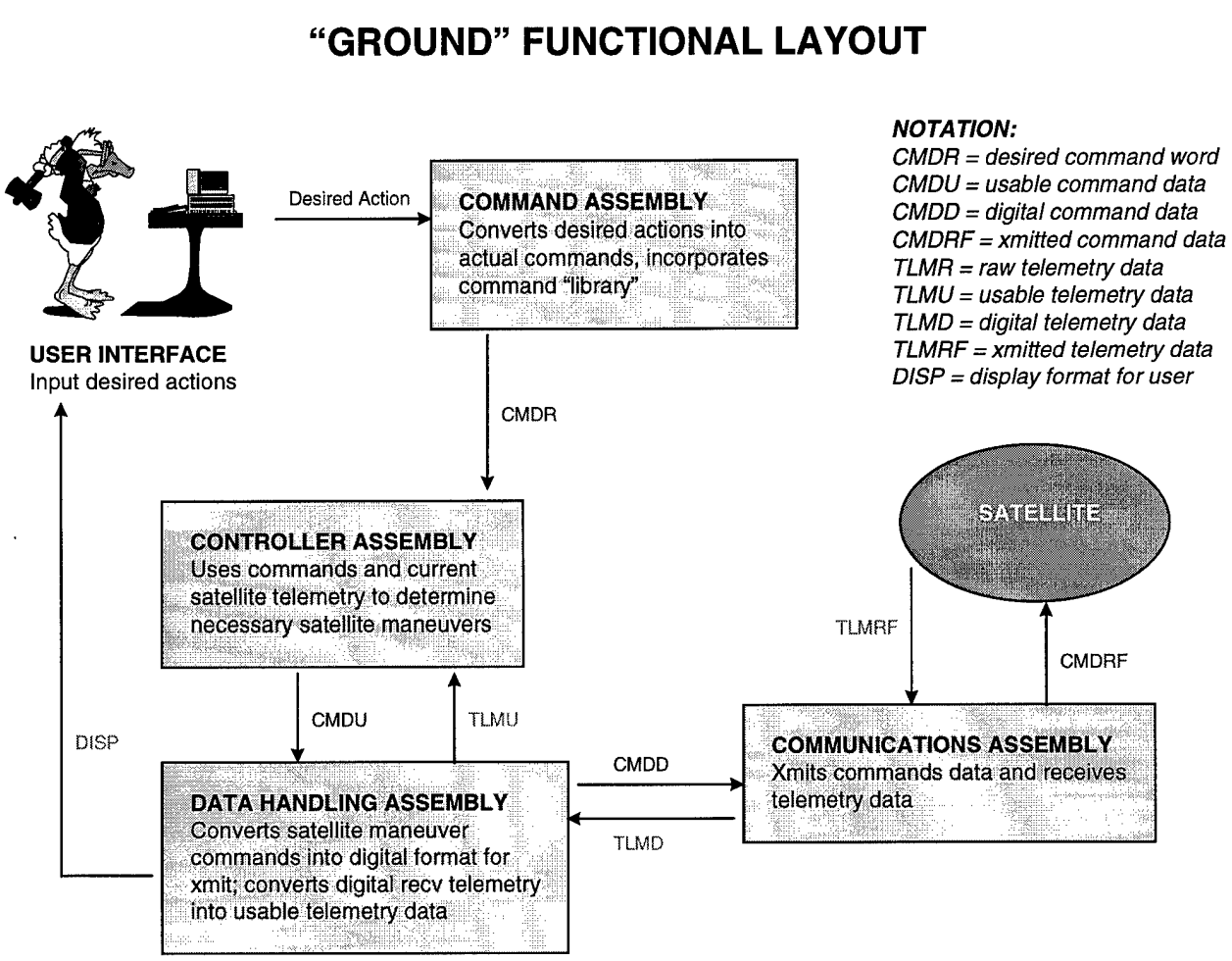


Figure 4.3 Ground Station Functional Layout [8]

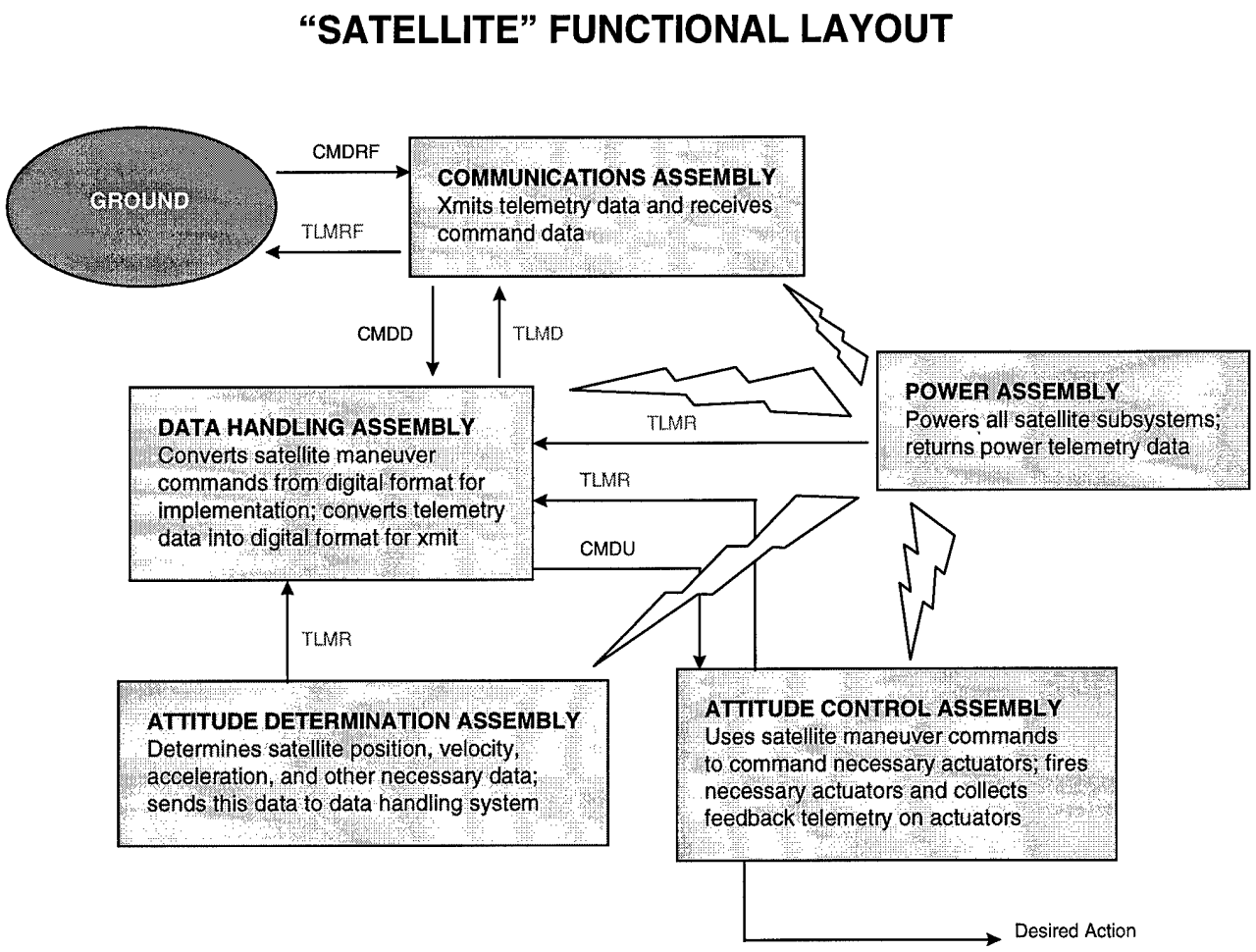


Figure 4.4 Satellite Functional Layout [8]

- **Completeness:** required to ensure adequate differentation between alternatives can be made (also known being "collectively exhaustive"). This led to a more exhaustive values hierarchy, particularly for the fundamental *Performance* value. There are two aspects to this property:

  1. the measures/metrics provide realistic representation of how well a given alternative satisfies the user
  2. to know how well an alternative satisfies the user, one needs only know how well the alternative performed at the lowest level of evaluation considerations (the level just above the measures)

- **Nonredundancy:** none of the measures/metrics should directly or indirectly score how well an alternative satisfies the user more than once (also known as being "mutually exclusive"). Violation of this principle would lead to excessive value ("double-counting") being placed on some aspect of an alternative.

- **Decomposability/Independence:** within each evaluation consideration, the value of each of the measures/metrics or evaluation considerations for the next level do not depend upon the score/value of the others. This drove the development of *Total time* as the measure to be evaluated (based upon the estimated total time to order, produce, ship, etc.) instead of trying to score each constituent time measure.

- **Operability:** every part of the hierarchy must be understandable to the user, and it must all fit together in a way that makes sense to him. As Kirkwood points out [28:18], "...it may be necessary to compromise ...some of the other desirable characteristics ..." to make it user-friendly. Kirkwood also contends that finding a way to strike this balance is more of an art than a science.

- **Small Size:** unless driven by some other property, the smaller the hierarchy the better. The three main reasons for this are:

  1. 'less' is easier to communicate (related to the *Operability* property above)
  2. less time would be required to collect and document the data for each measure/metric in the hierarchy

3. each additional measure requires the determination of the type of data to be collected, how to transform the data to "common units" to allow combining disparate measures for determining an aggregated alternative score, and how to communciate the final alternative evaluation results

With regard to the last property, Kirkwood cautions that a careful balance must be struck between the time spent developing/defining measures and the time that will need to be spent assessing alternatives [28:28]. He makes a case that, while naturally occuring metrics maybe tempting, there are few of them, and most natural measures cannot deal with interactions that typically need to be considered between related metrics [28:25]. His observation is that a carefully "defined metric" (also known as a "proxy") will not only ease data collection, it can use the proxy definition to capture the interaction between natural measures, making it easier to communicate the result of the alternative evaluations.

For all these reasons, the overall values hierarchy shown in Figure 4.5 was developed in coordination with the CDM to make sure the assessment of design alternatives considered the issues important to him, while observing the guidelines above. The fundamental values of the previous hierarchies remained unchanged, but the refinement of the supporting details (called "evaluation considerations" [28:11–12]) led to a significant modification of the hierarchy. While the next section describes what the evaluation considerations and and measures are trying to capture, a more formal definition of the Measures of Merit (MOMs) can be found in Section 4.3.4 (page 4-21). How those measures are used to quantitatively evaluate the alternatives is covered in Section 4.4.1 (page 4-28).

4.2.3 Measures Defined. This section describes the pertinent aspects of the alternatives being evaluated with respect to the CDM's values. Figure 4.5 shows how the team further elaborated the fundamental values (capitalized boldface) of Figure 2.3 into 15 evaluation considerations (boldfaced) and 24 metrics (in italics):

- **COST**
  - **Capital Cost:**

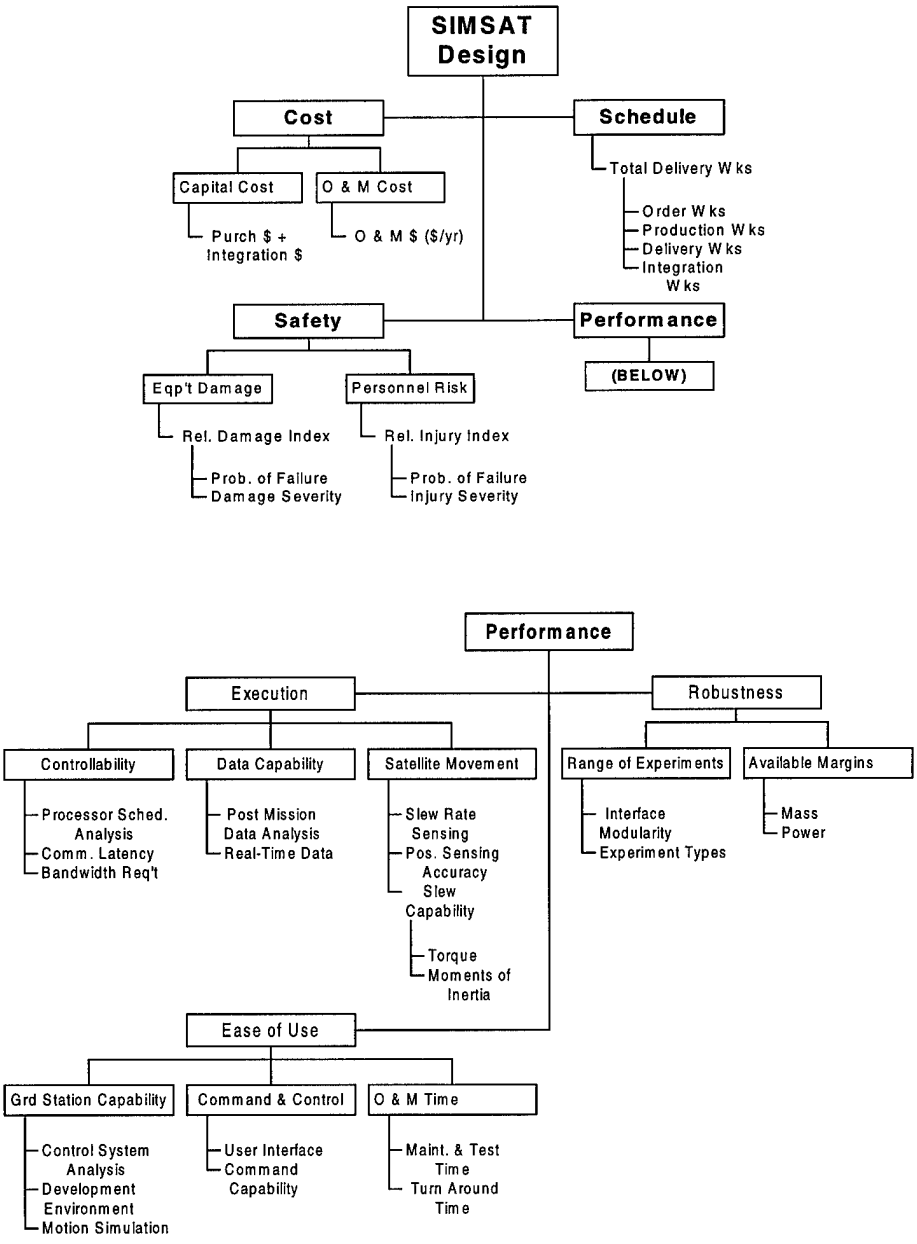


Figure 4.5 Full *SIMSAT* Values Hierarchy

    * *Purchase + Integration Cost:* what are the one-time dollars spent on parts and labor to implement *SIMSAT*?

  - **O&M Cost:**

    * *Annual Operations and Maintenance Cost:* what is the annual cost to operate and maintain *SIMSAT*?

- **PERFORMANCE**

  - **Execution:** how well will the alternative perform experiments?

    * **Controllability:** how much can *SIMSAT* do and still be controllable?

      · *Processor Schedulability Analysis:* how much support for RMA schedulability analysis does the alternative support?

      · *Communications Latency:* how great is the delay from sensor to processor to effector?

      · *Bandwidth Requirement:* how much bandwidth is required to maintain control of *SIMSAT*?

    * **Data Capability:** how much data does the alternative provide?

      · *Post-Mission Data Analysis:* able to collect data for post-mission analysis?

      · *Real-Time Data:* able to see how/what *SIMSAT* and its payload are doing during experiment execution?

    * **Satellite Movement:**

      · *Position Sensing Accuracy:* how accurately can position be measured?

      · *Slew Capability:* how far can the satellite move in a given period of time? Consider torque generation and the moment of inertia of the satellite system

  - **Robustness:** what variety of experiments can *SIMSAT* support?

* **Range of Experiments:** what range of experiments does the alternative support?
  - *Interface Modularity:* how easy is it to install rearrange assets (a measure of flexibility)?
  - *Experiment Types:* of the desired experiment types, which can be run?

* **Available Margins:** how much margin is available for experiment use?
  - *Mass:* how much more mass can *SIMSAT* support after the "base" *SIMSAT* systems are installed?
  - *Power:* how much more power can be drawn from the *SIMSAT* power system after the "base" *SIMSAT* systems are installed?

– **Ease of Use:** how easy is the total alternative to use?

* **Ground Station Capability:** how easy is the ground station part of the system to use for pre-deployment activities?
  - *Control System Analysis:* how easy is it to define the system in MATLAB and generate Bode Plots, etc. to validate system control stability? (A one-time "cost" to develop missing system elements)
  - *Development Environment:* how easy is it to program the control laws?
  - *Motion Simulation:* can the alternative predict and show how the satellite will move before actually moving the satellite?

* **Command and Control:** how easily can *SIMSAT* be controlled during experiment execution?
  - *User Interface:* how intuitive is the user interface?
  - *Command Capability:* how much of the satellite and payload/experiment activities can be controlled from the ground?

* **O & M Time:** how much time will be spent keeping the system going?

- *Maintenance and Test Time:* how much time (relatively) will be spent getting the system ready to run experiments (how easy is it to connect the payload to the satellite)?
- *Turn-Around Time:* how much time, in a one hour experiment, will be spent changing batteries?

- **SAFETY**: The *Relative Hazard Index* used for each measure below is documented in Appendix C, Figure C.4, page C-6.
  - **Equipment Damage:**
    - *Relative Damage Index:* what is the index for damage? (a proxy measure combining the probability of failure and the estimated injury severity for a failures)
  - **Personnel Risk:**
    - *Relative Injury Index:* what is the index for injury? (a proxy measure combining the probability of failure and the estimated injury severity for a failures)
- **SCHEDULE**
  - *Total Delivery Time:* how much time will be required to order, produce, deliver, and integrate all the parts from all the subsystems?

The results of this iteration of the VSD task were the key to the rest of the process. In the next section, alternatives were generated to address at least one of the fundamental values. Section 4.3 documents the tools and techniques the team used to collect data and perform the initial analysis of the alternatives considering the attributes developed in this step. In the final section for this life-cycle phase (Section 4.4), the CDM provides the weighting criteria for each of the MOMs and evaluation considerations for the "ideal" solution. That final step algebraically totals the "level of satisfaction" measures each solution provides the CDM. To aid in the *Decision-Making* task, the team used LOGICAL

DECISIONS [56], which uses the values hierarchy as its foundation (see 4.4.1, page 4-28, for more details on LOGICAL DECISIONS).

4.2.4 System Synthesis. If there are no alternatives to choose from, there is no decision to be made. While there are a number of different ways to generate alternatives, the team decided to continue using the Strategy Generation Table (SGT) technique [28:47]. While at the end of this phase it turned out that the C&DH architecture selection was the only decision made, each subsystem expert initially developed a non-dominated set of subsystem alternatives[4] to establish a system baseline. The system level solutions were then generated by permuting those subsystem solutions into integrated system solutions. The following subsections mention the choices of non-C&DH subsystem alternatives. For more details on those alternatives, consult the full team thesis [8].

4.2.4.1 Attitude Determination and Control System. The main concern for this system was the generation of torque to move the satellite—the determination of attitude, position, velocity, and acceleration was not considered during this iteration. The classes of alternatives available were Control Moment Gyros (CMGs) and Momentum Wheels (MWs).

4.2.4.2 Power System. At this stage of the life-cycle, the main concern for this subsystem was the generation of power for all the other systems on the satellite. The classes of alternatives initially considered were Single and Distributed sources.

4.2.4.3 Structures. The structural issues the team tried to address at this phase of the life-cycle was the gross arrangement of subsystems. The two main alternatives considered for this subsystem was whether the subsystems would be built-up in a "milkcrate" which would then be attached to the ends of the satellite, or if each subsystem would be built up on a disk, then the disks got stacked on top of one another in a "barbell" fashion.

[4] *Dominated Alternatives* are those alternatives that score worse on every attribute than at least one other alternative. Thus, the *non-dominated set* is that set of alternatives that are the "best" in at least one attribute than every other potential alternative.

4.2.4.4 C&DH. During *Concept Exploration*, it became clear the C&DH subsystem was dependent upon the communications subsystem. At that point, the team effectively combined the two subsystems, assumed an adequate wireless communications solution existed somewhere in the commercial world. Then at the end of this C&DH design effort, the AutoBox[5] communications requirements were passed to the next life-cycle phase for the overall *SIMSAT* design effort as a design constraint. The combined C&DH/communication subsystem alternatives generated in this step focused on satisfying the overall control system requirements.

At a coarse level, the alternatives available for using the dSPACE system were to either install an AutoBox on the satellite or run discrete sensor and effector (those elements that cause something to happen) signals from the satellite systems back to the ground. Within these major alternatives, there were some other choices related to where the control system task set executes.

First, the potential task set had to be defined: what were the likely group of tasks the C&DH system would need to handle? Based upon Figure 4.2, 4.3, 4.4 and [29], this notional task set was developed:

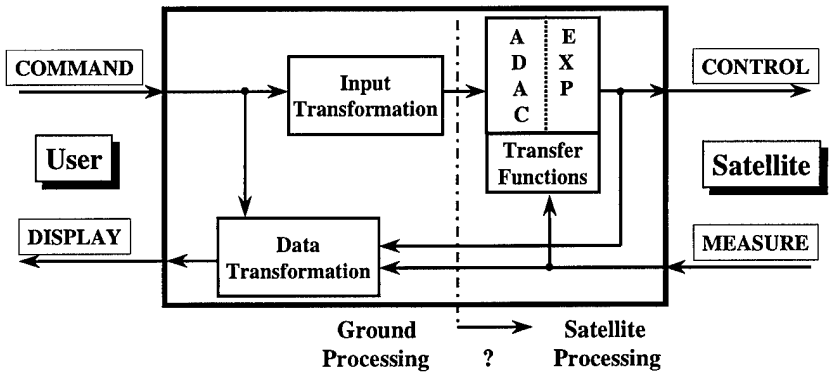
Attitude Determination and Control: the set of tasks that evaluates user commands, determines satellite status, and reacts to resolve any discrepancies between the two. Major sub-tasks include:

- detection of satellite position, velocity, and acceleration (called ADAC.MEASURE)
- resolution of error between ADAC.MEASURE and the commanded satellite position, velocity, and/or acceleration (called ADAC.EXEC)

Experiment Control: the set of tasks that evaluates user commands, determines experiment status, and reacts to resolve any discrepancies between the two. Major sub-tasks include:

- determination of experiment status (called EXP.MEASURE)

[5]A dSPACE, Inc. product that houses the processor and data acquisition boards in a self-contained, power-conditioned, shock mounted enclosure. The AutoBox is therefore ideal for satellite installation. It is designed with an integrated ethernet card to send a 10Mbps data stream to another computer running the user interface.



ADAC and EXP tasks include the corresponding MEASUREMENT task

Figure 4.6 *SIMSAT* Real-Time Software Architecture

- resolution of errors between EXP.MEASURE and the commanded experiment (called EXP.EXEC)

User Interface: the set of tasks that allows the user to command the satellite and determine the status of the system

- translate user inputs to control system commands (INPUT Transformation)
- control the data displayed to the user (DISPLAY Transformation)
  - translate user inputs to displayed graphics
  - translate satellite response to displayed graphics

The inter-relationship between these problem elements, the user, and *SIMSAT* can easily be seen in the *Real-Time Software Architecture*, Figure 4.6.

The diagram also portrays the idea that some of the tasks could be located on different processors. Separating the control and associated measurement tasks would lead to a very fine-grained software solution that would require significant communications

overhead. This would lead to the multi-processor saturation effect shown in Figure A.3 and discussed on page A-17. To reduce the impact of this effect, associated control and measurement tasks should always be hosted on the same processor.

The question mark in Figure 4.6 represents the primary decision of this phase:

what is the best allocation of real-time control tasks between satellite and ground processing?

The communication requirement between ADAC and EXP sets of tasks was expected to be low enough that the overhead would not prevent the separation of those task sets, at least for simple task sets.

From the multi-processor saturation effect discussion mentioned earlier, running all the tasks on a single processor (the most coarsely grained, totally independent solution) would lead to the best performance. However, the task set could become too complicated to execute all the above tasks on a single processor. The alternatives that arise from that difficulty include purchasing additional processor boards or using the available dSPACE assets and reallocating some of the tasks to the ground station. Considering the limited time and schedule budget resources available to this research effort, the purchase of additional processing hardware was considered infeasible.

One additional consideration used to generate the C&DH alternatives: if the tasks execute on the ground station, how will the sensor and effector signals be connected to the ground processor? The two alternatives identified to satisfy that requirement were: 1.) use the AutoBox strictly for signal consolidation, or 2.) design/select a different COTS solution. The second alternative would require additional research and integration work, but could also provide additional flexibility in the arrangement of parts and power efficiency. These, and other issues, were factored into the values hierarchy, and will be evaluated in the following sections.

4.2.5 Alternatives Summary. In summary, the potential task allocations for this control system structure were:

- ADAC and EXP tasks on the satellite (**All on Sat**)

- EXP on the satellite, ADAC on the ground (**Split**)
- All tasks executing on the ground—AutoBox providing signal consolidation (**Grd w/ AutoBox**)
- All tasks executing on the ground—a COTS solution providing signal consolidation (**Grd w/o AutoBox**)

Table 4.1 presents a summary of the subsystem alternatives; system solutions could then have been generated by permuting through the table to generate 32 ($2 \times 2 \times 4 \times 2$) solutions. As no other subsystem decisions were made during this life-cycle phase[6], the following sections evaluate the C&DH solutions, seeking the best solution to the CDM's needs, from a systems perspective.

ADACS	Power	C&DH	Structures
		All on Sat	
CMGs	Single Source	Split	Milkcrate
MWs	Distributed	Grd w/ AutoBox	Barbell
		Grd w/o AutoBox	

Table 4.1 Strategy Generation Table

4.3 Analysis

This section addresses the team's efforts at collecting data, converting that data into common units, and assessing the alternatives developed in the previous section for potential improvement in areas of weakness. The actual collection of data was only a minor part of the process—as mentioned in the previous chapter, gathering "good" data without first defining the type and character of the desired data would be pure luck.

Likewise, converting the collected data to common units was typically just a matter of plugging the measured value into a straight-forward formula for that aspect of the solution to determine the value of that "score". But determining an appropriate conversion formula

[6]The narrowing of ADACS alternatives to Momentum Wheels is not strictly a "decision"—the economic elimination caused the Momentum Wheels to become one of the first defined parts of the baseline system.

was typically an iterative process. The information documented in this section reflects the final formulas developed in conjunction with the CDM and detailed in Appendix C.

To quantitatively determine the "best" solution in this part of the problem-solving process, data must be collected. But since the various measures are inherently unrelated in what they measure and the units used to do the measureing, there needs to be a way to determine how much the CDM values the data collected for each of the alternatives. These issues are addressed by the Hall tasks *Systems Modeling* and *Systems Analysis*, respectively, within this Sage step. Each of the Hall tasks will be addressed in a separate section.

Before getting into the details of those steps, four aspects of data collection need to be addressed: resolution, scale, data quality, and range. Each is addressed in the following subsections.

4.3.1 Resolution. Before deciding on how to implement the measures, called "Measures of Merit" (MOMs) in this document (called "metrics," "measures of effectiveness," and simply "measures" in the literature), certain considerations have to be addressed. First, what resolution would be needed? This choice of resolution should be based on the intrinsic nature of the attribute, the level of detail available from the data source, and what resolution the CDM requires to make a decision. Of course every MOM is different, so these considerations had to be addressed for each one. Each measurable has a resolution, but all measures do not need to have the same type. Three resolution classes were considered for each attribute:

1. **Binary:** yes/no, off/on, does/doesn't exist, etc.
2. **Finite Number of classes:** low, medium, high; level of technology: old, current, cutting edge, etc.
3. **Continuous Real Numbers:** continuous response, cost of hardware, 0–1000, 1000–1500 etc.

4.3.2 Scale. The next aspect the team addressed was *Scale.* Closely related to resolution, *Scale* deals with how the levels of a measure are arranged and the relationship of those levels to each other. There were four classes of scales considered for each attribute:

1. **Nominal:** arbitrarily named with no particular order (e.g., apples, oranges, grapefruit, raspberries, etc. are types of fruit)
2. **Ordinal:** inherent ordering, no specific spacing (e.g., None, Some, Partial, Full)
3. **Interval:** units of measure are equal, no fixed zero
4. **Ratio:** units of measure are equal, fixed zero, ratio comparisons possible (i.e., "A is twice as good as B")

Relations inherently exist between resolution and scale. Nominal and ordinal measures are typically related to resolution levels 1 and 2 above, interval and ratio relate to level 3.

4.3.3 Quality of Data. In any data collection effort, there will be some estimates of data. In many cases, this "uncertainty" is handled by using a "confidence indicator" to derate the score for a given measure when there are varying degrees of confidence between alternatives and their measures. A typical confidence derating scheme might be:

Confidence Indicator	De-Rating Multiplier
Highly Confident (HC)	0.9
Very Confident (VC)	0.7
Confident (C)	0.5
Semi-Confident (SC)	0.3
Not Confident (NC)	0.1

Table 4.2 Confidence De-Rating

In this research effort, the team chose not to use derating multipliers because of the degree of mental modeling employed in the evaluation of alternatives This led to a fairly consistent level of data quality between alternatives (i.e., the same level of data uncertainty exists between alternatives for any given measure), negating the need for the multiplier.

In that case, derating the data needlessly narrows the score margins between competing alternatives.

4.3.3.1 ***Range.*** Compared to the other aspects of data collection, establishing a measure's range is relatively intuitive: the range of values the CDM is willing to accept for a particular measure. Anything outside that range will either be unacceptable, or represents an improvement in the measure the CDM is not willing to pay any extra for. Keep in mind that some of these measures will have greater value for lower scores (such as *Cost*), while others will have greater value from higher scores (such as *Ease of Use*). That conversion, as well as the "normalization" of all the different measures to common units, will be addressed in *System Analysis*, Section 4.3.6 (page 4-23).

4.3.4 System Modeling. As mentioned in Chapter II, this step in the Problem Solving process has three main sub-steps:

1. define the scales and resolution for each of the measures of merit identified in the values hierarchy
2. determine the expected quality and range for each of the measures of merit identified in the values hierarchy
3. collect the data using the scale and resolution decisions above. If there is any uncertainty about the actual value a given alternative will have, try to find ways to describe the uncertainty probabilistically

With the measures in the values hierarchy completely defined, the team developed the scale and resolution for each measure. Table 4.3 records the results of the team's effort.

4.3.5 Data Collection. During the *Concept Exploration* life-cycle phase, the team collected all data through mental modeling, largely based on knowledge previously accumulated. For this iteration, additional information was required to fill in the details for some of the measures. To collect the data, the team conducted open publication searches, group consensus, manufacturers data (either brouchures and catalogs, or their

Measure	Resol'n	Scale	Range	Units
Capital Cost	Real	Ratio	10 - 100	$1,000
O & M Cost	Real	Ratio	0 - 20	$1,000
Total Del. Time	Real	Ratio	12 - 56	weeks
Rel. Damage Ind.	Finite	Interval	10–20	—
Rel. Injury Ind.	Finite	Interval	10–20	—
Bandwidth Req'ts	Finite	Ordin.	Low, Med, High	—
Comm. Latency	Finite	Ordin.	Minimal, Mod., Significant	—
Ctrl. Sys. Analysis	Finite	Ordin.	Minimal, Partial, Full	—
Development Env	Finite	Ordin.	Text, Graphical, Visual	—
Exp. Types	Finite	Ordin.	None, Educ, Rigid, Flex	—
Interface Modul'y	Finite	Ordin.	None, Some, Partial, Full	—
Maint. & Test Time	Finite	Ordin.	V. Low, Low, Med, High, V. High	—
Mass Margin	Real	Ratio	0–300	kgs
Motion Sim.	Binary	Nomin.	Yes, No	—
Pos. Sens'g Acc.	Real	Ratio	$10^{-1} \rightarrow 10^{-3}$	—
Post-Msn Anly's	Binary	Nomin.	Yes, No	—
Power Margin	Real	Ratio	0–15	Amp-Hrs
Proc. Sched. Anly's	Finite	Ordin.	None, Unsupp'd, Mod., Full	—
Real-Time Data	Binary	Nomin.	Yes, No	—
Slew Capability	Real	Ratio	3–10	deg/sec
Turn Around Time	Real	Ratio	0–8	hrs
User Interface	Finite	Ordin.	Minimal, Partial, Full	—

Table 4.3 Measures of Merit Details

web sites), and observations of "obvious" characteristics[7]. The information gathered was then typically used in mental modeling to assess the alternative attributes. Direct measures, such as *Capital Cost*, could be used directly, while most of the others, such as *Total Delivery Time* required some extrapolation and assumptions developed during the mental modeling exercises.

Appendix B contains the tables of raw data the team collected for the four alternatives. Appendix B also provides the basis for the scores assigned to the various measures. As mentioned before, the data in Appendix B reflects *SIMSAT* system-level data (not just for the C&DH subsystem) as the C&DH subsystem alternatives were varied. To capture

[7] *Obvious* means the data for the measurable was readily apparent to even the most casual observer. For the MOMs the team employed this technique on, there was always an extremely limited range of data to be discovered (typically, the MOM had a Y/N type of resolution).

the uncertainties in the collected system-level data, the appendix also contains an estimate of variability in the data around the most-likely value for each of the alternative measures. To summarize the nominal data, Table 4.4 contains the most-likely score for each measure of each C&DH subsystem alternative (measures arranged alphabetically within each fundamental value).

From the data in this table, none of the alternatives was dominant—all four had strengths in different areas. But the CDM would consider some attributes more critical than others. To factor that consideration into the analysis, the data collected had to be converted to common units, and weights assigned to each measure. This provides a mechanism to algebraically combine the score in a way that reflects how valuable a given piece of data is across the spectrum of possible scores for that MOM, and how relatively important a given MOM is to the CDM. Both of these issues are addressed in Section 4.4.1, starting on page 4-28.

4.3.6 System Analysis. Chapter II discusses the aspects of this step in detail. In summary, this step is used to:

1. define the functions used to convert the measures to common units
2. convert the raw data to common units and consider the results—if what was expected to be a "good" alternative turns out to be less remarkable, determine why and either
   - develop a new altermative to address the shortcoming or
   - change the raw data to common units transform to mirror CDM expectations

The team routinely met with the CDM to develop the transform functions for each of the measures of merit—the common units range was established as 0–10. Appendix C contains the resulting system-level common units transform functions. Table 4.5 consolidates the results of those functions in one place with Table 4.6 as the key to the abbreviations.

Based upon this information, the nominal C&DH alternative values (from the raw data in Table 4.4) are shown in Table 4.7 (again, with a baseline system configuration).

Alternative	Cost		Time	Safety		Performance		
	Capital	O& M	Delivery	Damage	Injury	B'width	Cmd Cap	Comm Lat'cy
All on Satellite	$ 29.5K	$ 8K	34	18	18	Low	Full	Min
Split	$ 34.5K	$ 8K	35	18	18	Mod	Full	Mod
Grd w/ AutoBox	$ 34.5K	$ 8K	37	18	18	High	Full	Signif
Grd w/o AutoBox	$ 34.5K	$ 8.5K	40	16	16	High	Full	Signif

Alternative	Performance (cont'd)							
	Ctrl Anly	Dev Env	Exp Typ	Intfc	M/T Time	Mass Mrg	Mot Sim	Pos'n Acc
All on Satellite	Full	Object	Full	Part'l	V Low	100	Yes	10^{-2}
Split	Full	Object	Full	Part'l	V Low	100	Yes	10^{-2}
Grd w/ AutoBox	Full	Object	Full	Part'l	V Low	100	Yes	10^{-2}
Grd w/o AutoBox	Part'l	Graph	Rigid	Full	Low	120	Yes	10^{-2}

Alternative	Performance (cont'd)							
	Post Msn	Pwr Mrg	Proc Sch	Real-Time	Slew Cap	Slew Sens	T-A Time	UI
All on Satellite	Yes	7	Full	Yes	60	0	1	Full
Split	Yes	7	Full	Yes	60	0	1	Full
Grd w/ AutoBox	Yes	7	Full	Yes	60	0	1	Full
Grd w/o AutoBox	Yes	10	Mod	Yes	70	0	1	Partial

Table 4.4 Alternative Raw Data

Attribute	Units	Ref. Fig.	0	1	2	3	4	5	6	7	8	9	10
Capital Cost	$1,000	C.1	100							50			10
O & M Cost	$1,000/yr	C.2	10							5			0
Total Delivery Time	weeks	C.3	56					48					25
Relative Damage Index	—	C.5	10		12		14		16		18		20
Relative Injury Index	—	C.6	10		12		14		16		18		20
Bandwidth Requirements	—	C.7	H					Mod					Low
Command Capability	—	C.8	S/S							ADAC			Full
Communications Latency	—	C.9	Sig					Mod					Min
Control Systems Analysis	—	C.10	Min							Par			Full
Development Environment	—	C.11	T							Gr			OO
Experiment Types	—	C.12	N		Ed					R			Full
Interface Modularity	—	C.13	N		S				Par				Full
Maintenance & Test Time	—	C.14	VH			H		Mod		L			VL
Mass Margin	kgs	C.15	0								100		300
Motion Simulation	—	C.16	N										Y
Position Sensing Accuracy	deg/sec	C.17	$10^{-}1$									$5 \times 10^{-}3$	$10^{-}3$
Post-Misson Analysis	—	C.18	N										Y
Power Margin	amps/hr	C.19	0					2					15
Processor Schedul. Analysis	—	C.20	N		Unsup					Mod			Full
Real-Time Data	—	C.21	N										Y
Slew Capability	deg/10 sec	C.22	30							60			100
Slew Rate Sensing	—	C.23											
Turn-Around Time	hrs	C.24	8	3									0
User Interface	—	C.25	Min							Par			Full

Table 4.5 Consolidated Utility Table

Abbr.	Meaning	Abbr.	Meaning	Abbr.	Meaning	Abbr.	Meaning
VH	Very High	Y	Yes	T	Text	S/S	Start/Stop Only
H	High	N	No/None	Gr	Graphical	ADAC	ADACS Control Only
Mod	Moderate	Sig	Significant	OO	Object-Oriented	Ed	Education Only
L	Low	Min	Minimal	S	Some	R	Rigid and Educ. Exp. Only
VL	Very Low	Par	Partial	Unsup	Unsupported	Full	Full Capability

Table 4.6 Legend for Table 4.5

Alternative	Cost		Time	Safety		Performance		
	Capital	O& M	Delivery	Damage	Injury	B'width	Cmd Cap	Comm Lat'cy
All on Satellite	6	4	7	8	8	10	10	10
Split	5	3	6	8	8	5	10	5
Grd w/ AutoBox	5	3	5	8	8	0	10	0
Grd w/o AutoBox	5	3	4	6	6	0	10	0

Alternative	Performance (cont'd)							
	Ctrl Anly	Dev Env	Exp Typ	Intfc	M/T Time	Mass Mrg	Mot Sim	Pos'n Acc
All on Satellite	10	10	10	6	10	8	10	8.6
Split	10	10	10	6	10	8	10	8.6
Grd w/ AutoBox	10	10	10	6	10	8	10	8.6
Grd w/o AutoBox	7	7	7	10	7	8.5	10	8.6

Alternative	Performance (cont'd)							
	Post Msn	Pwr Mrg	Proc Sch	Real-Time	Slew Cap	Slew Sens	T-A Time	UI
All on Satellite	10	9.1	10	10	7	0	4.7	10
Split	10	9.1	10	10	7	0	4.7	10
Grd w/ AutoBox	10	9.1	10	10	7	0	4.7	10
Grd w/o AutoBox	10	9.7	7	10	8.1	0	4.7	7

Table 4.7 Alternative Common Units

4.3.7 Summary. Unlike the previous life-cycle phases, the "common units" answers in this phase show no dominant alternative; the `All on Sat` option would appear to be the best if all that was considered was the number of high marks it got. But if the CDM determines the weight of some attribute in the `Grd w/ AutoBox` solution was *THE* most important attribute (not likely since that would imply all the rest do not matter), the latter solution could turn out to be the "preferred" solution. The next section goes through the multi-attribute comparison process required in a situation like this.

4.4 Interpretation

Now that all the data has been collected, converted to common units, and weights assigned to each portion of the values hierarchy, the final step in the problem-solving process can begin. This section focuses on the *Decision-Making* task, taking the "common unit" values developed from taking the raw data of Table 4.4 through the value functions and corresponding details of Appendix C. After completing the analysis of the data, the chapter concludes with a recommendation for the best C&DH alternative, and the implications of that choice for the next phase (i.e., any new constraints, problem elements, etc.). A critical piece of that documentation is the physical interface and connector diagrams for power and signal wires to help define those constraints. The *Implementation* of the optimal architecture defined in this chapter is covered in Chapter V.

By the Sage summary documented in Table 1.2, this section documents the Hall *Decision-Making* and *Plan for the Next Phase* tasks. Each task is covered in a separate subsection as before.

4.4.1 Decision-Making. In many ways, this was the focus of all the previous work in the problem-solving process. The ultimate goal of the process was to determine the "best" solution to a problem, and the process to this point either formally defined the problem, elucidated the aspects of the solution the CDM would use to determine the value of a given solution, generated a variety of solutions, collected data for those solutions, normalized that data, or considered the implications of that data (determined the non-dominated solution set, optimized alternatives as appropriate, etc.). This chapter

documents how the team took the results of all that work, determined how to algebraically combine the data to rank alternatives, and make a final recommendation. Once the ranking of alternatives was complete, the final section of this chapter documents the implications of that decision to the rest of the life-cycle.

To complete the decision-making process, the system-level value of each alternative had to be determined. Since the C&DH subsystem was more defined than the other subsystems, some assumptions were made about the expected scores for each of the system-level attributes without the C&DH system. The impacts of each C&DH alternative was then factored into the baseline system design to assess the integrated system. Then a sensitivity analysis was conducted to see if the alternative rankings changed with changes in the fundamental value weights. If any of those changes caused a re-ordering of the alternatives, a sensitivity analysis of the next level was conducted to provide detailed insight into the impact of the attributes. Finally, to determine the risk of selecting a given C&DH alternative, a range of values was included along with the original values and the results graphed to get a visual indication of the amount of overlap. Each of these steps is covered in a sub-section below.

*4.4.1.1 **Alternative Valuation.*** The technique used to algebraically combine the scores for each alternative to determine an alternative's *Total Value* (or TV of a given alternative) is explained by Kirkwood [28:72]. The process takes the normalized value of each measure, multiplies it by an appropriate weighting factor, and sums all the weighted, normalized values. A simplified version of the formula to determine TV of alternative (x) is:

$$TV(x) = \sum_{i=1}^{n} w_i v_i(x) \quad \left\{ \sum_{i=1}^{n} w_i = 1 \right\} \tag{4.1}$$

where

i = attribute under consideration
n = number of attributes
w_i = weight for attribute i
$v_i(x)$ = normalized value of attribute i for alternative (x)

4.4.1.2 Attribute Weight Development. The difficulty is in developing an analytically sound technique for determining weights for each attribute. Recall that the normalized values for each attribute were established by the team and CDM to vary from 0–10. To be analytically sound, $TV(x)$ must also vary from 0–10 [28:61]. To accomplish this, the weights associated with each attribute supporting a given evaluation consideration must sum to 1, all the way to the top design goal ("Best SIMSAT Design"). In other words, the weights associated with *Mass Margin* and *Power Margin* must sum to 1 (so that the *Available Margins* evaluation consideration falls in the range of 0–10), and the weights associated with the *Cost*, *Schedule*, *Safety*, and *Performance* fundamental values must sum to 1.

With the raw data available to them, the team worked with the CDM to assign weights using these guidelines. Kirkwood describes some rigorous techniques [28:68–72] for establishing attribute weights. In this effort, however, the weights were established in a less formal, bottoms-up approach. With the time constraints the team was operating under, and the familiarity the CDM had with this analytic technique made this compromise acceptable to him—he was satisfied the resultant weighting of the value hierarchy accurately reflected his perspectives.

Figure 4.7 depicts the final values hierarchy, with weights assigned to each evaluation consideration and alternative attribute. But the attribute weights shown in Figure 4.7 are not the same as the $w_i(x)$ in equation 4.1—w_i is the combined value of all the weights between the attribute and the top of the hierarchy. For example, $w_{Rel\_Inj\_Ind} = 1.0 \times 0.8 \times 0.1 = 0.08$. After the math is done for all the attributes, the total of the weights should still sum to 1 (as shown in the second part of equation 4.1). Table 4.8 records the effective weights for each attribute (w_i), arranged in order of overall importance. This table provided a final sanity check for the CDM to make sure that those attributes he considered important fell higher on the table than those attributes considered less important. In fact, analysis using data such as that shown in Table 4.8 led the CDM to revise his weights until the *Performance* attributes he considered critical were near the top of the list.

With all the data now in place, the nominal ranking of C&DH alternatives could take place. The team chose to use LOGICAL DECISIONS [56] to record the data and calculate the

Attribute	Weight
Total Delivery Time	0.20
Capital Cost	0.16
Relative Injury Index	0.06
Mass Margin	0.045
Power Margin	0.045
Ops & Maintenance Cost	0.04
Relative Damage Index	0.04
Experiment Types	0.036
Communications Latency	0.032
Slew Capability	0.032
Slew Rate Sensing	0.032
Command Capability	0.03
User Interface	0.03
Post-Misson Analysis	0.028
Bandwidth Requirements	0.024
Interface Modularity	0.024
Processor Schedulability Analysis	0.024
Control System Analysis	0.02
Development Environment	0.02
Motion Simulation	0.02
Position Sensing Accuracy	0.016
Maintenance & Test Time	0.015
Turn Around Time	0.015
Real-Time Data	0.012
TOTAL:	1.0

Table 4.8 Attribute Weights

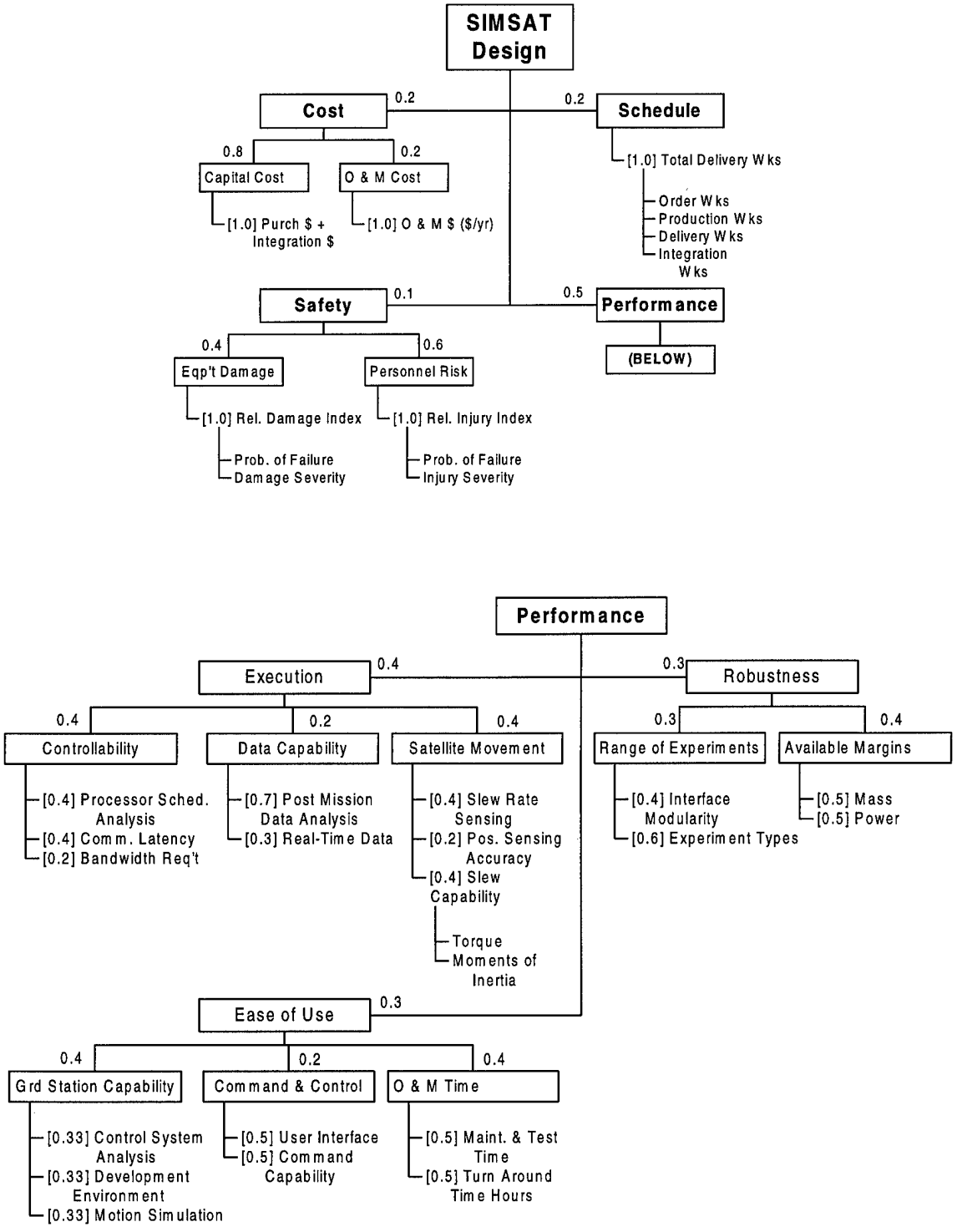


Figure 4.7 Weighted *SIMSAT* Values Hierarchy

results due to its power in dealing with multi-attribute problems. LOGICAL DECISIONS was very intuitive, using a values hierarchy-based model for its algorithmic solutions. It also provided graphical manipulation tools to help define the transform functions, and provided a wide variety of techniques for generating weights[8] for the entire values hierarchy.

After the value functions were defined for each attribute, and weights assigned, data entry for alternatives was accomplished through a spreadsheet-like interface. At that point, LOGICAL DECISIONS could produce a wide variety of reports, such as the ranking of alternatives (by fundamental values, evaluation considerations, and measures), sensitivity analysis, and effective weights (based upon the range of "value" for a given attribute and the weight of that attribute).

The first graph the team used was a pictorial ranking of the alternatives and how the fundamental values scored for each of those alternatives. The ranking of C&DH alternatives is shown in Figure 4.8.

4.4.2 Sensitivity Analysis. Sensitivity analysis is typically handled in two different, yet related, ways. *One-way* sensitivity analysis varies one characteristic at a time to see what happens to the ranking of alternatives. A typical one-way analysis is to vary the weights [28:82], one at a time to see where the rankings change (i.e., considering how close the "best" alternative is to changing if the CDM were to reassess the previously assigned weights). As mentioned in the reference, this analysis is not simply a matter of changing the weight of one attribute—the sum of all attribute weights must still sum to 1, and the ratio of the non-varying weights must be maintained. While Kirkwood provides a complex formula to address this issue, LOGICAL DECISIONS handles the issue automatically.

Another form of *One-way* sensitivity analysis compares two alternatives, to see where their relative strengths are. Because the attributes are typically ordered from the greatest impact to the least, the resulting diagram is cone-shaped and the diagram is commonly referred to as a "tornado" diagram [7:161–162]. A limited form of this type of analysis is

[8]Weights can be entered directly into LOGICAL DECISIONS, as in this effort, or LOGICAL DECISIONS can be used to help apply the more rigorous weight definition techniques described by Kirkwood [28:68–72]

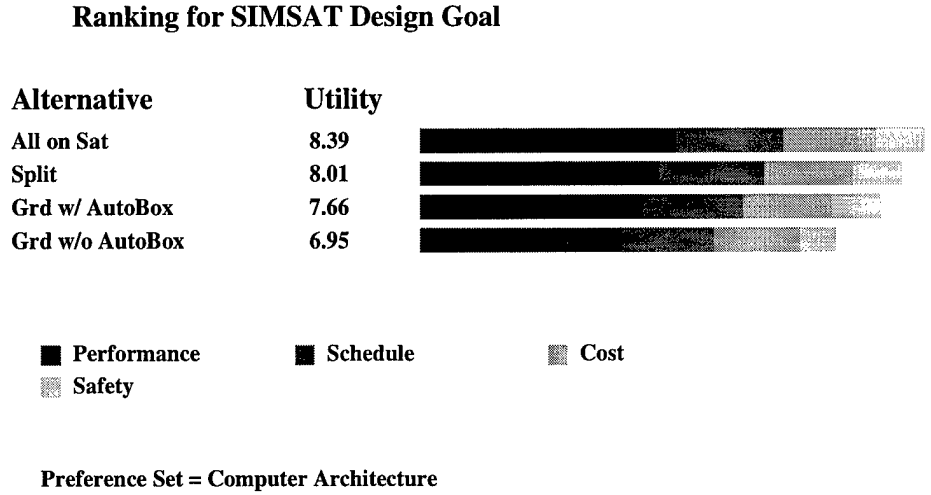


Figure 4.8 Ranking of C&DH Alternatives

also handled by LOGICAL DECISIONS (only looks at one pair of alternatives at a time), the results of which are contained in Section 4.4.2.2.

Two-way sensitivity analysis considers the results of the tornado diagrams. Using the two most critical characteristics (those with the longest bars), a comparison between them (one on the x-axis, the other on the y-axis) is done to find the boundaries between alternatives as each is varied across its range. However, no such tool was found in LOGICAL DECISIONS, and time did not permit transferring the model to another tool, such as DPL, to perform this analysis. Considering the limited overlap between alternatives shown in Figure 4.16, page 4.16, this shortcoming did not concern the CDM—he was satisfied with the results of the sensitivity and risk analysis documented below.

4.4.2.1 One-Way Analysis Pt. I. Producing an exhaustive set of one-way analysis charts is unnecessary; the only charts that have to be generated are those that convey useful information. In other words, if the first set of charts shows alternatives

A, B, and C were consistently ranked first, second, and third for the *Cost*, *Safety*, and *Schedule* values, but re-order to, say, C, B, and A under the *Performance* value, the only set of charts that would need to be explored further are those related to *Performance* (to determine why the ranking changed). This exploration is important for two reasons:

1. it provides a sanity check to ensure the ranking for a given measure/evaluation consideration turns out as expected
2. once the measure/evaluation consideration causing the reversal is found, the CDM can better assess the risks involved with choosing one alternative over another (i.e., the likelihood that the values assigned to that attribute are valid)

As will be seen in the following figures, there was no need to go any farther than the first level; there were no alternative rank reversals.

This rationale for determining how far to conduct this sensitivity analysis implies starting at the top of the value hierarchy, and continuing toward the bottom as required. Figure 4.9 provides the LOGICAL DECISIONS results of varying the weight of the *Cost* value. From this diagram, the recommended alternative was not sensitive to the weight assigned to *Cost*, so the "best" alternative would remain the alternative of choice.

Figure 4.10 provides the results of varying the weight of the *Schedule* value. From this diagram it was again clear that the "best" alternative would always remain the alternative of choice.

Figure 4.11 provides the results of varying the weight of the *Safety* value. Again, the "best" alternative would not change. It was interesting to note that, when *Safety* is the only thing that matters, the *Grd w/o AutoBox* alternative is totally unacceptable, but any of the *AutoBox* alternatives were acceptable. That makes sense based upon the scores assigned to each of the relative safety attributes (Table 4.4, page 4-24).

Figure 4.12 provides the results of varying the weight of the *Performance* value. As before, the "best" alternative would always remain the alternative of choice. In fact, as *Performance* becomes more important, the "best" alternative becomes an even better choice. And because of the order of the alternatives, choosing the "best" alternative will

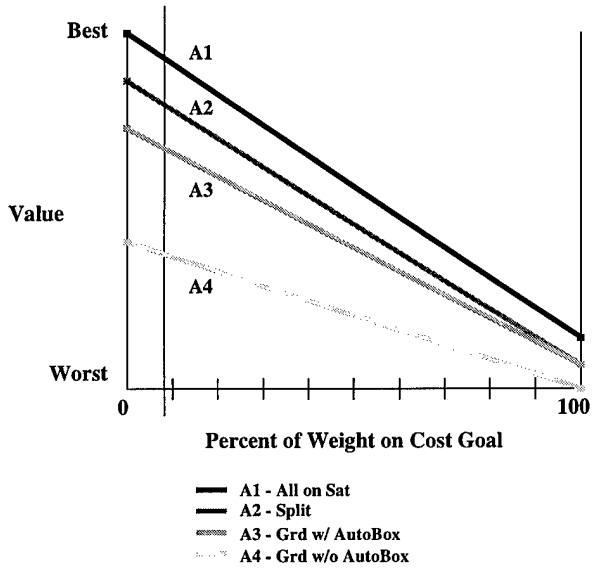


Figure 4.9 Sensitivity Graph—*Cost* Weight

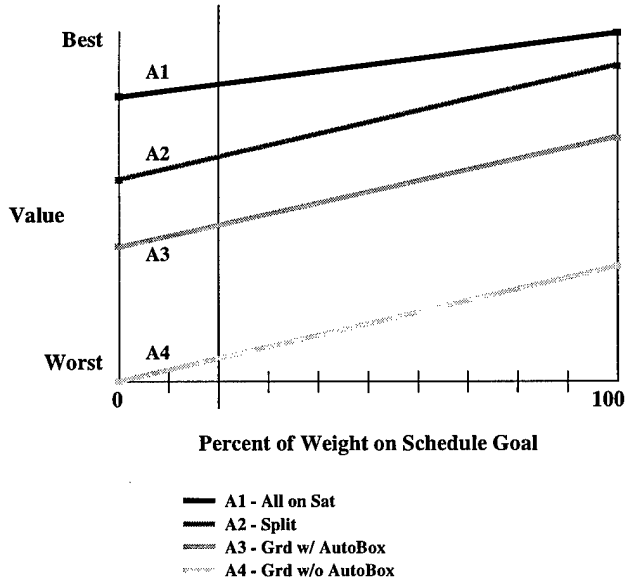


Figure 4.10 Sensitivity Graph—*Schedule* Weight

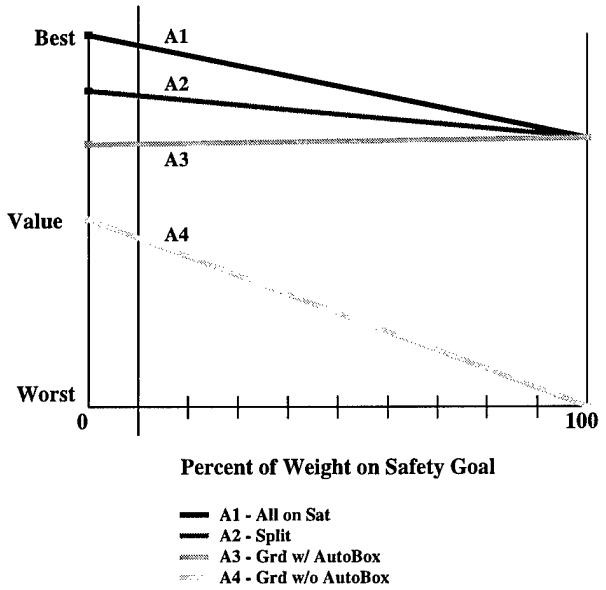


Figure 4.11 Sensitivity Graph—*Safety* Weight

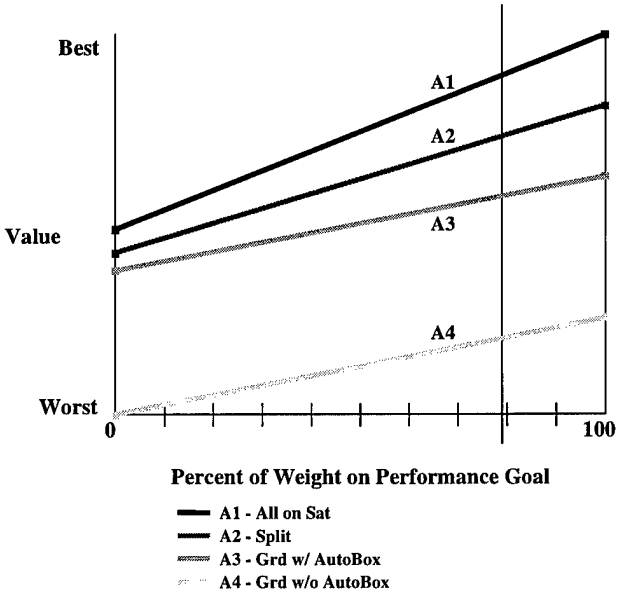


Figure 4.12 Sensitivity Graph—*Performance* Weight

have little risk: either of the next two alternatives could be implemented with a few more parts and a little extra integration time.

From all indications in this part of the one-way sensitivity analysis, there will be no change in the recommended alternative. As another check, the team ran the second type of one-way sensitivity analysis.

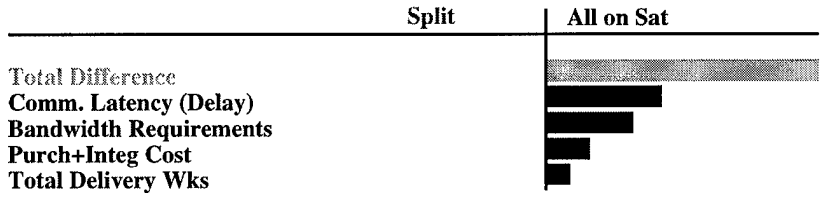
4.4.2.2 ***One-Way Analysis Pt. II.*** Following logic similiar to that of the *One-Way Analysis*, the comparisons of alternatives must start from the top and work its way down. If there is a indication that a given attribute for a lower ranked alternative is significantly better than that of the alternative being compared to, it may be worth doing a comparison with the next highest alternative to see whether it could factor into a rank reversal as well. Intuitively, it would seem that attribute should also have shown up in the one-way analysis, but this is not always the case (it may be weighted very lightly).

Using that philosophy, Figure 4.13 provides the results of comparing the first two alternatives. Clearly, the "top" alternative was better than the second choice.

Figure 4.14 provides the results of comparing the next two alternatives. Clearly, the "second" alternative was better than the third choice, nearly as convincingly as the first comparison. The reason *Total Cost* did not factor into this comparison was that the only difference between these alternatives was where the software was executing (satellite vs. the ground).

Figure 4.15 provides the results of comparing the final two alternatives. While the "third" alternative was still better than the non-AutoBox alternative, the reasons were much different than in the previous comparisons. The *Grd w/ AutoBox* alternative was preferred over the other alternative in a number of different areas, but not by much. But the *Grd w/o AutoBox* alternative was actually preferred in a few other areas (it was a little better for *Slew Capability* and "Other" categories). Based upon CDM input, the chances of those attributes out-weighing the other characteristics was very slim, so no additional comparisons were run.

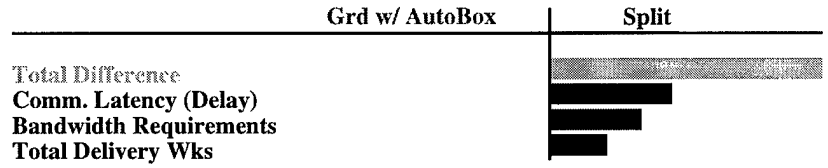
Overall Value for	**All on Sat**	**8.39**
	Split	**8.01**
	Difference	**0.38**



Preference Set = Computer Architecture

Figure 4.13 Tornado Diagram—*All Sat* vs. *Split*

Overall Value for	**Split**	**8.01**
	Grd w/ AutoBox	**7.66**
	Difference	**0.35**



Preference Set = Computer Architecture

Figure 4.14 Tornado Diagram—*Split* vs. *Grd w/ AutoBox*

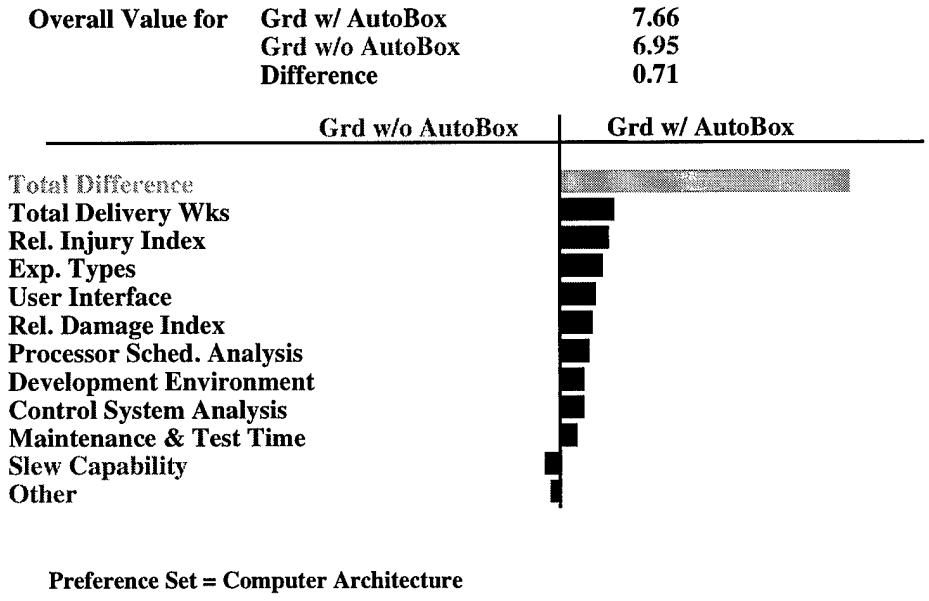


Figure 4.15 Tornado Diagram—*Grd w/ AutoBox* vs. *Grd w/o AutoBox*

From this one-way analysis, the recommendation remained the same: implement the DSPACE solution with the AutoBox and execute all the control software on the satellite.

4.4.3 Recommendation. From all this analysis on the C&DH system, a clear recommendation was delivered, as well as sound rationale for that choice. Since the other subsystem alternatives either were not able to differentiate between alternatives, or ended up with only one set of potential alternatives, there were no system-level interactions to worry about.

As one final check, the team surveyed the market to develop a reasonable range of values their subsystems might cause in the system-level measures. These ranges were then used to assess the potential that the "nominal" C&DH alternative results might overlap. This overlap would require CDM consideration of the risks involved in choosing the recommended alternative. The ranges of values the team developed are summarized in Appendix B, along with some background data on how all the numbers were arrived at. Figure 4.16 shows there was some minor overlap of the first and second, second and third alternatives.

This risk involved in this overlap can be dealt with fairly easily. Recall that the range of attribute scores used to generate that graph assumed a uniform distribution for the range to create a worst case scenario. As the team made fairly conservative estimates for the nominal values to begin with, a uniform distribution is overly conservative, particularly on the negative value size (i.e., it is less likely the cost estimates were too low than the estimates being too high. While no quanitification of that conservatism was attempted, the team expected it would be enough to overcome the minimal overlap between alternatives. And even if that were not enough to eliminate the risk completely, the team realized the time and money involved in changing from the primary to either the second or third alternative was very trivial.

For those reasons, the overlap does not impact the primary recommendation:

Implement the AutoBox solution with all the control software executing on the satellite

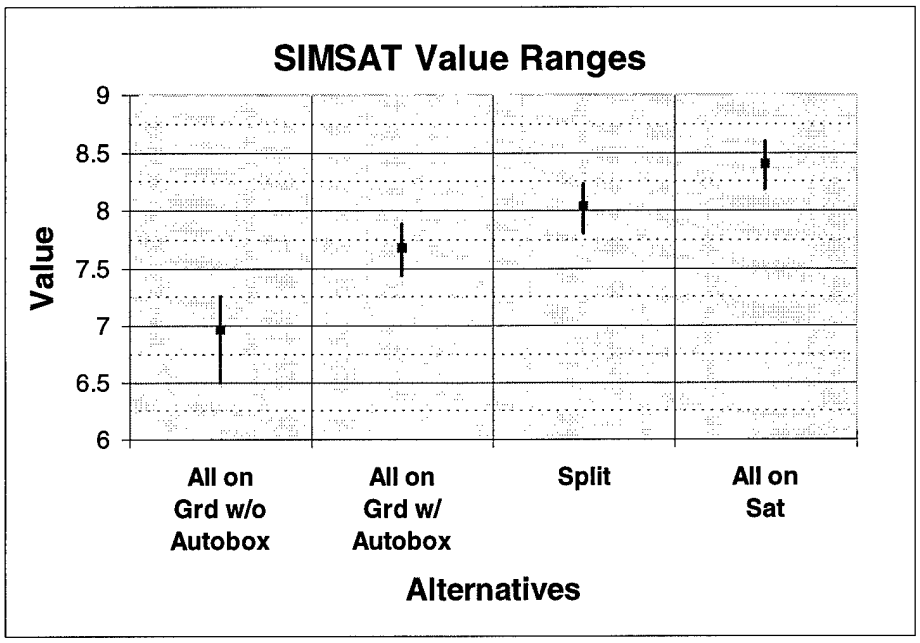


Figure 4.16 Risk Graph: Range of System Values

The results for this hybrid design phase are reflected in Table 4.9. As before, consult [8] for the rationale behind the non-C&DH subsystem choices.

4.5 Plan for Next Life-Cycle Phase

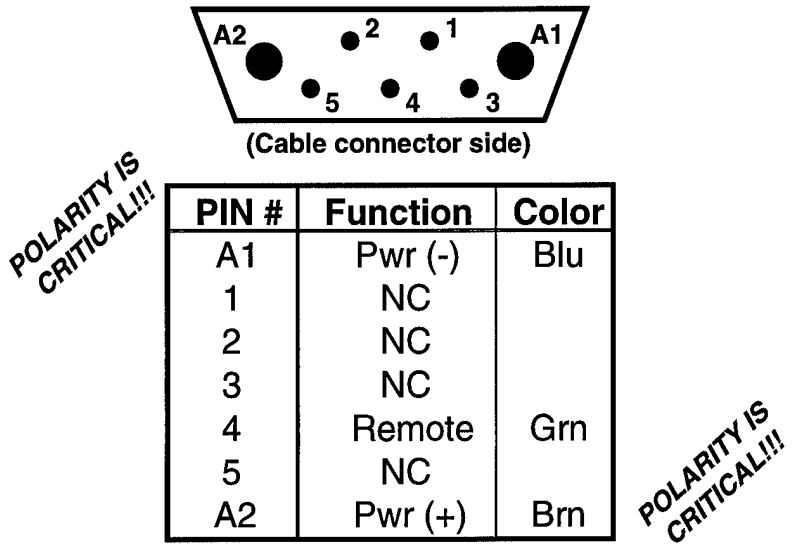
As the C&DH subsystem was defined and implemented before the remainder of the system, once the C&DH design was finalized, it effectively defined some new constraints for the interfaces for the on-going *SIMSAT* design. Since this was the final design phase for the C&DH subsystem (the next chapter documents the implementation of the system), this section documents the interfaces the rest of *SIMSAT* must now support. As a result of selecting the dSPACE/AutoBox solution, the rest of the system was now constrained to support the following interfaces:

Subsystem	Final Choice
Attitude Determination & Control	Momentum Wheels
Power Generation & Distribution	No Clear Choice
Command & Data Handling/ Communications	AutoBox On
Structural Support	No Clear Choice

Table 4.9 *SIMSAT* Subsystem Choices

1. **Power:**
   - 8–36VDC; overvoltage protected to 100V, but >36VDC not recommended for long-term usage [10:3]
   - 10–20A startup current (approximate startup power for <1 sec is about 240W; higher current for lower voltage) [10:3]
   - 60W steady-state power draw (empirically derived for setup with initial card complement)
   - 6 mm^2 minimum diameter wire due to initial current demand [10:1]
   - proprietary power interface pinout shown in Figure 4.17 [10:2]
     - install an in-line switch on the positive supply line so the power supply can reach full power before AutoBox tries to come on-line (otherwise AutoBox may prevent power supply from reaching nominal operating condition due to the high AutoBox startup current demands)
     - connect pins A2 (power) and 4 (remote sensing) to the switched positive power supply connection

2. **Physical:** Figure 4.18 provides an isometric drawing of the AutoBox to define the physical envelope it occupies, as well as the bolt pattern for mounting it on the satellite



PIN #	Function	Color
A1	Pwr (-)	Blu
1	NC	
2	NC	
3	NC	
4	Remote	Grn
5	NC	
A2	Pwr (+)	Brn

Figure 4.17 AutoBox Power Cable Interface

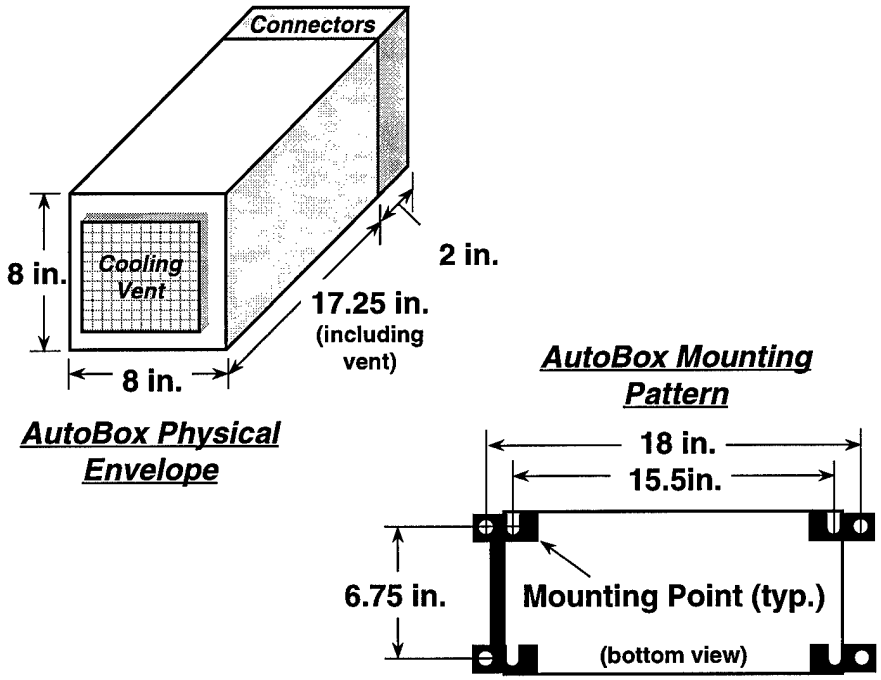


Figure 4.18 AutoBox Dimensions

3. **Signal:**

- 32 in (from the sensors); on-satellite termination panel along with associated cabling fabricated to interface with the DS2003 [12] card using the dSPACE-provided connector and protective shell.
- 32 out (to the effectors); on-satellite termination panel along with associated cabling fabricated to interface with the DS2103 [14] card using the dSPACE-provided connector and protective shell.
- all I/O signals are analog voltages to represent levels; possible range of voltages is ± 5 or ± 10 volts
- ethernet connection (10 Mbps) between the Simulation PC and AutoBox
- any cards to be installed in the AutoBox must use the ISA interface
- 2 serial lines (RS-422-A high density sub-D connector) between the RealMotion PC and AutoBox

V. Implementation

5.1 Overview

This chapter will document the implementation of the C&DH system, including hardware and software installation issues and system validation. And, while not included in this chapter, the Operator's Manual (Appendix D) is a natural extension of this chapter, taking the procedures used to validate the system installation in this chapter, and transformed them into the steps required to use the system: how to start the AutoBox, which icons to click on, and the dialog box inputs (and their rationale) when the required "inputs" may be unclear.

While this chapter represents a major shift in focus from design to implementation, it also represents another shift in the relationship between this design effort and that of the rest of the team. Since most, if not all, of the design decisions have been made for the C&DH subsystem, the "perceptual level" [38:606] this effort considers is now clearly separate from the rest of the *SIMSAT* design effort. Rather than using words to define this shift, consider Figure 5.1.

Levels	Supra-System (M)	System (S)	Sub-System (P)
A	Environment		
B	*Unit	Environment	
C	Sub-Unit	*Unit	Environment
D		Sub-Unit	*Unit
E			Sub-Unit

Figure 5.1 Perception Levels [38]

At this stage of the design effort, the overall *SIMSAT* effort is considered to be at the *Supra-System* level, while the C&DH subsystem could now be considered a system onto itself without any loss in SE process integrity. The C&DH *Unit* focus (level *C*) is clearly a *Sub-Unit* of *SIMSAT*, which is at level *B*. In addition, the rest of *SIMSAT* efffectively can be treated as *Environment* from level *C* because the interfaces between C&DH and *SIMSAT* have now been formally been defined. While the C&DH subsystem had been treated as an integral part of the total design effort until now, it effectively became a pre-defined, constrained subsystem while the rest of the *SIMSAT* design effort progressed into *Detailed Design.* This distinction was important because the focus of the problem-solving process for the two design efforts diverged significantly at this stage of development.

This portion of the system life-cycle documents the steps required to physically manifest the system designed in previous life-cycle phases. Until this phase, the application of the activity matrix has been clear—there have been obvious issues to be resolved. In the *Implementation/Operation* phase for a single subsystem, trying to use the same problem-solving framework became less clear. In fact, initial consideration of the aspects of the life-cycle documented in this chapter (Figure 5.2) seemed to indicate the process might even be inappropriate.

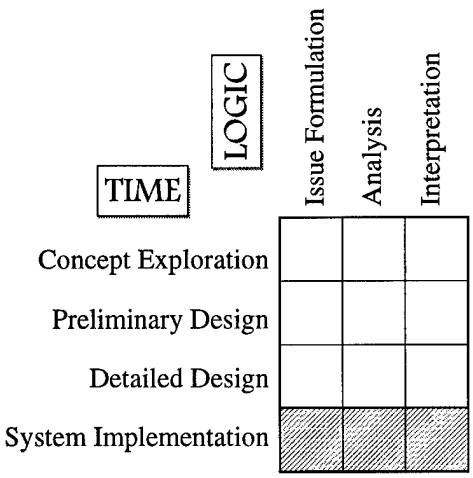


Figure 5.2 Implementation Phase Activity Matrix

Upon further consideration, the team realized the framework would be the basis for resolving any problems that might arise during *Implementation*—the actual steps required for implementation and operation would be guided by the needs of the customer (system requirements) and values hierarchy (what the CDM considered important). Using these as a foundation, the team implemented the C&DH system, and informally applied the remainder of the problem-solving steps any time a decision had to be made about implementation or operation details[1]. Because these issues were varied and insignificant compared to the systems issues addressed in previous chapters, each one will not be documented separately. The only steps formally readdressed for this phase were *Issue Formulation* (specifically, the *Problem Statement*, *Problem Elements*, and *Value System Design*), and *Interpretation* (specifically, *Plan for the Next Phase*). The first of these problem-solving steps will be documented in a section of this chapter, followed by sections for *Implementation* and *Operation.* The *Plan for the Next Phase* step will be documented in the next chapter (*Future Work*).

5.2 Issue Formulation

5.2.1 Problem Statement. To capture the change in emphasis from design issues to implementation and operational issues, the problem statement became:

> *Considering the satellite system simulator is to be used as an experimental test bed for Air Force Institute of Technology (AFIT), implement the* DSPACE *system so that it meets the original* SIMSAT *design requirements, maximizing as many of the previously defined design goals to support Air Force/DoD research, and provide a sound visual aid to AFIT instructors teaching satellite control and dynamics.*

5.2.2 Problem Elements. For the implementation of the C&DH subsystem, the focus turned to preparing the system for integration into the rest of the system, and making it ready for development work. The following list of *Problem Elements* reflects that shift in focus:

[1] CDM-approved due to the time-constraints of the design effort

- User Interface Design
- Development Work "Bench"
- Signal and Power cable layouts

5.2.3 Value System Design. As mentioned before, the problems faced during this life-cycle phase were minor compared to those encountered earlier in the system life-cycle. Rather than trying to update the values hierarchy everytime a problem had to be resolved during implementation of the C&DH subsystem, the team made decisions base upon the "spirit" of the values hierarchy used in *Detailed Design* phase, effectively eliminating the specific measures of the previous hierarchy while retaining the fundamental values and the first tier of evaluation considerations. Combined with their previously assigned weights, the more abstract hierarchy formed the basis for making good decisions (from the perspective of satisfying the CDM) during this life-cycle phase. The intent of the values hierarchy at this stage was to provide a "values focus" to the decisions being made, as opposed to quantitatively comparing alternatives. The abstract values hierachy is shown in Figure 5.3.

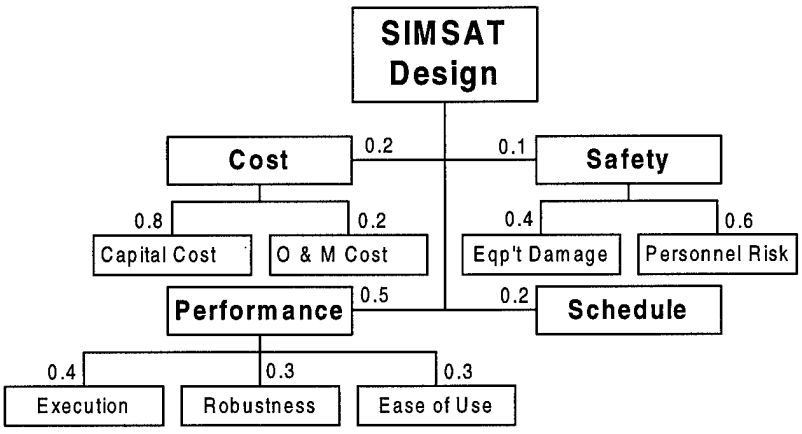


Figure 5.3 *SIMSAT* Abstract Value Hierarchy

5.3 Subsystem Installation

Once the framework was established to guide the C&DH subsystem implementation, the user manuals for the hardware and software were consulted to install the DSPACE system. Because the DSPACE system is as integrated and well-defined as it is, no significant decisions had to be made. The only issue that had to be resolved was a "test bench" power source to power the AutoBox prior to installation on the satellite. Using the notional values hierarchy in Figure 5.3, the desirable attributes established for the "right" power supply were:

- portable; rack-mounted preferred
- variable voltage:
  - AutoBox can function in the 8–36VDC range
  - ADACS motors need higher voltage to reach higher torque values
  - higher voltage requires lower current (AutoBox DC-DC power converter uses constant 60W)
- simulate satellite power bus as close as possible (300-400W)

Fortunately just such a power supply was found within the AFIT inventory and was used to power the AutoBox for system installation and validation. The first part of Appendix D provides a consolidated look at the procedures used to implement and validate the DSPACE system.

In accordance with the original desire to be user-friendly and adaptable, the Simulation PC and power supply were installed in a roll-around cart, along with a pair of DSPACE connection panels. These connection panels allows signals to go between the AutoBox and whatever experiment it might be controlling[2].

After the installation was validated, all the integration work that could be done with the DSPACE system was complete. Some Windows95 shortcuts were developed to ease the

[2]This installation decision provides capability beyond that strictly required for *SIMSAT*—this experimental control system can now be moved around the building and used for other facilities as deemed necessary by AFIT.

future developer's workload and are documented in Appendix D, beginning on page D-35. The latter half of that appendix also includes an operator's manual with detailed step-by-step instructions for implementing SIMULINK/DSPACE models, beginning on page D-26.

5.4 SIMSAT *Integration Issues*

Since this design effort led the rest of the *SIMSAT* design effort, DSPACE integration with the rest of the system could not take place. The issues in this category were intentionally left for the team to take care of during the final system integration. Design constraints for the rest of the subsystems to adequately support the C&DH subsystem were identified in Section 4.5, page 4-42. Once the rest of the *SIMSAT* subsystems are ready for DSPACE integration, the following issues need to be addressed:

5.4.1 Wireless System Integration. Mentioned above, this integration issue was borne out of the initial division of subsystems that separated the communications and C&DH subsystem. It is such a critical integration issue, it deserved specific mention. Design constraints for the communication subsystem to adequately support the C&DH subsystem were identified in Section 4.5, page 4-43. Some of the issues that need to be addressed include how to load drivers into the AutoBox, or how to install an "ethernet modem" instead, to take the signal from the existing ethernet cards and just modulate them for transmission to the ground. The impacts of the additional overhead need to be carefully considered.

5.4.2 Connector Panels. For *SIMSAT*, more compact connector boards need to be fabricated for interfacing with the DSPACE I/O boards. The connectors on those boards will need to be smaller than the BNC connectors on the DSPACE connection panels to reduce space and weight. The team will also need to fabricated the cabling to run between AutoBox and the on-satellite connector boards. These cables will need to be terminated the DSPACE-provided high density connectors and protective shells to allow connection to the AutoBox. Sensor/effector location and wiring modularity will also need

to be carefully planned to provide additional signal routing flexibility to and from the AutoBox.

5.4.3 Recording Data. While the original intent of the research effort may not have intended to develop the ability to record and replay data, that goal appeared feasible. Unfortunately, time did not permit determining how DSPACE implements data acquisition. While no mechanism was immediately apparent for retaining generated data in *Trace*, *Cockpit*, or *RealMotion*. Based upon the mixed levels of user-friendliness within the DSPACE system, the solution could be straight-forward or could be convoluted. But the dSPACE, Inc. tech support has always managed to provide answers in short order.

5.5 Summary

While the brevity of this chapter does not do justice to the time spent integrating this subsystem, the vast majority of the work was just following the "cookbook"—if you knew which book and page to use. To reduce the headaches and missteps for anyone that wants to duplicate the AFIT *SIMSAT* installation, the first half of Appendix D provides a detailed procedure for how the system was implemented and validated. The steps listed there reflect updates to the installation procedures in the hardware and software manuals based upon the original installation experience.

VI. Research Summary and Recommendations

6.1 Overview

This chapter concludes the body of this document by describing, at a top level, the results and products of this research effort, followed by some recommended research extensions.

6.2 Research Summary

This research effort went through four iterations, or life-cycle phases, of a systematic problem-solving process in search of the best implementation of the Command and Data Handling (C&DH) subsystem for the AFIT Simulation Satellite (*SIMSAT*). The research detailed in this document was only the first part of a team effort, the conclusion of which will be documented in [8]. This section summarizes the development of the team's Systems Engineering process, and the research results from each of those life-cycle phases.

6.2.1 System Engineering Process. As recommended by Sage [49:67–68], the team considered some of the potential SE processes documented in the literature and decided that a modified Sage/Hall process would be best for the application at hand. Once the *Knowledge* dimension of Hall's 3-D Morphology was defined (see Figure 1.4, page 1-14), the SE process boils down to an "activity matrix" [49:47] that the disciplines iterate through. After considering the factors recommended by Sage [49:68], the team developed the activity matrix in Figure 6.1 to guide their activities.

As the team moved down the rows in the activity matrix, the level of detail for their design activities increased. The next section summarizes the shifting levels of detail by describing the results produced during each life-cycle phase.

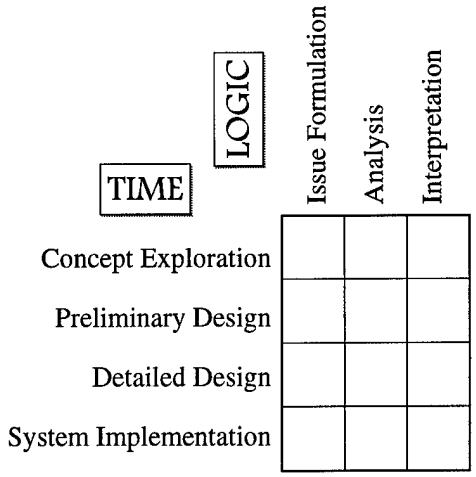


Figure 6.1 *SIMSAT* Team Activity Matrix

6.2.2 Life-Cycle Results. The evaluation of C&DH alternatives in each life-cycle phase considered total system impacts of the alternatives, as opposed to just assessing the alternatives in isolation. This prevented developing "optimal" subsystem solutions that would have actually led to sub-optimal system-level solutions.

During *Concept Exploration*, classes of solutions for the C&DH subsystem were qualitatively evaluated in Chapter II against a relatively abstract values hierarchy (Figure 2.3, page 2-13). That assessment determined that a *Graphical, Digital Computer Assisted Control System* would provide the best value to the CDM. The other alternatives were either too arcane, too complicated, or could not provide the capabilites the user desired.

The *Preliminary Design* life-cycle phase (Chapter III) conducted a trade study of some of the *Graphical Digital Computer* solutions (comparable to the DSPACE solution[1]) commercially available toady. The DSPACE solution turned out to be the best alternative in that study. While cost was one of the main drivers for that outcome, other aspects of the competing alternatives eliminated them from consideration. The performance impacts and lack of "user-friendly" attributes of the non-integrated solution, coupled with the time it would take to integrate and validate the subsystem, made it clearly the worst

[1]The DSPACE system had been partially purchased before this research was initiated. That effectively established it as a "no-cost" baseline against which the other two alternatives were compared.

alternative. The only other truly integrated solution found during the market survey was intended for developing flight-qualified software in a corporate environment (i.e., an army of programmers working to implement a large, complicated control system). While extremely robust and flexible, the MATRIX$_X$ solution was totally unsuitable for the user friendly, "rapid prototyping" type of development environment needed in this facility.

Chapter IV describes the *Detailed Design* steps used to determine how best to implement the software architecture within the DSPACE framework. As more detailed information could be gathered for this life-cycle phase (enough of the system was sufficiently defined to know what to look at), quantitative analysis tools were used to compare the alternatives. Hosting all the control software on the satellite, using the AutoBox, turned out to be the best software architecture, largely because of the *Performance* impacts of the other alternatives. As most of those measures were somewhat qualitative (using the constructed scales documented in Appendix C), there was some non-trivial risk in that choice. However, since the next two alternatives do not require substantial time or money to implement, the impact of improperly estimating the value of those measures was very limited, while the potential benefit was significant.

As the DSPACE solution (specifically with AutoBox) established a firm interface between the C&DH subsystem and the rest of *SIMSAT*, the C&DH was largely treated as an independent subsystem (while still maintaining focus on the CDM's values) in the *Implementation* life-cycle phase. Since all the design decisions had been made by this point, this phase integrated all the C&DH subsystem components and validated the installed DSPACE system. While the installation discussed in Chapter V, and documented in the first part of Appendix D, effectively followed the rout installation procedures provided in the various manuals, the team remained focused on the CDM's values whenever implementation decisions had to be made (i.e., computer configurations, software shortcuts, power supply selection, etc.). Some guidelines for system integration were also noted at the end of this life-cycle phase to support C&DH integration with the rest of the system.

As smooth as this summary makes the process sound, there were some tertiary issues that had to be addressed during the design effort.

6.2.3 Lessons Learned.

Several things occured during this effort that, had the team anticipated, would have saved them some aggravation.

The first problem the team encountered related to the way the system was functionally decomposed. Their lack of practical experience in the design of complex systems (particularly with respect to subsystem interactions, and how those interactions should factor into the system decomposition) caused them to overlook the critical inter-relationship between the C&DH and communications subsystems. The arbitrary decision to split those two systems appeared harmless during *Concept Exploration.* But it eventually became clear the communication system was an integral part of the C&DH system, and should have been included in this portion of the overall design effort to avoid having to revalidate the system after converting it from a wired to a wireless communication system. Based upon the compatibility problems that can occur between network cards[2], this could become a major integration issue.

The next major problem the team experienced was also related to integration issues. Faced with a short deadline to order subsystem components before the end of the fiscal year, thorough interface studies were not completed before the remaining DSPACE components and some motors were ordered. Only upon delivery did it become clear that the motors, as delivered, would not work in conjunction with the DSPACE system configuration required for other aspects of the *SIMSAT* system. New motors have since been identified, but were not available for integration testing prior to the completion of this design effort.

6.3 Research Extensions

While the common adage tells us that "the work is not finished until the paperwork is done," research of any meaning is rarely ever complete—there are always things that could be undertaken as extensions to the "finished" work. Such is the case here—the following sections address several different types of research extensions more completely. There are at least four areas of research that could be supported by *SIMSAT* and could directly

[2]The entire operating system and development environment had to be reinstalled on the Simulation PC after trying to get two different network cards to work together.

improve satellite research and education efforts. Thge first three areas are concerned with the schedulability of task sets, while the fourth is concerned with upgrading the test facility.

6.3.1 Scheduling Analysis Tools. The first recommended extension to this research involves the development an analysis tool to estimate the ability of a task set to meet its deadlines on the TI C40 digital signal processor (DSP) prior to loading and executing it. dSPACE has done quite a bit of work estimating the amount of time required for a variety of overhead tasks. Some system lagtimes and overhead time requirements include:

Task	Time (ns)	Source
Context Switch	6,000–12,000	[19:71]
Mux Settle Time	333	[12:29]
Analog Settle Time	10,000	[14:24]
ADC Convert Time	7,000 (1)–75,000 (32ch)	[18:67]
DAC Convert Time	2,500 (1)–24,000 (32ch)	[18:74]

Building upon efforts such as [5] would allow experimenters to determine if a given task set is schedulable prior to actually executing the task set and would limit the amount of ad-hoc "code-and-test" steps required to develop a valid set of control laws.

But if the tool (or the dSPACE system) indicates the task set is unschedulable, what are the options? There are two main ways to solve the problem: increase the processing power or optimize the software. Each option will be addressed in next two recommended research extensions.

6.3.2 Distributed Control. To improve the processing power of the dSPACE system, one of three different techniques can be chosen: multiple C40's networked together, a quad-processor version of the C40 DSP, and a 400MHz DEC Alpha-based co-processor card. Each one of these turns out to be a type of distributed control; several considerations factor into the decision of which is the best solution.

There are really two aspects to this proposed extension: the increase in the size of control laws that can be executed using multiple, cooperative DSPs and determining the impact of using a wireless ethernet connection between those processors (as opposed to a direct, Proprietary High-Speed [PHS] bus connection) if part of the processing is done on the ground and part on the satellite. While related, the first is trying to increase the schedulable task set, while the second is intended to determine if the overhead associated with a wireless connection is too great a penalty to allow separation of tasks. In particular, the latter effort is intended to investigate the feasibility of separating the inner and outer control loops of the *SIMSAT* system so that more elaborate experiments can be run on the satellite. Each will now be addressed independently.

6.3.2.1 Enhanced Task Set Processing. If the control system can be logically partitioned into completely independent task sets, the multi-processor board might be a viable option. But, the greater the dependency between tasks residing on different processors, the more communication overhead there will be, increasing the blocking that will occur (see Figure A.3, page A-17 and the accompanying paragraphs for more insight into this issue). In addition, the farther apart the processors are, the greater the communication overhead will be due to the per-transaction transmission delay. In some specialized cases, the Quad C40 processor board would help reduce the per-transaction penalty, but any appreciable dependencies between the processor task sets would still impact the speedup of the multi-processor configuration.

Since a quad-processor solution cannot provide quadruple speed-up (with the very rare exception of 4 independent, equal loading task sets), conquering most schedulability problems can be accomplished by increasing the raw speed of the main processor (as with the Alpha solution). This is not contrary to the "increased speed does not mean better control" axiom previously mentioned—the context of that misconception was in comparing general and real-time computing. Once a real-time operating system (RTOS) is used to ensure predictability and prevent events (i.e., deadlock, uncontrolled priority inversion, etc.) from threatening the completion of time-constrained tasks, speed is a good thing. Since the SEMOS that comes from dSPACE, Inc. is a true RTOS, the speed available

from the Alpha co-processor (or the multi-processor board) could help solve scheduling problems.

6.3.2.2 Impact of Wireless Ethernet. The other related extension to this research is the use of the *SIMSAT* computer architecture to investigate/quantify the impact of splitting the control laws across a wireless ethernet connection. This is a short-coming of the data available from DSPACE—there is no data available addresing the amount of overhead involved in communicating over such a link. In fact, no data was even found for a wired ethernet connection. This would be important data to have to determine if a physically separated multiprocessor environment can provide ANY speedup, or if the communications penalty is too great. This additional data can also be factored into the analysis to help the experimenter make appropriate choices about how to alloact the control system task(s).

6.3.3 Optimizing Program Code. In case the analysis tools indicate the DSPACE compiled code is not schedulable, and multi-processor (or Alpha-based) operation is not feasible, another option is to develop optimization tools for the C-code generated by the DSPACE software. Automated generation tools are notorious for generating sub-optimum code due to their need to support a number of different uses and execution platforms. Since most of the code generated for *SIMSAT* and its experiments will be very similar, there may be some overhead code that could be stripped out of the "generic" DSPACE C code, resulting in optimized "*SIMSAT*-C" code that might be schedulable. Depending upon the techniques required to implement this "tool," the results of the research extension could be an automated tool that would read the generic code and produce the optimized code automatically, or it could just be a set of guidelines for a programmer to use based upon the particular control system being implemented. Either one would be useful, with the automated tool being the preferred solution.

6.3.4 Test Bench Fabrication. In Section D.4, page D-35, an experimentation philosophy was described. To adequately support that philosophy, it would be advantageous to develop a modular test bench "docking station" that would allow easy

installation of the AutoBox, ADACS, and experiment components on a test bench. The docking station would duplicate the *SIMSAT* signal and power interfaces (including a power system that would duplicate the *SIMSAT* power bus), mounting points and protective shields for devices that rotate or move, and high-fidelity simulated hardware (such as sensors) to allow for full hardware-in-the-loop testing prior to operating payloads on the satellite.

6.4 Summary

This design effort effectively developed and implemented a Systems Engineering process to design and implement the C&DH subsystem for the AFIT *SIMSAT*. But issues remain for follow-on *SIMSAT* integration and research extensions. While not an exhaustive list of potential extensions to this research, the extensions listed in this chapter represent those that were expected to have the most potential of significant return for AFIT/ENY. But other departments may also be able to benefit from some of the potential research topics the *SIMSAT* test platform, or its subsystems (even in a test bench setup), could provide.

Appendix A. Real-Time Control Issues

A.1 Overview

The intent of this appendix is to justify the selection of the Rate Monotonic scheduling technique as the most analytically sound technique in the literature today and describe the potential analytic power of the technique, assuming all operating system, hardware, and development tools support it. Unfortunately, no system on the market manages that level of support, so the enhancements still required were listed in Chapter VI.

As mentioned in Chapter I, computerized control systems (among others) fall into two types of "real-time" systems: hard and soft. "Hard" real-time control systems require accurate results from all necessary computations before some well-defined deadline, *guaranteed.* For example, if the tasks that define the behavior of a computerized flight control system miss deadlines, the aircraft could become uncontrollable. For "hard" real-time control systems, accurate results *must* be computed before some pre-defined, application-specific deadline is reached. For this type of system, a logically correct answer is useless if it is produced too late for the system to respond to user inputs or maintain system stability.

SIMSAT can be characterized as a "hard" real-time system: if the control system fails to meet its deadlines, the satellite could become unstable. The satellite could begin spinning out of control, or the control system might miss some of its sensing deadlines and not realize it needs to command a momentum wheel to slow, stop, or reverse. The latter case could theoretically cause one of the Momentum Wheels to rotate faster than the structural limits of its wheels, causing it to literally "throw" itself apart.

By the same token, the C&DH subsystem on *SIMSAT* needs to be as flexible as the rest of the subsystems—it should be able to support a variety of experiments, and be easily adapted to new experiments. It needs to allow the user to quickly compile, load, and run new control laws. And before actually operating the system, the development environment should give clear indications if the control laws, as programmed, will act "correctly"—logically and temporally.

Many development environments provide ways to simulate the system, but this is a limited way to validate a system—you can never do enough testing to *prove* a task set is temporally sound. Proving a system is sound can only be done analytically. And any such analysis assumes complete knowledge and quantification of the environment the system operates in as well as how the system itself functions. While thorough investigation might yield sufficient information about task periods, execution times, etc., in most systems there are certain stochastic (and even chaotic) behaviors that will wreak havoc on many control systems. Some scheduling algorithms can catastrophically fail when overload conditons occur while others may just miss deadlines. On the other hand, some scheduling techniques can tell you which deadlines will be missed, and which deadlines will be met, no matter how significant the overload. All these issues are key to being able to implement a systematic design and conduct technically sound analysis to *analytically guarantee* the system will act the way it should (given sufficient data about the system and environment).

This appendix begins by describing different preemptive scheduling techniques for determining the "schedulability" of a program containing "hard" real-time tasks. That is followed by an explanation of the basic RMS algorithm—an "optimal" scheduling technique that can be used to analytically validate the schedulability of any task set. RMS also has the desirable property of graceful degradation of system control (if it degrades at all) during overload conditions. After a summary of some of the extensions to the basic RMS algorithm, this review will illustrate what is to be gained from using the extensions for single and multi-processor real-time systems. In general, each section in this review is dedicated to the one main paper. Unless otherwise noted, all statements, examples, and data in that section are drawn from the main paper of that section.

To provide a context for RMS, the first paper [43] covers several different scheduling techniques. Since the focus of this review is the RMS family of techniques, the next paper summarized and evaluated is the classic RMS paper by Liu and Layland [39]. Subsequent papers examined in this review will describe some recent efforts to extend the restrictive academic theories of Liu and Layland into a more usable suite of scheduling tools. The Software Engineering Institute (SEI) prepared a tutorial [52] on the basic RMS technique and then outlined an anthology of RMS extensions dealing with relaxing the utilization

bound, aperiodic tasks, and task synchronization (a mechanism to allow interdependent tasks to work cooperatively). The fourth set of papers presents some of the scheduling complexities introduced by a network of interdependent processors.

A.2 Real-Time Scheduling

The number of techniques available to try and schedule a set of tasks to meet its deadlines is almost as numerous as scheduling theory authors. However, as C. D. Locke observes in [43], all the different priority-based scheduling techniques can be grouped into five major categories:

- Cyclic Executive
- Fixed Priority
- Shortest Process Time
- Earliest Deadline
- Shortest Slack Time

Each of these algorithms take their name from the technique used to determine the order of task execution. The first two of these techniques are of most interest to this review. The *Cyclic Executive* algorithm is of interest because it is the predominant scheduling technique used in real-time systems today [4:120]. The family of *Fixed Priority* algorithms is of interest because it includes the Rate Monotonic Scheduling technique, which is the primary focus of this review.

The remaining three algorithms are not useful for most real-time applications. The *Shortest Process Time* algorithm is unable to limit "lateness" to zero or negative values (i.e., meeting or beating deadlines). The *Earliest Deadline* technique will fail unpredictably when the processor becomes overloaded (i.e., cannot meet all the deadlines). And since there is a 100% certainty that transient overloading will occur at some time in nearly every real-time system, the likelihood of unpredictable system failure prevents the use of *Earliest Deadline* scheduling for any critical control systems.

The final algorithm, *Shortest Slack Time*, is possibly the most insidious. If the impact of system overhead is ignored, the technique appears very attractive: it has the advantage

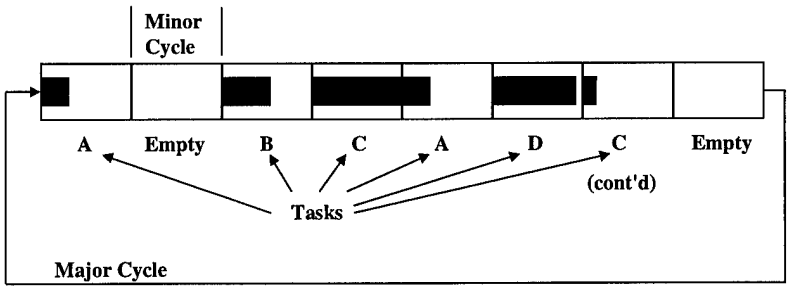


Figure A.1 Cyclic Executive Timeline

of providing an "optimal" schedule: "if a process set is feasibly schedulable, [this] scheduler will produce a successful schedule" [43]. This algorithm behaves as if it is waiting until a task is as close to its deadline as possible before beginning execution (zero slack-time). But since system tasks that need to execute will typically preempt any other executing task, a zero slack-time user task would miss its deadline. The results of this missed deadline could be "disastrous." Therefore, to preclude catastrophic failures, system preemption must be restricted. However, limiting system preemptions will produce unwanted side effects in handling other timing limitations. For these reasons, this algorithm is rarely used.

Since these three techniques are not useful for real-time systems, nothing further will be mentioned about them in this overview. The remainder of this section will be spent on the *Cyclic Executive* and *Fixed Priority* techniques.

A.2.1 Cyclic Executive. The *Cyclic Executive* appears to be a straightforward scheduling technique, and is one of the most common scheduling techniques in use today [4:120]. Its basic premise is that a program is made up of a set of tasks that will repeatedly execute (see Figure A.1). Each time this set of tasks starts to execute, a new "Major Cycle" begins. The major cycle will recur at a periodic rate, typically initiated by a hardware timer.

The major cycle is then divided into a number of equal sized *frames*, called "Minor Cycles." A block of periodic, non-preemptible statements (a "task") is then assigned to each minor cycle. Frame size is dictated by the task with the longest execution time in

the major cycle. To ensure all deadlines will be met, each task must finish its execution within its assigned frame.

If the task does not complete within its allotted time, *frame overrun* occurs. There are traditionally two ways to deal with this common error condition, but neither is very acceptable: abort the errant task, or allow the entire timeline to slip. Either "solution" will likely cause unexpected results, appearing to the user as non-repeatable system failures. These "intermittent faults" may be indistinguishable from transient hardware failures, such as a bad connection or failing logic device.

But if frame overruns can be avoided, a real-time system scheduled with this technique can be guaranteed to meet all its deadlines since the timing requirements are handled as the system is designed. Unfortunately this technique creates an inexorable connection between the logical and temporal correctness of the design. Tying these two distinct issues together creates an unnecessary and undesirable coupling of concerns: if some hardware or software changes force modification of the task code, the entire time-line will need to be reevaluated to ensure the temporal correctness has not been sacrificed. For example, the task that uses the most computation time determines the size of the minor cycle or "frame." If some change to the code for that task further increases the time it takes that task to execute, some adjustment to the timeline will be required: either the task will need to use more than one of the currently empty minor frames, or the minor frame must be enlarged. The latter option is less attractive since it also forces an revaluation of the major cycle. But if no free minor frames exist, the entire timing scheme will need to be redone.

Requiring programmers to piece code segments together to hand-tune the time-line of task execution is analogous to forcing them to go back to manual page overlays for memory management. As one of the major tenets of software engineering is information hiding to avoid this type of modification "ripple," finding a temporal version of encapsulation is very important. Establishing a separation between logical and temporal concerns would provide such an analog and would allow the application of disciplined engineering principles to the entire software development process, improving the maintainability of the system. [52:1,2]

A.2.2 Fixed Priority.

This class of techniques assigns a priority to each task based upon some "application-specific criteria." This priority will, in general, not change during execution. This class of techniques also tends to be preemptive in nature: when a task is ready to execute, it will interrupt any lower priority task that may be executing. With this technique it is obvious the highest priority task would meet its deadlines, but schedulability determination for lower priority tasks would be, in general, more difficult. This technique, therefore, will provide more graceful degradation (than the cyclic executive solution) during overload conditions.

The reason for difficulty in determining low priority task schedulability is related to non-deterministic order of execution inherent in multi-tasking control systems. For example, assume a low priority task, τ_l, is prevented from starting execution until:

$$t = t_{deadline_l} - t_{task_l.execution.time}$$

(this extreme case may or may not occur). But during this critical execution period for τ_l, high priority task τ_h becomes ready to execute and preempts τ_l. No matter how little time τ_h requires for execution, τ_l will not be able to meet its deadline.

The inability to guarantee schedulability when using this technique, coupled with the difficulty of dealing with non-deterministic task execution order, prevented frequent use of this type of algorithm until Liu and Layland published their landmark paper detailing the Rate Monotonic Scheduling theory [39]. As will be seen, their algorithm is an "optimal" fixed priority, preemptive scheduling technique that *CAN* guarantee whether a set of tasks will meet its deadlines or not, and provides a set of tools to help intuitively conceptualize task set execution behavior.

A.3 Rate Monotonic Scheduling (RMS)

In their classic scheduling theory paper [39], Liu and Layland recognized the lack of adequate formal techniques for determining whether a set of tasks could co-exist on a single processor and still meet all required deadlines. Their paper intended to provide the practicing computer scientist a set of formal analytical tools to answer the schedulability question prior to implementation.

A.3.1 Assumptions.

In developing their model, Liu and Layland made some simplifying assumptions as to how tasks are implemented, how those tasks execute, and how they interact. Their simplifying assumptions include:

1. Tasks only execute when they are "requested" and task requests are periodic
2. Tasks must be complete before the next request is acted upon
3. Tasks execute independent of other tasks
4. Tasks require the same amount of time to complete every time they execute
5. Aperiodic tasks are "special" and displace the normal set of tasks as "special" needs arise (fault recovery, etc.)

Since Liu and Layland sought to develop a quantitative approach to determining the schedulability of a task set, they needed to define the salient characteristics of a given task. Liu and Layland designated the amount of time a task, τ_i, requires for execution as C_i. And since the above simplifications assume periodic tasks, Liu and Layland assumed task τ_i has a consistent period, designated T_i. Liu and Layland contended (and later proved) that, based upon the above assumptions, a given task τ_i could be completely characterized by its corresponding C_i (Liu and Layland suggest using the maximum observed execution time as the estimate)and T_i.

A.3.2 Static Priority Scheduling.

Once they defined the necessary variables, Liu and Layland developed an "*Optimum*" *Static Priority Scheduling Algorithm* by proving a number of theorems. Defining each of these italicized terms will provide an overview of their algorithm.

By "*Optimum*," Liu and Layland meant that if any fixed/static priority scheduling algorithm could schedule a task set, their RMS algorithm could also determine a way to successfully schedule that task set. In other words, if RMS cannot determine a scheduling assignment to allow a set of tasks to meet their deadlines, no other fixed priority scheduling algorithm can either.

A *Scheduling Algorithm* is a technique for determining which task should execute at any given moment. The algorithm Liu and Layland described assigns a priority to a task prior to program execution, and maintains the priority during execution (i.e., the *Priority*

is *Static*). The algorithm determines task priority based exclusively upon the period (T_i) of task τ_i. As the task period decreases, the *Rate*, or frequency, of task execution increases. The more frequently a task must execute, the higher the assigned priority. The assigned priority, then, increases "*MONOTONIC*ally" as a function of the rate of task execution. Hence the name Liu and Layland assigned to this algorithm: *Rate Monotonic Scheduling.*

A.3.3 Schedulability Determination. To determine task set schedulability using the RMS algorithm, Liu and Layland defined processor utilization as the sum of $U(i) = \frac{C_i}{T_i}$ for all tasks, τ_i, assigned to a processor. They went on to prove the foundational theorem for RMS schedulability:

> **THEOREM 5:** For a set of m tasks with fixed priority order [of execution, determined by the RMS algorithm], the least upper bound to processor utilization is $m(2^{1/m} - 1)$.

This theorem states that for a large task set (m tasks $\rightarrow \infty$), the bound for processor utilization is $\ln(2) = 0.6931$. This result states that, ignoring all other factors, if the total processor utilization of a given task set remains below 69.31%, the task set will meet all its deadlines. This algorithm has the added benefit that, even if a set of m tasks is unschedulable, as long as the cumulative utilization factor for the first k of those m tasks is less than $k(2^{1/k} - 1)$, those k tasks will meet all their deadlines. For this reason, RMS is called a "stable algorithm," with the set of k tasks called the *Stable Set.*

While Liu and Layland provided the basis for calculating the ultimate processor utilization limit, they generally left the impact of smaller task sets to the reader's imagination. For a smaller task set, the limit to processor utilization is less restrictive, but rapidly approaches the asymptotic limit of $\ln(2)$:

TASK SET SIZE	UTILIZATION BOUND
1	1.0
2	0.8284
3	0.7798
6	0.7348
16	0.7084
32	0.7007
64	0.6969
128	0.6950

Even with all these benefits, RMS did not find much of an audience initially. Part of the acceptance problem was the overly restrictive 69.31% processor utilization bound for large task sets. In addition to this problem, assuming a task set of only independent, periodic tasks reduces the general applicability of the model (few, if any, applications have only periodic tasks). Liu and Layland realized their 'periodic task' assumption was one of the most "indefensible" they made, but gave no technique for overcoming this major flaw. The next paper reviewed will provide several techniques to deal with these (and other) limitations.

In summary, RMS is built around a set of simple equations for determining task priorities and whether a set of prioritized tasks will meet its deadlines or not. If RMS indicates that all deadlines cannot be guaranteed, the equations (and the above chart) indicate which set of task deadlines *can* be guaranteed. This *Stable Set* of tasks $(\tau_1 \ldots \tau_k)$ will have a total utilization lower than the bound defined by $k(2^{1/k}-1)$. This characteristic is distinctive: in other unstable scheduling algorithms, if one deadline is missed, the rest of the deadlines may not be met. Therefore, *RMS* provides for more graceful degradation during overload, and the set of tasks that will likely fail is well-defined.

A.3.4 Dynamic Scheduling. This same paper included a technique to support dynamic priority assignment, called the *Deadline-Driven Scheduling Algorithm* [39:55]. If this technique can be shown to be as easy as the static RMS technique, it would seem, at first glance, to be preferrable. However, the literature seems to indicate that commercial developers prefer static priority scheduling techniques for the following reasons [53:3]:

1. In practice, the difference in performance is minimal.
2. While potentially less efficient, static priority scheduling techniques tend to be more stable in overload conditions than dynamic techniques.

Since this technique is essentially an "Earliest Deadline First" scheduling technique (which has already been shown to fail unpredictably when overload conditions occur [43:52]), this technique has not received much attention. This review, therefore, will not investigate this, or any other, dynamic technique further.

A.4 RMS Extensions

Because of the increasing complexity of numerous real-time computer applications and the advent of Ada, there has been renewed interest in the basic RMS technique for two main reasons:

- The *Cyclic Executive* model does not map well to the Ada real-time programming paradigm [52:1,2], or any other modern, object-based programming language.
- There was no existing scheduling model that provided the tools necessary for a disciplined, analytical, engineering approach to determining the timing behavior of programs.

Many papers have been published since the mid-1980s addressing ways to overcome the initial RMS weaknesses. In [52], the authors provided a very understandable explanation of the basic RMS technique, and then summarized an anthology of the techniques necessary to make RMS a viable design and analysis tool:

- a method to "relax" the worst-case utilization limit
- several different ways to handle aperiodic tasks
- resolving task synchronization issues (a mechanism to allow interdependent tasks to work cooperatively)

Even though this reference includes "Ada" in its title, the authors spend most of their time explaining some extensions to RMS. This summary of RMS and its enhancements shows what makes the *GRMS* techniques an applicable suite of tools for any language (in fact, by the time the SEI was publishing articles, reports, and a book intended for more than just a military audience, they made it clear that *RMS* was not strictly applicable to Ada).

A.4.1 RMS Reiterated and Relaxed.

The authors begin their report by restating and explaining theorem 5 of Liu and Layland (above). They note that this theorem is a sufficient (worst-case) algorithm. To provide a more realistic limit, the authors include another, more complicated, technique for use if the simple, sufficient algorithm fails to verify task set schedulability. The authors extract this technique from the results of a

theorem in [37]. For a set of tasks to be schedulable, one of the inequalities from the family of equations described by the following formula will be true:

$$\forall i, 1 \leq i \leq n, \min(\sum_{j=1}^{i-1} C_j \frac{1}{lT_k} \lceil \frac{lT_k}{T_j} \rceil + \frac{C_i}{lT_k}) \leq 1 \qquad (A.1)$$
$$(k,l) \in R_i$$

$$\text{where } R_i = \{(k,l) \mid 1 \leq k \leq i, l = 1, \ldots, \lfloor \frac{T_i}{T_k} \rfloor\}$$

Equation A.1 produces a family of inequalities and appears very complicated. By way of explanation, the authors provide a set of examples to aid in understanding how equation A.1 is applied. Their most complete example follows:

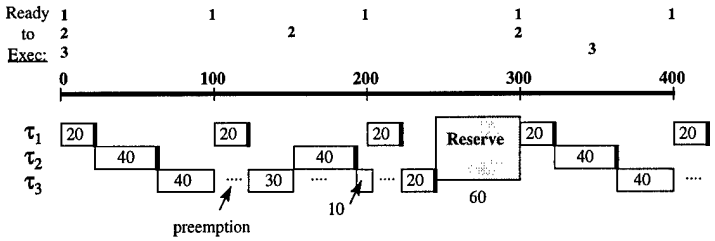
Example 3: [52:6 – 7] 3 Periodic Tasks:

Task	Exec. Time C_i	Period T_i	Utiliz'n Factor U_i	Cumul. Utilization U	RMS Limit
τ_1	40	100	0.400	0.400	1.0
τ_2	40	150	0.267	0.667	0.828
τ_3	100	350	0.286	0.953	0.780

(See figure A.2 for a graphical representation)

By Liu and Layland's theorem 5, the first two tasks are guaranteed to meet their deadlines since their Cumulative Utilization was less than the RMS Limit (i.e., τ_1 and τ_2 are the *Stable Set*). Equation A.1 (which develops the idea of scheduling points) is necessary to determine if any phasing of the task set will allow τ_3 to meet its deadlines (as long as one of the task phasings holds, the task set will meet all its deadlines):

1.)	$C_1 + C_2 + C_3 \leq T_1$	$40 + 40 + 100 \leq ? 100$	no	$l = 1, k = 1$
2.)	$2C_1 + C_2 + C_3 \leq T_2$	$80 + 40 + 100 \leq ? 150$	no	$l = 1, k = 2$
3.)	$2C_1 + 2C_2 + C_3 \leq 2T_1$	$80 + 80 + 100 \leq ? 200$	no	$l = 2, k = 1$
4.)	$3C_1 + 2C_2 + C_3 \leq 2T_2$	$120 + 80 + 100 \leq ? 300$	YES	$l = 2, k = 2$ or $l = 3, k = 1$



	Ci	Ti	U(i)	Priority
τ_1	20	100	0.20	1
τ_2	40	150	0.267	2
τ_3	100	350	0.286	3

Total: 0.753 [$\leq$ 0.779 = $(2^{(1/3)}-1)$]

Figure A.2 Rate Monotonic Timeline

5.) $4C_1 + 3C_2 + C_3 \leq T_3$ $\quad 160 + 120 + 100 \leq ? \; 350$ no $\quad l = 1, k = 3$

This example shows that, while theorem 5 of Liu and Layland could not guarantee all deadlines would be met, applying equation A.1 shows that after 300 units of time (#4 above), τ_1 will run three times, τ_2 will run twice, and τ_3 will run once. Since an arrangement of tasks has been found that allows all the necessary computations to complete prior to their required deadlines, the tasks will meet their deadlines at all times [52:7].

A.4.2 Aperiodic Tasks. In general, real-time tasks (and their timing requirements) are well-defined and the majority occur on a regular, repeating basis. In their paper [39], Liu and Layland used those facts to make the simplifying assumption of programs with *only* periodic tasks. Unfortunately, most real-time systems also have aperiodic task requirements driven by aperiodic events (a pilot pushing a button to change a display, a sensor failure, etc.). Since aperiodic events occur unexpectedly, they do not have deadlines the same way periodic events do. In general, aperiodic events are urgent in nature, and quick control system reaction. Several techniques exist to allow aperiodic tasks to be handled by the *GRMS* family of algorithms.

The authors detail the Deferrable Server (DS) and Sporadic Server (SS) algorithms. The DS technique was obtained from [36]. The SS technique was borrowed from [58]. Both techniques create a "Server" task defined by some C_s execution time every T_s units of time. In other words, the Server is treated as a periodic task and is assigned a portion of the available computational resources ($U_s = \frac{C_s}{T_s}$). These Server tasks can then be treated as any other periodic task and used to evaluate schedulability.

A.4.3 Task Synchronization. Since one of the most important things in real-time systems is that critical deadlines are met, anything that can threaten the ability to accurately predict schedulability must be addressed. Two such issues recur in the literature: priority inversion and mutual deadlock. How mutual deadlock can prevent deadlines from being met is self-evident: if two tasks are waiting for a semaphore to be released the other task has locked (called deadlock), neither of them will meet any of their deadlines.

One task sending data to another is a form of task synchronization. This type of synchronization could be called "communicating." Multiple tasks sharing a resource, such as memory or an input/output device, is a more complex use for synchronization. To prevent interference between processes accessing the shared resource, each task using the shared resource implements a "critical section" to prevent multiple tasks from simultaneously accessing that shared resource. The coordination of critical sections is a form of task synchronization. This type of synchronization makes use of one or more mechanisms to prevent tasks from accessing a shared resource if another task is using it. Typically, semaphores are used for this type of mutual exclusion, and that is where caution must be exercised to avoid creating deadlock.

For real-time programming, task priorities also factor into the concerns about critical sections. Lower priority tasks that begin to use a shared resource will prevent higher priority tasks from executing until the lower priority task finishes its critical section. This problem is called "priority inversion" and results in a period of "blocking" time. An admirable design goal would be to completely eliminate priority inversion; the best that can be done is to limit the inversion to only one out-of-order task execution. If a set of

low priority tasks prevent a high priority task from executing, the high priority task may end up not being able to meet its deadlines. While protective mechanisms for critical sections is obviously needed, these mechanisms can threaten task set schedulability since they can lead to unbounded blocking [30:2-4–2-5]. So if priority inversion is not tightly controlled, schedulability and responsiveness can be hurt. While priority inversion and blocking cannot be completely eliminated, they can be limited.

The Liu and Layland paper assumed that the tasks being analyzed were independent, and therefore no support for synchronization was necessary. Unfortunately, real-world applications rarely require only independent tasks to produce even a simple result. The authors of [37] include a summary of a technique that provides for bounded blocking and freedom from mutual deadlock. The technique is called the Priority Ceiling Protocol (PCP). This protocol is discussed in a number of different papers and technical reports, but the authors extracted their data from [25].

To implement the protocol in any object-based language (like Ada), several rules must be followed (directly from [25:20, 21]):

1. There must be no critical sections guarded by a semaphore (one of the mechanisms available to protect critical sections; in other words, all the CLIENT tasks that would access this semaphore before proceeding into their critical section are contained in this SERVER task as the bodies of the Ada ACCEPT [or other language equivalent] statements).
2. There must be no conditional or timed entry calls.
3. Each task must be assigned a priority.
4. A SERVER task must have an (assigned) priority lower than that of its CLIENT tasks.

While these requirements may seem overly restrictive, the authors of [25] are confident that "a useful set of real-time applications can be programmed using the limited form of SERVER tasks currently allowed by [this] protocol." For the reader interested in "pure" PCP, [25] provides a thorough set of examples to help understand PCP functionality.

While this protocol is very powerful, it is difficult to implement, and requires compiler support not available in many real-time kernel runtimes [50:pt. 2, pg. 12]. A suitable emulator, called the Highest Locker's Protocol (*HLP*), can be implemented in a number

of real-time programming languages fairly easily. Since the performance difference between the two models is relatively insignificant [50:pt. 2, pg. 12], modeling this simpler technique in anlaytical tools should not be very hard, and would respresent a conservative estimation of schedulability (if a task set does use PCP, it should be even more schedulable than the *HLP* implemented task set). Once a set of tasks is implemented using PCP (or an emulator, such as *HLP*), the authors prove that the task set will meet its deadlines if the following inequality is true:

$$\frac{C_1}{T_1} + \cdots + \frac{C_n}{T_n} + \max(\frac{B_1}{T_1}, \ldots, \frac{B_{n-1}}{T_{n-1}}) \leq n(2^{1/n} - 1) \tag{A.2}$$

Equation A.2 is a generalization of Theorem 5 of [39] with the addition of blocking factors (remember, Theorem 5 assumed independent tasks). Because equation A.2 is a generalization of the original theorem, it is also a worst-case ('sufficient') scenario. This equation also has a more complicated counterpart for use if the simple, sufficient algorithm fails to verify task set schedulability. The authors extract this technique from the results of a theorem in [37]. If a set of tasks is schedulable, one of the inequalities from the family of equations described by the following formula would be true:

$$\forall i, 1 \leq i \leq n, \min(\sum_{j=1}^{i-1} C_j \frac{1}{lT_k} \lceil \frac{lT_k}{T_j} \rceil + \frac{C_i}{lT_k} + \frac{B_i}{lT_k}) \leq 1 \tag{A.3}$$
$$(k,l) \in R_i$$

where R_i is defined in equation A.1 and B_i is the worst-case blocking for τ_i

Like equation (A.1), this equation produces a family of inequalities and appears very complicated. By way of explanation, the authors again provide an example as an aid to understanding. Since the equations just add a blocking factor to those used to explain equation A.1, the example will not be repeated here. The most important new idea is that the longest blocking possible ($\max(B_i)$) is the largest C_i of all the lower priority tasks. This limited blocking is due to the implementation of PCP.

A.5 Distributed Real-Time Control Issues

The use of a distributed system of cooperating multi-processors for a large "plant" (such as a manufacturing facility or modern aircraft) has become common practice in the last few years. The same has occurred for the embedded control of aircraft and space-oriented systems. The days of isolated/specialized computers to control aircraft and space-oriented subsystems are gone. Integrated multi-processor/multi-tasking "computers" that work closely together are planned for several new aircraft and space-oriented systems to provide improved reliability, maintainability, and redundancy management while making more efficient use of available resources.

Integrating the control of related systems provides more efficient control of all systems involved. The premise is that, for example, the control of a thrust vectoring engine and the influence of the aerodynamic control surfaces could work cooperatively to change the attitude, direction, etc. of aircraft. Providing a means for these controllers to collaborate would relieve the pilot of some workload and improve performance.

However, this type of integrated control system would greatly increase the synchronization (and therefore the communication) requirements of the system. As more systems become interdependent, the penalties of interprocessor communication increase. This is called the "saturation effect" and is pictured in Figure A.3 [23]. The probability of contention-free communication (i.e., no other processor trying to communicate at the same time) will quickly and drastically decrease as the communication requirements increase. The accompanying reduction in throughput is due to increased contention for shared resources and the additional amount of overhead required to support appropriate interprocessor communication (i.e., more processors, more overhead). As tasks must wait an unpredictable amount of time for media access (since contentions may or may not exist), task execution time will appear to fluctuate, potentially threatening the schedulability of the task set. Any attempt to model integrated multi-processor distributed control systems will need to consider this dynamic behavior and its impact on predicting task set schedulability.

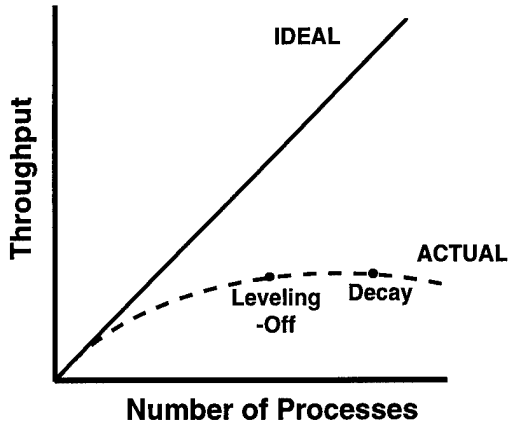


Figure A.3 The Multi-Processor Saturation Effect [23]

Shaffer, in [51] showed that assigning interdependent tasks to the same processor will reduce interprocessor synchronization. Since most of the synchronization requirements are now within the same processor, this regime will help reduce communication overhead for the entire system.

The more independent the threads of execution are on each processor, the greater the increase in the computation to communication ratio (effectively enlarging the granularity of the application), reducing the impact of the communication workload. In fact, [61] states "... ideally, the controller task size must be very large, so that [the resulting communications overhead] does not become the primary limit on the sample time achievable." In this context, sample time is an indication of how well the control system is performing. In other words, if the "granularity" of the application (size of the task set within one processor) is large enough, the communication penalties will be minimized. So judicious distribution of tasks can help reduce the impact of interprocessor synchronization and communication, no matter what communication media is used.

A.5.1 Interprocessor Communication and I/O. Interprocessor synchronization can be modeled as a set of server tasks [41]. And since aperiodic tasks can be modeled as periodic ones using the Deferrable or Sporadic Server [52:11,12], communications between periodic and/or aperiodic tasks on multiple processors can be handled

this way. In addition, since input and output (I/O) could be characterized as a type of communication, it can, under certain circumstances, also be modeled with a set of periodic servers [31], similiar to what is shown in Figure A.4. This technique does require some underlying communication subsystem support (such as in [60] and [54]).

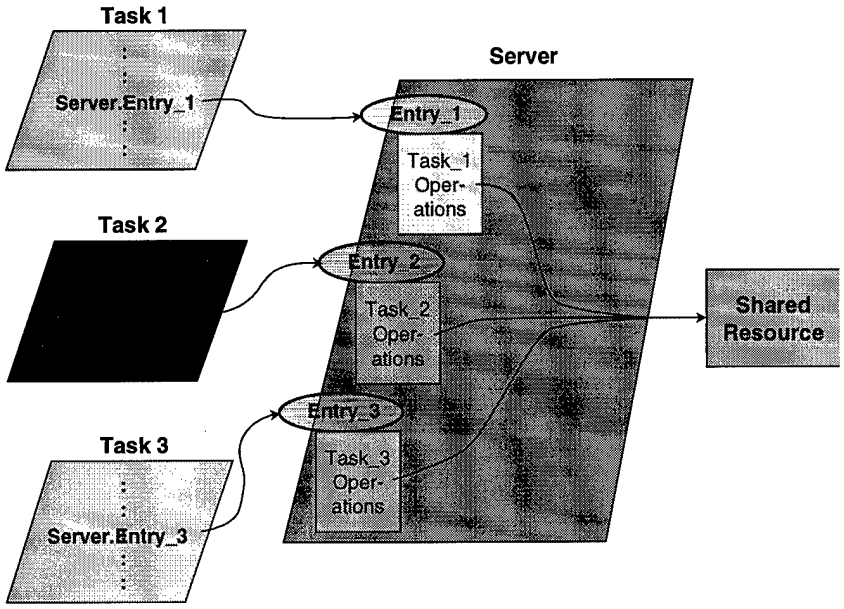


Figure A.4 Communication Server

A.5.2 Required Hardware Support. Though this review focuses on predicting the schedulability of software tasks using *GRMS*, support from the underlying hardware cannot be ignored. The general requirements for *GRMS* support from the communication subsystem [60:43] are the abilities to:

- Establish message priorities
- Resolve media contention by preemption
- Prioritize resolution by having sufficient number of priority levels comparable to priorities of the tasks simultaneously competing for media access

Just as in the computational arena, the difference between real-time and general communications is the introduction of concerns about time [60:42]. Current communication

architectures may not provide the underlying support necessary to implement *GRMS*, and may impact the schedulability of any distributed control system. The next section provides some additional insight into the impact communications systems can have on real-time systems.

A.6 Communication Architectures

Most common networking communication systems are not designed to support real-time communications, particularly the type of support *GRMS* requires (limited priority inversion, etc.). Ethernet, one of the most common communications architectures, falls into that non-support category. But since the baseline *SIMSAT* control system uses that architecture, the remediation of the impact of using the protocol needs to be considered.

Venkatramani [63] considers Ethernet the architecture of choice for most networked environments because

- newer Ethernet technologies are being developed to increase the data rates to 100Mb/sec
- recent hardware support has enhanced the scalability of the architecture
- prices will continue to be low (even for the newer technologies) due to economies of scale

But as he indicates, the major problem with Ethernet as a real-time communications network is the contention-based protocol it uses for medium access:

> "Multiple nodes compete for access to the channel and on detecting a collision, backoff for a random interval before attempting retransmission. Hence, Ethernet does not provide deterministic access times to the network. Although the Ethernet protocol has a provision for prioritized access arbitration, this mechanism does not in itself offer guaranteed bandwidth to an arbitrary pair of nodes. Besides most commodity Ethernet controllers do not necessarily implement this feature." [63:282]

His article details a real-time Ethernet protocol (called 'RETHER') that can be implemented without any hardware modifications, addressing one of the short-comings he found with several other proposed modifications to the Ethernet protocol. One other problem

the RETHER protocol addresses is how to limit the impact on non-real-time traffic on the network—it is a hybrid implementation to allow on existing networks. While the main thrust of this development was to improve the performance of multi-media applications over commercial LANs, the issues addressed in the article still apply to real-time control applications.

Since *SIMSAT*, implemented with the dSPACE AutoBox system, is based upon the Ethernet protocol, the issues raised in this article could become an issue. But as long as the Ethernet system being used remains a peer-to-peer network (the only nodes on the network are the Simulation PC and the AutoBox), there will be no interference from other nodes. This paper, therefore, provided good rationale for NOT connecting the AutoBox to the existing AFIT network.

A.7 Analytical Tools

As indicated in Chapter I, *GRMS* and other theoretical techniques require additional analytical support to handle the impact of the underlying run-time environment. That is where *RATESIM* began [5]. *RATESIM* was intended to simulate the impact of system tasks on the ability of the user task set to meet its deadline when executing in a particular run-time environment (with user provided information about the run-time being used and the salient characteristics of the intended user task set: C_i, T_i, and task interdependencies). But *RATESIM* requires extensions to provide analytical support for periodic and aperiodic tasks, task synchronization (intra- and inter-processor), and various input/output principles. Even if extended, *RATESIM* would require the user to determine all the system task parameters, including those for the communication system. While this would be a one-time task, the value of extending *RATESIM* to exclusively support the *SIMSAT* effort should not be dismissed.

A.8 Conclusion

Why, if the *Cyclic Executive* has been the scheduling technique of choice for numerous real-time systems in the past, should *GRMS* replace it? The *Cyclic Executive* has several problems, which include:

- Incompatibility with many object-based real-time programming paradigms (such as Ada) [52:1]
- Inflexibility when computational loads vary leads to missed deadlines or wasted resources [5:2-3], [60:42]
- Frailty during frame overrun error handling can lead to a wave of missed deadlines [43:51]
- Necessary use of harmonic task execution rates results in unnecessary additional "computational" load on the processor (processor time spent doing nothing, waiting for the next minor cycle to start) [41:182]
- Complexity of hand-crafted time-lines and complications of handling multiple modes leads to excessively high development costs [60:42]
- Interaction of logical constructs and scheduling decisions causes expensive design modifications since the entire time-line must be reevaluated if code changes alter the timing assumptions made in the original development [43:51]
- Informality of time-line development and documentation reduces maintainability (increasing overall lifecycle costs) [60:42]

Every one of these problems has been effectively handled by *GRMS* techniques:

- Direct support for the Ada real-time programming paradigm [52:2] (and by extension, other object-based languages [24:20])
- Critical tasks (in the *Stable Set*) will ALWAYS meet their deadlines, no matter what other tasks get added (unless those tasks are higher priority) [41:182], [52:9]

- Separation of logical and temporal concerns possible with the RMS scheduling technique remove the concerns about the relationship of execution time between tasks and schedulability of the task set [52]
- Adapts to changing computational loads [60:42]
- Use of simple, formal tools to handle all temporal concerns reduces development costs [52]
- Use of formal mathematical analysis tools for schedulability determination reduces design modification costs and therefore improves maintainability [52]

In short, *GRMS* provides a formal, systematic approach to schedulability analysis, matching very well with contemporary systems and software engineering philosophies. Chapter VI provides some suggestions on how to update the experimental setup and analytical tools to more adequately support true pre-experiment temporal analysis.

Appendix B. Raw Data Details

B.1 Overview

This appendix provides a detailed description of the system level data used to evaluate alternatives during the *Detailed Design* life-cycle phase. The data also includes the range of data the team estimated each attribute might take. The estimates are Rough Order of Magnitude estimates based upon typical manufacturers information for the various subsystems and also consider how the subsystems interact with respect to these system level measures.

Variability data was based upon engineering estimates generated after researching the various subsystem alternatives. As the data for the DSPACE system was readily available and more defined than the data for the non-DSPACE alternative, the variability data in the tables is greater for the latter alternative. While all the continuous, direct measures include variability, the constructed scale measures do not for two main reasons:

1. the attribute was determined to only be influenced by the C&DH subsystem
2. the attribute variability is captured by the inherent "range" of any constructed score assignment (i.e., a **Partial** *Control Systems Analysis* score assessment for a given alternative does not mean the measure was expected to take on one value, but rather falls in the range of "50–90% of the elements are defined")

As in the rest of this document, the four alternatives addressed in this appendix are:

All on Sat: both the ADAC and EXP task sets execute in the AutoBox

Split: ADAC tasks run on the ground, while the EXP tasks are running on AutoBox

Grd w/ AutoBox: all tasks executing on the ground, with AutoBox providing signal consolidation to and from the satellite

Grd w/o AutoBox: all tasks executing on the ground, using something other than the AutoBox to consolidate the data signals. DSPACE software environment was maintained.

The following sections present the data for each fundamental value: *Cost*, *Schedule*, *Safety*, and *Performance*, followed by a section that consolidates the results.

B.2 Cost Data

Alternative	Purch + Integ $	O & M $	Notes
All on Sat	$29.5K ± $0.5K	$8.0K ± $0.25K	
Split	$34.5K ± $0.5K	$8.0K ± $0.25K	Extra Processor
Grd w/ AutoBox	$34.5K ± $0.5K	$8.0K ± $0.25K	Extra Processor
Grd w/o AutoBox	$34.5K ± $0.5K	$8.5K ± $0.5K	Aerospace Parts

Table B.1 Raw Data Range—*Cost*

The estimated purchase and integration costs were broken down as:

- ADACS: $17,000 (motors, wheels, sensors, misc. parts)
- Power System: $5,000 (batteries, chargers, distribution/regulation system, etc.)
- Structures: $500 (misc. parts and fabrication costs)
- C&DH: $7,000 (AutoBox, wireless LAN, misc. cabling)

These baseline costs were for the *All on Sat* option. The increase in cost for the *Split* and the *Grd w/ AutoBox* reflect the need for an additional processor and software upgrade to support a multi-processor environment. The increase in cost for the final alternative is related to the purchase other parts to replace the functionality of the AutoBox; the expectation was that those parts would likely approximate the *AutoBox* alternatives.

The estimated annual operation and maintenance costs (periodic maintenance, consumables, misc. parts) were broken down as:

- ADACS: $5,000
- Power System: $2,000
- Structures: $0
- C&DH: $1,000

These baseline costs were for all the *AutoBox* options. The increase in cost for the final alternative was attributed to the increased likelihood that components of a non-integrated solution are less reliable than those of an integrated solution. This would obviously drive up O&M costs.

B.3 Schedule Data

Alternative	Total Delivery Weeks	Notes
All on Sat	34 ± 1 week	
Split	35 ± 1 week	Additional Processor
Grd w/ AutoBox	37 ± 1 week	Software Integration
Grd w/o AutoBox	40 ± 2 week	More Integration

Table B.2 Raw Data Range—*Schedule*

The CDM needs the system available for usage a little more than a year after this study was started. For the system to be "available," subsystem parts have to be ordered, delivered, and integrated. The order and delivery time for each system has to be completed before system-level integration can begin, so the data in the table was based upon how long it would take to receive all the subsystems, followed by the integration time, which was assumed to be the time required to make the subsystem functional.

To capture the impact of both subsystem and system-level integration, the team assumed the integration time would not start until all subsystems had delivered, and then would proceed sequentially rather than concurrently. This decision was based upon the idea that only limited integration could be done before all systems were available, and some of the initial integration would have to be reaccomplished during system-level integration. Past experience suggested the need for a rather significant contingency buffer as well.

The times shown in the table above were based upon the following baselines:

- ADACS: 5 (order) + 10 (delivery) = **15 weeks**; 6 weeks integration
- Power System: 8 (order) + 4 (delivery) = **12 weeks**; 4 weeks integration
- Structures: 2 (order) + 3 (delivery) = **5 weeks**; 2 weeks integration

- C&DH: 4 (order) + 15 (delivery) = **19 weeks**; 3 weeks integration

From this data, the baseline *All on Sat* schedule impact was 19 + 15 = 34 weeks. The additional week for going to the *Split* option was for the impact of ordering additional hardware and software. The time increase for the *Grd w/ AutoBox* option reflects that same impact as well as an increase in software integration time. The score shown for the *Grd w/o AutoBox* option reflects the additional order time required (6 vs. 4 weeks) to determine the parts to be ordered, and increased integration time (6 weeks) required to make disparate parts function as a subsystem.

B.4 Safety Data

Alternative	Rel. Damage Ind.	Rel. Injury Ind.	Notes
All on Sat	18 ± 1	18 ± 1	
Split	18 ± 1	18 ± 1	
Grd w/ AutoBox	18 ± 1	18 ± 1	
Grd w/o AutoBox	16 ± 2	16 ± 2	More Parts

Table B.3 Raw Data Range—*Safety*

As all of the subsystems were expected to either be commercial products or carefully designed, both the equipment and personnel safety indicators were expected to be very high. To provide some conservatism, the team decided to reduce the safety ratings a little for unanticipated interactions between the subsystems (the baseline scores) and reduce it further for non-validated integration (the final alternative).

B.5 Performance Data

As the *Performance* value "sub-tree" is so big, the following measures are organized alphabetically, two measures described at a time.

The baseline *All on Sat* alternative is defined as only needing to send the Display and Command updates. The more functionality that moves to the ground, the higher the

Alternative	B'width Req'ts	Cmd Capab.	Notes
All on Sat	Low	Full	
Split	Mod	Full	More Signals
Grd w/ AutoBox	High	Full	All on Grd
Grd w/o AutoBox	High	Full	All on Grd

Table B.4 Raw Data Range—*Performance* (Bandwidth/Command Capability)

Bandwidth requirements. Since both *Grd* options have all the tasks running on the ground station, their *Bandwidth* requirements are the worst.

For *Command Capability*, the team assumed that some form of that alternative could be found to provide that functionality. This measure was originally included to measure more than just C&DH functionality, but when other subsystem alternative choices were no longer possible or necessary, the usefulness of this measure was minimized.

Alternative	Comm Lat'cy	Ctrl Sys. Anlys.	Notes
All on Sat	Minimum	Full	
Split	Moderate	Full	Incr. Overhead
Grd w/ AutoBox	Significant	Full	High Overhead
Grd w/o AutoBox	Significant	Partial	High Ovhd/Parts

Table B.5 Raw Data Range—*Performance* (Comm. Latency/Ctrl. Sys. Analysis)

While correlated to *Bandwidth*, *Communications Latency* was concerned with what part of the control system is impacted by communication systems delays. An impact to the outer control loop creates different problems than an impact to the inner control loop. By definition, the *All on Sat* alternative would have the most impact, and both the *Grd* options would have the worst impact.

For *Control Systems Analysis* support, the dSPACE integrated solutions are all assumed to provide at least 90% of the needed system elements. About the only parts not previously defined would be those common to each of the alternatives (communication system, sensors, effectors, etc.). The *Grd w/o AutoBox* option, though, will require definition of most elements.

Alternative	Develop. Env	Exp. Types	Notes
All on Sat	Obj. Oriented	Full	
Split	Obj. Oriented	Full	
Grd w/ AutoBox	Obj. Oriented	Full	
Grd w/o AutoBox	Graphical	Rigid	Non-Integrated

Table B.6 Raw Data Range—*Performance* (Develop. Environ./Exper. Types)

Again, the dSPACE *Development Environment* is intuitive and Object-Oriented since it is built upon the SIMULINK foundation. As the last alternative is not using dSPACE hardware on the satellite, there is a significant gap in the available "building blocks", at least initially.

The *Experiment Types* measure was established with the assumption that if an alternative could do the *3-Axis (Rigid)* experiments it could do the *Educational* experiments as well. And if a alternative could do *3-Axis (Flex)* experiments, it was assumed to be capable of doing all three. One of the initial assumptions for this design effort was that the integrated dSPACE solution had sufficient power to handle all the *Experiment Types.* While the performance may suffer as the tasks move off the satellite (as reflected in other measures), the team expected the integrated dSPACE solution would still support a full complement of experiment types. Once additional, non-dSPACE hardware got added to the system, the team thought *SIMSAT* might not be able to support more than the *3-Axis (Rigid)* experiment.

Alternative	Intfc Modul'ty	Maint/Test Time	Notes
All on Sat	Partial	Very Low	
Split	Partial	Very Low	
Grd w/ AutoBox	Partial	Very Low	
Grd w/o AutoBox	Full	Low	Part: Flex./Time

Table B.7 Raw Data Range—*Performance* (Interface Modularity/Maint. & Test Time)

To maintain system integrity, the team assumed dSPACE components could not be easily replaced, giving each of the integrated dSPACE solutions only a *Partial* score for *Interface Modularity.* But since the *Grd w/o AutoBox* alternative was based upon

the integration of variety of pieces, it would allow for *Full* component substitution or replacement.

But this has an adverse impact when considering *Maintenance and Test Time.* Changing something that impacts a set of non-integrated parts will take longer to validate, thus the down-graded score for the *Grd w/o AutoBox* alternative.

Alternative	Mass Marg.	Motion Sim.	Notes
All on Sat	100 ± 5	Yes	
Split	100 ± 5	Yes	
Grd w/ AutoBox	100 ± 5	Yes	
Grd w/o AutoBox	120 ± 10	Yes	No AutoBox

Table B.8 Raw Data Range—*Performance* (Mass Margin/Motion Simulation)

The team developed the baseline system mass by estimating the mass required for each of the subsystems:

- ADACS: 18 kg (motors, wheels, sensors, misc. parts)
- Power System: 7 kg (batteries, distribution/regulation system, etc.)
- Structures: 15 kg (misc. integration and support parts)
- C&DH: 10 kg (AutoBox, wireless LAN, misc. cabling)

The baseline mass for the dSPACE-based options is 50 kg, leaving a 100 kg *Mass Margin* beyond the air-bearing capacity of 150 kg. The increase in mass margin for the *Grd w/o AutoBox* reflects the likelihood that the signal consolidation equipment required would be lighter and more compact than the AutoBox solution.

The *Motion Simulation* measure was a vestige of the *Simulation Fidelity* consideration in the *Concept Exploration* values hierarchy that the team originally thought might play a factor during this life-cycle. As it turned out, the data seemed to indicate that all the alternatives evaluated in this effort would support some ability to simulate the actions of the system prior to actual implementation.

Alternative	Pos. Sens'g	Post Msn Anlys.	Notes
All on Sat	10^{-2}	Yes	
Split	10^{-2}	Yes	
Grd w/ AutoBox	10^{-2}	Yes	
Grd w/o AutoBox	10^{-2}	Yes	

Table B.9 Raw Data Range—*Performance* (Pos'n Sensing/Post-Mission Anlys.)

Like *Motion Simulation* above, both these measures were expected to be a factor when the values hierarchy was developed, but once the data was collected, the team found the measures to have little use in helping to differentiate alternatives.

Alternative	Pwr Margin	Proc. Sch. Anlys.	Notes
All on Sat	7 ± 2 A-hr	Full	
Split	7 ± 2 A-hr	Full	
Grd w/ AutoBox	7 ± 2 A-hr	Full	
Grd w/o AutoBox	10 ± 5 A-hr	Mod	Optimized Parts

Table B.10 Raw Data Range—*Performance* (Power Margin/Proc. Sched. Anlys.)

The team developed the baseline system power needs by estimating the power required for each of the subsystems:

- ADACS: 200 W (motors, wheels, sensors, misc. parts)
- Power System: 0 W (assuming negligible losses in the system)
- Structures: 0 W (no powered parts)
- C&DH: 145 W (AutoBox, wireless LAN)

The baseline power requirement for the DSPACE-based options is 345 W; assuming a nominal voltage of 24 VDC and experiment duration goal of 60 minutes, the demand works out to 12.32 A-hr. Adding a little more load for line losses, and considering the power system was originally sized to support 20 A-hr, the DSPACE baseline alternatives *Power Margin* were estimated at 7 A-hr. The increase in power margin for the *Grd w/o AutoBox* reflects the likelihood that the signal consolidation equipment required could be chosen to be more energy-efficient than the AutoBox solution (the power C&DH requirement dropped to 50 W $\Longrightarrow$ 250 W total $\Longrightarrow$ 10 A-Hr requirement and margin).

The DSPACE alternatives clearly support Rate Monotonic Analysis [19:75], so receive full credit for the *Processor Schedulability Analysis* measure. And while the *Grd w/o AutoBox* alternative still uses the same software, some hand-coding would likely be required to support schedulability analysis using the non-DSPACE hardware (a *Moderate* score).

Alternative	R-T Data	Slew Capab.	Notes
All on Sat	Yes	60 ± 5 deg/10 sec	
Split	Yes	60 ± 5 deg/10 sec	
Grd w/ AutoBox	Yes	60 ± 5 deg/10 sec	
Grd w/o AutoBox	Yes	70 ± 10 deg/10 sec	Parts Flex.

Table B.11 Raw Data Range—*Performance* (Real-Time Data/Slew Capability)

Like the *Post Mission Data Analysis* measure, the *Real-Time Data Acquisition* measure was expected to provide some differentiation between alternatives. As it turned out, all the alternatives supported some type of data acquisition.

The team made the assumption that the Momentum Wheels being designed would be able to support a *Slew Capability* of 60°/10 sec in the baseline system configuration (with AutoBox on the satellite). As the *Grd w/o AutoBox* option integrate piece parts housed on the satellite, the team assumed there would be more flexibility in locating those parts to reduce moments of inertia (MOI). The team also expected the MOI would improve with the reduced subsystem weights.

Alternative	Turn Time	User Intfc	Notes
All on Sat	1 hour	Full	
Split	1 hour	Full	
Grd w/ AutoBox	1 hour	Full	
Grd w/o AutoBox	1 hour	Partial	Non-DSPACE Hardware

Table B.12 Raw Data Range—*Performance* (Turn-Around Time/User Interface)

At the beginning of this life-cycle phase, the team anticipated the need for a measure to determine how easy it was to remove and replace batteries. *Turn-Around Time* was intended to measure the impact of structural designs on the "maintainability" of the system.

As neither *Structures* or *Power* alternatives (i.e., those that interact with this measure) were not considered during this life-cycle phase, this became another "dead" measure, not adding any information to the alternative selection process.

As the DSPACE solutions provided a tightly integrated solution, including a well-developed *User Interface*, it satisfied the criteria for a *Full* score. Adding non-DSPACE hardware to the last solution in the table meant the user interface would require some work before the system would be ready for full implementation.

Finally, as mentioned before, since team reached no resolution regarding the *Slew Rate Sensing* measure, the evaluation of C&DH alternatives assumed no value being added to any of the alternatives for this measure. Had time permitted, the team would have gone back, updated the values hierarchy, and changed the LOGICAL DECISIONS model to get more spread between the alternatives considered.

B.6 Consolidation of System Data

Alternative	Min. Value	Nom. Value	Max. Value
All on Sat	7.53	7.73	7.94
Split	7.08	7.28	7.48
Grd w/ AutoBox	6.64	6.84	7.04
Grd w/o AutoBox	5.34	5.77	6.12

Table B.13 Overall Value Range

This table reflects the range of values after taking the data from the previous tables, transforming it into common units, and adjusting it with the weights shown in Table 4.8 (page 4-31). To speed the analysis conducted during *Detailed Design*, the variability data in the tables of this appendix was assumed to be uniformally distributed for each measure. The resulting variability in the *Overall* value for each alternative was conservative. If there had been no overlap in the value scores, that conservatism would not have been an issue. In addition, the minimal overlap that does occur in this application does not create a high risk because the implementation impact between the top three alternatives was

very minimal (primarily a software update). As a result, the *All on Sat* alternative is the preferred solution.

Appendix C. Common Units

C.1 Overview

A variety of techniques have been developed to quantitatively compare multiple attribute alternatives, the most analytically accurate of which is Multi-Attribute Utility (or Value[1]) Theory [32]. This technique requires the normalization of real-world data to a common scale (or units) to allow for algebraic summation which could be called a *decision-making equation.* The mathematical functions to do this conversion are called utility (or value) functions. To ensure the CDM's values are appropriately factored into that summation, weights are assigned (as in Section 4.4.1.2, page 4-30) to each evaluation consideration and real-world measure of merit (MOM) after the data is collected to ensure the range of data is considered in determining those weights.

The conversion from actual data to common units should always be done with a relatively continuous "function," even for the Y/N constructed measures. The need for a continuous function when dealing with continuous data is inherenty obvious; the need for such a function for discrete data is not quite so clear. Kirkwood points out [28:26] that typically constructed scales will not capture every consideration that may need to be factored into an MOM assessment for a given solution. This may lead to a situation where technical judgement leads to a score that is between two values of the constructed scale. Having built a piecewise linear value function provides a means for converting that intermediate score to an appropriate value for the decision-making equation [28:63].

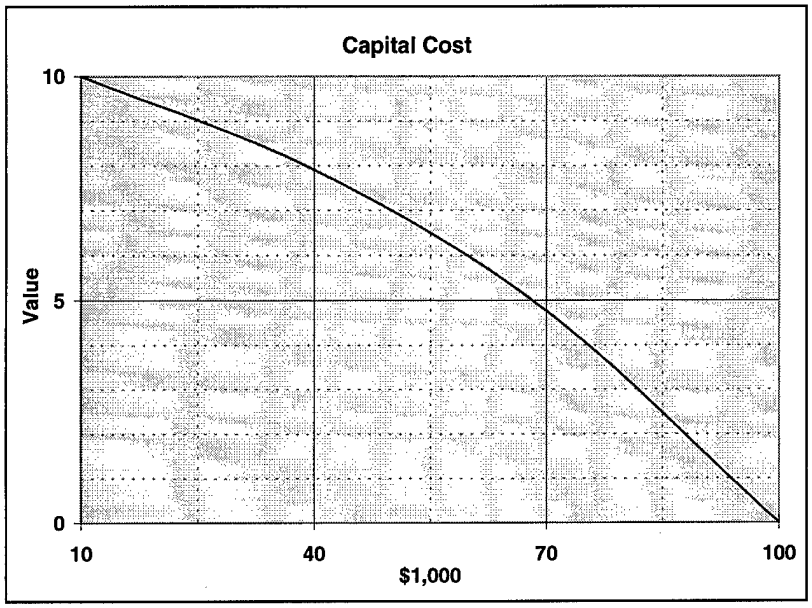
With that in mind, the following pages document the functions the developed with the CDM to convert the raw data collected into common units. The functions are arranged in alphabetic order, by fundamental value (*Cost*, *Schedule*, *Safety*, *Performance*), to make them easier to locate. Also included is the definition of terms used in constructed measures (i.e., "Moderate" vs. "Low" for each measure) as well as the value used to establish the continuous transform functions used for some natural measures (such as *Capital Cost*).

[1]The difference between *Value* and *Utility* is the introduction of uncertainty. Since uncertainties are being handled in other ways in this design effort, uncertainty in this context, and the risks it introduces, will be ignored

The transform for these continuous functions is based upon [28:68], whereby the team worked with the CDM to choose a single comparison between a real-world data point and the value that point has to the CDM. While this may not be the most technically exhaustive method for building the transform functions (see [7:469–487] for more robust techniques), the CDM was comfortable enough with how the decision-making tools fit with his perspectives that he was willing to live with this compromise for the sake of expedient analysis.

Rather than having to worry about all the math that goes into transforming that one point to a continuous exponential function, the team again used LOGICAL DECISIONS to derive the transform function. The "transform" point is included on the figure for the continuous measures to confirm the curve represents the CDM's perspectives[2].

[2]Due to some rounding problems in EXCEL, the curves do not always match exactly. However the functions included with the graph do represent the CDM's transform function for the "transform" point.



Capital Cost	*50*	10	40	70	100	$1,000
Value	*7.00*	10.00	7.91	4.76	0.00	

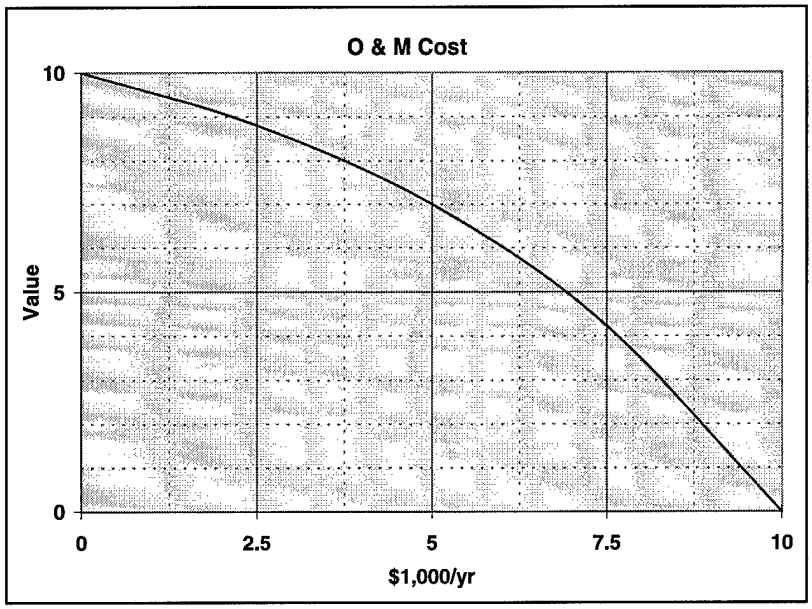
FORMULA: =14.11-3.581*EXP(0.01371*x)

Figure C.1 Capital Cost Value Function

C.2 Cost

C.2.1 Capital Cost. This measure is a continuous "direct" measure reflecting the estimated total cost to purchase and integrate system. The costs include all the primary subsystem components, support parts, and any labor required for the one-time fabrication of the system. The CDM generated this function by selecting the following value comparison; LOGICAL DECISIONS generated the rest:

Value($50K) = 7



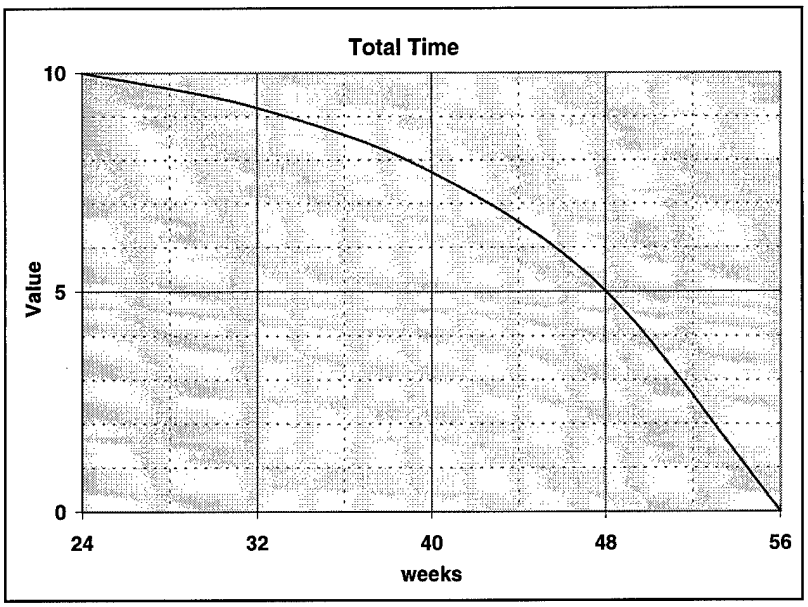
O & M Cost	0	2.5	*5*	7.5	10	$1,000/yr
Value	10	8.81	*7.00*	4.23	0.00	

FORMULA: =12.25-2.25*EXP(0.1695*x)

Figure C.2 O & M Cost Value Function

C.2.2 Operations and Maintenance Cost. This measure is a continuous "direct" measure reflecting the estimated yearly cost to operate and maintain the system. The recurring costs include all the consumables, repair parts, and labor required to keep the system running each year. The CDM generated this function by selecting the following value comparison; LOGICAL DECISIONS generated the rest:

Value($5K) = 7



Total Time	24	32	40	***48***	56	weeks
Value	10.00	9.20	7.72	***5.00***	0.00	

FORMULA: =10.96-0.1539*EXP(0.07617*x)

Figure C.3 Total Time Value Function

C.3 Schedule

C.3.1 Total Time. The only measure for the *Schedule* fundamental value is *Total Time.* This measure was a continuous "direct" measure reflecting the summation of the time required to order, produce, deliver, and integrate the entire *SIMSAT* system. The CDM generated this function by selecting the following value comparison; LOGICAL DECISIONS generated the rest:

Value(48 weeks) = 5

C.4 Safety

The *Safety* measures documented in this appendix for *Equipment Risk* and *Personnel Risk* each use the term "index," based on the table shown in Figure C.4. That table was developed as suggested in [3:7–8, A3–A4], as coordinated with the CDM.

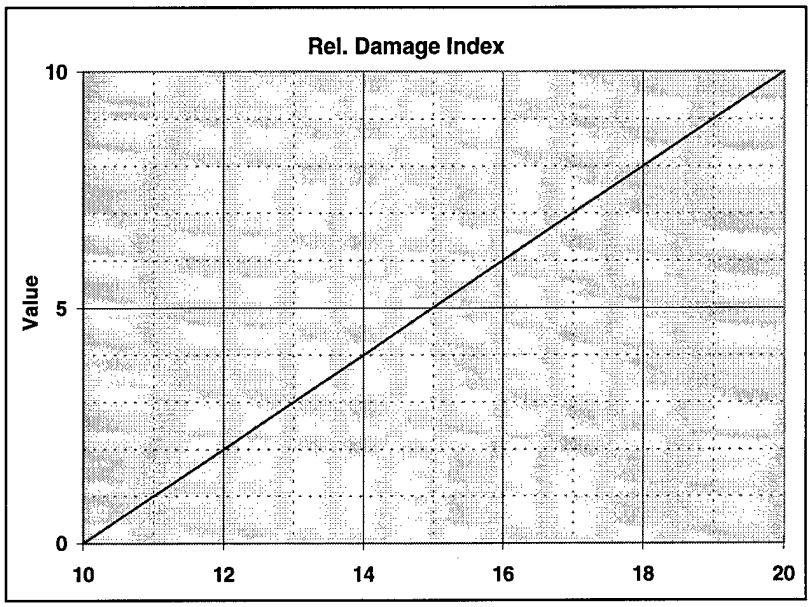
Failure Probability:		Severity of Failure:	Death or System Loss	Severe Injury, Major Damage	Minor Injury, Minor Damage	Less than Minor Injury/Damage
			Catas	Crit	Marg	Negl
0.01	Likely to occur frequently	Freq	1	3	6	10
0.0001 -> 0.01	Occur several times in 5 years	Prob	2	5	9	12
0.00001 -> 0.0001	Likely to occur sometime in 5 years	Occas	4	9	11	15
0.000001 -> 0.00001	May occur sometime in 5 years	Rem	8	12	14	18
< 0.000001	So unlikely can assume may not fail	Improb	10	14	18	20

System Loss means at least 90% of SIMSAT must be replaced
Major Damage means 50-90% of SIMSAT must be replaced
Minor Damage means 25-50% of SIMSAT must be replaced
Less than Minor damage means 0-25% of SIMSAT must be replaced

Severe Injury means at least 1 day of work is missed
Minor Injury means a visit to the hospital is required, but no work is missed
Less than Minor Injury means that, at worst, only minimal first aid is required

Risk Index	Acceptability
1-5	Unacceptable
6-9	Undesirable
10-17	Acceptable with review
18-20	Acceptable as is

Figure C.4 Hazard Index Table

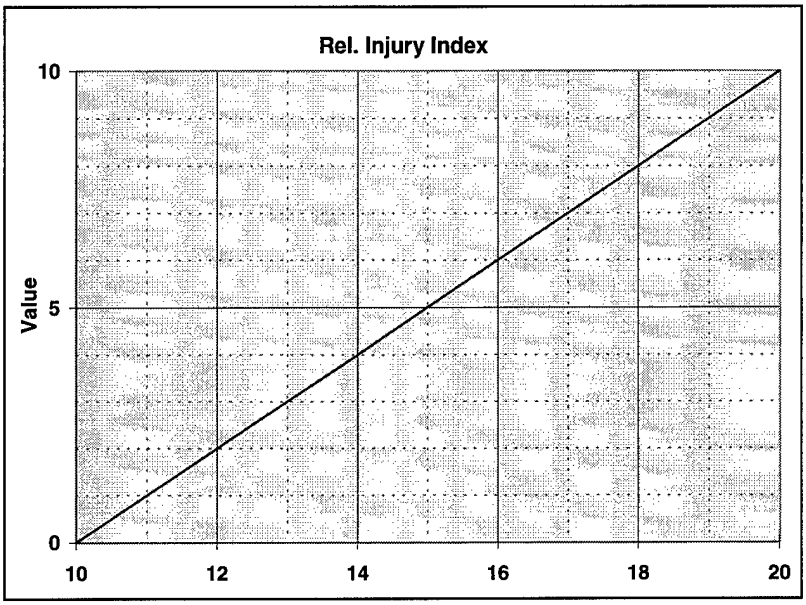


Rel. Damage Index	10	12	14	16	18	20
Value	0.00	2.00	4.00	6.00	8.00	10.00

FORMULA: =-10+x

Figure C.5 Relative Damage Index Value Function

C.4.1 Relative Damage Index. See Figure C.4 (page C-6) for definition of this constructed scale, a combination of probability of failure, and severity of that failure.

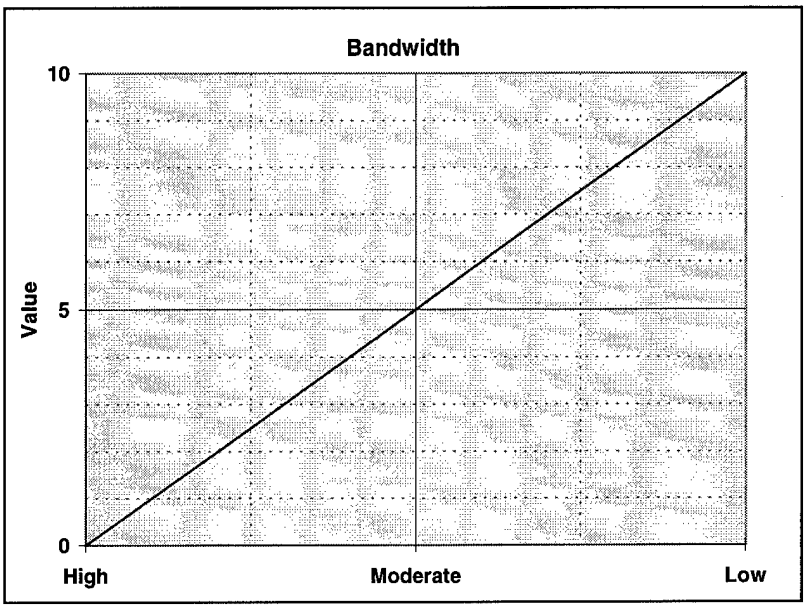


Rel. Injury Index	10	12	14	16	18	20
Value	0.00	2.00	4.00	6.00	8.00	10.00

FORMULA: =-10+x

Figure C.6 Relative Injury Index Value Function

C.4.2 Relative Injury Index. See Figure C.4 (page C-6) for definition of this constructed scale, a combination of probability of failure, and severity of that failure.



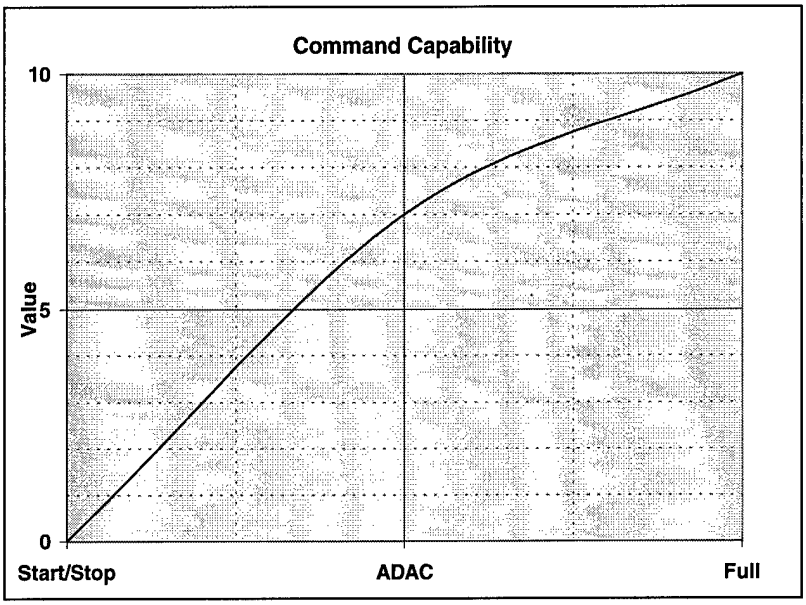
Bandwidth	High	Moderate	Low
Value	0	5	10

Figure C.7 Bandwidth Requirements Value Function

C.5 Performance

C.5.1 Bandwidth Requirements.

LEVEL	DEFINITION
High	All signals need to be sent
Moderate	Display/Command and ADACS signals need to be sent
Low	Only Display/Command updates need to be sent

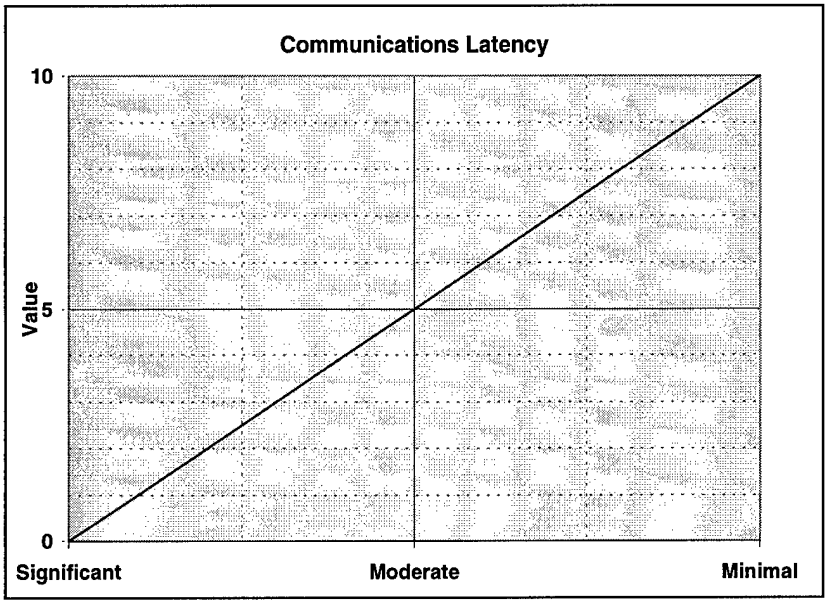


Command Capab.	Start/Stop	ADAC	Full
Value	0	7	10

Figure C.8 Command Capability Value Function

C.5.2 Command Capability.

LEVEL	DEFINITION
Start/Stop	Sat. runs independently; Grd only command start and stop
ADAC	Only Satellite attitude and direction controlled
Full	Everything (ADAC and payload) controllable from Grd

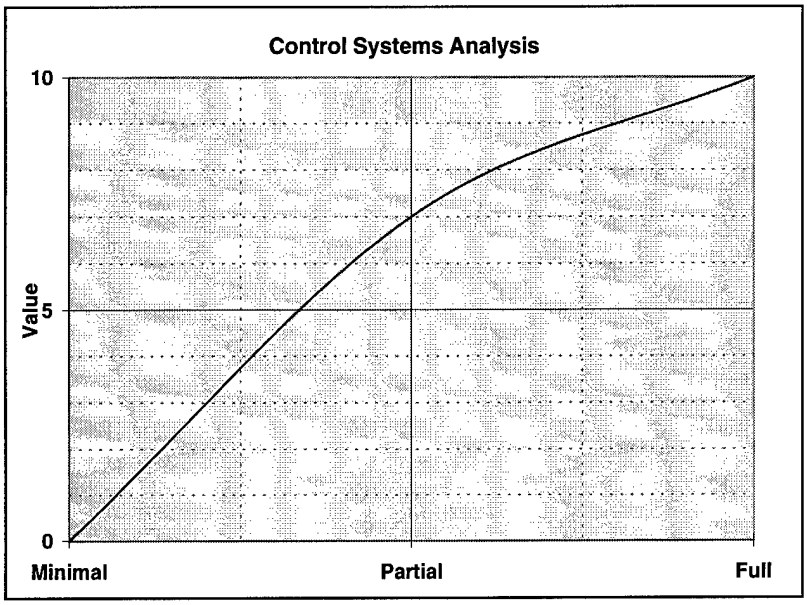


Comm. Latency	Significant	Moderate	Minimal
Value	0	5	10

Figure C.9 Communications Latency Value Function

C.5.3 Communications Latency.

LEVEL	DEFINITION
Significant	Delay impacts both inner and outer control loops
Moderate	Delay impacts only outer control loop
Minimal	Only delay is between user interface and control loop(s)

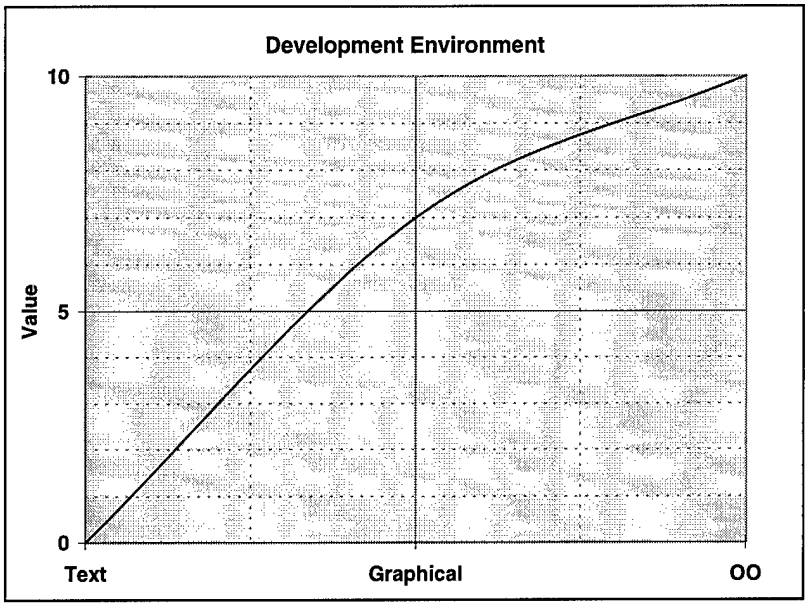


Control Sys. Analysis	Minimal	Partial	Full
Value	0	7	10

Figure C.10 Control Systems Analysis Value Function

C.5.4 Control Systems Analysis.

LEVEL	DEFINITION
Minimal	<50% of desired system elements defined or the remaining elements are difficult to define
Partial	50-90% of desired system elements defined; simple to define the rest
Full	>90% of desired system elements defined

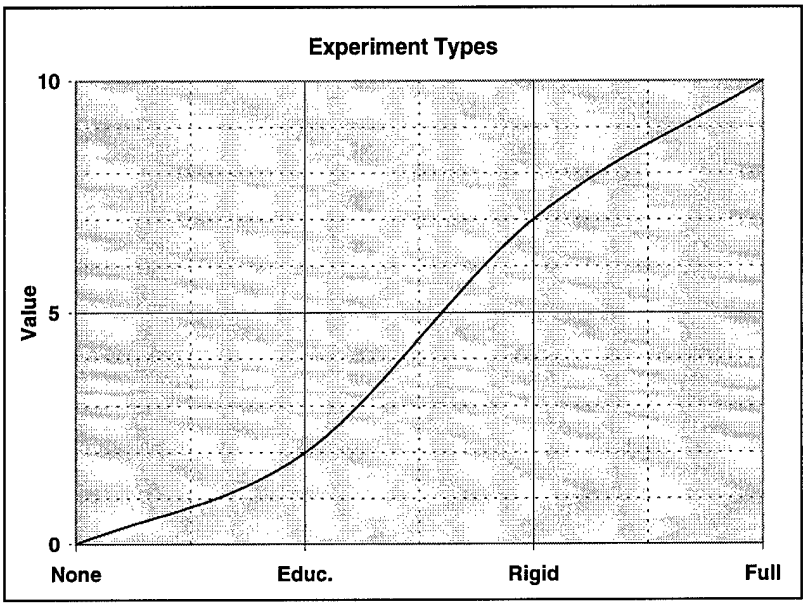


Develop. Environment	Text	Graphical	OO
Value	0	7	10

Figure C.11 Development Environment Value Function

C.5.5 Development Environment.

LEVEL	DEFINITION
Text	Time-intensive entry of control laws; prone to errors
Graphical	Not all aspects of control system available as "building blocks" but more user-friendly than *Text*
Object-Oriented	Graphical; all critical elements available as "building blocks"

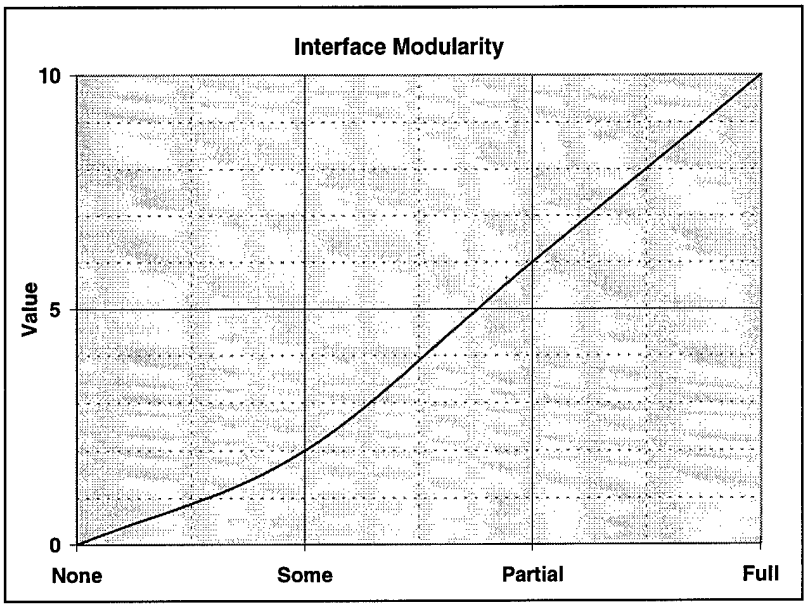


Experiment Types	None	Educ.	Rigid	Full
Value	0	2	7	10

Figure C.12 Experiment Types Value Function

C.5.6 Experiment Types.

LEVEL	DEFINITION
None	No experiments possible
Educational	Education/teaching usage only
Rigid	Can do Education and 3-Axis Rigid experiments
Full	Can do all the desired experiments

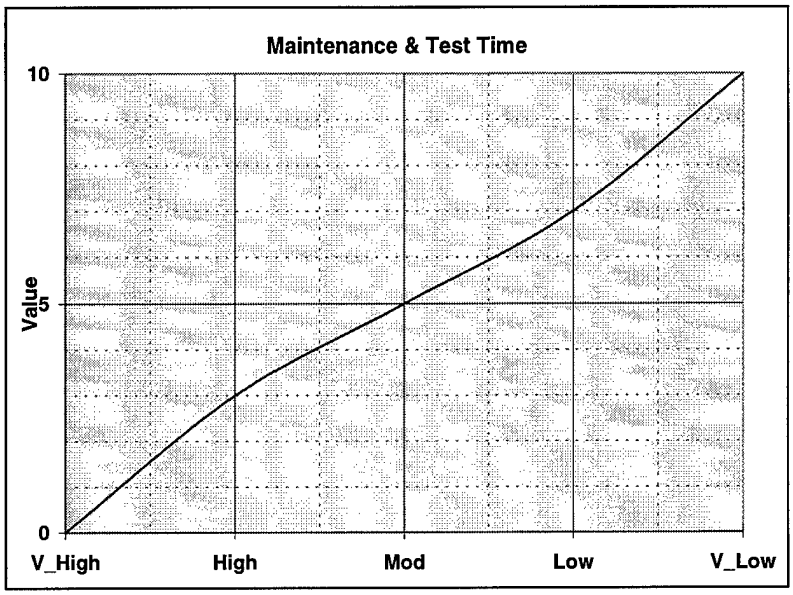


Interface Mod.	None	Some	Partial	Full
Value	0	2	6	10

Figure C.13 Interface Modularity Value Function

C.5.7 Interface Modularity.

LEVEL	DEFINITION
None	Only complete subsystems can be replaced with payload parts, not components
Some	10-50% of components/sub-sub-systems can be relocated or substituted with payload parts
Partial	50-75% of components/sub-sub-systems can be relocated or substituted with payload parts
Full	All components/sub-sub-systems can be relocated or substituted with payload parts

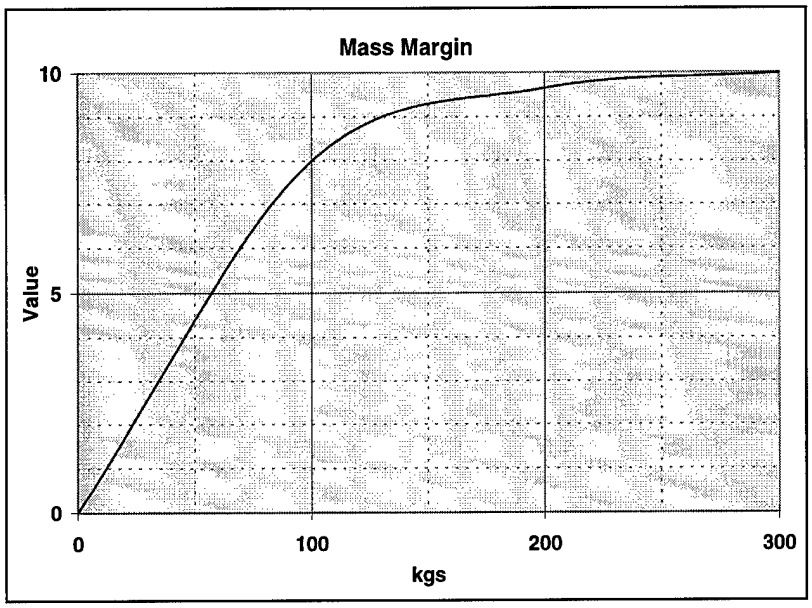


Maint & Test Time	V_High	High	Mod	Low	V_Low
Value	0	3	5	7	10

Figure C.14 Maintenance & Test Time Value Function

C.5.8 Maintenance and Test Time.

LEVEL	DEFINITION
V_High	Completely reconfigure to conduct new experiments; requires system validation before test run (> 100 min)
High	Experiment installation and validation requires 76–100 min total time
Mod	Experiment installation and validation requires 36–75 min total time
Low	Experiment installation and validation requires 16–35 min total time
V_Low	Snap-in/snap-out; experiment installation and validation requires 0–15 min total time



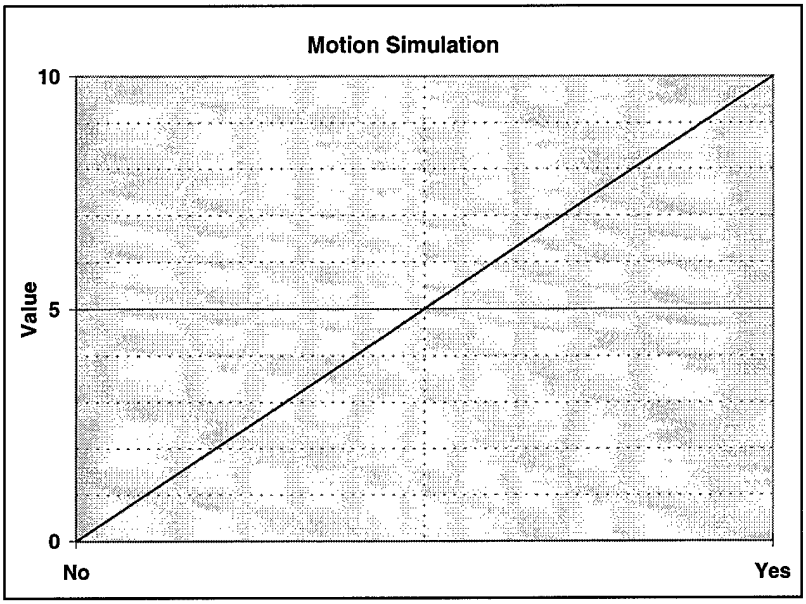
Mass Margin	0	***100***	200	300	**kgs**
Value	0	***8.0***	9.6	10.0	

FORMULA: =10.08-10.08*EXP(-0.01575*x)

Figure C.15 Mass Margin Value Function

C.5.9 Mass Margin. This measure is a continuous "direct" measure reflecting the estimated mass the air bearing assembly can support after all the required baseline *SIMSAT* components are installed. The CDM generated this function by selecting the following value comparison; LOGICAL DECISIONS generated the rest:

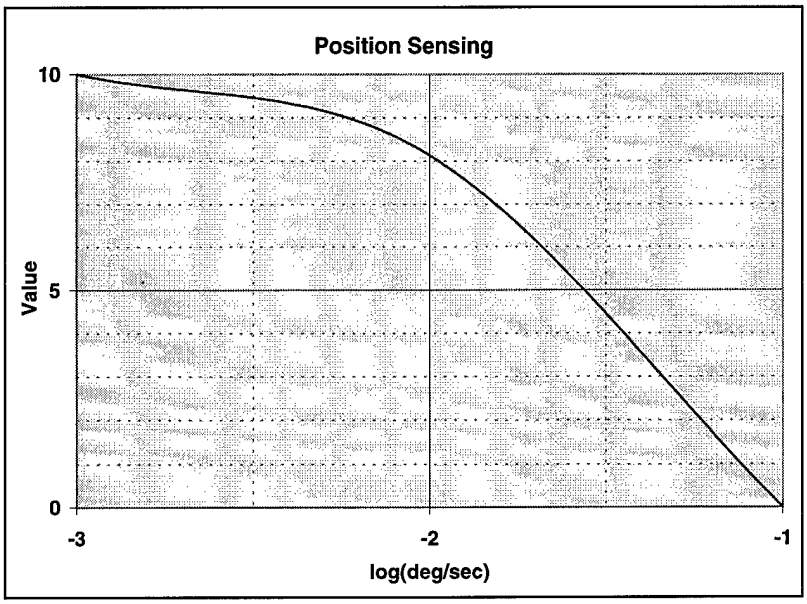
$$\textbf{Value(100 kg)} = \textbf{8}$$



Motion Simulation	No	Yes
Value	0	10

Figure C.16 Motion Simulation Value Function

C.5.10 Motion Simulation.



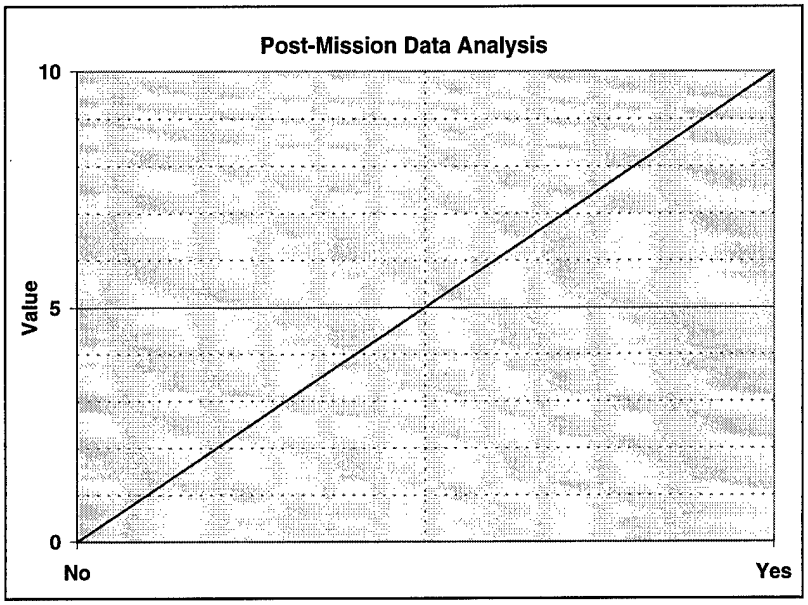
Pos. Sensing Acc.	-2.301	-3	-2	-1	log(deg/sec)
Value	9.00	10.00	8.13	0.00	

FORMULA: =10.56-45.98*EXP(1.471*x)

Figure C.17 Position Sensing Capability Value Function

C.5.11 Position Sensing Capability. This measure is a continuous "direct" measure reflecting how accurately *SIMSAT* can sense position. The CDM generated this function by selecting the following value comparison; LOGICAL DECISIONS generated the rest:

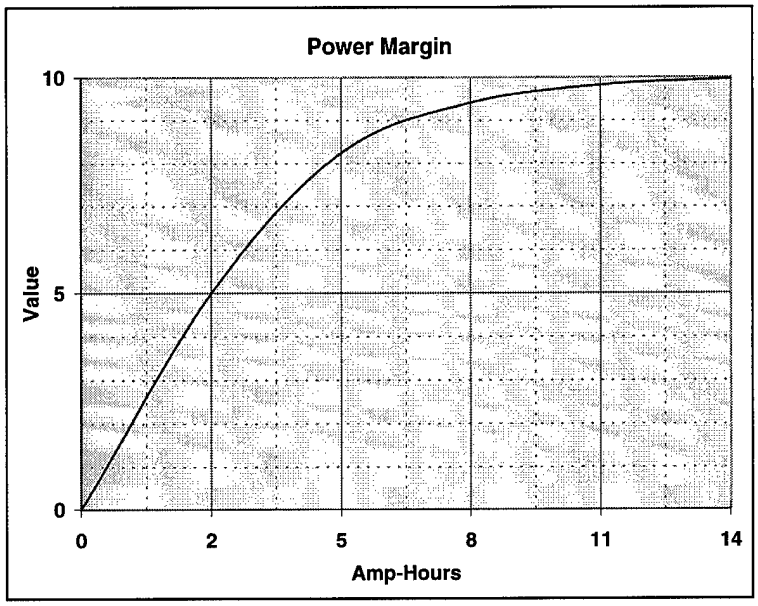
$$x = 0.005 \rightarrow \log(x) = -2.301\ldots; \mathbf{Value}(x) = \mathbf{9}$$



Post-Mission Analysis	No	Yes
Value	0	10

Figure C.18 Post-Mission Data Analysis Value Function

C.5.12 Post-Mission Data Analysis.



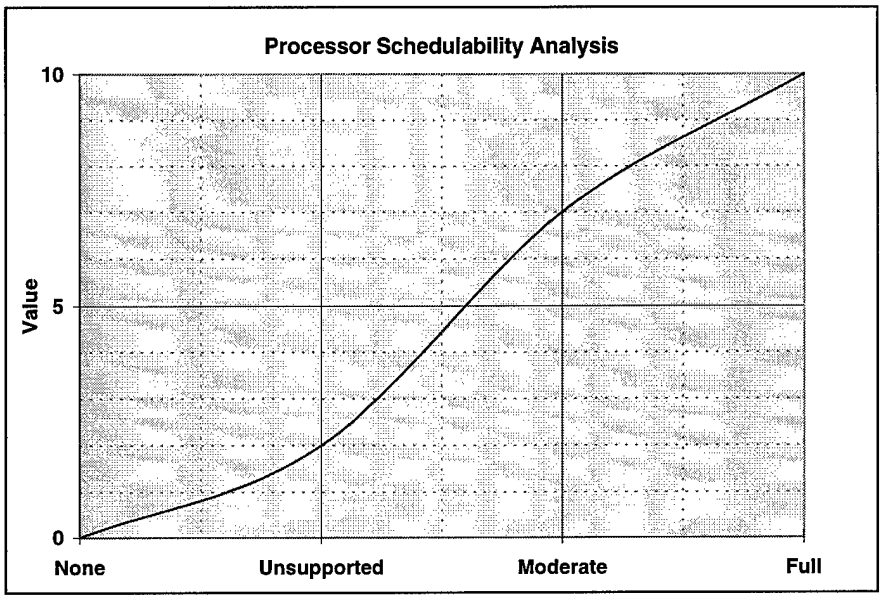
Power Margin	0	*2*	5	8	11	14	15	Amp-Hrs
Value	0.00	*5.00*	8.25	9.41	9.82	9.97	9.99	

FORMULA: =10.05 - 10.05 * EXP(-0.3437*x)

Figure C.19 Power Margin Value Function

C.5.13 Power Margin. This measure is a continuous "direct" measure reflecting the estimated power the battery system can support after all the required baseline *SIMSAT* components are installed (at the nominal system voltage). The CDM generated this function by selecting the following value comparison; LOGICAL DECISIONS generated the rest:

Value(2 Amp-hrs) = 5

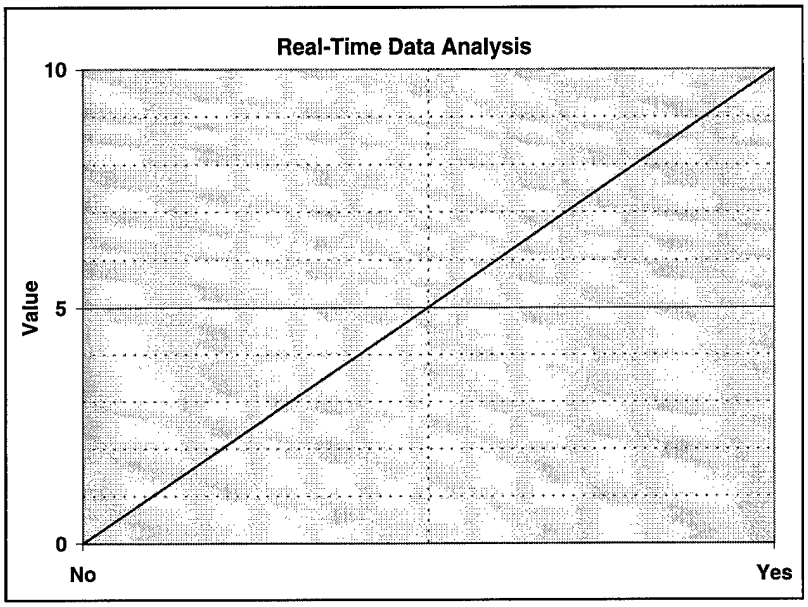


Proc. Sched. Analys.	None	Unsupported	Moderate	Full
Value	0	2	7	10

Figure C.20 Processor Schedulability Analysis Value Function

C.5.14 Processor Schedulability Analysis. See Appendix A for additional rationale on Rate Monotonic Analysis (RMA) as the scheduling technique of choice.

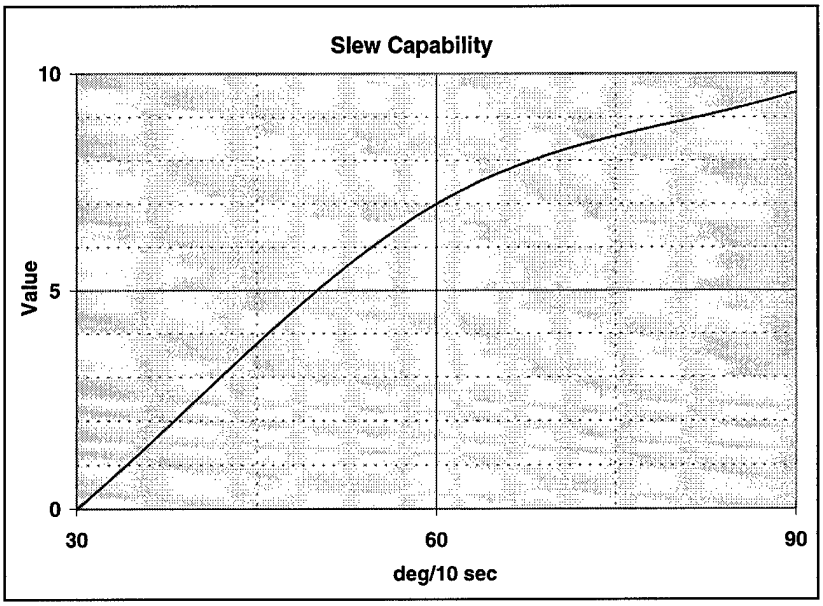
LEVEL	DEFINITION
None	RMA is not supported; insufficient data regarding OS scheduling technique to assess likelihood and impact of missed deadlines
Unsupported	RMA is not supported; but sufficient data data regarding OS scheduling technique to assess likelihood and impact of missed deadlines
Moderate	RMA supported at least indirectly; some hand-coding required to fully implement
Full	RMA supported directly in both hardware and software



Real-Time Analysis	No	Yes
Value	0	10

Figure C.21 Real-Time Data Value Function

C.5.15 Real-Time Data.



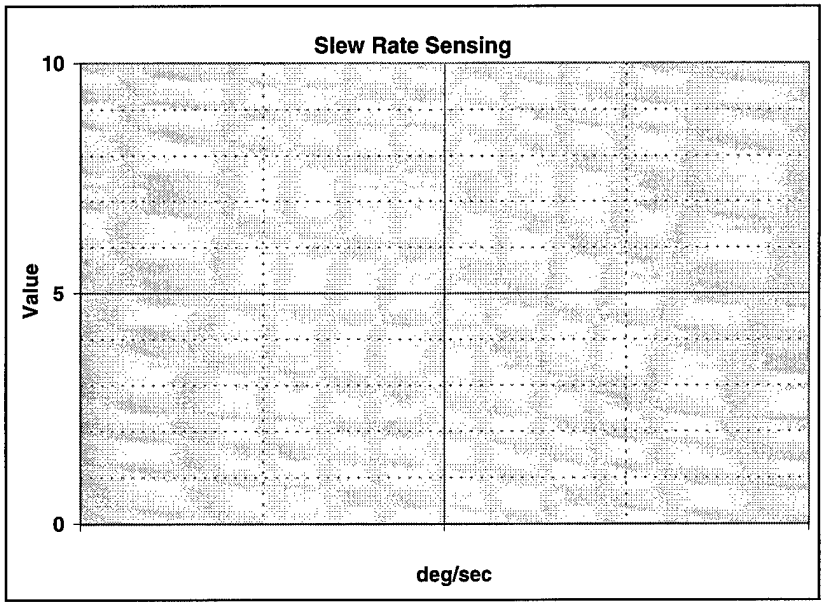
Slew Capability	30	*60*	90	100	deg/10 sec
Value	0.00	*7.00*	9.57	10.00	

FORMULA: =11.07-30.1*EXP(-0.03333*x)

Figure C.22 Slew Capability

C.5.16 Slew Capability. This measure is a continuous "direct" measure reflecting how far *SIMSAT* can slew in a 10 second timeframe. The CDM generated this function by selecting the following value comparison; LOGICAL DECISIONS generated the rest:

Value(60 degrees) = 7

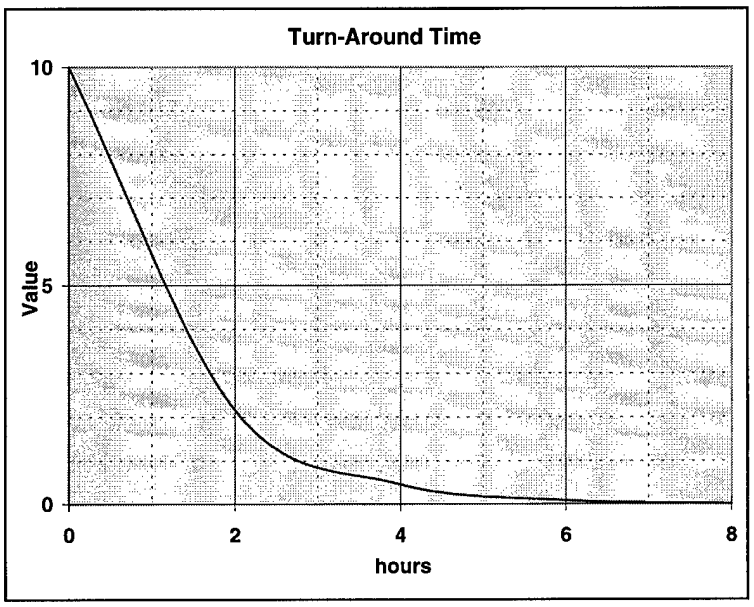


Slew Rate Sensing **deg/sec**
Value

FORMULA:

Figure C.23 Slew Rate Sensing

C.5.17 Slew Rate Sensing. Prior to the completion of the *Detailed Design* phase for the C&DH subsystem, the team could not settle on a useful definition for this measure. Faced with the choice of taking the measure out of the values hierarchy (and the corresponding LOGICAL DECISIONS models), then having to re-implement it later, the team decided to leave it in, and give the C&DH alternatives a '0' score for it. Had more ADACS issues been addressed during this design phase, this issue would have been resolved. As the choice of a C&DH subsystem really has little impact on this aspect of the system design (i.e., none of the alternatives would be distinguishable strictly on the basis of this measure), this decision would not cause any impact to the rankings of the alternatives, only their numerical score (some of the overall weight of the measures was consumed by a non-contributing measure).



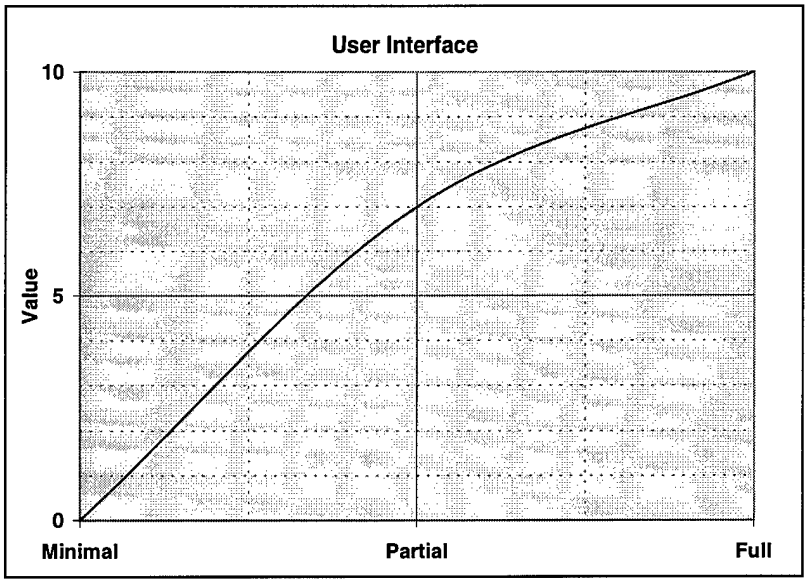
Turn-Around Time	*3*	0	2	4	6	8	hours
Value	*1.00*	10.00	2.17	0.45	0.08	0.00	

FORMULA: =-0.02279+10.02*EXP(-0.7608*x)

Figure C.24 Turn-around Time Value Function

C.5.18 Turn-Around Time. This measure is a continuous "direct" measure reflecting the worst case time between when batteries were available for experiments. The measure is impacted by both the number of spare batteries available and the recharge cycle time. For example, a single set of batteries that last 4 hours and take 8 hours to recharge would score an 8 and a value of 0 (based upon Figure C.24). Adding another set of 4 hour batteries would score a 4 for a value of 0.45, while 3 sets of batteries would allow the first set to recharge by the time the third ran down, resulting in 0 turn-time for a value of 10. The CDM generated this function by selecting the following value comparison; LOGICAL DECISIONS generated the rest:

Value(3 hours) = 1



User Interface	Minimal	Partial	Full
Value	0	7	10

Figure C.25 User Interface Value Function

C.5.19 User Interface.

LEVEL	DEFINITION
Minimal	<50% of controls and displays can be done graphically
Partial	50-90% of controls and displays can be done graphically
Full	>90% of controls and displays can be done graphically

Appendix D. Installation and Operation Manual

D.1 Overview

This appendix is organized into four sections. The first section provides detailed instructions for installing the DSPACE system as configured at the conclusion of this research. The next section adapts those procedures to explain how to generate, compile, and run experiments on the *SIMSAT* DSPACE installation. Immediately following those is a section that discusses experiment implementation philosophy. Finally, all the Windows95/NT shortcuts implemented to improve the user-friendliness of the development and control environment are shown.

D.2 DSPACE *Installation*

The following procedures were those used to prepare the DSPACE system for integration with the rest of *SIMSAT* as it became available. The implementation steps listed were written in the tone of an instruction manual to assist in re-implemenetation of the system if required. That precaution is not overly conservative since computer system stability, while certainly enhanced once implementation was complete, cannot be guaranteed: catastrophic hardware or software failures could still occur.

D.2.1 Preliminaries. Before installing hardware and software, two preliminary steps have to be taken to ensure the rest of the installation goes smoothly:

1. For the NT machine (RealMotion PC host), ensure you have administrator priviledges to install software
2. Install the execution keys (dongles) on the parallel ports of the appropriate host PCs:
   (a) Simulation PC serial number 2624
   (b) RealMotion PC serial number 2627

D.2.2 Implementing the Simulation PC/AutoBox Environment.

This is the heart of the control system and development environment for *SIMSAT*. While the steps below do not necessarily reflect the actual order used to install the dSPACE system, the order of tasks reflect the lessons learned during the installation to help others avoid our mis-steps. This consolidated installation process also reduces the need to have the individual instruction manuals available.

1. Install the dSPACE cards into AutoBox

   - NOTE: position the I/O cards as far from the power supply as possible to reduce EMI concerns [44]
   - Install the boards into the following AutoBox ISA slots (slot 0 is the Power Supply):

     1. 486 Single Board Computer (SBC) with an ethernet interface for control of the ISA bus and communication with the Simulation PC, along with a FlashDisk to store data and software (SBC pre-installed by dSPACE, Inc.) [1]

     2. DS820 Interface card for communications between the AutoBox and the RealMotion PC over dedicated serial lines (RS-422-A; high density sub-D connectors) [9]

     3. DS1003 TI C40 Digital Signal Processing Board (with 1.256M memory); the actual control system processor and Proprietary High Speed (PHS) bus master [11]

     4. RESERVED (potential future expansion slot for wireless LAN—to be determined in the companion thesis [8])

     5. empty

     6. DS2003 Multi-Channel A/D Converter I/O Board [12]

     7. DS2103 Multi-Channel D/A Converter I/O Board [14]

   - Connect the PHS (gray) ribbon cable from the DS1003 PHS connector [11:78] to the I/O boards. NOTE: the connector mated to the termination strip must be attached to the last I/O board (in this installation, the DS2103 board)

- Connect the thin-film ribbon cables from DS1003 port 0 [11:78] to DS820 plug 1 [9:3] and DS1003 port 3 to DS820 port 2 (see Figure D.1)

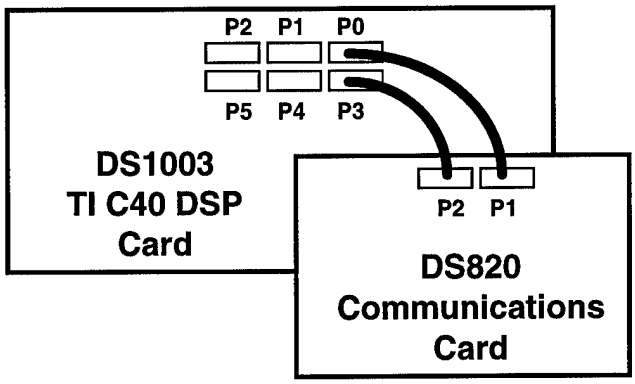


Figure D.1 Ribbon Cable Connection: DS820 to DS1003

- Mount the CP2003/2103 connection panels [13] and connect them to the corresponding AutoBox I/O cards. The ports on the rear of the AutoBox are designed so the connectors can only be installed one way; no specific description of the connectors is required. This setup will be sufficient for bench testing, but see Chapter V, page 5-6 for more details on the requirements for the satellite installation.

2. Install the DSPACE cards into the RealMotion PC (and connect to the AutoBox)

- Install the remaining DS1003 TI C40 Digital Signal processing board (512K memory) into the RealMotion PC
- Install the remaining DS820 Interface board into the RealMotion PC
- Connect the ribbon cables from the DS1003 to the DS820 as previously (again, see figure D.1)
- Connect the serial cables from the RealMotion PC DS820 to the AutoBox DS820 [9:3]

3. Power the AutoBox

   (a) fabricate a power cable with a connector wired IAW Figure D.22 (page D-27)

!!WARNING!!

Ensure the polarity is correct

(Pin A2 (Brown) and 4 (Green) to [+]

and A1 (Blue) to [-] on present cable)

   (b) connect power cable to AutoBox and power supply; VERIFY POLARITY

   (c) ensure power supply voltage is set in the 8–36VDC range

   - steady state power is $\sim$ 60W in the present configuration
   - startup power required for the DC-DC power supply is quite high ($\sim$ 240W)
   - RECOMMENDATION: set voltage to at least 20VDC—most power supplies available during this installation were current limited to 15–20 A

   (d) turn on the power suppy, then the in-line switch. If properly wired, powered, and configured:

   - the power supply should indicate a draw equivalent to $\sim$ 60W
   - the AutoBox should first sound a single beep
   - a triple beep indicates the AutoBox is prepared to communicate with the Simulation PC

4. Install the Software

   - Install MATLAB and the TI C Compiler on the Simulation PC machine before the DSPACE software. Make note of the installation directories—for the present *SIMSAT* installation:

     (a) MATLAB (ver. 5.x): `C:\MATLAB`; must include these toolboxes:

     – SIMULINK

- Real-Time Workshop

(b) TI C Compiler (ver. 5.0): `C:\C30TOOLS`

- Install the DSPACE software

(a) while not required, copying the files from the dSPACE, Inc. provided "Key Disk" to a hard drive directory (such as `C:\dSPACE.KEY`) makes the installation faster (and effectively backs up the license files). Ensure the "Key-Disk" being used matches the dongle for the machine the software is being installed on.

(b) insert the DSPACE CD-ROM into the CD-ROM drive and run `CDSETUP.EXE`

- install the DSPACE software on the same hard drive as MATLAB
- specify the path to the "Key-Disk" files as that directory created in the previous step or `A:\` (the default)
- verify the DSPACE software to be installed:

i. for the Simulation PC host, the installed software must include:

* TI C Compiler v. 4.7 (the ver. 5.0 compiler installed above so DSPACE sees a valid C Compiler installation. But ver. 5.0 has bugs, so the DSPACE software installs ver. 4.7 in its place)
* BASEN (v. 1.4.2) [18]
* Real-Time Kernel (v. 1.1) [18]
* SEMOSN (v. 1.5.1) [18]
* COCKPIT40N (v. 3.2.4) [16]
* MLIBN (v. 3.0.1) [21]
* MTRACE40N (v. 3.0.1) [22]
* RTI1003 (v. 3.1) [19]
* TRACE40N (v. 3.2.4) [17]

ii. for the RealMotion PC host, the following software must be installed:

* BASEN (v. 1.4.2) [18]
* SEMOSN (v. 1.5.1) [18]

* RealMotion (v. 1.03) [15]

iii. Once the software is finished installing,

* go to a DOS prompt
* change to the `C:\DSP_CIT\C40}`directory
* type `ADDLIB RMSERV` to complete the installation of the REALMOTION software

(c) DSPACE software configuration options used for this installation:

- installation group: "dSPACE Tools"
- single processor installations
- Simulation PC memory configuration (C40 card in AutoBox):
  * Local Bank 0: 1 MB
  * Local Bank 1: 0 MB
  * Global Bank: 256K
- RealMotion PC memory configuration:
  * Local Bank 0: 256 KB
  * Local Bank 1: 0 MB
  * Global Bank: 256K

(d) Build the Parallel Runtime Library:

- go to a DOS prompt
- change to the `C:\DSP_CIT` subdirectory
- type `mk30 -v40 --h -o2 prts40.src` [20:2]

- After the software installation is complete, open `Control Panel` and double-click the `WIBU Key` icon. If all is in order, a graphic of the dongle will appear with a set of numbers in it[1] (Figure D.2)

5. Establish the AutoBox/Simulation PC Network

[1]Some PC's may have the parallel port turned off in BIOS. If the *WIBU Key* Icon in Control Panel does not display numbers as in Figure D.2, check the BIOS setup to determine if the port is turned on or off

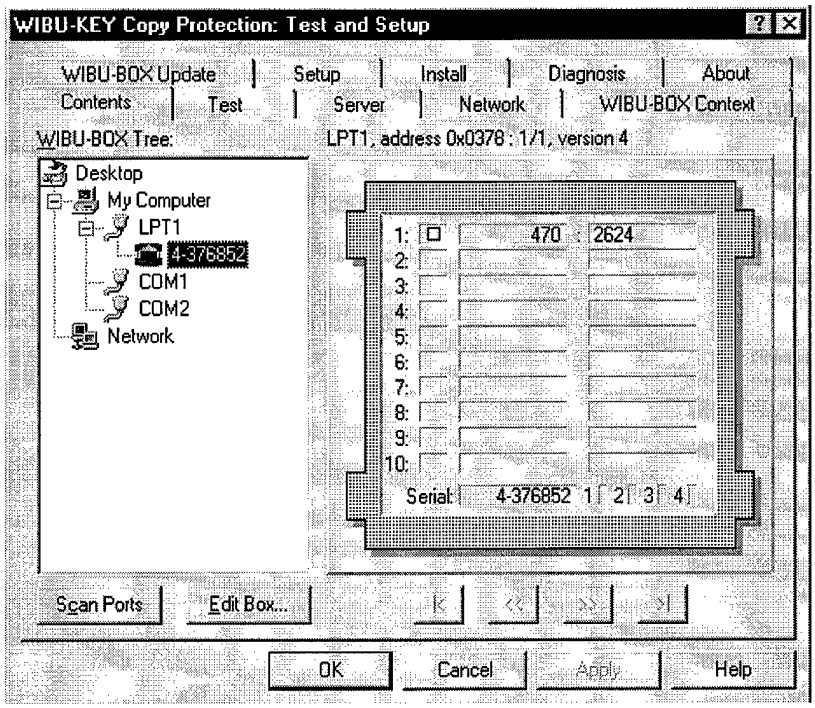


Figure D.2 WIBU-Key Dialog Box

- no concerns about I/O addresses and interrupts; cards are configured before shipment to use the defaults of AutoBox
- ethernet protocol implements a non-deterministic means of media access (the problems that can introduce into real-time control systems is addressed in Appendix A, page A-19)

!!WARNING!!

Only install the Simulation PC/AutoBox
system in a peer-to-peer configuration to prevent
control system instability (though [20:5] implies otherwise)

- Simulation PC requires connection to AFIT network. To avoid network hardware and software difficulties, use two network cards from the same manufac-

turer (one for connecting to AFIT and one for connecting to AutoBox) for the initial system installation and validation

- default AutoBox address set by dSPACE, Inc. as 192.100.100.98
- validate the networked development/command environment (ethernet between AutoBox and Simulation PC)
  (a) in `Control Panel | Network | Protocols | TCP/IP Protocol | IP Address` set the TCP/IP address for the AutoBox network card in the Simulation PC to 192.100.100.1 (disable all other protocols)
  (b) physically connect the Simulation PC and AutoBox together using the provided RJ-45 cable
  (c) set the Simulation PC card to use the appropriate connection, if required (10BaseT instead of BNC)
  (d) PING the connections to make sure the network is in place. Go to a DOS prompt and type:
    - `PING 192.100.100.1` to confirm Simulation PC network card is powered and responsive
    - `PING 192.100.100.98` to ensure AutoBox is powered and responsive to the Simulation PC
    - will see the responses in Figure D.3 if everything is working correctly. Will see a "Request timed out" reply if not
  (e) determine if dSPACE can use the network:
    - connect the Simulation PC to the AutoBox: at the DOS prompt, type `DCONTN /C 192.100.100.98`
    - if all is well, the system will respond as in Figure D.4
  (f) at the DOS prompt, type `SED40NET /b AutoBox` to find the board; the system editor will not immediately find it (Figure D.5):
    i. Choose `Search DS1003 processor board <3>` to locate the card

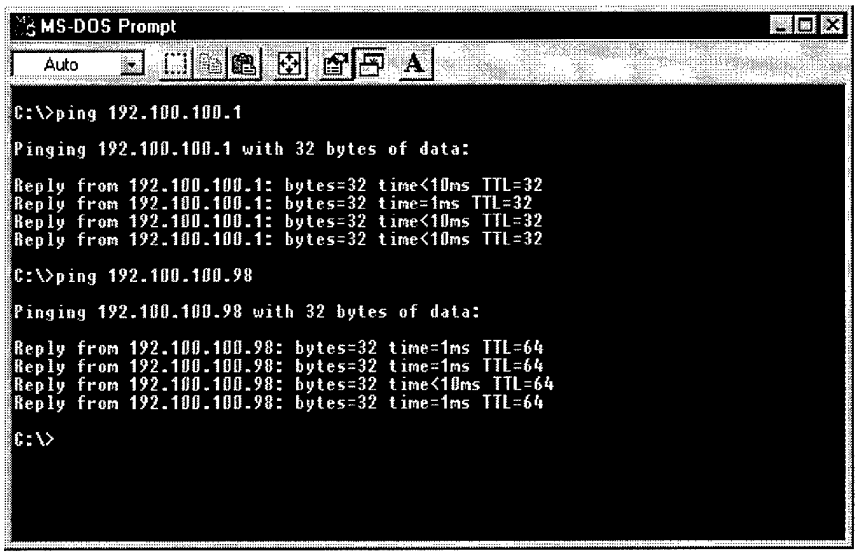


Figure D.3 PING Session Results

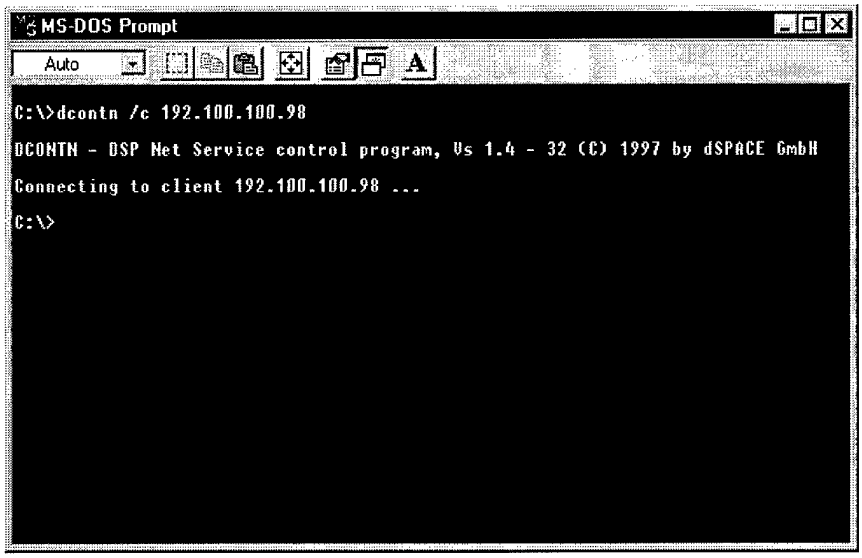


Figure D.4 Connection to AutoBox Confirmation

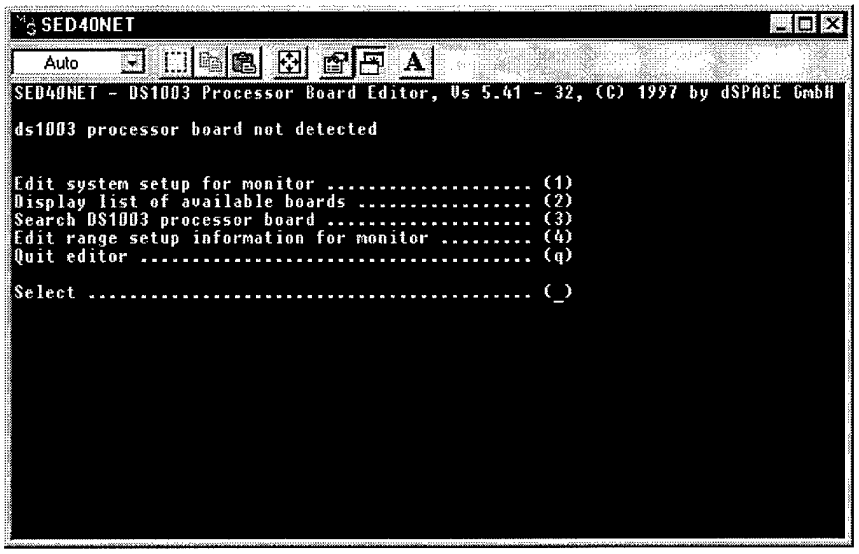


Figure D.5 Initial System Editor Menu

ii. Input the name, and memory/interrupt configurations (accept the defaults)

- Name: `AutoBox`
- Memory Offset: 0DH
- I/O Port Offset: 0318H

iii. Save the setup by choosing `Edit system setup for monitor <1>`; accept the defaults (see Figure D.6)

iv. Choose `Display list of available boards <2>`; editor should find the I/O boards (Figure D.7):

- DS2003 (20H offset)
- DS2103 (90H offset)

v. Quit the System Setup by typing `<q>` once

vi. Quit the System Editor and return to the DOS prompt by typing `<q>`

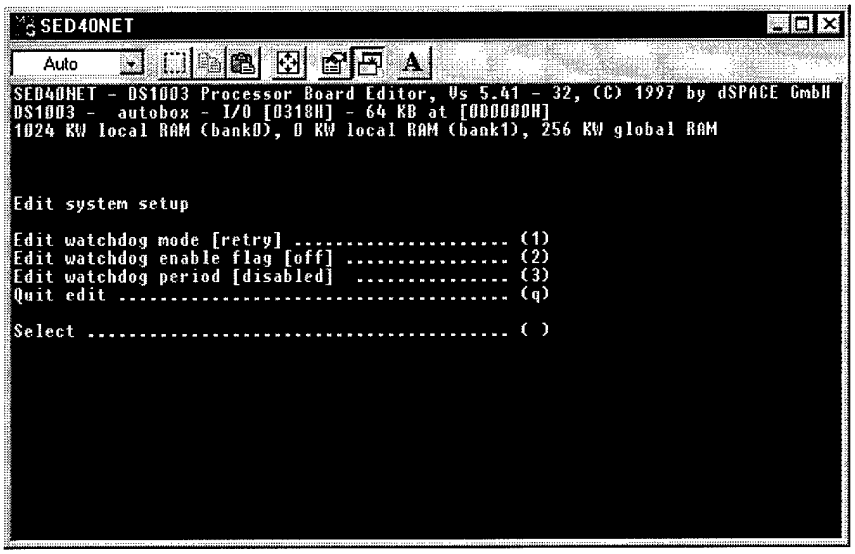


Figure D.6 Edit System Setup Menu

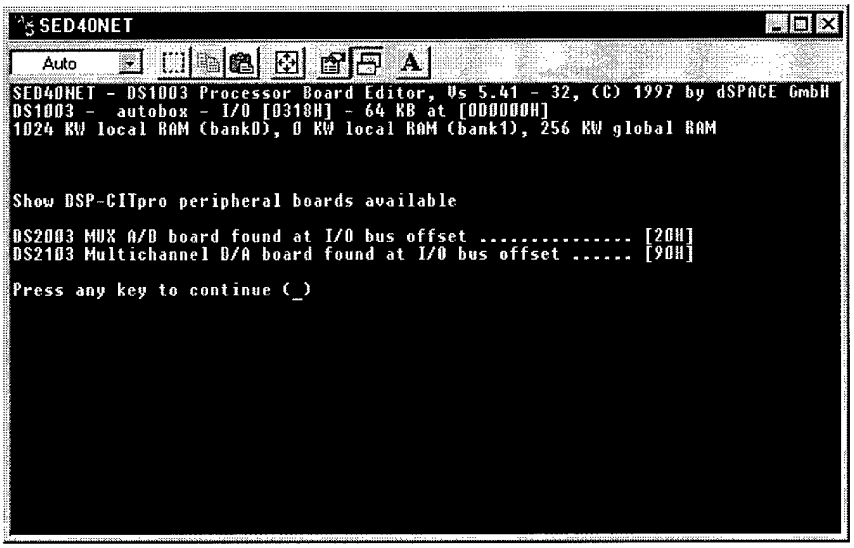


Figure D.7 I/O Board Search Results

6. Setup the Local RealMotion PC Processor

   (a) at a DOS prompt, type `SED40 /b RealMot`; the system editor will again not immediately find it (similiar to Figure D.5):

      i. Choose `Search DS1003 processor board <3>` to locate the card

      ii. Input the name, and memory/interrupt configurations (accept the defaults)

         - Name: `RealMot`
         - Memory Offset: 0DH
         - I/O Port Offset: 0318H

      iii. Save the setup by choosing `Edit system setup for monitor <1>`; accept the defaults (see Figure D.6)

      iv. Choose `Display list of available boards <2>`; editor should not find any I/O boards

      v. Quit the System Setup by typing `<q>` once

      vi. Quit the System Editor and return to the DOS prompt by typing `<q>`

At this point the system installation is complete. A set of simple models to validate the system will be used in the next section to validate the system and provide a short tutorial on operating the software. Fortunately the complicated steps listed above were only required until the team developed tools to take advantage of the Windows95 graphical interface. The next section will also demonstrate those tools developed to ease the experimenter's workload.

D.2.3 AutoBox/Simulation PC Operation Demonstration. This section will use a short example to demonstrate how to use the AutoBox/Simulation PC portion of the system (the next section will cover the RealMotion PC validation). Section D.3 (the Operator's Manual starting on page D-26) should be usedto for more detailed information on how to develop and implement a *SIMSAT* experiment using the AutoBox.

The following example assumes the system has not been turned off or disconnected from the AutoBox since completing the tasks in the previous section. To re-accomplish

those tasks, see the steps spelled out in Section D.3 (page D-26). To begin the demonstration, double-click on the `dSPACE Files` icon on the desktop (all the files contained in that folder are detailed in Section D.5, page D-35). Figure D.8 will appear, which has all the shortcuts necessary to build, compile and load, monitor, and control an experiment. Each of these will be demonstrated in a following subsection.

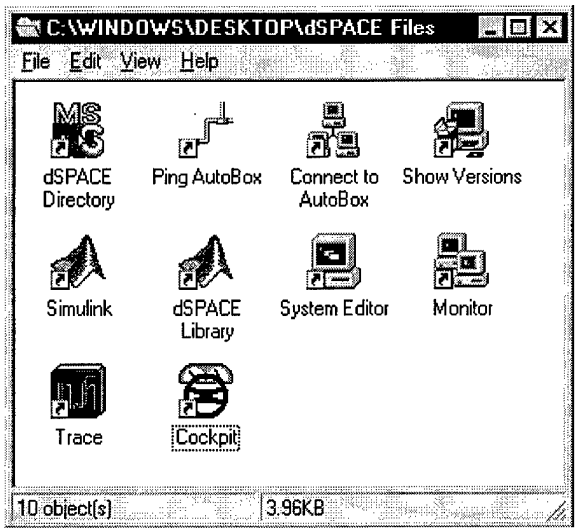


Figure D.8 `dSPACE Files` Folder Contents

D.2.3.1 Building the Model. Double-clicking on the `dSPACE Library` choice will start MATLAB and load the Real-Time Interface library for the dSPACE components currently installed (Figure D.9). Double-clicking the `DEMO2` button will bring up a set of choices, including 2 pre-defined demonstration models for the dSPACE system (Figure D.10): one developed by dSPACE, Inc., the other adapted for the particular I/O boards currently installed (`PT2 with IO`). The latter will be used for this demonstration.

Double-clicking `PT2 with IO` opens up the demonstration SIMULINK model (Figure D.11), enhanced with dSPACE building blocks for the installed I/O boards. This model has been previously validated, so all that needs to be done is compile and load the model onto the DSP. See Appendix D, page D-29, for more information on building a model from scratch using traditional SIMULINK blocks.

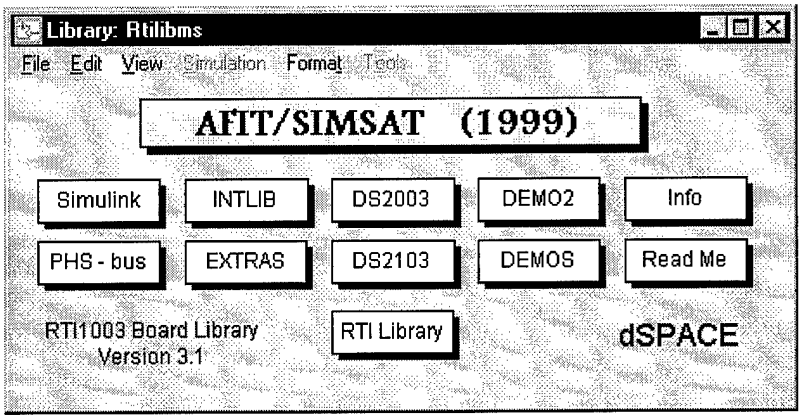


Figure D.9 dSPACE Library Menu

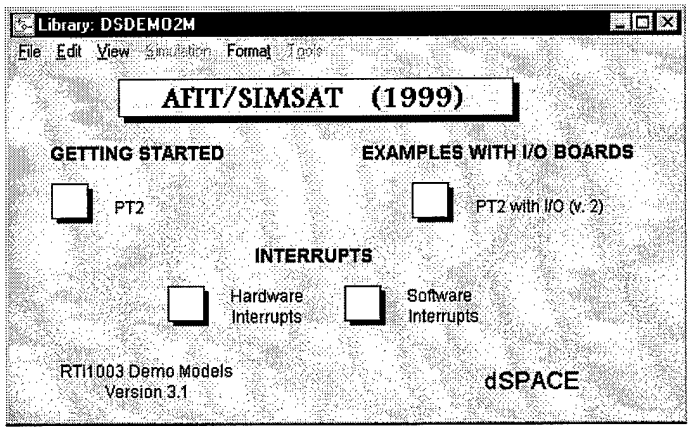


Figure D.10 DEMO2 Library Menu

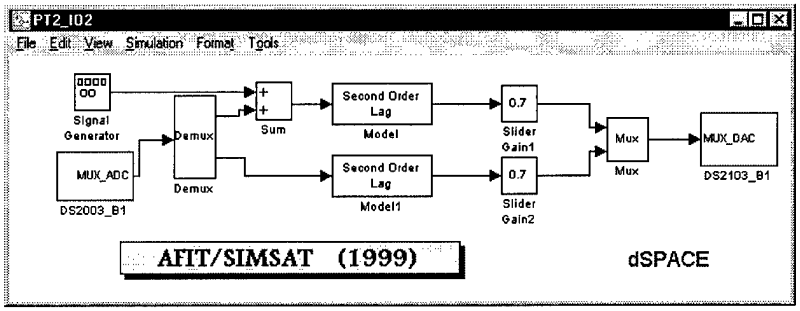


Figure D.11 PT2 with IO SIMULINK Model

D.2.3.2 Compile and Load the Model. Before compiling the model, verify the `Tools | RTI Options | RTW` dialog box has been updated so the `Make Command` entry reflects the *AutoBox* board name (as in Figure D.12 [19:51]).

After validating the `RTI Options` have been changed, compile the model by choosing `Tools | RTI Build`. The process of compiling, linking, and loading can be observed by opening the MATLAB command window where error messages will be displayed. If everything works correctly, the following words will appear as the compilation proceeds:

```
*** Starting RTI build procedure with RTI1003 3.1 (28-Apr-1998)

### Starting RTW build procedure for model: PT2_IO2
### Invoking Target Language Compiler on PT2_IO2.rtw
tlc -r PT2_IO2.rtw c:\dsp_cit\matlab\rti1003\tlc\rti1003.tlc -O.
-Ic:\dsp_cit\matlab\rti1003\tlc
-Ic:\dsp_cit\matlab\rti1003\sfcn\ml5_2\
-Ic:\dsp_cit\matlab\rti1003\sfcn\ml5_2\
-IC:\MATLAB\rtw\c\tlc -aInlineParameters=0
```

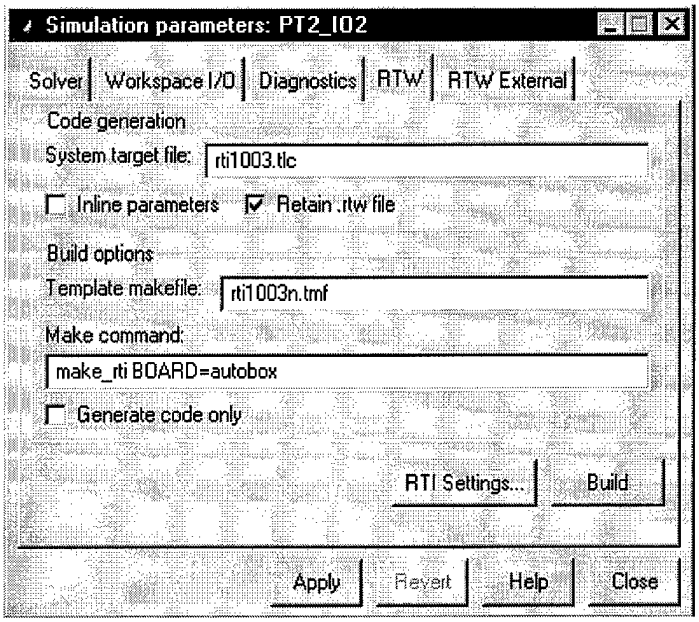


Figure D.12 `RTI Options` (Board Name)

```
### Creating project marker file: rtw_proj.tmw
### Creating PT2_IO2.mk from rti1003n.tmf
### Building PT2_IO2: dsmake -f PT2_IO2.mk BOARD=autobox

BUILDING PROGRAM (single timer task mode)

Initial SimState: default (defined in srtframe.c)

[srtframe.c]
[pt2_io2.c]
[rt_sim.c]
[ode1.c]

LINKING PROGRAM ...

LOADING PROGRAM ...


MON40NET - DS1003 Processor Board Monitor, Vs 5.4 - 32, (C) 1997 by dSPACE GmbH

DS1003 -  autobox - I/O [0318H] - 64 KB at [0D0000H]
1024 KW local RAM (bank0), 0 KW local RAM (bank1), 256 KW global RAM
```

```
Searching DS1003 peripherals ...
Loading system setup ...

Loading setup PT2_IO2.stp ...
Loading object module PT2_IO2.obj ...

DSP started ...

DOWNLOAD SUCCEEDED

### Successful completion of RTW build procedure for model: PT2_IO2

*** Finished RTI build procedure for model PT2_IO2
```

The model is now loaded and running, ready to be monitored and controlled using TRACE and COCKPIT.

D.2.3.3 Monitoring and Controlling the Model. For a control system application such as *SIMSAT*, a program running on the DSP in the AutoBox has little use if the user has no ability to observe and control the process. That is where the DSPACE TRACE and COCKPIT packages come into play.

TRACE ***Initialization.*** Continuing with the previous model, and assuming the AutoBox DSP is loaded and running (based upon the procedures from the previous section), TRACE is loaded first. Double-clicking the TRACE icon in the `dSPACE Files` folder opens the TRACE control window, which opens with the last file used by TRACE. To load the `.trc` file of interest, use the `File | Load Trace File` to locate the `PT2IO2.TRC` file to support the program currently running on the AutoBox. Figure D.13 shows what the control panel looks like when the correct file is loaded.

At the same time the control window for TRACE is opened, a window to display a trace of the desired signals is opened. Once the experiment for `PT2 with IO` is loaded, and the `START` button is pressed in the control window, the plots in the trace window begin to display real-time data. Figure D.14 shows the input signal as well as the output signal from the top `Second Order Lag` shown in Figure D.11.

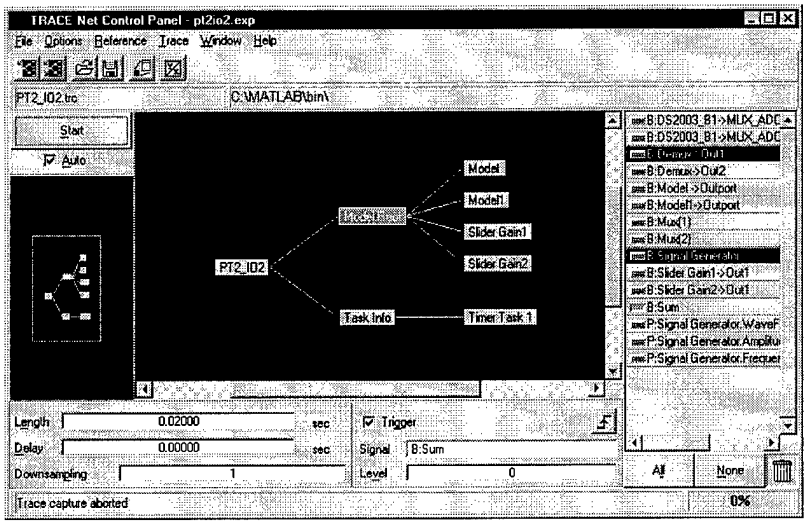


Figure D.13 TRACE Control Window—PT2IO2 Loaded

Figure D.14 TRACE Plot Window—PT2IO2 Running

Just seeing the response of the system without changing parameters to see how the response changes does not fully demonstrate the dSPACE system is ready for integration into the *SIMSAT* system. Once the TRACE model is running with the initial values of the various model parameters, COCKPIT can be started to control certain pre-selected parameters.

COCKPIT ***Initialization.*** COCKPIT provides the means to control the application running on the AutoBox. This section validates our installation using the same model as in the previous section. This validation assumes the model is still running on the AutoBox, and TRACE is still active. Double-clicking the COCKPIT icon in the `dSPACE Files` folder opens the COCKPIT window, which is initially blank. If previously defined, a `.ccs` file can then be opened (`File | Open`) which loads the corresponding `.trc` file, as well as any previously defined controls for that program. If no `.ccs` file has been previously saved, the TRACE file must first be loaded, then the desired controls can be defined. To load the `.trc` file of interest, use `File | Load Trace File` to locate the file; `PT2I02.TRC` in this case, to match the file currently in use by TRACE for this demo. Figure D.15 shows what a simple cockpit control (a slider for the top `Gain` block in Figure D.11) looks like.

As the controls and displays are placed on the user interface "form," the parameter they control is established by double-clicking on the control or display, and selecting the `P:` parameters for controls, and the `B:` parameters for the displays. Once the design is complete, and saved, it is ready to run. Clicking the `Start` button links the COCKPIT controls to the program executing on the AutoBox, and displayed in TRACE. For this simple control, the display changes appearance (editing is not possible when COCKPIT is connected to the DSP—see Figure D.16).

Moving the slider all the way to the value shown in Figure D.16 impacts the TRACE outputs (Figure D.17—the amplitude of the signal drops, as expected). This validates the entire AutoBox/Simulation PC installation. It is now ready for integration with the remainder of the *SIMSAT* system. The AutoBox/RealMotion PC installation still needs to be validated, but it is not a requirement to monitor and control *SIMSAT* (it provides for the real-time and post-mission display of data).

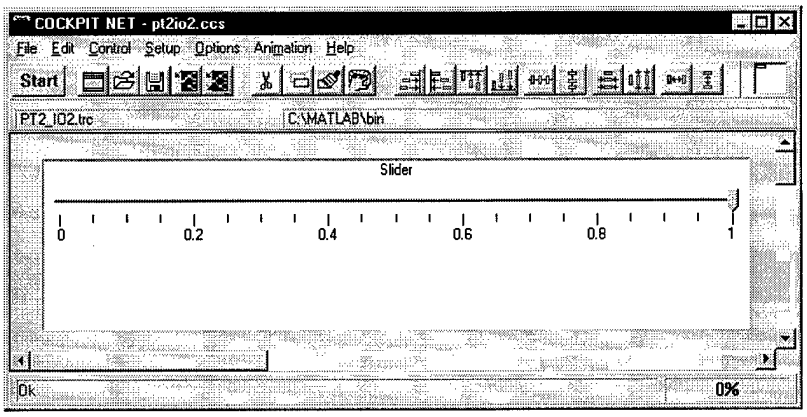


Figure D.15 COCKPIT Window—PT2IO2 Loaded

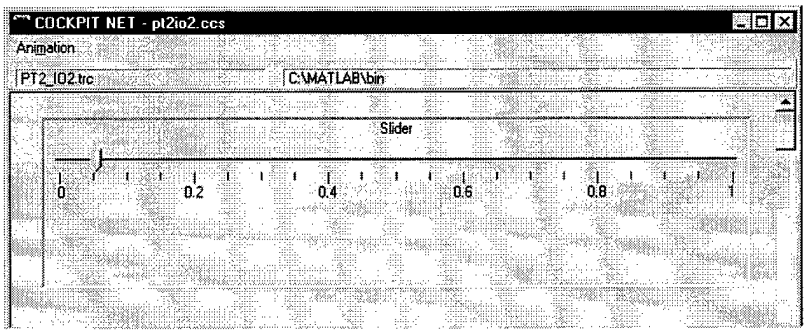


Figure D.16 COCKPIT Window—PT2IO2 Running

Figure D.17 TRACE Plot Window—COCKPIT Adjusted Output

This simple implementation only validated that COCKPIT could be used to access the program running on the AutoBox. Further information regarding the COCKPIT interface and its capabilities can be found in Section D.3.5, page D-32.

D.2.4 RealMotion PC Operation Demonstration. This section will use a short example to demonstrate how to use the RealMotion PC portion of the system. The following example does NOT assume the system has previously loaded the correct model into the AutoBox since this program uses a different model for validation and does not run on the Simulation PC. This section describes how to link a REALMOTION simulation to a program running on the local C40.

D.2.5 Initialize RealMotion PC DSP. To begin the demonstration, double-click on the `dSPACE Files` icon on the desktop of the WindowsNT machine (again, all the files contained in that folder are detailed in Section D.5, page D-35). Figure D.18 will appear, which has all the shortcuts necessary to build, compile and load, monitor and control an experiment. Each of these will be demonstrated in a following subsection.

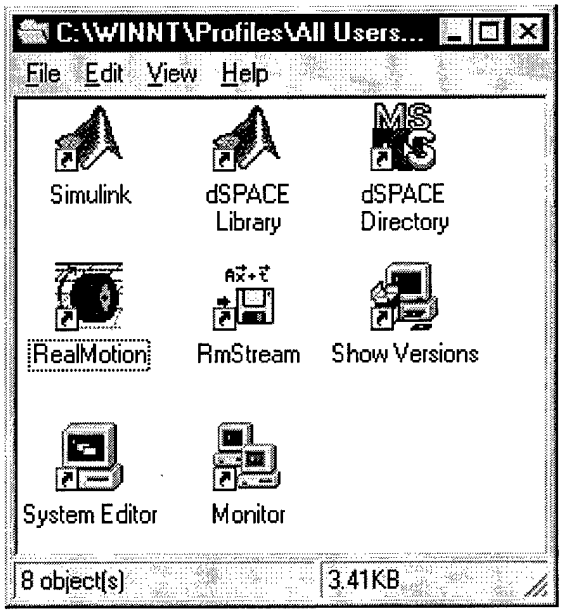


Figure D.18 `dSPACE Files` Folder Contents (NT Machine)

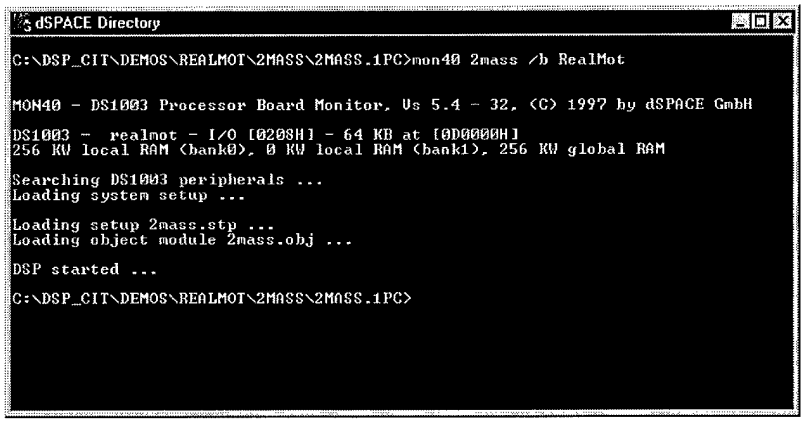


Figure D.19 Loading 2MASS Program

D.2.5.1 Loading the File. Before the REALMOTION system can link to the DSP, the DSP has to have the corresponding file already executing on the DSP. As the demo model (`2MASS`) has already been compiled, it just needed to be loaded onto the RealMotion PC DSP. This was accomplished by double-clicking on the `dSPACE Directory` and changing to the directory containing the `2MASS` object file. Loading the object file was then accomplished by typing `MON40 2MASS -B RealMot`. The results are shown in Figure D.19; the file successfully loaded onto the DSP and the DSP began executing the file (essentially the same response as that documented in the *AutoBox/Simulation PC Operation Demonstration*) section.

Double-Clicking the `RealMotion` icon opens up the dual REALMOTION windows. The first, graphical window, initially appears blank. Upon loading the `2MASS` model (Figure D.20), the pieces start to move in the off-line mode (simulating the motion using the RealMotion PC microprocessor). The second window is a status window that, much like the MATLAB command window, provides a textual record of user initiated changes made to the graphical window (re-sizing, changing the processing mode, etc.). Figure D.21 shows

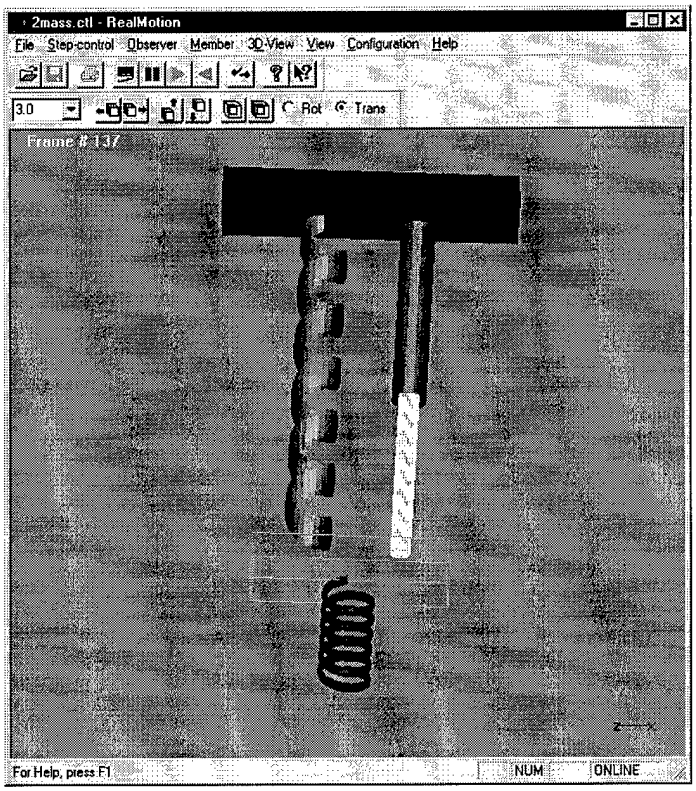


Figure D.20 REALMOTION Graphical Window

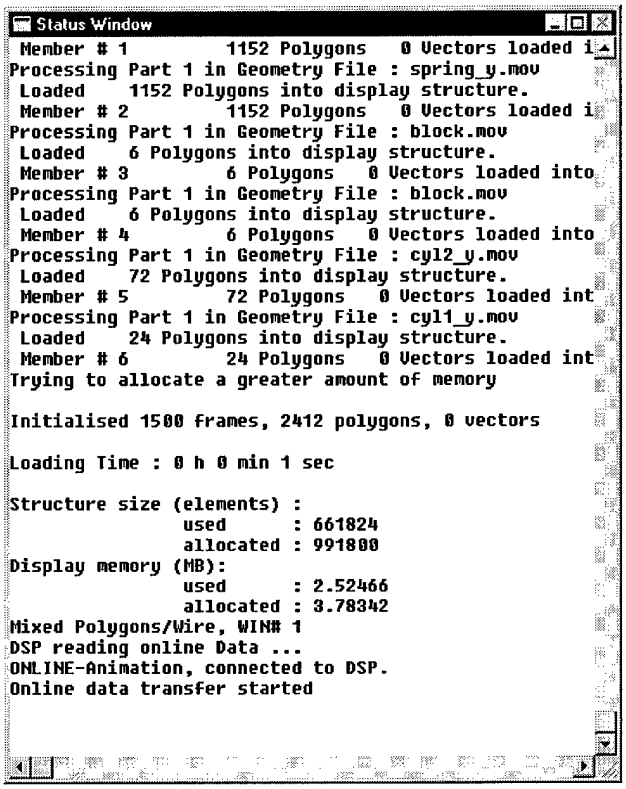


Figure D.21 REALMOTION Status Window

the status window after the 2MASS model is loaded, and REALMOTION is switched to the on-line mode.

As Figure D.21 shows that REALMOTION switched to the on-line mode, this completes the validation of the RealMotion PC local processor installation. Difficulties with the disparate Simulation PC and RealMotion PC installations prevented any further validation of the RealMotion PC subsystem. Prior to using REALMOTION for any *SIMSAT* experiments required the 3-D representations from REALMOTION, the rest of the installation must be validated.

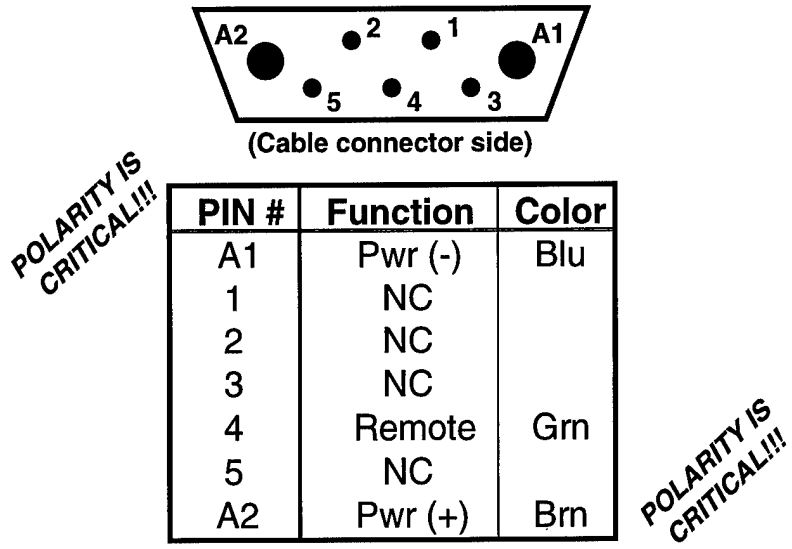
D.2.5.2 Summary. This section described how to install the system (in case it had to be done again in the future), including how to implement a very simple, pre-defined model on both the Simulation PC and the RealMotion PC. Implementation of those models served to validate the installation as well as make use of the shortcuts documented in Section D.5, starting at page D-35.

D.3 Operations Manual

This section adapts the installation instructions of the previous section into the procedures required to build a model, generate the source code, compile the program and load it onto the AutoBox, and control the system.

D.3.1 Power the AutoBox.

1. Make sure the in-line switch is in the OFF position
2. Verify the power connection is consistent with the Figure D.22 [ENSURE pins A2 and 4 are connected to the positive terminal and terminal A1 is connected to the return side of the power bus (negative)]
3. Turn on the power supply and make sure the voltage is set to at least 20VDC to reduce the required startup current
4. Turn on the AutoBox using the in-line switch



PIN #	Function	Color
A1	Pwr (-)	Blu
1	NC	
2	NC	
3	NC	
4	Remote	Grn
5	NC	
A2	Pwr (+)	Brn

Figure D.22 Power Cable Interface

5. If the AutoBox beeps once, followed shortly afterward by three beeps, the AutoBox is ready to operate

D.3.2 Connect to the AutoBox.

1. Open the `dSPACE Files` folder from the Windows95 desktop (Figure D.23)
2. Verify the AutoBox is ready to communicate by double-clicking the `Ping AutoBox` icon in the folder
3. If do not get a `Request Timed Out` response, AutoBox is ready to communicate [20:9]
4. Double-click on the `Connect to AutoBox` icon in the folder (Figure D.23); the screen will flash and quickly return to Windows95 interface, indicating the connection is complete
5. To validate the connection, double-click the `System Editor` icon; if the connection is valid, the screen will appear as in Figure D.24 (the `autobox` notation on the second line, after the 'DS1003' label)

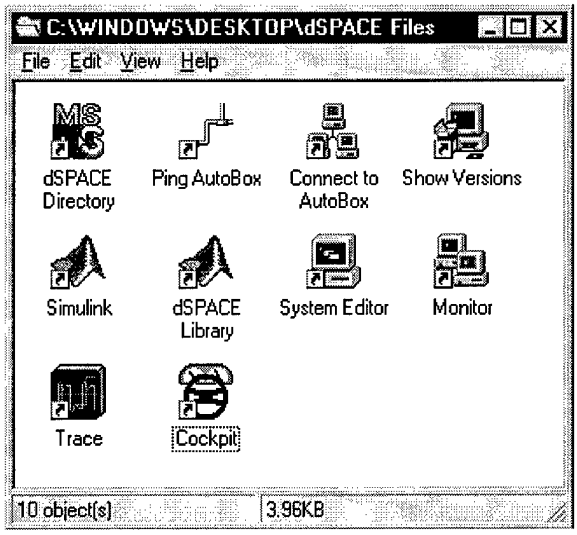


Figure D.23 `dSPACE Files` Folder Contents

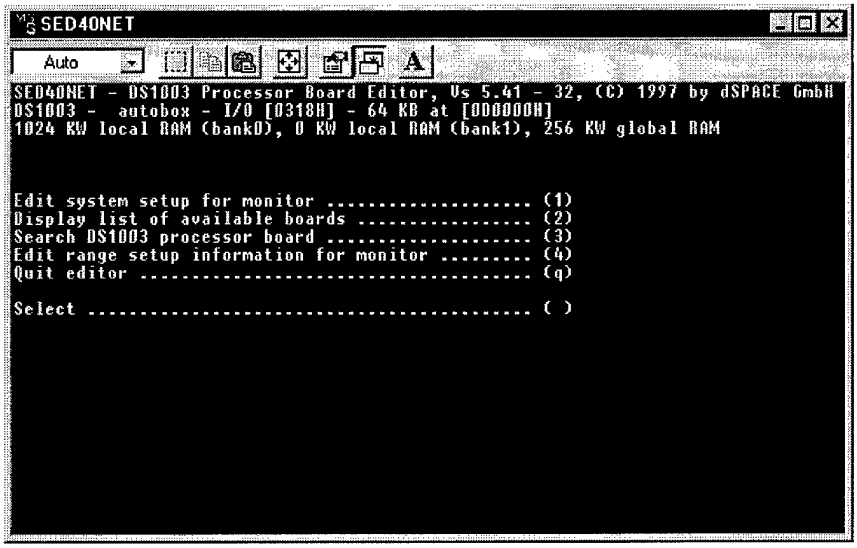


Figure D.24 Connected System Editor Menu

The next step is required only if the model of interest has not been previously complied, or must be updated before being loaded. If an executable image of the subject model has been developed, proceed directly to Section D.3.4.

D.3.3 Build and Compile the Model.

1. From the `dSPACE Files` folder (Figure D.23), double-click `dSPACE Library` icon; Figure D.25 will appear

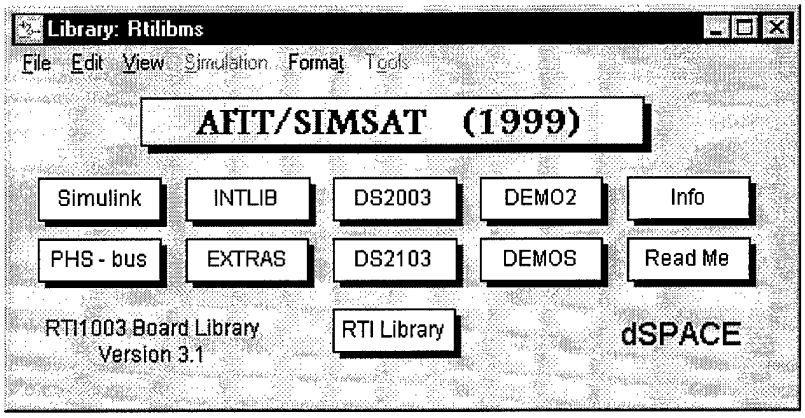


Figure D.25 `dSPACE Library` Menu

2. Double-click on the SIMULINK button to open the SIMULINK models interface; build (or open) your model as required
3. Before compiling the model, ensure the `Tools` | `RTI Options` | `RTW` dialog box appears as in Figure D.26 (notably the `Make Command` choice reflects the AutoBox board name [19:51])
4. Compile and load the model (called `XXX` for this example) by choosing `Tools` | `RTI Build`. The process can be observed by opening the MATLAB command window

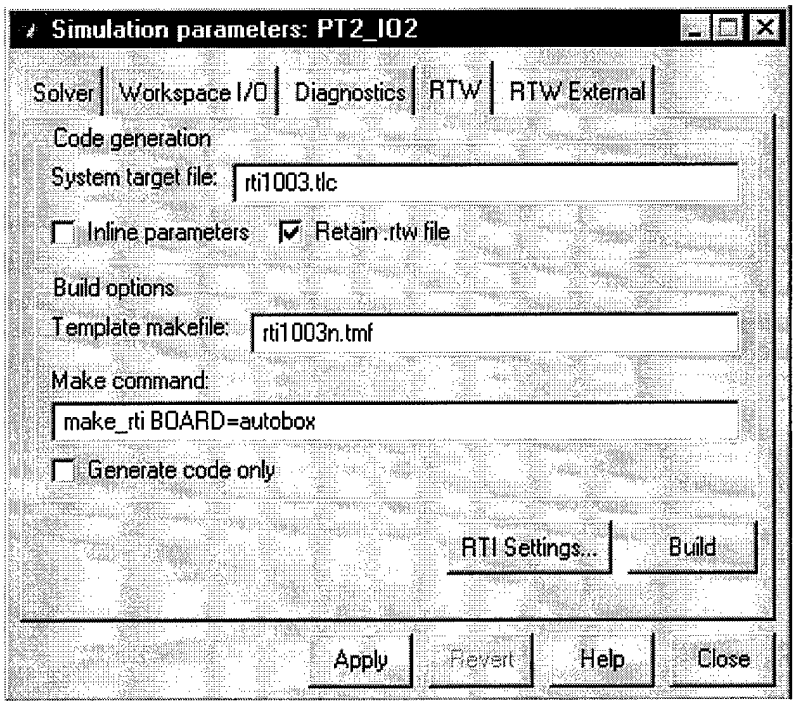


Figure D.26 `RTI Options` (Board Name)

where error messages will be displayed as problems occur. If the compilation and load process are flawless, the last three lines of output will say something like:

```
DOWNLOAD SUCCEEDED

### Successful completion of RTW build procedure for model: XXXX

*** Finished RTI build procedure for model XXXX
```

5. The model is now loaded and running, ready for control (COCKPIT) and monitoring (TRACE).

D.3.4 Load the Model on the AutoBox.

1. From the `dSPACE Files` folder (Figure D.23), double-click `Monitor` icon; Figure D.27 will appear

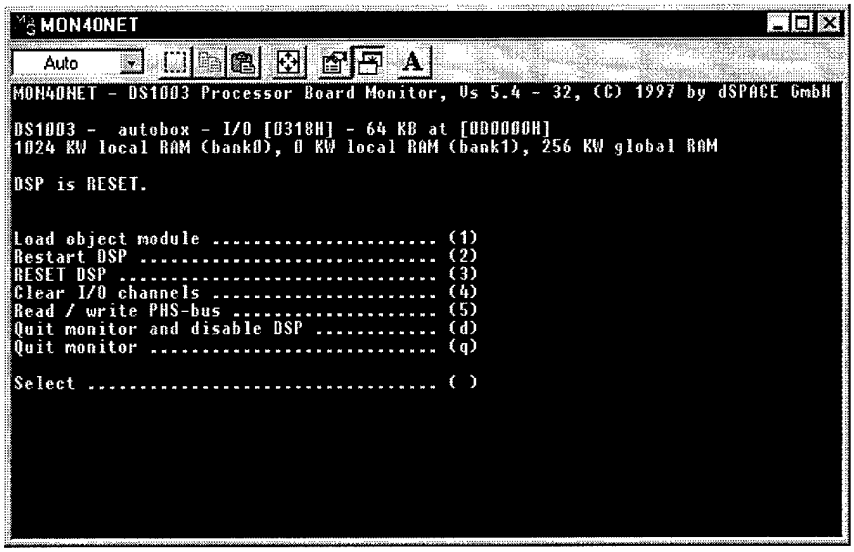


Figure D.27 dSPACE Monitor Menu

2. Choose option `Load Object Module <1>` and input the `.OBJ` file (full pathname) to send to the AutoBox (`XXXX` for this example)

3. The process can be observed by opening the MATLAB command window where error messages will be displayed, or the following words if compilation was successful:

```
MON4ONET - DS1003 Processor Board Monitor, Vs 5.4 - 32, (C) 1997 by dSPACE GmbH

DS1003 -  autobox - I/O [0318H] - 64 KB at [0D0000H]
1024 KW local RAM (bank0), 0 KW local RAM (bank1), 256 KW global RAM

Searching DS1003 peripherals ...
Loading system setup ...

Loading setup XXXX.stp ...
Loading object module XXXX.obj ...

DSP started ...

DOWNLOAD SUCCEEDED
```

```
### Successful completion of RTW build procedure for model: XXXX

*** Finished RTI build procedure for model XXXX
```

4. The model is now loaded and running, ready for control (COCKPIT) and monitoring (TRACE).

TRACE and COCKPIT are DSPACE software packages that provide the user access to the executing program to see the results of certain operations and control aspects of the program execution, respectively. When a program is compiled, using the steps above, a `.TRC` file is created that contains the information these programs need to tie into the program executing in the AutoBox. Rather than repeating the TRACE and COCKPIT procedures here, refer to the procedures specified in Section D.2.3.3 (page D-17) on how to link TRACE and COCKPIT to the program running on the DSP.

D.3.5 COCKPIT ***Capabilities.*** While only a simple interface was "designed" in previous sections, COCKPIT can be used is used to construct a very robust, user-friendly interface to the contol system. Besides readily accessible form design tools (i.e., size and alignment of controls, grouping of controls, fonts and colors, etc.), COCKPIT provides a wide variety of controls and displays, including:

- ***Input Controls*** [16:85–93]
  - **Button Array**—a user-designed matrix of radiobuttons; when one of the buttons is pushed, the specified value is sent to the DSP. One of the buttons must be specified as the initial value. Similiar to a `RadioButton` (below), but has more design flexibility.
  - **Data Array**—a list of files/data arrays that, when selected and the `Download` button pushed, will be downloaded to the DSP at the specified starting address.
  - **OnOff Button**—a momentary switch; when pressed, one piece of data is sent to the DSP, when released (or not pressed), the initialization data is sent.
  - **Pushbutton**—data is only sent to the DSP when the button is pressed; typically used to reset some value modified by the DSP.

- ***Output Controls*** [16:94–110]

  - **Alert**—a `True/False` LED-like indicator. When the logical condition is `True`, the LED is green, when `False` it turns red. Normal equality and inequality conditions can be checked for the parameter of interest. `Alert` can also be set to momentarily beep when the condition transitions from `False` to `True`.

  - **Bar**—a vertical gauge, reminiscent of a thermometer, to indicate the status of a program variable within user-specified bounds.

  - **Display**—a user-configured segmented numeric display. Numeric format is specified using a subset of C format codes for the `printf()` function.

  - **Gauge**—a dial gauge, reminiscent of an analog speedometer, with user-specified minimum and maximum values (no "out of range" condition indicated).

  - **Message Bar**—displays different text messages as the parameter of interest crosses pre-defined thresholds.

  - **Multi-Stage Alert**—looks like the `Alert` display, but changes color as the parameter of interest crosses pre-defined thresholds.

  - **Sound**—generates a tone who's frequency is based upon the value of the parameter of interest.

- ***Input/Output Controls*** [16:111–127]

  - **Check Button**–a toggle button that displays the state it is currently in. On of the states must be specified as the initial state.

  - **Incremental Input**—a display box for the parameter of interest that starts at a pre-defined value, but is adjusted up and down (in pre-defined increments) by clicking on the up/down arrows on the side of the display box. Each time one of the buttons is pressed, the DSP is sent the updated value. The value can also be updated directly from the keyboard when the control is selected.

  - **Knob**—a rotary control that can vary between the user-specified bounds. A scaling factor is also displayed with the knob. An initial value (and an increment

for moving the knob with the up/down arrow keys) must also be stipulated. When not active, the control indicates the current status of the parameter.

- **Numeric Input**—a direct numeric input block that only updates the value in the DSP when the `ENTER` key is pressed. When another control is selected, the block will provide real-time indication of the parameter's value. An initial value is required.
- **RadioButton**—a device with up to eight states for the parameter of interest, only one of which can be selected at a time. An initial value can be specified. While less flexible than the `ButtonArray` control, this control indicates the current state of the parameter.
- **Slider**—a linear control that can vary between user specified bounds. A scaling factor and "out of range" indicators are also displayed with the slider. An initial value (and an increment for moving the indicator with the up/down arrow keys) must also be stipulated. The control can be oriented vertically or horizontally. When not active, the control indicates current status of the parameter.
- **Table Editor**—a complicated spreadsheet-like lookup table (one- or two-dimensional); see [16:124–127] for more information.

- ***Special Controls*** [16:128–134]
  - **Frame**—a fixed rectangle to establish a border around a set of COCKPIT controls. Font, foreground and backgound colors are all user-specified.
  - **Image**—a box to hold a `.BMP` or `.GIF` images to display logos or other visually appealing backgrounds. Box is sized automatically to hold the specified image file. Other COCKPIT controls can overlay the image.
  - **Start Executable**—a button to start another program. The text displayed on the button is the same as the program name.
  - **Static Text**—a box used to hold descriptive text. Limited to 255 characters. Typically used for labelling the desktop or providing descriptive explanations on how to use elements of the desktop.

D.4 Experiment Philosophy

Just as the SE Process had a time dimension that increased the level of detail considered in each iteration, the implementation of an experiment should go from a total software implementation (all satellite and experiment hardware simulated by the software) to a fully functional *SIMSAT* system/experiment. And as the logic dimension of the SE process was applied during each life-cycle iteration, each of the development steps listed in Section D.3 (analogous to the "problem-solving process") should be applied during each of the experiment implementation iterations. Before moving to the next phase of implementation, any anomalous behavior must be explained or cleared up. To reduce the risks of a "bad" experiment, the experiment implementation phases should progress as shown in this list:

1. entire control system simulated; control laws executing on the Simulation PC
2. Processor-In-the-Loop (PIL);
3. Partial Hardware-In-the-Loop (HIL); baseline *SIMSAT* installed (such as the momentum wheels, sensors, etc.), but on a test bench
4. Evolution to full HIL; also an ordered progression:
   (a) baseline *SIMSAT* and payload implemented on a test bench
   (b) baseline *SIMSAT* and payload verified on the satellite, motion constrained
   (c) unrestricted experiment execution

D.5 Windows95/NT Implementation

The value of the graphical interface is the intuitive approach the user can take to accomplish complicated tasks. As shown in Section D.2.3, the tasks required to get a model running on the DSPACE system with only the tools provided by dSPACE, Inc. can be very complicated. To ease that workload, a folder called `dSPACE Files` (Figure D.31, page D-38 for the Simulation PC and Figure D.39, page D-43) was developed to house the following "shortcuts" to the tools the experimenter needs to make implementation (and

of *SIMSAT* control systems easier. Each of the figures below shows the entries required to establish a particular shortcut in the event any of them need to be recreated. The icons shown for each shortcut are either native to the application (as in the case of COCKPIT, TRACE, etc.) or selected arbitrarily from one of the icon libraries provided by Windows95 and NT (specifically, SHELL32.DLL).

D.5.1 Common Shortcuts. The SHOW VERSIONS, SIMULINK and `dSPACE Library` icons are common to both the Simulation PC and RealMotion PC machines.

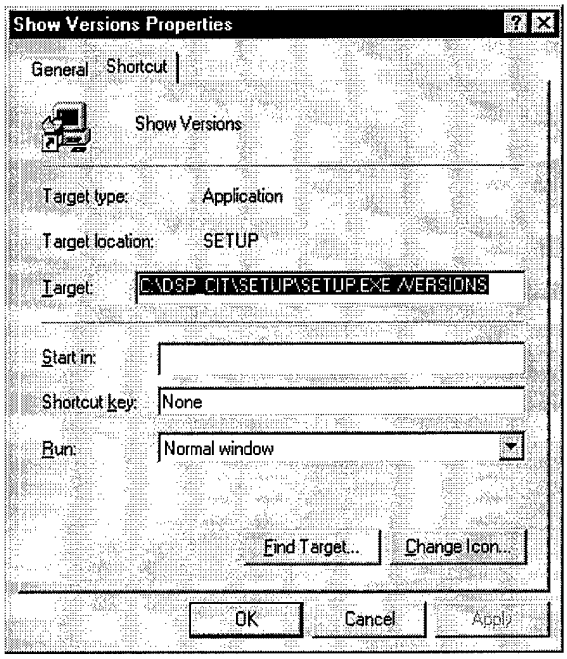


Figure D.28 SHOW VERSIONS Shortcut

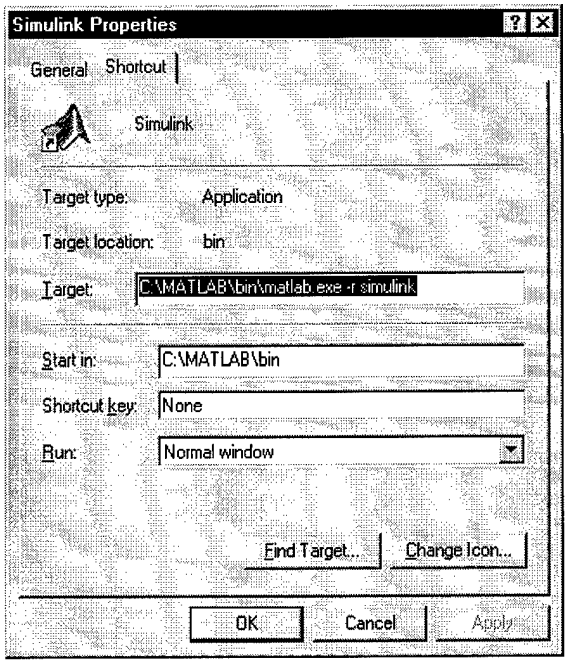


Figure D.29 SIMULINK Shortcut

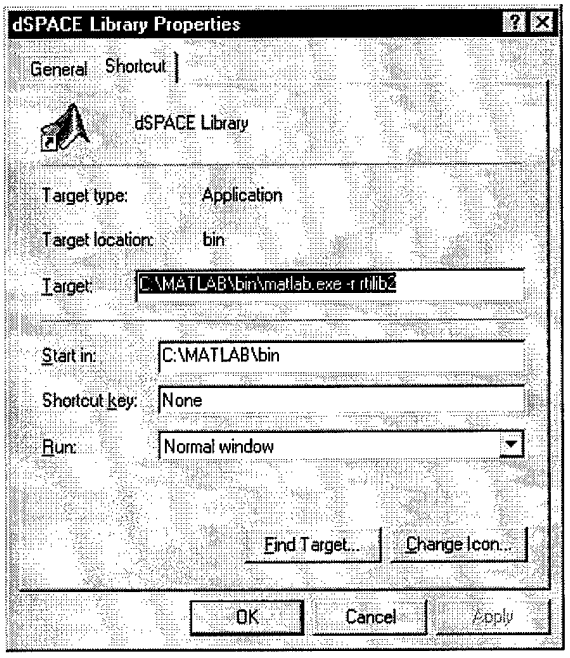


Figure D.30 `dSPACE Library` Shortcut

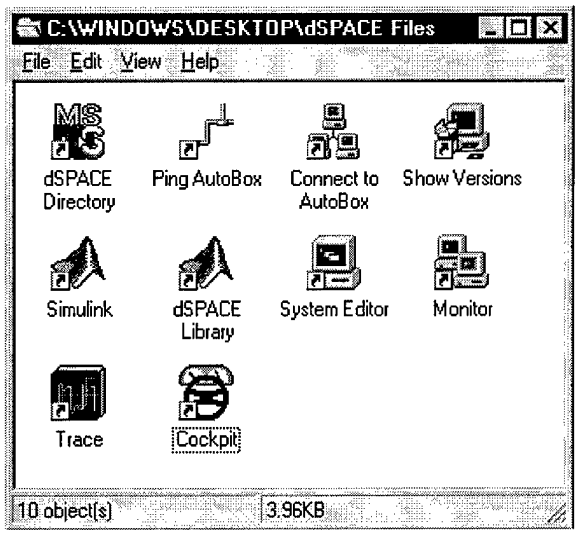


Figure D.31 `dSPACE Files` Folder Contents

D.5.2 Simulation PC Shortcuts.

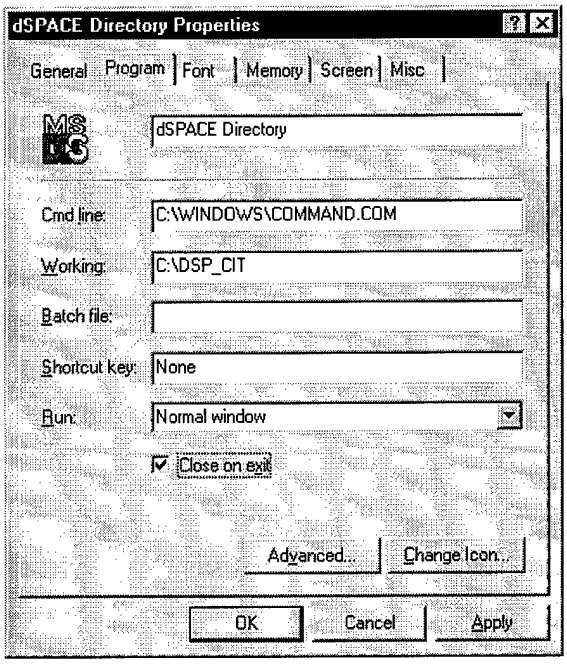


Figure D.32 `dSPACE Directory` Shortcut—Simulation PC

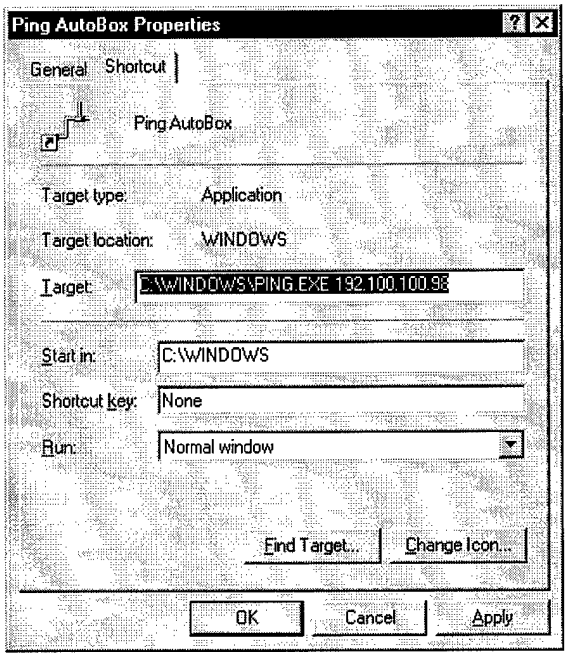


Figure D.33 `Ping` Shortcut

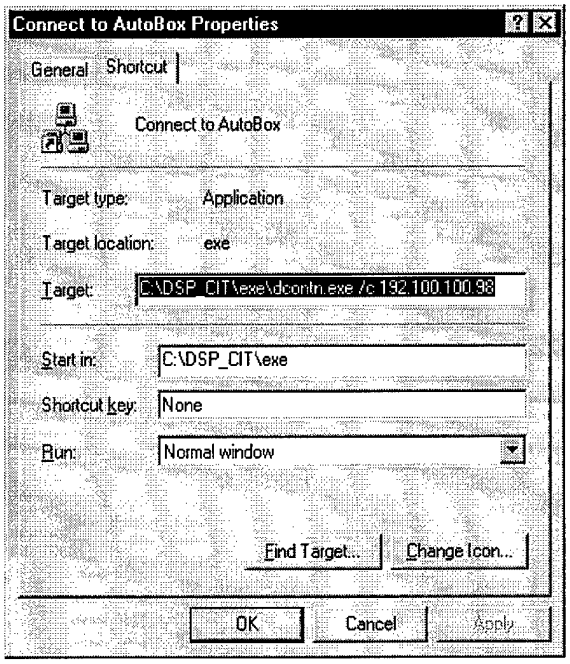


Figure D.34 `Connect To AutoBox` Shortcut

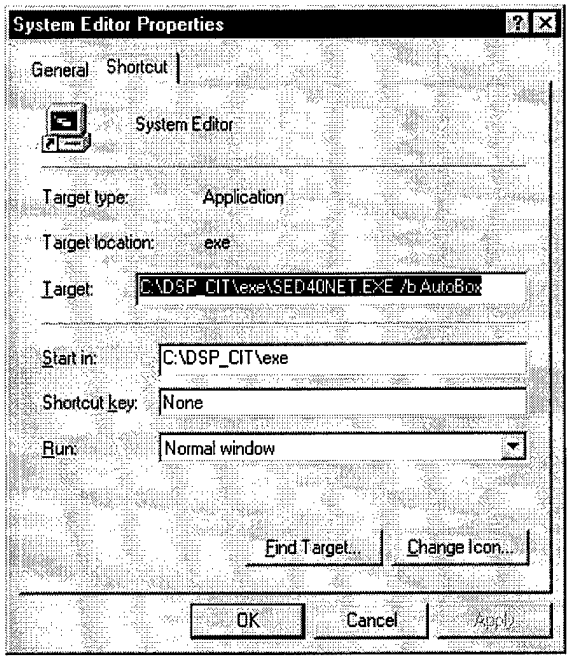


Figure D.35 SYSTEM EDITOR Shortcut—Simulation PC

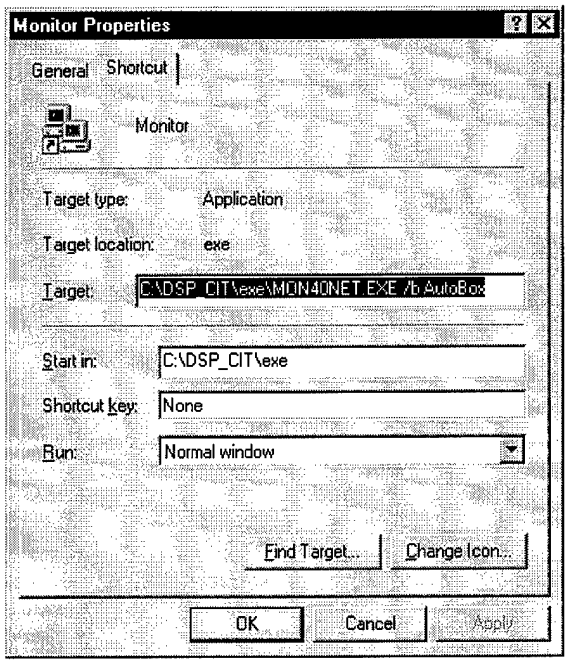


Figure D.36 MONITOR Shortcut—Simulation PC

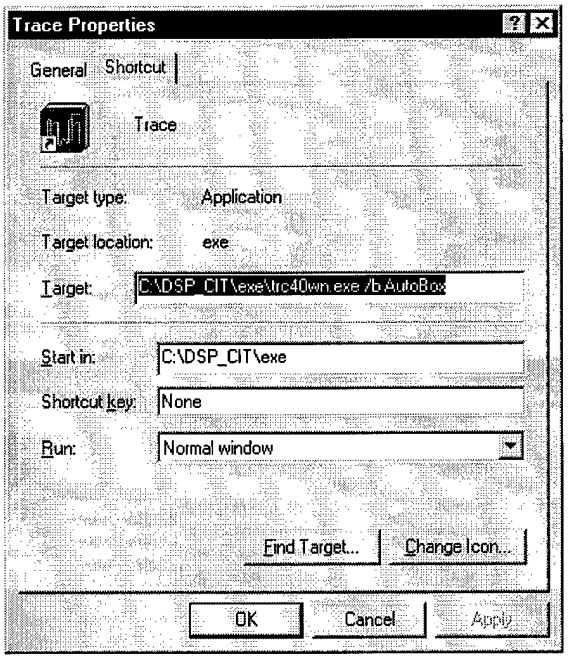


Figure D.37 TRACE Shortcut—Simulation PC

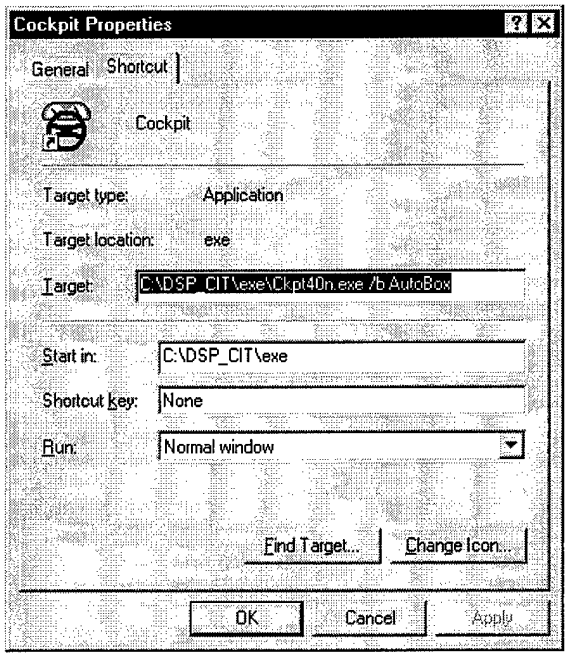


Figure D.38 COCKPIT Shortcut—Simulation PC

Figure D.39 `dSPACE Files` Directory—RealMotion PC

D.5.3 RealMotion PC Shortcuts.

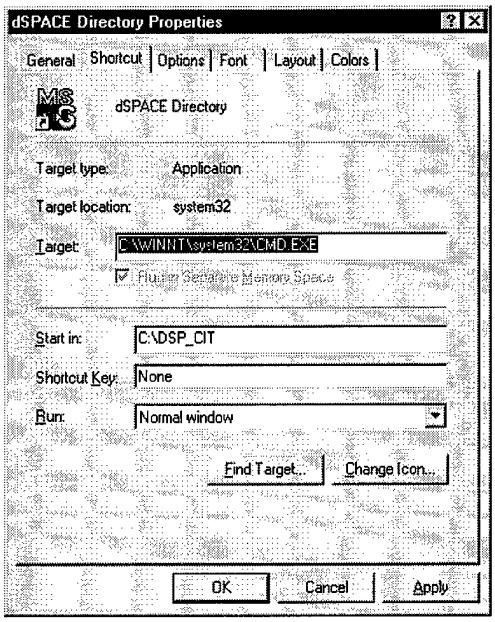


Figure D.40 `dSPACE Directory` Shortcut—RealMotion PC

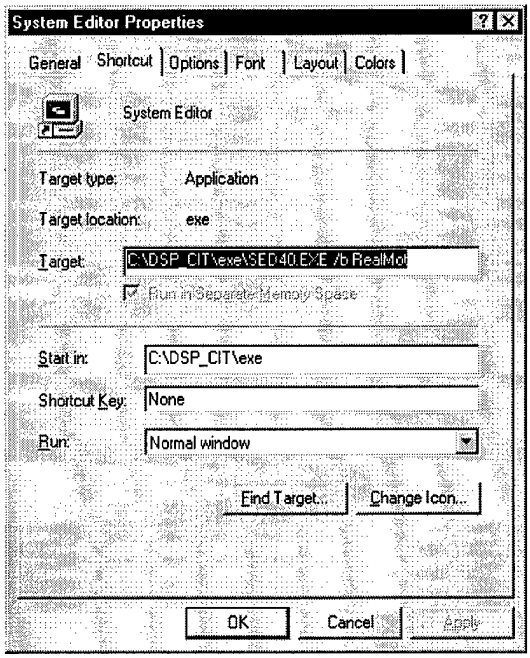


Figure D.41 SYSTEM EDITOR Shortcut—RealMotion PC

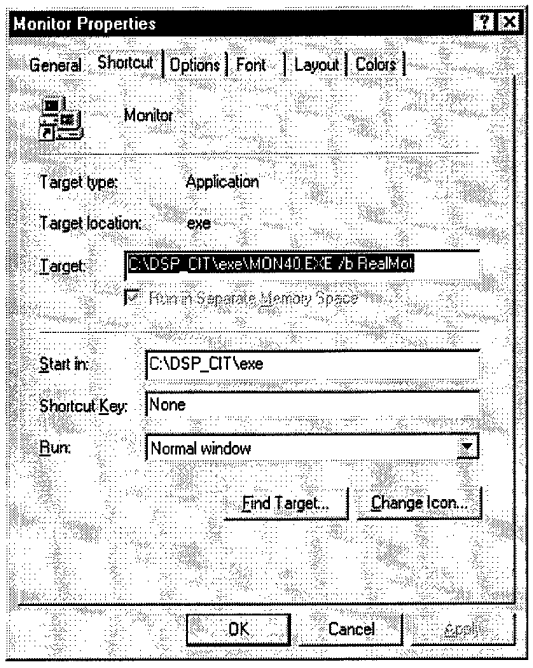


Figure D.42 MONITOR Shortcut—RealMotion PC

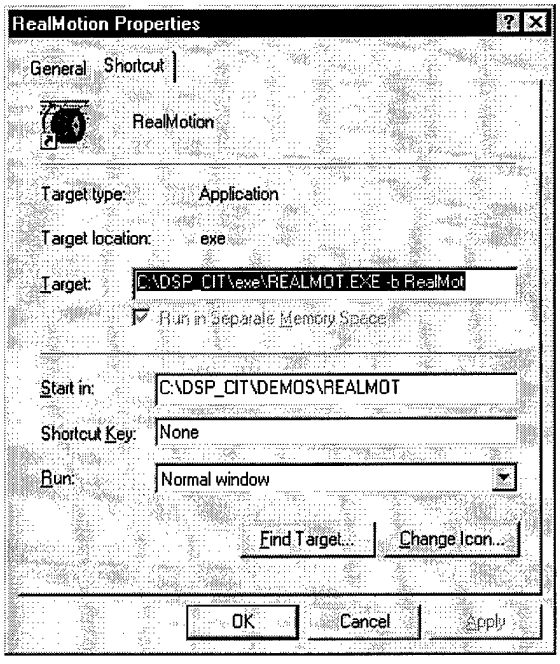


Figure D.43 REALMOTION Shortcut

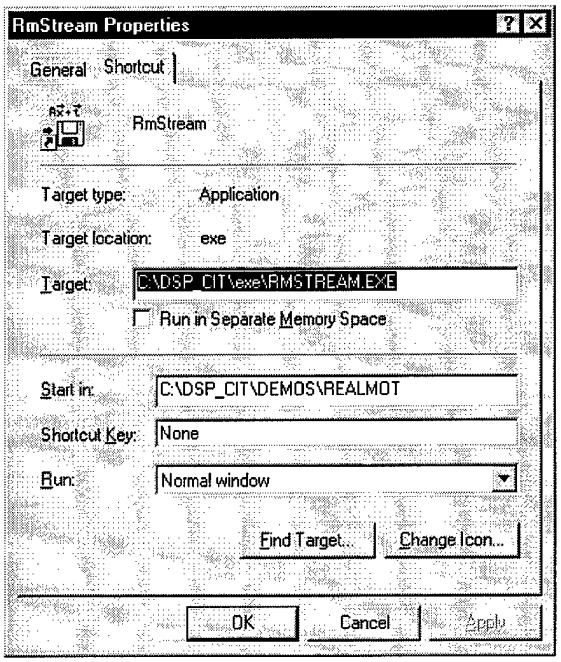


Figure D.44 REALMOTION STREAM Shortcut

Bibliography

1. Advantech Co., Ltd., no address provided. *PCA-6145B/PCA-6145L—Half-Size 486 All-in-One CPU Card with Panel/CRT/Ethernet Interface* (edition 1 Edition), 1997. (Single Board Computer manual).

2. Air Force Material Command, U. S. Air Force. *Engineering Management, MIL-STD-499*, 1977.

3. Air Force Material Command, U. S. Air Force. *System Safety, MIL-STD-882B*, 1984.

4. Baker, T. P. and A. Shaw. "The Cyclic Executive Model and Ada." In REAL [27], pp. 120–129.

5. Baldwin, R. O. *A Model For Determining Task Set Schedulability in the Presence of System Effects*. MS thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH 45433, December 1992.

6. Blanchard, Benjamin S. and Wolter J. Fabrycky. *Systems Engineering and Analysis*. Prentice Hall Inc., 1990.

7. Clements, Robert T. *Making Hard Decisions: An Introduction to Decision Analysis* (2nd ed. Edition). Duxbury Press—Brooks/Cole Publishing Co., 1996.

8. Colebank, James, et. al. "*SIMSAT*: A Satellite System Simulator and Experimental Test Bed For Air Force Research." 1999 Joint Space/Systems Engineering Team Thesis—complete integration of *SIMSAT*, to be published March 1999.

9. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *DS820 User's Guide* ((v. 2.0) Edition), 1992.

10. dSPACE, Inc., Technologiepark 25, D 3310 Paderborn, Germany. *DS 400 AutoBox User's Guide* (User's guide (v. 1.0) Edition), 1992.

11. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *Floating-Point Processor Board (DS1003)* (User's guide (v. 1.0) Edition), 1992.

12. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *Multi-Channel ADC (A/D Converter) Board (DS2003)* (User's guide ver. 2.0 Edition), 1994.

13. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *Connector Panels and LED Panels* (User's guide (v. 1.2) Edition), 1995.

14. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *Multi-Channel D/A Converter Board (DS2103)* (User's guide ver. 1.1 Edition), 1995.

15. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *3-D Animation Tool (RealMotion)* (User's guide (v. 1.0) Edition), 1996.

16. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *COCKPIT Instrument Panel* (Reference and user's guide (v. 3.3) Edition), 1996.

17. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *Real-Time TRACE Module* (User's guide/reference guide (v. 3.1) Edition), 1996.

18. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *Software Environment for Modular Systems (SEMOS); Manual 1: Single Processor Systems* (User's guide (v. 1.0) Edition), 1996.

19. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *Real-Time Interface to Simulink 2 (RTI1003)* (User's guide (v. 3.2) Edition), 1997.

20. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *General Installation Guide for Ethernet Systems* ((v. 2.1) Edition), 1998.

21. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *MATLAB-DSP Interface Library (MLIB)* (User's guide (v. 3.0.1) Edition), 1998.

22. dSPACE, Inc., Technologiepark 25, D 33100 Paderborn, Germany. *Real-Time TRACE Module for MATLAB (MTRACE)* (User's guide (v. 3.0.1) Edition), 1998.

23. Fathi, E. T. and M. Krieger. "Multiple Microprocessor Systems: What, Why, and When," *IEEE Computer*, pp. 23–32 March, 1983.

24. Fowler, Priscilla and Linda Levine. *Technology Transition Push: A Case Study of Rate Monotonic Analysis (Part 1).* Technical Report CMU/SEI-93-TR-29, ESD-TR-93-203, Carnegie-Mellon University/Software Engineering Institute, Pittsburgh, PA 15213, December 1993 (AD-A275616). (CMU/SEI effort).

25. Goodenough, J. B. and L. Sha. "The Priority Ceiling Protocol: A Method for Minimizing the Blocking of High Priority Ada Tasks," *Ada Letters, Special Edition: Proceedings of the 2nd International Workshop on Real-Time Ada Issues, vol. 8* no. 7, pp. 20–31 1988. (CMU/SEI effort).

26. Hall, Andrew D. "Three Dimensional Morphology of Systems Engineering," *IEEE Transactions on Systems Science and Cybernetics, vol. SSC-5* no. 2, pp. 156–160 1969.

27. Institute of Electrical and Electronic Engineers. *Proceedings of the IEEE Real Time Systems Symposium*, 1988.

28. Kirkwood, Craig W. *Strategic Decision Making.* Duxbury Press, 1997.

29. Klein, Mark H, et al. "Rate-Monotonic Analysis for Real-Time Industrial Computing," *IEEE Computer*, pp. 24–33 January, 1994.

30. Klein, M. H., et al. *A Pratitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems.* Kluwer Academic Publishers, 1993.

31. Klein, M. H. and T. Ralya. *An Analysis of Input/Output Paradigms for Real-Time Systems.* Technical Report CMU/SEI-90-TR-19, ESD-92-TR-220, Carnegie-Mellon University/Software Engineering Institute, Pittsburgh, PA 15213, July 1990. (CMU/SEI effort).

32. Kloeber, Jack (LTC). OPER745 Class Notes and Discussions, October 1998.

33. Kramer, Stuart (LtCol) and Greg (Capt) Agnes. Initial Meeting, March 1998.

34. Kramer, Stuart (LtCol). Interview, July 1998.

35. Larson, Wiley J. and Wertz James R., editors. *Space Mission Analysis and Design* (Second Edition). Microcosm, Inc. and Kluwer Academic Publishers, 1993.

36. Lehoczky, J. P., et al. "Enhanced Aperiodic Responsiveness in a Hard Real-Time Environment." *Proceedings of the 1987 IEEE Real Time Systems Symposium.* pp. 261–270. 1987. (CMU/SEI effort).

37. Lehoczky, J. P., et al. "The Rate Monotonic Scheduling Algorithm—Exact Characterization and Average Case Behavior." *Proceedings of the 1989 IEEE Real Time Systems Symposium.* pp. 166–171. 1989. (CMU/SEI effort).

38. Lendaris, George G.. "On Systemness and the Problem Solver: Tutorial Comments," *IEEE Transactions on Systems, Man, and Cybernetics*, No. 4, pp. 604–610 July/August, 1986.

39. Liu, C. L. and J. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment." *Journal of the Assoc. of Computing Machinery.* pp. 46–61. January 1973. vol. 20, no. 1.

40. Locke, C. D. and J. B. Goodenough. "A Practical Application of the Priority Ceiling Protocol in a Real-Time System," *SIGAda Ada Letters: Special Edition, vol. 8* no. 7, pp. 35–38 Fall, 1988. (CMU/SEI effort).

41. Locke, C. D., et al. "Building a Predictable Avionics Platform in Ada: A Case Study." *Proceedings of the 1991 IEEE Real Time Systems Symposium.* pp. 181–189. 1991. (CMU/SEI effort).

42. Locke, C. D. "Priority Inversion and Its Control: An Experimental Investigation," *SIGAda Ada Letters: Special Edition, vol. 8* no. 7, pp. 39–42 Fall, 1988.

43. Locke, C. D. "Scheduling in Real-Time," *UNIX Review, vol. 8* no. 9, pp. 48–54 September, 1990.

44. Moudgal, Vivek, et al., July–August 1998. Personal Communications with dSPACE Tech Support.

45. Pohl, Ed (Maj). SENG520 Class Notes, 1996.

46. Rajkumar, R. Personal Communications, 1993.

47. Rechtin, Eberhardt, editor. *Systems Architecting: Creating and Building Complex Systems.* Prentice Hall, 1991.

48. Sage, Andrew P. "A Case for a Standard Systems Engineering Methodology," *IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-7* no. 7, pp. 499–504 July, 1977.

49. Sage, Andrew P. *Systems Engineering.* Wiley Series in Systems Engineering, John Wiley and Sons, Inc., 1992.

50. "An Introduction to Rate Monotonic Analysis." A Software Engineering Institute (SEI) Seminar Handout, 1992.

51. Shaffer, P. L. "Minimization of Interprocessor Synchronization in Multiprocessors with Shared and Private Memory." *Proceedings of the 1989 IEEE International Conference on Parallel Processing*. pp. III.138–III.142. 1989.

52. Sha, L. and J. B. Goodenough. *Real-Time Scheduling Theory and Ada*. Technical Report CMU/SEI-89-TR-14, ESD-TR-89-22, Carnegie-Mellon University/Software Engineering Institute, Pittsburgh, PA 15213, April 1989. (CMU/SEI effort).

53. Sha, L., et al. *Rate Monotonic Analysis for Real-Time Systems*. Technical Report CMU/SEI-91-TR-5, ESD-91-TR-5, Carnegie-Mellon University/Software Engineering Institute, Pittsburgh, PA 15213, March 1991. (CMU/SEI effort).

54. Sha, L., et al. "Real-Time Scheduling Support in Futurebus+." In REAL [27], pp. 331–340. (CMU/SEI effort).

55. Sha, L. and S. Sathaye. "A Systematic Approach to Designing Distributed Real-Time Systems," *IEEE Computer* September, 1993. (CMU/SEI effort).

56. Smith, Gary. *Logical Decisions For Windows 95* (ver. 5.009 Edition). Logical Decisions, 1014 Wood Lily Drive, Golden, CO, 80401, April 1998. 1-800-35-LOGIC (800-355-6442).

57. Space Electronics, Inc., 81 Fuller Way, Berlin, CT 06037-1540. *Tri-axis Gas Bearing Model 2630 Instruction Manual*, 1997.

58. Sprunt, B., et al. *Scheduling Sporadic and Aperiodic Events in Hard Real-Time System*. Technical Report CMU/SEI-89-TR-11, ESD-TR-89-19, Carnegie-Mellon University/Software Engineering Institute, Pittsburgh, PA 15213, April 1989. (CMU/SEI effort).

59. Stankovich, J. A. "Misconceptions About Real-Time Computing," *IEEE Computer*, pp. 10–19 October, 1988.

60. Strosnider, J. K., et al. "Advanced Real-Time Scheduling Using the IEEE 802.5 Token Ring." In REAL [27], pp. 42–52. (CMU/SEI effort).

61. Thompson, H. A. and P. J. Fleming. "Fault-Tolerant Transputer-Based Controller Configurations for Gas-Turbine Engines." *1990 IEE Proceedings*. pp. 253–260. July 1990. vol. 137, pt. D, no. 4.

62. United States Air Force. *Global Engagement: A Vision of the 21st Century Air Force*, 1996.

63. Venkatramani, Chitra and Chiueh Tsi-cker. "Supporting Real-Time Traffic on Ethernet," *Proceedings of the 1994 IEEE Real-Time Systems Symposium*, pp. 282–286 1994.

64. "Definition of Systems Engineering." web page: http://www.incose.org/whatis.html, 1996.

Vita

In August, 1984, Michael P. Hanke graduated with distinction from the University of Michigan having earned a Bachelor of Science degree in Electrical Engineering. He completed Officer Training School and received his commission in December 1984. An honor graduate from Basic Communication-Electronics Officers School, he was assigned to the 485th Engineering Installations Group (a part of then-AFCC), Griffiss AFB NY in March of 1985. During that assignment, he engineered wideband communication systems for numerous CONUS and USAFE bases. After separating from active duty in July 1989, he came to Wright-Patterson AFB as a digital controls engineer in (then) Aeronautical Systems Division (a division of then-AFSC). His primary responsibilities involved the acquistion of airborne embedded computerized control systems. To bolster his knowledge of real-time control systems and formal engineering processes, he has taken both computer and systems engineering courses at the Air Force Institute of Technology as a part-time and full-time student. His is currently assigned to the C-17 System Program Office (AFMC) as the Senior Landing Gear Systems Engineer.

He is married to the former Judy L. Walsh of Dearborn Heights, MI, and they have two children, Justin and Alicia.

Permanent address: 7660 Stoncrest Dr.
Huber Hts, OH 45424
e-mail: mhanke@bigfoot.com