A PHIGS-Based Spreadsheet for Conceptual Design

by

Eric V. Schrock

thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Mechanical Engineering

APPROVED:

Sankar Jayaram

Dr. Sankar Jayaram, Chairman

Dr. A. Myklebust

Dr. W. H. Mason

May 24, 1991

Blacksburg, Virginia

Abstract

There are several computer aided design systems which exist to aid the conceptual phase of design. These CAD systems provide adequate geometry input, but often lack the capability to interactively obtain large amounts of numerical and text input often needed by analysis software. An interactive method for facilitating this non-geometric input to conceptual design analysis systems is needed. This thesis describes the creation of a spreadsheet input system to fulfill this need. The requirements, functional specification, and design of the system are addressed using a software engineering methodology. Accepted standards for programming and graphics, namely C and PHIGS, are used in the implementation. This is a system which can be used either to enhance existing CAD software or in the creation of new CAD systems. The use of this spreadsheet system in the enhancement of the well-known aircraft conceptual design system ACSYNT is also described in this thesis.

Acknowledgements

Without a strong foundation, any undertaking is fundamentally weak, and often meaningless. The individuals cited in these acknowledgements are recognized for their contributions in building my ethical and professional foundation, preparing me for a career in the "real world".

First, I would like to thank my parents, Dorothy and Harold "Bud" Schrock for their love, support, and encouragement during my academic career. They taught me that hard work and dedication is most often the greatest contributor in overcoming any obstacle.

To my advisor, Dr. Sankar Jayaram, thank you for all of the advice, encouragement, guidance, and patience. I am proud to be his first graduating Master's student, and hope that future supervisors I have are as understanding and willing to listen as him.

Thanks are also extended to "the boss" of the Computer Aided Design Laboratory, Dr. Arvid Myklebust, for the opportunity to work on the ACSYNT project. My work with ACSYNT gave me an opportunity to interact with representatives from the aircraft industry and to develop myself professionally unlike many other graduate programs.

Thanks are also extended to Dr. W.H. Mason of the Aerospace Engineering department for serving on my committee. It was nice to have an "old hand" from the aerospace design community around to give advice on new ideas and current happenings in the business. Along these lines, I would also like to thank Paul Gelhausen of NASA-Ames for all of the suggestions, comments, and interesting transcontinental phone conversations.

My work was supported by the ACSYNT Institute, a collaborative venture between industry, government, NASA-Ames, and Virginia Tech. The ACSYNT Institute was founded to promote the use of ACSYNT and to provide a basis for research and support. Thanks are extended to the members of the Institute for my financial support.

I would also like to thank Kris Kolady, the resident C, Script, and UNIX god, and Uma Jayaram, Fred Marcaly, Michele Grieshaber, Bob Jones, J.R. Gloudemans, Daniel Pawtowski, and Steve Fleming for all of the help and fun over the last eighteen months. It was great working with you, and I wish all of you success and happiness.

Table of Contents

List of Illustrations

1.0 Introduction

There are many computer aided design (CAD) software systems on the market. These do not always fit the engineering design requirements of a particular organization. As a result, many companies develop their own CAD software in-house, devoting large amounts of time and resources toward fulfilling their needs. A recent survey of Fortune 500 companies has shown this to be true [Penn91]. Approximately 73 percent of the companies surveyed develop customized software for their engineering design needs.

Several approaches have been taken to reduce the expense involved with producing CAD software. Graphics standards such as GKS (Graphics Kernel System) and PHIGS (Programmer's Hierarchical Interactive Graphics System) provide portability across different workstations and allow for commonality in lower-level graphics instructions such as line drawing and viewing. These basic graphics programming building blocks do not always provide the high-level functions needed by most CAD programs (e.g. nested menus, geometric modeling of objects, wireframe construction, rendering, etc.). The programmer must design and code these functions each time a new piece of CAD software is written. This is not a trivial task, as thousands of lines of documented code

are necessary to implement even a simple nested menu or rendering function. This process is time consuming and repetitive, and should be eliminated to increase reliability and speed of development of commercial computer aided design software. A "programmer's toolkit" that incorporates standard routines for typical CAD functions would eliminate many of the problems caused by repeated coding of the same functions by different organizations.

Functions that could be included in a high-level graphics programmer's toolkit include menu management, viewing operations, wireframe geometry construction, and shaded view creation. Incorporating this functionality in a higher level CAD programming language would boost commonality between software systems, reduce learning times, and cut the cost of custom software creation.

Some progress has been made towards the development of high-level functions for the creation of custom CAD software. For instance, advanced capabilities in PHIGS+ include the creation of parametric curves and surfaces, graphical trimming of curves and surfaces, and surface and polygon shading functions.

However, not enough is being done to support the actual creation of interactive design software using device-independent and standardized methods. Jayaram [Jaya90] outlined this problem and conceptualized an applications programming environment called CADMADE (Computer Aided Design and Manufacturing Applications Development Environment). Any implementation of CADMADE should include standardized functions to aid the CAD software developer in creating geometric modeling, mechanical design, manufacturing, expert system, and user interface software. Complete application of CADMADE principles and functions will reduce development times and increase standardization among computer aided design software systems.

CADMADE includes a User Interface Environment (UIE) which contains procedures to assist the software developer in quickly building a highly interactive user interface for CAD programs [Jaya89]. The UIE is comprised of routines that manage the following common CAD functions:

- Menu and Icon Control
- User/Program Communication
- Viewing Transformation Manager

Although this User Interface Environment will provide for the easy creation of hierarchical menus, icons and dialog areas, the problems associated with the creation of an interactive system for large amounts of numerical input have not been addressed. This thesis describes the creation and implementation of a spreadsheet method for handling the input of analysis data for conceptual design CAD systems.

1.1 Problem Definition

Custom software development is done to aid all aspects of engineering design, which is often broken down into three principal phases; conceptual, preliminary, and detail. The first phase of design concentrates on the definition of a concept and involves determining the size and overall layout of the product. Initial performance estimates and "first- cut" analyses are performed which reveal whether the configuration in question can fulfill the

requirements of the design. The preliminary phase of design extends the analysis, and closely investigates material properties and makes refined estimates of performance. By the completion of the preliminary design phase, the geometry is no longer allowed to change. Detail design takes the "frozen" geometry output from the preliminary phase and addresses production questions and other engineering details necessary to bring the product to the marketplace.

The speed and reliability of computer aided design methods can greatly enhance the conceptual design process. Computer programs to assist the designer in the conceptual phase have been developed for several different engineering disciplines. Several initial designs can be evaluated in a short period of time, allowing the designer to "brainstorm" and select the best of a wide range of possible solutions.

Most CAD software systems developed for the conceptual phase of the design process handle geometrical input well, but often do not consider the large amounts of numerical input necessary for the other portions of conceptual design analysis. The user of these programs often leaves the CAD environment to use a text editor or other general method to construct a file with the proper numerical data for input to the analysis system. In cases where numerical input is provided in the CAD program, it is often done in an awkward manner, without regard to the type or amount of data required.

Because the input to the analysis system is an essential and time-consuming task, the user should have a very effective method at hand to accomplish the data entry. In order to satisfy this need, the CAD software programmer should be able to create an easy-to-use interactive system to facilitate the data entry process. Methods for large amounts of data entry have been implemented in various systems (e.g. financial and accounting software) using spreadsheets. However spreadsheet methods have not been

used to facilitate data entry for conceptual design CAD systems. CAD software programmers should have available a set of high-level functions that assist in the creation of spreadsheet systems for conceptual design analysis software.

The CAD software programmer will welcome this new tool, as it incorporates the features of modularity and flexibility that could make it a high-level function in a set of standardized CAD routines to aid program development. In addition, if the graphics functions are created using the PHIGS standard, portability across all workstations and mainframe computers can be ensured.

1.2 Organization of Thesis

In addition to the creation of a spreadsheet system, this thesis discusses the research that produced its requirements. The application of the spreadsheet to an existing conceptual design CAD system for aircraft, ACSYNT, is also discussed. The general layout of the thesis is as follows:

- Research Objectives
- ACSYNT as a Case Study
- Requirements for Spreadsheet Software Design
- Spreadsheet System Functional Specification

- Design of Spreadsheet Software
- Integration of the Spreadsheet with ACSYNT
- Summary and Conclusions
- Recommendations

It is anticipated that readers of this thesis will be investigating the methods behind the spreadsheet operation or trying to integrate the spreadsheet into a new or existing CAD program. Readers interested in integration should refer to the ACSYNT-specific portions of the thesis, while those interested in the software engineering aspects of the system should concentrate on the Requirements, Functional Specification, and Design sections.

2.0 Research Objectives

As described in the previous section, there is a need for large amounts of numerical input for conceptual design analysis, and this is not handled in an efficient manner by many current computer aided design systems. The spreadsheet software discussed in this thesis must address the shortcomings of today's conceptual design CAD programs and demonstrate flexibility to adapt to new codes that are yet to be developed.

The system must be designed so that it also can be integrated with existing conceptual design programs. Because many CAD programs use extensive amounts of memory due to geometric modeling and analysis requirements, the size of the spreadsheet's data structure should be kept to a minimum. Help facilities built into the spreadsheet should also be memory sparing, and not excessively slow.

The software development cycle consists of a series of steps, including requirements generation, functional specification, program design, production, and verification [Taus79]. Programming and software development standards must be stressed

throughout the software development cycle. The resulting product will be of higher quality and more easily maintained than one designed without these ideals in mind.

Throughout the design, human factors and usability issues must be considered from both the user's and the programmer's viewpoint. In addition to making the system easy to use, effort should be directed towards making the actual code understandable and clear. Thus, consistent application of sound human factors and software engineering principles will make the resulting program both easy to use and modify.

In summary, the objectives of this research were:

1. Survey existing spreadsheet and computer aided conceptual design programs.

2. Formulate the user and software requirements and functional specifications of a spreadsheet system which can be used in existing and new conceptual design CAD systems.

3. Design and implement the system.

4. Integrate the spreadsheet system with ACSYNT to test the functional completeness and robustness of the software.

5. Examine the results and suggest recommendations and enhancements.

3.0 ACSYNT as a Case Study

The ACSYNT (AirCraft SYNThesis) program was chosen for initial implementation of the spreadsheet. ACSYNT was developed in the 1970's at NASA-Ames Research Center to study the effects of advanced technology on the conceptual design of aircraft. ACSYNT was the first aircraft conceptual design code to be designed in a modular fashion, with each discipline of aircraft design analysis assigned to a different "module", or structured group of routines intended to handle that particular phase of analysis. Currently, ACSYNT uses the following modules to analyze supersonic or subsonic transport, bomber, or fighter aircraft [Gelh90].

- Geometry
- Trajectory (Mission Profile and Performance)
- Aerodynamics
- Propulsion

- Stability
- Weights
- Supersonic Aerodynamics
- Economics
- Takeoff Performance

The different modules of ACSYNT can be executed in a user specified order, and not all modules need to be run at once. A control program supervises module execution, and can be instructed to call the modules repeatedly until the aircraft weight is converged.

ACSYNT's modular structure lends itself easily to optimization techniques. The optimization program COPES is coupled with ACSYNT. COPES (COntrol Program for Engineering Synthesis) gives the user the power to perform sensitivity analysis, optimization, two-variable function space analysis, and approximate analysis using ACSYNT variables and computation methods with up to 128 constraints and/or objective functions.

In 1987, the Computer Aided Design Laboratory at Virginia Tech began constructing an interactive CAD version of ACSYNT [Wamp88a][Wamp88b]. This system uses the ISO graphics standard, PHIGS, and standard programming languages (FORTRAN and C) to add portability and a degree of interactivity previously unavailable in aircraft conceptual design codes. This interactive system allows the designer to create and modify aircraft geometry parametrically. Additionally, the CAD portion of ACSYNT

has advanced device-independent rendering and viewing capabilities, interactive mission profile input and graphing and file viewing functions for better visualization of the results of aircraft analysis. Figure 1 on page 12 shows the capabilities and interrelationship of the different parts of ACSYNT to one another.

ACSYNT provides excellent methods for geometry input [Wamp88a]. Aircraft components are defined and modified parametrically, instead of through the construction of lines and planes by specifying points in 3-D space. The designer using ACSYNT uses design parameters such as "aspect ratio" or "taper ratio" to change the displayed geometry. Geometry is only one of the ACSYNT discipline modules, and most of the others require numerical input that cannot be visualized using geometric construction. Early releases of ACSYNT included a numerical input format structured for the beginning user. Numerical input was only offered for the geometry, trajectory, propulsion, and aerodynamics modules, COPES input, and the ACSYNT control program. Input to the other modules required leaving ACSYNT and modifying the analysis input file with a text editor.

Changes to those analysis variables supported in early releases of the interactive version of ACSYNT were accomplished by descending a menu tree up to five layers deep. When a variable required changing several times in order to investigate its impact on the design, the menu path must be traversed with each iteration. This process detracted from the ease of use of the software. For inexperienced and first-time users, this method worked well, giving ample help information and a good sense of the relevance and location of the variables in the analysis code. However, after the user became familiar with the variables in question, all of the menu levels still had to be traversed, or an

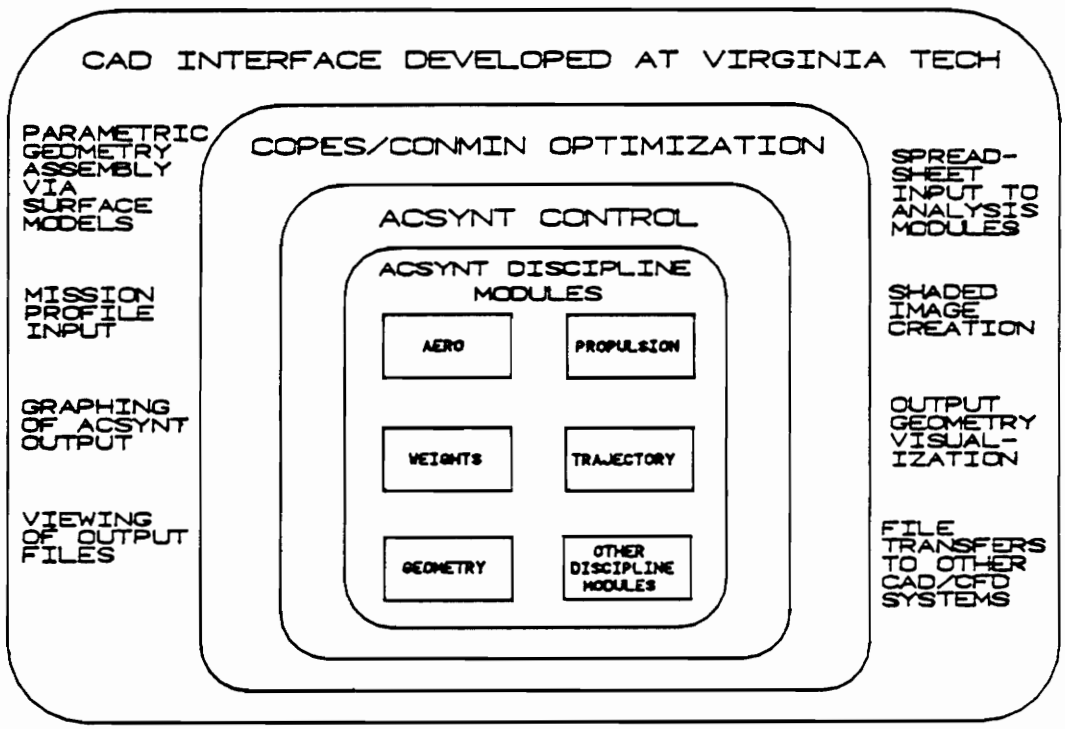


Figure 1. ACSYNT Program Structure

unwieldy amount of numerical input was needed to bypass the menu levels and access the design variables in ACSYNT's data structure.

In order to accommodate both the novice and expert ACSYNT user, a streamlined and flexible input format for numerical data is required. The spreadsheet method described in this thesis seems ideal for application to ACSYNT to remedy its shortcomings in analysis input. A spreadsheet can be employed to facilitate input to each analysis module, COPES and ACSYNT Control. Each spreadsheet will hold all the analysis variables supported by its analysis module or control section. Utilizing several spreadsheets in this manner will separate the analysis variables and eliminate unnecessary searching.

4.0 Literature Survey

Since the problem of creating a spreadsheet for conceptual design codes touches on many different disciplines of engineering and software design, the literature survey for this thesis is broken down into the following parts:

- Existing Spreadsheet Programs
- General Conceptual Design Programs
- Aircraft Conceptual Design Programs
- Human Factors Issues

4.1 Existing Spreadsheets

Spreadsheets present numerical information in an orderly manner. Examples of spreadsheets include the accountant's ledger, itemized long distance telephone bills, and pay statements. The automation of these paper spreadsheets marked a significant stride forward in the evolution of business and accounting software. The first computerized spreadsheet appeared in 1978, with the introduction of VisiCalc [Town85]. VisiCalc was designed to run on the new generation of small computers running the CP/M operating system and was later modified to support MS-DOS machines. Lotus 1-2-3, the name most synonymous with spreadsheets, was introduced in 1983, and combined data management, business graphics, and an improved version of VisiCalc's spreadsheet in one software package [Ewin87].

Since 1983, other spreadsheet programs such as *Microsoft Excel* and *Wingz* have appeared on the market for both IBM-PC and Apple Macintosh machines. While differing in small details of their function and "look and feel," the philosophy behind their operation is essentially the same as those of the original VisiCalc and Lotus 1-2-3.

4.2 General Computer Aided Conceptual Design Programs

Some industries have been using CAD in conceptual design for years. One example is the automobile industry, particularly for the design of internal combustion engines. Geometry and design parameters such as the number of cylinders and cylinder bore and stroke must be considered in the conceptual phase of engine design. One instance of the application of CAD programs for this type of design is covered by McClelland [McCl84]. The paper describes methods used to design and analyze internal combustion engines. Separate drafting and analysis programs are used, with the analysis program receiving its data through file transfer. No advanced interactive methods are employed.

In another paper by McClelland [McCl86], the requirements for a new engine design software system are described. Transfer of data between analysis and design (drafting) programs is handled by IGES (Initial Graphics Exchange Specification), which can pass only graphics data between programs. While it is mentioned that the design portion of the CAD program can calculate mass properties and perform cam design calculations, this is done often without any form of interactive feedback. When entering an external finite element system for vibration or stress analysis, the designer must specify the stiffnesses and material properties necessary to complete the analysis using conventional techniques.

Low and Baruah [Low81] discuss the development of a CAD system for manifold design. Detailed analysis via a finite difference method for non-steady reacting flows using the method of characteristics is employed. The interactive portion of the program handles

the input of data and presentation of the results. Over 100 pieces of numerical data are necessary for the analysis. This CAD system handles only the geometry input, which accounts for forty percent of the data. Other parameters must be entered in a file which is read by the analysis software.

The Toyota Motor Company's CADETT design system has been extended for the design of engine components, and this work is described by Taniguchi, Shimizu, and Sakakura [Tani87]. The paper highlights the geometrical features of the software system, and little is mentioned of the additional numerical input required for analysis or production of the designs.

4.3 Aircraft Conceptual Design Programs

In the late 1970's and early 80's, the aerospace industry began moving away from conventional pencil and paper drafting methods and toward sophisticated computer aided design systems. Hammond's paper "Graphics in Conceptual Aircraft Design - A Designer's Viewpoint" [Hamm86], discusses the advantages and disadvantages of moving to a CAD system during the conceptual design phase. An aircraft conceptual design tool called CDM (Configuration Definition Module), which is part of the IDAS (Integrated Design and Analysis System) is reviewed. CDM has powerful CAD capabilities that allow complete configurations to be assembled in a few hours, ready for the analysis modules of IDAS. Each component of a CDM configuration can have up to 45 lines

of numeric data associated with it, including mass properties. However, in the absence of a quick and efficient input format, the user must enter this data line by line.

During his tenure at Rockwell International, D. Raymer supervised the programming of CDM (then known as CDS - Configuration Design System). A paper presented to the AIAA Aircraft Design Short Course in 1982 describes CDM and computer aided design in detail [Raym82a]. A later publication by Raymer [Raym82b] explains the capabilities of CDM and its interface to the analysis routines of IDAS. Geometric interface files are written in CDM and transferred to the analysis routines. No mention is made of non-geometric analysis data, or the input method used.

All major aerospace companies use a conceptual design program that has been constantly improved throughout the years. Some of these codes have CAD capabilities and others do not. The report "A CAD System for Automated Conceptual Aircraft Design" [Mykl89] provides an overview of existing conceptual design programs in the aerospace industry. Codes in use by Grumman, Lockheed, Northrop, NASA-Ames, NASA-Langley, Boeing, and General Dynamics are described. Few of them have integral CAD capabilities and none of the programs mentioned in this review show any signs of high performance interactive numerical input for analysis variables.

Work began in the late 1980's under the direction of J. Roskam to write an aircraft conceptual design program based on his successful "Airplane Design" series of books. The resulting program, called AAA for "Advanced Aircraft Analysis", incorporates a graphics interface and is intended for use on Apollo workstations [Kohl90]. Despite the program's graphics interface, there is no spreadsheet interface to the database of analysis variables.

Hays [Hays89] discusses the use of commercial spreadsheet systems in aircraft conceptual design. He discusses the advantages of spreadsheets and proposes several Lotus 1-2-3 spreadsheet templates that can be used for different disciplines of aircraft conceptual design. This work was intended primarily for use as a teaching tool.

Wampler et al. [Wamp88a] describe the CAD interface to ACSYNT in a 1988 AIAA paper "Improving Aircraft Conceptual Design - A PHIGS Interactive Graphics Interface for ACSYNT". This paper describes the geometric modeling philosophy underlying the computer aided design capabilities of ACSYNT. Data handling of the analysis variables is not discussed.

The design of the interactive mission profile input system for ACSYNT is presented in A. Taylor's thesis "Specification of Mission Cycles for Aircraft Conceptual Design Using the PHIGS Standard" [Tayl88]. This document describes the input of non-graphical analysis variables for early versions of ACSYNT. The program's analysis data structure and its editing levels are explained. Three levels of data input or menu selections were required for changing one analysis variable. This approach worked well for the beginning user, and was also intended to support the needs of advanced users. All elements of the data structure, including parameter description files for initialization of the database are covered.

4.4 *Human Factors Issues*

A review of human factors topics in CAD and analysis software design was also conducted. While it was difficult to find spreadsheet-specific human factors publications, several articles proved helpful in the design of the spreadsheet software.

Engineering design software presents a unique set of requirements for human/computer dialogues. Suggestions for future dialogues are presented by Sinclair [Sinc87]. He states that quick calculation facilities must be readily available during the CAD process, so that the designer is free to concentrate on the design problem at hand. In addition to other requirements, the design tool must also be able to keep the user posted on the status of the system's activity, answering questions about the current sequence of commands and future operations.

Powrie [Powr87] emphasizes the use of design guidelines for a human/computer interface during the development of a CAD system. Evaluation procedures for CAD interfaces are also presented.

A novel approach to the visualization of the design optimization process is shown by Colgan, et al [Colg89]. "The Cockpit" monitors the action of objective functions and design variables during optimization of electrical circuits. It was found that the application of this technique encouraged electrical engineers to use the optimization capabilities of a circuit design system more often.

Human-computer interaction styles are discussed by Shneiderman [Shne88]. He presents different methods of interaction and the advantages and disadvantages of each. Special

attention is paid to the best choice of interaction style for a given application and a specific hardware device.

5.0 Requirements for the Design of the Spreadsheet

Successful software projects start with the specification of requirements for the function and coding of the program. The spreadsheet project described in this thesis followed a rigorous course of software design steps as set forth in the book "Standardized Development of Computer Software" [Taus79]. This portion of the thesis introduces the reader to the software development process and discusses the user requirements, software requirements, and functional specification that were used to arrive at the goal of an operating spreadsheet for conceptual design programs.

5.1 The Software Development Process

The term "software development" is used to encompass the span of time between the recognition of the need to create a new program and the installation of the operational code. The software development process consists of several stages [Taus79]:

- Requirements Generation
- Program Definition
- Program Design
- Production
- Verification

Requirements generation involves the description of the general tasks the program should perform. These are ideas stated by the organization requesting the finished program. Formal requirements also include an estimate of the cost, manpower, and schedule needed to complete the project in a realistic period of time.

The program definition stage provides a precise outline of what the program should perform. At this stage, human/computer interaction styles are set, and interfaces to other programs are considered. After this phase is complete, the program's function is fully defined, including all inputs and outputs, and error handling methods.

Design of the software is carried out in an iterative manner, studying the best and most efficient way to arrive at the goals spelled out in the definition and requirements stages. Several methods can be used to aid the software design process. Diagrams to depict program control flow, as well as the movement of data items across program modules may be used to help the designer visualize the design problem. At the conclusion of the design process, a set of abstractions in the form of diagrams or flowcharts may be produced to aid in the program production portion of the software development cycle [Taus79].

Coding takes place in the program production phase. Here, the diagrams produced in the design phase are analyzed and used to produce the actual program. Portions of the program are tested as they are coded to ensure that they perform as designed.

The verification process examines the performance of the program as a whole unit, and compares its results to those enumerated in the requirements and definition phases of the software development cycle. Insufficiencies or problems in the code found at this point are corrected before the program is released for operation.

The software development cycle often proceeds along with other engineering efforts to get a complete system, such as an air traffic control system, operational. The software development process is as important (if not more than) many of the "physical" engineering tasks going on around it. It presents many interesting problems that must be addressed. No ethical engineer would submit a design for production if all the load-bearing members had not been properly analyzed. However, software often enters the production phase before all of the program requirements are identified. The software engineer must make many modifications to the code if the requirements change, weakening and decreasing the reliability of the final product. In order to try to minimize these shortcomings in production code, several standardized methods for software development have been suggested [Taus79].

One such standardized method was followed during the design of the spreadsheet software. Hierarchical decomposition of the program modules using ANSI standard flowcharting symbols proved to be an extremely efficient way to organize the design of the program after a formal set of requirements was created. These requirements are described in the next section.

5.2 *Requirements for the Spreadsheet Design*

Prior to the creation of the functional specification and actual design of the program, the requirements of the program must be identified. It is difficult to separate the information and ideas generated during the early phases of program development into requirements and functional and design specifications. Design constraints can masquerade as requirements, and vice versa. In a formal, business environment, when one organization contracts another to design and write a program, the requesting organization develops a requirements document that is used by the developer to guide the initial development of the code. This document specifies the user environment, data processing needs, available resources and timetable, and the relationship with other ongoing projects, if any [Taus79].

The development of the program in this thesis deviates from the path of formal requirements generation described above. The author generated the requirements, functional specification, and design of the code. No independent developing and requesting organizations existed, so no documents with hard and fast guidelines were passed. Because the "coding organization" is the same as the "requesting organization", the requirements have a high degree of detail and contain many direct applications to the program design.

For the purposes of this thesis, the requirements are broken down into two sections: user requirements and software requirements.

5.2.1 User Requirements

User requirements define the program's human/computer interface and specify other necessary information about the actual use of the program. The user requirements for the spreadsheet established the screen design, location and placement of the spreadsheet cells, the use of different devices for moving the cell pointer, and methods to handle text input.

5.2.1.1 Screen Design

Since the ACSYNT program was selected as the initial testbed application for the spreadsheet, the basic layout of the screen was specified to be similar to that of ACSYNT. Figure 2 on page 27 shows the layout and terminology associated with the ACSYNT screen.

The portions of the spreadsheet program that govern the operation of the basic screen should be easily modified so the spreadsheet can be added to existing codes or appended to new ones. A highly modular program structure is suggested to attain this goal.

As with ACSYNT, the spreadsheet should use a scrolling message or dialog area to keep the user informed of the current status of the program [Wamp88b]. Messages about system status and use and suggestions for user action appear in this area of the screen.

Two menu areas exist on the ACSYNT screen. The "standard menu" presents options that are available in all of the functions of the program. The regular menu area shows

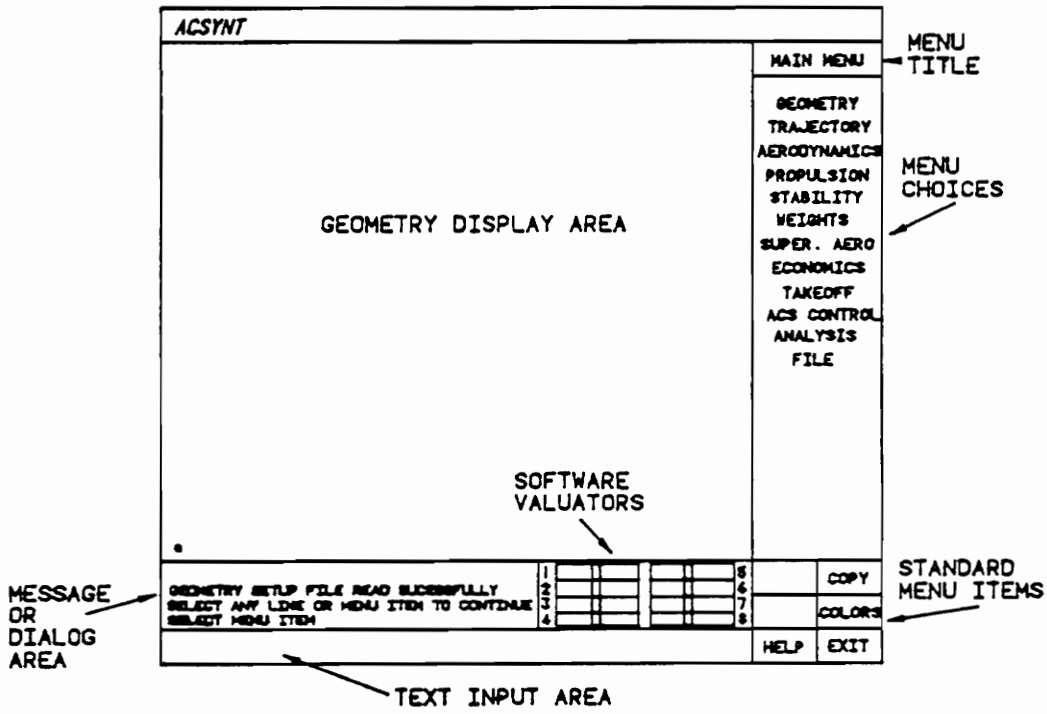


Figure 2. ACSYNT Screen Layout and Terminology

options for the current menu level and is organized in a hierarchical fashion. The spreadsheet should mimic the same structure for its menu items to ensure consistency with ACSYNT.

ACSYNT's text input area occupies the lower five percent of the screen. The spreadsheet should use the same area, and care should be taken that no spreadsheet input will exceed the length of this area (70 characters).

The geometry area of ACSYNT will be utilized for the display of the spreadsheet cells. The layout of the spreadsheet screen is shown in Figure 3 on page 29. This figure shows how the spreadsheet system should appear when integrated with ACSYNT.

Each cell should have an address in the form of a letter and a number to reference its position in the spreadsheet (A1, B7, F10, etc.). For reference, indices should be placed around the borders of the screen to assist the user in locating cells by their address. In addition to the cells and indices, cursor movement aids should be placed around the border of the cell area. The function of these aids is explained in the Functionality section.

The number of cells displayed on the screen should be governed by established human factors standards for text size [ANSI88]. Three lines of text should be displayed in each cell, one for the variable name, one for its value, and a third for a comment line, to give a quick description or reference about the use of the variable.

A subtitle area should appear above the spreadsheet cell area. Information about the current set of spreadsheet data should be displayed in this area, to keep the user informed about the current data set being manipulated.

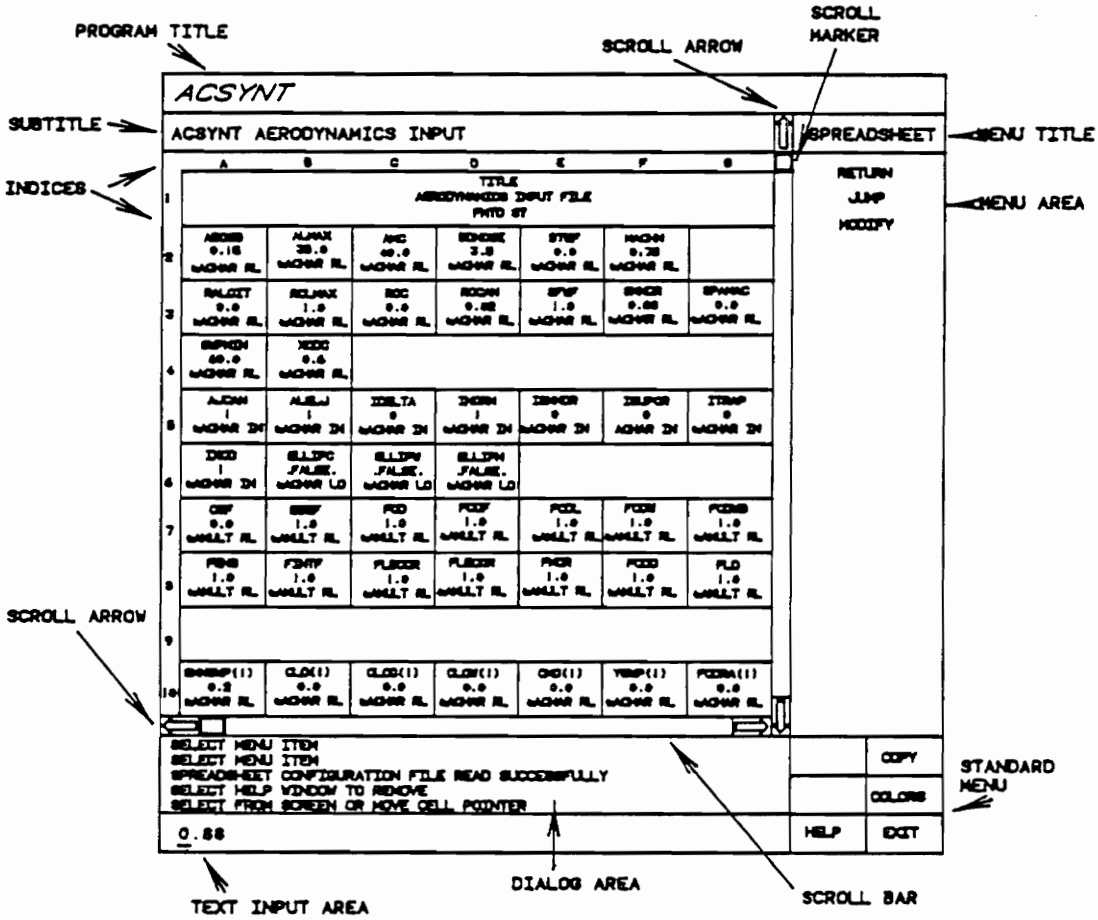


Figure 3. Spreadsheet Screen Layout and Terminology

5.2.1.2 Scrolling

The displayed cells consist of only a portion of a much larger spreadsheet. Figure 4 on page 31 shows the entire field of cells, and its relationship to the spreadsheet window. The term "scrolling" refers to moving the window of currently displayed cells, revealing different parts of the spreadsheet. Several methods of scrolling should be employed. Graphics workstations provide several different input devices, and all of them should be utilized to provide scrolling action. A choice of scrolling devices will allow each user to acquire his/her own personal style of moving about the spreadsheet, enhancing its ease of use.

5.2.1.3 Text Input

Keyboard input for changing cell values should be provided in a manner analogous to that of ACSYNT. The string should be echoed in the text input area of the screen as it is typed. In addition to cell values, the user should be able to type formulae and enter commands that circumvent menu driven operations. As with the various scrolling devices, the use of quick, abbreviated commands will enable the user to develop an individual style of using the spreadsheet.

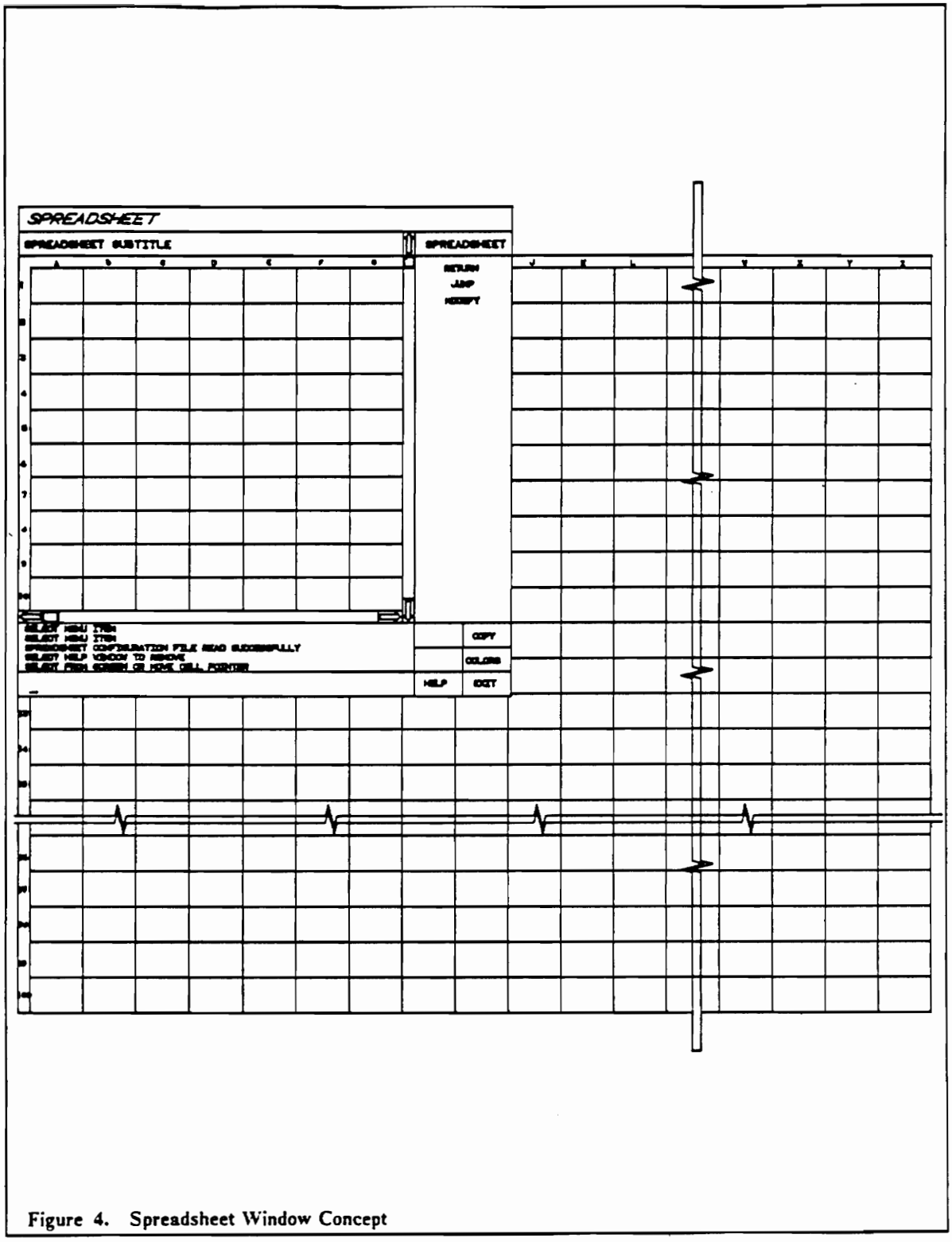


Figure 4. Spreadsheet Window Concept

5.2.2 Software Requirements

Software requirements specify the hardware and software used to code and test the program. Additionally, any specifications that will restrict the data structure of the program are contained in this portion of the requirements.

5.2.2.1 Language Selection

FORTRAN is used for most conceptual design analysis codes, and would make an obvious choice for the spreadsheet coding language. However, the data structure required by the spreadsheet cells makes a language with programmer-defined data types such as C or Pascal attractive. C was chosen as the development language for the spreadsheet for the following reasons:

- User defined data types (as mentioned above)
- Better program control and looping facilities
- Free-form coding (no columnar restrictions)
- Increasing popularity of UNIX workstations in the engineering community, and their universal support of C
- Rich set of type conversion functions (string to integer, real to integer, etc.)
- Powerful standard string manipulation functions

- Standardization of the C language across UNIX platforms
- Typecasting ability between standard data types
- Pointers
- Shorter compile times
- Selective declaration of static variables
- Superior file input/output standard functions
- Standardized preprocessor
- Inline formatting of print statements

5.2.2.2 Data Structure Requirements

Since the C language is specified for the coding of the spreadsheet, a data structure can be created that will exactly match the spreadsheet's needs. The following attributes should be considered in the design of the data structure for each cell:

- Cell location
- Variable name
- Variable value

- Variable type (real, integer, logical, etc.)

- Print format for the variable

- Comment information (for the third line of the cell)

The complete data structure for the spreadsheet should consist of an array of cell data structures, and a block of data to contain any other information specific to one spreadsheet, such as file names and minimum and maximum dimensions.

5.2.2.3 Graphics Standards and Selection

Although standardized programming languages have been in existence since the 1960's, standards for graphics programming have only recently evolved. Graphics standards provide a common set of programming functions for developers to use to eliminate wasted effort when dealing with differing sets of computer hardware [Brun89]. One of the primary drawbacks to the adaptation of graphics standards has been the wide variety of graphics hardware in use and the direct dependence of graphics applications on this hardware [Wamp88a]. The Graphical Kernel System, or GKS, became the first ISO (International Standards Organization) sanctioned standardized set of graphics routines in 1985. GKS was limited, allowing only two-dimensional graphics data and the GKS system did not support graphics data editing functions.

The Programmer's Hierarchical Interactive Graphics System, or PHIGS, was developed in the mid and late 1980's and was accepted as the ISO standard for 3-D graphics in 1988. PHIGS significantly improved upon the GKS standard. PHIGS graphics data

was organized in "structures" which hold sets of graphics primitives such as lines, markers, and polygons, and attributes of the primitives which determine their orientation, color, line style, and visibility. These structures can be placed into hierarchical networks which allow structures to invoke and display one another. Attributes of the elements in each level of the hierarchical network can be passed to, or inherited by, elements below them in the tree.

PHIGS also provides thorough handling of user input via logical input devices. Logical devices enhance the portability of PHIGS-based software by handling the actual physical device in a manner transparent to the programmer and the user. PHIGS logical input devices include:

- Locator - returns one (x,y,z) position
- Stroke - returns a series of (x,y,z) positions
- Valuator - returns real number values
- Choice - returns an integer value
- Pick - returns values identifying an item selected from the display
- String - returns text input

The locator, pick, and stroke logical devices all could use a single hardware mouse, joystick, or digitizing tablet. Regardless of the hardware available, the programmer needs to only specify the correct logical device for input, PHIGS takes care of the rest.

This ability allows the programmer to easily specify several different devices for input, allowing for high user interactivity with different input methods.

PHIGS has been steadily gaining popularity since its adaptation as an ISO standard. It is highly likely that future CAD systems will be developed using this standard to ensure their portability. ACSYNT was one of the first large-scale computer aided design systems to utilize the power of PHIGS, and has been successfully ported to several different workstations.

One of the goals of the spreadsheet design is to develop code that can be easily attached to any existing analysis program. Because of the portability of the PHIGS standard, the chances of the analysis code and the spreadsheet's graphical display co-existing on the same platform are increased tremendously. For this reason, PHIGS was chosen for the manipulation and display of the spreadsheet graphics data.

6.0 Spreadsheet Functional Specification

The requirements of the spreadsheet software identified the objectives of the program, coding guidelines, and specified the interaction of ACSYNT (or any other conceptual design system) with the spreadsheet. The spreadsheet's functional specification gives a more detailed description of the workings of the program, and serves as a template for its design. The next few sections detail the functional specifications for different aspects of the spreadsheet.

6.1 Cell Selection and Changing Cell Values

The user must be able to distinguish the currently active cell from the others around it. (The term "currently active" describes the cell that is ready for input). A highlighting scheme that changes the cell and text color should be used in the spreadsheet. The highlighted cell is the currently active one, and the cell's color should change to yellow

and the value displayed in the cell to red when it is highlighted. The other cells should be colored gray with black text, so that the currently active cell is immediately visible.

Not all areas of the display are covered by cells. When the cell highlight is moved onto one of these empty areas, the empty space should turn yellow with a red border, preserving the continuity of cell highlight movement. This will produce a smooth movement effect, similar to the cursor in a text editor.

After a cell is highlighted by selecting it with the logical pick device (which is most often a digitizing tablet or mouse physical device) it should become ready for input. The value of the cell should change after a new value is entered. Because of the limited amount of space available in the cell, the spreadsheet should check the string and change the format of the value to make it fit into the cell if necessary. For instance, the value:

1000000000000000

should be displayed as:

1E10+15.

The spreadsheet should allow only numerical input into cells that hold numerical data. Numerical input includes the letters "E", "e", "D", and "d" to allow FORTRAN exponential and double precision constants to be entered. This flexibility will require extensive checking of the input string and comparing it with the type of variable that the cell contains. If the input is in error, a message should be displayed in the dialog area that calls the user's attention to the mistake.

6.2 *Scrolling*

The user should be presented with as many scrolling options as possible. The scrolling methods employed by the spreadsheet include:

- Scroll bars
- Scroll arrows
- Valuators
- Choice device

Scroll bars on the right and bottom of the cell display area show the relative location of the highlight in the spreadsheet, and will move the highlight to a new relative position when that position is selected. The scroll arrows located at the extremes of the scroll bars move the highlight one cell in each direction.

The valuator logical device, which usually corresponds to either hardware or software dials on the workstation, should be used to move the cell highlight. When the valuator value is increased, the highlight should move in a positive (up or right) direction, and vice versa. This action is in keeping with the common stereotype for dial movement [Join72].

The choice device can become a cursor pad, with arrow and page up and down keys. The arrow keys should advance the cell highlight one unit in any direction, while the paging keys should move the cell highlight one page, or 10 cells, at a time.

6.3 Help

Two help modes should be incorporated into the spreadsheet. Help on the operation and use of the spreadsheet will assist novice users. The other help mode will give the user information on the variable in the currently highlighted cell.

6.3.1 Functional Help

Generalized help should be available for spreadsheet functions, and accessed by selecting the HELP standard menu item. When help is requested, a full screen help window should appear over the cell display area with information about the current menu items. After the user has read the information on the help screen, a simple method for removing the help display window should be used to redisplay the cells. A spreadsheet screen with a generalized help window overlaid upon it is shown in Figure 5 on page 41.

6.3.2 Cell Help

The cell help feature of the spreadsheet will greatly aid the user during everyday spreadsheet use. After the current cell is highlighted, selecting with the pick device a second time should produce a help window with information about the currently selected variable. Figure 6 on page 42 shows an ACSYNT spreadsheet with a cell help window displayed.

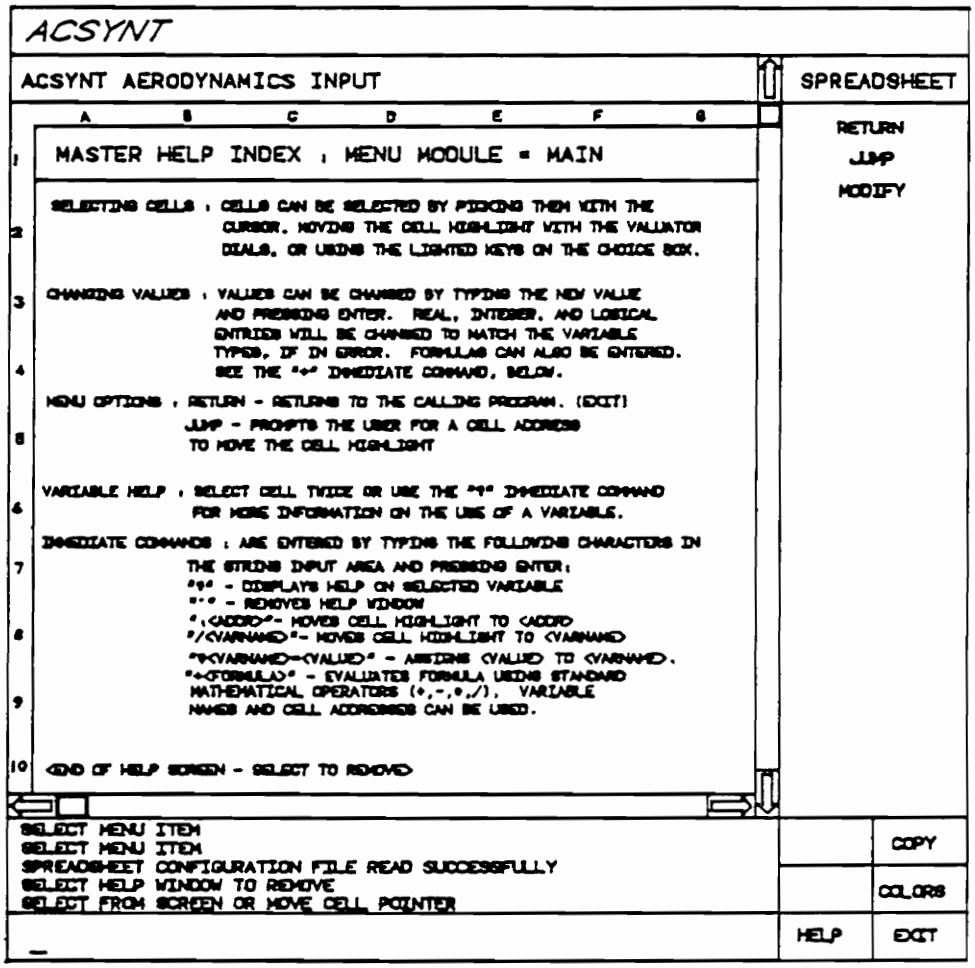


Figure 5. Spreadsheet Screen Layout for Functional Help

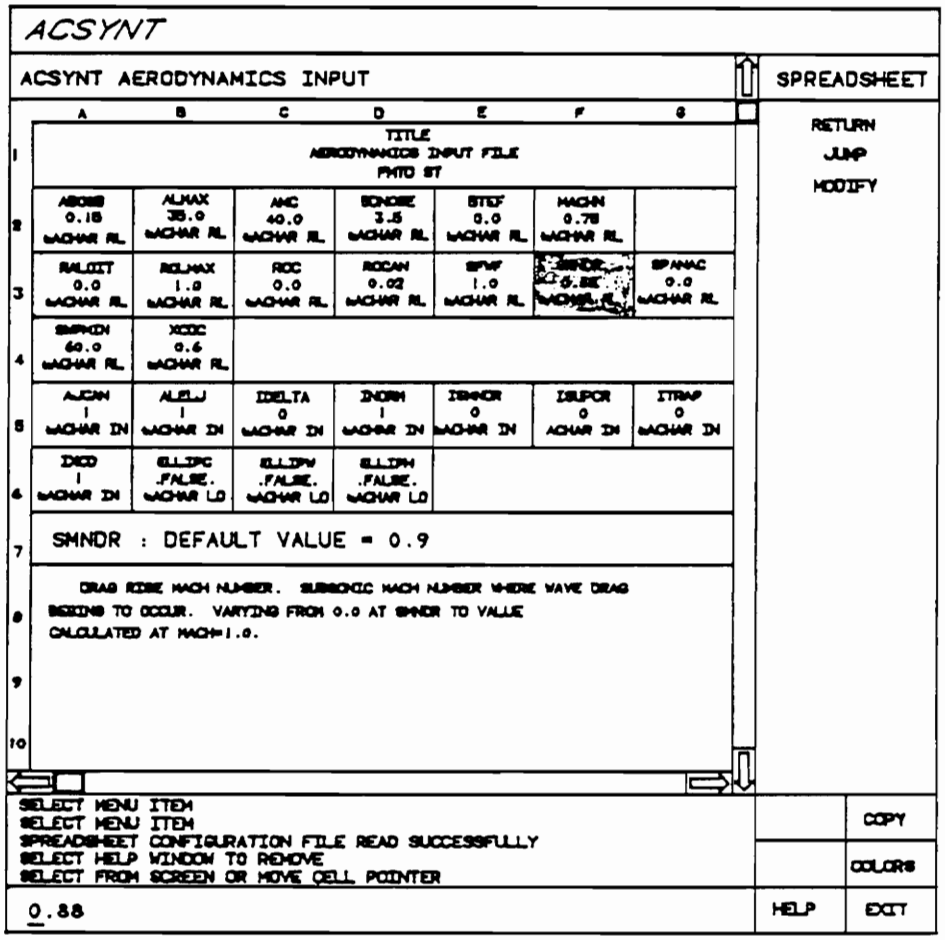


Figure 6. Display of Help for Individual Cells

A brief description of the variable, its use in the analysis system, and any formulae that relate this cell to other quantities in the analysis program should be included in the help information. This information should appear in a manner such that the current cell is not obscured.

As with the generalized help, the help information should be removed by selecting the help window area. Selecting the cell another time should also remove the window, resulting in a toggling effect.

6.4 Immediate Commands

Immediate commands that can be typed by the user should be provided. These single-character commands will enhance the performance of the experienced user by allowing quick manipulation of the spreadsheet contents. Immediate commands that should be designed into the spreadsheet include:

- **Slash "/ < variable name > "** - search for, display, and highlight the cell containing the variable name.

- **Colon ": < address > "** - search for, display, and highlight the cell at the specified address.

- **Question mark "?"** - display the help information for the currently active cell.

- **Exclamation point "!"** - remove the generalized or cell help window.

- **Dollar and equal sign "$ < variable name > = < value > "** - search for, display, highlight, and assign the value to the cell containing the variable name.

- **Plus sign "+"** - signify the beginning of a formula for evaluation (formula evaluation is described in detail in the following section).

- **Double quote """** - display the last formula entered.

The immediate commands will speed simple spreadsheet operations by keeping the user's hands on the keyboard. If a value has just been entered for one cell, and another cell needs to be highlighted, a search command could be used instead of using the pick device.

6.5 Formula Calculation

As an aid to filling the cell values, a formula calculation ability should be incorporated. Because the spreadsheet is not intended for all-purpose use (a'la Lotus 1-2-3), but as an input aid for conceptual design programs, the formulae need not be retained. However, the ability to type mathematical expressions for a cell's value and have them evaluated will assist the user. The spreadsheet's formula evaluation capabilities will eliminate the need for using a desktop calculator along with the program.

Mathematical capabilities supported by the spreadsheet should include addition, multiplication, subtraction, division, exponentiation, and nested levels of parentheses. The formulae are evaluated in the normal algebraic order of operations, with exponentiation calculated first, followed by multiplication and division, then addition and subtraction. Evaluation is analogous to FORTRAN and C formula statements.

Expressions should be able to reference values of other cells, either by address or variable name. The ability to reference other cells in formulae is essential for calculating design quantities that depend on other variables in the same spreadsheet. For instance, the formula expression:

+0.5*RHO*(VMAX*528/360)^2

calculates a dynamic pressure value using the design variables RHO and VMAX. Instead of looking in the spreadsheet for the design variable values, the user can simply enter their names, and the expression will be calculated.

The formula evaluation routines must trap most of the common errors that occur. These include:

- Division by zero
- Unbalanced parentheses
- Negative numbers raised to a non-integer power
- Unknown variable addresses or names in a formula
- General syntax errors (e.g. using two division operators together)

After one of these errors occurs, significant feedback should be given to the user. The cell highlight color should change to red, the color most often associated with severe warnings [Sand87], and information about the error should appear in the dialog area.

6.6 *Jump Command*

A JUMP item should be provided on the spreadsheet menu to allow the user to enter an address of a cell to display and highlight. The operation of this function is identical to the colon (":") immediate command, but it is located on the menu. Novice users not familiar with immediate commands can access this function readily.

6.7 *Spreadsheet Layout Modification*

The spreadsheet cells are created each time the spreadsheet is started. Trying to add a value to a region where there is no cell by simply entering a value will result in an error message. The spreadsheet software should have a modification facility where new cells can be created, existing ones edited, and the layout of the spreadsheet altered.

Creation of new cells will require adding data structure information in an interactive fashion. The variable name, type, format, and comment information must be displayed and accepted for input. Editing existing cell contents should work in the same manner,

allowing data structure items to be modified. These functions will become useful when the spreadsheet is adopted in a new analysis code, or when input variables of an existing analysis system are modified.

The ability to arrange the input variables in a logical, tabular fashion is one of the strengths of the spreadsheet. However, one user's "ideal" grouping of variables may not coincide with another's idea of the "best" arrangement. Thus, the spreadsheet modification software should allow for the following functions to change the layout of the spreadsheet:

- Insert row
- Insert column
- Delete row
- Delete column
- Delete range
- Move range

All of the above functions should use the dialog area to keep the user informed of the current operation. The insert and delete operations should prompt the user for confirmation before completing their tasks.

Because the insert and delete options could be easily confused by a novice user, care should be taken to distinguish between the operations. One approach used to

differentiate between the modification options described above is the color of the highlighted cell. When the delete option is chosen, selected cells and ranges should turn black with red borders and yellow text. Before inserting rows, the cell highlight should become white with blue text. These color changes provide instant feedback about the operation about to take place. If the user inadvertently chooses an improper function, an ABORT menu item should be available to allow a quick escape before any damage is done to the spreadsheet structure.

After the user modifies the spreadsheet layout, an option should be provided to save the current spreadsheet cell arrangement as a default. This default configuration should be used in place of the original one whenever the spreadsheet is invoked.

7.0 Design of the Spreadsheet Software

The previous section described requirements for the spreadsheet. This section examines the actual design of the software, including the algorithms used. The design of all major components of the spreadsheet are outlined.

7.1 Design Priorities

Before detailing the design of each portion of the spreadsheet software, the priorities that drove the design are listed below. They are listed in order of importance, and design trade-offs were made considering their relative priority.

1. **Reliability**: The spreadsheet must work normally under all conditions and be free of errors.

2. **Ease of Use**: The level of effort required to use the spreadsheet must be lower than that to use another input method, where one is available (e.g. opening an editor window and adding analysis variables to a file "manually"). The on-line help requirement contributes to this design priority by providing the user with instantaneous help for all analysis quantities.

3. **Portability**: The spreadsheet code must be portable with minimum modification, if any. Use of the C programming language and the PHIGS graphics standard will ensure that this design priority is satisfied.

4. **Flexibility**: In order to use the spreadsheet input for the creation of analysis data for different programs, the spreadsheet code must be structured such that the analysis variables can be changed or modified with little or no changes to the analysis program. Help screen data must be easily updated so that when the use of analysis quantities change, the spreadsheet can be kept up to date.

5. **Speed of Execution**: The program's speed must not hamper its ease of use. This requirement includes the help utility, which must scan large amounts of data to serve the user's requests.

7.2 Spreadsheet Data Structure

The data structure of the spreadsheet holds information about each cell and the structure of the spreadsheet. Most of the cell data is read from a file when the spreadsheet is started, and is written when the user leaves the spreadsheet.

The spreadsheet data structure was designed to be memory sparing. Most CAD programs use large amounts of computer memory, and the spreadsheet should not add severely to this overhead. Because each cell can hold either a real, integer, character, or logical value, four data types are possible for each cell value. One solution to representing each available data type in a cell would be to have four different variables of different types for each cell value. However, only 25 percent of the data structure would be filled at any time, resulting in excessive empty memory space. This approach was considered to be too wasteful for use in the spreadsheet.

Instead of four different data types to represent one cell value, the spreadsheet uses only one. All data is held in a string (array of character) format that is 11 characters long. Numeric values are converted to string representations for storage. Because the C language places a null character at the end of the string for termination, the maximum available string length is 10 characters. This is sufficient to hold any value, real or integer, that can be represented by the computer. Exponential form is used to control the size of the numeric string, and hence, large numbers are expressed as 10 raised to a power.

When string input is received, the string is placed directly into the data structure after it is thoroughly checked. The string is displayed directly in the cell area, without conversion from a numerical (i.e. integer or real) representation.

Strings longer than 10 characters are held in a special "string" region in the spreadsheet data structure. There is a maximum number of 10 strings of up to 80 characters that can be contained in the spreadsheet. (This restriction can be lifted by modifying a line in the program's include file and re-compiling the code). These "string" cells appear larger than regular cells on the screen, and are scaled to fit the size of the string.

Character variables of 10 characters or less can be placed in regular cells and are displayed in a normal fashion.

There are two different data structure types: "cell_contents" and "cell_data". Each "cell_contents" data structure holds information for one cell. As specified in the requirements, this includes:

- Variable name (11 character string)
- Value contained in the cell (11 character string)
- Comment or FORTRAN namelist name (9 character string)
- Variable type (real, integer, etc.) (4 character string)
- Output format (6 character string)
- x and y cell address locations
- PHIGS pick identifier for the cell
- PHIGS structure number for the cell
- Delete flag that controls whether a cell is retained or erased the next time that cell data is saved

The "cell_data" data structure contains a "cell_contents" data structure for each cell and the following information about the current state of the spreadsheet:

- String data set for long character strings
- Title of the set of spreadsheet data
- Spreadsheet data and help file names
- Total number of cells in the spreadsheet
- PHIGS pick identifiers of the last two cells selected
- Largest cell indices in the spreadsheet
- Current cell X and Y location
- Minimum and maximum cell addresses currently displayed
- X and Y locations of last window limits in a scrolling operation

Figure 7 on page 54 shows the different data items necessary to display each cell. Figure 8 on page 55 details the data items that control spreadsheet scrolling and other attributes of the current status of the spreadsheet.

7.3 Spreadsheet Configuration File Format

Portions of the spreadsheet data structure are written to disk at the end of spreadsheet use. The files created are also read when the spreadsheet is started. These files contain

```
struct cell_contents
    {
    char name[11];              /* variable name */
    char def_val[11];           /* value of cell */
    char namelist[9];           /* namelist name or comment */
    char var_type[4];           /* variable type: real, int, etc */
    char format[6];             /* string for variable format */
    char ss_x_pos;              /* x position in ss 'A', 'B', etc */
    int ss_x_loc;               /* x location corr. to pos. (A=1,B=2) */
    int ss_y_loc;               /* y location of cell */
    int pick_id;                /* pick id for cell */
    int snum;                   /* PHIGS structure number of cell */
    int delete;                 /* flag for cell deletion; 0=ok, 1=del */
    };
```

Figure 7. Data Structure Items for Each Cell

```
struct cell_data
    {
    struct cell_contents cell[MAXCELLS];      /* one cell_contents for each cell */
    char strings[10][81];                     /* group of strings for long ch vars*/
    char ss_title[40];                        /* subtitle for spreadsheet */
    char ss_file_name[15];                    /* spreadsheet config. file name */
    char help_file_name[15];                  /* spreadsheet help file name */
    int current_exist;                        /* flag if the current cell exists */
    int numcells;                             /* number of cells in current sheet */
    int last_picked;                          /* pickid of previous cell picked */
    int last2_picked;                         /* pickid two cells ago */
    int high_cell_x;                          /* x limit of spreadsheet */
    int high_cell_y;                          /* y limit of spreadsheet */
    int current_cell_x;                       /* x location of current cell */
    int current_cell_y;                       /* y location "  "     " */
    int min_x_disp;                           /* minimum and maximum cells displayed */
    int max_x_disp;                           /*      in current spreadsheet window */
    int min_y_disp;
    int max_y_disp;
    float last_xfm_min_x;                     /* actual window locations of last */
    float last_xfm_max_x;                     /*      spreadsheet scroll in x    */
    float last_xfm_min_y;                     /*      and y                      */
    float last_xfm_max_y;
    };
```

Figure 8. Data Structure Items for Entire Spreadsheet

the spreadsheet title, and the cell names, variable types, contents, and locations necessary to display the spreadsheet data. A configuration file is necessary to successfully invoke the spreadsheet, and is easily modified to suit the needs of a particular analysis program.

The format used is shown in Figure 9 on page 57. As shown in the figure, the first line reads "SPREADSHEET CONFIGURATION FILE" to identify the file. This is followed by the spreadsheet subtitle line and two positioning and column title lines which aid in identifying each column of the file.

The cell addresses are entered with a letter and a number, e.g. "A 1". The spreadsheet converts the letters to integer positions and stores these in the data structure for quick lookup during execution.

During configuration file read or write, if cell addresses larger than 100 or "Z" are encountered, an error message is sent to the dialog area of the screen. The erroneous cell position is set to Z if the X location is found to be in error, and 100 if the Y location is incorrect.

7.4 Cell Display Strategy

Each cell occupies its own PHIGS structure. All the cell structures are created before the spreadsheet is displayed. The cell structures are not "used", or actually displayed, until the spreadsheet is scrolled over their location. At that time, the limits of the

```
SPREADSHEET INPUT FILE
ACSYNT AERODYNAMICS DATA
  NAME        COMMENT  VT    FMT    X  Y     DEF_VAL
1234567890  12345678  123  12345  1  nn  1234567890
TITLE       FMTD     ST   A80    A  1  AERODYNAMICS INPUT TITLE
ABOSB       $ACHAR    RL   F7.3   A   2  0.15
ALMAX       $ACHAR    RL   F7.3   B   2  35.0
AMC         $ACHAR    RL   F7.3   C   2  40.0
BDNOSE      $ACHAR    RL   F7.3   D   2  3.5
BTEF        $ACHAR    RL   F7.3   E   2  0.0
MACHN       $ACHAR    RL   F7.3   F   2  0.75
RALOIT      $ACHAR    RL   F7.3   A   3  0.0
RCLMAX      $ACHAR    RL   F7.3   B   3  1.0
ROC         $ACHAR    RL   F7.3   C   3  0.02
ROCAN       $ACHAR    RL   F7.3   D   3  0.02
SFWF        $ACHAR    RL   F7.3   E   3  1.0
SMNDR       $ACHAR    RL   F7.3   F   3  0.9
SPANAC      $ACHAR    RL   F7.3   G   3  0.0
SWPMAX      $ACHAR    RL   F7.3   A   4  60.0
SWPMIN      $ACHAR    RL   F7.3   B   4  0.0
XCDC        $ACHAR    RL   F7.3   C   4  0.6
XCDW        $ACHAR    RL   F7.3   D   4  0.6
AJCAN       $ACHAR    IN   I3      A   5  1
ALELJ       $ACHAR    IN   I3      B   5  1
IDELTA      $ACHAR    IN   I3      C   5  0
INORM       $ACHAR    IN   I3      D   5  1
ISMNDR      $ACHAR    IN   I3      E   5  0
ISUPCR      $ACHAR    IN   I3      F   5  0
ITRAP       $ACHAR    IN   I3      G   5  0
IXCD        $ACHAR    IN   I3      A   6  1
ELLIPC      $ACHAR    LO   L1      B   6  .FALSE.
ELLIPH      $ACHAR    LO   L1      C   6  .FALSE.
ELLIPW      $ACHAR    LO   L1      D   6  .FALSE.
CSF         $AMULT    RL   F7.3   A   7  0.0
ESSF        $AMULT    RL   F7.3   B   7  1.0
```

Figure 9. Spreadsheet Configuration File Format

current window are found, and those cells inside the limits are displayed. Alternatively, displaying all the cells at once and scrolling the window over a field of many cells results in excessive overhead for PHIGS, and poor performance.

At the beginning of this thesis, the program was coded so that all the cells formed one large structure. The structure was always displayed, and the spreadsheet window was moved over the field of cells. Each spreadsheet cell has about 15 PHIGS structure elements, and the single structure could get very large for spreadsheets with many cells. For sufficiently large spreadsheets, (over 200 cells) the interaction times were unacceptable. Because of poor interaction times, the display strategy described above was employed. The current method performs about four to five times faster than the one-structure approach.

7.5 Scrolling

Because spreadsheet scrolling is essential, all the scrolling functions were consolidated in one function called **scroll_sheet**. Using only one routine increases the code's reliability. The function accepts the number of cells in the x and y directions that the spreadsheet highlight is to be moved. Both positive and negative numbers can be passed to the function, with positive indicating increasing index numbers in the x and y directions.

Before the spreadsheet is scrolled or moved, the highlighting of the current cell is removed. The spreadsheet window is moved the correct number of cells in order to

display the new "scrolled-to" cell. It is highlighted by changing its background and text colors.

If the scrolling routine is called with a number insufficient to move the spreadsheet window, the cell on the current display corresponding to the new position will be highlighted. If no cell exists in the cell position, a blank highlight (yellow box) will appear in the location, as stated in the requirements.

7.6 Help

Two kinds of help were specified in the requirements, cell help for each variable in the spreadsheet, and generalized help for spreadsheet operation. The program was designed so that both kinds of help use the same set of routines. A flag is passed, along with a menu or cell help index, to the help modules that determines which kind of help was requested.

All help information is stored in ASCII files that can be created from the program's documentation. Most word processors can write format-free ASCII files, making the creation of help files easy. The following guidelines must be observed while creating help files for cell help:

- Each variable must be in upper case at the beginning of a line of text.

- After the variable name, the default value should appear, separated from the variable name by one or two spaces.

- The description of the variable should follow the default value, separated by one or two spaces. The description can continue for up to eleven lines, and the lines should be no longer than 70 characters each.

- Before starting the next variable's information, separate the help information with four asterisks on a new line.

Figure 10 on page 61 shows the format of a spreadsheet variable help file.

Spreadsheet function help information is stored in the same manner, with the menu module name replacing the default value. The menu module name should have no spaces. A sample of the spreadsheet generalized help file is shown on Figure 11 on page 62. As with the variable help file, help information for different menu modules is separated by four asterisks.

Per the requirements, spreadsheet variable help can be summoned when the cell is selected twice, and removed when the cell is selected a third time. The pick identifiers for the last two cells selected are stored in the spreadsheet data structure for this reason. The exclamation point and question mark immediate commands are also used to remove and request help information. Selecting the standard menu item "HELP" displays a full-screen help window on the current menu item.

When variable help is selected, the variable name is looked up in the help file. When it is found, a help window is opened in the correct portion of the screen (away from the

SPREADSHEET HELP FILE
ABOSB 0.15 Ratio of body base area to maximum body area (an estimate to approximate separation on the body).
****

ALMAX 35.0 Angle of attack for Clmax.
****

AMC 40.0 Maximum angle of attack for subsonic compressibility or interference. It is recommended to use the default.
****

BDNOSE BDMAX Width of the fuselage at the canard or in front of the wing (ft). Used in calculating the pitching moment contribution of the fuselage as well as spreading out the canard when NCAN = 2.
BDMAX is in geometry namelist FUS.
****

BTEF 0.0 Boattail exposure factor. Only works with pods (PODREF # 0). Number of engine cowls which contribute to drag; e.g., 3 engines, but 1 has no boattail (inside body), then BTEF =.66667, (2/3).
****

CLO 10*0.0 Additive CL (for misc. such as body) versus SMNSWP, array of 10.
****

CLOW 10*0.0 Wing CL at zero angle of attack versus SMNSWP, array of 10.
****

CMO 10*0.0 Zero lift pitching moment as function of SMNSWP, array of 10.
****

MACHN 0.75 Desired normal component of Mach number. Used for variable sweep/oblique wings. Wing will be swept to keep normal component to value specified.
****

RALOIT 0.0 Loiter Radius – function unknown. Used with ICODE = 11.
****

RCLMAX 1.0 CLMAX scale factor for use with flaps.
****

Figure 10. Variable Help File

```
SPREADSHEET HELP FILE
 MAIN   Main
' Selecting cells: Cells can be selected by picking them with the
'                  cursor, moving the cell highlight with valuator
'                  dials, or using the lighted keys on the choice
'                  box.  When the cell highlight passes over an area
'                  not occupied by a cell, the highlight becomes a
'                  red outline.
'
' Changing Values: Values can be changed by typing the new value
'                  and pressing ENTER.  Real, integer, and logical
'                  entries will be changed to match the variable
'                  types, if in error. Formulas can also be entered.
'                  See the "+" immediate command, below.
'
' Menu Options:   RETURN – returns to the calling program. (EXIT)
'                 JUMP   – prompts the user for a cell address to
'                  place the cell highlight.
'
' Variable Help:  Select cell twice or use the "?" immediate command
'                 for more information on the use of a variable.
'
' Immediate Cmds: are entered by typing the following characters in
'                 the string input area and pressing ENTER:
'                 "?" – displays help on selected variable
'                 "!" – removes help window
'                 ":<addr>" – moves cell highlight to <addr>
'                 "/<varname>" – moves cell highlight to <varname>
'                 "$<varname>=<value>" – assigns <value> to <varname>
'                  and moves cell highlight to the cell occupied
'                  by <varname>
'                 "+<formula>" – evaluates formula using standard
'                  mathematical operations (+,–,*,/,^).  Variable
'                  names or cell addresses may be used.
'
' <END OF HELP SCREEN – SELECT TO REMOVE>
****
 MODIFY   Modify
' The MODIFY option has different menu items for altering the
```

Figure 11. Generalized Help File

currently active cell) and the variable name and default value are displayed. Lines of information are read from the file and placed on the help window until the terminating four asterisks are found.

When requested by selecting HELP from the menu, generalized help works in the same manner as variable help. An index for the current menu item, instead of a variable name, is looked up in a different help file. A larger window is created, and more text is displayed.

The help window is shown until it is selected or an exclamation point immediate command is entered. When the request for removal is processed, the PHIGS structure containing the window is deleted from the workstation, restoring the full-screen view of spreadsheet cells.

7.7 String Input

String input for the spreadsheet is very versatile. As requested in the requirements, cell addresses, commands, and variable values can be entered and processed. Because of its versatility, considerable effort was invested in its design.

After the string is received from the PHIGS input queue, it is capitalized to remove any case-sensitive dependencies that might arise in the spreadsheet. All leading blanks are stripped to give the user the ability to type strings anywhere in the string input area.

Next, the input "type" must be determined. There is an input type for each immediate command, formulae, and regular string input.

To determine the input type the first letter of the string is examined. If any of the immediate command characters are encountered, the type number is assigned and the first character is removed. The proper routine to utilize the remainder of the input string is called to execute the immediate command.

If a plus sign is the first character, the string is assumed to be a formula, and the mathematical parsing routine is subsequently called. If none of the immediate command characters are found, the string is added to the cell data structure as the cell's value.

See Figure 12 on page 65 for a flow diagram of the string input process.

7.7.1 Cell Values

Simply typing a new value for a cell causes extensive error checking to take place. The variable type of the currently active cell is found from the data structure, and the input string is "corrected" for variable type.

Cells with integer or real contents accept only numeric input. Text input to these cells causes an error message to appear in the dialog area. Integer values are truncated past the decimal point. Real and double precision values are screened to see that the number of decimal points is no more than one, and that the size of the string will fit in the cell. If it will not fit, the string is placed in the cell using exponential notation.

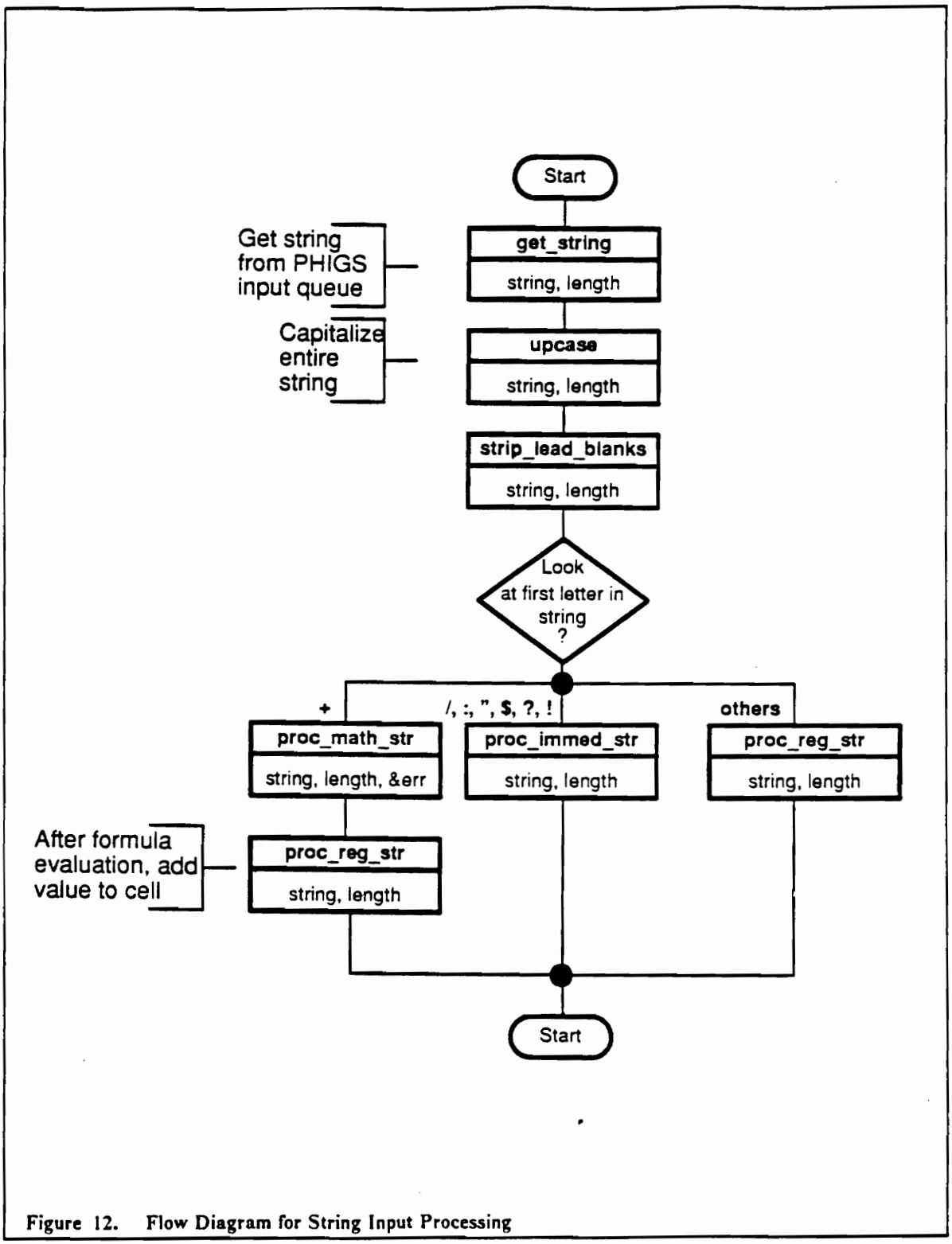


Figure 12. Flow Diagram for String Input Processing

7.7.2 Immediate Commands

After the first letter is removed from the immediate command string, routines are called based upon the input type that was determined from the leading letter. If the search ("/") command was used, the variable name in the input string is found in the data structure. The x and y location for the variable is then found, and the spreadsheet is scrolled to its position. The jump immediate command (":") is used in much the same way, scrolling the spreadsheet to the cell address following the colon.

The two help immediate commands require no additional parsing. If the exclamation point is typed, the current help window is removed. When the user enters a question mark, the help lookup routine finds help for the currently selected analysis variable.

Direct assignment of analysis variables can be accomplished using the dollar sign/equal command combination. The spreadsheet processes the variable name after the dollar sign in the same manner as the search command, finding the variable and scrolling the spreadsheet such that the cell is in view. Then the portion of the string after the equal sign becomes the value of the cell in the data structure after checking the variable type. If a plus sign is the first character in the string following the equal, the rest of the string is treated as a formula and its value is added to the data structure after evaluation.

7.8 Formula Processing

The design requirements stated that a formula processing capability be included. As required, the spreadsheet has an extensive formula handling capability that can use embedded cell names and addresses, nested parentheses, and the following mathematical operators: addition, subtraction, unary minus, multiplication, division, and exponentiation.

An existing implicitly recursive algorithm for evaluating mathematical expressions from string input was used [Grog84]. Although it originally supported only addition, subtraction, multiplication and division, it was extended to allow exponentiation. This algorithm divides a mathematical formula into characters, numbers, factors, terms, and expressions. Separate functions are used to read each type of division, and calculations are made based on their relationship to one another. For instance, in the taxonomy described above, factors are comprised of numbers, and are combined by multiplication or division. Terms are made up of factors and addition and subtraction are the only permissible operators. Expressions are enclosed by parentheses and consist of terms.

Cell values are included before the formula string is evaluated. The string is examined character by character. When a cell variable name or address is encountered, the name (or address) is removed from the string, and the value of the cell put in its place. The string representation of the cell value is used in formula evaluation. Each digit in the string is multiplied by a power of ten corresponding to its distance from the decimal point to obtain its numerical value during formula processing.

Cell addresses have priority over variable names when this phase of formula processing is executed. Hence, subscripted array variables must be referenced by their cell addresses in formula calculation. If subscripted variables are placed in formulae, the subscript number in the variable name will be interpreted as a cell address, and an error will occur.

Exponentiation and unary minus operations were added by the use of a formula preprocessor. The preprocessor places parentheses around exponentiation operations to ensure that they are evaluated in the proper order. The unary minus portion of the preprocessor places parentheses and zeros around the terms preceded by a unary minus.

After all cell values have been replaced, and the formula preprocessor has made the correct adjustments to the original string to ensure that the proper order of operations will be followed, the evaluation routines compute the value of the formula. During evaluation, error checking is performed for unbalanced parentheses, division by zero, and negative numbers raised to a non-integer power. If errors are encountered, the color of the cell is changed to red, and an error message is displayed in the dialog area as specified in the requirements. If no errors are encountered, the value is replaced according to the method outlined above. The formula evaluation process is depicted in Figure 13 on page 69.

7.9 Interactive Spreadsheet Data Structure Modification

The user of the spreadsheet can modify data structure items by editing the configuration file. However, this can be a time-consuming and tedious task. The spreadsheet

User Enters: **+0.5*(3.125*cell1+a10)*–2^2**

Step 1:

Replace Cell Values:

	A
CELL 1 **10.5** **SAMPLE**	**10** **CELL 2** **100.0** **SAMPLE**

Formula Becomes:**+0.5*(3.125*10.5+100.0)*–2^2**

Step 2:

Place Parentheses Around Unary Minus:

Formula Becomes:**+0.5*(3.125*10.5+100.0)*(0–2)^2**

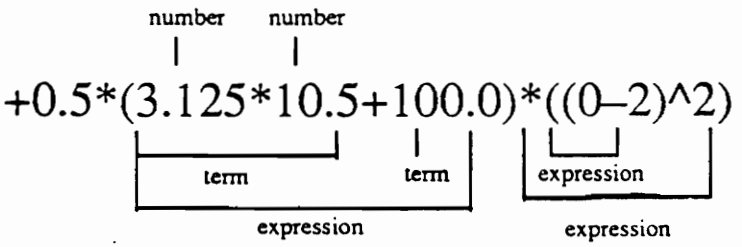
Step 3:

Place Parentheses Around Exponentiation Operation:

Formula Becomes:**+0.5*(3.125*10.5+100.0)*((0–2)^2)**

Step 4:

Separate Into Numbers, Factors, Terms, and Expressions:



Step 5:

Evaluate:

Formula Becomes:**265.625**

Figure 13. Formula Evaluation in the Spreadsheet

incorporates modification routines that allow the user to change data structure items such as cell location, variable type, comment information, and format interactively, eliminating the need to edit the spreadsheet configuration file.

The interactive spreadsheet modification facilities are accessed by using the MODIFY menu item. Options for inserting and deleting portions of the spreadsheet, creating new cells, and editing the contents of existing ones can be chosen. Another option (SET DEFAULT) is presented for updating the changes to the master configuration file. A tier chart of the options available in the MODIFY menu item is shown in Figure 14 on page 71.

The INSERT and MOVE options simply re-assign the positions of cells in the data structure. In addition to re-assigning cell locations, the DELETE option sets a deletion flag for each cell in the deleted range that prevents it from being displayed. When the cell information is saved at the end of spreadsheet use, only those cells with the delete flag turned off are written, erasing the others from the spreadsheet data structure the next time the configuration file is read.

The data structure manipulation for the INSERT and DELETE menu options is accomplished by one function. As with the spreadsheet scrolling function, consolidating all the "action" in one routine contributes to reduced debugging and increased reliability of the code.

When the user wishes to add a new spreadsheet cell, the CREATE option displays a list of data structure items on a small window. Items on the list can be changed by selecting them with the pick device and entering the new values. After the new items are entered, a cell is added to the spreadsheet with the new data.

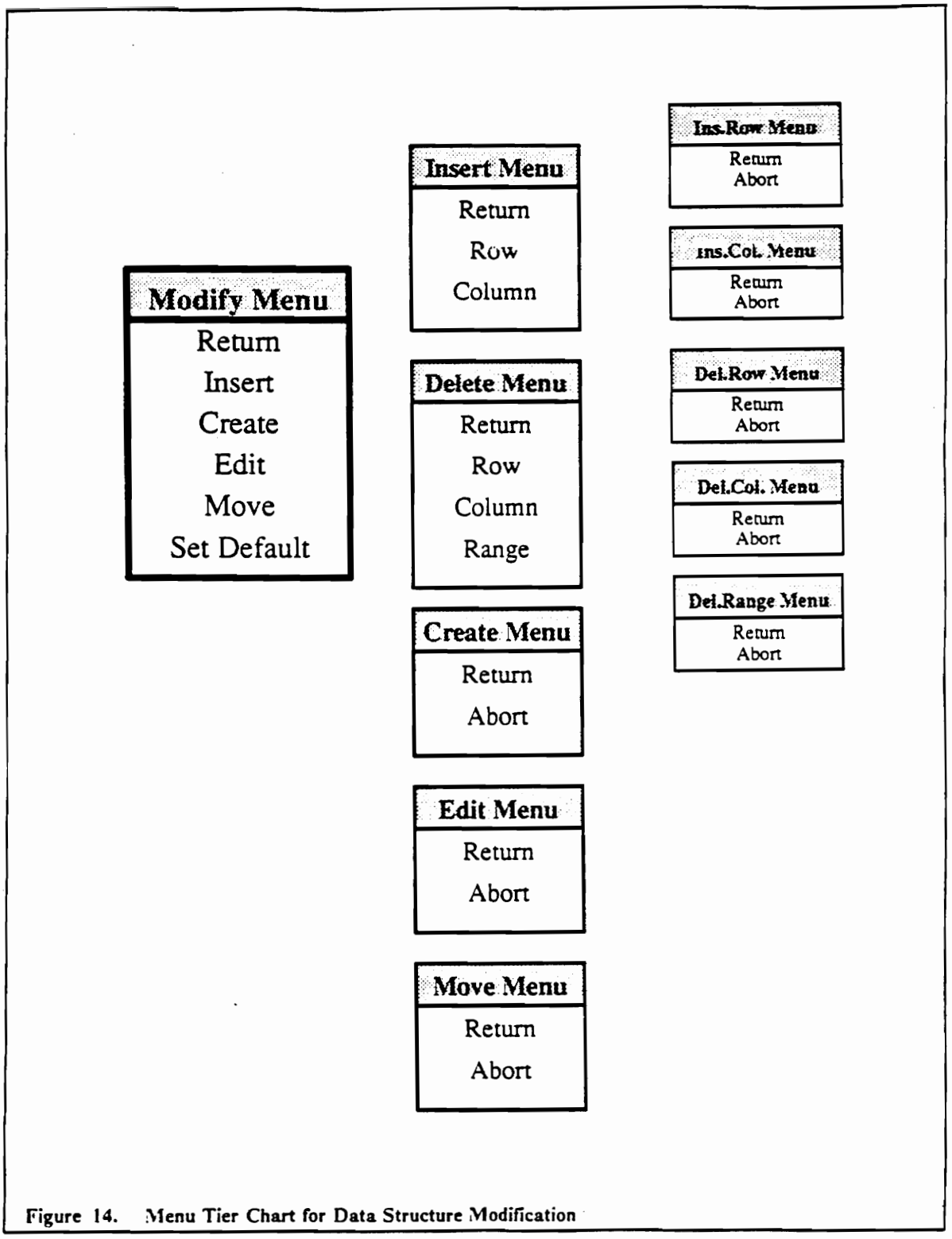


Figure 14. Menu Tier Chart for Data Structure Modification

The EDIT option works in the same manner, displaying the contents of a cell data structure. As the user modifies the items, the data structure is updated.

A special set of "2-D window" functions were created to handle the display and selection of items from the windows discussed in the EDIT and ADD options. One routine sets up the window, controlling the background and text color, and displaying the title. Another routine fills the lines of the window, assigning pick identifiers as determined by the programmer. The third 2-D window function displays the window and prepares it for input.

Because there are several different menu levels in the modification system, a standard naming convention was developed for the different modules that comprise one menu level. The naming convention proceeds as follows, with "*" designating the name of the function (e.g. del_col for delete column):

1. **ss_*** - main routine in menu module - displays new menu and dialog area messages

2. **get_*_input** - gets PHIGS input and calls functions to process input from each available logical input device

3. **proc_*_pick** - acts on pick input for a menu module

4. **proc_*_menu** - processes menu item picks

8.0 Integration of the Spreadsheet with ACSYNT

This section of the thesis discusses the requirements, functional specification, and design of the ACSYNT/spreadsheet integration system. It should prove helpful for future readers of this document who wish to utilize the capabilities of the spreadsheet in a different analysis program.

The spreadsheet was coded and debugged on an IBM RT PC workstation prior to linking it with an analysis system. When coding was complete, the ACSYNT computer aided conceptual design system for aircraft was selected as a testbed for application of the spreadsheet. ACSYNT is used extensively and is developed in the workplace of the author, and the author's extensive experience with both its CAD and analysis systems aided the integration process.

8.1 Integration Plan and Requirements

As mentioned in the "ACSYNT as a Case Study" section of this thesis, each discipline of conceptual aircraft design analysis has a separate analysis module in ACSYNT. These modules are:

- Geometry
- Trajectory (Mission Profile and Performance)
- Aerodynamics
- Propulsion
- Stability
- Weights
- Supersonic Aerodynamics
- Economics
- Takeoff Performance

To facilitate input to these modules, each is represented in ACSYNT's CAD system as a menu item. The input for the modules is processed by different interactive menu options and an analysis input file can be written based on the user input. The input file

is then processed by the analysis routines of ACSYNT. Output of the analysis can be examined by ACSYNT's interactive graphing, geometry, and file scanning features.

To consolidate all the input data for one of the analysis modules of ACSYNT, a spreadsheet should be used for each discipline module. The ACSYNT Control program and the COPES optimization system should also accept spreadsheet input. Per this plan, ACSYNT will have nine separate spreadsheets to aid input for the analysis routines.

Using the CAD facility to create the geometry of an aircraft requires data to be sent to both the Geometry module (for fuselage length, fineness ratios, wing area, taper ratio, etc.) and the Propulsion module (inlet type, capture area, design Mach number, etc.). Thus, the geometry creation system of ACSYNT must have a link to the spreadsheets in the Geometry and Propulsion modules.

Because the nine spreadsheets that comprise the full set of available ACSYNT input data must be able to get their information from existing ACSYNT analysis input files, a software system for reading the ACSYNT input file must be created. Likewise, the information in the nine spreadsheets must be consolidated in an ACSYNT input file for processing by the analysis code. This requirement dictates that another software system for writing ACSYNT input files from spreadsheet data must be created.

8.2 Functional Specification of Integration Software

As outlined above, the ACSYNT integration problem breaks down into three distinct areas:

- Adding Spreadsheet Modules into ACSYNT
- Linking the Geometry CAD Facility with the Geometry and Propulsion Spreadsheets
- Reading and Writing ACSYNT Analysis Input Files

The functional specifications of each one of these three software design problems will now be outlined.

8.2.1 Invoking the Spreadsheets from ACSYNT

Because all the analysis modules of ACSYNT, COPES, and ACSYNT Control are equally important, input for all of them should start at the main menu. The Geometry and Trajectory modules already have an advanced input capability, so the spreadsheet menu item should be placed conveniently within these menu modules. All the other modules should start the spreadsheet upon selection of the proper option from the main menu. Placing the spreadsheets in this manner inside the interactive menu system will

lessen the amount of time spent moving between menu modules while modifying analysis variables.

Dialog messages from the spreadsheet should appear in the same location as those in ACSYNT, and not overlap. Upon entering the spreadsheet, the five dialog messages from ACSYNT must remain in the dialog box. After leaving the spreadsheet, any dialog messages in the message area should be transferred back to ACSYNT to preserve continuity in the message area.

Although there are nine separate spreadsheets, only one function should be called to start them. Different configuration files will be used to distinguish the spreadsheets. The spreadsheet subtitles, read from the configuration files, will serve to remind the user of the current set of spreadsheet data.

ACSYNT will probably differ from other applications of the spreadsheet because more than one spreadsheet is utilized in the program. The user must be able to modify analysis variables in one module, exit it, and return to the spreadsheet and see the changes made. Additionally, a default set of cell values must be preserved when starting a new set of input data. This will require keeping a default set of values in one configuration file, and saving it under a different name after modification. The file with the modified data must be read if changes have been made to the spreadsheet.

8.2.2 Linking CAD Geometry with the Spreadsheet

The CAD facility extracts geometric parameters from the ACSYNT aircraft geometry model for analysis. These parameters must be transferred to the spreadsheets in the Geometry and Propulsion modules of ACSYNT to allow the user to check the geometric input parameters for the analysis code. This can be accomplished by writing an "addendum" file containing ACSYNT geometry variable names and values. When the CAD system computes the geometric parameters of the model, the addendum file should be written.

After the configuration file for the Geometry (or Propulsion) module is read, the addendum file should be read. This will set all of the parameters that were determined by the analysis of the display model. If variable names are found in the addendum file that are not in the current spreadsheet, no action should be taken. This will allow the same addendum file to be read for both the Propulsion and Geometry input modules.

8.2.3 Reading and Writing ACSYNT Analysis Input Files

The ACSYNT analysis input file is composed of three major parts:

- COPES Input Section
- ACSYNT Control Information
- ACSYNT Analysis Module Data

Typical analysis input files are three to five pages long. The COPES input portion occupies the first part of the file. It consists of formatted or free form data describing the optimization parameters for the analysis file. An example of COPES input is shown in Figure 15 on page 80.

Not only single variables, but entire lines of COPES input values can be defaulted by skipping them with a comma or blank space. Often, these defaults depend on values previously assigned. The spreadsheet must find these default values and assign them properly when reading the COPES input portion of ACSYNT input files.

ACSYNT Control information appears after the COPES input and consists of six lines of formatted data, as shown in Figure 16 on page 81. The first line is the aircraft type, a 9 character string. The next line of integer data determines, among other things, the number of ACSYNT modules that have data in the remainder of the input file. These modules are listed by their module number in the fourth line. The ACSYNT file input/output software must recognize these numbers and use them to read or write the module data in the proper order.

Most of the ACSYNT module analysis data is in FORTRAN namelist format. Namelists allow data to be grouped and entered format free, with sets of variable names, equal signs, and values separated by commas. Namelist formatting allows arrays to be entered as well as single variables of any type. Array elements can be multiply assigned, with values such as "10*1.0" resulting in ten elements of an array being assigned the value 1.0.

```
$ DATA BLOCK A
**** GRUMMAN A-6E INPUT FILE FOR RANGE TRADES
$ DATA BLOCK B
         1          0          0          3
$ DATA BLOCK R
       570          5        422          5
$ DATA BLOCK S
       694        581        567
$ DATA BLOCK T
     58000      60000      62000      64000      66000
$ DATA BLOCK U
       700        800        900       1000       1100
$ DATA BLOCK V
END
```

Figure 15. Sample COPES Input Data

```
BOMBER
5     4     5   570   570     1     0     1     0     0     7     0
0.0001        0.80 100000.00  60000.00      80.00        0.30
6     1     2     3     4
1     2     6     2
4     2     1     6     3
```

Figure 16. Sample ACSYNT Control Input Data

The file reading and writing software should have robust namelist input/output capabilities that allow this type of data to be processed. In addition, functions should be included to allow handling of single variables and arrays of any data type.

A small portion of the module data for an ACSYNT input file is shown in Figure 17 on page 83.

8.3 Design of Integration Software

The requirements and functional specification outlined an integration plan for the spreadsheet and ACSYNT. That plan will now be discussed in detail.

8.3.1 Interlanguage Calls

ACSYNT is written in FORTRAN, and the spreadsheet is coded in C. Linking these two languages presented an interesting technical challenge. Compilers on different machines handle interlanguage calls in different ways. For example, the FORTRAN compilers on the Silicon Graphics Iris and IBM RT workstations require that an underscore be placed after the C function name that is called from a FORTRAN program. The FORTRAN compiler on the IBM RISC/System 6000 workstation does not require an underscore. The underscore identifies the C subroutine as an external

```
BOMBER
***** A-6E WEIGHTS DATA
 &OPTS
   WGTO    = 56000.000, SLOPE(7)  =   0.650, IGRAPH    =       1,
   &END
 &FIXW
   WAF    = 14209.000, WFEQ    =  5464.000, WPS     =  6935.000,
   WCREW   =   450.000,
   &END
**** A-6E  AERODYNAMICS DATA ***
&ACHAR
   ABOSB     =   0.150, ALMAX   =    30.000, AMC     =    40.000,
   BDNOSE    =   4.820, BTEF    =     0.000, SFWF    =     1.000,
   XCDW      =     0.600, ALELJ     =       3, ISMNDR  =          0,
   ISUPCR    =         0, ITRAP     =       0, IXCD    =          1,
   &END
 &AMULT
   CSF       =     1.000, ESSF      =     1.000,
   &END
```

Figure 17. Sample ACSYNT Analysis Module Input Data

module to be found during linking. If the underscore is omitted on machines that require it, the compiler will search for the function name in the FORTRAN code.

The "ifdef" statement in the C preprocessor was used to selectively include function names with underscores on machines that need it. Using the preprocessor allows a command line option to be specified to alter the code as it is compiled, resulting in one program that can be run on all machines.

Parameters can be passed in the argument list of a C function called from FORTRAN. The FORTRAN functions **%ref()** and **%val()** are used to send pointers (references) and values of arguments respectively. Figure 18 on page 85 shows the use of these functions and the preprocessor directives used to compile and link FORTRAN and C code with compilers that do and do not support the underscore character.

All the FORTRAN subroutines that call C functions are in one file - "bridge.brf". Likewise, all C functions that use FORTRAN subroutines are consolidated in the file "bridge.c". Encapsulation of the routines in this manner allows for quick changes to be made to all of the interlanguage calls, should this become necessary in the future.

8.3.2 Parameters Passed from ACSYNT to the Spreadsheet

When the spreadsheet is called from ACSYNT, the name of the configuration file and help file for the current module are arguments for the main spreadsheet routine. This ensures that the spreadsheet can operate on different sets of analysis data without changing the program.

FORTRAN Variable Types and Function Call:

```
INTEGER WKID, BEGSTR, IADD
CHARACTER*14 HELPFILE, SSFILE
CALL spread(%val(WKID), %val(BEGSTR),
$           %ref(SSFILE), %ref(HELPFILE), %val(IADD))
```

The FORTRAN function %ref is used to pass a reference (i.e. a C pointer) for the starting address of the character strings. The other arguments use the %val FORTRAN function to send the values of WKID, BEGSTR, and IADD to the C routine.

C Function Called:

```
#ifdef WK==IRIS
   spread_(workid, bstrid, ssfile, helpfile, add_flag)
#else
   spread(workid, bstrid, ssfile, helpfile, add_flag)
#endif
   int workid;
   int bstrid;
   int add_flag;
   char ssfile[];
   char helpfile[];
   {
   /*  function appears here  */
   }
```

Figure 18. Interlanguage Call Example

Because ACSYNT and the spreadsheet both use PHIGS, certain PHIGS data must be passed so that the spreadsheet does not interfere with the PHIGS data used by ACSYNT. Thus, before starting the spreadsheet, the last PHIGS structure number used by ACSYNT is found in the graphics data structure of the program. This number is sent to the spreadsheet for its starting PHIGS structure identifier. Using this structure number ensures that the spreadsheet graphics data will not overwrite or corrupt existing ACSYNT graphics information. Also, the workstation identifier is sent to the spreadsheet so that the graphics data is displayed on the correct device.

The requirements state that the same spreadsheet function should be used for all the ACSYNT modules. However, the Geometry and Propulsion modules read an addendum file, and the other modules do not. A flag that controls the reading of the addendum file is included in the parameter list of the spreadsheet. When the flag is set to one, the spreadsheet will read the addendum file after the configuration file.

In summary, the design above requires that the following parameters be passed to the spreadsheet:

- Workstation Identifier
- First Structure Number for Spreadsheet Data
- Spreadsheet Configuration File Name
- Spreadsheet Help File Name
- Addendum File Read Flag

8.3.3 Dialog Area Message Transfer

The requirements specified that there should be continuity in the ACSYNT and spreadsheet dialog area. Because of this requirement, all five lines of the ACSYNT message area are sent to the spreadsheet data structure before the spreadsheet is invoked. When the spreadsheet is displayed, the messages in the dialog area are the same as they were in ACSYNT.

After RETURN is selected from the spreadsheet menu, all the messages in the dialog area are passed into ACSYNT. These messages are in the dialog area when the ACSYNT screen reappears.

8.3.4 Flowcharts of ACSYNT/Spreadsheet Integration

Figure 19 on page 88 shows the sequence of events undertaken when the spreadsheet is invoked from ACSYNT. The ACSYNT graphics data structure query and message transfer routine described above are shown in this figure.

Because the user can re-enter the ACSYNT spreadsheet and modify values after they have been changed, the spreadsheet must decide whether to use the default file or the modified one. All modified files are deleted when the program is started and when a new file is read. When a modified file exists, the spreadsheet should read it. This decision is shown in the flowchart.

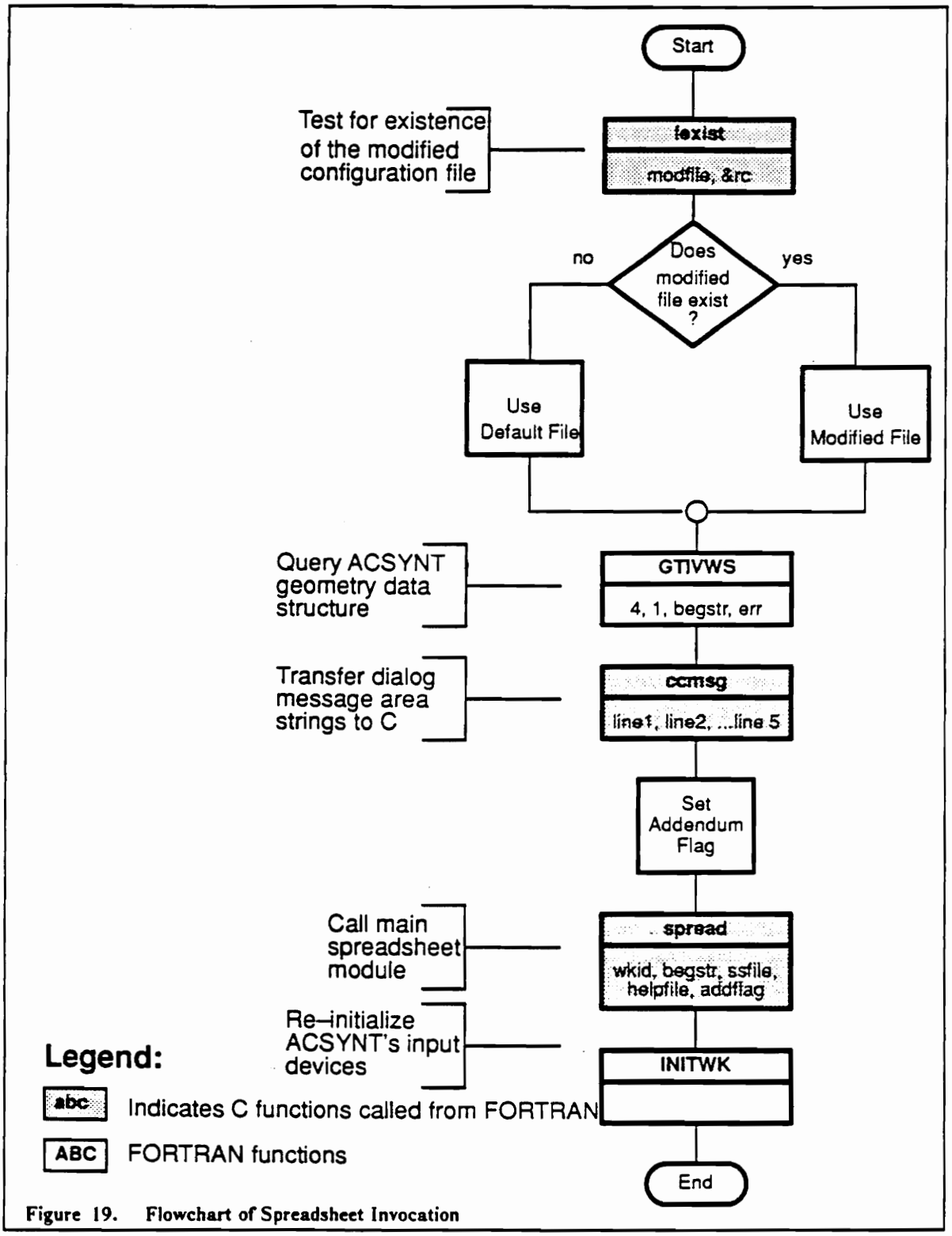


Figure 19. Flowchart of Spreadsheet Invocation

When the user leaves the spreadsheet and returns to ACSYNT, another sequence of events takes place. This is highlighted in Figure 20 on page 90. The configuration file is saved with a ".MOD" extension to separate it from the original default file. After the file is saved, message lines are sent back to ACSYNT, and all PHIGS structures, cell information, and global spreadsheet variables are deleted.

8.3.5 ACSYNT Analysis Input File Read/Write System

The design of this portion of the integration system contains functions that can be used to interface the spreadsheet with any analysis code. Only the order and manner in which they are used are specific to ACSYNT.

Functions for reading variables and arrays from an open analysis input file and placing them in the data structure as real, integer, logical, character, or string values were developed. Because ACSYNT input is namelist intensive, robust functions for reading namelist formatted data and placing it into the data structure were written. Modules for the reverse process, taking data structure values and writing analysis input files based on this information were also created.

Emphasis is placed on the design of the functions to accomplish the read and write operations in the following sections. This will best serve readers of this thesis who are trying to use the spreadsheet with another analysis system.

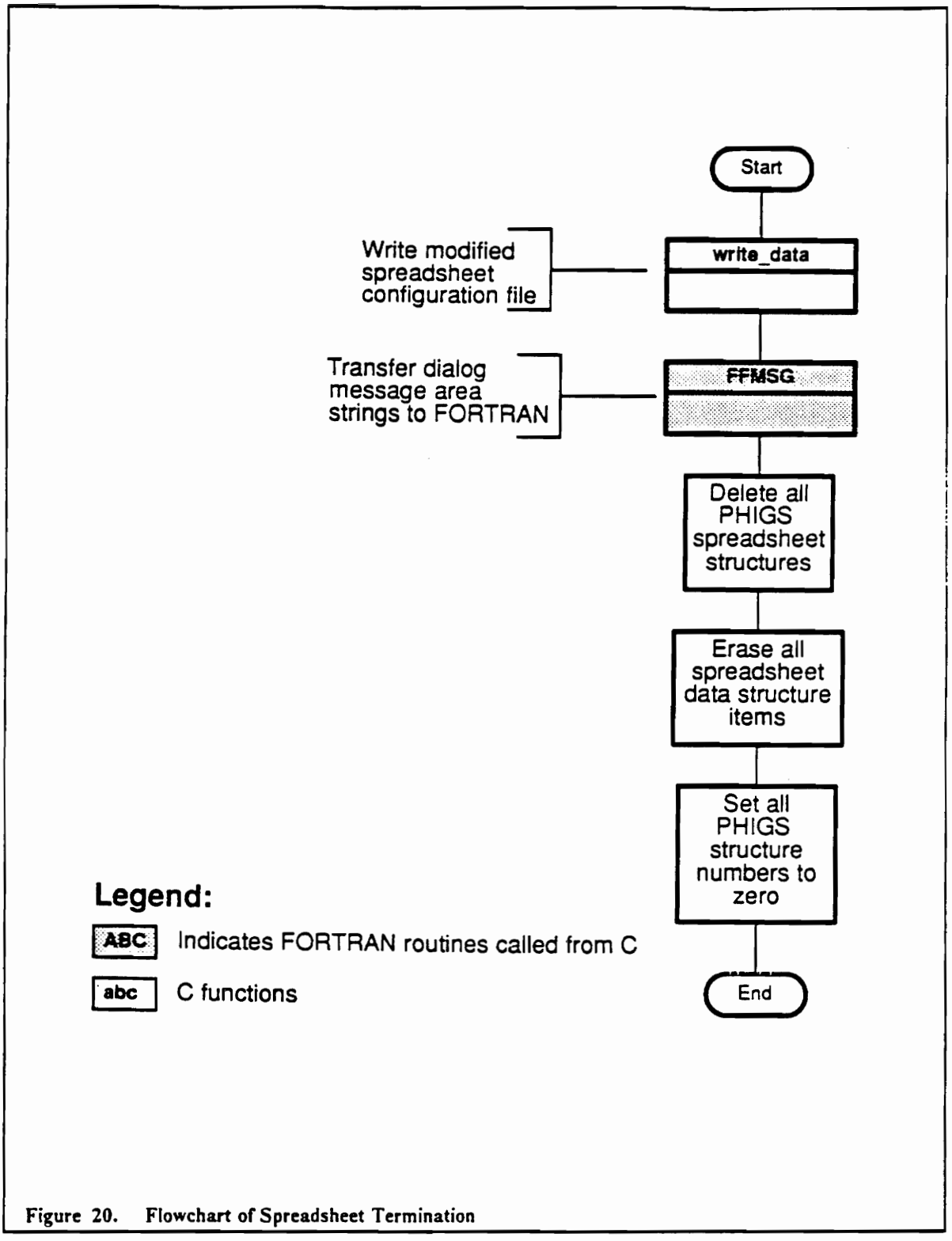


Figure 20. Flowchart of Spreadsheet Termination

8.3.5.1 Reading Single Variable Values

There are five functions that place information in the data structure after the value of a single variable has been determined. They are:

- **spput_i** - puts integer values into the spreadsheet data structure
- **spput_r** - puts real values into the spreadsheet data structure
- **spput_ch** - puts character string values (length < = 10) into the spreadsheet data structure
- **spput_l** - puts logical values into the spreadsheet data structure
- **spput_st** - puts string variable values (length > 10) into the spreadsheet data structure

Each accepts two arguments, the value of the variable and its name. When these routines are called, the data structure is checked to see if the variable name exists. If it exists, its variable type is checked in the spreadsheet data structure. It is then added in the data structure. If the variable type does not match, a warning message is printed, and the variable is added in its current format.

8.3.5.2 Reading Array Variables

Arrays can be placed into the spreadsheet data structure by using one function call. The functions that allow this are:

- **spputa_i** - puts integer array values into the spreadsheet data structure
- **spputa_r** - puts real array values into the spreadsheet data structure
- **spputa_ch** - puts character string array values (length < = 10) into the spreadsheet data structure
- **spputa_l** - puts logical array values into the spreadsheet data structure

Three parameters are passed to all of the above functions: the array to be placed in the data structure, the name of the array (without subscripts), and the number of values to be stored.

The function forms array element names (e.g "ELEM(1)") by adding subscripts to the array name passed in the parameter list. The array element name and its value are sent to one of the single element placement routines (spput_i, spput_r, etc.) to be included in the data structure. This process is repeated until all array elements are filled.

8.3.5.3 Reading Namelists

The C language does not support namelist input and output. Specialized functions were created to parse FORTRAN namelists and insert the information in the spreadsheet data structure. The function **spput_nml** accomplishes this task. It accepts three parameters, the current input file pointer, the name of the namelist, and the namelist number. ACSYNT can use multiple namelists with the same name that hold different data items. Because of this requirement, the namelist read routine must be able to read namelists with the same name and store their data properly. The namelist number is equal to the number of times that the namelist has been read. It is usually set to 1.

The namelist read module is called when a namelist is to be read. It finds the namelist name in the file, and positions the file pointer at the beginning of the first variable in the namelist. The variable name is read, and looked up in the data structure to find its index (i.e. data structure location) and variable type. If the index is not found, the program assumes that the variable name is the name of an array. An array element name is then created from the variable name by appending a subscript to it, and this variable is looked up in the data structure. If it is not found, an error message is printed, and the file pointer skips to the next variable name. If it is found, the proper data structure addition function is called, based on whether it is an array or a single variable.

Arrays inside a namelist can take several forms. Acceptable formats for array elements inside a namelist are shown below:

ELEM(1) = 1.5, ELEM(2) = 2.0, ELEM(3) = 2.0, ELEM(4) = 2.0

ELEM = 1.5, 2.0, 2.0, 2.0

ELEM = 1.5, 3*2.0

A special function was created to handle namelist array input. Any of the above formats can be processed. This function, called **nml_array** accepts four arguments:

1. Pointer to an open input file

2. Array variable name (w/o subscripts)

3. Namelist name to which the arrays belong

4. Namelist number for multiple namelists

8.3.5.4 Writing Single Variable Values

Because all information is stored in character format, only one function is needed to extract variable values from the spreadsheet data structure. The routine **sp_get** returns a character string representation of the variable's value after being passed its name and repetition number. The repetition number is used for instances where the same variable name is used several times in one spreadsheet data structure.

The **sp_get** function can be called several times in succession to extract array information from the spreadsheet data structure.

8.3.5.5 Writing Namelist Formatted Data

Extracting namelist data from the data structure and writing it to a file presents less of a problem than reading it. A routine named **spget_nml** accepts the namelist name, number, the first variable name in the namelist, and the file pointer to the open input file as arguments. When this function is called, it steps through the data structure, writing the name and value for each previously read or modified variable in the namelist.

The output format is neatly spaced, with three sets of names and values per line. The resulting namelist is written in a columnar fashion that is very easy to read.

Different machines use different "namelist characters", which precede the namelist name and the END keyword at the end of the namelist. For example, IBM mainframes, Silicon Graphics Iris, and Hewlett-Packard workstations use the dollar sign. IBM workstations use an ampersand. This device dependent feature is controlled by the namelist write module, which contains a conditional compilation statement similar to the one described for the underscore character in interlanguage calls. The namelist character will always be correct for the hardware used.

8.4 Results of ACSYNT-Spreadsheet Integration

The design described above was successfully implemented, providing ACSYNT with a highly interactive and efficient method for analysis variable input. All existing modules of ACSYNT analysis (including COPES and ACSYNT Control) are supported by this

implementation. This spreadsheet system will be distributed to industrial and governmental users as a part of Version 1.1.1 of ACSYNT.

9.0 Summary and Conclusions

9.1 Summary

This thesis described the creation and implementation of spreadsheet input for conceptual design systems. The requirements, functional specification, and design of the system were discussed. Integration of the spreadsheet with the aircraft conceptual design system ACSYNT was performed, and this process was thoroughly documented. A set of functions for extracting and placing analysis information into the spreadsheet data structure were designed and these modules were used to combine the spreadsheet with ACSYNT.

The design of the spreadsheet and its integration with ACSYNT has been well received. During a formal ACSYNT training seminar held at the Virginia Tech Computer Aided Design Laboratory during April, 1991, the spreadsheet was used by experienced aircraft designers. Representatives from Lockheed, Northrop, McDonnell Douglas, Boeing, NASA, and the U.S. Navy used the spreadsheet as a part of ACSYNT to modify

analysis variables for aircraft conceptual design. As a group, the aircraft designers concluded that the spreadsheet was an intuitive and efficient method for changing quantities used by the analysis system. The spreadsheet performed as designed, and was bug-free.

9.2 Conclusions

The spreadsheet creation process was conducted using standardized software development techniques. Requirements and functional specifications were written before the software was designed. Design (using hierarchically decomposed flowcharts) before coding spared the author many changes in design strategy after the program was written. Using the software engineering approach described above, the spreadsheet system and integration with ACSYNT took about one-half man-year to design and code. In the end, roughly 27000 lines of well-documented (60% comments) C code were written. The 27000 lines are broken down as follows:

- 60% - Spreadsheet system
- 10% - Spreadsheet data structure utilities
- 23% - ACSYNT analysis input file read and write
- 7% - ACSYNT integration code

The approximate percentages of total development time on each of the major divisions are summarized below: (using a figure of 260 working days per man-year)

- 70% (91 working days) - Spreadsheet system
- 12% (16 working days) - Spreadsheet data structure utilities
- 13% (17 working days) - ACSYNT analysis input file read and write
- 5% (6 working days) - ACSYNT integration

Using the figures above, a conservative time estimate for interfacing the spreadsheet to an existing analysis code would be roughly one and one-half months. ACSYNT used nine separate spreadsheet modules, and file input and output had to be divided among these modules, complicating design and increasing development time. Integration times for other, more simple analysis codes will probably be shorter.

Time estimates for integrating the spreadsheet system into an evolving CAD software system are more difficult to make. Depending on the data structure of the CAD system, and the amount of data to be sent to the spreadsheet and vice versa, one to two and one-half months are reasonable predictions.

Seventy percent of the development time was invested in the creation of the spreadsheet system. This considerable effort can be utilized by the CAD software developer by integrating the spreadsheet into an existing or new CAD software system. The integration process will consume only one-sixth of the time compared to designing and coding the spreadsheet from "square one".

As demonstrated in the development time figures above, CAD software developers can use the spreadsheet to quickly create a portable, robust, and responsive input system for conceptual design analysis data. A high-level standardized programmer's development environment incorporating spreadsheet functions would benefit the CAD applications programmers of the future by giving them direct access to generalized routines for a spreadsheet system for large amounts of numerical input such as the one presented in this thesis.

10.0 Recommendations

A list of research and development areas follow that would extend the capability of the spreadsheet to enhance computer aided conceptual design.

10.1 Spreadsheet Input/Output Programming Language

The greatest hurdle in integrating the spreadsheet with an existing analysis system is the coding of the read and write functions for the input file of the analysis program. If an abstraction of the contents of the input file could be used by the spreadsheet to handle file I/O, integration times would be reduced dramatically.

One method for this has been studied. The spreadsheet would read a "command file" that contains instructions for reading and writing the input file. These instructions would be compiled by the spreadsheet when the file read or write routines were called.

Keywords used by this system should include variable, array, and namelist statements that identify the locations of blocks of analysis data in the file. Looping structures using both "do-while" and "for" constructions should be implemented when blocks of data are repeated either conditionally or a fixed number of times. Control and decision statements should be supported using "if" and "case" structures. A capability for comment lines in the file should also be provided.

The command files constructed would be smaller than a typical analysis input file. Constructing it would require considerably less effort than designing, coding, and implementing software to read and write the analysis input files.

10.2 Analysis Results

The spreadsheet enhances the input of analysis quantities. A facility should be connected to the spreadsheet that allows the results of the analysis to be inspected. Such a facility would allow the interactive display and modification of graphs of pertinent design variables, and a direct link to all of the output information. Many current CAD systems for conceptual design already possess this capability. However, the design and development of standardized routines to provide inspection of analysis results would be welcomed by CAD software developers using a standardized programmer's toolkit for the engineering of CAD systems for conceptual design.

10.3 Interactive Modification of Help Information

The EDIT and CREATE options of the MODIFY menu allow the user to change cell data structure items. The user should also have the option to add and update the help information for that variable. If a HELP INFO item were added to the menu options under EDIT and CREATE, it would allow for the update or addition of help information for the variable currently selected. Choosing the HELP INFO item would open a 2-D editor window and display the current help information for that variable, if it existed. The primary design challenge in this proposed addition is the development of an interactive editor 2-D window using PHIGS.

10.4 Static Formulae

The current spreadsheet system accepts formulae as a calculation aid. They are not retained, as they are in commercial spreadsheets such as Lotus 1-2-3 and Microsoft Excel. To retain the formulae, an additional cell data structure element could be added. These formulae could be recalculated using the formula processing capability of the spreadsheet. Recalculation could take place when a menu option RECALC is chosen or whenever a cell value is changed.

10.5 Selectively Restricted Access to Modification Functions

The MODIFY menu offers several powerful options for spreadsheet alteration. Some of these can be "dangerous", and if used improperly, can result in ranges of the spreadsheet being deleted. Some applications with future implementations of the spreadsheet system may want to "lock" these features so only a spreadsheet superuser with the proper authorization can have access to these functions.

10.6 Increased Flexibility of the User Interface

The menus, messages, and display areas for the spreadsheet system were determined by the user interface for ACSYNT. These should be made more flexible for easy integration with other CAD systems. If, at any time, high-level user interface functions became standardized, the spreadsheet should be modified to incorporate these functions. This would provide the CAD software programmer with the flexibility to modify the attributes, locations, sizes, etc. of the menus, messages and the cell display area and attributes of the spreadsheet.

11.0 References

[ANSI88] American National Standards Institute, "American National Standard for Human Factors Engineering of Visual Display Terminal Workstations ANSI/HFS Standard Number 100-1988", *Human Factors Society*, Santa Monica, California, 1988.

[Brun89] Bruns, R., "The Programmer's Hierarchical Interactive Graphics System and Its Use by Applications Programmers", *Journal of Engineering Computing and Applications*, Winter 1989, pp. 29-36.

[Colg89] Colgan, L., Spence, R., Rankin, P., and Applerley, M. D., "Designing the 'Cockpit': The Application of a Human-Centered Design Philosophy to Make Optimization Systems Accessible", *SIGCHI Bulletin*, 21, Number 1, 1989, pp. 92-95.

[Ewin87] Ewing, D. P., "Using 1-2-3, Special Edition", *Que Corporation*, Carmel, Indiana, 1987.

[Gelh90] Gelhausen, P., ACSYNT User Manual, *NASA-Ames Research Center*, Moffett Field, CA, 1990.

[Grog84] Grogono, P., "Programming In Pascal", *Addison-Wesley Publishing Company*, Reading, Massachusetts, 1984.

[Hale84] Hale, F. J., "Introduction to Aircraft Performance, Selection, and Design", *John Wiley*, New York, 1984.

[Hamm86] Hammond, D. L., "Graphics in Conceptual Aircraft Design, A Designer's Viewpoint", AIAA Paper 86-2733, 1986.

[Hays89] Hays, A. P., "Spreadsheet Methods for Aircraft Design", AIAA Paper 89-2059, 1989.

[Jaya89] Jayaram, S., "CADMADE - An Approach Towards a Device-Independent Standard for CAD/CAM Software Development", Dissertation - Doctor of Philosophy in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1989.

[Jaya90] Jayaram, S., and Myklebust, A. "Towards A Standardized Environment for the Creation of Design and Manufacturing Software", Proceedings of the International Conference on Engineering Design (ICED), Dubrovnik, Yugoslavia, 1990.

[Join72] Joint Army-Navy-Air Force Steering Committee, "Human Engineering Guide to Equipment Design", *John Wiley*, New York, 1972.

[Kohl90] Kohlman Systems Research, "AAA - Advanced Aircraft Analysis", Marketing Brochure, 1990.

[Low79] Low, S. C., Benson, R. S., and Winterbone, D. E., "Computer Aided Design Package for Diesel Engine Manifold System", SAE Paper 790277, 1979.

[Low81] Low, S. C. and Baruah, P. C., "A Generalized Computer Aided Design Package for I.C. Engine Manifold System", SAE Paper 810498, 1981.

[McCa89] McCartney, J. and Kenny, R., "Parametric Design and Manufacture of Transfer Port Cores for Experimental Cylinder Block Castings", *International Journal of Production Research*, 27, Number 8, 1989, pp. 1385-1404.

[McCl84] McClelland, M. J., "Computers and the Design Process", *Materials and Design*, 5, June/July 1984, pp. 112-118.

[Mykl89] Myklebust, A., Mahan, J. R., Brown, E. F., Jayaram, S., Taylor, A. K., Kolady, K., Malan, P., Jayaram, U., Gloudemans, J. R., and Grieshaber, M., "A CAD System for Automated Conceptual Aircraft Design", *CAD/CAM Laboratory, Virginia Polytechnic Institute and State University*, Report Number 313161-1E, July, 1989.

[McCl86] McClelland, M. J., and Smith, A. J., "The Application of Computer Aided Design to Engine Design and Analysis", *Proceedings of the Institute of Mechanical Engineers*, 200, Number B2, 1986, pp. 115-122.

[Penn91] Pennington, S. L., "A Software Engineering Approach to the Integration of CAD/CAM Systems", Dissertation - Doctor of Philosophy in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1991.

[Powr87] Powrie, S. E., "Software Ergonomics at the Sharp End: Development of a New Interface for a CAD Package", *Contemporary Ergonomics 87* (Megani, E. G., Editor) *Taylor and Francis*, London, 1987.

[Raym82a] Raymer, D. P., "Use of Computers in the Design Process", Presented at the 1982 AIAA Aircraft Conceptual Design Short Course, 1982.

[Raym82b] Raymer, D.P., "CDS Grows New Muscles", *Aeronautics and Astronautics*, June, 1982, pp. 22-32.

[Raym89] Raymer, D. P., "Aircraft Design - A Conceptual Approach", *American Institute of Aeronautics and Astronautics*, Washington, D.C., 1989.

[Raym90] Raymer, D. P., "Aircraft Conceptual Design", Course Notes by the American Institute of Aeronautics and Astronautics, 1990.

[Sand87] Sanders, M. S., and McCormick, E. J., "Human Factors in Engineering and Design", *McGraw Hill*, New York, 1987.

[Shne88] Shneiderman, B., "We Can Design Better User Interfaces: A Review of Human-Computer Interaction Styles", *Ergonomics*, 31, Number 5, 1988, pp. 699-710.

[Sinc87] Sinclair, M. A., "What Is a Good CAD Dialogue?", *Contemporary Ergonomics 87* (Megani, E.G., Editor), *Taylor and Francis*, London, 1987.

[Tani87] Taniguchi, H., Shimizu, S., and Sakakura, K., "The Development of a CAD/CAM System for Engine Parts", *Proceedings of the Institute of Mechanical Engineers*, C03, 1987, pp. 1-5.

[Taus79] Tausworthe, R. C., "Standardized Development of Computer Software", *Prentice-Hall*, Englewood Cliffs, New Jersey, 1979.

[Tayl88] Taylor, A. K., "Specification of Mission Cycles for Aircraft Conceptual Design Using the PHIGS Standard", Thesis - Master of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1988.

[Town85] Townsend, A., "Multiplan in Plain English", *Simon and Schuster*, New York, 1985.

[Vand76] Vanderplaats, G. N., "Automated Optimization Techniques for Aircraft Synthesis", AIAA Paper 76-909, 1976.

[Wamp88a] Wampler, S., Myklebust, A., Jayaram, S., and Gelhausen, P., "Improving Aircraft Conceptual Design - A PHIGS Interactive Graphics Interface For ACSYNT", AIAA Paper 88-4481, 1988.

[Wamp88b] Wampler, S., "Development of a CAD System for Automated Conceptual Design of Supersonic Aircraft", Thesis - Master of Science in Mechanical Engineering, Virginia Polytechnic Institute and State University, 1988.

Appendix A. ACSYNT Spreadsheet User Guide

This appendix contains the spreadsheet user guide from the ACSYNT documentation.

The spreadsheet is used for input to ACSYNT's analysis modules that are not handled by other advanced methods within ACSYNT (such as those provided by the Geometry and Trajectory menu items).

The spreadsheet is constructed of cells that contain information about the variables. Each cell has three lines of text: a variable name, a variable value, and a comment line. In ACSYNT, the comment line consists of a FORTRAN namelist name and a variable type. The namelist name begins with a dollar sign. All FORTRAN variable types are supported. The variable conventions used in the ACSYNT spreadsheet are:

- RL - real
- IN - integer
- LO - logical
- CH - character strings of length 10 or less
- ST - character strings longer than 10 characters

The namelist names are provided for reference and to assist in looking up information in the analysis documentation.

The cells shown in the spreadsheet window are only a small part of the entire available spreadsheet. Each cell has an "address", or location that can be referenced by a letter and a number. For instance, the uppermost cell on the left side has the address A1. The spreadsheet can handle addresses up to Z100. Large cells that contain strings (e.g. TITLE cells) are referenced by their leftmost location.

A.1 Changing Variable Values and Scrolling

Purpose:

to change ACSYNT analysis variables and move around in the spreadsheet

Menu Path:

```
MAIN (AERODYNAMICS or PROPULSION or STABILITY or WEIGHTS
etc.)
```

Description:

The cells in the spreadsheet contain information about ACSYNT analysis variables. One cell is always "highlighted", or displayed with a yellow background. Highlighting indicates that a cell is ready for input.

Example:

- Choose AERODYNAMICS from the MAIN MENU.
- Highlight the TITLE cell by selecting it with the pick device.
- Type "I'M GETTING USED TO ACSYNT'S SPREADSHEET" and hit ENTER.
- The cell's value should change.
- Select the XCDW cell.
- The highlight will move to that cell.
- Enter 0.333 as a new value.
- Select the ELLIPC logical variable.
- The highlight will move to that cell.
- Type T to set a new value for ELLIPC.
- The displayed value will change to ".TRUE.".

The scroll bars and arrows on the right side and bottom of the cell display area allow the spreadsheet highlight and window to be moved. The scroll arrows move the cell highlight one unit in any direction. The scroll bars perform two functions. The red and white scroll markers indicate the relative position of the highlight in the entire spreadsheet. If the highlight is near the top of the spreadsheet, the vertical scroll bar marker will be close to the top of the scroll bar. The horizontal scroll marker functions in a similar fashion. The pick device

can be used to select a point anywhere in the scroll bars, and move the current cell automatically to that relative position in the spreadsheet.

Example:

- From the MAIN MENU, select AERODYNAMICS.
- The cell highlight starts in location A1.
- Select the "down" scroll arrow.
- The cell highlight moves to address A2.
- Select the "right" scroll arrow.
- The cell highlight moves to address B2.
- Choose a point in the vertical scroll bar near the "down" scroll arrow.
- The spreadsheet scrolls to the bottom of the Aerodynamics module input data.

A.2 Using Formulae

Purpose:

to aid calculation of analysis input parameters

Menu Path:

```
MAIN (AERODYNAMICS or PROPULSION or STABILITY or WEIGHTS
etc.)
```

Description:

The ACSYNT spreadsheet has calculation capabilities. If a mathematical expression is entered, preceded by a plus sign, the spreadsheet will attempt to evaluate the expression. Simple mathematical operators can be used (+, -, *, /, and exponentiation with a caret) with nested parentheses. The usual algebraic order of operations is followed, with exponentiation performed first, followed by multiplication and division, then addition and subtraction.

Example:

- From the MAIN MENU, select AERODYNAMICS.
- Highlight the CDSTR(1) variable, which represents a Cd*S value for stores. It is in cell H36.
- Type "+0.04*21.5" and press ENTER.
- The cell value changes to 0.864.

Cell addresses and variable names can be added into formulae. They are replaced with variable values when the formulae are evaluated.

Example:

- From the MAIN MENU, select AERODYNAMICS.
- The SFWF variable is in cell E3. Highlight it.
- Type "+ABOSB*SWPMAX/0.05*(A10+5)" and press ENTER.
- The value of the cell will change to 936.0.

A.3 Using the Spreadsheet Help Facilities

Purpose:

to get on-line help for analysis variables and spreadsheet functions

Menu Path:

```
MAIN (AERODYNAMICS or PROPULSION or STABILITY or WEIGHTS
etc.)
```

Description:

The spreadsheet includes a facility that provides help on spreadsheet functions and the currently selected variable. Help can be accessed by selecting a variable twice with the pick device or typing a question mark. A help window will appear

on one-half of the screen away from the currently highlighted cell. The help window is removed by selecting it or typing an exclamation point. Help is available for all ACSYNT analysis variables.

The HELP standard menu item summons help on the functions of the current menu item. The resulting help window can be removed in the same manner as the variable help windows.

Example:

- Choose AERODYNAMICS from the MAIN MENU
- Select the XCDW cell.
- Select it again.
- A help window appears with the default value and a description of XCDW.
- Select the help window.
- The help window disappears.
- Choose the HELP standard menu item.
- A full screen window appears with help for the main menu.
- Type an exclamation point to re-display the spreadsheet screen.

A.4 Immediate Commands

Purpose:

a shortcut for many typical spreadsheet functions

Menu Path:

```
MAIN (AERODYNAMICS or PROPULSION or STABILITY or WEIGHTS
etc.)
```

Description:

Immediate commands speed up the use of the spreadsheet. Commands for searching for cells, assigning variables, and finding and removing help information can be entered from the string input area. Available immediate commands are:

- **?** - request help on the currently highlighted cell variable
- **!** - remove the help window
- **: <cell address>** - move the cell highlight to the cell address
- **/ <variable name>** - move the cell highlight to the cell with the variable name given
- **$ <variable name> = <value>** - assign the value to the variable name and highlight the cell
- **"** - display the last formula entered

Example:

- From the MAIN MENU, select AERODYNAMICS.
- Type "/SWPMAX".
- The SWPMAX variable is highlighted.
- Access the help information on this variable by typing a question mark.
- Remove the help window by typing an exclamation point.
- Type ":G47".
- The cell highlight will move to cell G47, highlighting the variable LDLAND.
- Type "$ABOSB = 0.21".
- The spreadsheet will scroll to the ABOSB cell, which was assigned the value 0.21.

A.5 Jump

Purpose:

to jump to a certain cell address

Menu Path:

```
MAIN (AERODYNAMICS or PROPULSION or STABILITY or WEIGHTS
etc.( JUMP ))
```

Description:

The JUMP command from the SPREADSHEET menu prompts the user for a cell address to highlight. It is functionally the same as the colon (:) immediate command. When this command is selected, the user is prompted as follows:

```
message line 5
message line 4
message line 3
message line 2
ENTER CELL ADDRESS (i.e. A4) TO JUMP
```

Example:

- Select AERODYNAMICS from the MAIN MENU.
- Choose the JUMP option.
- Type "H18" and press ENTER.
- The FLDM(9) variable is highlighted in cell H18.

A.6 Modifying the Spreadsheet

The layout of the spreadsheet cells and individual cell data items such as namelist name and variable type can be modified interactively. The MODIFY menu option gives the user the options described below:

- INSERT - inserts rows and columns of blank space into the spreadsheet.
- DELETE - deletes rows, columns, and ranges of cells in the spreadsheet.
- CREATE - allows the user to create new cells.
- EDIT - permits existing cell data items to be modified.
- MOVE - allows ranges of cells to be moved in the spreadsheet.
- SET DEFAULT - saves the current spreadsheet data set as the default file for an ACSYNT module.

A.6.1 Inserting Rows in the Spreadsheet

Purpose:

to insert a row of blank space

Menu Path:

```
MAIN (AERODYNAMICS, PROPULSION, STABILITY, etc.( INSERT (
ROW )))
```

Description:

When the INSERT ROW option is selected, the user is prompted as follows:

```
message line 5
message line 4
message line 3
message line 2
SELECT CELL OR MENU ITEM, OR ENTER CELL ADDRESS TO INSERT
```

Selecting a cell or typing a cell address will cause a new row of cells to be inserted in the specified location. Choosing the ABORT option from the menu will result in cancellation of this command.

A.6.2 Inserting Columns in the Spreadsheet

Purpose:

to insert a column of blank space

Menu Path:

```
MAIN (AERODYNAMICS, PROPULSION, STABILITY, etc.( INSERT (
COLUMN )))
```

Description:

When the INSERT COLUMN option is selected, the user is prompted as follows:

```
message line 5
message line 4
message line 3
message line 2
SELECT CELL OR MENU ITEM, OR ENTER CELL ADDRESS TO INSERT
```

Selecting a cell or typing a cell address will cause a new column of cells to be inserted in the specified location. Choosing the ABORT option from the menu will result in cancellation of this command.

A.6.3 Deleting Rows from the Spreadsheet

Purpose:

to delete a row of cells

Menu Path:

```
MAIN (AERODYNAMICS, PROPULSION, STABILITY, etc.( DELETE (
ROW )))
```

Description:

When the DELETE ROW option is selected, the user is prompted as follows:

```
message line 5
message line 4
message line 3
message line 2
SELECT CELL OR MENU ITEM, OR ENTER CELL ADDRESS TO DELETE
```

Selecting a cell or typing a cell address will result in a confirmation prompt. Answering "Y" to this prompt will result in the row of cells being removed from

the spreadsheet. All cells to the right of the deleted row will be moved to take the place of the deleted row. Choosing the ABORT option from the menu will result in cancellation of this command.

A.6.4 Deleting Columns from the Spreadsheet

Purpose:

to delete a column of cells

Menu Path:

```
MAIN (AERODYNAMICS, PROPULSION, STABILITY, etc.( DELETE (
COLUMN )))
```

Description:

When the DELETE COLUMN option is selected, the user is prompted as follows:

```
message line 5
message line 4
message line 3
message line 2
SELECT CELL OR MENU ITEM, OR ENTER CELL ADDRESS TO DELETE
```

Selecting a cell or typing a cell address will result in a confirmation prompt. Answering "Y" to this prompt will result in the column of cells being removed from the spreadsheet. All cells below the deleted column will be moved to take the place of the deleted column. Choosing the ABORT option from the menu will result in cancellation of this command.

A.6.5 Deleting Ranges from the Spreadsheet

Purpose:

to delete rectangular cell ranges from the spreadsheet

Menu Path:

```
MAIN (AERODYNAMICS, PROPULSION, STABILITY, etc.( ( DELETE (
RANGE )))
```

Description:

When the DELETE RANGE option is selected, the user is prompted as follows:

```
message line 5
message line 4
message line 3
message line 2
ENTER ADDRESS OR SELECT UPPER LEFT CORNER OF DELETE RANGE
```

After specifying the upper left corner of the range, the program will prompt the user for the lower right corner address. Choosing ABORT from the menu will cancel the deletion process. After both corners of the range have been specified, a confirmation prompt appears. Answering "Y" to this prompt results in the range of cells being removed from the spreadsheet.

Example:

- From the MAIN MENU, select AERODYNAMICS.
- Choose MODIFY, then DELETE, and finally, RANGE from the menu.
- At the first prompt, select cell B3 (variable RCLMAX).
- Choose cell D6 for the lower right corner of the delete range.
- The range to be deleted is highlighted in black.
- Answer "Y" to the confirmation prompt.
- The range of cells will be deleted, leaving a "hole" in the spreadsheet.

The delete range command can be used to remove single cells from the spreadsheet. Specify the same address for both the upper left and lower right corners of the deletion range.

A.6.6 Moving Ranges in the Spreadsheet

Purpose:

to move ranges of cells

Menu Path:

```
MAIN (AERODYNAMICS, PROPULSION, STABILITY, etc.( ( MODIFY (
MOVE )))
```

Description:

Selecting the MOVE option results in a series of three prompts to determine the range to move and the destination range. The first prompt asks for the upper left corner of the range, the second for the lower right corner, and the third prompt requests the address of the upper left corner of the destination. The range must be specified from upper left to lower right. Choosing ABORT from the menu will cancel the move.

NOTE: *The destination range must not include any existing cells. Moving cells on top of existing ones can produce unpredictable results.*

Example:

- From the MAIN MENU, choose AERODYNAMICS.
- Choose MOVE from the MODIFY menu.
- At the first prompt, select cell B4 (variable SWPMIN).
- Choose cell E7 for the lower right corner of the delete range.
- The range to be moved is highlighted in blue.
- Enter I2 for the upper left corner of the destination range.
- The range of cells will be moved, and the spreadsheet will scroll to the new position of the cells.

A.6.7 Editing Cell Data Items

Purpose:

to edit cell items such as variable name, type, and namelist name

Menu Path:

```
MAIN (AERODYNAMICS, PROPULSION, STABILITY, etc.( MODIFY (
EDIT )))
```

Description:

Selecting the EDIT item and choosing a cell will display a template of cell data items. These items can be modified if they are first selected from the screen, and new values entered. This option allows the user to modify variable items when their use changes in the analysis code.

Example:

- From the MAIN MENU, select AERODYNAMICS.
- Choose EDIT from the MODIFY menu.
- At the prompt, select cell F3, (SMNDR).
- The cell will turn green, and a template will appear on the screen.
- Choose the VARIABLE NAME line from the template.
- At the ENTER VARIABLE NAME prompt, type "EDITCELL"
- The variable name will change from SMNDR to EDITCELL.
- The other template values can be modified in the same manner.

A.6.8 Creating New Cells

Purpose:

to create new cells in empty locations of the spreadsheet

Menu Path:

```
MAIN (AERODYNAMICS, PROPULSION, STABILITY, etc.( MODIFY (
CREATE )))
```

Description:

The CREATE option allows the user to place new cells in the ACSYNT spreadsheet when new analysis variables have been added. After determining a location for the cell, a template appears requesting cell data structure values. Completing the information on the template will create a new cell in the specified location.

Example:

- From the MAIN MENU, select AERODYNAMICS.
- Choose CREATE from the MODIFY menu.
- At the prompt, type the address F4.
- A white cell highlight will appear in the empty area.
- Choose the VARIABLE NAME line from the template.
- At the ENTER VARIABLE NAME prompt, type "NEWCELL"
- A cell will appear in the new location, with the name NEWCELL.
- The other template values can be modified in the same manner.

A.6.9 Saving the Modified Spreadsheet Configuration

Purpose:

to save the modified spreadsheet so that it is used as a default

Menu Path:

```
MAIN (AERODYNAMICS, PROPULSION, STABILITY, etc.( MODIFY (
SET DEFAULT )))
```

Description:

After the spreadsheet has been modified, the changed configuration can be stored as the default spreadsheet configuration for a menu module. Once the SET DEFAULT option is chosen, the following prompt appears:

```
message line 5
message line 4
message line 3
message line 2
OK TO SAVE THE CURRENT FILE AS THE DEFAULT (Y/N) ?
```

Answering "Y" to this prompt will cause the current configuration to be used each time the spreadsheet is called for the ACSYNT discipline module. All of the data, including the current variable values, will be used in the new default file. Thus, this command must be used with caution.

Appendix B. Description of Spreadsheet Functions

This appendix is presented as a guide to general spreadsheet functions. Readers wishing to integrate the spreadsheet with an existing or new application should look closely at the descriptions provided. The descriptions are broken down into the following sections:

- Setup and Invocation
- Spreadsheet Configuration File I/O
- Screen Display
- Spreadsheet Data Structure Utilities

The functions in the Setup and Invocation section are used to initialize the spreadsheet and start its execution. Routines for reading and writing spreadsheet configuration files are in the second portion of this appendix. The functions that control the appearance of the spreadsheet on the screen are described in the third part. These routines should be modified when the spreadsheet is adapted to different computer aided design systems so the spreadsheet screens match with those of the CAD system. The last section of the appendix covers the data structure lookup, placement, and query routines.

B.1 Setup and Invocation Functions

B.1.1 spread - Starts the Spreadsheet

Purpose:

Invokes the spreadsheet.

Description:

spread(workid, bstrid, ssfile, helpfile, addflag)

int	workid
int	bstrid
char	ssfile[]
char	helpfile[]
int	addflag

Input Arguments:

workid	PHIGS workstation identifier for spreadsheet
bstrid	PHIGS beginning structure identifier
ssfile	Spreadsheet configuration file name.
helpfile	Variable help file name.
addflag	If = 1, read addendum file.

Output Arguments:

None

Function Output:

None.

B.1.2 setup_spread - Initializes PHIGS data

Purpose:

Initializes structure numbers and file names in data structure.

Description:

setup_spread(workid, bstrid, ssfile, helpfile)

int	workid
int	bstrid
char	ssfile[]
char	helpfile[]

Input Arguments:

workid	PHIGS workstation identifier for spreadsheet
bstrid	PHIGS beginning structure identifier
ssfile	Spreadsheet configuration file name.
helpfile	Variable help file name.

Output Arguments:

None

Function Output:

None.

B.1.3 putsch_spread - Erases All Spreadsheet Data

Purpose:

"Cleans out" all spreadsheet graphics data, and blanks data structure.

Description:

putsch_spread(flag)

int	flag

Input Arguments:

flag	If zero - erase both the data structure and the PHIGS structures. If one, erase the data structure and only 2 PHIGS structures for the message area.

Output Arguments:

None

Function Output:

None.

B.1.4 setup_phigs - Initializes the PHIGS Environment

Purpose:

Starts the PHIGS environment, initializing the views, input devices, and color table for the spreadsheet.

Description:

setup_phigs()

None

Input Arguments:

None

Output Arguments:

None

Function Output:

None.

B.2 Spreadsheet Configuration File Input/Output

B.2.1 read_cfg_file - Reads a Specified Configuration File

Purpose:

Reads a specified configuration file.

Description:

read_cfg_file(fname, rc)

char	fname[]
int	*rc

Input Arguments:

fname[]	Spreadsheet configuration file name.

Output Arguments:

*rc	Error return code: zero if no read errors occurred.

Function Output:

None.

B.2.2 write_cfg_file - Writes a Specified Configuration File

Purpose:

Writes a specified configuration file.

Description:

write_cfg_file(fname)

char	fname[]

Input Arguments:

fname[]	Spreadsheet configuration file name.

Output Arguments:

None

Function Output:

None.

B.3 Screen Display Functions

B.3.1 draw_screen - Draws the Spreadsheet Menus and Borders

Purpose:

Draws all non-scrolling spreadsheet screen items.

Description:

draw_screen()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.3.2 draw_statics - Draws the Static Screen Areas

Purpose:

Draws title, dialog area, menu, and standard menu areas.

Description:

draw_screen()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.3.3 draw_scroll - Draws the Scroll Bars

Purpose:

Fills PHIGS structures with scroll arrow and bar information.

Description:

draw_scroll()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.3.4 draw_std_menu - Draws the Standard Menu Area

Purpose:

Fills PHIGS structures with standard menu area information.

Description:

draw_std_menu()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.3.5 screen_assemble - Assembles PHIGS Structures for Screen

Purpose:

Hierarchically links PHIGS structures for the screen.

Description:

screen_assemble()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.3.6 draw_all_cells - Draws Spreadsheet Cells

Purpose:

Constructs and displays PHIGS cell data for spreadsheet.

Description:

draw_all_cells()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.3.7 draw_subtitle - Draws the Spreadsheet Subtitle

Purpose:

Displays spreadsheet subtitle.

Description:

draw_subtitle()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.3.8 make_index - Draws Indices

Purpose:

Draws horizontal and vertical indices based on current window position in spreadsheet.

Description:

make_index()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.3.9 cell - Draws Individual Cells

Purpose:

Fills PHIGS structures for each cell, using values from the data structure.

Description:

cell(cell_index, pickid)

int	cell_index
int	pickid

Input Arguments:

cell_index	Index of cell in data structure cell array.
pickid	Pick identifier for each cell.

Output Arguments:

None

Function Output:

None.

B.3.10 build_cells - Displays Cells in Spreadsheet Window

Purpose:

Selectively displays only those cells in the current spreadsheet viewing window.

Description:

build_cells()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.3.11 kill_cells - Erases All PHIGS Cell Data

Purpose:

Deletes all PHIGS cell structures.

Description:

kill_cells()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.3.12 draw_markers - Draws Scroll Bar Markers

Purpose:

Places scroll bar markers on the scroll bars.

Description:

draw_markers(x_pos, y_pos)

int	x_pos
int	y_pos

Input Arguments:

x_pos	X-position of scroll bar marker on horizontal bar (0-19).
y_pos	Y-position of scroll bar marker on vertical bar (0-19).

Output Arguments:

None

Function Output:

None.

B.3.13 brmsg - Puts Message in Dialog Area

Purpose:

Writes message and scrolls message area.

Description:

brmsg(instring)

char	instring[]

Input Arguments:

instring	Message string to be displayed.

Output Arguments:

None

Function Output:

None.

B.3.14 iomsg - Puts Message in Dialog Area With Update

Purpose:

Places message in dialog area and updates the workstation so the message is displayed. Useful when a message must be displayed before waiting for input. (e.g. ACSYNT input file read and write)

Description:

iomsg(instring)

char	instring[]

Input Arguments:

instring	Message string to be displayed.

Output Arguments:

None

Function Output:

None.

B.3.15 highlight_cell - Highlights Cell

Purpose:

Highlights cell with yellow background and red text at a given address.

Description:

highlight_cell(x,y)

int	x
int	y

Input Arguments:

x	X-address of cell.
y	Y-address of cell.

Output Arguments:

None

Function Output:

None.

B.3.16 highlight2_cell - Highlights Cell in Specified Colors

Purpose:

Highlights cell with user-supplied text and background colors at a given address.

Description:

highlight2_cell(x, y, ccell, cval, ctext)

int	x
int	y
int	ccell
int	cval
int	ctext

Input Arguments:

x	X-address of cell.
y	Y-address of cell.
ccell	Color index of cell background.
cval	Color index of cell value.
ctext	Color index of variable name and comment text in cell.

Output Arguments:

None

Function Output:

None.

B.3.17 unhighlight_cell - Removes Cell Highlighting

Purpose:

Removes cell highlighting from a cell at a given address.

Description:

unhighlight_cell(x,y)

int	x
int	y

Input Arguments:

x	X-address of cell.
y	Y-address of cell.

Output Arguments:

None

Function Output:

None.

B.3.18 empty_mark - Draws an Empty Cell Highlight

Purpose:

Places a yellow box on an empty cell for scrolling to give continuity of motion.

Description:

empty_mark(x,y)

int	x
int	y

Input Arguments:

x	X-address of cell.
y	Y-address of cell.

Output Arguments:

None

Function Output:

None.

B.3.19 empty2_mark - Draws a Highlight in Specified Colors

Purpose:

Places a box on an empty cell for scrolling to give continuity of motion. User supplied colors are used.

Description:

empty2_mark(x, y, col)

int	x
int	y
int	col

Input Arguments:

x	X-address of cell.
y	Y-address of cell.
col	Color index of empty highlight.

Output Arguments:

None

Function Output:

None.

B.3.20 scroll_sheet - Moves the Cell Highlight

Purpose:

Moves the spreadsheet cell highlight a specified number of cells. Scrolling takes place when necessary.

Description:

scroll_sheet(relx, rely)

int	relx
int	rely

Input Arguments:

relx	Number of cells to move horizontally relative to the current one. The positive direction is to the right.
rely	Number of cells to move vertically relative to the current one. The positive direction is down.

Output Arguments:

None

Function Output:

None.

B.3.21 ss_2d_init - Initializes 2D Window

Purpose:

Initializes 2-D window display information.

Description:

ss_2d_init(lincol, bkgcol, location)

int	lincol
int	bkgcol
int	location

Input Arguments:

txtcol	Color index of 2D window border.
bkgcol	Color index of 2D window background.
location	Flag for window position. 0 - upper half of cell window, 1 - bottom half of cell window, 2 - automatically position away from the current cell, 3 - full size of cell window.

Output Arguments:

None

Function Output:

None.

B.3.22 ss_2d_disp - Displays 2D Window

Purpose:

Fills and displays 2D window.

Description:

ss_2d_disp(nlines, title, pickids, size, txtcol, lines)

int	nlines
char	title[]
int	pickids[]
int	size
int	txtcol
char	lines[][]

Input Arguments:

nlines	Number of text lines for display.
title	Title of 2D window.
pickids	Pick identifiers for the text lines.
size	0 for half screen, 1 for full screen.
txtcol	Color index of 2D window text.
lines	Lines of text to be displayed on window.

Output Arguments:

None

Function Output:

None.

B.3.23 ss_2d_remove - Removes 2D Window

Purpose:

Removes 2D window.

Description:

ss_2d_remove()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.3.24 newmenu - Displays New Menu

Purpose:

Displays a new menu in main menu area.

Description:

newmenu(title, no_items, items)

char	title[]
int	no_items
char	items[][]

Input Arguments:

title	Title of menu.
no_items	Number of menu items for display.
items	Menu items.

Output Arguments:

None

Function Output:

None.

B.3.25 oldmenu - Displays Old Menu

Purpose:

Displays last menu on "stack" of menus.

Description:

oldmenu()

None.

Input Arguments:

None.

Output Arguments:

None

Function Output:

None.

B.4 Spreadsheet Data Structure Utilities

B.4.1 spput_i - Puts Single Integer Value in Data Structure

Purpose:

Places a single integer variable into the data structure when given its value and variable name.

Description:

spput_i(value, varname)

int	value
char	varname[]

Input Arguments:

value	Variable value.
varname	String containing variable name in data structure.

Output Arguments:

None

Function Output:

None.

B.4.2 spput_r - Puts Single Real Value in Data Structure

Purpose:

Places a single real variable into the data structure when given its value and variable name.

Description:

spput_r(value, varname)

float	value
char	varname[]

Input Arguments:

value	Variable value.
varname	String containing variable name in data structure.

Output Arguments:

None

Function Output:

None.

B.4.3 spput_ch - Puts Character String Value in Data Structure

Purpose:

Places a single character string (length < 10) variable into the data structure when given its value and variable name. For character strings longer than 10 characters, use the spput_st function.

Description:

spput_ch(value, varname)

char	value[]
char	varname[]

Input Arguments:

value	Variable value.
varname	String containing variable name in data structure.

Output Arguments:

None

Function Output:

None.

B.4.4 spput_l - Puts Logical Value in Data Structure

Purpose:

Places a single logical variable into the data structure when given its value in character form (e.g. ".TRUE." or ".FALSE.") and variable name.

Description:

spput_l(value, varname)

char	value[]
char	varname[]

Input Arguments:

value	Variable value.
varname	String containing variable name in data structure.

Output Arguments:

None

Function Output:

None.

B.4.5 spput_st - Puts a String Variable in the Data Structure

Purpose:

Places a string variable (character strings longer than 10 characters) into the data structure when given its value and variable name.

Description:

spput_st(value, name)

char	value[]
char	name[]

Input Arguments:

value	Character string value.
name	Name of string variable.

Output Arguments:

None

Function Output:

None.

B.4.6 spputa_i - Puts an Integer Array in Data Structure

Purpose:

Places an array of integer variables into the data structure when given its value, size, and variable name.

Description:

spputa_i(aray, name, size)

int	aray[]
char	name[]
int	size

Input Arguments:

aray	Array of integer values.
name	Name of array.
size	Number of elements in "aray" above.

Output Arguments:

None

Function Output:

None.

B.4.7 spputa_r - Puts a Real Array in Data Structure

Purpose:

Places an array of real variables into the data structure when given its value, size, and variable name.

Description:

spputa_r(aray, name, size)

float	aray[]
char	name[]
int	size

Input Arguments:

aray	Array of real (float) values.
name	Name of array.
size	Number of elements in "aray" above.

Output Arguments:

None

Function Output:

None.

B.4.8 spputa_ch - Puts a Character Array in Data Structure

Purpose:

Places an array of character variables (each less than 10 characters) into the data structure when given its value, size, and variable name.

Description:

spputa_ch(aray, name, size)

char	aray[][]
char	name[]
int	size

Input Arguments:

aray	Array of character values.
name	Name of array.
size	Number of elements in "aray" above.

Output Arguments:

None

Function Output:

None.

B.4.9 spputa_l - Puts a Logical Array in Data Structure

Purpose:

Places an array of logical variables (in character form) into the data structure when given its value, size, and variable name.

Description:

spput_l(aray, name, size)

char	aray[][]
char	name[]
int	size

Input Arguments:

aray	Array of character representations of logical values.
name	Name of array.
size	Number of elements in "aray" above.

Output Arguments:

None

Function Output:

None.

B.4.10 spput_nml - Puts a Namelist in Data Structure

Purpose:

Reads a FORTRAN namelist from a file and places all values from it into the current data structure.

Description:

spput_nml(file, nname, nnum)

FILE	**file
char	nname[]
int	nnum

Input Arguments:

**file	Pointer into open input file.
nname	Namelist name.
nnum	Number of namelist, if multiple namelists are possible. Usually = 1.

Output Arguments:

None

Function Output:

None.

B.4.11 spget_nml - Writes Namelist Data to a File

Purpose:

Retrieves all data for a given namelist from the current data structure and writes it in an open file.

Description:

spput_nml(file, nname, nnum, vname)

FILE	**file
char	nname[]
int	nnum
char	vname[]

Input Arguments:

**file	Pointer into open input file.
nname	Namelist name.
nnum	Number of namelist, if multiple namelists are possible. Usually = 1.
vname	Name of first variable in data structure belonging to this namelist.

Output Arguments:

None

Function Output:

None.

B.4.12 spget - Retrieves Variable Values from Data Structure

Purpose:

Gets variable values from data structure.

Description:

spget(varname, value, vnum)

char	varname[]
char	value[]
int	vnum

Input Arguments:

varname	Variable name to retrieve.
vnum	Variable number if the same variable name occurs multiple times in the current data structure. Set to 1 in most cases.

Output Arguments:

value	Variable value.

Function Output:

None.

Appendix C. Integration Guide

This thesis has addressed the creation of a spreadsheet input format for conceptual design CAD systems. The design, functional specification, requirements, and integration of the system with ACSYNT were discussed. This appendix presents a generalized guide to integrating the spreadsheet with other analysis systems.

The integration problem breaks down into three areas:

- Screen Compatibility
- Data Structure Connections
- Analysis Input File I/O

C.1 Screen Compatibility

Cosmetic changes will probably be necessary for the spreadsheet that was presented in this thesis to integrate smoothly with other CAD systems. These changes will need to be made to the way that the cells, menu area, dialog box, text input area, and spreadsheet subtitle are displayed. The functions that perform these operations are described in the previous appendix.

PHIGS view 1 is used for all of the displayed data with the exception of the cells. The spreadsheet cells use view 7, and the help and 2-D windows use view 8. This viewing system may present integration problems for systems that use view optimization. The view numbers can be changed in the "spreadsheet.h" include file.

The spreadsheet invocation function **spread** is passed a beginning structure number for all of the displayed static structures. The PHIGS cell structure numbers start at a value contained in the "spreadsheet.h" include file. For ACSYNT integration, the first cell structure number was 20000, sufficiently out of range of any ACSYNT structure identifier. However, this limitation may change with different systems.

Cell dimensions and the number of cells per viewable page could also change as the spreadsheet is moved between systems. Some of the values that control these attributes are parameterized in the include file, while others must be found in the code. The file "draw_cells.c" holds the routines that control the display of the cells.

C.2 Data Structure Connections

The spreadsheet data structure can be easily connected with a data structure of an existing program by using the routines described in Appendix B. For each variable of a given type, a loop can be entered to place each value in the data structure. Then, each time the spreadsheet is exited, a configuration file should be written that holds the data structure values. These values can be read the next time that the spreadsheet is invoked.

If the CAD system changes data structure values while the spreadsheet is not in operation, an addendum file should be read. The CAD system must be modified to write the addendum file, and this file should be read after the configuration file. The addendum file read capability is built into the current spreadsheet invocation function call. The add_flag parameter must be set in the **spread** function to read the addendum file.

C.3 Analysis Input File Input/Output

Each variable must be read from existing analysis input files and placed into the data structure. Unfortunately, the current implementation of the spreadsheet must accomplish this in a "brute force" fashion. The Recommendations section of this thesis suggests an advanced method for this task.

Namelist intensive input can be done rather easily with the **spput_nml** function. Formatted input poses more of a problem. The **read_fmt_traj** function in the "read_acs.c" file contains an example of formatted data reading and direct placement into the spreadsheet data structure. An approach similar to this one should be followed for formatted data I/O.

Vita

Eric Schrock was born on December 13, 1966 and was raised in the thriving metropolis of Farmington, Pennsylvania, about 50 miles southeast of Pittsburgh. Enthralled by the United States' outstanding achievements in space flight in the 1960's and '70's, the author initially wanted to become an astronaut and a fighter pilot. He developed a strong interest in aircraft design during his teen years, building hundreds of models and dragging his parents to countless air shows and museums. This interest in aircraft design would not disappear, so the author enrolled in Virginia Polytechnic Institute and State University's Aerospace Engineering department and completed a Bachelor of Science Degree in May, 1989. After completing the degree requirements for a Master of Science from the same school in Mechanical Engineering, the author will begin his ideal career - conceptual aircraft design with Lockheed.